UNIVERSITY OF CALIFORNIA
Los Angeles

Naive Mechanics: A Computational Model for Representing and Reasoning about Simple Mechanical Devices

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Engineering

by

John Barnett Hodges Jr.

1993

The dissertation of John Barnett Hodges Jr. is approved

Rochel Gelman

Moshe Rubinstein

Gerald Estrin

Michael G. Dyer, Committee Chair

University of California, Los Angeles

1993

DEDICATION

To Bear, I owe you bud.

Table of Contents

List of Figures

Chapter 3

Chapter 4

Chapter 5

PART II

Chapter 6

Chapter 7

PART III

Chapter 8

Appendix

List of Tables

List of Knowledge Structures

ACKNOWLEDGEMENTS

First, I want to thank my advisor Michael Dyer, for without his willingness to take an engineering student under his wing to work on AI and invention, this research would never have taken place. Michael has played many roles in my life since 1984: professor, advisor, mentor, and antagonist. Not all of them have been pleasant, but in retrospect they have all been necessary stages in shaping my ability to perform independent and thorough research, and to document it appropriately.

The psychology department at UCLA has played an important role in my research. Professor Bjork provided guidance and encouraged me to learn more about the cognitive side of AI. Professor Gelman showed me how intriguing development can be when applied to simple mechanical devices. I am indebted to Professor Moshe Rubinstein: for introducing me to decision-making strategies, for his constant uplifting mood, and for his support and encouragement throughout this research program. I am also grateful to Professor Gerald Estrin for serving on my committee and for his comments and suggestions on my dissertation, and to Professors Margot Flowers and Michel Melkanoff.

I also owe a great deal to the environment I was able to work in: the UCLA Artificial Intelligence Laboratory. The AI lab was not an easy place to get welcomed into, but it brought under one roof one of the most intellectually lively and eclectic bunch of people I have ever run into. The lab and its researchers have played a very special role in my life, not to be forgotten. In particular, I would like to thank John Reeves, whose high standards and patience have served as a research role model. Other members of the lab: Sergio Alvarado, Stephanie August, Charlie Dolan, Ric Feifer, Maria Fuenmayor, Mike Gasser, Seth Goldman, Edward Hoenkamp, Stuart Levine, Eric Mueller, Risto Miikkulainen, Valeriy Nenov, Michael Pazzani, Walter Read, Ron Sumida, Scott Turner, and Uri Zernik have provided me with enticing conversation and fun. Probably least likely to be an asset, and to whom I owe the most in terms of that elusive and honest ear has been Trent Lange.

This research could not have been performed had it not been for Professors David L. Sikarskie and Richard A. Scott, who excited my interest in Engineering Mechanics when I was studying at the University of Michigan.

A bulk of my research was funded by the Office of Naval Research, under contract N00014-86-0615, and I am grateful to that agency for their support.

I would personally like to thank Verra Morgan for her unreserved and unending support in my graduate program. She has been a pillar of strength for altogether too many graduate students. There have been many others in my life over the past decade who have played supportive roles. Much of my strength has come from my family: my wife Nancy, Bear, Sally and Calvin. Nancy has only known me during the hard years, and she has put up with more than anyone could ever expect. Bear's capacity to love never ended and was always a measure to live up to. Both the UCLA Scuba Club and the Bay Area Paragliding Association have helped me to develop myself and to keep sane during the inevitable research lulls. Finally, I would like to thank my mother and my father: my mother for my creativity and my father for my curiosity and tenacity.

VITA

June 16, 1954	Born, Louisville, Kentucky
1976	B.S. Aerospace Engineering University of Michigan Ann Arbor, Michigan
1977-1978	Aerospace Technologist NASA Langley Research Center Hampton, Virginia
1978	M.S. Aerospace Engineering University of Michigan Ann Arbor, Michigan
1978-1990	Technical Specialist Rocketdyne Division Rockwell International Canoga Park, California
1985-1990	Research Assistant UCLA Artificial Intelligence Laboratory Los Angeles, California
1989-1993	Assistant Professor Computer Science Department San Francisco State University

PUBLICATIONS AND PRESENTATIONS

Hodges, J.B. (1994) *Berkeley UNIX: Principles and Practice*. Prentice-Hall. (in preparation).

Hodges, J.B., and Flowers, M., and Dyer, M. (1992). Knowledge Representation for Design Improvisation. In Volume I of *Artificial Intelligence Approaches to Engineering Design*, Edited by Tong, C. and Sriram, D., Academic Press, pp 193-217.

Hodges, J.B. (1992). *NAIVE MECHANICS: A Computational Model of Device Use and Function in Design Improvisation*. IEEE-Expert, Intelligent System and their Applications, IEEE Computer Society, 14 – 27.

Dyer, M., and Hodges, J.B., and Flowers, M. (1986). EDISON: An Engineering Design Invention System Operating Naively. *The International Journal of Artificial Intelligence in Engineering*, Vol. 1, No. 1, 36 – 44.

Lange, T., and Hodges, J.B., and Fuenmayor, M., and Belyaev, L. (1989). Simulating Hybrid Connectionist Architectures. *Proceedings of the 1989 Winter Simulation Conference*, Washington, DC.

Hodges, J.B. (1989). Device Representation for Modeling Improvisation in Mechanical Use Situations. *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, by Lawrence Erlbaum associates, Hillsdale, NJ, 643 – 650, Ann Arbor, MI.

Lange, T., and Hodges, J.B., and Fuenmayor, M., and Belyaev, L. (1989). DESCARTES: Development Environment for Simulating Hybrid Connectionist Architectures. *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, by Lawrence Erlbaum Associates, Hillsdale, NJ, 698 – 705, Ann Arbor, MI.

Hodges, J.B., and Flowers, M., and Dyer, M. (1987).Knowledge Representation for Design Creativity. *Proceedings of the ASME Winter Annual Meeting, by the American Society of Mechanical Engineers*, 81-93, Boston, MA.

Dyer, M., and Flowers, M., and Hodges, J.B. (1987).Comprehension and Invention in EDISON. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* – IJCAI-87, Vol. 2, 696-699, Morgan-Kaufmann, Milan, Italy.

Dyer, M., and Hodges, J.B., and Flowers, M. (1986). EDISON: An Engineering Design Invention System Operating Naively. *Proceedings of the First International Conference on Application of Artificial Intelligence to Engineering Problems*, Southampton, U.K., Vol. 1, 327-341, Springer-Verlag.

Hodges, J.B. (1990). *The UCLA Cognet: A Tutorial Introduction*. University of California Cognitive Science Technical Report UCLA-CSRP-90-4.

Hodges, J.B., and Marzwell, N., and Mogil, E. (1983). Development of An Analytical Model to Perform Parametric Studies to Determine Performance Improvement of a Benchless Laser Hardware and Its Control System. *Presented at the AIAA Structures, Structural Dynamics, and Materials Conference*, Lake Tahoe, NV.

ABSTRACT OF THE DISSERTATION

Naive Mechanics: A Computational Model for Representing and Reasoning about Simple Mechanical Devices

by

John Barnett Hodges Jr.

Doctor of Philosophy in Engineering

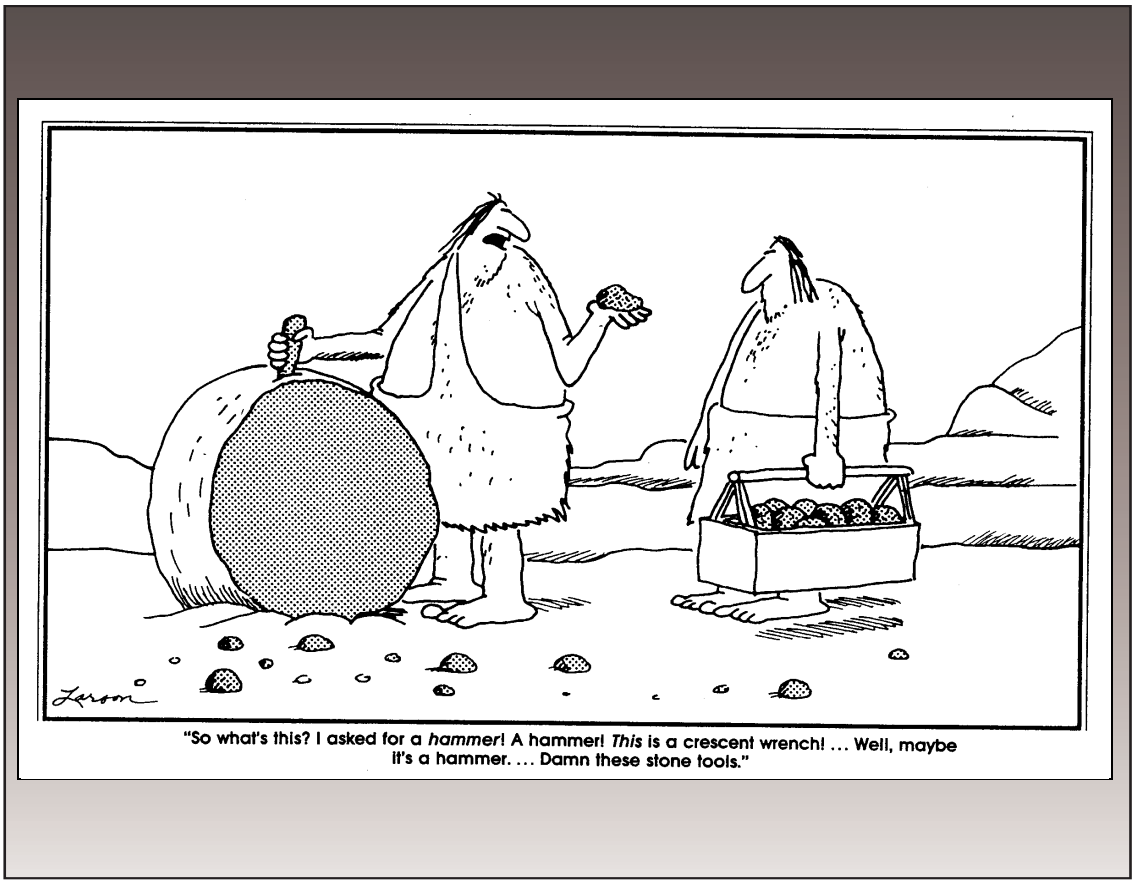
University of California, Los Angeles, 1993

Professor Michael G. Dyer, Chair

A symbolic theory for representing mechanical device behavior, function and use for commonsense reasoning is presented. Individual objects are described as a collection of states which represent property values and appearance. Object combinations are described by orientation, placement, and connectivity. Mechanical behavior describes the causal dependencies between objects which produce changes in state. Five behavioral primitives comprise a taxonomy of mechanical behavior. The behavioral primitives combine to describe complex behavior. Device function describes behavioral sequences which have observable initial and terminal states. Eleven machine primitives are proposed for describing objects which are associated with a single function. Machine primitives are used as building blocks for representing complex device functions. Object use describes the problem-solving context in which an object function is applied. Object use is represented with device-use plans, which associate object behavior and function to a set of mechanical subgoals.

The combined representation theory is called FONM: A Functional Ontology for Naive Mechanics. FONM is designed to support high-level reasoning models requiring access to knowledge at multiple abstraction levels, particularly naive problem solving, improvisation, and invention. FONM representations address issues in representation breadth and combinatorics, by providing a finite set of representation primitives which are integrated to describe device structure, behavior, function and use.

FONM device representations are used here in two demonstration models: (1) device experimentation through mutation of mechanical features, and (2) comprehension of English device descriptions.



"So what's this? I asked for a *hammer*! A hammer! *This* is a crescent wrench! ... Well, maybe it's a hammer. ... Damn these stone tools."

Chapter 1

Mechanical Understanding and Naive Mechanics

FONM (Functional Ontology for Naive Mechanics) is a representation theory for describing simple mechanical devices and mechanical problem solving situations. FONM is a central component of the EDISON (Engineering Design Invention Operating Naively) project, whose goal is to design a reasoning model (by the same name) for interpreting and evaluating simple mechanical devices in reasoning tasks requiring different degrees of mechanical understanding. FONM embodies a theory of mechanical device interactions, and EDISON models a problem solver who interprets and evaluates mechanical devices in problem solving situations.

1.1 FONM: Representing Commonsense Mechanical Knowledge

In FONM, mechanical knowledge reasoning processes are represented in terms of symbolic structures called *machine primitives* and *behavioral process primitives*. An object *function* describes what an object does, i.e., the effect an object has on its environment when it is perturbed in some way. For example, when a screwdriver is turned, the related function is driving, and the effect is magnified torque on the screw. Machine primitives represent the functions associated with simple mechanical objects. An object *behavior* describes how an object works. A behavior represents object state changes, such as motion through space. A moving object is changing its position along a specified path. Behavioral process primitives represent the causal dependencies between objects which affect their behavior. For example, object motion is described by the interactions which enable or disable motion. Behavioral process primitives are combined into sequences which represent object function.

FONM represents devices by the functions which they can produce, so devices which are representationally similar can be applied in similar ways. For example, the device representations for a bottle opener and screwdriver both include mechanical leverage, thus they can both potentially be used for prying.

The recognition that two objects used in different contexts can be interchanged because they instantiate the same device function is essential in mechanical improvisation.

1.2 Naive Mechanics and Mechanical Understanding

A *naive* reasoner is someone whose knowledge is based on experience and whose reasoning is based on commonsense principles. While mechanics is the study of objects and their interactions, *naive mechanics* is a description of mechanical devices which is based on knowledge shared by members of a culture through experience with objects in their everyday lives. Naive mechanics includes the study of objects and their interactions at a descriptive level which can be observed or inferred by naive reasoners. In addition, naive mechanics addresses the interactions between objects and the intentions of the naive rea-

soner.

Mechanical understanding is the ability to recognize and comprehend the relationship between the use of an object and the manner in which the object behaves. Object functions relate the use of an object to the states its application can produce.

In order to represent naive mechanics and understand naive mechanical reasoning, the following questions have to be addressed: What characterizes a mechanical problem? How are mechanical problems solved? What knowledge is used in solving problems? How does mechanical reasoning differ from other types of reasoning? What parts of the problem-solving process are common to all disciplines, and what parts are unique to mechanical devices? How is experience applied during problem solving? How are mechanical devices associated with problem-solving experiences?

In an attempt to provide answers to these questions, this dissertation presents FONM, a system for representing simple devices and the processes of naive mechanical reasoning. FONM is intended to serve as an experimental vehicle for studies in mechanical improvisation and invention.

There are three fundamental assumptions of this research. First, it is assumed that functional evaluations made by the problem solver are central to mechanical problem solving. Representing device function supports device evaluations based on their potential in a given context. Second, it is assumed that problem solvers remember devices by the goals the devices help achieve. Third, it is assumed that naive problem solvers can recognize what states simple behaviors produce and when simple behaviors are disabled.

A problem solver in a problem-solving situation interprets his goals in the environmental context in which the situation occurs. This is called *situation interpretation*. The context sets up constraints on which plans and device functions can be applied in pursuit of the goal. The context also sets up physical requirements that determine which devices are suitable in the problem-solving environment. The combination of these constraints restricts the choice of effective solutions.

1.3 FONM's Notion of Mechanics

Mechanical reasoning involves addressing the following three issues:

- **Object Appearance and Composition.** What object characteristics enable an object to accomplish various tasks? How are the potential uses of an object determined, based on its physical characteristics?
- **Object Perturbation and State Change**. How do a user’s actions affect the behavior of a device? How do interactions between devices and objects effect changes in the environment?
- **Goals and Situations**. How are objects related to the intentions of the user? How do a problem-solver's goals interact with and motivate the use of devices in mechanical problem solving?

FONM knowledge structures include the problem solver's goals and plans, the device, how the problem solver interprets the device, and how the device is applied in context. This knowledge provides (1) the intentional and causal interactions between the problem solver and the problem-solving context, (2) the causal interactions between objects and the problem-solving context, (3) the relative applicability of device functions in plans, and (4) the causal interactions between objects.

The representation of mechanics and mechanical reasoning can be addressed from an engineering point of view or from a problem-solving point of view. The engineering point

of view is based on the physical interactions between objects, and is called a *behavioral* approach. The problem-solving point of view is based on the interactions between the problem solver and the object, and is called an *intentional* approach.

Behavioral theories (e.g., [Forbus 1985]) of mechanical reasoning attempt to specify an object's function and behavior from descriptions of its composition and connectivity. The problem with implementing a strictly behavioral theory of naive mechanics is that it ignores task considerations, such as how object behavior and function are integrated in problem solving, how object behavior can be used to make inferences about device application in contexts not experienced, and the role of device function and behavior on learning and performance improvement.

Intentional theories (e.g., [Lehnert 1978]) of mechanical reasoning attempt to specify object function with respect to the actions it supports. These theories construct relationships between the problem solver and the role the object plays in achieving goals. The problem with a strictly intentional theory is that it ignores the dependencies between how objects are perceived and how they behave.

The mechanical theory embodied by FONM is a combination of behavioral and intentional approaches to object representation. FONM symbolically describes a device with respect to the goals and plans with which each of its functions is associated, so that the device can be understood in the context of a problem solver's actions. FONM also describes a device with respect to the behaviors its physical description supports, so that the device can be understood in the context of how it can, and does behave, as opposed to only how it is expected to behave.

1.3.1 Device Statics, Dynamics, and Pragmatics

FONM represents devices in three layers: (1) device statics, (2) device dynamics, and (3) device pragmatics. Device *statics* describes an object by its appearance, its properties, and by its components and their connectivity. Device *dynamics* describes object behavior, such as part motions and force transmission. It describes how an object works. Observable behavior is called device function, which describes what an object does. Device *pragmatics* describes human-object interactions, such as the plans the device is used in and the goals which device use is intended to achieve. For example, consider a screwdriver:

- **Statics**. The screwdriver has three components: a handle, a shaft, and a tip. The appearance and composition of each part, their relative orientation and placement, and the ways they are connected enable them to support specific behaviors.
- **Dynamics.** When a screwdriver is used to attach objects with screws, its handle is turned about its center. This action imparts an applied torque to the handle which is transmitted to the other screwdriver components. Because the tip edges are in contact with the screw's slot, the applied torque is also transmitted to the screw. The relative size of the handle, shaft, and tip satisfy the requirements for torque magnification. If the magnified torque exceeds the friction between the screw and the substrate (e.g., wood) with which it is in contact, then the screw rotates. The dynamics of screwdriver and screw motion, motion restraint, and force transmission/magnification are represented as behavioral process primitives. Behavioral process primitives can be combined in sequences to describe complex mechanical behaviors associated with device function. The screwdriver function of driving screws is a combination of screwdriver rotation, an interference fit between the screwdriver tip and the screw slot, and force transmission and magnification from the screwdriver to the screw slot. A device can have many functions, depending on what type of action is applied and on what object.

For example, the screwdriver can be used to punch holes, to pry, or to probe. Any device which can be applied in these capacities shares functionality with the screwdriver.

- **Pragmatics.** The manner in which a device function is applied is called a device *use*. It is a plan comprised of subgoals and actions. For example, to use the screwdriver to drive a screw into wood, a person must grasp the screwdriver handle, hold the screw tip to the wood, insert the screwdriver tip into the screw's slot, push the screwdriver toward the wood, and turn the handle. Each plan is associated with a unique function so that knowing one will provide access to the other.

1.4 Methodology and Approach

Two different approaches can be taken in constructing computational systems: (1) exploring a single task in depth, and (2) exploring multiple tasks each at a shallow level.

Naive mechanical problem solving is a complex interaction of knowledge types, their dependencies, their interpretation and their evaluation, and cannot be modeled as a single processing task. FONM is intended to identify the different knowledge types and their interactions, and to support a multiple task, shallow level approach to developing models of creative problem solving and invention. Our assumption is that the interactions between the reasoning abilities needed for creative problem solving are too complex to isolate and study independently with a single, in-depth approach.

Two processing tasks were selected and modeled using FONM. Each task addresses specific components of the representation approach, how they interact, and how they interact with the general problem-solving abilities of the naive reasoner. The two tasks are: (1) device mutation, and (2) device description comprehension.

1.4.1 Device Mutation in EDEXP

Mutation involves modifying a known device and then evaluating the new device. Mutation requires the ability to describe and manipulate object statics and its role in device behavior and function. A mutation model can help determine the depth of reasoning required in naive mechanics by determining the role device statics plays in naive mechanical reasoning, and by determining which objects and properties are important to device statics. EDEXP (EDison EXPerimenter) is a computer program that takes a mechanical device and a goal, and, given a device mutation evaluates its new behavior.

1.4.2 Device Description Comprehension in EDCA

Comprehension involves reading a description of device behavior and constructing an overall model of the device described. For example, a person might read the phrases "an object is pushed into a barrel, against a spring." The reader must infer relationships not mentioned in the text, e.g. that the object comes into contact with the spring, and that the object motion will be hampered by contact with the spring.

EDCA (EDison Conceptual Analyzer) is a computer program that reads mechanical device descriptions in English, constructs representations of the device behavior being described, and relates the behavioral representations to device functions. To accomplish these tasks, EDCA implements a theory of behavioral primitives based on FONM, and models a reader who reads and understands mechanical device descriptions.

The representation of knowledge is central to both of these reasoning tasks. A mutation model takes an existing device representation, selectively modifies it, and evaluates the behavior and function of new devices with respect to those of the original device. A device

comprehension model takes textual descriptions of device behavior and constructs representations of device function by recognizing how the behaviors in the description can be combined to describe complex behavior.

1.5 FONM and the EDISON Project

The EDISON project at UCLA is a study of computational invention and improvisation in the domain of naive physics. The project consists of two goals: (1) to design a representation approach for describing the behavior, function, and use of mechanical devices, and (2) to construct an integrated process model which demonstrates the manipulation of mechanical device representations in reasoning tasks necessary for creative problem solving, particularly mechanical invention. FONM addresses the first goal of the EDISON project, by providing an integrated approach for representing mechanical devices which supports the knowledge required for creative problem solving tasks. Although a single process model has not been developed using FONM, the programs EDEXP and EDCA were developed and demonstrate some of the range of reasoning about mechanical devices needed to satisfy the long-range goals of the EDISON project.

1.6 Limitations

FONM is designed to support the implementation and testing of theories of mechanical comprehension, diagnosis, improvisation, and invention. These theories require support from visual, spatial, environmental and other types of knowledge. To provide a complete theory of knowledge for each domain is beyond the scope of FONM. However, a closed theory for each domain is *assumed* and the knowledge from each domain are implemented in terms of *ad hoc* rules and procedures. The assumption of a closed domain theory allows the processing models that implement FONM to identify what is required from the domain and interactions between reasoning domains, without implementing a complete, general model for the domain. FONM can specify the general cognitive architecture that is required for mechanical understanding without completely implementing each individual piece. For example, consider the shape of the simple bottle opener illustrated in Fig. 1.1 below:

Figure 1.1 Complex shapes such as the curvature on the ends of a bottle opener are not explicitly represented in FONM.

The bottle opener is made of a single piece of metal, stamped and shaped in a complex way. If FONM had to explicitly describe each line, shape, and volume in order to describe the object's statics, dynamics, and pragmatics, then device representations would be very complex, and the similarities which objects share would be more difficult to discern. The detail associated with complex shape is not necessary for the naive mechanical reasoning tasks FONM is designed to support. What is necessary is that FONM represent those characteristics that directly affect a device's function. When an object such as the bottle opener is represented with FONM, the shape which is traditionally associated with the object is presented to the reader as a reference, with the parts labeled for clarity. FONM has no cor-

responding representation for complex shapes, access to pictures or other spatial descriptions, nor can it describe the paths of motion of the object. Instead, the bottle opener shown in Fig. 1.1 is represented with symbolic structures which describe the abstract patterns associated with the object which affect its dynamics and pragmatics.

In FONM, the bottle opener is represented as a single object. The blunt and sharp ends, the center, the holes, and the hooked pieces underneath are described only by their relative shape, size, and location. These *regions* play roles in the representation of device dynamics and pragmatics.

1.7 Representation of Knowledge

The representation of knowledge in FONM follows the four design principles shown below:

Knowledge structures follow a canonical form. Canonical form means that two concepts which are the same type of knowledge will be represented with the same structure, will have the same roles, and will share the same semantic meaning.

The breadth of knowledge is captured in primitive taxonomies. A *primitive* represents the lowest level at which inferences are made in a system. A *taxonomy* refers to a finite set of class specializations, within the canonical form, which completely describe the knowledge type. A taxonomy of primitives provides a finite and mutually exclusive set of building blocks for representing inference-related knowledge.

All inferences are made explicit by the representation. The representation of knowledge incorporates explicit reference to the normal conclusions to be made about the knowledge.

Knowledge structures are organized hierarchically. The different kinds of knowledge used to represent the knowledge domain should be constructed in layers so that the lower-level primitives can be used as explanations for what happens in the higher level constructs, and that the higher-level constructs provide expectations for what lower-level primitives will follow.

The following sections will describe the knowledge structures in this dissertation and the links which describe their dependencies.

1.7.1 FONM Knowledge Dependencies

FONM consists of 11 representation constructs which are used to describe device statics, dynamics, and pragmatics, as shown in Fig. 1.2. Of these, the action, script, plan, and goal constructs and their dependencies are defined and described in [Schank and Abelson, 1977]. The FONM specific structures are the phys-obj, state (quant, orient, place, and region), process, and function. The quant, orient, place and region constructs support the representation of device statics. The dependencies between these knowledge constructs are

illustrated graphically (called a *knowledge dependency graph*) in Fig. 1.2

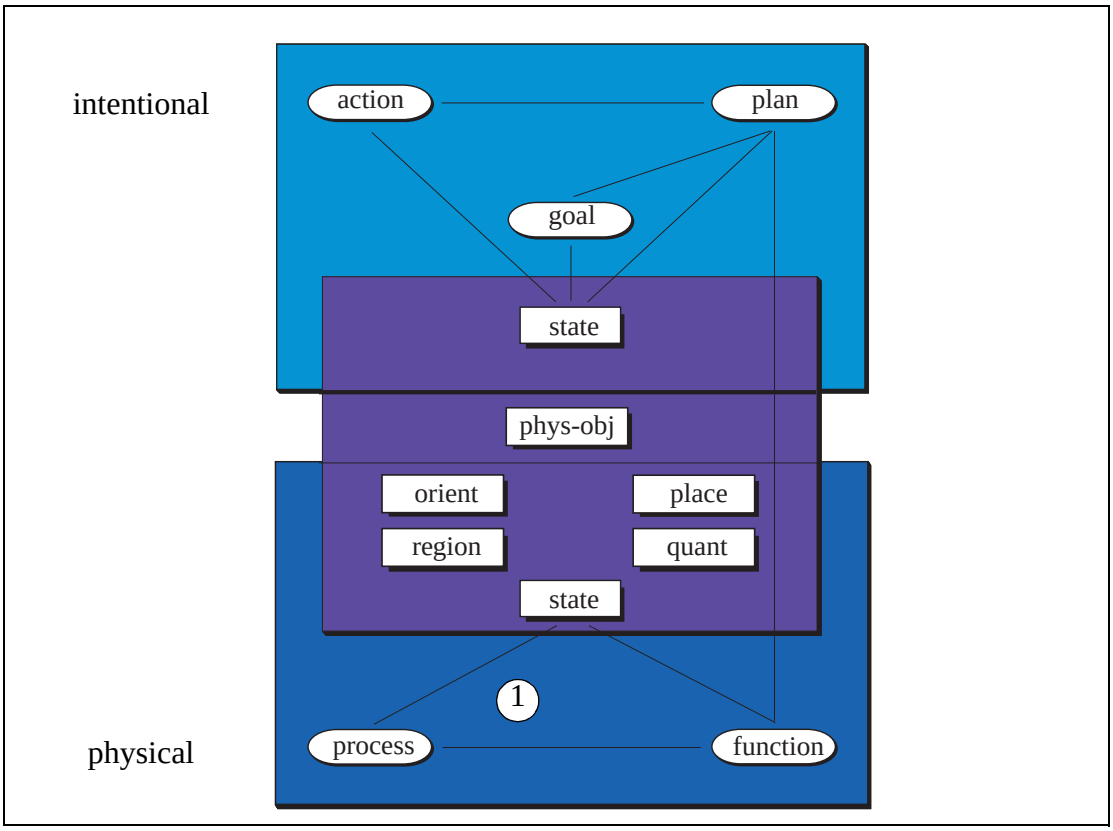


Figure 1.2 FONM knowledge dependency graph.

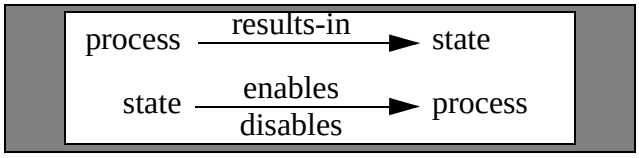
A knowledge dependency describes a causal relationship between two knowledge classes and is identified as an arc between them (e.g., between processes and states, at 1). The figure is divided into two components: (a) an intentional component, which describes knowledge dependencies between the goal, plan, action, and state classes, and (b) a physical component, which describes dependencies between function, process, state, region, orient, place, and quant classes. The phys-obj class is shown straddling the two components, because its dependencies are interpreted differently in each component but it always represents the same object. Two state classes are depicted in the figure, one which has dependencies with intentional knowledge classes and the other which has dependencies with physical knowledge classes. In FONM, object properties are represented with states and quantities.

The FONM representation constructs depicted in Fig. 1.2 are divided into two class types: (1) dynamic classes, and (2) static classes. *Dynamic* classes represent the cause of state changes (e.g., process, function and action), while *static* classes represent the participants in the change (e.g., phys-obj, region, orient, place, and quant) or the results of change. The knowledge dependencies associated with dynamic classes are the ones which affect the ability to make inferences about device statics, dynamics, and pragmatics:

(1) process-state dependencies
(2) function-state dependencies
(3) intentional dependencies

1.7.2 Process-State Dependencies

Device dynamics describes the types of behavioral interactions which bring about changes in object behavioral states (position, restraint, applied force, internal force, and size). The causal relationship between process and state classes is represented with the following links:



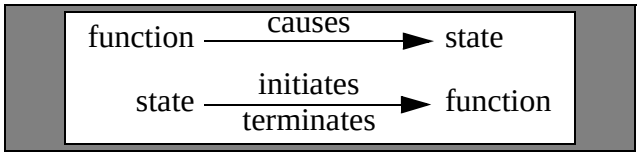
(1-1)

(1-2)

These relationships are also shared with the dependencies between actions and states. The *results-in* link indexes a process with the state which is associated with the *to* role in the process frame, while the *enables* (and *disables*) link is associated with the process *from* role (these are used in Chapter 3). Dependencies between classes are bi-directional, so there are similar links for the opposite direction (e.g., state to process links). For example, in Eqn. (1-1), the opposite relation is state *results-from* process.

1.7.3 Function-State Dependencies

Device dynamics also defines the types of behavioral interactions which bring about changes in object functional state as a result of behavioral process sequences. The causal relationship between a function and a state is represented with the following links:



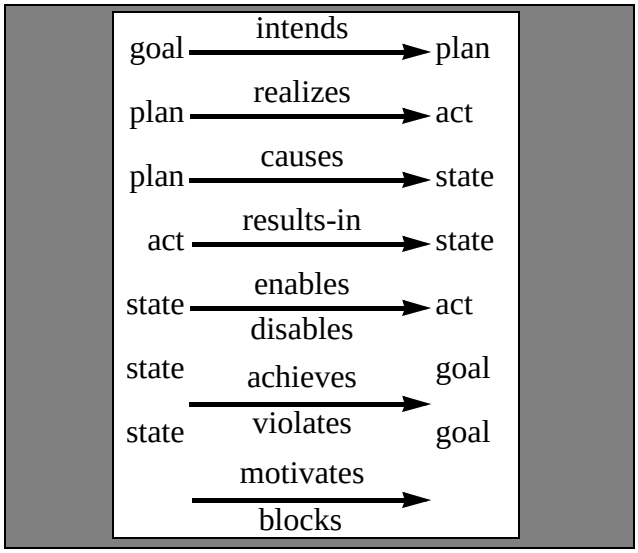
(1-3)

(1-4)

1.7.4 Intentional Dependencies

Some of the knowledge dependencies between classes in the upper portion of have been adopted or modified from Dyer;s I-links [Dyer 1983] and Reeve's I-links [Reeves 1991],

as shown below:



(1-5)
(1-6)
(1-7)
(1-8)
(1-9)
(1-10)
(1-11)

1.8 Notational Conventions

Textual references to objects which appear in illustrations are displayed in `courier` font to distinguish them from regular text. The following convention is used when new knowledge classes are introduced and illustrated:

Figure 1.3 Knowledge class naming convention.

where "class" means the knowledge class name, "subclass" and "specialization" refer to subclass names, and "token-name" refers to the reference name used for discussion purposes. The asterisk means that zero or more subclasses may be linked in this way. When these classes are first introduced, the long form shown above is used. When a class is later discussed, only the specialization name may be used. When the class instance is later discussed, only the token-name is used. Table 1.1: presents the class names, their representational abbreviations, and examples as they are found in the text of this dissertation.

Knowledge Class	Class Name Abbreviation	Example of Class Instance
action	act	act:propel-push
function	func	func:pry-open
goal	g	g:open-container

Table 1.1: Notation for knowledge class examples.

Knowledge Class	Class Name Abbreviation	Example of Class Instance
orient	ori	ori:horizontal-bw
phys-obj	o	o:linkage-sp
place	pla	pla:above-bt
plan	p	p:crack-open-container
process	proc	proc:motion-linear-sp
region	reg	reg:end-p1-point
state	st	st:force-sp
quant	q	q:force-lbs

Table 1.1: Notation for knowledge class examples.

1.8.1 Slot Fillers

Fig. 1.4 illustrates the convention used for depicting slot-filler objects

Figure 1.4 Template for slot-filler notation in knowledge class representation examples.

where "token-name" is bolded, slot names are in lower case, and slot fillers are in bold uppercase. When slot filler notation is depicted graphically, the same notational conventions apply. Slot fillers are always depicted in a box, with the class type depicted by box shape

as shown in Fig. 1.5 below.

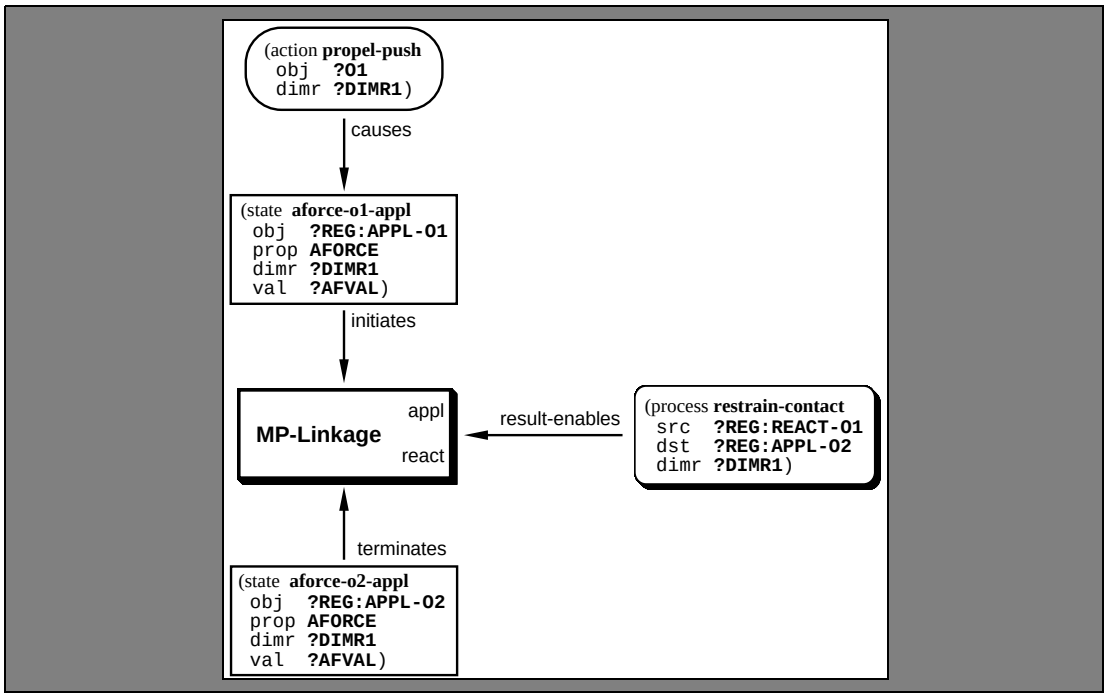


Figure 1.5 Frame notation used in FONM representation examples.

Rounded rectangles are used to depict process classes, such as `restrain-contact`. Rounded ovals are used to depict actions, such as `propel-push`. Rectangles are used to represent states, such as `prox-02-appl`. Class dependencies are illustrated with arrows, with the arrow pointing in the direction of causality, and the relation name next to the originating structure, as in `result-enables`.

1.8.2 Pattern Variables

The convention of preceding the variable-name with a question mark (?) is followed in all memory schemata. For example, if a variable representing a "human" class fills the 'actor' slot name of an "action," then it would be illustrated as follows:

Figure 1.6 Template for variable representation in slot-filler examples.

where `propel-push.345` is the token-name and `hum` is a variable name. In all EDISON processing models the question-mark notation is expanded to *(*var* var-name)* for pattern-matching and variable-binding purposes.

1.9 Guide to the Reader

This dissertation is divided into three major parts. Part 1 is a presentation of the FONM approach for representing mechanical devices. Part 2 discusses the processing applications which FONM was designed to address, and the process models (EDEXP and EDCA) which were implemented while developing the approach. Part 3 is an evaluation of FONM as a representation approach, and discusses its (1) relation to other research, (2) shortcomings, (3) implications, and (4) future.

Each chapter is organized by its major points and an example at the beginning, so that the reader can skim the introduction of the chapter to get an overview, and return later for a detailed treatment. Readers interested in an overview of FONM should read the first section of the chapters in Part 1 and Part 2, and the conclusions chapter. Those who are interested in knowledge representation should read chapters 2, 3, 4, and 5 of Part 1, while readers interested in language comprehension and problem solving should read chapters 6 and 7 in Part 2. Theoretical discussions and related research are contained in Part 3.

Chapters 2 - 5 discuss device statics, dynamics, and pragmatics of two simple devices: a screwdriver and a nutcracker, and then present representations for a number of devices at the end of each respective chapter. Chapter 2 discusses the notion of *device statics*, which describes the properties and composition of simple mechanical objects.

Chapter 3 presents the notion of *low-level device dynamics*, which describes the structure of device behavior. Device behavioral process primitives represent the abstract structure of mechanical behavior. A taxonomy of mechanical behavior is presented, and the role of behavioral process primitives in structuring memory for access to knowledge about objects, device function, and pragmatics is discussed.

Chapter 4 presents the effect of device statics and dynamics on the structure of *high-level device dynamics*, or device function. Device machine primitives represent the abstract structure of machine function. A taxonomy of machines is presented, and the role of machine primitives in structuring memory for access to knowledge about device statics, dynamics, and pragmatics is discussed.

Chapter 5 presents the notion of *device pragmatics*, which describes the structure of device use and experience in problem solving. Device use plans represent the abstract structure of device application in a problem-solving context. Behavioral and mechanical subgoals, and device attributes, are patterns that represent the components of device use which relate device-use plans and device functions. Types of device-use plans, subgoals, and attributes are presented, and their role in structuring memory for access to knowledge about device function, dynamics, and statics is discussed.

Chapter 6 presents EDEXP, a process model (based on FONM constructs) which mutates simple door representations and constructs relationships describing the behavioral and functional dependencies between doors and doorhinges. An annotated trace of door hinge mutation by EDEXP is included in the chapter.

Chapter 7 presents the EDCA process model for reading device behavior descriptions and constructing representations of device function. The annotated trace of EDCA (used to read an English description of a toy gun) is included in the chapter.

Chapter 8 presents related areas of research and their impact on the development of the FONM device representation and related process models.

Chapter 9 discusses the theoretical foundations and limitations of the FONM approach, and an evaluation of the associated process models.

Chapter 10 discusses directions for future research, some applications associated with

this research, and conclusions.

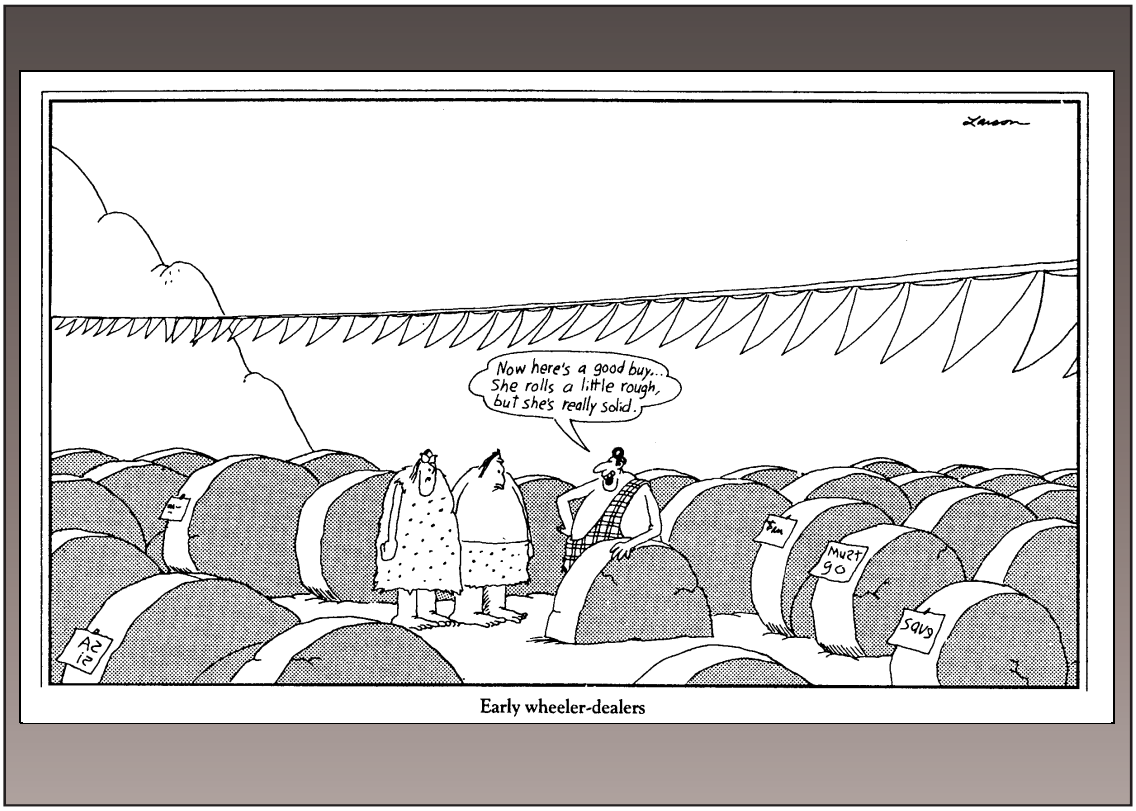
There is a glossary and an appendix in this dissertation. The Appendix presents static, dynamic, and pragmatic representations for the devices (18) described in this dissertation.

Part I

Representing Mechanical Devices

There are three reasons for building computer representations of mechanical devices: (1) identify the types of structures used, so as to extend the types of inference that can be applied to device-related applications, (2) identify structures that globally organize the problem, so as to control knowledge access and inferencing, and (3) find structures that focus attention on the functionally salient elements of a problem, so that suitable plans and devices can be identified or constructed. In FONM, the structures are associated with a physical object and its function, because an object is used for a reason (or purpose), to achieve some end, and an object function is applied and produces specific behavior. Function organizes how objects are viewed by those who use them and by how objects behave regardless of how they are viewed.

Representing mechanical problem-solving requires access to diverse types of knowledge, and to their interactions, on many levels. The central problems for any computational system are how information is represented, organized, accessed, and manipulated. This part of the dissertation addresses the first three problems, while Part 2 addresses the fourth point. Frame based approaches to knowledge representation ([Minsky 1975], [Schank and Abelson, 1977]) construct patterns as abstract knowledge templates that declaratively represent clusters of concepts, variables, and relations. As information is obtained, the patterns are instantiated and are then used to provide inferences.



Early wheeler-dealers

Chapter 2
Device Statics

Devices are used to perform problem solving tasks. Devices can range from simple geometric shapes to complex organizations of interacting subsystems. The description of a device at rest, its construction and appearance, is important in determining whether it can support a particular task. The device's *construction* refers to the composition and connectivity of its components. The device's *appearance* refers to the shape and size of its components and their features, and must support the device's behavior and remind the problem solver of the task. There are four elements of a device static description: the composition of its components, the appearance of its components, the orientations of its components, and the connectivity of its components. Device function is dependent on the orientations between and connectivity of its components. Device use is dependent on the functions the object can effect and also on how the object appears to the problem solver. Object composition, appearance, orientation, and connectivity are aspects of the static device description, and their interaction is called *device statics*.

In this chapter, the representation constructs which are used in FONM to describe device statics are presented. Every device and device component can be described as a collection of individual objects, so the discussion begins with a description of individual objects and their physical characteristics. The discussion ends with the presentation of static representations for fifteen mechanical devices of increasing complexity. Five topics are addressed in this chapter:

(1) Physical objects and individuals
(2) Geometric primitives
(3) Object primitives in FONM
(4) Relational characteristics and connectivity between objects
(5) Device static representations

2.1 Physical Objects and Individuals

Regardless of its overall complexity, the components of every device can be recursively decomposed to a level of individual objects. An *individual object* is defined as a set of physical characteristics associated with an object type. A stick, or rock, is an individual object, because to describe it further requires a description of its material properties, its size, and its shape.

2.1.1 Object Physical Characteristics

In FONM, an object has two types of physical characteristics: its *properties* and its *regions*. Object properties are divided into those related to the object's material composition, which determine how the object can behave under various conditions, and those related to me-

chanical perturbations, which determine what behavior has happened or can happen. Object properties are represented with physical states. Object regions describe general locations on the object by their shape, and size, and determine how and where the object can be connected and manipulated.

Physical States

An object can be represented as a collection of physical states, each of which describes a different property and its value. Physical states are represented using a knowledge structure with four roles: obj, prop, dimr and val. The *obj* role refers to the physical object whose property is being represented, the *prop* role is associated with a symbolic property name, the *dimr* role is associated with the applicable dimension and direction for which the property value is valid, and the *val* role is associated with the knowledge structure representing the quantity associated with the property value. In order to represent physical states, these roles and their fillers must be described.

Properties

Every object has a finite number of physical properties. In FONM, four material properties are used to represent object composition:

(1) material stiffness
(2) material density
(3) material strength
(4) material opacity

Material stiffness is a measure of whether an object can transmit mechanical force to other objects. Density is a measure of whether an object can restrain another object's motion. Strength is a measure of whether and how an object will absorb energy. Opacity is a measure of whether an object can transmit sensory information.

In addition, five properties are used to represent the types of mechanical behavior an object can exhibit:

(1) position
(2) restraint
(3) applied force
(4) internal force
(5) size

Position is associated with object motion. *Restraint* is associated with object connectivity. *Applied force* is associated with motion and force transmission. *Internal force* is associated with energy storage. *Size* (and shape) is associated with deformation.

Dimension and Direction

Each object property has an applicable dimension and direction. In FONM, dimension and direction are combined in a state's *dimr* role. The values associated with dimension are those for the cartesian axes (x, y, z) and their cylindrical equivalents (r, z), as shown in Table 2.1**:**

	X/Z	Y	Z	R
translation	along-length	along-width	along-depth	along-radius
rotation	about-length	about-width	about-depth	about-radius

Table 2.1 : Symbolic representations for the six cartesian coordinate degrees of freedom.

Each of the entries in Table 2.1 represents a motion *degree of freedom*. References to these dimensions will be used to represent behavior enablement and disablement. For example, when a restraint state is represented, the dimension (and direction) specify degrees of freedom which are fixed with respect to motion. When a force state is represented, the dimension (and direction) specify the degrees of freedom which, if unfixed, will result in motion, and, if fixed, will result in force transmission.

What is normally termed the theta dimension is captured by the ABOUT-LENGTH dimension.

Direction is described using a dimension and value. The value can be POS or NEG, with respect to the object's center of gravity (CG), or it can be an object or object region. A POS value indicates the direction toward the object CG, while a NEG value indicates the direction away from the object CG, (see Fig. 2.1a,b).

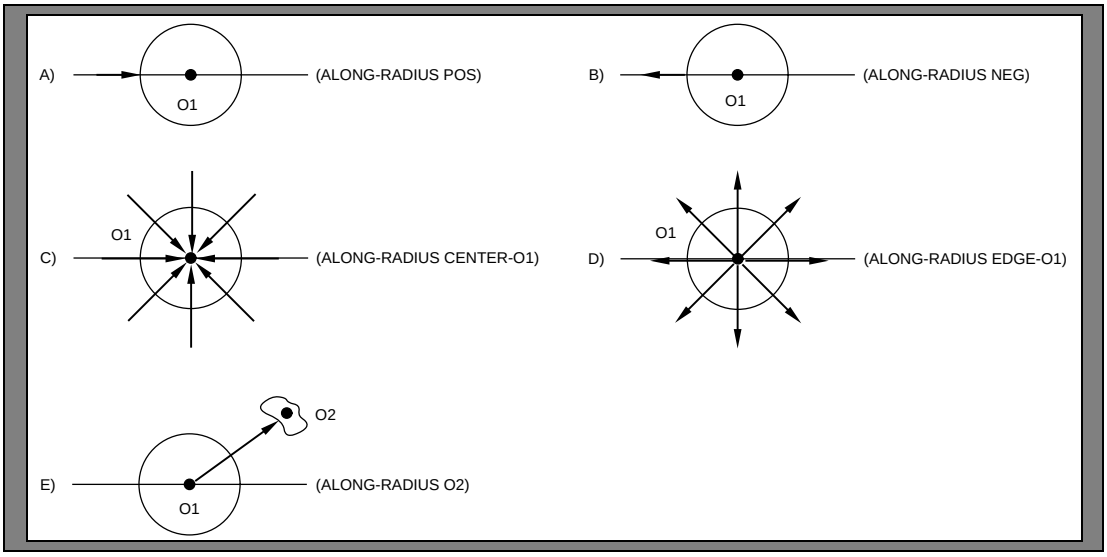


Figure 2.1 Direction in FONM. Examples A and B use a dimension and value, and examples C, D and E use a dimension and an object or region.

In Fig. 2.1, six ways to represent direction with radial dimensions are illustrated. O1 represents a circular object and the arrows represent directions. The same mechanisms are used to represent direction for translational dimensions. Consider the spoon illustrated in

Fig. 2.2

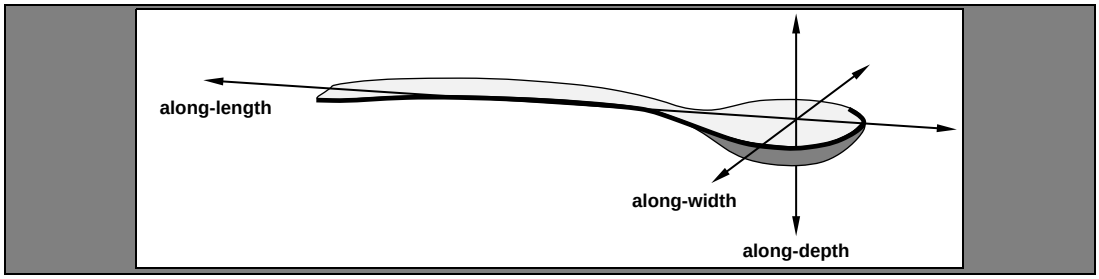


Figure 2.2 Cartesian dimension reference for a spoon.

Using the notational convention for representing direction illustrated in Fig. 2.1a, b, an applied restraint on a spoon handle, caused by a person holding the spoon, can be represented (using slot-filler notation) as follows:

```
(state REST-SP1
  obj      SPOON.1
  prop     RESTRAINT
  dimr     ((ALONG-DEPTH NEG))))
```

Figure 2.3 Representation example for restraint states.

The dimr role is bound to ((ALONG-DEPTH NEG)), which describes a restraint applied away from the spoon handle CG in the spoon's thickness dimension (i.e., cartesian z or global downward). Note that there is no value for the restraint state represented. The restraint dimension and direction are equivalent to the state value.

The representation of direction using a dimension and a reference location or object, as depicted in Fig. 2.1c-e, is shown in Fig. 2.4. Consider a soup spoon being lowered into some soup. The spoon is propelled toward the soup, which causes an applied force state on the spoon handle and enables motion.

```
(state FORCE-SP1
   obj          SPOON.1
   prop         AFORCE
   enables      (process MOTION-LINEAR-SP1
                   dimr     ((ALONG-DEPTH STUFF-SOUP))))
```

Figure 2.4 Representation example for explicit direction.

The force FORCE-SP1 enables spoon motion, represented by the process MOTION-LINEAR-SP1. The motion dimr role is bound to ((ALONG-DEPTH STUFF-SOUP)), which means 'toward the soup.' In this example, the dimension of motion is simply described because it coincides with a spoon axis. In the more general case, however, the dimension may be more difficult to describe, since motion toward the soup might not be along a know object axis. A representation which alleviates this problem references the di-

rection but ignores the dimension:

```
(state FORCE-SP1
   obj         SPOON.1
   prop        AFORCE
   enables     (process MOTION-LINEAR-SP1
                  dimr     ((STUFF-SOUP)))))
```

Figure 2.5 Representation example for implicit direction.

When a dimension is represented with no direction, both directions are assumed. In the representation of SPOON.1 length below, ((ALONG-LENGTH)) describes both directions of the spoon's longitudinal axis:

```
(state SIZE-AL-SP1
   obj         SPOON.1
   prop        SIZE
   dimr        ((ALONG-LENGTH))
   val         QUANT.1)
```

Figure 2.6 Representation for entire dimensions.

Quantity Values and Quantity Spaces

The state *val* slot refers to a quantity and is filled by a knowledge structure called quant. Quant has three roles: state, unit, and val. The *unit* role is used to distinguish between and enable translations between quantities. The unit filler always takes on the symbolic name for the value being described. The quant *val* slot can be represented with a quantitative value, as a point in a one-dimensional quantity space. A *quantity space* is a continuum of values associated with a specific unit type, such as inches or pounds. Each point on the quantity space represents a value of an object property. The quantity space is bound by the

maxima and minima of known values.

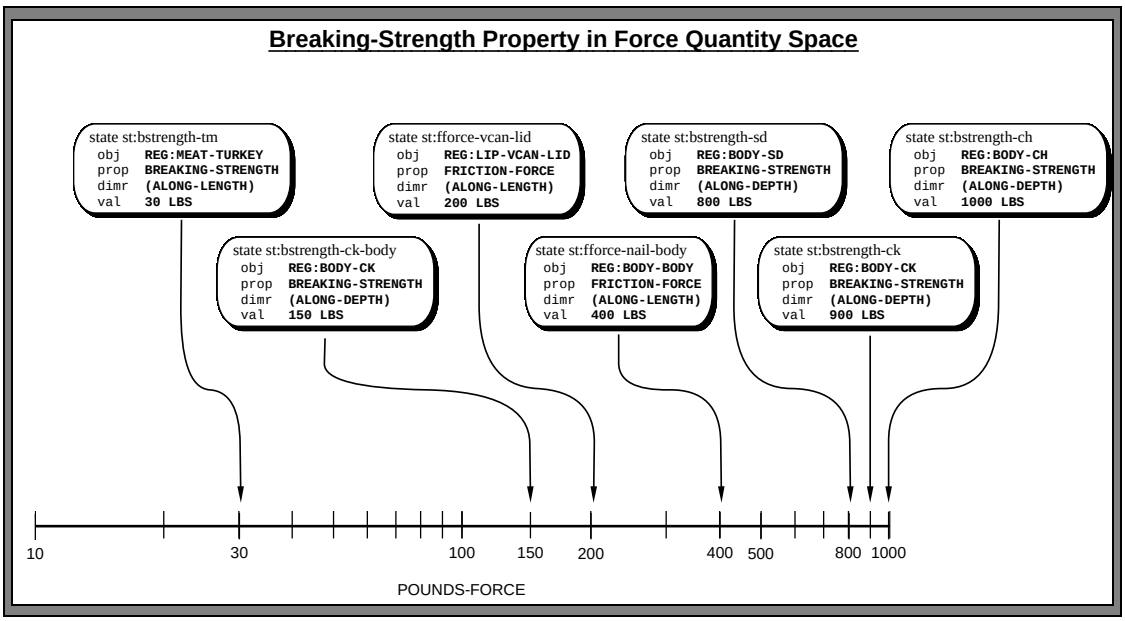


Figure 2.7 A portion of the quantity space for force, measured in pounds.

Fig. 2.7 illustrates a one-dimensional quantity space for pounds-force (a logarithmic scale) and a number of object states plotted on the scale. Object property values, represented as quantities, are related to other objects, and to other quantities, on a quantity space. The breaking strengths for three devices (screwdriver, a claw hammer, and a carving knife) and some turkey meat are plotted above on the pounds-force quantity space. Two friction force values (the friction force between a varnish can lid and its vessel, and a nail and its medium) are also represented on the line.

The quantity space enables a number of useful comparisons. For example, the breaking strengths of the devices plotted can be compared. Second, this quantity space provides a means for comparing breaking strengths and friction forces through the common unit of measure. Finally, a quantity space also provides a simple way to compare values in different units of measurement, such as pounds and kilograms, by their common objects.

A quantity value can also be represented qualitatively, using a simple (<NOM, NOM, >NOM) scale. In this alternate case, the *unit* role is empty. For example, a significant characteristic of a broken ruler is its change in size (shorter). Without knowing the exact size change, the new size can be represented qualitatively as shown in Fig. 2.8:

```
(state SIZE-AL-RUL1
  obj      RULER.1
  prop     SIZE
  dimr     ((ALONG-LENGTH))
  val      (quant
              state    SIZE-AL-RUL1
              unit     NIL
              val      <NOM)))
```

Figure 2.8 Representation example for quantity.

In this case, there is no way to determine what the new length of the ruler is with respect to other devices, but it may be suitable for function-related inferences, such as the ruler not being suitable for precise measurement. It should be noted that this representation doesn't make mention of which piece of the ruler is being represented, only that the ruler is no longer its original size.

Object Regions

In FONM, appearance is represented with a knowledge structure called a region. Regions describe general areas or volumes of an individual object by their shape, size, and location. Regions are used to represent objects having complex shapes.

Regions are used to describe objects having locations which have something to do with their function and behavior, but which may not require specific reference in a particular coordinate system. For example, consider a table top. In FONM, the table top is represented as a single flat surface, and all points on it can be applied equally well to the support-related tasks that any specific point can. In this respect, the region is a general location specifier for behavior, and the number of points that need to be represented in order to make inferences about table top behavior is dramatically reduced. Now consider a gear tooth. The tooth may be a complex, repetitive, shape but that shape can be described simply as a convex surface on the edge of a wheel. The fact that it is repetitive allows us to ignore the other teeth, since they all can be used to accomplish the same tasks. Stating that the gear tooth is located on the edge, but not where on the edge, does not detract from the ability to say what behavior the tooth can engage in, or what the function of the gear will be.

A region knowledge structure is defined by four roles: obj, shape, size, and loc. The *obj* role references the object in question. The *shape* role is associated with the curvature of the region. The *size* role is used as a comparison metric with other regions. The *loc* role is used to describe the general vicinity of the region with respect to the object's center of gravity or to other regions.

Shape

A region's shape is its most apparent feature. In FONM, regions are described as surfaces or surface intersections, and shape is represented as a the curvature of the surface or intersection. A *surface* region describes an affected volume. There are three types of surface regions defined by their general curvature: concave surfaces are called *indentations*; and are associated with containment volumes, flat surfaces are called *faces*; and are associated with support areas or volumes, and convex surfaces are called *protuberances*, and are associated with path interference and confinement. A tree illustrating the region classes used in

FONM, with respect to specialization along these metrics, is shown in Fig. 2.9

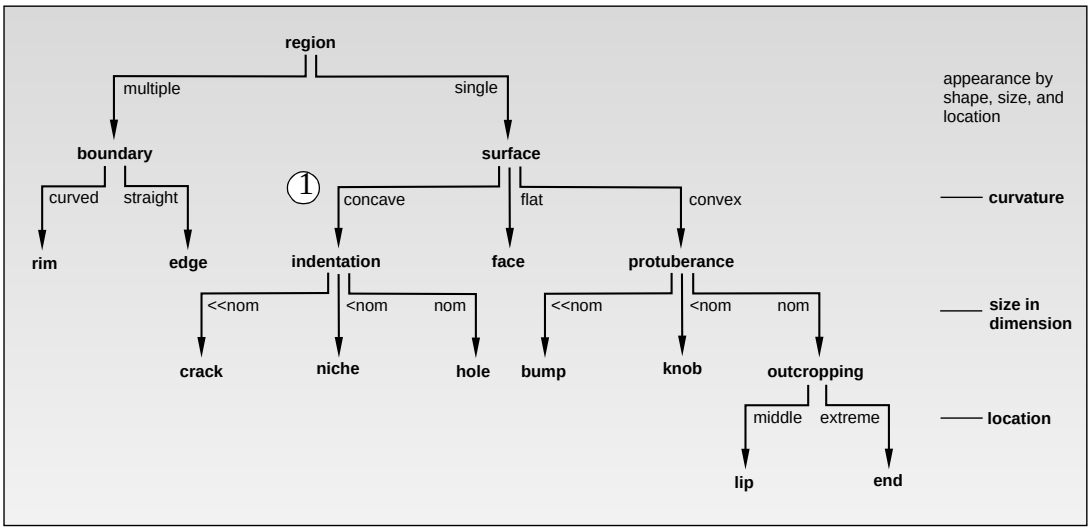


Figure 2.9 Region hierarchy in FONM.

The items along the right hand side of Fig. 2.9 describe the different characteristics of a region (i.e., curvature, size, and location). For each of these characteristics, the tree is differentiated by specializing along a particular value. For example, at branch (1) the curvature is concave and the subtree is associated with indentations.

Surface intersections (or junctions) are called *boundaries*. In Fig. 2.9, an edge is represented as a straight boundary (e.g., between two faces), and a rim is represented as a curved boundary (e.g., between an indentation and a protuberance).

Location

A region's general location is described with respect to some known location on the object. A location is represented with a knowledge structure called *location*. Locations have two roles: ref and dimv. The *ref* role refers to a reference region or to the object's center of gravity. The *dimv* role is used to locate the region, with respect to the reference, with dimension/value pairs. The values used in FONM are the discrete set: FAR-LEFT, LEFT, NEAR-LEFT, MIDDLE, NEAR-RIGHT, RIGHT, and FAR-RIGHT. Using this notation, the eraser end of a pencil is represented as a convex region located along the length of the pencil at its extreme value:

```
(region PR-END1-P1
   obj         PENCIL.1
   shape       CONVEX-BLUNT
   loc         (location
                  dimv     ((ALONG-LENGTH FAR-RIGHT))))
```

Figure 2.10 Representation example for region location values.

In this example, the region's size is ignored, and the ref role is empty because the value refers to the object CG (all regions). In FONM, no distinction is made between the extrema, such as FAR-LEFT and FAR-RIGHT, unless another region is represented. In the pencil

case, there is another end, the pointed end:

```
(region PR-END2-P1
   obj          PENCIL.1
   shape        CONVEX-POINTED
   loc          (location
                   dimv     ((ALONG-LENGTH FAR-LEFT))))
```

Figure 2.11 Representation example for an end region.

which now enables a distinction with respect to the other end region's location. The location notation follows the same convention as does dimension/direction for state descriptions.

A pencil has a third region called a handle. The handle location can be anywhere along the length of the pencil, since a person could hold the pencil anywhere along its length equally well.

```
(region PR-HANDLE-P1
   obj          PENCIL.1
   shape        CONVEX
   loc          (location
                   dimv     ((ALONG-LENGTH))))
```

Figure 2.12 Representation example for a body region.

If a dimension is referenced with no value, as with the pencil handle above, then the entire dimension is assumed.

Size

Regions are also distinguished by their size, but the values for size used in this context refer to the region type rather than a unit type. The baseline assumption is that the region's size is the same in all dimensions, so the size state value is based on selecting the dimension or dimensions which are used as a comparison. The first dimension given is the reference size and the others, if given, as values, refer to it. For example, a crack is an indentation which is very long with respect to its width. The size of a crack is represented with respect to its width, so the crack size in the ALONG-WIDTH dimension is the reference. The size value of the crack in the other dimensions is then taken relative to the crack's width: the size along its length is much greater than nominal (>>NOM) and the size along its depth is greater than nominal (>NOM).

```
(state CR-CM1
  obj      IND-CR-CM1
  prop     SIZE
  dimr     ((ALONG-WIDTH))
  val      ((ALONG-LENGTH >>NOM)
            (ALONG-DEPTH  >NOM)))
```

Figure 2.13 Representation example for a crack indentation size.

2.2 Object Types

2.2.1 Linkages and Stuff

There are two classes of individual objects in FONM: *linkages* and *stuff*. Linkages make up the components of devices, and stuff is everything else. Linkages and stuff are differentiated by their material stiffness properties. A *linkage* is an elastic object; meaning that it is made of a material capable of transmitting mechanical force in at least one direction along at least one of its dimensional axes. Most mechanical objects are linkages, because they can transmit force along entire dimensions. For example, a pipe is a linkage in all dimensions, and directions. Wires, cables, and ropes are examples of linkage objects in a single dimension/direction, because they can transmit force under tension but not under compression.

Objects are represented with a phys-obj knowledge structure which is associated with an object type and its physical characteristics. Consider the representation of a linkage object shown below.

```
(phys-obj LINKAGE-OBJECT
   has-phys    (MS-LINKAGE-STIFFNESS
                LINKAGE-END1
                LINKAGE-END2))

(state MS-LINKAGE-STIFFNESS
   obj         LINKAGE-OBJECT
   prop        MATERIAL-STIFFNESS
   val         (quant ELASTIC-MS
                  units    PPI
                  val      >NOM))

(region LINKAGE-END1
   loc         (location LOC1-LINKAGE
                  dimv     ((?DIM      FAR-LEFT))))

(region LINKAGE-END2
   loc         (location LOC2-LINKAGE
                  dimv     ((?DIM      FAR-RIGHT))))
```

Figure 2.14 Representation example for a linkage object.

Since the linkage object is defined by its ability to transmit force, only its material stiffness and the regions where it can transmit the force are included in the FONM representation. The unit for stiffness shown in the representation for `MS-LINKAGE-STIFFNESS` is pounds-per-inch, PPI. The nominal stiffness value is somewhat arbitrary, and is intended to represent the minimum stiffness which would preclude an object from being able to transmit mechanical force.

Stuff is used to represent objects which cannot transmit mechanical forces in any dimension or direction. Examples of objects which can be represented with stuff are liquids and gases, since they require a container (such as a piston and cylinder) to effect mechanical force transmission. Stuff may become a linkage through state change (e.g., thermodynamic), however, in FONM, non-mechanical effects on objects are not represented.

2.2.2 Physical Type and Geometric Object Primitives

In FONM, every linkage object has a physical type based on one of ten idealized geometric

objects: cylinder, column, filament, cone, sphere, block, plate, film, wedge, and pyramid. Each of these objects is defined by its (a) regions, (c) center of gravity, (d) size, and (e) shape, as illustrated in Fig. 2.15. Each object also has associated with it a set of dimensions for referring to its size and shape.

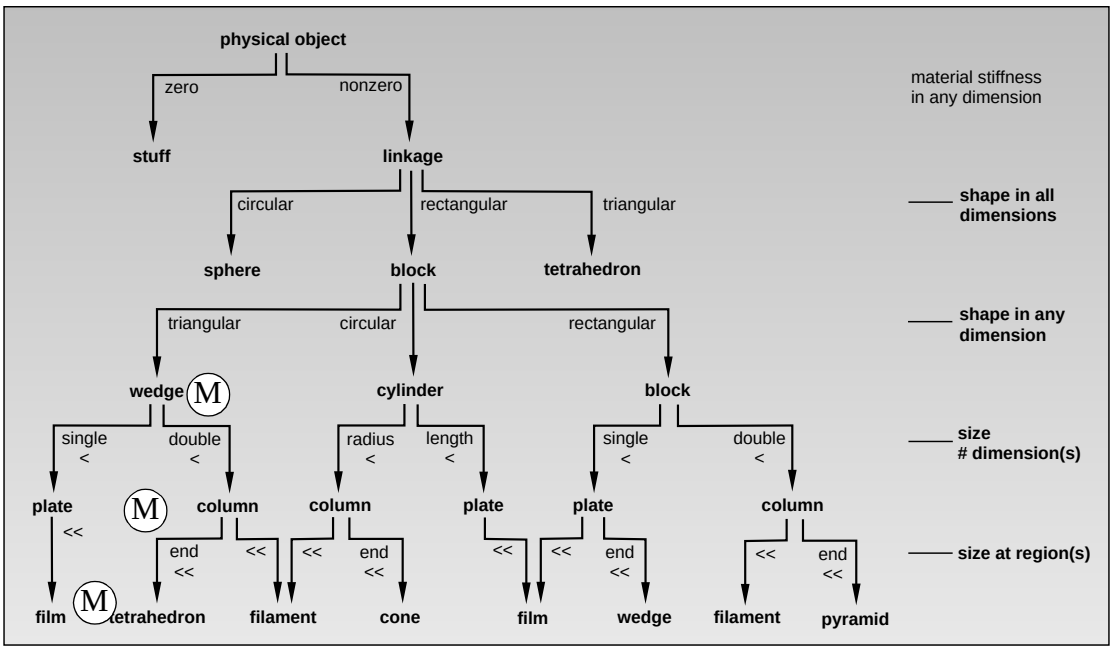


Figure 2.15 Geometric object hierarchy in FONM.

Fig. 2.15 is organized by shape and size specializations, similar to that of regions. Objects which have the same shape in every dimension (circular => "sphere", rectangular => "block", and triangular => "tetrahedron") are shown in the first tier. The "block" sub-hierarchy is shown expanded. It should be noted that this figure does not represent a unique tree. There are different ways to combine these specializations and come up with the same categories. For example, a tetrahedron can be described as a linkage object which is triangular in all dimensional views. A tetrahedron can also be described as a rectangular block that is first modified to be triangular in one dimension (a wedge), then squished and lengthened into a column, and then sharpened at one end. This path is labeled with (M)s. Note the repetitive nature of shapes in Fig. 2.15. Every shape leads to a plate and column, as well as to other standard shapes. This suggests that the shapes chosen in the first tier are acceptable geometric primitives as long as size modifications are allowed. The geometric primitives

chosen for FONM include size modifications.

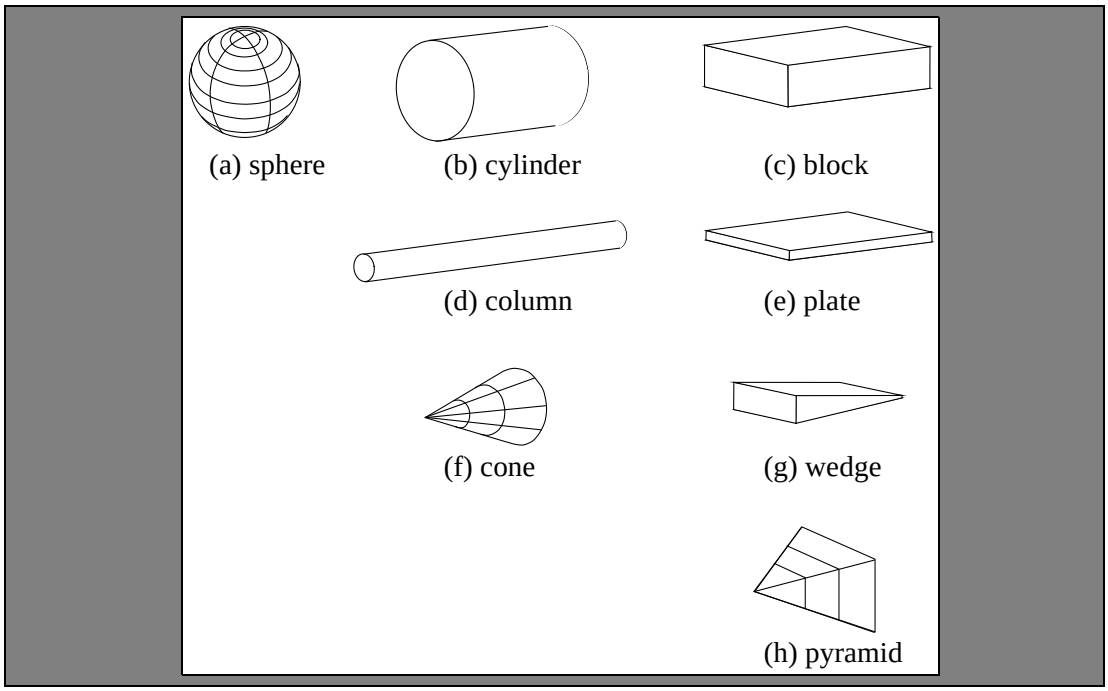


Figure 2.16 Geometric object primitives in FONM. Sphere, cylinder, column, cone, block, plate, wedge, and pyramid.

2.2.1 Cylinder

The cylinder is a fundamental geometric entity. An idealized cylinder is depicted in Fig. 2.17.

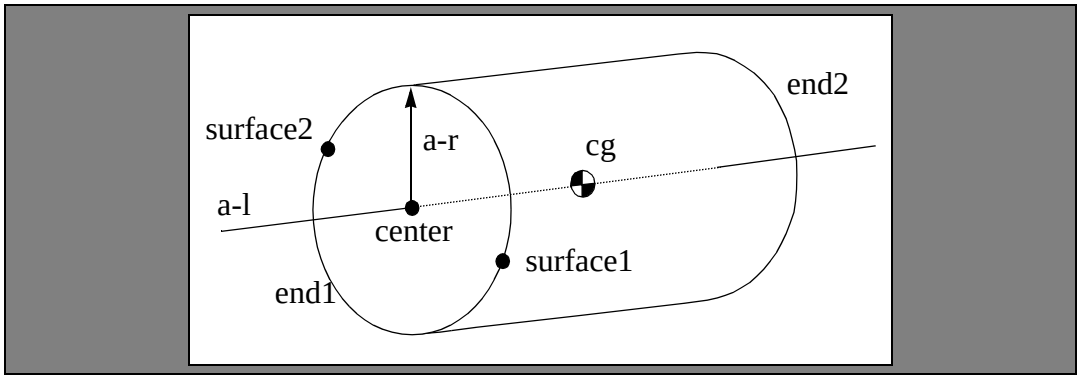


Figure 2.17 An idealized cylinder.

Cylinder Coordinate Axes

The cylinder is described with four dimensions: along-length (a-l), about-length (ab-l), along-radius (a-r), and about-radius (ab-r).

Cylinder Center of Gravity

Each object has a center of gravity (CG), which represents the location of the object in the absence of any other reference to object location. Center of gravity is not considered a region in FONM, because it is not associated with appearance. The cylinder CG is located at the mid-point on both translational dimensions (i.e., a-l middle a-r middle).

Cylinder Regions

The cylinder has five regions: a center [of rotation], two ends, and two surface points. The regions are located with respect to the object center of gravity. That is, the mid-points of

Name	Dimension	Value
center	along-radius	middle
end1	along-length	far-left
end2	along-length	far-right
surface1	along-radius	far-right
surface2	along-radius	far-left

Table 2.2 Cylinder Regions

the dimensions are coincident with the CG.

Surface1 and surface2 describe the outer surface of the cylinder as the endpoints of any particular diameter. Thus, surface1 and surface2 are position (ALONG-LENGTH) independent. The dimension and value entries in Table 2.2 represent the elements of the dimv location role, and the values are same as the symbolic notation presented for region location (e.g., MIDDLE).

Cylinder Size

The size of the cylinder is arbitrary, except for two requirements. First, the radial size is the same at all locations along the cylinder length. Second, the length and radial sizes are related as follows:

$$\text{(size along-length)} > \text{(size along-radius)} \qquad (2\text{-}1)$$

where the greater-than sign is a symbolic representation that means the size in one dimension is greater, but within an order of magnitude, of the other:

$$\text{(size along-radius)} > 1/10\ \text{(size along-length) and}$$
$$\text{(size along-radius)} < \text{(size along-length)} \qquad (2\text{-}2)$$

Cylinder Representation

The regions representations for the cylinder geometric object are shown below.

```
(phys-obj CYLINDER
   has-phys    (CYL-END1
                CYL-END2
                CYL-S1
                CYL-S2
                CYL-CENTER
                CYL-CG))

(region CYL-END1
   loc         (location END1-CYL
                  dimv     ((ALONG-LENGTH FAR-LEFT))))

(region CYL-END2
   loc         (location END2-CYL
                  dimv     ((ALONG-LENGTH FAR-RIGHT))))

(region CYL-S1
   loc         (location S1-CYL
                  dimv     ((ALONG-RADIUS FAR-LEFT))))

(region CYL-S2
   loc         (location S2-CYL
                  dimv     ((ALONG-RADIUS FAR-RIGHT))))

(region CYL-CENTER
   loc         (location CENTER-CYL
                  dimv     ((ALONG-RADIUS MIDDLE))))

(region CYL-CG
   loc         (location CG-CYL
                  dimv     ((ALONG-LENGTH MIDDLE)
                            (ALONG-RADIUS MIDDLE)))
```

Figure 2.18 Representation for cylinder geometric primitive.

The size characteristics of a cylinder geometric primitive can be specialized to represent three other geometric primitives: (1) circular plate, (2) column, and (3) cone. The plate and column (and filament) represent objects which have size specializations which are effected at both end regions of the cylinder, while the cone represents an object where the size specialization occurs at one region. A plate results from having a size in the along-radius dimension being much larger than that in the along-depth dimension. The column is the opposite case, where the size in the along-depth dimension is much larger than that in the along-radius dimension. The filament is simply an extreme case of a column. A cone, described below, is a cylinder specialization where the radius at one end is much smaller than that at the other end. This same specialization can be applied to a column to produce a point.

2.2.2 Cone

The idealized cone is defined as a cylinder where the radius at one end region is zero, as

shown below (Fig. 2.19).

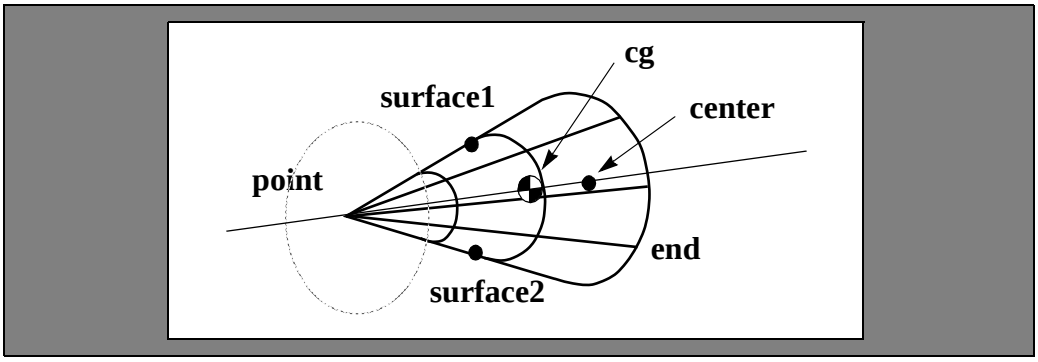


Figure 2.19 Idealized cone and regions.

The dotted line at the point end of the cone illustrates the end where the cylinder/column/ filament radius has been reduced. As a result, the surface regions (surface1 and surface2) have different radii along the length of the object, and the CG shifts towards the end.

Cone Axes and Size

The dimensions used to define a cone, and the sizes of a cone, are identical to those for a cylinder.

Cone Center of Gravity

The cone CG is located at the mid-point along the cone radius (a-r) and one-third the distance from its end (2/3 from its point).

Cone Regions

The cone has five regions differing slightly from those of a cylinder: a center [of rotation], a point, an end, and two surface points. The region locations, with respect to the cone CG,

Name	Dimension	Value
center	along-radius	middle
point	along-length along-radius	far-left middle
end	along-length	far-right
surface1	along-radius	far-right
surface2	along-radius	far-left

Table 2.3 Cone Regions

are shown in Table 2.3

Surface1 and surface2 describe endpoints of any particular diameter and change as the radius changes along the length of the cone.

Cone Representation

The cone representation is illustrated with respect to its differences from the cylinder At

```
(phys-obj CONE
   isa          CYLINDER
   has-phys     (CONE-END1
                 CONE-POINT
                 CONE-S1
                 CONE-S2
                 CONE-CENTER
                 CONE-CG
                 SIZE-CONE-POINT-AR))

(region CONE-END1
   loc          CYL-END2)

(region CONE-POINT
   loc          CYL-END1)

(region CONE-S1
   loc          CYL-S1)

(region CONE-S2
   loc          CYL-S2)

(region CONE-CENTER
   loc          CYL-CENTER)

(region CONE-CG
   loc          (location CG-CONE                    <-- (1)
                    dimv     ((ALONG-LENGTH RIGHT)
                              (ALONG-RADIUS MIDDLE)))

(state SIZE-CONE-POINT-AR
   obj          CONE
   prop         SIZE                                 <-- (2)
   dimr         ((ALONG-RADIUS))
   val          (quant POINT-RAD
                    val      <<NOM))
```

Figure 2.20 Representation for cone geometric primitive.

(1), the CG location has a symbolic location at the RIGHT value, which is somewhat arbitrary and intended to represent the shifting toward the end rather than be the true location of the CG. At (2), the radial size of the point region is represented symbolically, with respect to the object radial dimension, as <<NOM.

2.2.3 Sphere

A sphere is a fundamental geometric shape that is circular in every dimensional view. An

idealized sphere is illustrated in Figure 2.21

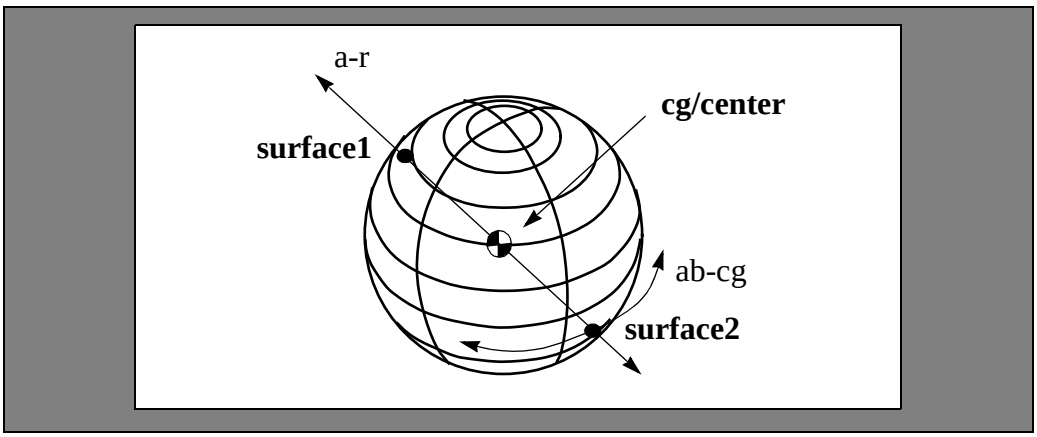


Figure 2.21 Idealized sphere and regions.

Sphere Coordinate Axes

The sphere has three dimensional axes, along-radius (a-r), about-radius (ab-r), and about-cg (ab-cg).

Sphere Center of Gravity

The sphere CG is located at the mid-point along the sphere radius, and coincides with the center region.

Sphere Regions

The sphere has three regions: a center [of rotation] and two surface points. The regions are

Name	Dimension	Value
center	along-radius	middle
surface1	along-radius	far-right
surface2	along-radius	far-left

Table 2.4 Sphere Regions

located, with respect to the sphere CG, as shown in Table 2.4. Surface1 and surface2 describe endpoints of any particular diameter.

Sphere Size

There are no size restrictions on sphere objects.

Sphere Representation

```
(phys-obj SPHERE
   has-phys    (SPHERE-S1
                SPHERE-S2
                SPHERE-CENTER
                SPHERE-CG
                SHAPE-SPHERE))

(region SPHERE-S1
   loc         (location S1-SPHERE
                  dimv     ((ALONG-RADIUS FAR-LEFT))))

(region SPHERE-S2
   loc         (location S2-SPHERE
                  dimv     ((ALONG-RADIUS FAR-RIGHT))))

(region SPHERE-CENTER
   loc         (location CENTER-SPHERE
                  dimv     ((ALONG-RADIUS MIDDLE))))

(region SPHERE-CG
   loc         SPHERE-CENTER)

(state SHAPE-SPHERE
   obj         SPHERE
   prop        SHAPE
   val         CIRCULAR)
```

Figure 2.22 Representation of sphere geometric primitive.

2.2.4 Block

A block is a fundamental geometric shape which is rectangular in all dimensional views. An idealized block is any rectangular solid with planar faces.A rectangular block is illustrated in Figure 2.23, however, a block can be any parallelepiped or trapezoid as well.

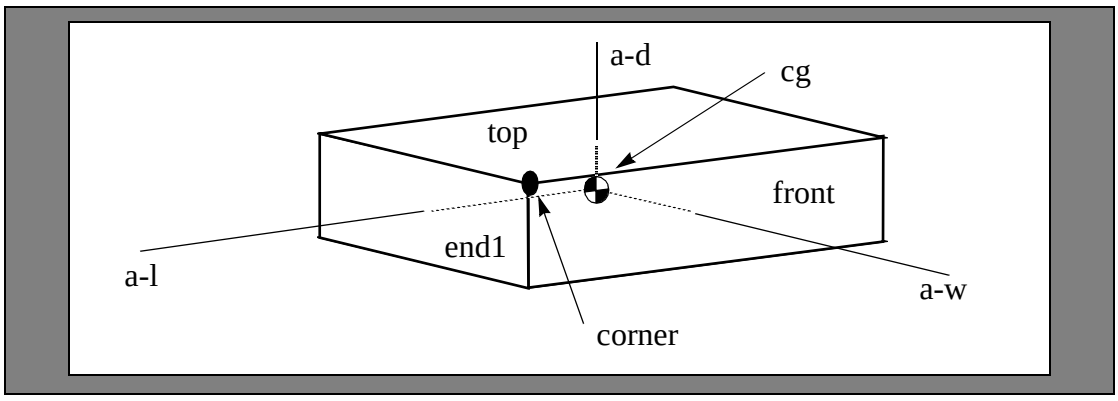


Figure 2.23 Idealized block and some face regions (top, front, and end1) labeled. A corner region is also labeled.

Block Coordinate Axes

The block has three dimensional axes, along-length (a-l), along-width (a-w), and along-depth (a-d).

Block Center of Gravity

The block CG is located at the intersection of mid-points along all parallel face translational axes, and at the mid-point between non-parallel face translational axes. In the right-angled block, the CG is located at (a-l middle a-w middle a-d middle).

Block Regions

A block has six face, twelve edge, and eight corner regions. The face region locations are defined in Table 2.5.

Face Name	Dimension	Value
top	along-depth	far-left
bottom	along-depth	far-right
front	along-width	far-left
back	along-width	far-right
end1	along-length	far-left
end2	along-length	far-right

Table 2.5 Block Regions

Edge regions are defined as boundaries of two faces, such as front and top. Corners are represented as protuberances, which in this case are boundaries of three faces, such as front, end1, and top.

Block Size

The block size restrictions similar to that of a cylinder. All three sizes are approximately the same:

(size along-length) = (size along-width) = (size along-depth) (2-3)

where the equals sign means approximately the same size, and, if anything, the size in length and width is greater than that in depth, but not as much as an order of magnitude.

Block Shape

The block is quadrilateral (rectangular, parallelogram, or trapezoidal) in all dimensions.

Block Representation

The representation for a block is large owing to the number of edges and corners. A portion of the block representation is presented below, with all faces, three edges, and a corner. The faces are represented as shown in Table 2.5. The edges are represented as boundary loca-

tions on faces. For example, at (1) and (3), below, the top-front edge, and the top-end

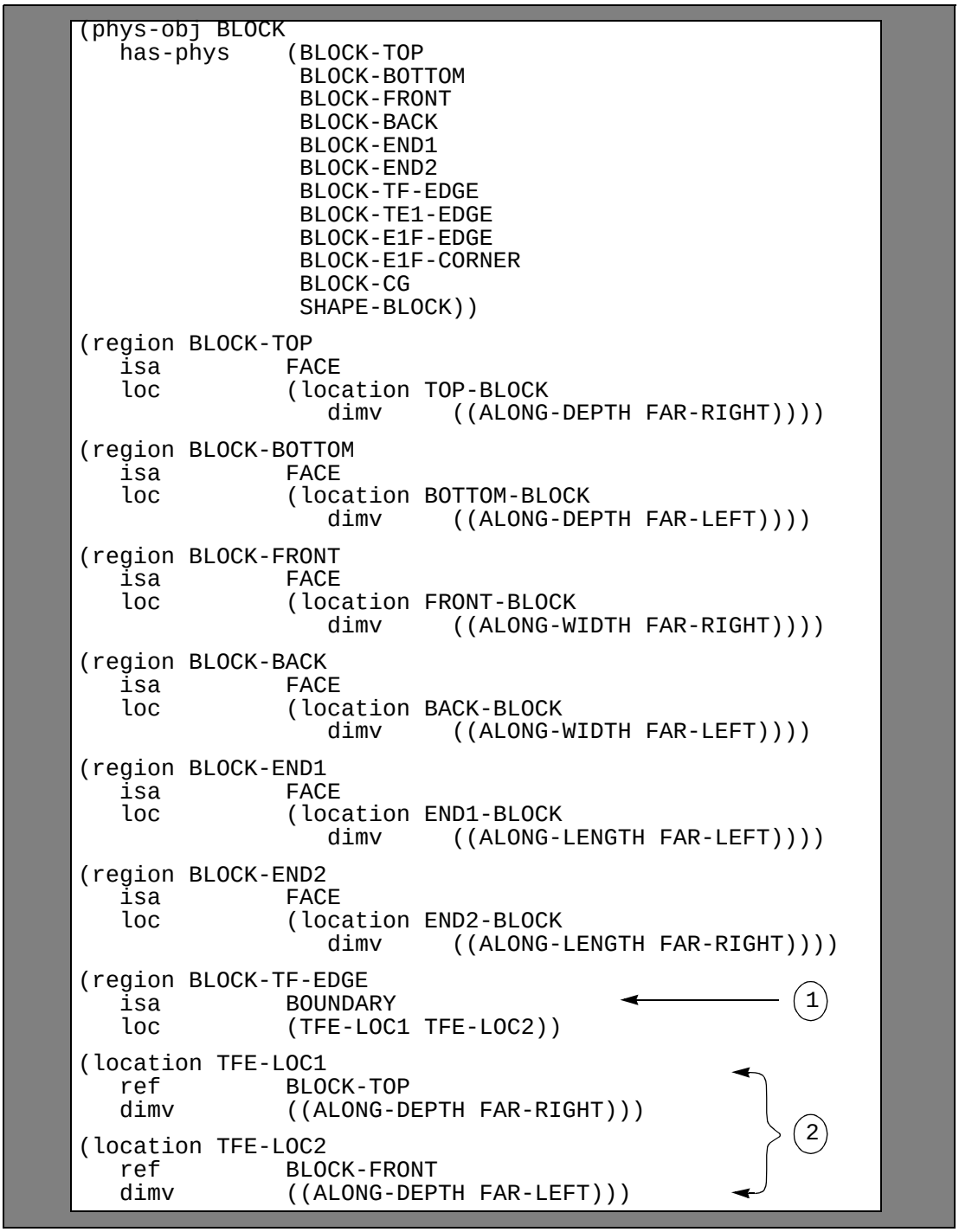


Figure 2.24 Representation for block geometric primitive (plate 1).

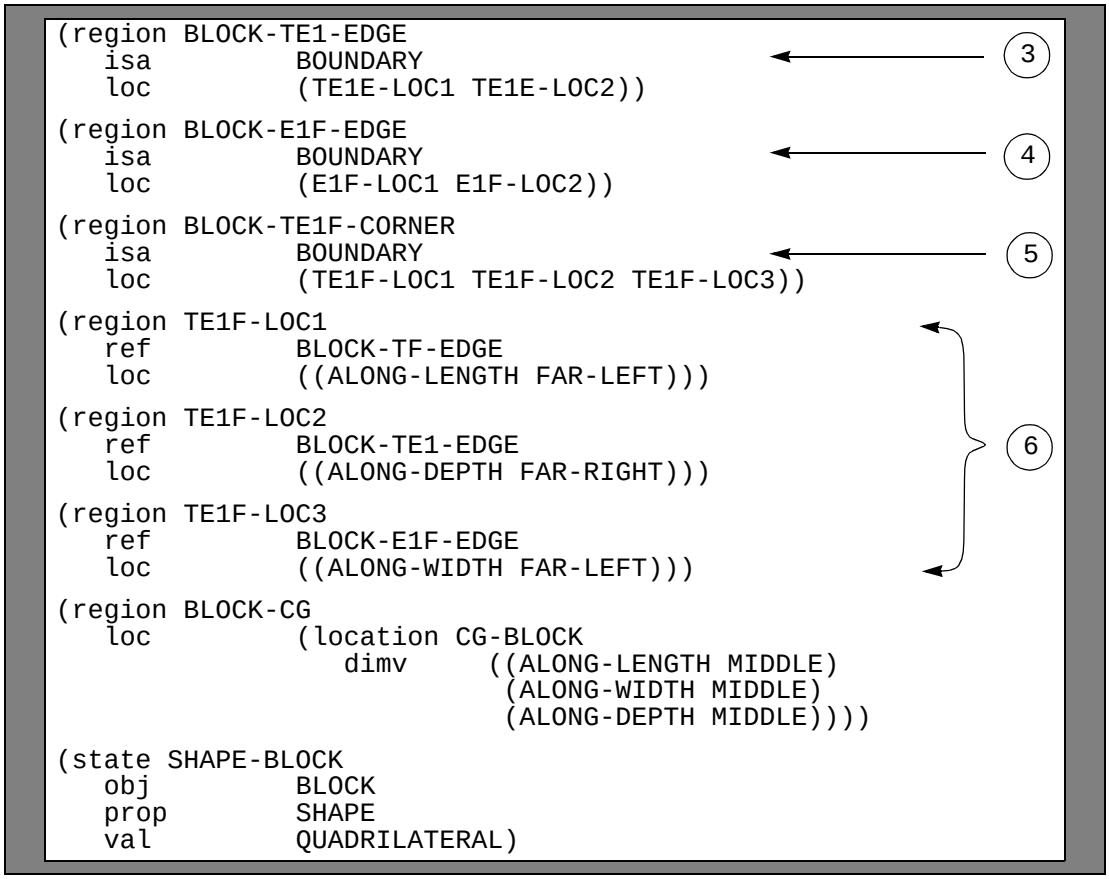


Figure 2.25 Representation for block geometric primitive (plate 2).

edge are represented. At (2), the top-front edge is located with respect to both the top and front faces. The locations of the top-end1 and end1-front edges are similarly represented. The top-end1-front corner (5) is represented as the boundary of the edges just described, and the locations are represented at (6). Other corners are represented in a similar manner.

Only one edge and one corner have been shown in this representation. The other edges and corners are represented in the same fashion. Note that edges require references to each face, and that corners are boundary intersections just as boundaries are surface intersections.

Like the cylinder, the block can be specialized with respect to size and location to describe four other geometric primitives: (1) square plate, (2) column, (3) wedge, and (4) pyramid. The plate and column (and filament) represent objects which have size specializations which are effected at both opposing face regions of the block, while the while the wedge and pyramid represent objects where the size specialization occurs at one face region. The pyramid extrapolates the size change from one dimension to both dimensions at the region. A plate results from having a size in the along-length and along-width dimensions being approximately the equal but much larger than that in the along-depth dimension. The column is the opposite case, where the size in the along-depth dimension is much larger than that in the along-length and along-width dimensions (but they are still roughly equivalent). The filament is simply an extreme case of a column. A wedge, described below, is a block specialization where the depth at one end is much smaller than

that at the other end. This same specialization can be applied to a plate or column to produce a blade.

2.2.5 Wedge

A wedge is a block where the size at one end is zero, similar to the cone for cylinders. An idealized wedge is illustrated in Fig. 2.26.

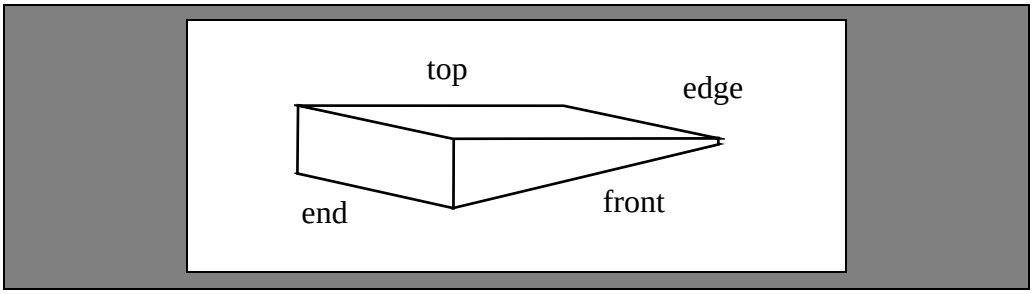


Figure 2.26 Idealized wedge.

Wedge Axes

The dimension axes for a wedge are identical to those for a block.

Wedge Center of Gravity

The wedge CG is located at the mid-point along the width and depth, and 1/3 from the end along the length.

Wedge Regions

The wedge has the same types but different numbers of regions than the block. There are five faces, six corners, and nine edges. Each face has a center. The locations of the wedge regions, with respect to the wedge CG, are shown in Table 2.6.

Name	Dimension	Value
top	along-depth	far-left
bottom	along-depth	far-right
front	along-width	far-left
back	along-width	far-right
edge	along-length	far-left
end	along-length	far-left

Table 2.6 Wedge Regions

Edges and corners are defined the same as for the block. The edge (edge) region is of primary interest, and is represented by the boundary between the top and bottom faces.

Wedge Size

The wedge size is the same as the block, except that the depth on the edge end is zero.

Wedge Shape

The wedge is rectangular when viewed along its length and along its depth, and triangular when viewed along its width.

Wedge Representation

The wedge representation is illustrated with the regions instantiated (1) for the block The

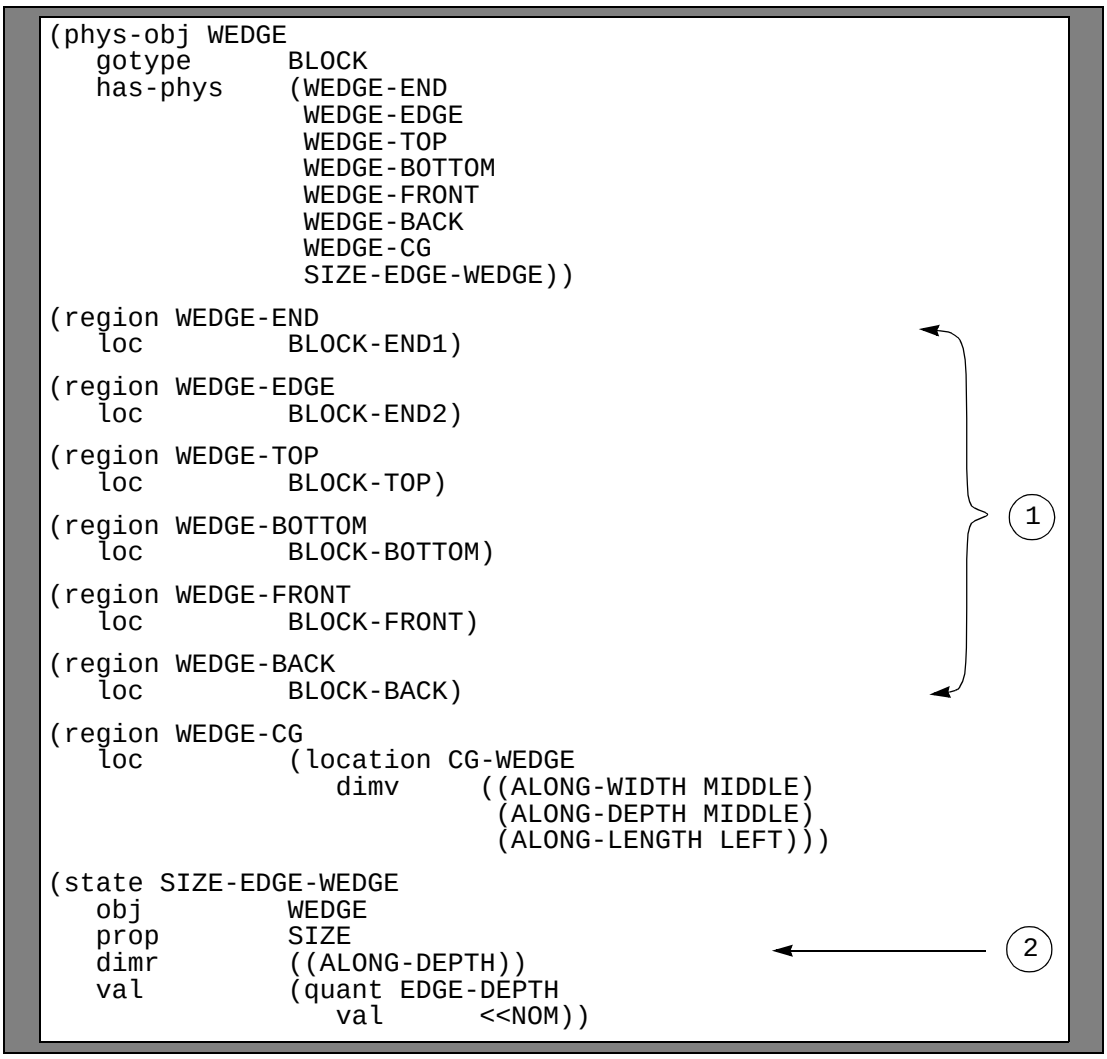


Figure 2.27 Representation of wedge geometric primitive.

size is represented, with respect to the size in along-depth, as <<NOM.

2.3 Object Primitives

In FONM, devices are described as compositions of object primitives rather than geometric primitives. Object primitives represent objects at a level where they are commonly recog-

nized, so most are spatially more complex than a geometric primitive, and all are associated with a device function. An object primitive may be described as a composition of many geometric primitives, or as a single geometric primitive (or object primitive) and a number of specialized regions. Although objects presented in the following sections will be illustrated in both ways, the intent is to show the complexity of a geometric approach over a region approach. Finite element models use a geometric approach. FONM objects are all represented with a region approach.

2.3.1 Object Classification

There are eleven primitive objects, one for each for the objects which can instantiate a machine primitive. Idealized versions of these objects are illustrated in Fig. 2.28

2.3.2 Linkage-Object

A linkage-object is any object which can instantiate a linkage machine primitive. A linkage-object is any of the simple geometric objects described. For the purposes of this dissertation, a linkage object will be illustrated with a column (i.e., a thin rectangle, shown in Fig. 2.29).

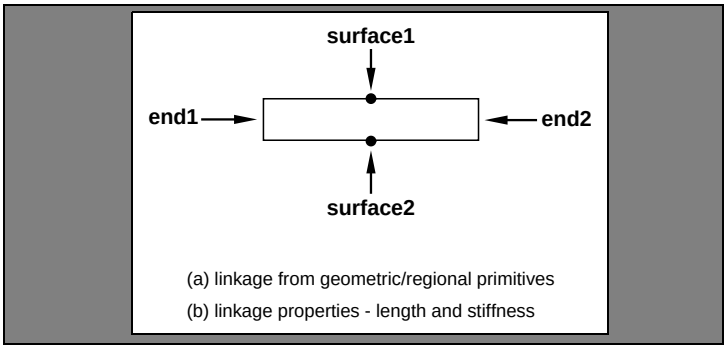


Figure 2.29 An idealized primitive linkage object. The linkage is illustrated with a rectangle (representing a column).

Linkage-objects are described with four regions, two ends and two surface points which represent the entire outer surface along the linkage length, regardless of its shape. These regions are identical to those of an idealized column.

The linkage-object is represented simply as an object with a material stiffness property

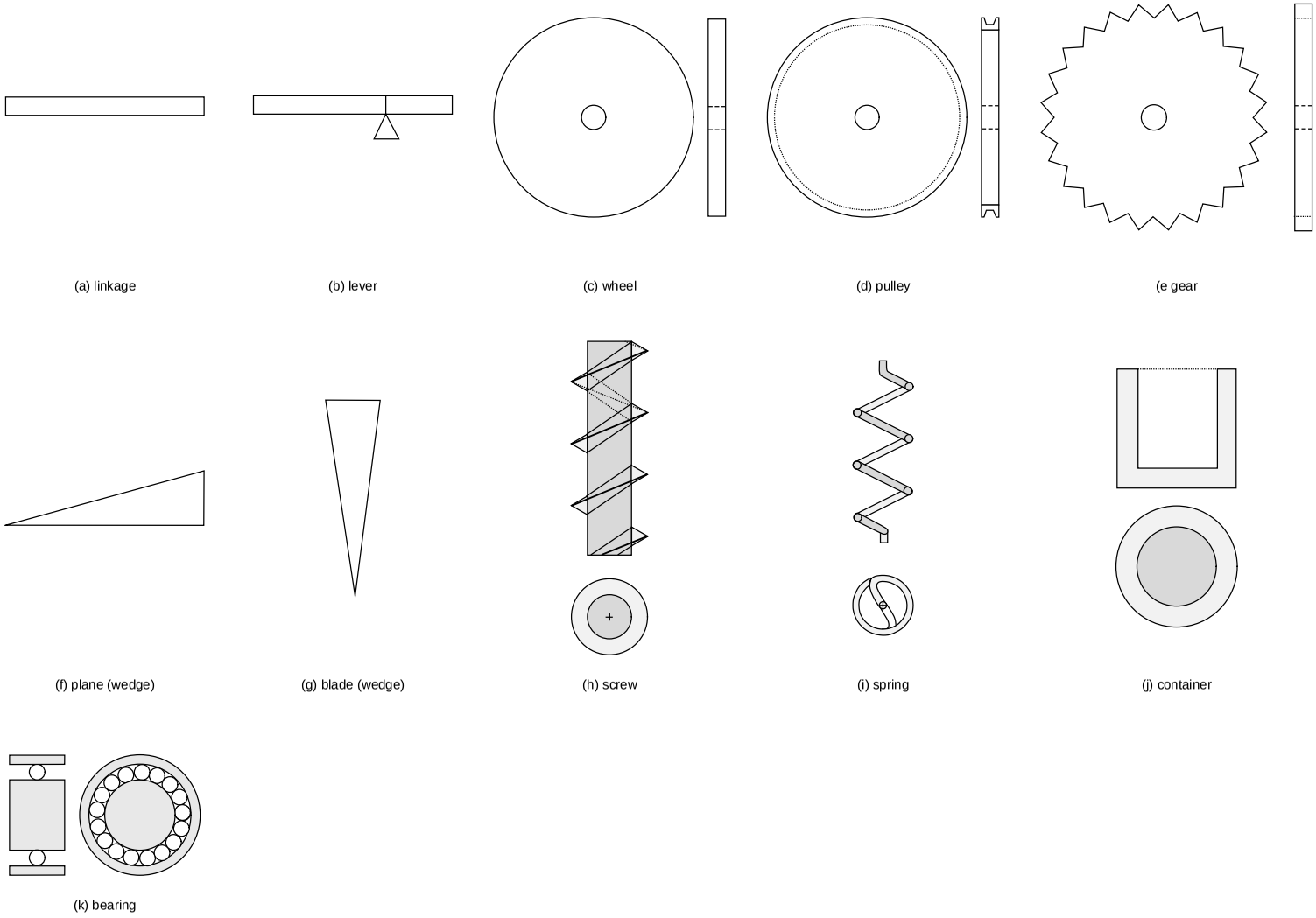


Figure 2.28 Primitive objects in FONM.

value >NOM, where nominal is defined as a value which enables force transmission

```
(phys-obj LINKAGE-OBJECT
   has-phys    (MS-LINKAGE-STIFFNESS
                LINKAGE-END1
                LINKAGE-END2))

(state MS-LINKAGE-STIFFNESS
   obj         LINKAGE-OBJECT
   prop        MATERIAL-STIFFNESS
   val         (quant ELASTIC-MS
                  units    PPI
                  val      >NOM))

(region LINKAGE-END1
   loc         (location LOC1-LINKAGE
                  dimv     ((?DIM      FAR-LEFT))))

(region LINKAGE-END2
   loc         (location LOC2-LINKAGE
                  dimv     ((?DIM      FAR-RIGHT))))
```

Figure 2.30 Representation of the linkage object primitive.

The linkage-object end locations are represented with respect to a variable dimension, since there is no geometric requirement that a linkage-object be a column, only that the ends be at extreme values in the dimension chosen.

2.3.3 Lever-Object

A lever-object is instantiated by any linkage object primitive with the addition of a pivot or fulcrum region. An idealized version, which will be used to illustrate devices in this dissertation, is shown in Fig. 2.31:

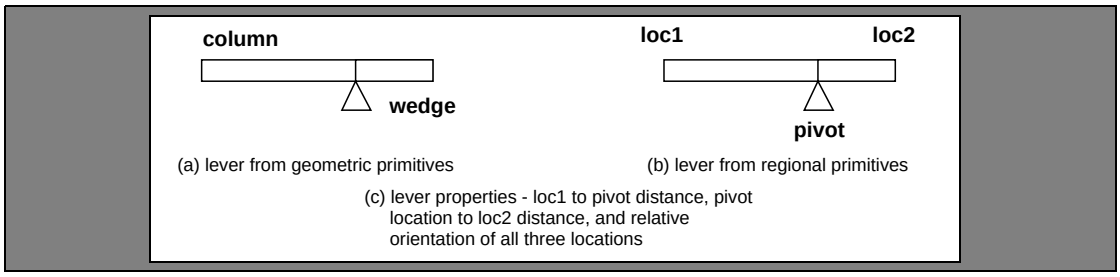


Figure 2.31 Idealized primitive lever objects, (a) based on composition of geometric primitives, and (b) based on a single geometric primitive and regions.

Two versions of the lever-object are shown in Fig. 2.31, one based on the composition of an idealized column and wedge geometric primitives, and one based on a linkage primitive object and three regions. In FONM, the latter description is used for representing devices, because it does not enforce the notion of a particular shape on how the primitive is instantiated.

The lever-object regions are represented as instantiations of the linkage-object regions plus the addition of the pivot region which can be located anywhere along the dimension

defined by the loc1 and loc2 regions (Fig. 2.32).

```
(phys-obj LEVER-OBJECT
   isa          LINKAGE
   has-phys     (LEVER-LOC1
                 LEVER-PIVOT
                 LEVER-LOC2))

(region LEVER-LOC1
   loc          LOC1-LINKAGE)

(region LEVER-PIVOT
   loc          (location LOC-LEVER-PIVOT
                   dimv     ((?dim      (LEFT NEAR-LEFT MIDDLE
                                          NEAR-RIGHT RIGHT)))))

(region LEVER-LOC2
   loc          LOC2-LINKAGE)
```

Figure 2.32 Representation of the lever object primitive.

2.3.4 Wheel-Object

A wheel-object is an object which can instantiate a wheel-axle machine primitive. The difference between a primitive object described with geometric objects and one described with a single geometric object and regions becomes more apparent with the introduction of the wheel. A wheel is an object which has a circular cross section, as depicted in Fig. 2.33.

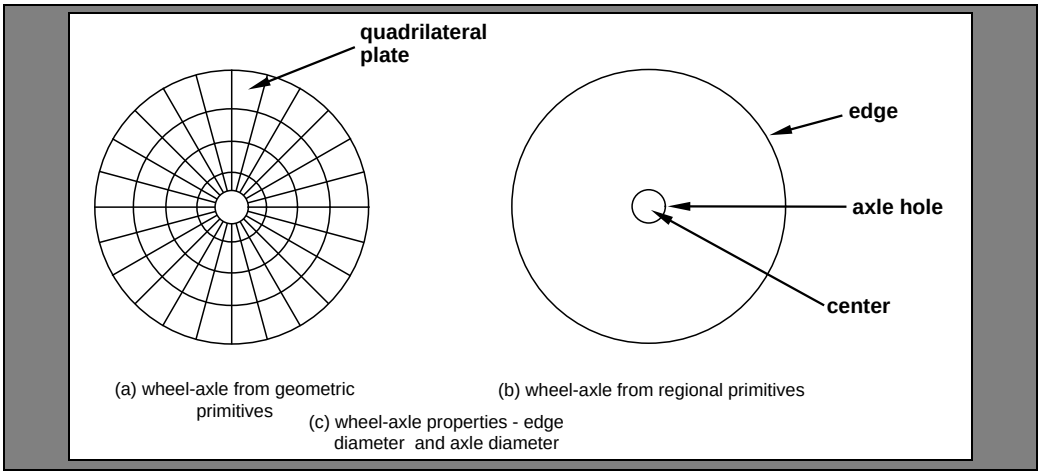


Figure 2.33 Idealized wheel primitive objects, (a) based on composition of geometric primitives, and (b) based on a single geometric primitive and regions.

The wheels shown in Fig. 2.33a and b differ in how many components are required to represent them, and in what types of inference they will support. The wheel in Fig. 2.33a is comprised of many quadrilateral plates connected together. This is identical to a simple finite element mesh. By representing the wheel in this fashion, the entire object is discretized and simulations can be performed on the position of every corner point of every plate. The wheel in Fig. 2.33b is described as a single geometric object, a cylinder or cylindrical plate, and four regions: two edge positions, a center, and an axle hole. These regions are identical to those of an idealized cylinder with the addition of the hole. This latter definition cannot

be used to perform detailed simulations, because it represents the edge as a continuous surface, but it can be used to describe global behavior.

Just as important as the number of objects required to describe the wheel primitive is the ability of the wheel-object to inherit and share characteristics with the lever-object. Since the lever-object has two outer edge regions and a pivot region, these can be associated with the center and edge locations on the wheel-object. The same cannot be done between the versions based on geometric composition. The inheritance of these regions is represented below (1).

The wheel-object hole is also represented below. The size along-radius is represented as <NOM, meaning that the hole must be smaller than the size of the wheel. The size along-depth is represented as NOM, since the hole penetrates the entire wheel.

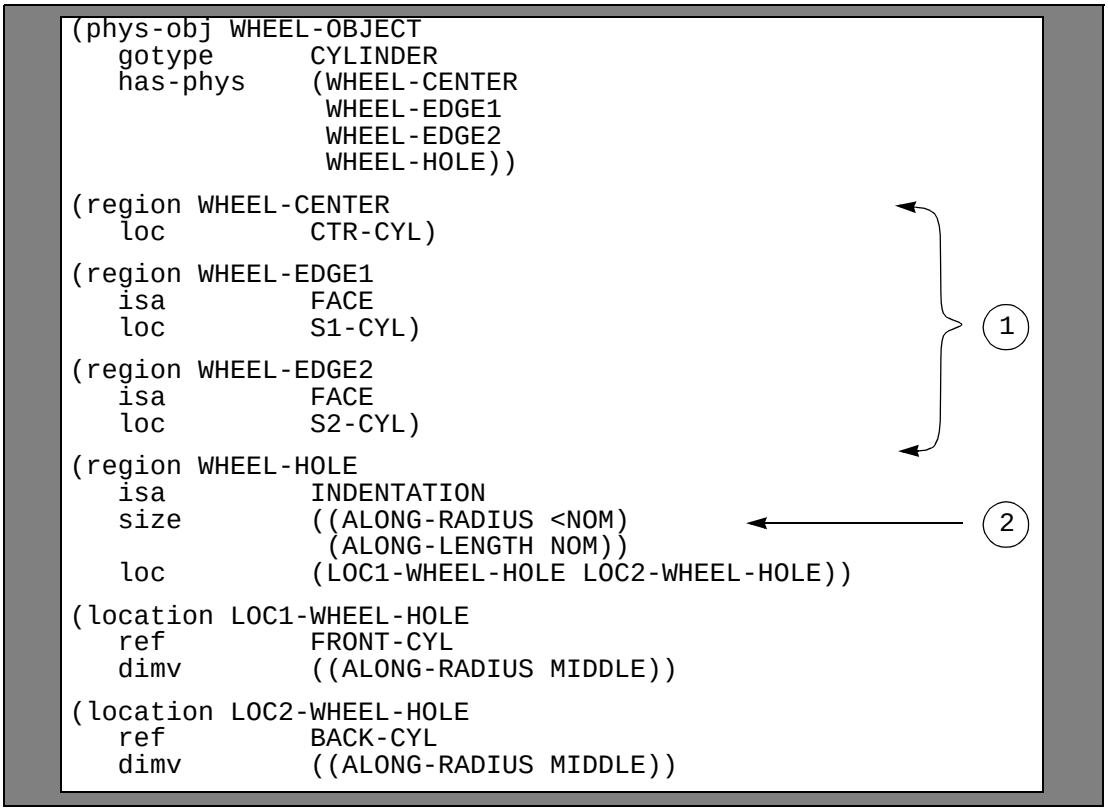


Figure 2.34 Representation of the wheel object primitive.

2.3.5 Gear-Object

A gear-object is an object which can instantiate a gear machine primitive. A gear-object is comprised of two gears, or a gear and a rack, in tooth contact. The gear primitive object is modified from the wheel-object by adding teeth to the outer edge, as shown with the ideal-

ized versions in Fig. 2.35

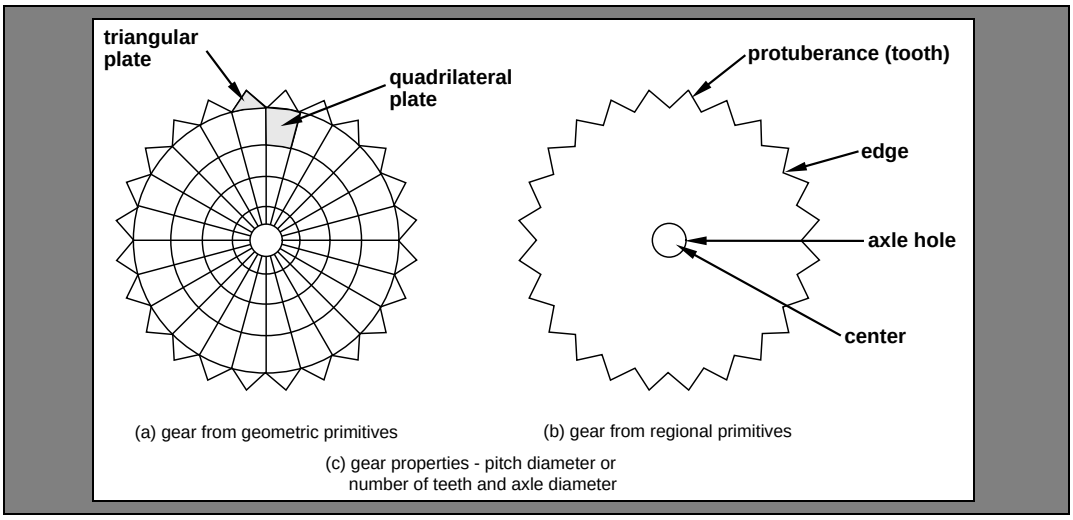


Figure 2.35 Idealized gear primitive objects, (a) based on composition of geometric primitives, and (b) based on a single geometric primitive and regions.

In Fig. 2.35a, the gear tooth is represented with the addition of many triangular plates along the outer edge. In Fig. 2.35b, the wheel-object edge region is specialized as a protuberance. The figure is a bit erroneous, since the gear-object tooth regions are really represented with two regions as with the wheel-object. The figure is left as is to illustrate that the tooth protuberances can be anywhere along the edge without affecting the representation. The gear properties described in Fig. 2.35c: the gear-object's pitch diameter or number of teeth, and the axle diameter, are the only property values needed to make predictions about a gear's behavior. The pitch diameter can be obtained from the two tooth regions without the need to have explicit information about the number of teeth on the gear-object.

The representation for the gear-object is shown below, and represents the tooth regions

```
(phys-obj GEAR-OBJECT
   isa          WHEEL-OBJECT
   has-phys     (GEAR-CENTER
                 GEAR-TOOTH1
                 GEAR-TOOTH2
                 GEAR-HOLE))

(region GEAR-CENTER
   loc          WHEEL-CENTER)

(region GEAR-TOOTH1
   isa          PROTUBERANCE
   loc          WHEEL-EDGE1)

(region GEAR-TOOTH2
   isa          PROTUBERANCE
   loc          WHEEL-EDGE2)

(region GEAR-HOLE
   isa          INDENTATION
   loc          WHEEL-HOLE))
```

Figure 2.36 Representation of the gear object primitive.

as protuberances located at wheel-object edge regions.

2.3.6 Pulley-Object

A pulley-object is an object which can instantiate a pulley machine primitive. A pulley- object is comprised of two components: a pulley and a belt or rope. An idealized pulley is illustrated in Fig. 2.37 below.

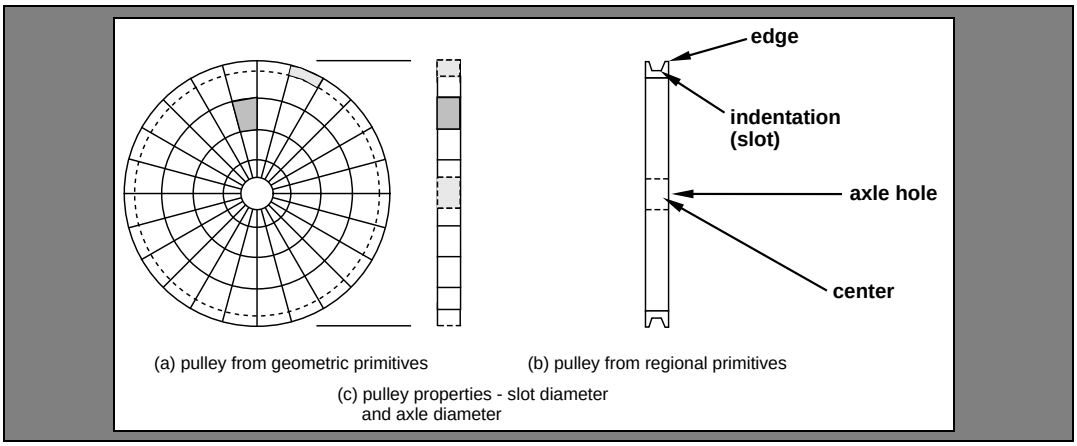


Figure 2.37 Idealized pulley primitive objects, (a) based on composition of geometric primitives, and (b) based on single geometric primitive and regions.

As with the gear-object, Fig. 2.37a, b, and c represent a combined geometric object version, a region based version, and the properties needed for behavioral analysis, respectively. The version in Fig. 2.37a replaces the slot and axle of a real pulley with solid plates that have proportional mass properties. The version in Fig. 2.37b represents the edge and hole as indentation regions. Indentations are associated with behavioral containment, so these regions capture the function of the pulley.

The rope or belt is represented as a one-direction (along-length, away from CG) link-

age. The belt can transmit mechanical force in both directions along its length.

```
(phys-obj PULLEY-OBJECT
   isa         WHEEL-OBJECT
   has-phys    (PULLEY-CENTER
                PULLEY-SLOT1
                PULLEY-SLOT2
                PULLEY-HOLE))

(region PULLEY-CENTER
   loc         WHEEL-CENTER)

(region PULLEY-SLOT1
   isa         INDENTATION
   loc         WHEEL-EDGE1)

(region PULLEY-SLOT2
   isa         INDENTATION
   loc         WHEEL-EDGE2)

(region PULLEY-HOLE
   isa         INDENTATION
   loc         WHEEL-HOLE)
```

Figure 2.38 Representation of the pulley object primitive.

The pulley-object representation in Fig. 2.38 shows the slot regions being represented as indentations at the wheel edge locations, analogous to the gear-object.

2.3.7 Bearing-Object

A bearing-object is an object which can instantiate a bearing machine primitive. A bearing-object can be instantiated by an object with a circular cross section. An idealized bearing-object is illustrated in Fig. 2.39.

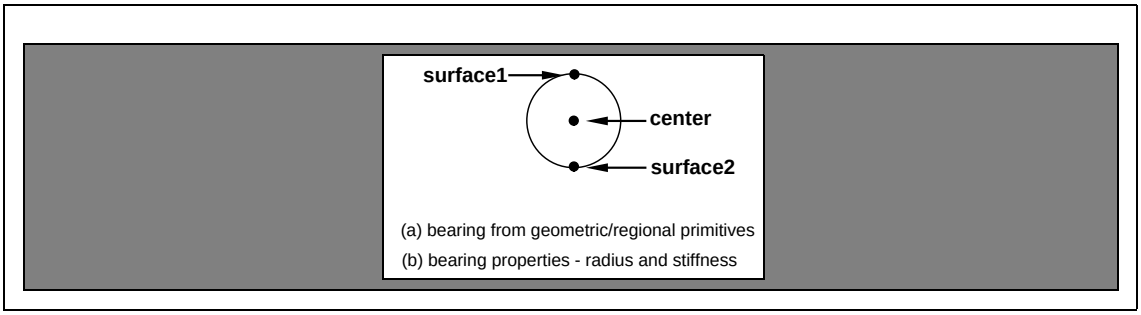


Figure 2.39 Idealized bearing primitive object. A bearing is a cylinder, a cone, or a sphere geometric primitive.

Like the wheel-object, the object in Fig. 2.39 is represented with the cylinder and sphere regions. Unlike the wheel-object, the bearing-object does not have an axle hole region. There is no access to the center of the object, so force and motion are transmitted on the surface alone.

The representation for the bearing object allows for a conical shape as well as a spher-

```
(phys-obj BEARING-OBJECT
   gotype      (SPHERE CYLINDER CONE)
   isa         WHEEL-OBJECT
   has-phys    (BEARING-CENTER
                BEARING-S1
                BEARING-S2))

(region BEARING-CENTER
   loc         WHEEL-CENTER)

(region BEARING-S1
   loc         WHEEL-EDGE1)

(region BEARING-S2
   loc         WHEEL-EDGE2)
```

Figure 2.40 Representation of the bearing object primitive.

ical or cylindrical one, since the bearing functions by transmitting force passively through its surface regions rather than actively as with the other wheel-like objects.

2.3.8 Plane-Object

A plane-object is an object which can instantiate the plane machine primitive. An idealized version of the plane-object, for use in illustrating device representations in FONM, is shown in Fig. 2.41.

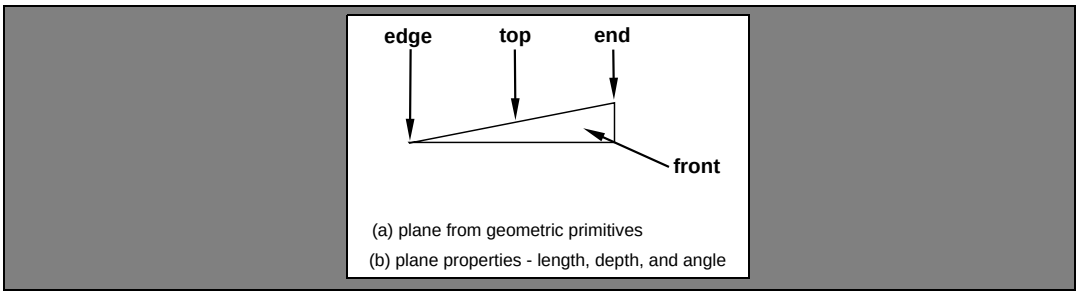


Figure 2.41 Idealized plane primitive object. A plane is a wedge geometric primitive.

The idealized plane-object is represented with a wedge geometric primitive. In general, a plane-object can be instantiated by an object that is offset with respect to the path of motion, so the triangular shape is an implied, rather than an explicit geometric, requirement of the object. The plane-object is represented with two face regions and an edge region. The plane of motion is the top (face) region, and the end (face) and edge (edge) regions deter-

mine the offset.

```
(phys-obj PLANE-OBJECT
   gotype      WEDGE
   has-phys    (PLANE-END
                PLANE-EDGE
                PLANE-TOP))

(region PLANE-END
   loc         WEDGE-END)

(region PLANE-EDGE
   loc         WEDGE-EDGE)

(region PLANE-TOP
   loc         WEDGE-TOP)
```

Figure 2.42 Representation of the plane object primitive.

2.3.9 Blade Object

A blade-object is an object which can instantiate the blade machine primitive. A blade- object, unlike a plane-object, has an explicit triangular shape requirement, as illustrated by the idealized blade in Fig. 2.43.

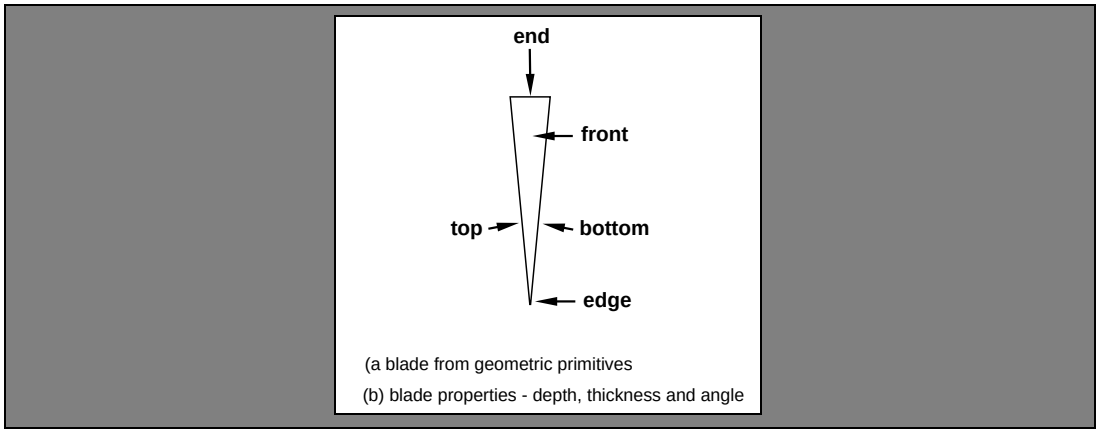


Figure 2.43 Idealized blade primitive object. The blade is a wedge geometric primitive.

The blade-object is represented with a wedge geometric primitive, so their regions are

identical.

```
(phys-obj BLADE-OBJECT
   gotype      WEDGE
   has-phys    (BLADE-END
                BLADE-EDGE
                BLADE-TOP
                BLADE-BOTTOM
                BLADE-FRONT
                BLADE-BACK))

(region BLADE-END
   loc         WEDGE-END)

(region BLADE-EDGE
   loc         WEDGE-EDGE)

(region BLADE-TOP
   loc         WEDGE-TOP)

(region BLADE-BOTTOM
   loc         WEDGE-BOTTOM)

(region BLADE-FRONT
   loc         WEDGE-FRONT)

(region BLADE-BACK
   loc         WEDGE-BACK)
```

Figure 2.44 Representation of the blade object primitive.

2.3.10 Screw-Object

A screw-object is an object which can instantiate a screw machine primitive. The screw object can be visualized as a cylinder or cone geometric primitive with a triangular column wrapped around its edge, as depicted in Fig. 2.45.

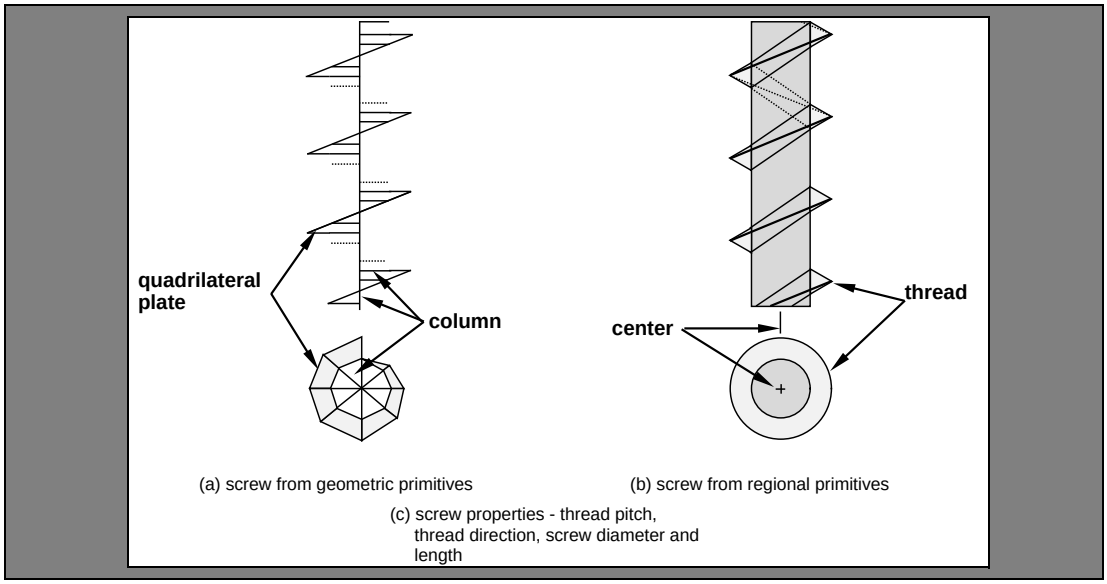


Figure 2.45 Idealized screw primitive object, (a) based on composition of geometric primitives, and (b) based on single geometric primitive and regions.

Two versions are shown. Fig. 2.45a represents a screw as a helical series of columns cantilevered off a central column, with quadrilateral plates connected to the columns. This is a finite element representation of a screw. Fig. 2.45b represents the screw as a cylinder whose edge is a protuberance, like a gear, formed from a wedge wrapped around it, as shown in Fig. 2.46a. Another way of thinking about the screw-object representation is as a cylinder with a wedge-shaped column wrapped around it as shown in Fig. 2.46b. The first figure illustrates why a helical shape need not be described, while the latter figure illustrates how a thread can be represented so as to account for cutting without changing the geometric primitives used to describe the object

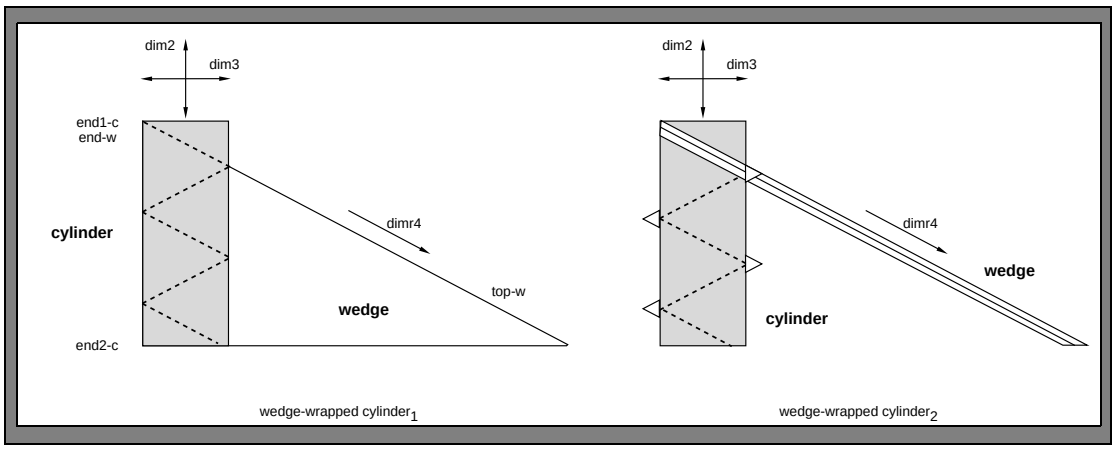


Figure 2.46 Concept of a screw as a combination of a cylinder and a wedge.

The screw-object is represented with a cylinder (or cone) which inherits properties from a plane-object (at 1 in Fig. 2.47). The screw-object regions are the two cylinder ends, a center, and two surface regions.

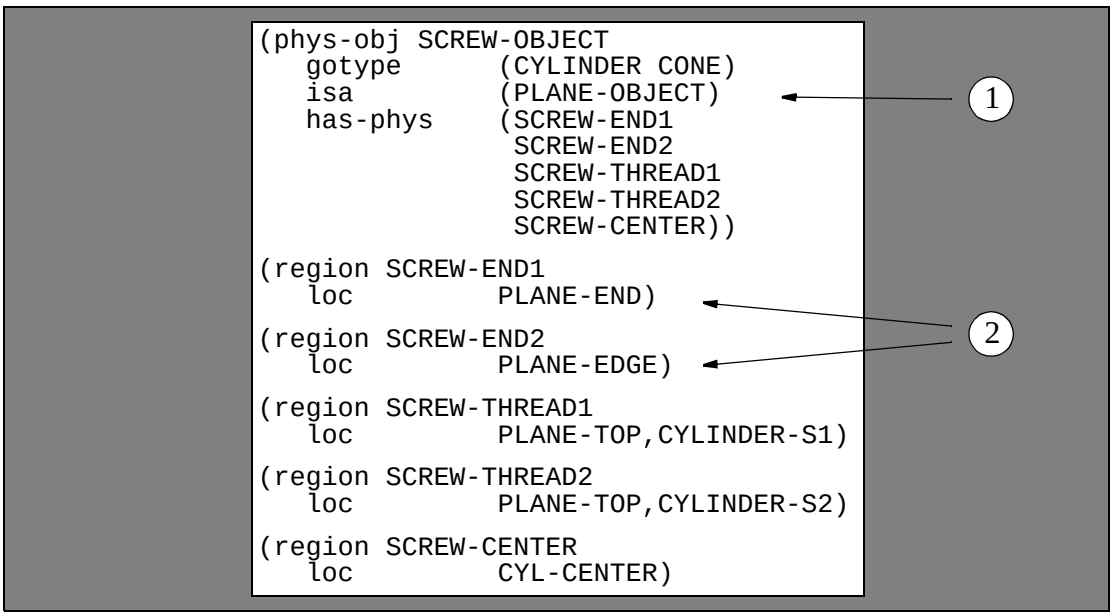


Figure 2.47 Slot-filler representation of the screw object.

The ends are related to the end and edge of the plane-object (2), respectively. The threads are related to the plane-object top region and the cylinder s1 and s2 regions.

2.3.11 Spring-Object

A spring-object is an object which can instantiate a spring machine primitive. A spring- object can be instantiated by any elastic object, so any linkage-object can instantiate a spring-object. An idealized spring-object is depicted as a coiled filament geometric object in Fig. 2.48.

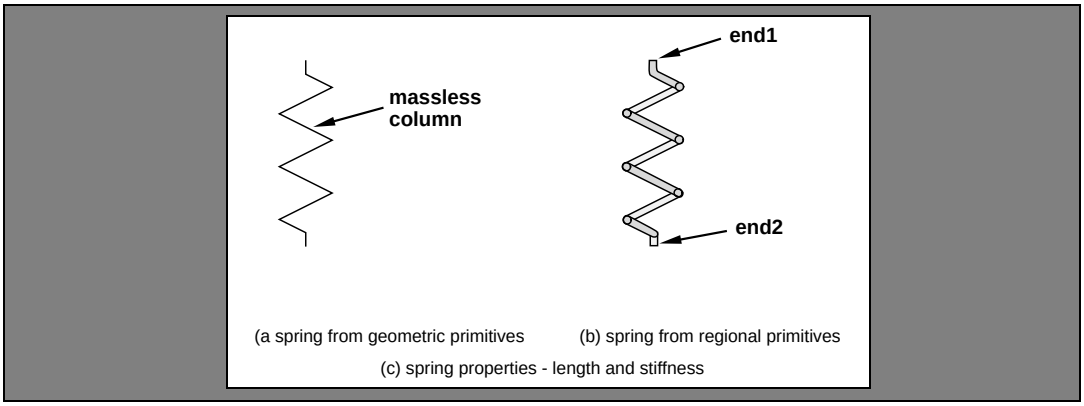


Figure 2.48 Idealized spring primitive objects, (a) based on composition of geometric primitives, and (b) based on single geometric primitive and regions.

An idealized spring, based on a finite element version, is shown in Fig. 2.48a. In this version, the spring is represented with a massless column which has a material stiffness. The version in Fig. 2.48b is represented as a filament wrapped around a cylindrical shape, with the two cylinder end regions.

```
(phys-obj SPRING-OBJECT
   isa          LINKAGE-OBJECT
   has-phys     (SPRING-END1
                 SPRING-END2))

(region SPRING-END1
   loc          LINKAGE-END1)

(region SPRING-END2
   loc          LINKAGE-END2)
```

Figure 2.49 Representation of the spring object primitive.

2.3.12 Container-Object

A container-object is an object which can instantiate a container machine primitive. A container-object is instantiated by an object with an indentation, as depicted by the idealized

version in Fig. 2.50.

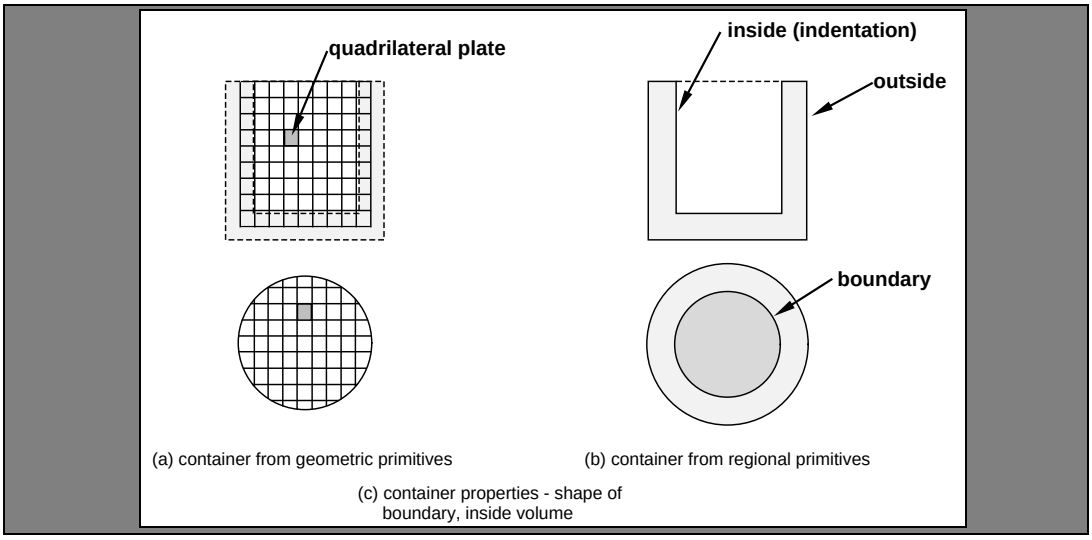


Figure 2.50 Idealized primitive container objects, (a) based on composition of geometric primitives, and (b) based on single geometric primitive and regions.

Two container-objects are depicted in Fig. 2.50a and b. Fig. 2.50a is a container comprised of a triangular and quadrilateral plates connected together in a tube. This tube is connected to another set of triangular and quadrilateral plates which are themselves connected together to form a circular plate. The two generalized objects form a vessel. Fig. 2.50b represents the container-object as a cylinder geometric object with an indentation region along its length and coaxial with its center region. The surface regions are called the container-object *outside*, and the indentation forms the container-object *inside*. The boundary between the two is called the container-object *boundary*

```
(phys-obj CONTAINER-OBJECT
   isa          LINKAGE-OBJECT
   has-phys     (CONTAINER-INSIDE
                 CONTAINER-OUTSIDE
                 CONTAINER-BOUNDARY))

(region CONTAINER-INSIDE
   isa          INDENTATION)

(region CONTAINER-OUTSIDE
   loc          LINKAGE-END1)

(region CONTAINER-BOUNDARY
   loc          LINKAGE-END2)
```

Figure 2.51 Representation of the container object primitive.

2.4 Devices

In FONM, a device's components are each represented with one or more object primitives. A device may also have additional region specifiers at the region locations of its object primitive components. Consider the nutcracker depicted in Fig. 2.52. The nutcracker is a device comprised of four components. The handles (LEVER1, LEVER2) are represented

with two lever-objects, the pivot piece (PIVOT) is represented with a linkage-object, and the spring (SPRING) is represented with a spring-object.

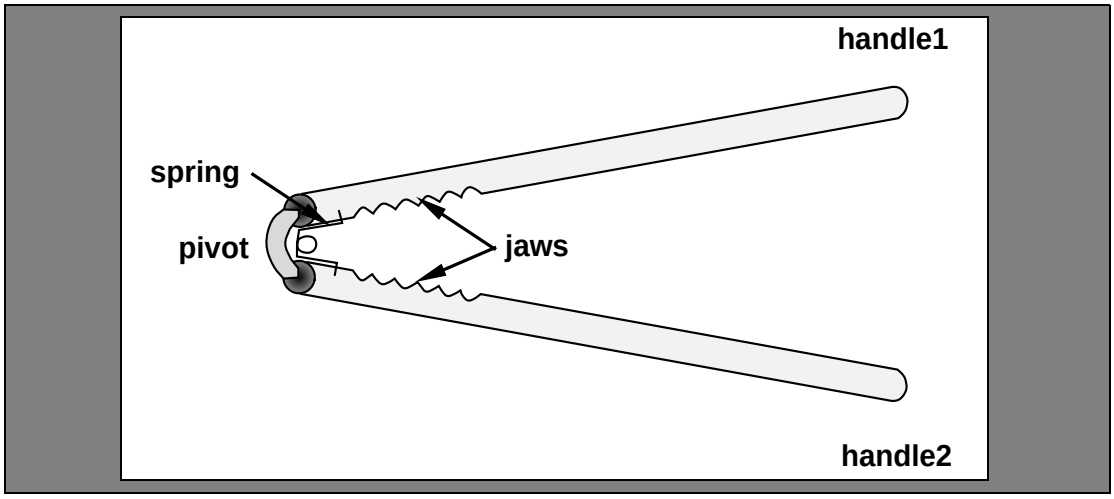


Figure 2.52 Four components in a nutcracker: two levers, a pivot, and a spring.

Each nutcracker component has region specifiers which are associated with, and directly support the component's role in a particular nutcracker function. For example, when used for cracking nuts, the nutcracker is a hand-operated device, so it must have some location (e.g., HANDLE-L1 on LEVER1) to operate the device (i.e., by grasping and pressing). It must also have some location where the force which cracks the nut is produced. Force can be transmitted at contact locations, so the jaw region, JAW-L1 on LEVER1, wherever it is physically located, is a necessary region for nutcracking. LEVER1 is connected to PIVOT, so the location of that connection must also be represented. This location is called END-L1. These three regions instantiate the LOC1, LOC2, and PIVOT lever object regions. By specializing the shape (JAW-L1 is a protuberance) and size (HANDLE-L1 will fit the hand), the lever object regions take on the functions of those region types. The same set of relationships holds for LEVER2 (with HANDLE-L2, JAW-L2, and END-L2), PIVOT (END1-P and END2-P), and SPRING (END1-S and END2-S). The overall nutcracker device is represented with four components and ten regions.

In order to fully describe nutcracker statics, the spatial relationships between its components, and their connectivity, must also be represented.

2.5 Relational Characteristics

A relational characteristic describes a spatial relationship between two objects. Object relational characteristics constrain the interactions in which the objects can participate. For example, if the nutcracker handles illustrated in Fig. 2.52 weren't coplanar, it would be difficult to operate, let alone crack nuts with the device. *Coplanar* is an object orientation. Similarly, in some situations it is important to know how regions of different objects are positioned in space. This is called object *placement*. In FONM, object orientation and placement are the two relational characteristics used to represent spatial interaction.

2.5.1 Orientation

Orientation describes dimensional alignment between two objects and determines how they may interact. For example, a nut and bolt must have their centers aligned in order to thread them together. Objects whose rotational centers are aligned are *coaxial*. Orientation be-

tween objects is represented with a knowledge structure called *orient*. Orient has three roles: obj1, obj2, and dim. The *obj1* and *obj2* roles describe the objects concerned. The *dim* role describes the dimensions shared by obj1 and obj2, or the offset angle between them. When the dim slot has multiple elements, then each element represents a shared dimension. For example, a nut and bolt, when connected, are both colinear and coaxial, because they share longitudinal and rotational axes.

```
(orient COAXIAL-N-B
  obj1    NUT
  obj2    BOLT
  dim     ((ALONG-LENGTH) (ABOUT-LENGTH))
```

Figure 2.53 Coaxial object orientation in FONM.

In this example, the dim role has two elements (ALONG-LENGTH) and (ABOUT-LENGTH), meaning that the nut longitudinal axis is the same as the bolt longitudinal axis, and the same for the rotational axis. The orientations which are represented in FONM are shown in Table 2.7.

Orientation Type	Dimensional Comparison
colinear	shared translational (length) axis
coaxial	shared rotational (center) axis
coplanar	two shared translational axes
coincident	all translational axes shared
parallel	length axis oriented same as global
perpendicular	axis offset is 90 degrees
skewed	axis offset is not 90 degrees

Table 2.7 Object orientations as a function of shared dimensions

Table 2.7 shows orientation between objects as a function of the number of shared dimensional axes. A ramp for loading goods onto a truck is leaned against the ground and against the rear of the truck. The orientation, called an *offset*, is represented by the dimension mismatch and the angle between them. For example, the ramp ALONG-LENGTH dimension makes an angle with respect to the ground.

```
(orient SKEW-LRAMP
  obj1    LOADING-RAMP
  obj2    GROUND
  dim     ((ALONG-LENGTH
            ALONG-LENGTH
            (quant SLRG
               units  degrees
               val    15)))
```

Figure 2.54 Skew object orientation in FONM.

In this example, the dim role has a single element, so there is only one dimension needed to describe the orientation between LOADING-RAMP and GROUND. The dimensions described are the same and the offset between them is represented by the quantity SLRG (Skew-LRamp-Ground, 15 degrees)

2.5.2 Placement

To represent placement, a frame of reference is required. For example, a person would say that a book which is supported by a table is above the table, because we assume earth as a frame of reference. In FONM, the frame of reference must be made explicit, so in the example above, the dimension of comparison is earth radial and the table is closer to earth's center than the book. The term *above* describes the relative placement of the book and table with respect to a reference dimension and location. Placement and location are different. Location is used to locate regions of an object, whereas placement locates unique objects, either with respect to one another or to a global reference frame. Location can be used when two regions of unique objects are in contact, since their positions will be coincident and their spatial orientation won't matter as long as they are in contact. When objects are not in contact, placement must be used, because the orientation of one object with respect to an inertial frame of reference may change while the orientation of the other object may not.

Placement is represented with a knowledge structure called *place*, which has four roles: obj, odimr, ref, and rdimv. The *obj* role describes the region on the object (e.g., a book) being located. The *odimr* role describes the dimension and direction of the object region with respect to the reference object. The *ref* role describes the reference object or region (e.g., a table) from which the placement of the named object will be determined. The *rdimv* role describes the pertinent dimension and qualitative value with respect to the reference object's static description. For example, the placement of the book *above* the table is represented below:

```
(place ABOVE-B-T
  obj     COVER-BOOK
  odimr   (ALONG-DEPTH NEG)
  ref     TOP-TABLE
  rdimv   (ALONG-DEPTH FAR-LEFT)
```

Figure 2.55 Representation of the 'above' object placement in FONM.

In Fig. 2.55, the object being placed is BOOK, and the region COVER-BOOK is the location, on the book, which is being located in the reference frame. The location of COVER-BOOK on BOOK is represented with the same dimension as the placement comparison, ALONG-DEPTH, however, that is not generally the case. The reference object is TABLE, and the region is TOP-TABLE. The reference location is described as ((ALONG-DEPTH FAR-LEFT)). The meaning of ABOVE-B-T is that the book cover's relationship to the middle of the table top is along the book's depth dimension and toward the table top. The middle is assumed because no other dimensions or values are represented in the placement, so the CG locations in those dimensions are used.

In FONM, placement is primarily a function of dimension, in earth coordinates, as

shown in Table 2.8

Placement Type	Reference DImension	Reference Value
ABOVE	vertical (along-radius)	left, near-left, middle, near-right, right, far-right
BELOW	vertical (along-radius)	far-left, left, near-left, middle, near-right, right
LEFT-OF	horizontal (about-radius)	far-left, left, near-left, middle, near-right, right
RIGHT-OF	horizontal (about-radius)	left, near-left, middle, near-right, right, far-right

Table 2.8 Placement in FONM.

In Table 2.8, the reference value defaults to ground level (FAR-LEFT), so an object sitting on the ground is above the ground if the reference value is anything but (FAR-LEFT). When the reference object is not earth, then the object dimension which coincides with earth vertical is used for representing above and below for other objects.

2.6 Object Connectivity

A mechanical device is comprised of components connected together in spatially meaningful ways. In FONM, object connectivity is represented as a mutual restraint state on two objects, where the local region and restraint dimension are identical. Object connectivity is understood by the degrees of freedom which are restrained by the presence of each object. For example, consider two wood blocks which are bolted together along their depth, as shown in Fig. 2.56.

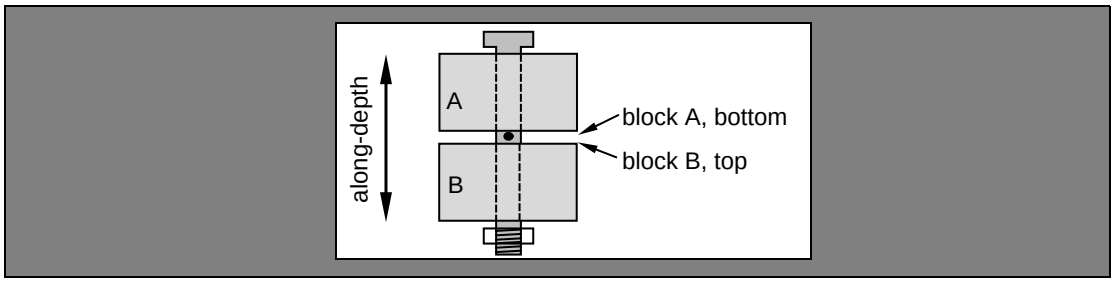


Figure 2.56 Object connection in FONM. Two bolted blocks.

Block A's bottom face is in contact with block B's top face. Both objects are restrained, by the other, or by the bolt (it doesn't matter for this example), in all dimensions. This is called a *clamped*, or *fixed*, connection. In FONM, the representation of connection states is directly a function of the number and type of restrained degrees of freedom, as shown in

Table 2.9.

Connection Name	Degrees of Freedom Restrained
free	None
contact	One direction
pivoted	One local translational direction
fitted	Two translational and one rotational
roller	Two translational and two rotational
pinned	All translational and two rotational
rigid	All local D.O.F.
clamped or fixed	All D.O.F.

Table 2.9 Connections in FONM.

2.7 Device Static Descriptions

A static device description, or representation, is comprised of the primary object for each component, its physical characteristics, and its relational characteristics to the other components. Device static representations fall into three categories: (1) simple devices, (2) compound devices, and (3) complex devices. A *simple* device is one where all the components are rigidly connected together and move as a single unit. A *compound* device is one where at least two components can move relative to one another. A *simple* compound device has two components which can move relative to one another, and a *multiple* compound device is one with many moving components, but each is a simple device. A *complex* device is comprised of systems, each of which can be a multiple compound device. In the following sections, static representations for devices in each of these categories will be presented. For each device, an illustration of what a real device of this type looks like is presented, along with an illustration of how FONM primitive objects are used to represent the device. A *static device diagram* (SDD) for each device is also presented, rather than fleshing out the full representation, and representations are presented where necessary to make a point. Each SDD presents the components and their connectivity, as detailed in the previous sections. Devices which are comprised of similar components or are connected in similar ways should have similar static representations regardless of how they behave and function. Device representations presented in this section will illustrate these similarities, and, in other chapters, the differences will be shown in dynamic representations of the same devices.

2.7.1 Simple Devices

A simple device has no moving components. Simple devices may have complex shapes, so their regional descriptions play an important role in how they behave and how they are viewed by problem solvers. Two simple devices are represented and described in this section: (1) a bottle opener, and (2) a screwdriver. Each device is comprised of regions which take on the characteristics of one or more primitive object. Four additional simple devices: (3) carving knife, (4) hammer, (5) shovel, and (6) spoon are represented in Appendix A.1.

Bottle Opener Statics

A bottle opener is a simple object which has a blunt end and a pointed end. At each end there is a piece which protrudes and enables the generation of leverage. An illustration of the true bottle opener shape is shown in Fig. 2.57.

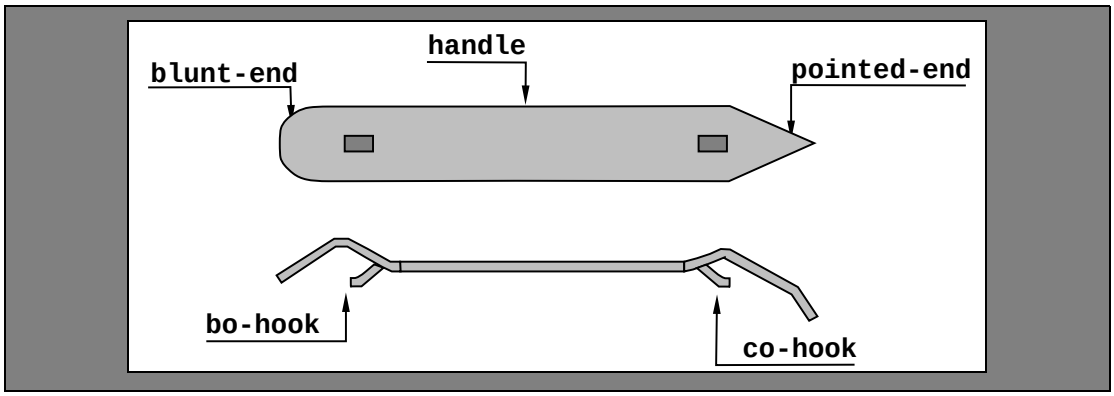


Figure 2.57 Bottle opener illustration.

Notice in the figure that the actual object is made of a single piece and that the shape at both ends is complex. The FONM idealization of the bottle opener is shown in Fig. 2.58 below.

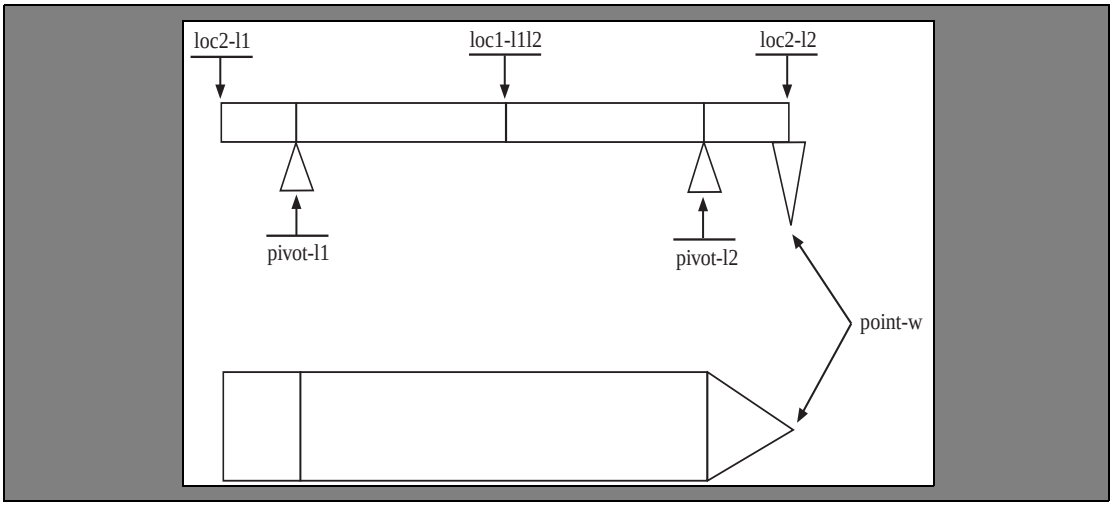


Figure 2.58 Bottle opener object primitives. Two lever-objects and a blade-object.

In Fig. 2.58, the device is shown as two lever-objects connected at their ends (`loc1-l1l2`). The blunt end of the bottle opener is represented with `loc2-l1`, its associated protrusion with `pivot-l1` and its body with `loc1-l1`. A similar relationship holds for the pointed end, accept a blade object is connected at `loc2-l2`. Three FONM objects are used to represent the device, two lever-objects and a blade-object, as shown in the static device

diagram below (Fig. 2.59).

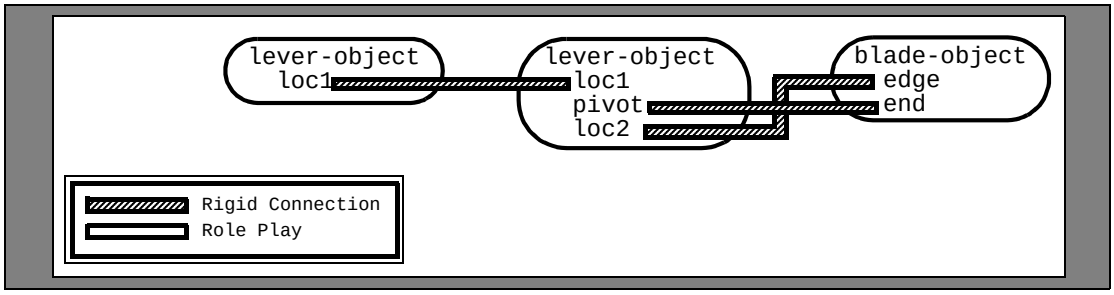


Figure 2.59 Bottle opener static device diagram (SDD).

A static device diagram illustrates the FONM primitive objects which comprise a device's components, and their connectivity as arcs. Both lever-objects share the same region, `loc-l1`. This is represented as a rigid connection between the two components at `loc-l1`, where the rigid connection is depicted as an arc with a diagonal stripe. The `pivot` region of the right-hand lever-object is also the `end` region of the blade-object. These regions are both shared between the two objects and the locations of additional rigid connections between the two objects.

The `pivot` and `loc2` regions for the left-hand lever-object are not shown here, since they are not important in illustrating the component connectivity. In addition, the fact that the protrusions of the bottle opener are represented with protrusions is not illustrated. The colinear spatial relations between these objects are represented below.

```
(phys-obj BOTTLE-OPENER
   isa          (LEVER-OBJECT BLADE-OBJECT)
   has-rel      (COLIN-LO1LO2
                 COLIN-LO2BO1))

(orient COLIN-LO1LO2
  obj1     LEVER-OBJECT1
  obj2     LEVER-OBJECT2
  dim      ((ALONG-LENGTH))

(orient COLIN-LO2BO1
  obj1     LEVER-OBJECT2
  obj2     BLADE-OBJECT1
  dim      ((ALONG-LENGTH))
```

Figure 2.60 Spatial relationships for the bottle opener static representation.

Note that the bottle-opener is labeled as a lever-object and a blade-object. In FONM, when a simple device is represented with an object primitive, it inherits the static capabilities of that object. The most specific physically unique (geometric) object type will be used for inheritance purposes, because it provides access to the largest family of inherited characteristics. Since the blade-object and lever-object come from distinct geometric primitives (any vs. wedge), both are listed.

Screwdriver

A simple straight bladed screwdriver is illustrated in Fig. 2.61.

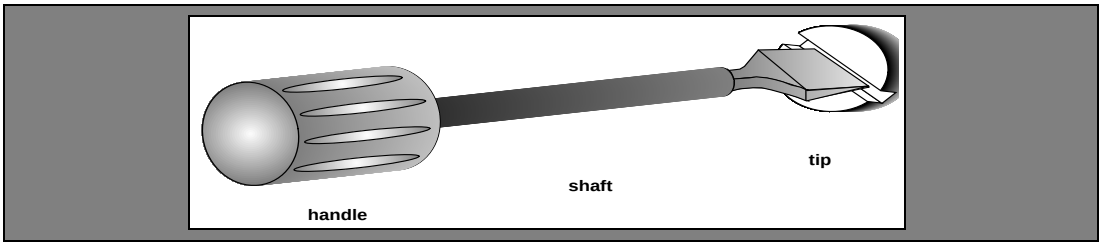


Figure 2.61 Screwdriver illustration.

The screwdriver has three components, a handle, a shaft, and a tip. These components are idealized with a wheel-object, a linkage-object, and a blade-object, as shown in Fig. 2.62.

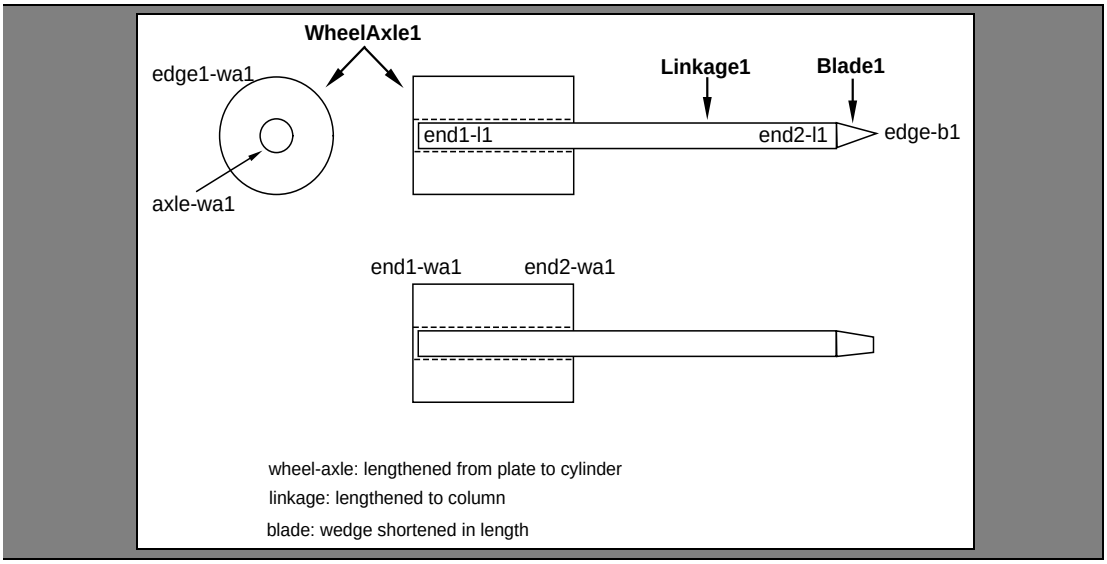


Figure 2.62 Screwdriver object primitives. A wheel-object, a linkage-object, and a blade-object.

The SDD for the screwdriver is shown in Fig. 2.63.

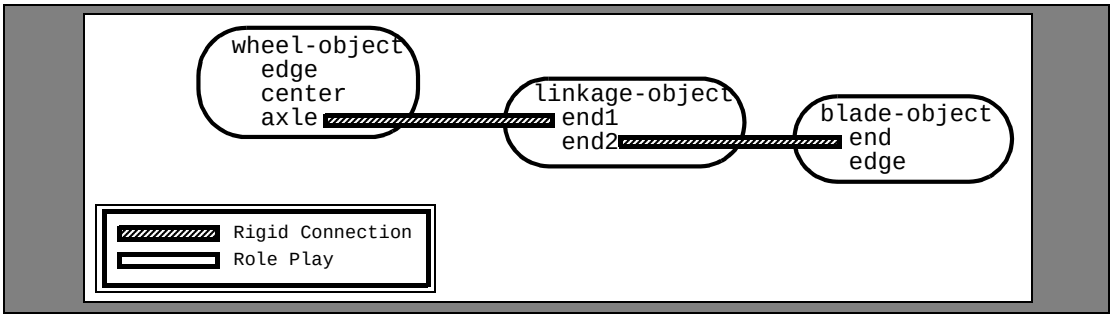


Figure 2.63 Screwdriver static representation diagram.

Like the other devices in this section, the screwdriver components are rigidly connected. Like the bottle opener, the screwdriver components are colinear, and, in the screwdriver, they are also coaxial, as represented (at 1) in Fig. 2.64.

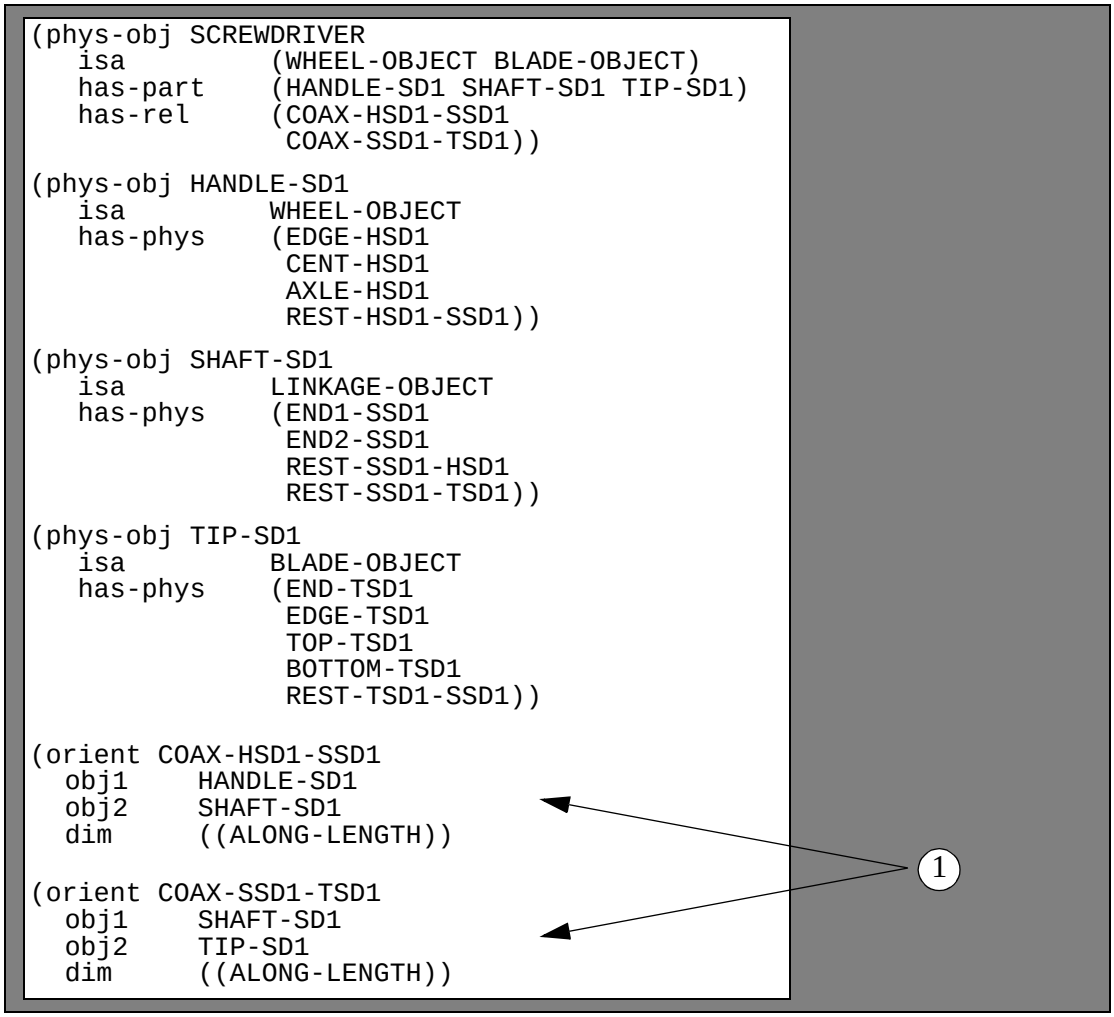


Figure 2.64 Representation of the coaxial component orientation of screwdriver components.

In this figure, the screwdriver components are related through their orientations and through their connections (although the connections are not represented in Fig. 2.64). The component regions have not been represented in this figure, as they simply point to the region equivalents of the wheel-object, the linkage-object, and the blade-object, respectively. Likewise, the restraint states have not been represented, since they are all locally rigid and have been represented elsewhere.

2.7.2 Simple Compound Devices

Simple compound devices have components which can move relative to one another. The complexity of the static description changes due to the inclusion of relational characteristics and bounding states of each component. In the following sections, two simple component devices are represented: (1) a nutcracker, and (2) a clothes pin. Three additional simple compound devices: (3) a pair of scissors, (4), a pair of pliers and a (5) door are represented in Appendix A.2

Nutcracker

The simple nutcracker (initially introduced to show device components, in Fig. 2.52) is pictured again in Fig. 2.65.

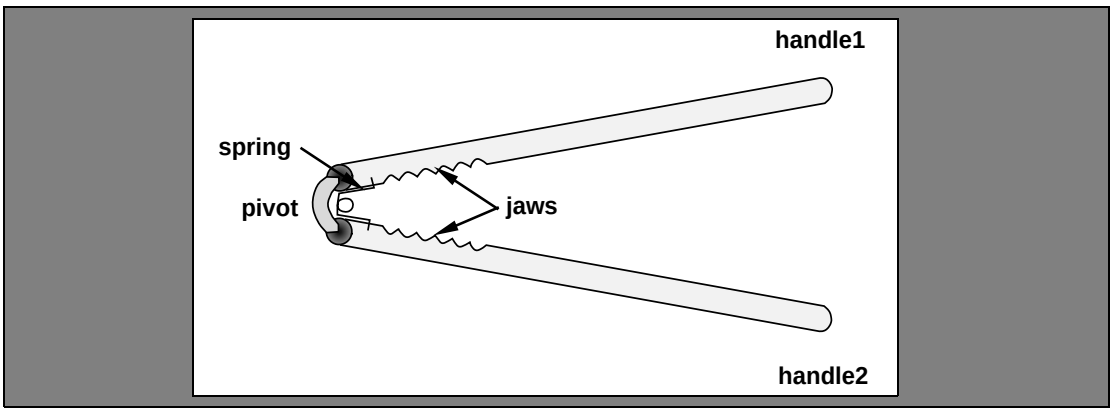


Figure 2.65 Nutcracker illustration.

Each lever component has three areas of interest: a place to hold and manipulate the device, a place to connect the two components, and a place where the force is produced and the object is held. The nutcracker components and the regions used to represent their functional areas are idealized as two lever-objects, a linkage-object, and a spring-object (Fig. 2.66). Fig. 2.66 shows two lever-objects where the location of the pivot and loc2 regions

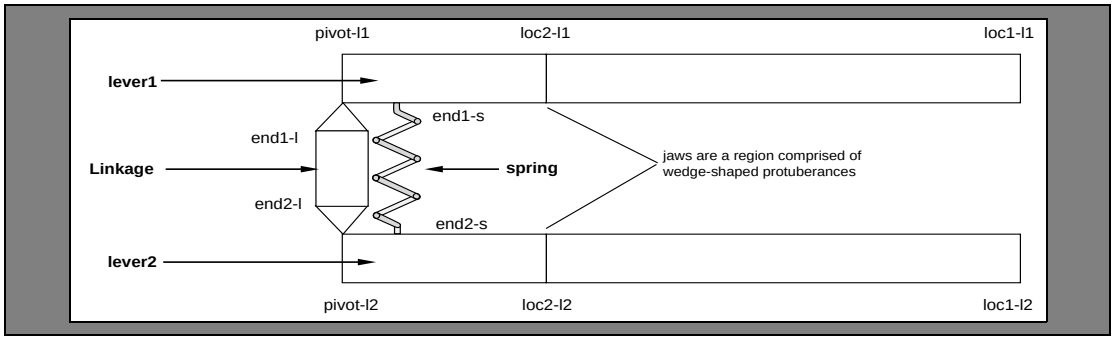


Figure 2.66 Nutcracker object primitives. Two lever-objects, a linkage-object, and a spring-object.

are reversed. The lever-objects are connected to the linkage object at their pivot regions, similar to the scissors and pliers devices. A spring is located at the same place and is in contact with the lever-objects. The jaw regions are idealized as the loc2 region for the lever-

objects, but can be replaced by a more elaborate jaw as shown in Fig. 2.67.

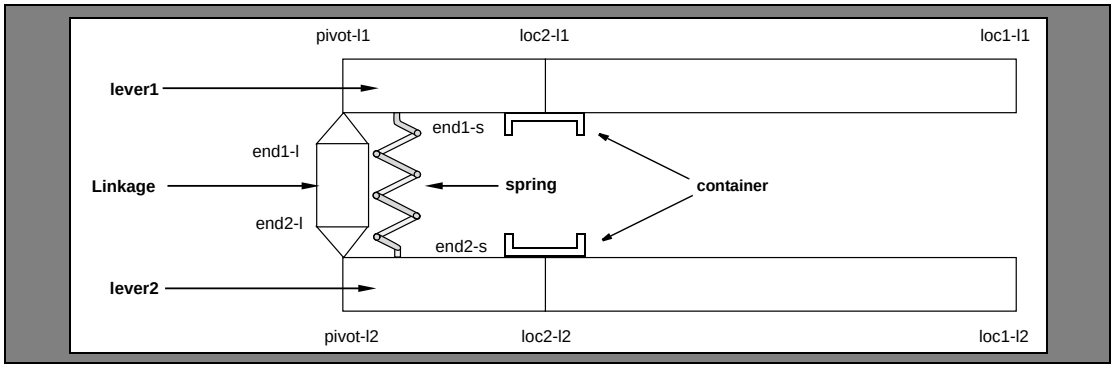


Figure 2.67 Modified idealization of nutcracker with jaw region represented with a container-object.

In this second version, the jaw is represented with a single container-object. A third version, which is shown as an idealized jaw, alone, is depicted in Fig. 2.68. Although these

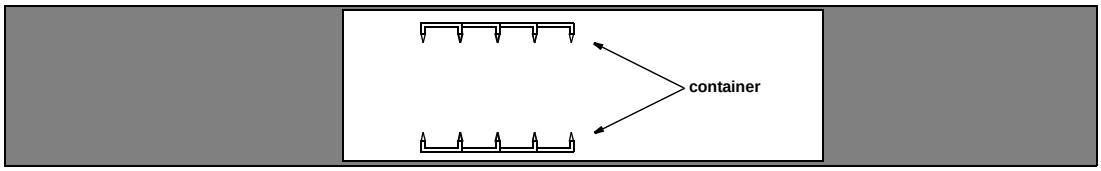


Figure 2.68 Jaw region represented as a series of container-object/blade-object pairs.

elaborations improve the ability to resolve the role of the jaw, the simple version in Fig. 2.66 represents the primary role of the jaw for constraining motion of a nut against the direction of applied force. The container and container/blade versions add to the representation the ability to represent motion constraint in more than one dimension, and the ability to represent holding ability through deformation, respectively, but they also increase the complexity of the representation. The accompanying SDD (Fig. 2.69) illustrates the connectivity of the object primitives in Fig. 2.66.

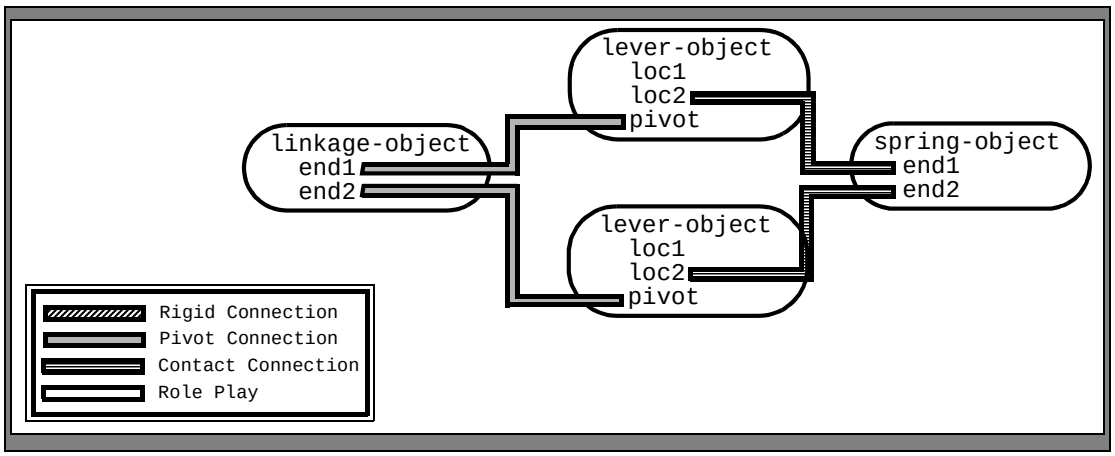


Figure 2.69 Nutcracker static representation diagram.

Two new connection arcs are shown in this figure, for: (1) contact, and (2) pivot. As previously mentioned, connection arcs depict a dynamic relationship between the components which gives rise to the restraint states associated with connectivity. The gray arcs between the linkage-object and the two lever-objects depicts a pivot relationship which allows relative rotational movement of either object. Likewise, the striped arcs between the lever-objects and the spring-object depicts a simple contact between the components.

Clothes Pin

A clothes pin is depicted in Fig. 2.70. The clothes pin is similar to a nutcracker inasmuch as they both have two handles and a spring. When the handles of either device are pressed together, there is resistance provided by the spring. To this extent, then, the static representations of the nutcracker and clothes pin should be similar if not identical. However, the spring is located at a different handle region, and the clothes pin actually works differently, so a dynamic representation should show differences which do not show up in the static representation. Fig. 2.71 shows an idealized version of the clothes pin. The device is repre-

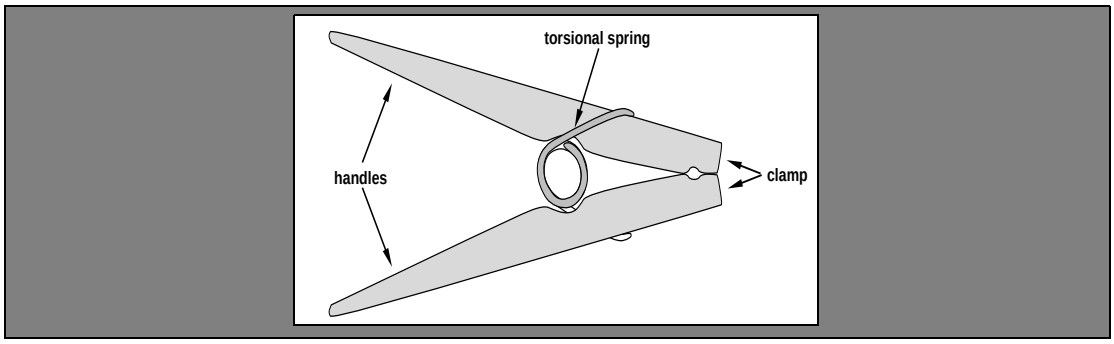


Figure 2.70 Clothespin illustration.

sented with two lever objects and a spring-object as with the nutcracker. The positions of the pivot and jaw regions are swapped: the clothes pin "jaw" or clamp is located where the nutcracker pivot was, and the clothes pin pivot is located where the nutcracker jaw was. An idealized version of the clothes pin is illustrated in Fig. 2.71. Unlike the nutcracker, the

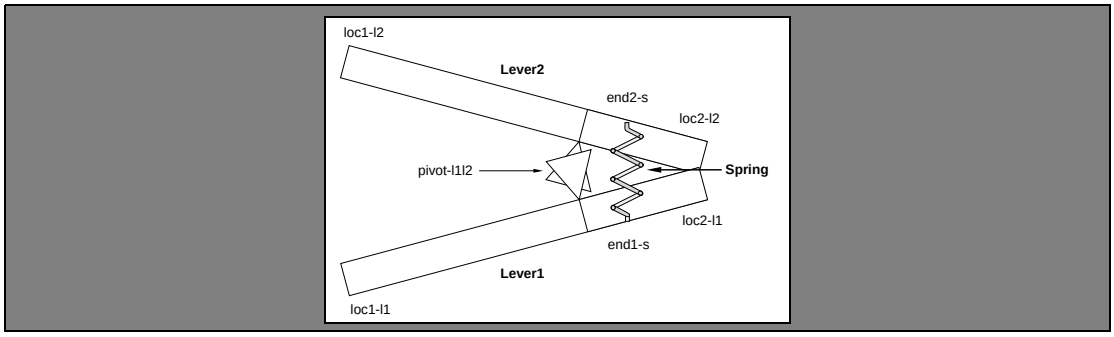


Figure 2.71 Clothes pin object primitives. Two lever-objects and a spring-object.

clothes pin does not have a pivot component. The spring plays the role of the pivot and the

spring, so the SDD illustrated in Fig. 2.72 is commensurately simplified.

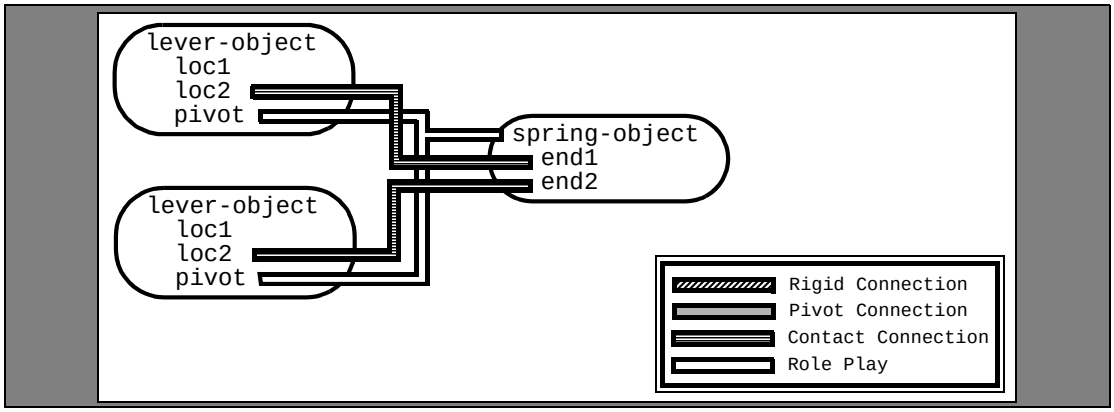


Figure 2.72 Clothes pin static representation diagram.

A new connection arc is presented in this figure. Since the spring actually plays the role of a linkage, a role-play arc is used to represent the associated connectivity, since there is really no connectivity between the spring-object and the lever-objects which isn't captured by the contact arc. The endpoints of the role-play arc are the regions of each component which are identical.

2.7.3 Multiple Compound Devices

A multiple compound device has multiple interacting components and multiple primitive object categories. Not only can components move relative to one another, they can do so in parallel. A component of a multiple component device may instantiate more than one type of device and the representation must capture this possibility. One multiple component device, a fingernail clipper, is represented in this section. Two additional multiple compound devices: (2) a press, and (3) a crank can opener, are represented in Appendix A.3.

Fingernail Clipper

A fingernail clipper is shown in Fig. 2.73. The device has four physically distinct compo-

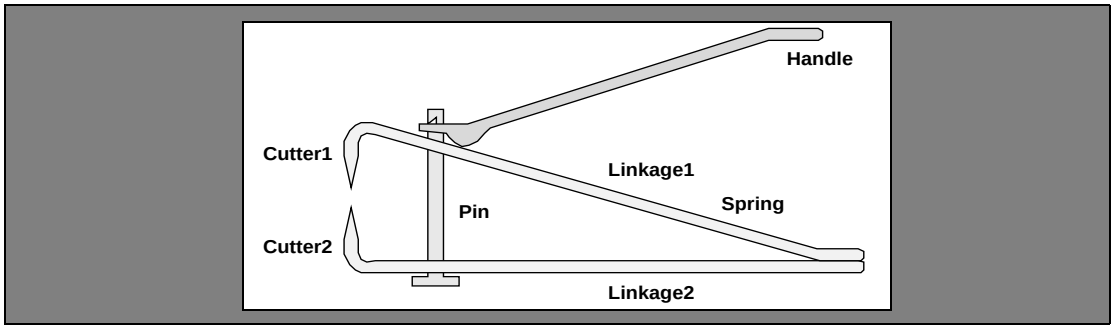


Figure 2.73 Fingernail clipper illustration.

nents: (1) a handle, (2) a pin, and (3)(4) two cutters. In this device, the cutters play the role of blade-objects, of spring-objects, and of linkage-objects. An idealized version of the fingernail clipper is depicted in Fig. 2.74. In this figure, the separate functions of the individual components are fleshed out as unique object primitives, so there are seven idealized

objects used to represent the device rather than four. The fingernail clipper is very similar

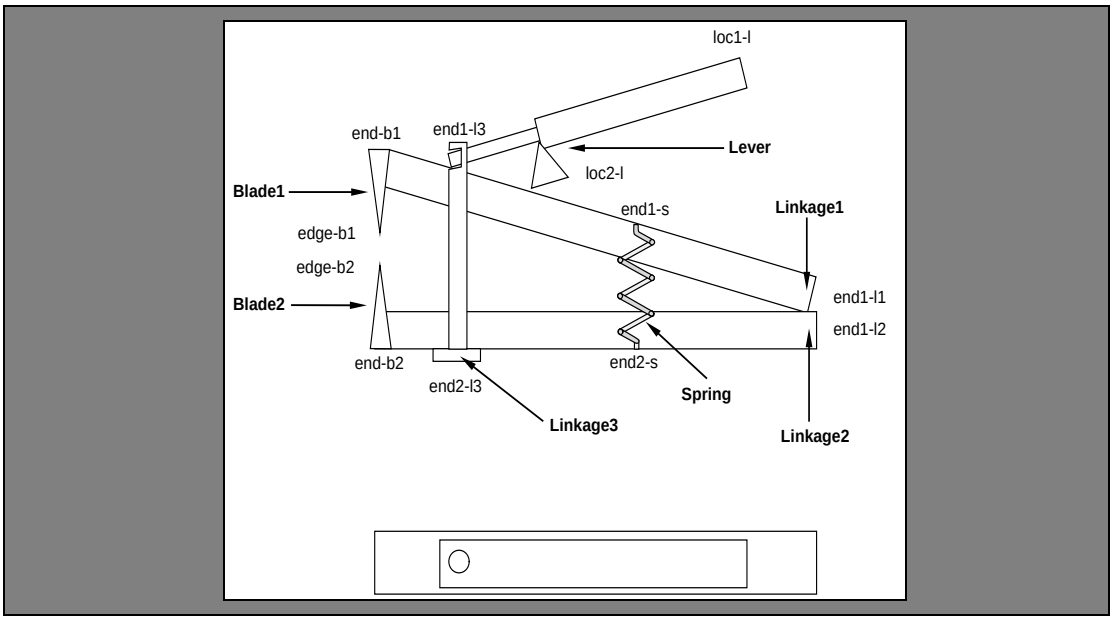


Figure 2.74 Fingernail clipper object primitives. A lever-object, a spring-object, three linkage-objects, and two blade-objects.

to a nutcracker and to a clothes pin. The device is similar to a clothes pin because the spring is not really at the end but in the middle where the force is applied. The device is similar to a nutcracker because the linkage-objects pivot at the end. The device is dissimilar to both in that force is applied at some intermediate region. These similarities and differences must be made explicit in both the SDD and the representation in general. The fingernail clipper SDD is shown in Fig. 2.75. The lower right-hand portion of the figure (at 1) shows a strong similarity to the SDD for the clothes pin, except that instead of a lever-object being in con-

tact we have a linkage object and role-play relationships. Note that there is a rigid connec-

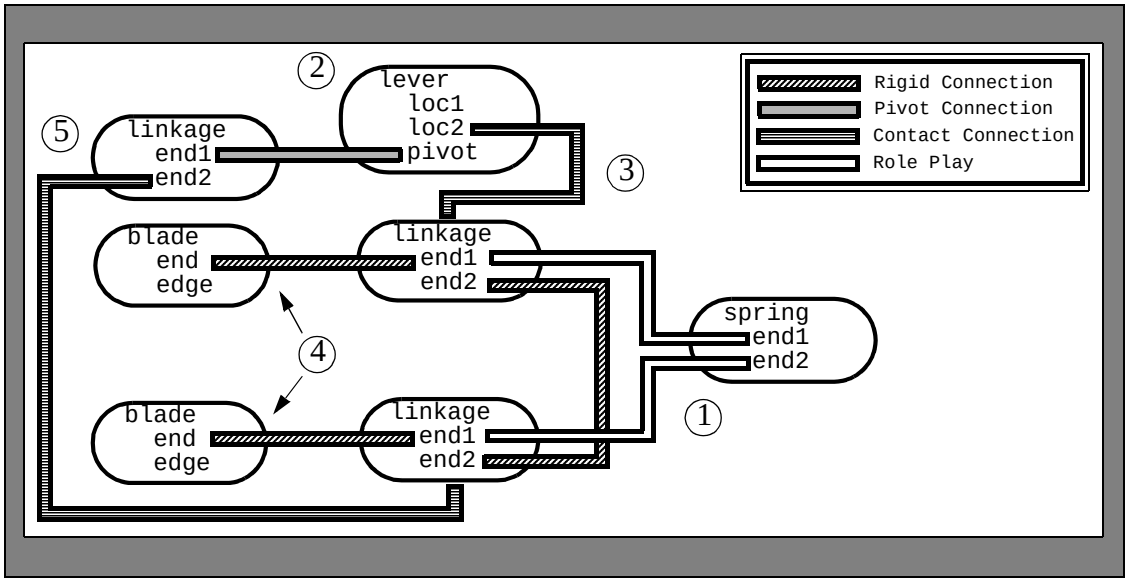


Figure 2.75 Fingernail clipper static representation diagram.

tion represented between the ends of the linkage-objects instead of the pivot connection between the handles of the nutcracker. The components have been arranged in a similar pattern to their physical appearance in Figs. 2.73, 2.74 to aid in following the connectivity between object primitives. The handle is represented with a lever-object (at 2). The region where force is applied, `loc2`, is represented as having a contact relationship with the upper linkage-object (3), but a region of contact is not specified. The pivot region of the lever-object is shown with a pivoting connection with the linkage-object at (5), which represents the fingernail clipper pin. This same linkage-object is also in contact with the lower linkage-object. The two blade-objects are depicted rigidly connected with the linkage-objects (4).

2.7.4 Complex Devices

Complex devices increase the number and type of components to the device, often requiring subgroups or mechanisms which perform specific tasks. One such device, a toy gun, is discussed and represented in this section. Three additional complex devices: (2) a corkscrew, (3) an eggbeater, and (4) a mousetrap are represented in Appendix A.4.

Toy Gun

The toy dart gun is a device with four major components: (1) a dart, (2) a body, (3) a trigger, and (4) a spring. The spring is single component, the dart and body are simple devices, and

the trigger is a simple compound device. Unlike other devices mentioned in this text, the

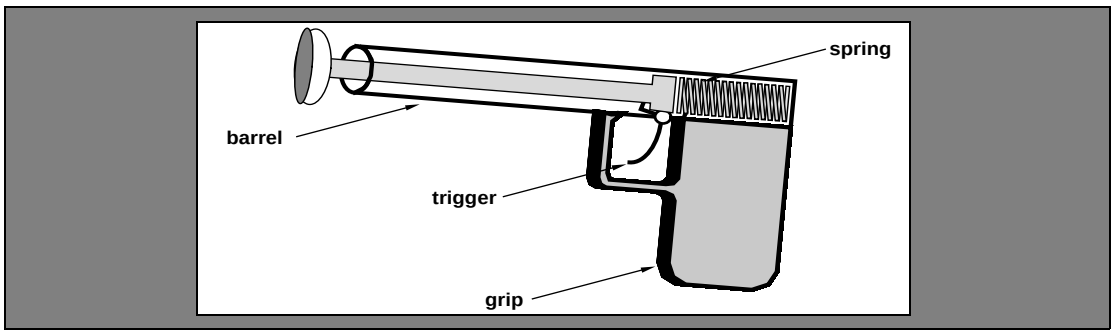


Figure 2.76 Toy gun illustration.

toy dart gun has a component which completely separates from the device in normal operation. The complexity of this device is seen in the idealized toy gun depicted in 2.77. In this figure, the dart is comprised of three linkage-objects of different sizes. The gun body is idealized as a container-object and a linkage-object, and the trigger mechanism is represented as a lever-object. The accompanying SDD depicts the dart as located in the barrel of the toy

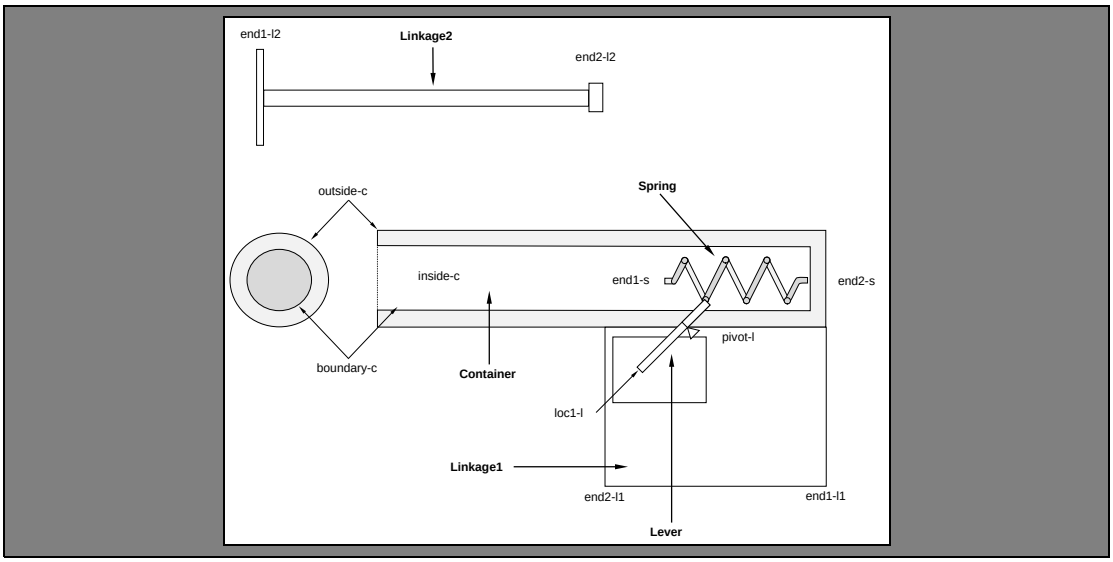


Figure 2.77 Toy gun object primitives. A container, a linkage, a spring, and a lever.

gun (1). One end of the linkage-object, `end2`, is represented as being in contact with the spring-object `end1` region and `inside` region of the container-object (2), and with the

loc2 region of the lever-object which represents the trigger (3). The lever-object pivot re-

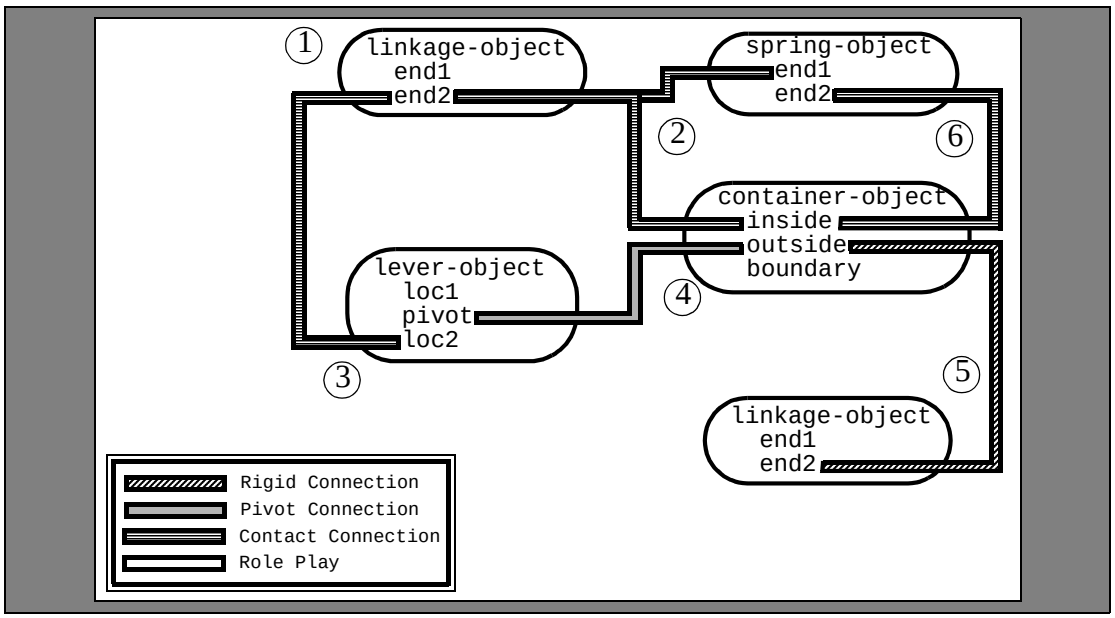
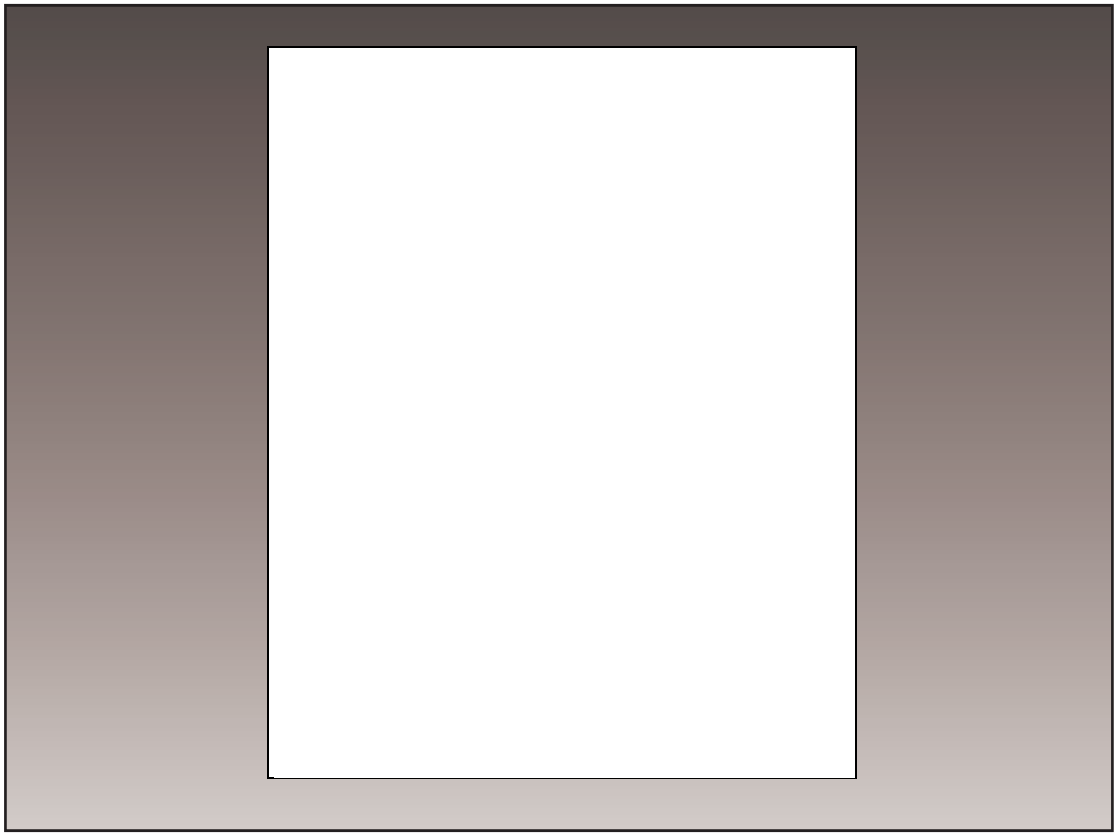


Figure 2.78 Static representation diagram for toy gun.

gion has a pivot connection with the outside region of the container-object. The container object outside region is also rigidly connected to the linkage-object (5). Finally, the other spring-object end region, end2, is in contact with the inside region of the container-object (6).



Chapter 3
Low-Level Device Dynamics: Behavior

Devices are used to perform problem solving tasks by undergoing change or by producing change in other objects in the environment. When a force is applied to an object, the object is said to be *perturbed* and, depending on its static description, may undergo state changes. Changes in state describe object *behavior*. For example, if an object is unrestrained in a dimension and is then forced in that dimension, then the object will change position. Because of this, object position is called a behavioral property. Every device function can be described as a combination of mechanical behaviors.

There are two ways of describing device behavior: (1) by what a device does, and (2) by how it does it. What a device does is called a *function*. Device function describes what perturbations are applied to a device and what happens as a result of the perturbation. This is called global or *high-level* behavior. Component behaviors and their combinations are called local or *low-level* behavior. Both high-level and low-level behavior describe physical change at different levels of granularity. The combination of low-level and high-level mechanical behaviors describes how objects behave and how the interactions between objects are perceived. In FONM, the causal dependencies which are used to represent low-level and high-level device behavior are called *device dynamics*. An object can be used in a specific task if its dynamic description produces the same states that its intentional description does under the same conditions.

When a device is perturbed in some way, the manner in which the perturbation is propagated throughout the device, and how it affects other objects, is dependent on the combined behavior of its components. For example, the nutcracker illustrated in Fig. 2.52 can be used to crack nuts open by applying leverage. When the user pulls the handles together, a force is produced in the handles. This force is transmitted through the pivoted ends, to the pivot component, and to the pecan. Because of the pivoted connection, the handles are able to rotate, but the pecan constrains their motion. The relative distance between the handles, the pivot location, and the pecan contact location magnify the force to an extent that its value exceeds the breaking strength of the pecan shell, and it cracks open. These interactions: rotation, pivot, transmission, magnification, and cracking describe behaviors which must be represented in order to describe complex mechanical behavior and function. As a result, mechanical behavior is the basic dynamic component in the FONM approach, because representing and recognizing behavior enables process models which use FONM representations to predict and explain object perturbations and their causal dependencies.

In this chapter, the representation constructs which are used in FONM to describe low-level device dynamics are presented. All device behavior can be described as permutations in the static descriptions of device components, so the discussion begins with a description of permutations which can be described for static object descriptions. The discussion ends with the presentation of device dynamics for six mechanical devices of increasing com-

plexity. Three topics are addressed in this chapter:

(1) Low-level object behavior
(2) Behavioral processes
(3) Process sequences and high-level object behavior

3.1 Low-Level Object Behavior

Each aspect of the object's static description affects its behavior. The behavioral properties describe what behaviors can take place, such as knowing an object's position, which is important in determining whether it is in contact with another object. The object's material properties causally enable or disable behaviors in which it might engage. For example, it is reasonable to assume that a steel knife can be used to whittle wood and that a rubber one cannot. To make this prediction, the dynamic relationship between material stiffness/strength and cutting must be made explicit. That is, FONM must represent that steel being harder than wood and rubber being softer than wood are causally related to cutting. Finally, the object's regions determine where behavior will take place, and the connections to other objects determine where behaviors will be propagated. For example, if it is known that two regions share the same location, then force can be transmitted between them at that location.

Each of the five FONM behavioral states is used to represent changes a mechanical object may undergo. A sequence of any particular behavioral property states for an object is called a *history* for the property, and the physical interaction responsible for the state change is called a behavioral *process*. A process and state history describe an object's low-level behavior for a particular property.

3.1.1 Continuous and Discrete Models of Mechanical Behavior

In FONM low-level physical behavior is represented qualitatively; as discrete models of continuous behavior. Mechanical objects move and stop, and they store, transmit, and transform energy. In so doing, they may transform their own physical form or that of other objects. In Applied Mechanics, these physical interactions are described as continuous behavioral processes. In qualitative physics, each of these interactions is considered to be qualitatively different; each describes a unique behavioral process. The difference is illustrated with the following example. Consider a spring stretching and a paperclip bending. In Applied Mechanics, stretching and bending are both described by a material's stress-strain characteristics in a theory of bending. Any object strain below the elastic limit describes stretching, while strain beyond the elastic limit describes bending. Stretched objects will, when released, return to their original size and shape. Bent objects, when released, retain some of their modified size and shape. In FONM, stretching and bending result in different behavioral property states, so they are represented with different behavioral processes. Instead of a single theory which describes all the behavioral dependencies, the FONM approach is to have a theory for each process and describe the dependencies between those processes.

3.1.2 Process Consistency with Applied Mechanics

A discrete model of device dynamics is only as good as its agreement with the quantitative models of Applied Mechanics. There are three conditions under which the discretization of physical behavior into behavioral processes can be considered consistent with Applied Mechanics, namely when they are: (1) dynamically consistent, (2) predictively consistent, and (3) continuity consistent. Behavioral processes are *dynamically* consistent with Applied Mechanics if their behavior is based on the same principles upon which Applied Mechanics is based. FONM processes are based on a restraint model. Objects which are not otherwise

specified are free to participate in processes, and processes which are enabled remain enabled unless otherwise acted upon. For example, if an object is not specifically restrained in a dimension and direction, then it is free to move in that dimension and direction. Unless other knowledge is specified, a generic object is free to move in any dimension and could be recognized as a falling object. The same object, once falling, will continue to fall unless it collides with an object or is otherwise acted upon by another process.

Behavioral processes are *predictively* consistent with Applied Mechanics if, given the same object and conditions, predictions based on qualitative processes agree with those based on Applied Mechanics. Consider an object which is stretched beyond its elastic limit. If the object is initially recognized as a participant in a stretching process, and then as a participant in a bending process, and the resulting states qualitatively agree with the stress-strain analysis predicted by Applied Mechanics, then the model is consistent for this example. Predictive consistency must hold regardless of the representational granularity. For example, it doesn't matter how a game of catch is represented as long as those ball physical states which are qualitatively described by the model agree with those predicted by Applied Mechanics.

Behavioral processes are *continuity* consistent with Applied Mechanics if they describe continuous behavior when Applied Mechanics would describe continuous behavior. This condition guarantees that transient behavior, which is ignored in FONM behavioral processes, does not produce discontinuities unless those discontinuities would be produced in a continuous model. For example, a stretching process results in a size state change, and an internal force state change on the stretched object. If these states collectively enable the bending process, then the two discrete processes are continuity consistent with Applied Mechanics.

3.2 Behavioral Processes

Behavioral processes describe dynamic behavior: they are enabled and disabled by physical states, and they result in physical state changes. In FONM, behavioral processes are represented with the process knowledge structure. Processes have six roles: src, dst, dimr, from, to, and inst. The *src* and *dst* roles refer to the source and destination locations (regions or objects) where the interaction takes place. The *dimr* role refers to the applicable dimension and direction of the process. The *from* and *to* roles refer to enabling, and resulting states of the process, respectively, and the *inst* role refers to an object or region whose function mediates the process. There are two kinds of process enablement, as shown in Fig. 3.1: behavioral preconditions and compositional preconditions. *Behavioral* preconditions (1) are

behavioral property states and take the same form as the resulting state (2).

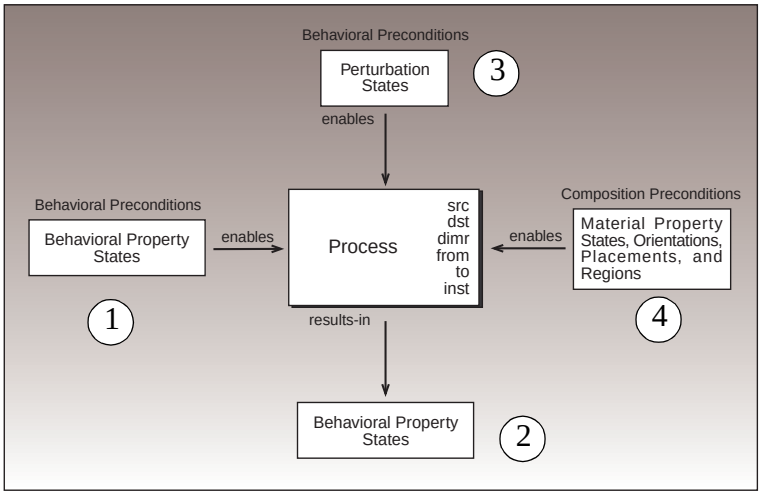


Figure 3.1 Process enablements and resulting states.

Perturbation states (3) are aforce states, and are set apart in Fig. 3.1 because they can result from actions (such as PROPEL-PUSH) as well as from processes which effect aforce state changes. *Compositional* preconditions (4) are material property states. In the rubber knife example discussed above, the material stiffness property value is an enablement for cutting. When specific property values are associated with object behavior, they enter the representation as compositional preconditions.

Object behavior has both local and global effects. Locally, each object region can participate in processes, resulting in local behavior states. The region then fills the *src* or *dst* process roles, and the *to* role states resulting from an enabled process are *local* states associated with the region at which the process takes place. The union of object local states describes a *global* state for the entire object and property. For example, the union of local aforce states on an object describes the global object aforce state.

When the process src and dst roles are filled with objects, rather than regions, the object CG is assumed and no local behavior can be inferred, because the locations where the behavior takes place are not specified.

The process representation can be statically illustrated by considering the action of squeezing the two nutcracker handles together in the absence of a nut (Fig. 3.2). The transmission of force from each linkage to the pivot piece is represented using a process which

describes force transformations, called TRANSFORM.

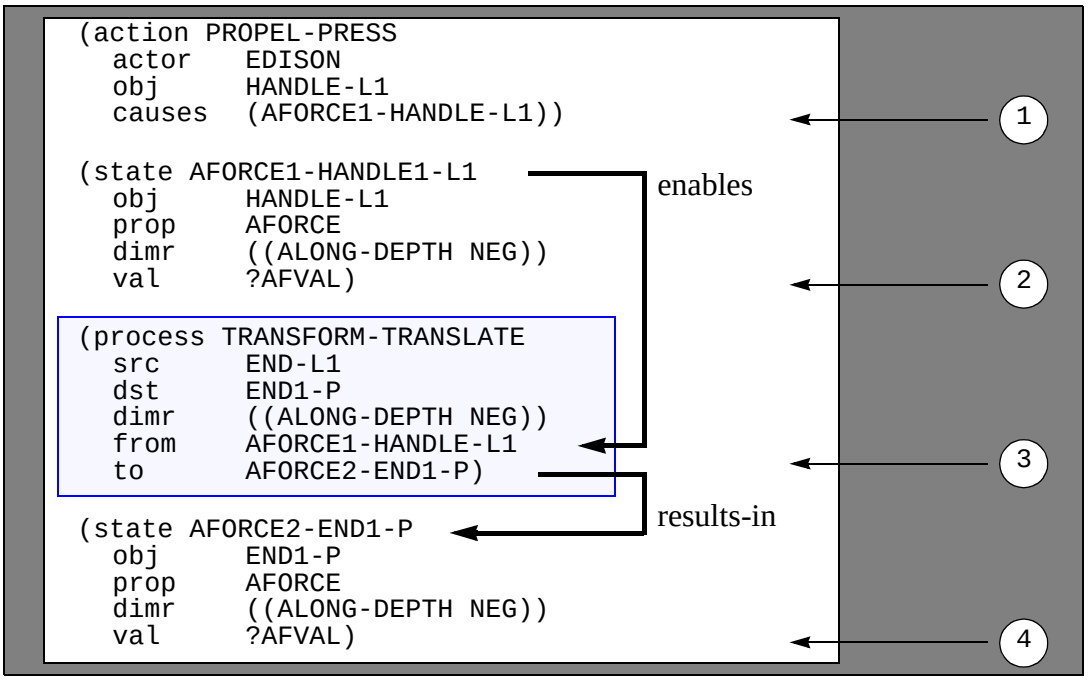


Figure 3.2 Process representation for force transmission.

Fig. 3.2 represents the states which enable the translation of force applied at the nutcracker handle, through the pivoted connection with the pivot component, to the pivot end. In the example, the nutcracker components LEVER1 (L1) and PIVOT (P) are assumed to be in contact in the dimension and direction (ALONG-DEPTH NEG). The lever handle, HANDLE-L1, when pressed (1), causes an applied force state at the same location and in the same (ALONG-DEPTH NEG) dimension and direction. In FONM, when a force is applied to an object, the force is conveyed to all regions. That is, there are no transient effects modeled in FONM. The aforce value is divided into the number of connections which can support force transmission in the specified dimension and direction. Thus the aforce at HANDLE-L1 (2) has the same value (AFVAL) as that at the region END-L1, because it is LEVER1's only regional location which is connected to PIVOT. The *src* role of TRANSFORM-TRANSLATE is filled by the region HANDLE-L1. The *dst* role is similarly filled by the region END1-P. The *from r*ole is filled by a state (3) which represents the LEVER1 force in dimr (ALONG-DEPTH NEG) prior to the process, and the *to r*ole is filled by a state which represents the force transmitted by HANDLE-L1 to the pivot piece (4). When the same interactions are described for LEVER2 and PIVOT, a force of the same magnitude is transmitted to END2-P, but in the (POS) direction. When the global aforce state on PIVOT is updated, these two forces cancel one another and the representation produces the same result as a nutcracker in static equilibrium.

3.2.1 Behavioral Process Primitives

TRANSFORM is called a behavioral process primitive (BPP). In FONM, there are five BPPs, distinguished by the type of behavioral property state which they affect: MOTION, RESTRAIN, TRANSFORM, STORE, and DEFORM. These primitives form a causal hierarchy for describing mechanical behavior, because the state changes which they represent

describe the range of possible mechanical behavior, as illustrated in Fig. 3.3 and Table 3.1:.

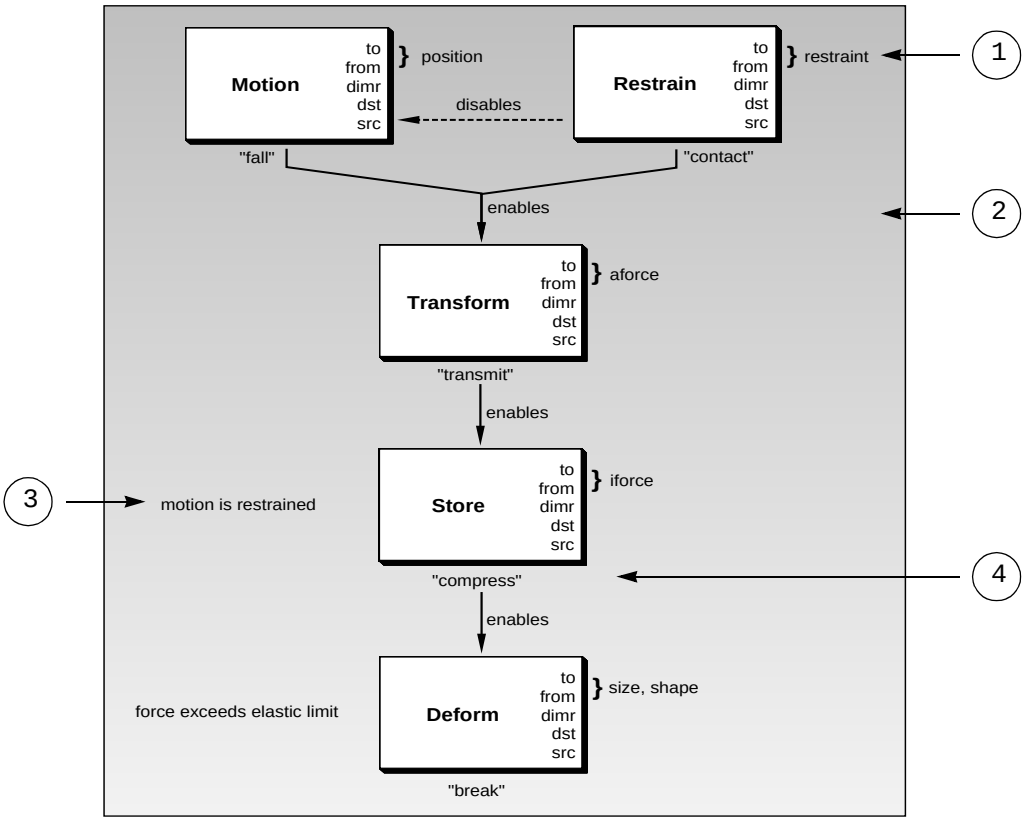


Figure 3.3 Behavioral process primitives.

In Fig. 3.3, each primitive class is depicted as a process. The behavioral property state type resulting from an enabled primitive is labeled outside the right side of each process (e.g., restraint at 1). The causal dependencies between primitive classes are depicted on the arcs connecting the processes (2). Representative enabling preconditions are shown (where appropriate) to the left of each primitive (3). Beneath each primitive is an example of a lexical specialization for each primitive (at 4, for reference).

Each BPP is represented with a process knowledge structure. MOTION describes object motion, and represents changes in object position. RESTRAIN describes object connectivity, and represents changes in object restraint. TRANSFORM describes force transformation between objects, and represents changes in applied force dimension, direction, and magnitude. In Applied Mechanics, the field of rigid-body statics describes force transformations which do not modify the object. TRANSFORM therefore describes qualitative rigid-body statics. TRANSFORM is enabled by an applied force on one object which (minimally) comes into contact with a second object. STORE describes non-permanent (elastic) deformation, and represents changes in internal force (energy). STORE is enabled by an applied force, TRANSFORM (nominally TRANSFORM-TRANSMIT), and a restraint state on the objects in contact. DEFORM describes permanent object deformation, and represents changes in physical size or shape. When an object is deformed elastically, such that its original size and shape are retained, STORE is implied. If the elastic limit is exceeded, for example when an object is bent or broken, DEFORM is implied.

Table 3.1: illustrates the relationships between BPPs, their behavioral and quantity pre-

conditions, and the physical states they cause.

Process Type	Behavioral Preconditions	Perturbation Preconditions	Resulting State
RESTRAIN	none	equivalent region position	restraint
MOTION	RESTRAIN (disables)	nonzero global force	position
TRANSFORM	RESTRAIN, MOTION	none	aforce
STORE	TRANSFORM	aforce < elastic limit	iforce
DEFORM	STORE	aforce > elastic limit	size

Table 3.1: Behavioral process primitives.

In FONM, process representations are always depicted as unidirectional. However, behavioral processes are bidirectional. For example, when an object is in motion its center of gravity will change position, but behaviorally there may be no difference between the object changing position and the object CG remaining stationary and everything else moving. The FONM representation implies directionality for the purpose of continuity, but the directionality is not intended to preclude the alternative point of view.

3.2.2 MOTION

Each BPP primitive represents a generic process class, and specializations on its roles represent various process subclasses. A *generic* process defines the minimum structure whereby the process can be recognized. For example, object motion is represented with the generic process MOTION, which is based solely on two behavioral preconditions: global restraint freedom in a dimension/direction, and global unbalanced force in the same dimension/direction. Once enabled for a particular object and dimension/direction, MOTION represents the resulting change in object position in dimension/direction. These relationships are illustrated in Fig. 3.4.

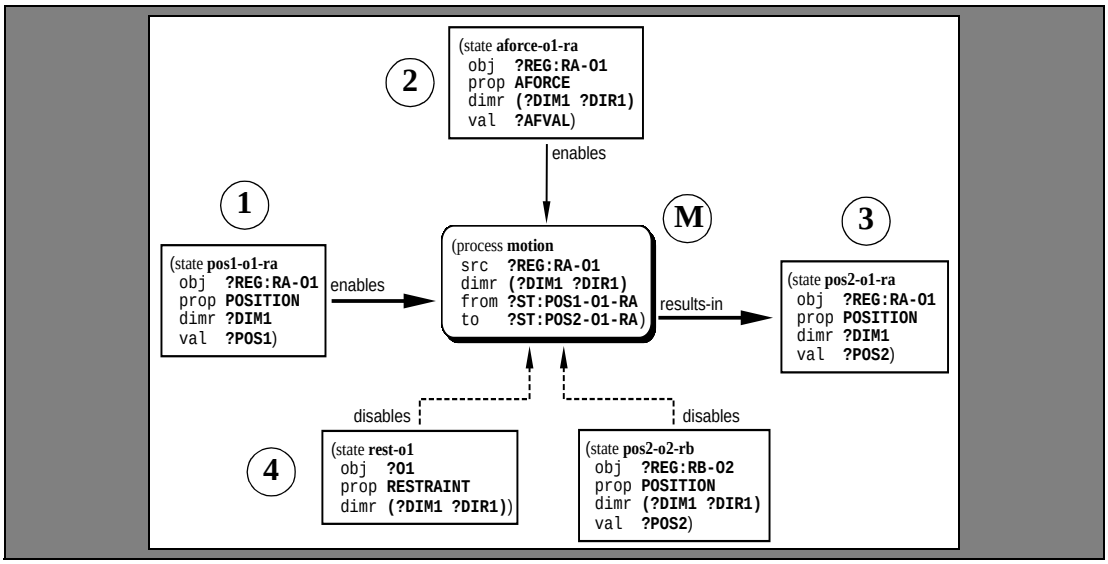


Figure 3.4 Frame representation for the MOTION behavioral process primitive.

Fig. 3.4 depicts the MOTION process as a schema (M) which takes an object position (1), and an applied force (2) and describes a change in position (3). The applied force defines the dimension and direction in which motion will occur, while the positions are defined by the dimension and a value. The original position state fills the BPP *from* role, while the new position state fills the BPP *to* role. The two disabling states (4) represent conditions which can disable the MOTION process. The first is a restraint state at the region where the force is applied, and the second is a second object which is positioned where the first will be when the process completes. The disablements are depicted with dotted lines to distinguish them from enabling conditions.

The generic MOTION process does not specify the object's path, only that the object's position will change and what the dimension will be. Motion specializations are represented by further specifying the type of applied force, the type and location of object restraints, or other physical property values. By specializing the MOTION enabling conditions, the process roles are specialized and path information can be described. Fig. 3.5 illustrates the family of MOTION class specializations which can be represented this way.

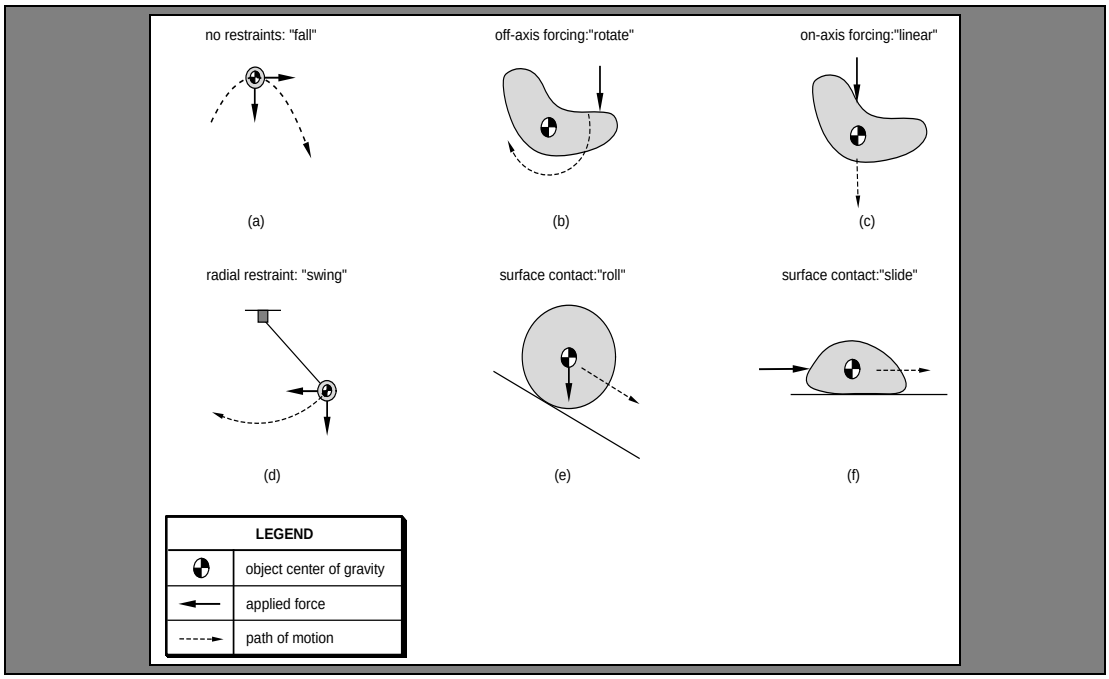


Figure 3.5 MOTION process specializations, (a) fall, (b) rotate, (c) linear, (d) swing, (e) roll, and (f) slide.

In Fig. 3.5, the dark arrows represent the location, dimension, and direction of applied force and the thin dotted arrows represent the path of motion. The motions depicted in Figs. 3.5a-f are representative of motions which MOTION should be able to describe. Fig. 3.5a illustrates MOTION-FALL, which represents an unrestrained object forced by a gravitational field. Figs. 3.5b and c illustrate rotational and linear motion as MOTION specializations where the forcing is off the centroidal axis dimension, and on the centroidal axis dimension, respectively. These three motion types are specialized to swinging, rolling, and sliding, Figs. 3.5d, e, and f by specifying a type of contact. The hierarchical relationships between generic MOTION and these six specializations are illustrated in Fig. 3.6 and tabulated in

Table 3.2.

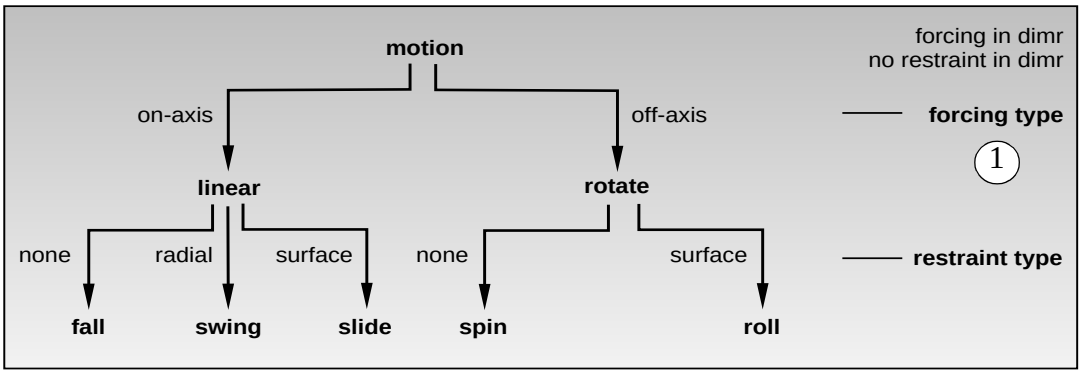


Figure 3.6 Hierarchical relationships between MOTION processes, differentiated by process role instantiations.

The hierarchy in Fig. 3.6 distinguishes motion processes based on how the MOTION behavioral preconditions are instantiated. The labels on the right hand side of the figure denote the precondition types (1). For example, by specifying whether or not the applied force is coaxial with the object CG, the motion class is distinguished between linear and rotation.

Motion Class	Unrestrained Dimensions Connection Type	Forcing Dimension	Motion Path
TRANSLATE	translation none	on-axis	straight
FALL	gravitational translation none	on-axis gravitational	vertical
SLIDE	translation surface contact	on-axis gravitational	planar
ROTATE	rotation pivot	off-axis	circular
SWING	gravitational translation hang	tangential gravitational	circular vertical
ROLL	translation, rotation surface contact	off-axis gravitational	circular

Table 3.2 Specializations of the MOTION behavioral process primitive.

Motion specializations are classified, individually, by their physical restraints and their forcing. Connection type is divided into translational freedom, rotational freedom, and surface contact. Translational and rotational freedom reference the dimension/direction of motion. Forcing type is divided into on-axis and off-axis forces. On-axis forcing is applied through the object's center of gravity, while off-axis forcing is applied elsewhere (as in tangential forcing). Motion types can be combined to describe complex paths. For example,

an object rolling down a hill is both rotating and falling. The forcing for rolling is obtained by an off-axis component of gravitational acceleration, as illustrated in Fig. 3.5e.

The representation for MOTION in Fig. 3.4 is shown instantiated for the SLIDE motion specialization in Fig. 3.7**.** The perturbation is applied at region RA-O1, in dimension DIM1, and in direction toward the O1 CG (shown at 1).

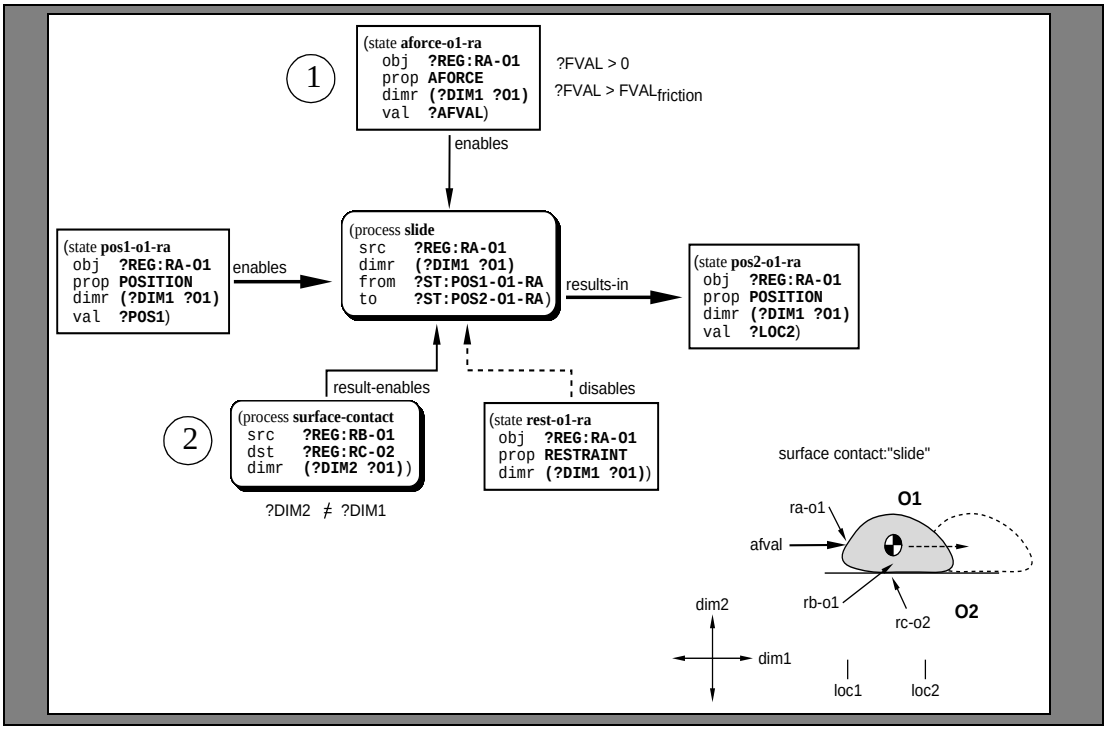


Figure 3.7 Graphical representation of SLIDE process specialization of the MOTION process class.

The value of the force state AFORCE-O1-RA, AFVAL, must be unbalanced to enable MOTION. To represent sliding with MOTION-SLIDE, AFVAL must exceed the friction force between object O1 and object O2. In addition, MOTION-SLIDE specifies surface contact with O2, and that the perturbation dimension, DIMR, not be the dimension of contact (2). These additional representational facets are shown at (1) and (2).

3.2.3 RESTRAIN

The relationships between objects in contact are often referred to as state relations, because they describe causal dependencies but do not appear to be dynamic. For example, when a book is lying on a table, it is being supported by the table, and "support" is considered a state relation between the book and table (i.e., the book is supported by the table). Although the observed result is a disabled potential for book motion, contact between the book and the table produces changes in the restraint states of each object: on the book a downward restraint, and on the table an upward restraint (see Fig. 3.8). It is the restraint states which disable the potential for book, and table, motion. These restraint states persist as long as the book and table remain in contact. When the contact is discontinued, the restraint states no longer exist. Thus described, "support" is not a state, per se, but a dynamic relation which

describes a form of physical behavior.

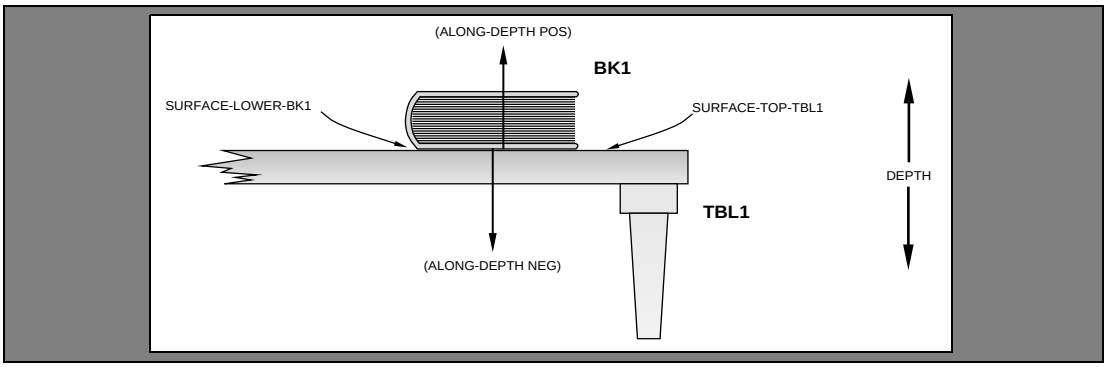


Figure 3.8 A book supported by a table describes restraint states on both the book and the table.

Fig. 3.8 illustrates the dependency between two objects in contact. The book cover, region SURFACE-LOWER-BK1, and the table top, region SURFACE-TOP-TBL1, share the same physical position in the vertical (ALONG-DEPTH) dimension. Each object has a local restraint state, in the vertical (ALONG-DEPTH) dimension, produced by the presence of the other object. The restraint state on the book is in the (ALONG-DEPTH NEG) dimension/direction, while the restraint state on the table is in the (ALONG-DEPTH POS) dimension/direction. According to the MOTION restraint disablement, BK1 motion is disabled in the (ALONG-DEPTH NEG) dimension/direction unless TBL1 also participates in MOTION in the same dimension/direction. The converse is also true; TBL1 cannot participate in MOTION in the (ALONG-DEPTH POS) dimension/direction without BK1.

In FONM, the RESTRAIN process describes objects in contact and the resulting restraint state changes. RESTRAIN involves two objects. Generic RESTRAIN, called CONTACT (Fig. 3.9), has a single behavioral precondition; each object has a region whose position is identical to that of the other object. The result of an enabled RESTRAIN is two local restraint states: one on each object, in the same dimension but opposite directions. RESTRAIN is a process, so the restraint states resulting from an enabled RESTRAIN remain

active until the process is explicitly disabled.

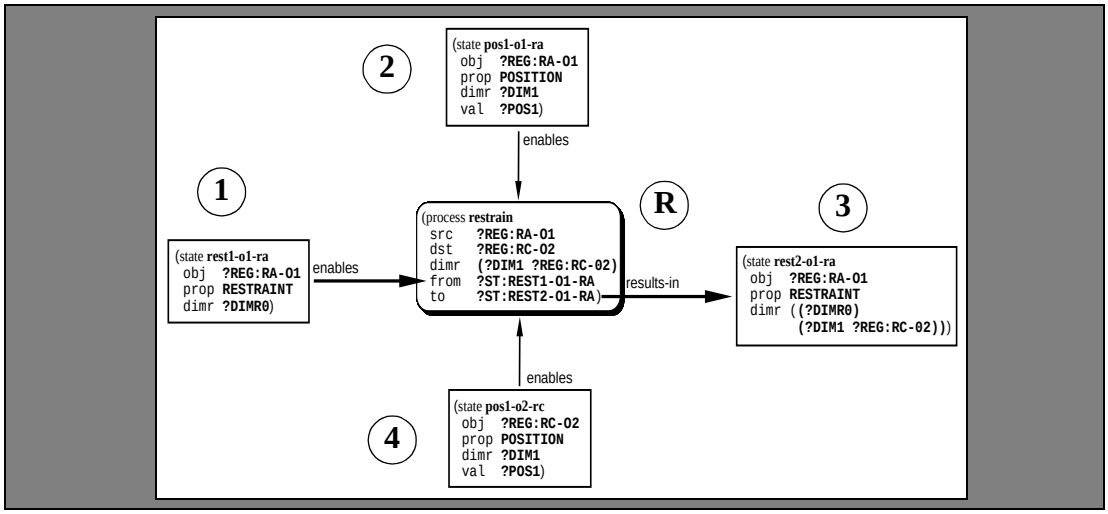


Figure 3.9 Frame representation for the RESTRAIN behavioral process primitive.

Fig. 3.9 shows how RESTRAIN organizes the interaction between restraint and position for objects in contact (R). The enabling condition, two object regions in the same location, is shown by the position states at (2) and (4), where the common position is POS1. The initial restraint state is labeled at (1) and the resulting state is labeled at (3).

Many contact relationships can be represented by specializing the RESTRAIN process. For example, the RESTRAIN process used to represent the book and table interaction is called SUPPORT. SUPPORT is a specialization of RESTRAIN-CONTACT where one object is placed above the other (i.e., in the vertical dimension). The opposite case, where one object is placed below the other is shown in Fig. 3.10, along with two other conditions

which can be represented as RESTRAIN class specializations.

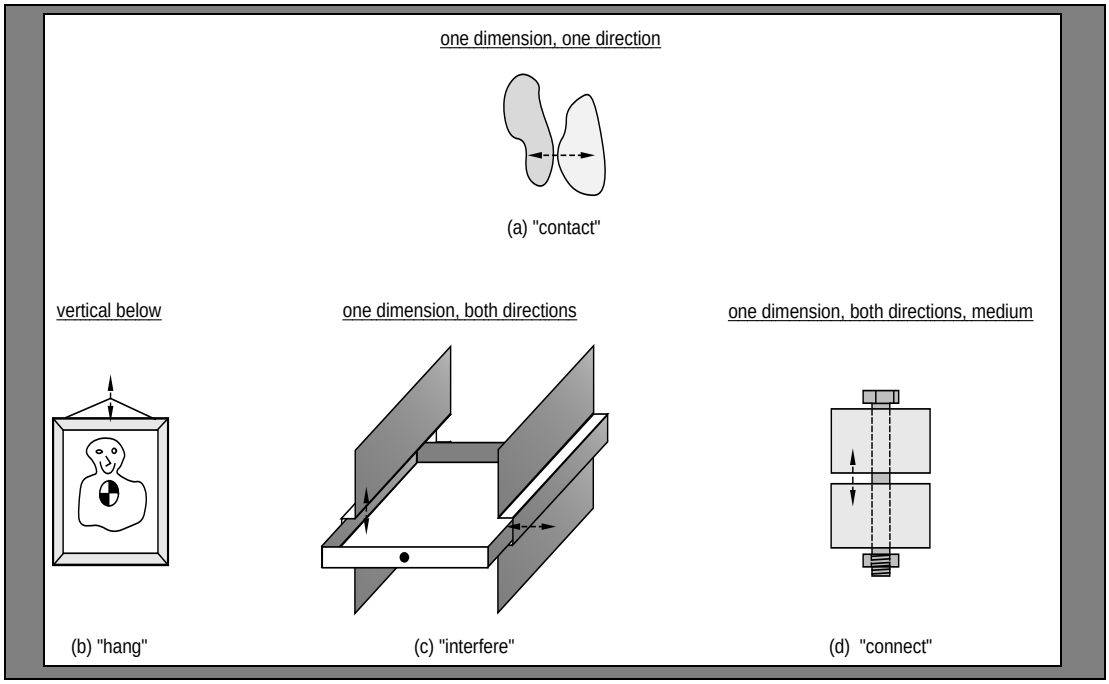


Figure 3.10 RESTRAIN process class specializations, (a) contact, (b) hang, (c) interfere, (d) connect.

Fig. 3.10 illustrates degrees of contact and their effect on object restraint. The arrows represent the dimension of contact between the two objects. Simple contact, shown in Fig. 3.10a, produces a restraint in a single dimension and direction. By specifying which dimension, number of directions, the relative placement of the two objects, and the presence of an instrumental object, such as a nut & bolt, three basic RESTRAIN class specializations: contact, interfere, and connect can be defined.

RESTRAIN-CONTACT: Motion for objects in contact is disabled in a single direction. The hanging picture illustrated in Fig. 3.10b is an example of RESTRAIN-CONTACT, called HANG, where the picture center of gravity is placed below the contact location.

RESTRAIN-INTERFERE: Contact between objects prevents motion along an entire dimension. One example of interference is an object sliding or rolling inside a channel where its motion is constrained to the channel by the channel walls. The desk drawer shown in Fig. 3.10c can only slide along the slot in which it fits; its motion is otherwise disabled by the slots on either side of it.

RESTRAIN-CONNECT: Objects which are connected lose mobility in multiple dimensions. A third object acts as an instrument, or medium, to fasten or join the objects (such as tacks, bolts, and clamps). The bolted blocks shown in Fig. 3.10d are held together by a nut and bolt, where the function of the nut and bolt is instrumental to the connection.

The hierarchical relationships between the generic RESTRAIN process, CONTACT, and the other RESTRAIN subclasses, INTERFERE, and CONNECT, are depicted in Fig.

3.11.

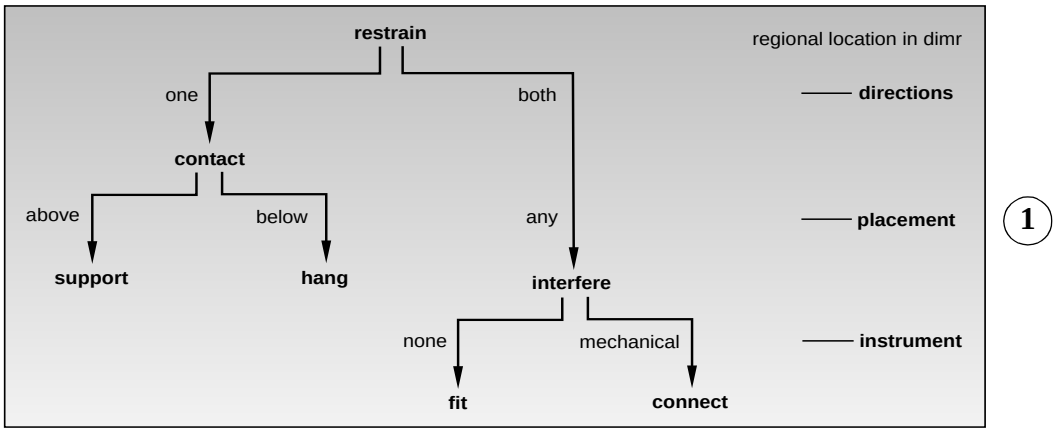


Figure 3.11 Inheritance hierarchy for RESTRAIN process, specialized by process role instantiations.

Specializing RESTRAIN processes enables representation of increasingly constrained motion. The process subclasses depicted in Fig. 3.11 are specialized according to how the behavioral process preconditions, specifically the statics-related preconditions, are instantiated. The process RESTRAIN-CONTACT can represent objects supported by or hanging from other objects, by specifying the dimension of contact to be vertical and by specifying the relative placement (1) of the two objects. These relationships are also shown in Table 3.3. In both HANG and SUPPORT, the process disables motion in the vertical dimension and the downward (NEG) direction. The supported object's center of gravity is placed above the restraining object, and the hanging object's center of gravity is placed below the restraining object.

Restrain Specialization	Restrain Class	Dimension	Placement	Instrument
SUPPORT	Contact	Vertical	Above	None
HANG	Contact	Vertical	Below	None
FIT	Interfere	Any	None	None
TIE	Connect	Any	None	Rope
BOLT	Connect	Any	None	Nut & Bolt

Table 3.3 Specializations of the RESTRAIN behavioral process primitive.

The process RESTRAIN-INTERFERE is used to represent objects which fit together. Two objects fit along a particular dimension when they contact in at least two colinear points. The result is restraint along the entire dimension of contact, which disables motion along the entire dimension of contact. The process RESTRAIN-CONNECT can represent objects which are tied or bolted together. In both INTERFERE and CONNECT, motion is disabled along the entire dimension and, depending on the object statics, along multiple dimensions. The instrumental object mediates the connection by providing the interference, rather than the restraining object. In RESTRAIN-SUPPORT and RESTRAIN-HANG, instrumental

objects *can* be used; however, in RESTRAIN-CONNECT instrumental objects *must* be used. This is why no instrument is specified for the former constructs in Table 3.3. As an example of a required instrumental object, RESTRAIN-CONNECT is represented in frame notation below (Fig. 3.7) for the two bolted blocks depicted in Fig. 3.10d.

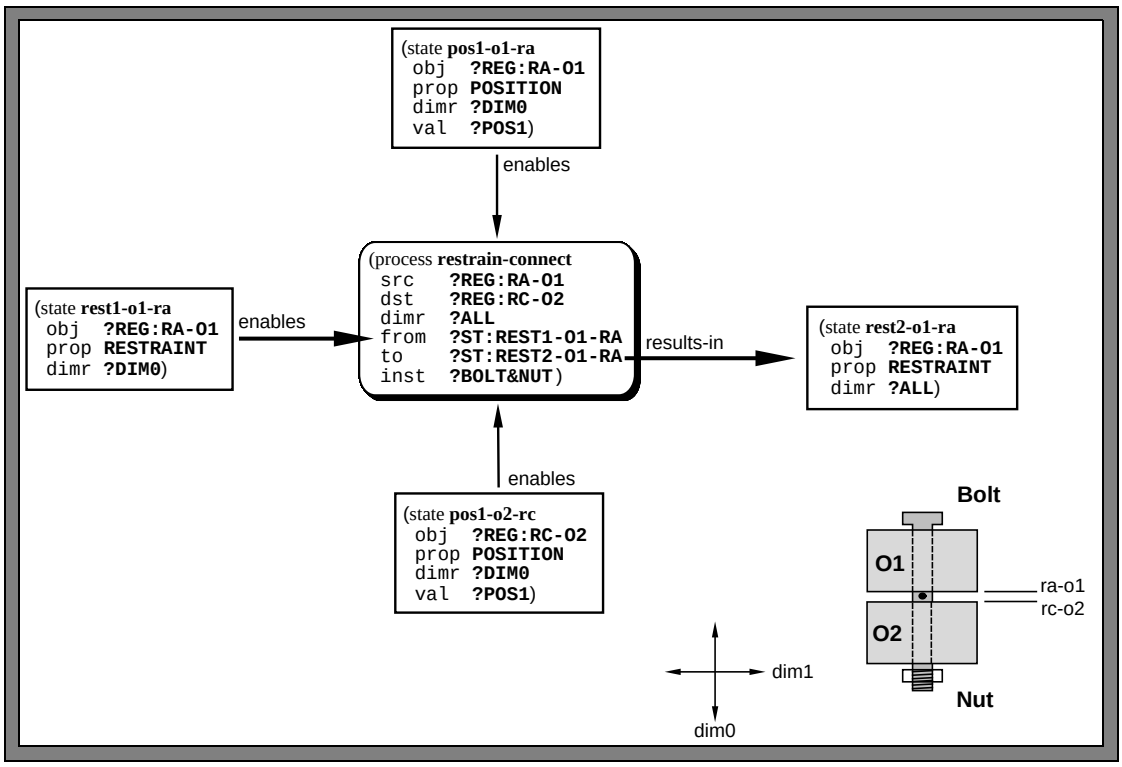


Figure 3.12 Graphical representation of CONNECT process specialization of the RESTRAIN process class.

In Fig. 3.7, the representation for RESTRAIN is shown instantiated for the subclass RESTRAIN-CONNECT**.** The states POS1-O1-RA and POS1-O2-RC represent the position of RA-O1 and RC-O2, in dimension DIM0 and value POS1. The bolt and nut provide additional contacts, and the overall result is full restraint in DIM0. In order for the bolt and nut to assist in restraining the blocks, independent RESTRAIN processes must be composable into generalized RESTRAIN processed. That is, RESTRAIN should be transitive.

RESTRAIN Process Combination

Processes can be combined to describe similar, albeit increasingly complex, behavior. RESTRAIN-INTERFERE and RESTRAIN-CONNECT restrain objects along entire dimensions. In some cases, such as a peg in a hole, interference results from multiple contacts between two objects. By combining the effect of multiple RESTRAIN processes, interference and connection involving more than two devices can be represented. The bolted blocks in Fig. 3.7 can only be represented with RESTRAIN-CONNECT with the help of a nut and bolt. This can be accomplished by combining two additional RESTRAIN-CONTACTs, in opposite directions, one for each block. This transitivity can also be applied to describing interference. Recall the nutcracker device presented on Page 86. One subgoal in using a nutcracker for cracking a pecan open is to restrain the pecan in the dimension in which the handles will be pressed together. Pecan restraint is achieved with a RESTRAIN-

INTERFERE-JAWS between JAW-L1 and the JAW-L2, both in the dimension of pressing, as illustrated in the lower diagram of Fig. 3.13.

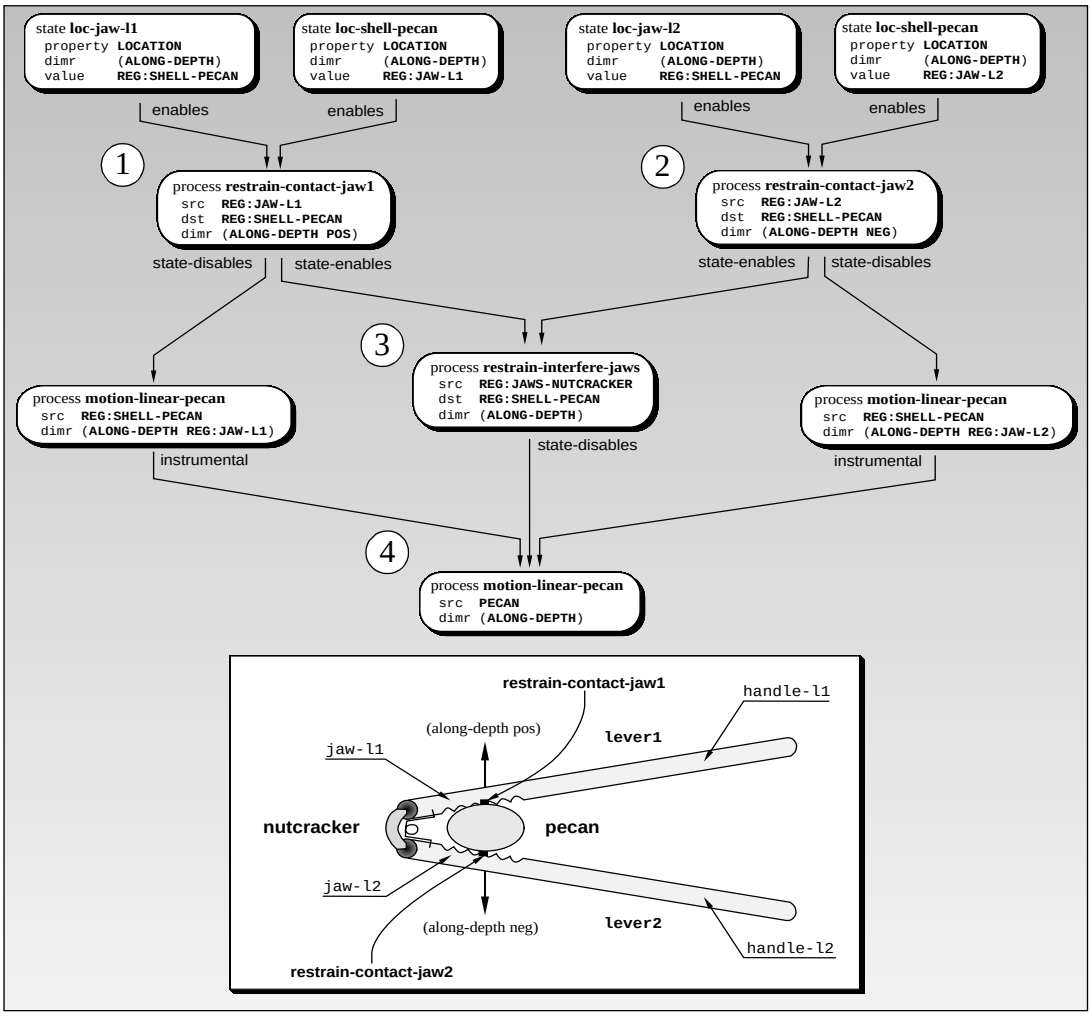


Figure 3.13 RESTRAIN-CONTACT and RESTRAIN-INTERFERE processes, instantiated for pecan nutcracking with a nutcracker.

The upper diagram in Fig. 3.13 represents the nutcracker jaw and pecan shell locations which enable RESTRAIN-CONTACT-JAW1 (1) and RESTRAIN-CONTACT-JAW2 (2). RESTRAIN-CONTACT-JAW1 results in a state which disables linear pecan motion in the (ALONG-DEPTH POS) dimension/direction. RESTRAIN-CONTACT-JAW2 results in a state which disables linear pecan motion in the (ALONG-DEPTH NEG) dimension/direction. The two processes combine to enable the RESTRAIN-INTERFERE-JAWS (3), whose resulting state disables linear pecan motion in the (ALONG-DEPTH) dimension (4).

3.2.4 TRANSFORM

The TRANSFORM process describes transmission of force between objects. Force magnification is a specialization of TRANSFORM which is used to represent mechanical advantage. Mechanical advantage is the principle by which mechanical devices function, so

TRANSFORM represents how mechanical devices work.

TRANSFORM involves two objects. One object is either in motion or is forced. This object comes into contact with a second object, and transmits force to the second object, as illustrated in Fig. 3.14. The process has two behavioral enablements: an applied force on the forced or moving object, and a restraint state resulting from a RESTRAIN-CONTACT between the two objects, both of which are defined in the same dimension/direction. When elastic objects (see item 1 in Fig. 3.14) come into contact (2), force is always transmitted, so the generic TRANSFORM process is TRANSFORM-TRANSMIT (T). An enabled TRANSFORM-TRANSMIT results in an applied force state on the unforced object in the same dimension/direction of, and having the same value as that of the forced object (3).

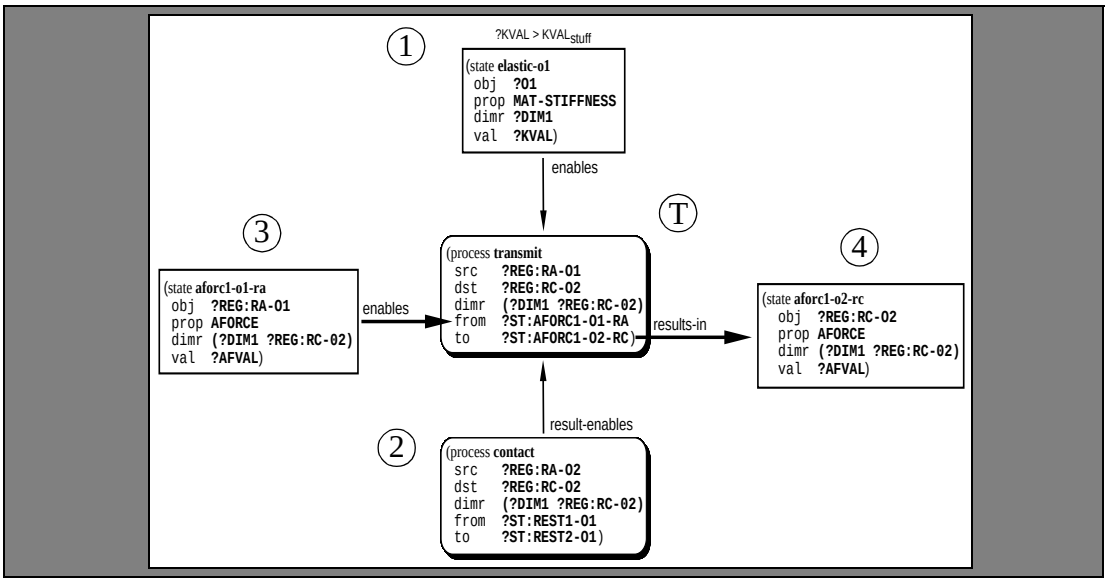


Figure 3.14 Frame representation for the TRANSFORM behavioral process primitive.

In Fig. 3.14, the state AFORCE-O1-RA (3) represents the process' perturbation behavioral enablement, applied at region RA on object O1. The dimension/direction of applied force is DIM1, and the associated value is AFVAL. The reaction to the applied force and contact between these objects is an applied force, AFORCE-O2-RC (4), at region RC on object O2. AFORCE-O2-RC has the same dimensions and value role bindings as AFORCE-O1-RA. In the figure, these applied force states are denoted with solid causal links to indicate that they are the states affected by the dynamic interaction.

The generic TRANSFORM process can be used to represent other force transformations by specializing the object statics of the transmitting object or the perturbation preconditions. For example, three transformation types are illustrated in Fig. 3.15. Force transmission (Fig. 3.15a) describes uniaxial force transfer from one object to another, and requires the two objects to be in contact in the same dimension and location as the applied force (an on-axis force). In the figure, force transmission is illustrated for both linear and rotational forces, as are the other transformations illustrated (force translation, Fig. 3.15b,

and magnification, Fig. 3.15c).

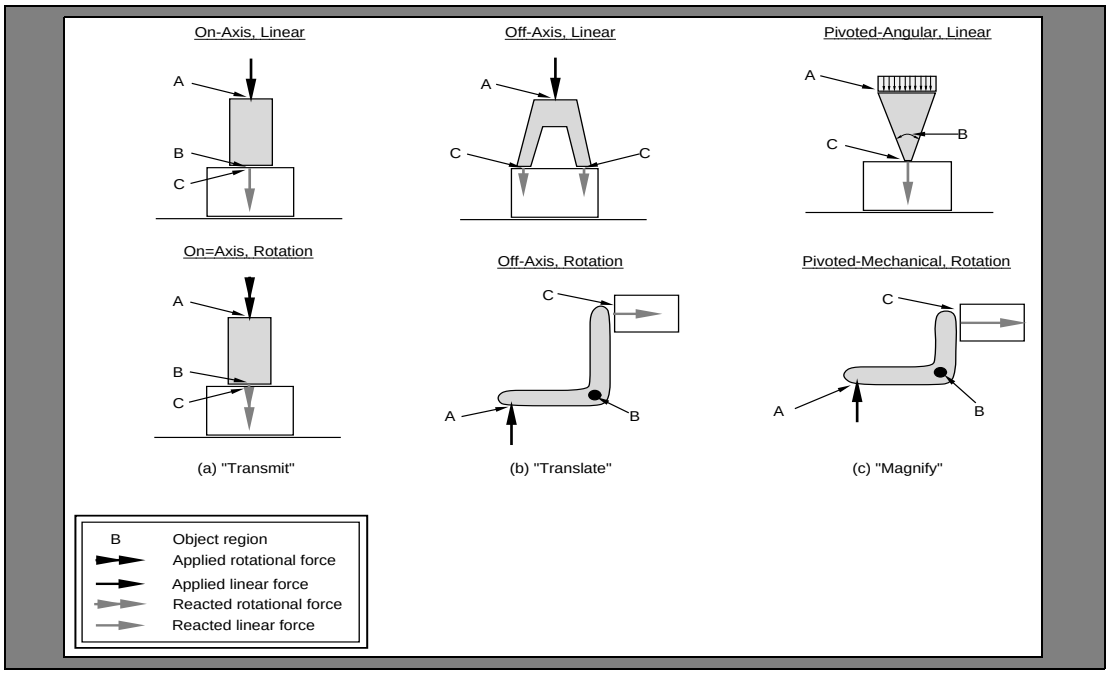


Figure 3.15 Illustrations for three TRANSFORM process specializations, (a) TRANSMIT, (b) TRANSLATE, and (c) MAGNIFY.

In Fig. 3.15, dark arrows represent applied forces (at A regions), while light arrows represents reacted forces (at C regions). Arrow length indicates the relative magnitude between the applied and reacted forces. For example, force transmission never changes dimension, direction, or magnitude, while magnification always changes magnitude.

TRANSFORM-TRANSMIT represents linear uniaxial force extension through transmission. The simplest example is a stick, where the point of contact is in line with the applied force, called on-axis forcing. The force dimension, direction, and magnitude remain the same, as depicted in Fig. 3.15a.

TRANSFORM-TRANSLATE represents transmission of force in a translational dimension other than that of the applied force. This is called off-axis forcing. Using a stick as a lateral probe, such as to knock a frisbee from a tree, is an example of linear off-axis translation. Linear translation also occurs when there are two or more contact locations, such as depicted in the upper portion of Fig. 3.15b. In such cases, the applied force magnitude is divided among the contact regions. Rotational force translation is similar to magnification, but the distances through which the applied force acts are equal.

TRANSFORM-MAGNIFY represents transmission of force from one region to another, whereby mechanical advantage is gained as a ratio of applied and reacted forces to the distance through which the applied and reacted forces travel, as shown in Fig. 3.15c. A simple example of rotational magnification is a crowbar, which has a handle where the force is applied, a fulcrum, and a reaction point where the crowbar comes into contact with the object to be pried. The crowbar rotates about the fulcrum, which remains stationary. The crowbar tip is located opposite the handle from the pivot, so it moves in the opposite direction. The dis-

tance between the tip and the fulcrum is less than the distance between the handle and the fulcrum, so the applied force is magnified.

Force magnification requires a pivot. There are three kinds of pivots: (1) a fulcrum pivot, which describes a simple contact in a dimension and allows rotation but not translation in the specified direction, (2) a fixed pivot which allows rotation in one or more dimensions but no translation, and (3) an offset pivot which is part of the object's shape definition. The angle on a wedge-shaped object, opposite the location of applied force, is called its *offset*. The upper example in Fig. 3.15c depicts a wedge object, and illustrates *translational magnification* using an offset pivot. The applied force is distributed over a surface, describing a pressure. The offset angle reduces the effective force reaction area, magnifying the pressure. This relationship is shown below (TR-1).

TR-1 AFVAL = A1/A2 * afval (3-1)

where AFVAL is the magnified force, afval is the applied force, A1 is the area where the force is applied, and A2 is the area where the force is reacted. The same relation can be calculated if the offset angle is known (TR-2), using the angle's tangent, or if the relative sizes of the wedge faces are known (TR-3):

TR-2 AFVAL = cotangent(offset) * afval (3-2)

TR-3 AFVAL = size(along-length)/size(along-width) * afval (3-3)

The lower example in Fig. 3.15c depicts a crank, which illustrates *rotational magnification* using a fixed pivot. The crank has a region which is stationary in the applied force dimension, but allows rotation about the applied force direction. Having its pivot located between the applied and reacted force regions results in a differential moment on the crank. Since the moments must sum to zero for static equilibrium (in Applied Mechanics), there is an effective magnification of the applied force at the contact region (C). The degree of magnification, or *mechanical advantage*, is governed by controlling the distance through which the applied force acts. Formally, mechanical advantage is defined as a trade-off between the force applied to a task and the distance (or area) through which the force is applied (TR-4):

TR-4 AFVAL = L1/L2 * afval (3-4)

where L1 and L2 represent the relative distances between the applied force location and the pivot location, and between the pivot location and the reacted force location.

Each of the transformation types can be represented with a TRANSFORM class. The hierarchical relationships between the generic TRANSFORM process, TRANSMIT, and the other TRANSFORM subclasses, TRANSLATE, and MAGNIFY, are depicted in Fig. 3.16. The TRANSFORM classes shown in Fig. 3.16 are distinguished by how the process preconditions are instantiated. An applied force which is coaxial with the contact location is called an *on-axis force*, and distinguishes pure force transmission from force translation (1). Fig. 3.15c (upper) showed the effect of object type (shape) on transmission, which is illustrated in Fig. 3.16 at (2). When the applied force is lateral with the contact location and the object CG, it is called an *off-axis force*. An off-axis force applied to an object without a pivot and a single contact location is called a nudge (as opposed to a prod, which is an axial transmission). The effect of adding multiple contact locations, illustrated in Fig. 3.15b (upper) is shown as a magnification in Fig. 3.16 at (3). This is actually a linear reduction

since the applied force is divided among the number of contact locations.

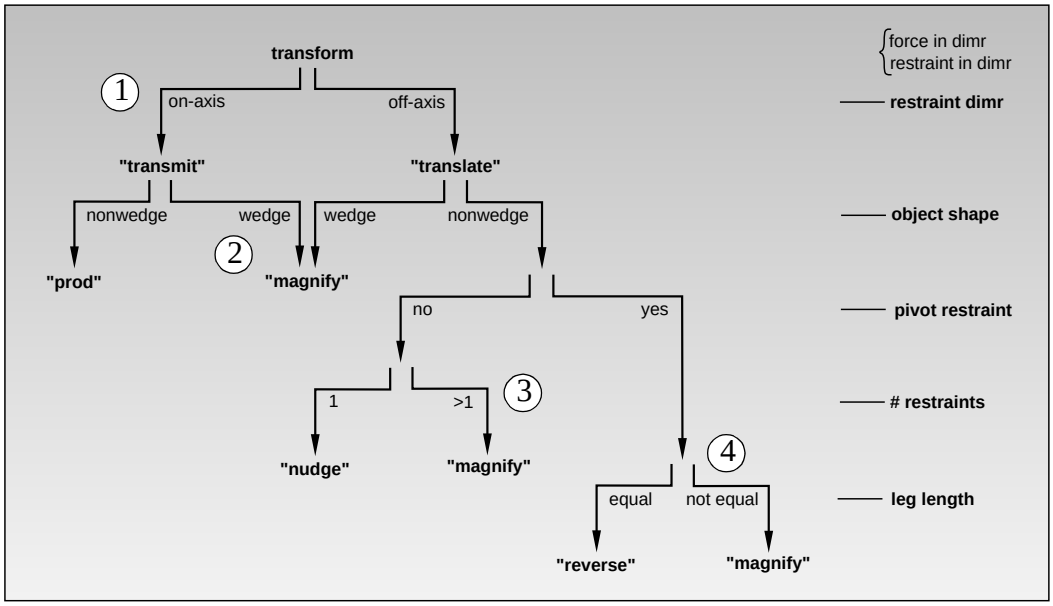


Figure 3.16 Inheritance hierarchy for the TRANSFORM process, specialized by process role instantiations.

Finally, the affect of adding a pivot, which was illustrated in Fig. 3.15b and c (lower), is shown in Fig. 3.16 at (4). The notion of leg length refers to the relative distances mentioned (as L1 and L2) in relations (TR-1) through (TR-4). When the leg lengths are equal, the effect of force transformation is to reverse the direction. When the lengths are unequal, then the result is magnification.

The relationship between applied and reacted forces for these TRANSFORM classes is shown in Table 3.4. The table is organized by the force dimension, direction, and magnitude.

Transform Class	**Applied vs. Reacted Force**		
	Dimension	**Direction**	**Magnitude**
TRANSMIT	Same	Same	Same
TRANSLATE	Same, Different	Same	Same
MAGNIFY	Same, Different	Same	Different

Table 3.4 Specializations to the TRANSFORM behavioral process primitive.

In the table, "same" means that the item is identical for both the applied and reacted forces. For example, the MAGNIFY class can represent forces which have the same, or different, dimensions, but the magnitudes are always be different.

The MAGNIFY subclass is used to represent all forms of force magnification described using equations (TR-1) through (TR-4). The differences between these relations depend on the object physical characteristics, so further discussion of mechanical advantage will be

delayed until the next section. Since the distinction necessary to identify a MAGIFY transformation over a TRANSMIT or TRANSLATE depends only on the type of force and local restraint, MAGNIFY can be represented without knowing much about the objects in question. For example, the MAGNIFY process is shown instantiated for a rotational force magnification in Fig. 3.17. The pivot relationship for the object which will be magnifying the force, O1, must be in the dimension and direction of the applied force. In this example, this relationship cannot be represented with a fulcrum pivot, because translational motion is not constrained in DIM1 POS by a fulcrum. The process represented at (1) must be a fixed pivot.

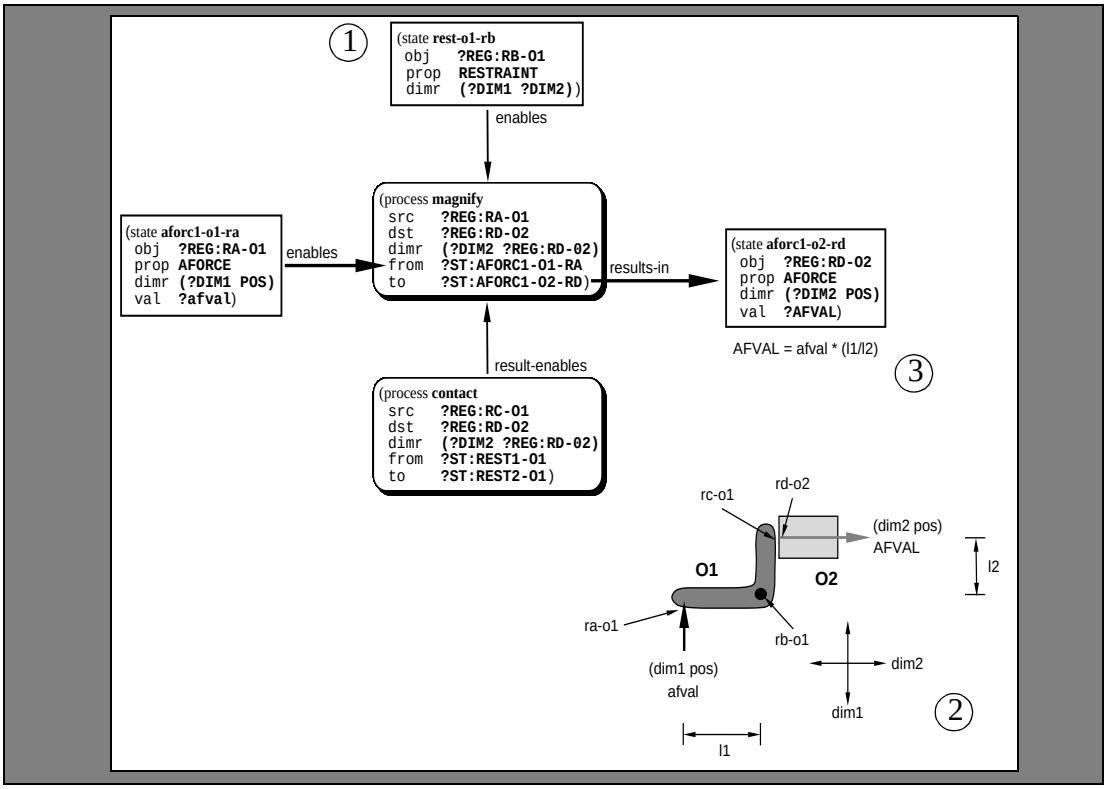


Figure 3.17 Graphical representation of MAGNIFY transform process.

The figure which accompanies this representation (2) depicts the relative distance between the force application location (RA-O1) and the pivot location (RB-O1) as l1, and the relative distance between RB-O1 and the reaction location (RC-O1) as l2. The force magnification produced by this process is represented by the aforce values afval and AFVAL. AFVAL is calculated, if l1 and l2 are known, by the (3-4) relationship illustrated under the reacted force state (3).

3.2.5 STORE

TRANSFORM describes interactions between objects where the object is considered rigid with respect to the magnitude of the applied force. That is, the TRANSFORM only describes the transfer of force from one object to another. The STORE process describes the effect of the new force on the reacted object when it is unable to move, such as compressing a spring between one's fingers.

Each material has a limit, called an *elastic limit*., or *yield strength*, depending on the

units chosen to describe it. When an object is perturbed, and its motion is constrained, the applied force is absorbed internally. This absorption is called internal energy. The mechanical effect of energy absorption is a change in size or shape. This change is called deformation, or strain. Deformation is dependent on the amount of internal energy, so as the applied force increases, so does the deformation. The internal energy used to deform the object is released when the applied force is removed. As long as the applied force doesn't exceed the elastic limit, the object will return to its original size and shape when the force is removed. This is called *elastic deformation*. Internal energy is said to be *stored* in the object because it is the internal energy release which produces the reverse change in object size and shape. In FONM, the process of energy absorption and elastic deformation is distinguished from permanent, or *plastic*, deformation. The STORE process describes interactions which produce temporary deformation and energy absorption in objects.

The STORE process involves a single object and has two behavioral enablements: an applied force whose value doesn't exceed the object's elastic limit (shown at 1 in Fig. 3.18), and a restraint state on the object (2) which constrains its motion in the dimension/direction of the applied force. The enabled STORE process results in two states: an internal force which opposes the applied force (5), and a size in the applied force dimension (6). The change in internal force state is unique to the STORE process, while the change in size is also associated with DEFORM processes. The STORE process (S) organizes these interactions.

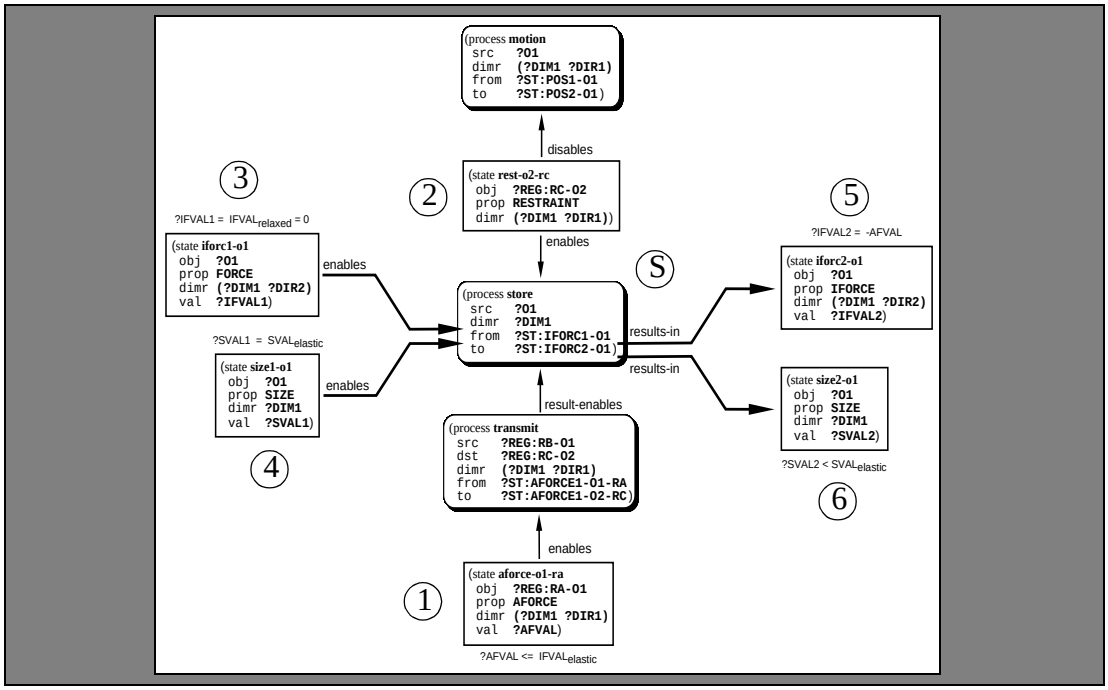


Figure 3.18 Frame representation for the STORE behavioral process primitive.

The IFORCE and SIZE states, labeled(3) and (4) in Fig. 3.18, represent the relaxed object, whose size is equivalent to the resting size for the object, and whose internal force is, subsequently, zero. The internal force resulting from STORE (5) has the same dimension as the applied force, but an opposite direction. The value, IFVAL, is identical to that of the applied force value, AFVAL.

STORE is used to represent how internal energy is stored and released in objects. It

must be able to represent the effects of tension, compression, bending and torsion, which describe energy absorption in objects, as well as relaxation, which represents energy release in objects. The first four of these are illustrated in Fig. 3.19a-d. STORE is used to represent these forms of energy storage by specializing the characteristics of the applied force.

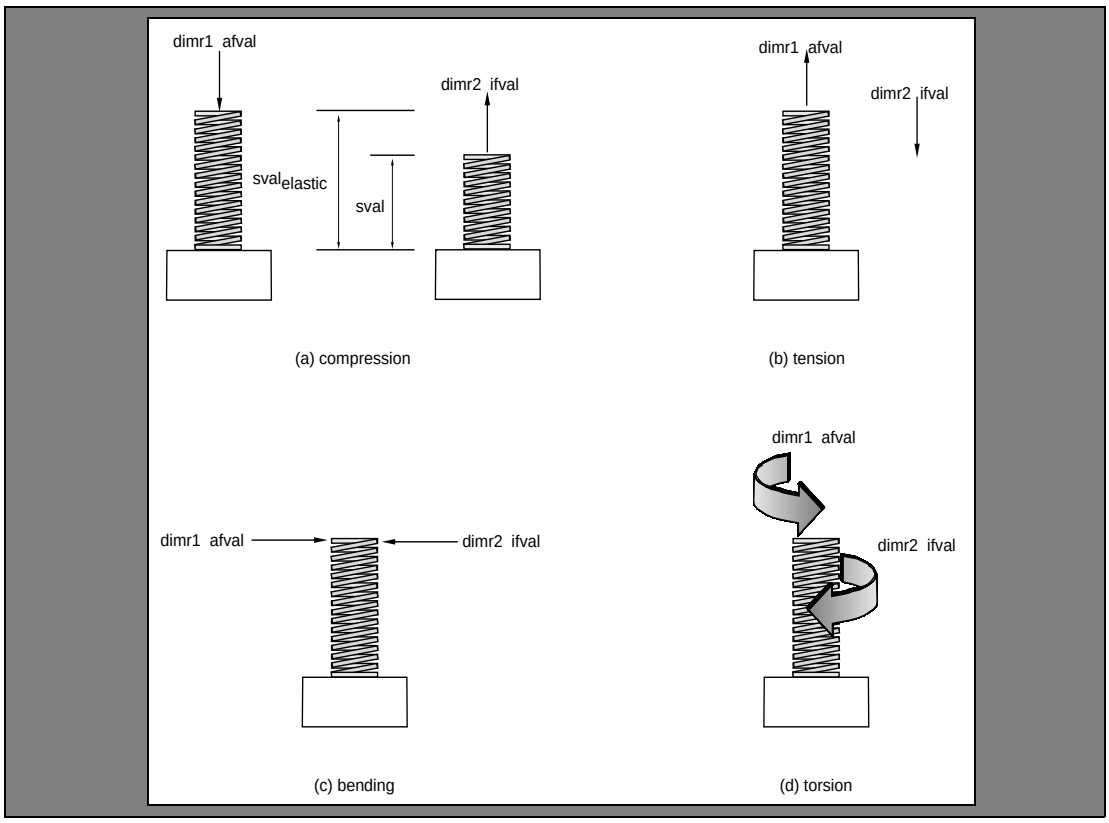


Figure 3.19 Specialization classes of the STORE behavioral process primitive, (a) compression, (b) tension, (c) bending, and (d) torsion.

The springs depicted in Fig. 3.19a, b are depicted with axial perturbations, and depict object compression and tension, respectively. The spring in Fig. 3.19c is perturbed laterally and depicts object bending. The spring in Fig. 3.19d is perturbed rotationally and depicts object torsion. DIMR1 represents the dimension and direction of the applied force, and AFVAL represents its magnitude. DIMR2 represents the dimension and direction of the internal force resulting from STORE, and IFVAL its magnitude. Each example in the figure describes the storage of internal force. When the internal force is initially nonzero and there is no applied force or its value is smaller than that of the internal force, then the object is said to be relaxing and describes the release of stored energy.

The $SVAL_{ELASTIC}$ and SVAL sizes shown in Fig. 3.19a represent the resting length or size of the object. When the object size in a particular dimension is equal to its resting length, then the STORE process doesn't predict any change in object internal energy. This means that the STORE process does not represent potential energy in objects.

The storage processes illustrated on the coil springs in Fig. 3.19a-d are represented as principal subclasses of the STORE process: STORE-COMPRESS, STORE-STRETCH, STORE-BEND, and STORE-TWIST. STORE-BEND and STORE-TWIST can both represent torsional compression or tension, depending on the applied force direction with re-

spect to the object shape.

STORE-COMPRESS represents processes where axial force is applied toward the object CG, as with the coil spring in Fig. 3.19a. The process causes a size reduction with respect to the spring's resting length, and an internal energy, or force, directed away from the spring CG.

STORE-STRETCH represents processes where axial force is applied away from the object CG, as with the coil spring in Fig. 3.19b. The process causes a size increase with respect to the spring's resting length, and an internal energy, or force, directed toward the spring CG.

STORE-BEND represents processes where lateral force is applied away from the object longitudinal axis, as with the coil spring in Fig. 3.19c. The process causes a size increase with respect to the spring's resting length, and an internal energy, or force, directed toward the spring CG.

STORE-TWIST represents processes where rotational force is applied about the object longitudinal axis, as with the coil spring in Fig. 3.19d. The process causes a size increase or decrease with respect to the spring's resting length, and an internal energy, or force, in the opposite direction.

Compression and stretching describe energy *absorption* by an object. When the perturbation force is removed, the object is free to return to its original size by *releasing* the stored energy. Thus the concepts associated with the english terms "tension," "compression," "relaxation," and "torsion" are all represented with STORE. The hierarchy of STORE processes is shown in Fig. 3.20.

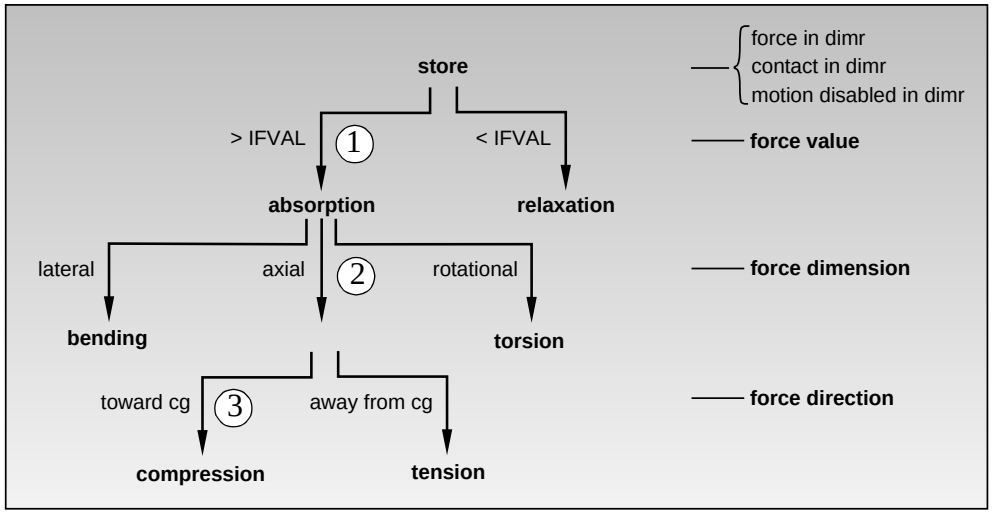


Figure 3.20 Inheritance hierarchy for STORE process, specialized by process role instantiations.

In this figure, the process subclasses are differentiated by how the process enablements are instantiated, specifically by the forcing type. For example, if the applied force is nonzero and greater than or equal to the value of IVAL (1), then energy is absorbed by the object. If the forced dimension is axial (2) and toward the object CG (3), then the object is in com-

pression. These STORE classes are shown with their role instantiations in Fig. 3.5.

Store Class	Force Value	Force Dimension	Force Direction
STRETCH	AFVAL > IFVAL	Axial	(CG NEG)
COMPRESS	AFVAL > IFVAL	Axial	(CG POS)
TWIST	AFVAL > IFVAL	Rotational	Any
BEND	AFVAL > IFVAL	Lateral	Any
RELAX	AFVAL < IFVAL	Any	Any

Table 3.5 Specializations to the STORE process.

The frame representation for the STORE process is shown instantiated for the COMPRESS subclass in Fig. 3.21.

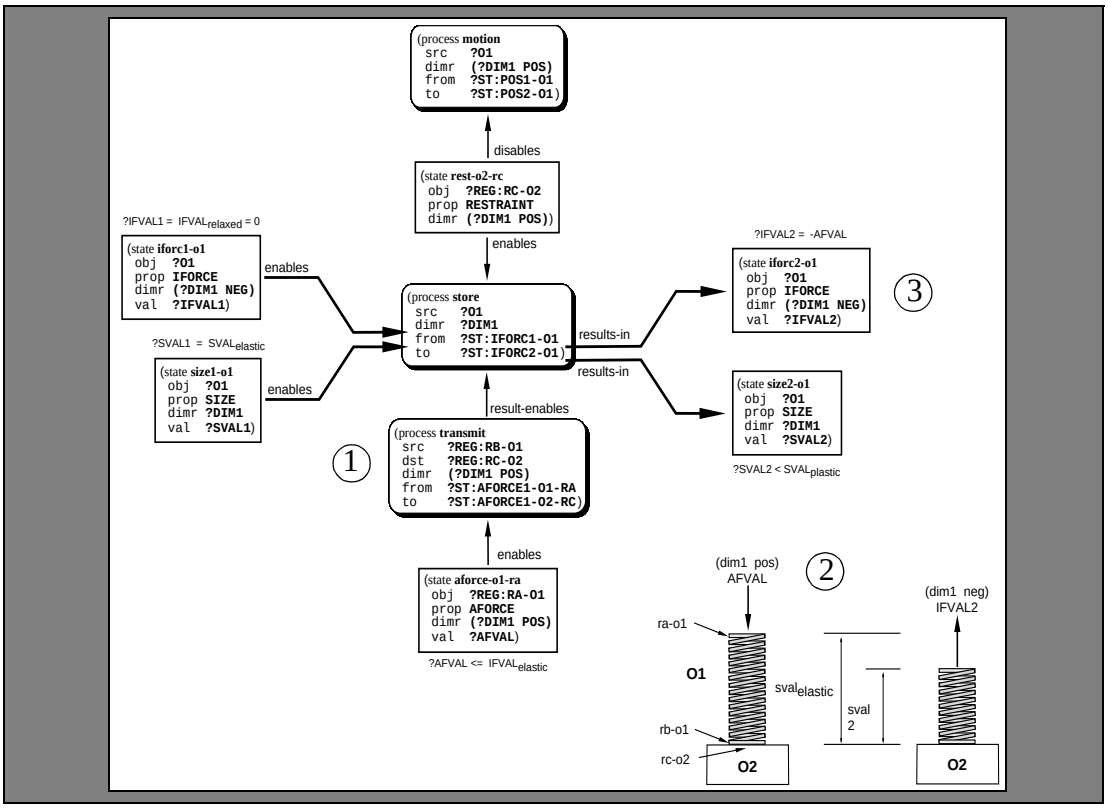


Figure 3.21 Graphical representation for the COMPRESS specialization of the STORE process class.

In the figure, AFORCE-O2-RC (not depicted in the figure) is the force that results from the TRANSMIT process (1) from O1 to the stationary object O2. AFORCE-O1-RA represents the perturbation force, as shown at (2), and is instantiated for dimension/direction DIMR1 and value AFVAL. In this case, the dimension is ALONG-LENGTH, and the direction is POS. The value AFVAL is less than or equal to the material elastic limit, $IFVAL_{ELASTIC}$. The states resulting from the enabled STORE-COMPRESS are IFORC2-O1 and SIZE2-

O1. IFORC2-O1 has dimension/direction DIMR2 which is opposite DIMR1 (ALONG-LENGTH NEG). IFORC2-O1 has a value IFVAL2 which is equal to AFVAL. SIZE2-O1 has the same dimension as DIMR1 and DIMR2, and a value SVAL2 which is less than the object's original size, or *resting length*, $SVAL_{ELASTIC}$.

The size state illustrated in Fig. 3.18 does not illustrate the relationship between the perturbation force value (AFVAL), an object's resistance to deformation (or *spring constant*), and the reacted force (IFVAL). The relationship is shown below in SR-1:

SR-1: IFVAL = K * X (3-5)

where K is the material spring constant and X is the object deformation. Further analysis of spring constants is ignored in MFRUP since the values of K and X are not easily measured, and the effect of applying force to an object is the same regardless of its value.

The relationship between applied force and resting length is represented for object stretching in Fig. 3.22, below. The naming conventions, regions, and dimensions in the figure are identical to those of Fig. 3.19a.

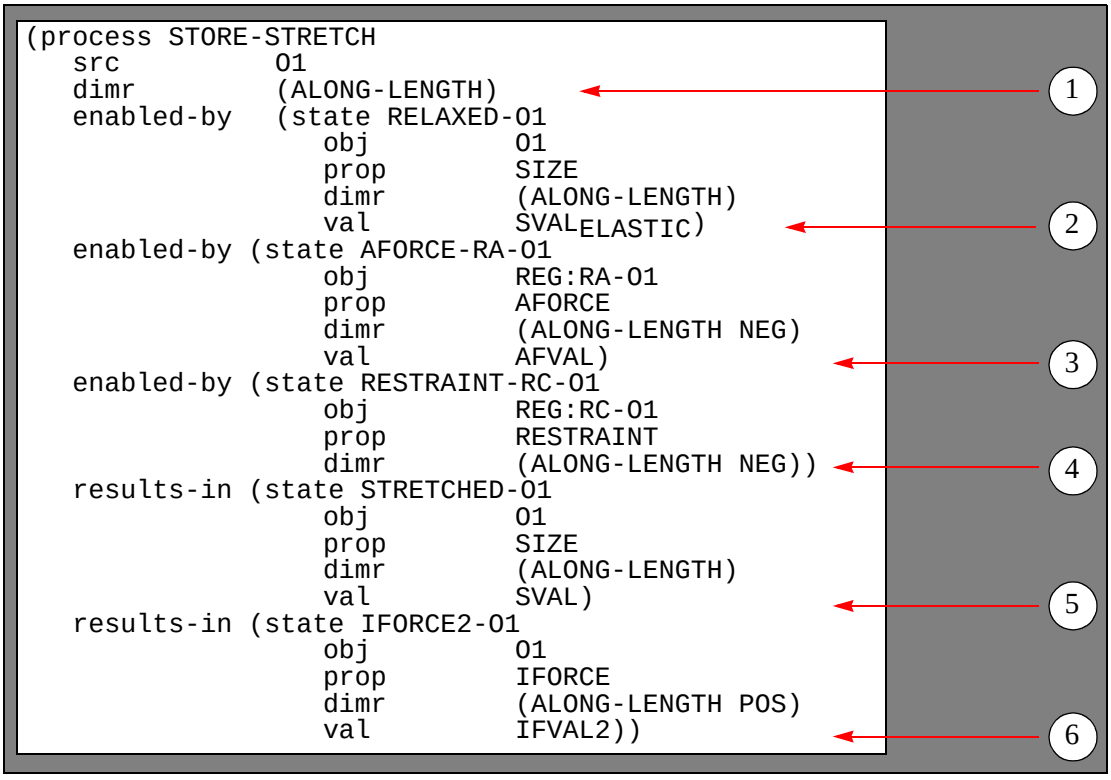


Figure 3.22 The relationship between resting length, $SVAL_{ELASTIC}$, internal force, IFVAL, and length, SVAL.

Fig. 3.22 shows the representation for STORE-STRETCH, which is specified by the dimension and direction of applied force and restraint, (ALONG-LENGTH POS), shown at (3 and 4) which defines the process dimension (1). Not represented are the dependency relationships between STORE-STRETCH and STORE-COMPRESS, which are mutually exclusive, or the dependency relationships between the stretched, compressed, and relaxed bounding states. When an object is stretched its resting length is shorter than its actual length (5), resulting in an internal force (IFVAL2) greater than the prior internal force (6).

3.2.6 DEFORM

Objects undergoing deformations which exceed a material's elastic limit retain some deformation when the applied force is removed. This is called permanent, or plastic, deformation. The process DEFORM represents permanent object size and shape changes, such as those associated with bending, cutting, or breaking. DEFORM can be used represent the size or shape changes when a piece of clay is thrown onto the floor, extruded from a press, or cut with a knife.

The generic DEFORM process involves one or two objects and has two behavioral enablements: an applied force which exceeds the object's elastic limit (shown at label 1 in Fig. 3.23), and an internal force resulting from STORE (2), both in the same dimension/direction. The enabled DEFORM process (D) results in a new size state (3 5). The dimension of the size change is the same as the STORE process and the applied force. The value associated with the size change exceeds the material elastic strain limit, and will be greater than, or less than, the resting length in that dimension, depending on whether the perturbation force is toward, or away from the object center of gravity.

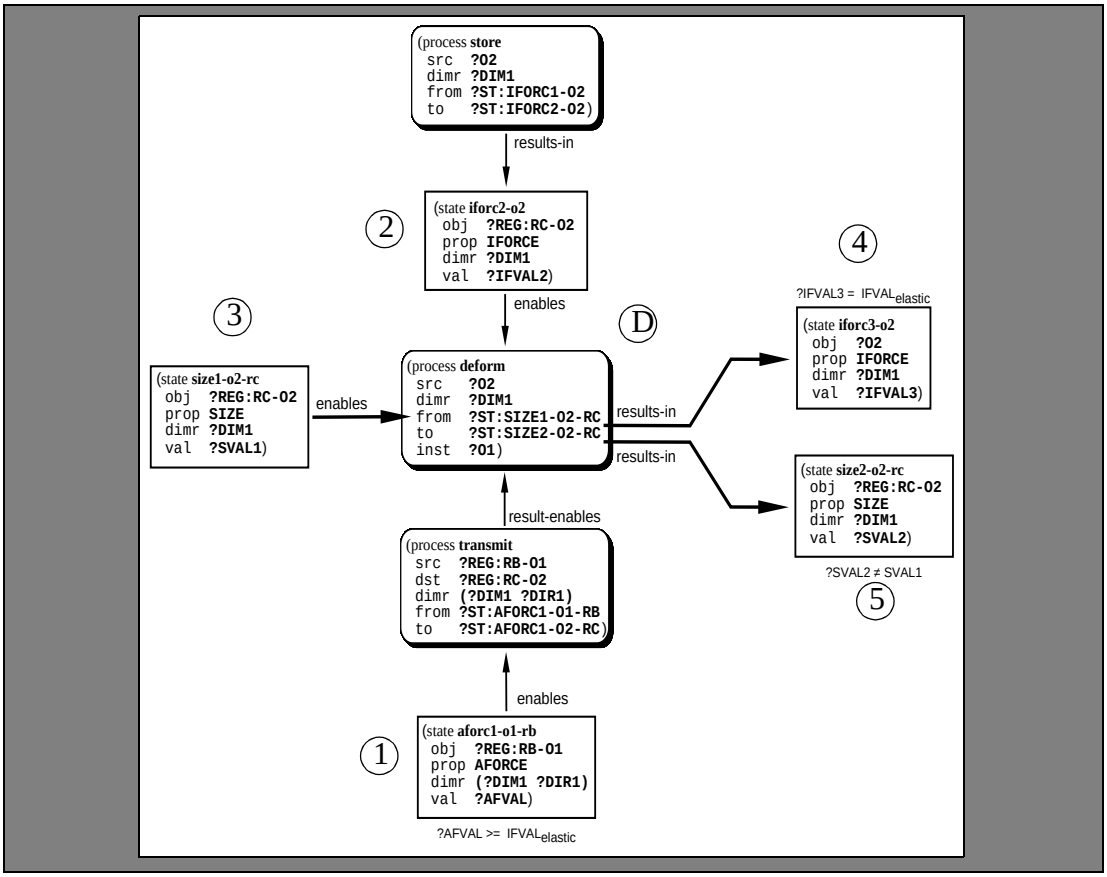


Figure 3.23 Graphical frame representation for the DEFORM behavioral process primitive.

In the figure, the resulting internal force has a value IFVAL3 = $IFVAL_{ELASTIC}$, which means that the internal force has reached its upper bound. When a cut is made, or an object is broken, material is removed from the original object and the internal force is released. The result is a new region, such as an indentation, whose size has the same dimension and value as the deformation. These examples of object deformation describe the kinds of be-

havior which DEFORM is used to represent, as illustrated in Fig. 3.24.

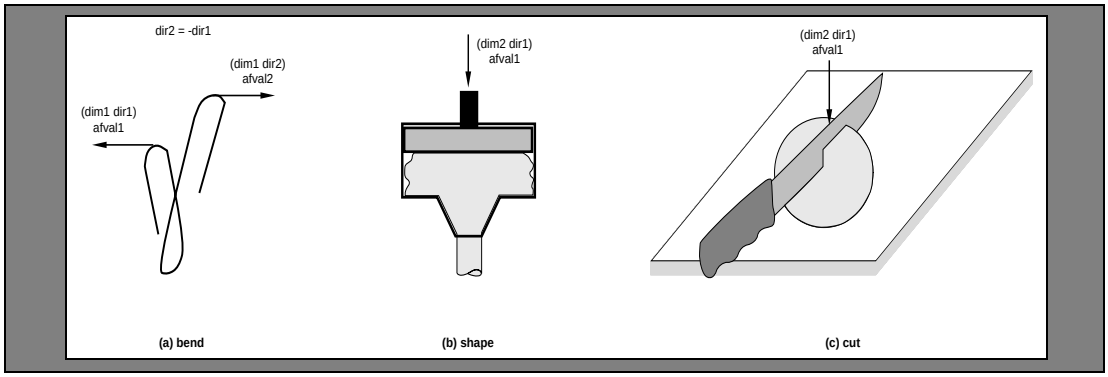


Figure 3.24 Three ways to mechanically deform an object: (a) bend, (b) shape, and (c) cut.

Three kinds of object deformation are shown in Fig. 3.24. The first (Fig. 3.24a) depicts the bending of a paperclip and illustrates a non-instrument deformation which results in a permanent shape change in DIM1, but no observable change in size or volume. Fig. 3.24b depicts the extrusion of some material from a manifold, which affects its shape and size but not its volume. Fig. 3.24c depicts the slicing of a tomato, which involves a knife as instrument and results in a cut in the tomato. The applied force in this instance exceeds the material's *breaking strength*, $\text{IVAL}_{\text{PLASTIC}}$. These three deformation types can be represented with the DEFORM process by prescribing the type and amount of deformation which the object undergoes: DEFORM-BEND, DEFORM-SHAPE, and DEFORM-CUT.

DEFORM-BEND represents object deformations in a single dimension which retain the object continuity. The applied force and restraint are in the same dimension and opposite directions. Bending results in shape changes but not noticeable size changes. For example, the paperclip length illustrated in Fig. 3.24a is increasing a small amount in dimr2 but appears to change a large amount in dimr1.

DEFORM-SHAPE represents object deformations in multiple dimensions which retain the object continuity, resulting in shape changes as well as size changes. The clay being extruded in Fig. 3.24b is undergoing shape and size changes in all dimensions, but is remaining a single piece of clay with the same volume.

DEFORM-CUT represents object deformations which affect object continuity, resulting in shape, volume, size, and, possibly, number changes. The tomato being sliced by the knife in Fig. 3.24c is changing size and shape. When the slice is completed, the tomato will be in two pieces.

The hierarchical relationships between these DEFORM classes are illustrated in Fig.

3.25 as specializations of the generic DEFORM process class.

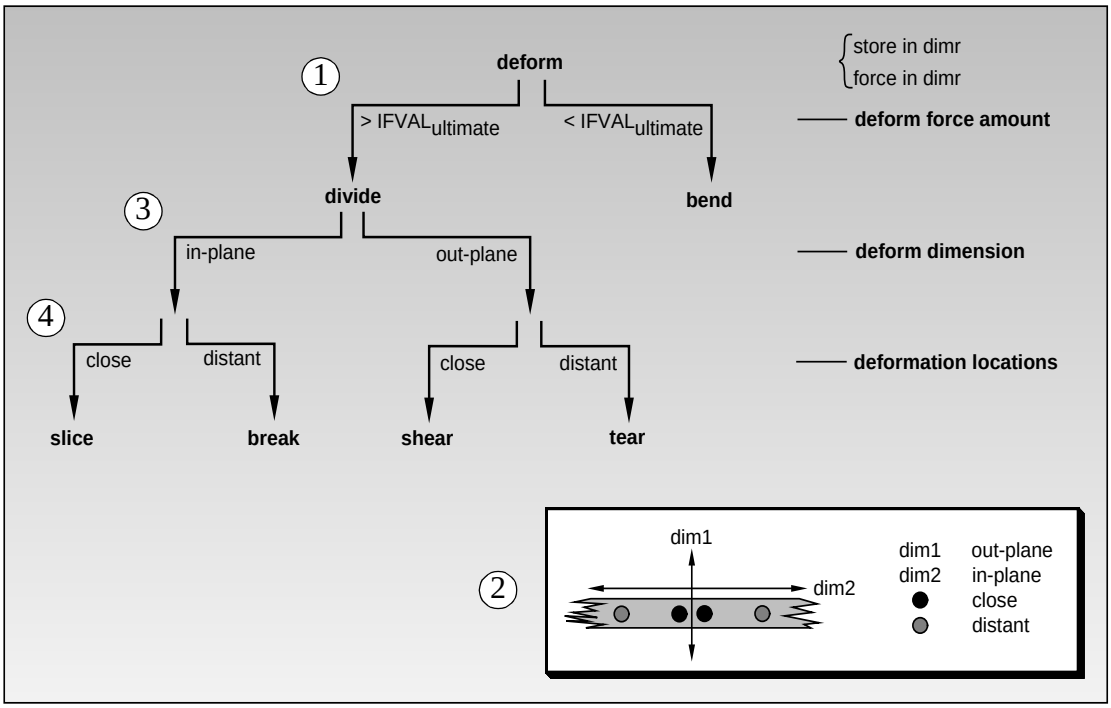


Figure 3.25 Inheritance hierarchy for DEFORM process, specialized by process role instantiations.

Processes in which the deformation exceeds the elastic limit (yield strength) but doesn't reach the plastic limit (breaking strength) are associated with bending. The paperclip bending illustrated in Fig. 3.24a instantiates DEFORM in this way. Other DEFORM instantiations are based on *dividing* material (shown at label 1 in Fig. 3.25). The legend (2) describes how divisions are defined. An *in-plane* division (3) is one in which the two material locations move away from each other in the material plane, such as in slicing and breaking. An *out-plane* division is one in which the two material locations move away from each other out of the material plane, such as in shearing and tearing. The division can be further defined by specifying how close the DEFORM locations are. Divisions where the perturbation force is physically close (4) to the deformed region, such as splitting and shearing, utilize instruments to force and restrain the object. When instruments are used, the resulting surface is *regular* or *smooth*. Divisions where the perturbation force is physically distant from the deformed region, such as tearing and breaking, result in *irregular* or *rough* surfac-

es. These forms of division are illustrated in Fig. 3.26a - d.

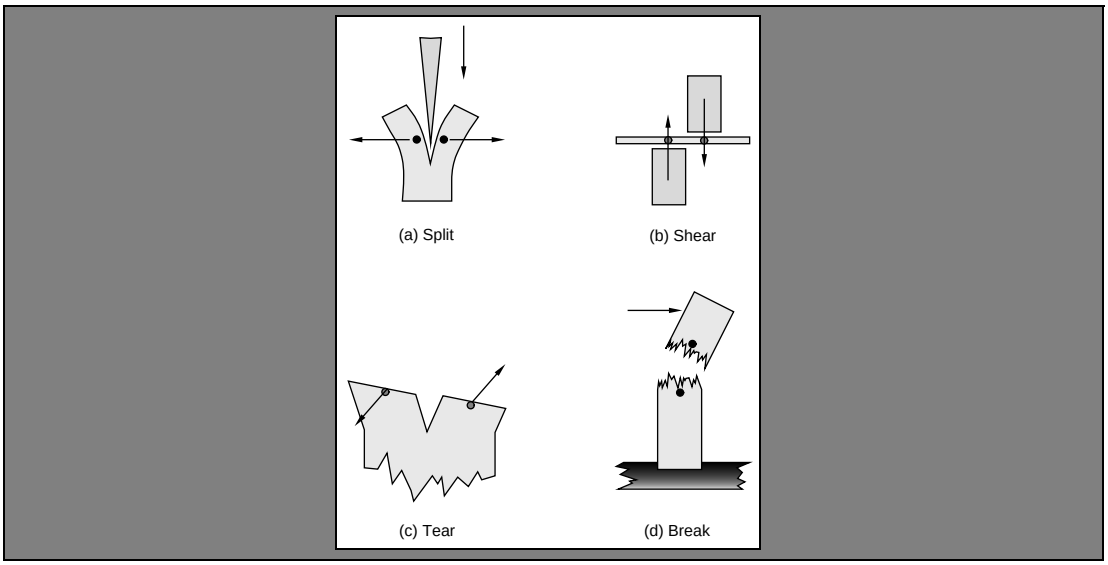


Figure 3.26 Graphical illustration for CUT specializations of the DEFORM process class.

The four division types illustrated in Fig. 3.26 can be represented as specializations on DEFORM-CUT by defining the deformation plane and the locations of restraint and applied force.

DEFORM-SPLIT represents a single wedge-shaped instrument forcing an object apart as in Fig. 3.26a. The translational magnification enabled by the wedge shape causes the force needed to exceed the plastic limit. The wedge faces then force the material [horizontally] away from the advancing wedge. SPLIT can be used to represent slicing and puncturing.

DEFORM-SHEAR represents the deformation caused by an instrument which impinges on another object from opposite directions of the same dimension. The perturbation force is directed toward the object CG, Fig. 3.26b.

DEFORM-TEAR is similar to shearing except that the object being deformed is not locally restrained, resulting in an uneven division. Also, the applied force is directed in opposite directions away from the object CG, Fig. 3.26c.

DEFORM-BREAK represents cuts which occur in lieu of bending. If the paperclip in Fig. 3.26a is replaced by one made of rigid aluminum, then instead of bending it would break, as in Fig. 3.26d.

Each of the DEFORM classes described above can be uniquely represented as a specialization of the process preconditions (behavioral and static), as shown in Table 3.5 below.

Deform Class	Instrument	Deform Dimension	Deform Locations	Length Change	Volume Change	Shape Change
BEND	No	Out-plane	Distant	No	No	Yes
SHAPE	Yes	N/A	N/A	No	Yes	Yes
SLICE	Yes	In-plane	Close	Yes	Yes	No
SHEAR	Yes	Out-plane	Close	Yes	Yes	No
TEAR	No	Out-plane	Distant	Yes	Yes	No
BREAK	No	In-plane	Distant	Yes	Yes	No

Table 3.6 Specializations to the DEFORM process.

The only characteristic missing from Table 3.5 is the applied force magnitude, which is between $IVAL_{ELASTIC}$ and $IVAL_{PLASTIC}$ for bending or shaping, and greater than $IVAL_{PLASTIC}$ for all division DEFORMs.

Using these definitions, the DEFORM representation for tomato slicing is presented in Fig. 3.27. The perturbation force is shown applied at RA-O1, which is a wedge-object in contact with object O2. Object O2 is in turn restrained from motion by O3. The force is magnified and transmitted to RC-O2 using TRANSFORM-MAGNIFY. The resulting force is applied at RC-O2 in dimension DIM1, direction POS, and value $AVAL > IVAL_{PLASTIC}$ (1). The force value and enabled STORE-COMPRESS enable the DEFORM process and identify the subclass as a DEFORM-CUT. The single instrument identifies the division as

DEFORM-SPLIT.

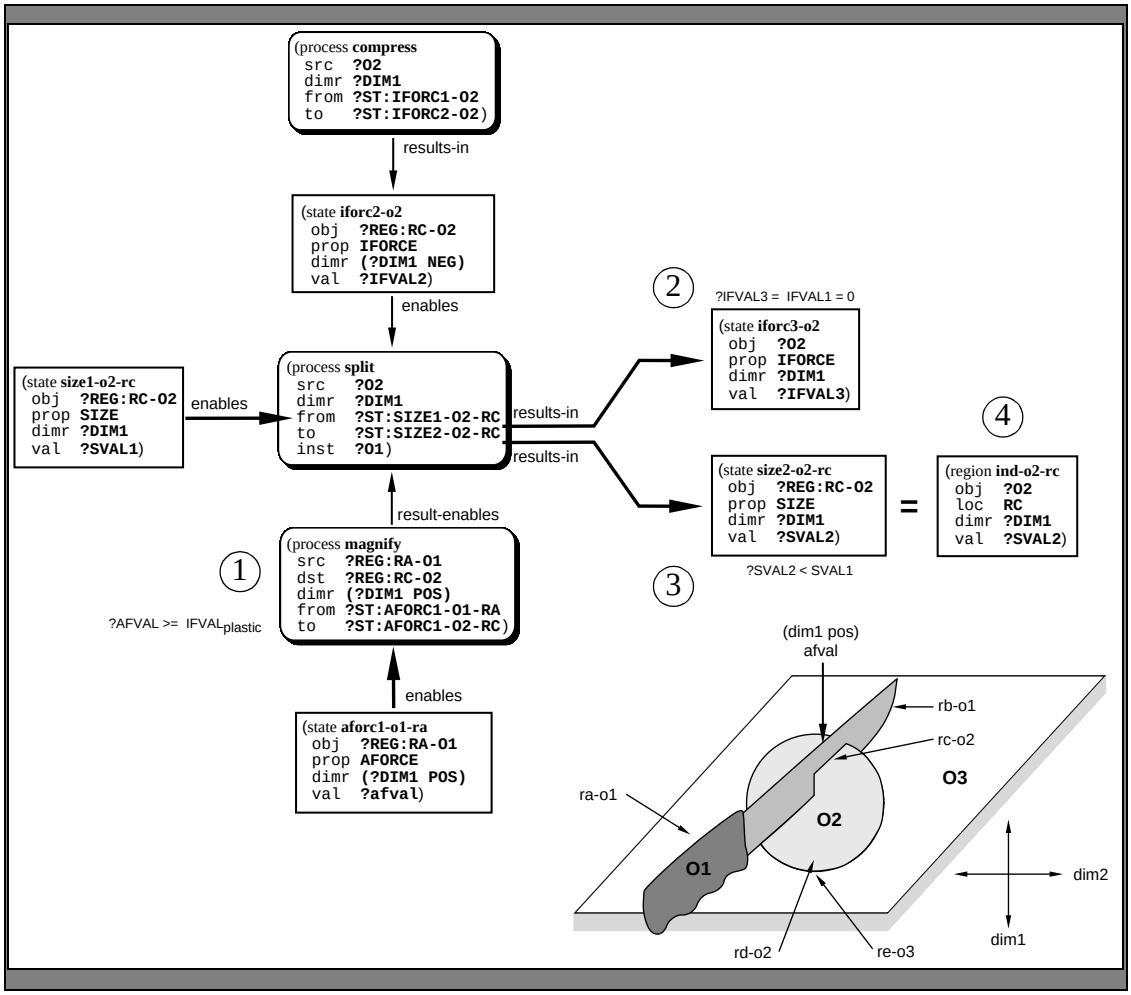


Figure 3.27 Graphical representation for the SLICE specialization of the DEFORM process class.

The states resulting from the enabled DEFORM-SPLIT are IFORC3-O2 (2) and SIZE2-O2 (3). The internal force state IFORC3-O2 has a magnitude of IFVAL3 = 0 because the split releases the force. SIZE2-O2 is defined in dimension DIMR1, and has a magnitude SVAL2 which exceeds the elastic strain limit $SVAL_{ELASTIC}$. The magnitude of SIZE2-O2 is less than the object's original size (SIZE1-O2) in dimension DIMR2. The change in size is attributed to the creation of a region whose size it reflects. In this example, the region is shown at (4).

It should be noted that the process illustrated represents the division of two locations on O1 as a single iteration in the slicing of a tomato and not the entire process, which constitutes a SLICE-APART function. In successive iterations, different locations are separated in DIM1, along DIM2, eventually resulting in separation of surface locations.

Complex cutting processes, such as chopping, clipping, sawing and mowing, can be represented by combining DEFORM specializations in the same way as interference was defined as the combination of RESTRAIN specializations. For example, sawing is a combination of slicing and breaking, where a wedge is forced into an object at an oblique angle and snaps a portion of the material off. Since there is a row of teeth which perform this pro-

cess in unison, a line is sawed in the material. In FONM, the saw's edge is represented as a tooth region, so the DEFORM-CHIP process resulting from the combination of DEFORM-SPLIT and DEFORM-BREAK, represents the behavior.

3.3 Device Behavior Sequences

Combinations of like processes can be used to describe process specializations, as described with RESTRAIN-CONTACT and RESTRAIN-INTERFERE, and illustrated in Fig. 3.13. Device behavior is generally more complex than one process or the combination of instances from one process class. General device behavior can be a complex interaction of different process types, each with its own enabling conditions. Complex object behavior can be represented by combining BPPs in causal sequences. Consider the behavioral sequence associated with cracking pecans open with a nutcracker, illustrated in Fig. 3.28.

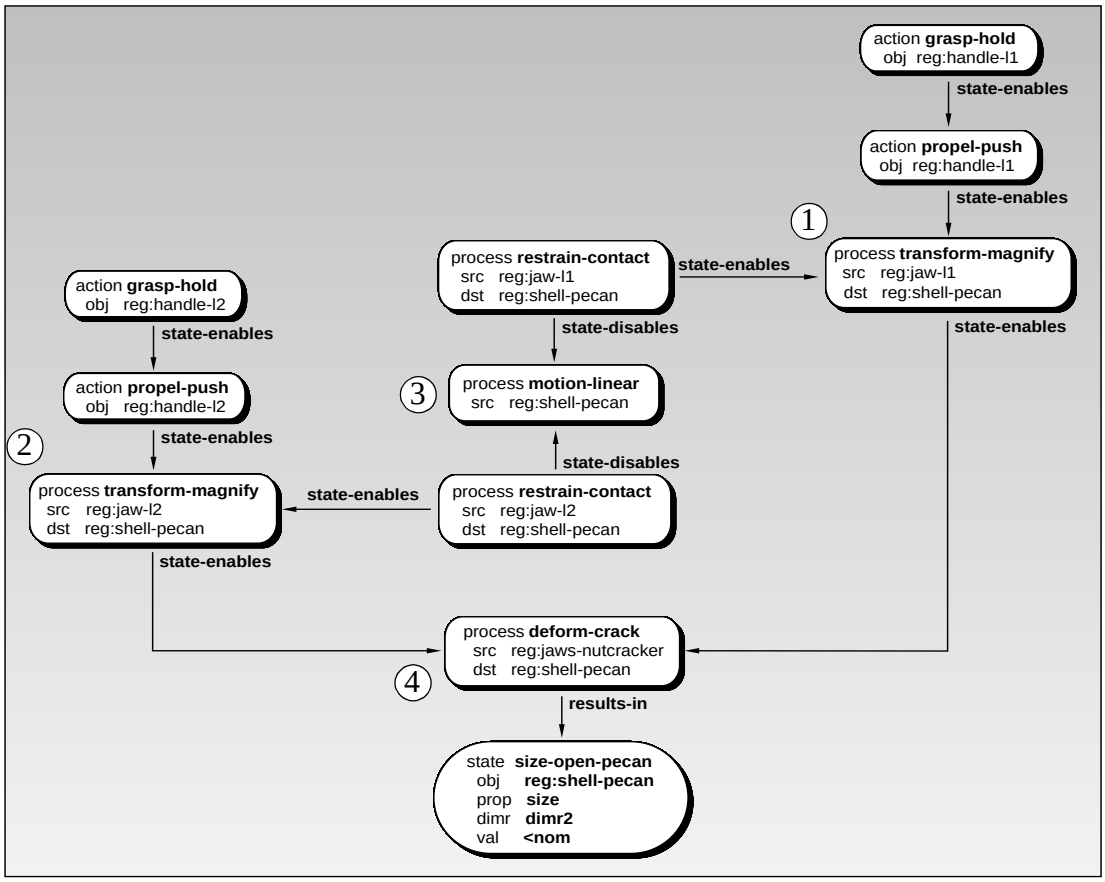


Figure 3.28 BPP sequences for representing nut cracking with a nutcracker

Fig. 3.28 illustrates the actions and processes which are involved in opening a pecan. Only the frame representation of each action or process is depicted, with the assumption that all action or process enablements are met. For example, the applied force states which enable TRANSFORM-MAGNIFY (at labels 1 and 2) are not depicted, but the relation between the PROPEL action, and the process TRANSFORM-MAGNIFY is labeled STATE-ENABLES. In FONM, a PROPEL action on an object causes an applied force on the object. The states are replaced with the STATE-ENABLES link, which is a notational shorthand

for X results-in STATE, STATE enables Y, and is used to reduce the size and visual complexity of the illustration. Similarly, the position states which enable the two RESTRAIN-CONTACT processes, and the restraint states resulting from them have also been omitted. The storage of elastic energy in the nutcracker spring (STORE-COMPRESS) is not illustrated, and no dimensions or values are shown on any of the frames, except for the resulting pecan size state.

According to Fig. 3.28, the nutcracker behaves as follows. Holding and pressing the nutcracker handles are actions which produce an applied force state along the length of their respective handles (HANDLE-L1 and HANDLE-L2). The sequence illustrated assumes that a pecan is positioned between the nutcracker jaws (JAW-L1 and JAW-L2), which enables the two RESTRAIN-CONTACT processes. The pecan is thus constrained from linear motion in both directions, as represented in Fig. 3.13. This motion disablement has been collapsed to the single process MOTION-LINEAR (3) in Fig. 3.28. The applied force on the handles, and the restraint states on the pecan jointly enable force transmission to the pecan shell at the JAW-L1 and JAW-L2 regions. Because the nutcracker handles are pivoted at one end (not shown in this figure), a TRANSFORM-MAGNIFY is enabled, and the force transmitted to the pecan shell is also magnified. If the magnified force exceeds the pecan shell elastic limit, then a DEFORM-CRACK (4) will be enabled. The state caused by DEFORM is a change in size. This state is called the pecan open state, SIZE-OPEN-PECAN.

The BPP sequences which are associated with device function can be collapsed into a single frame by considering only the information required to enable and initiate the function and the states which result from successful function application. One such example is shown in Fig. 3.29, which illustrates two screwdriver functions, *PRY*-OPEN and *POUND*-CLOSED. These functions illustrate the difference in representational granularity between device behavior and device function. The illustration at the bottom of Fig. 3.29 depicts a screwdriver being used to pry open a paint can. Fig. 3.29b represents PRY-OPEN-PCAN as a (low-level) sequence of behavioral processes, and depicts all the processes necessary to understand how the can is opened (similar to Fig. 3.28).

The function PRY-OPEN-PCAN is initiated by holding and pushing the screwdriver handle (region HANDLE-SD) toward the paint can (object PCAN). These actions produce an applied force on the screwdriver. The screwdriver tip (region TIP-SD) is shown in contact (RESTRAIN-CONTACT) with the lip of the can (region LIP-PCAN-LID). The force and restraint states enable transmission (TRANSFORM-TRANSMIT) of the applied force from the tip to the can lid (TIP-SD to LIP-PCAN-LID). The resulting force, at LIP-PCAN-LID, is magnified (TRANSFORM-MAGNIFY) because of the second contact (RESTRAIN-CONTACT) between the screwdriver shaft and the can's lip (BODY-SD and LIP-PCAN-VESSEL) acts as a fulcrum (screwdriver rotates about the pivot, not shown). The force state at LIP-PCAN-LID can, depending on its value, enable the lid (PCAN-LID) to move upward (MOVE-LINEAR). If motion is enabled, then contact (RESTRAIN-CONTACT) between can and lid (PCAN-VESSEL and PCAN-LID) is disabled. The PCAN-LID location resulting from the disabled interference process (between PCAN-VESSEL and PCAN-LID) is the PCAN restraint state, REST-OPEN-PCAN. The overall PRY-OPEN-PCAN function can thus described as a combination of six BPP process specializa-

tions: CONTACT, TRANSMIT, ROTATE, MAGNIFY, LINEAR, and INTERFERE.

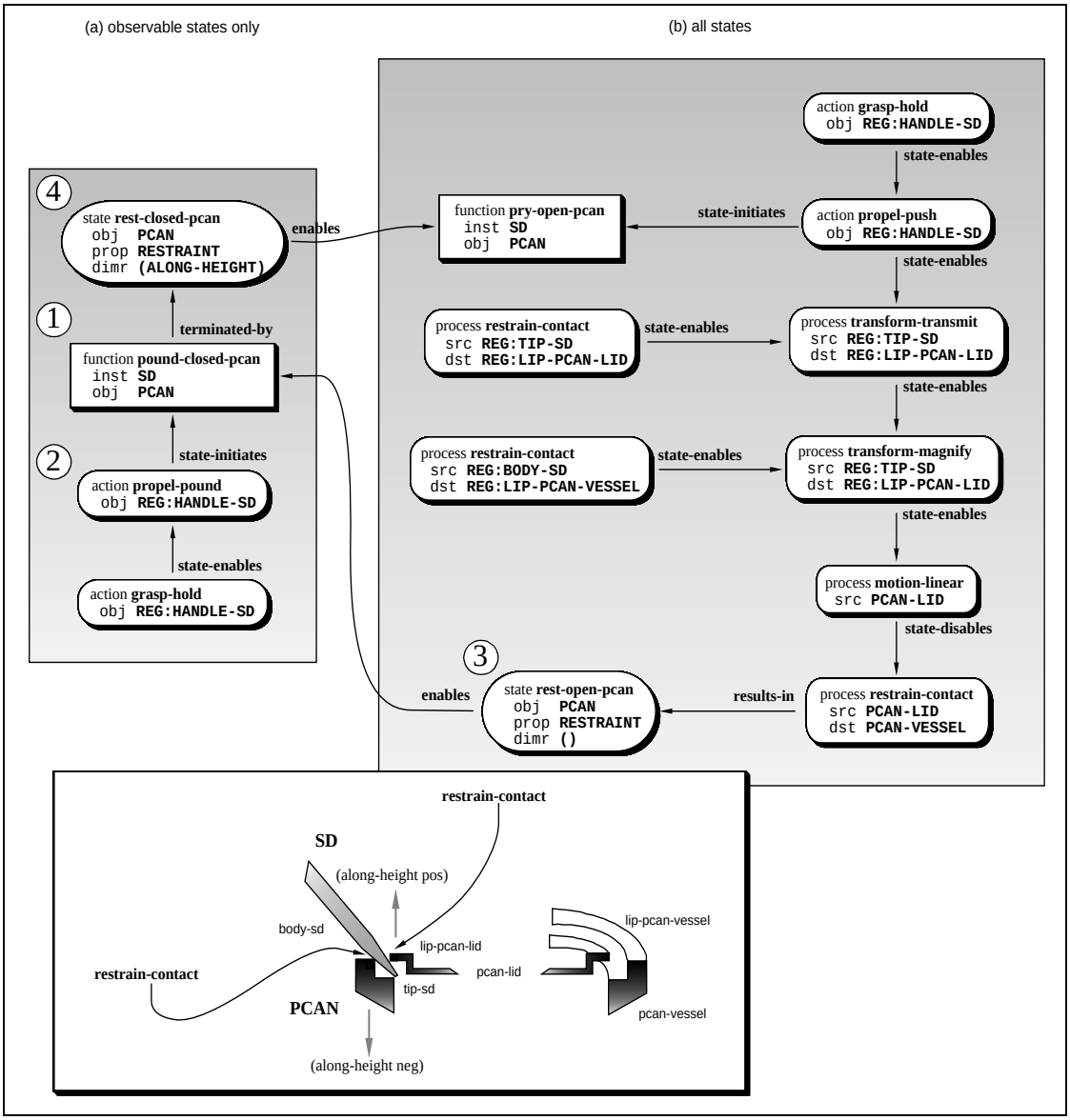


Figure 3.29 Behavioral sequences for representing the screwdriver functions associated with (a) pounding, and (b) prying a paint can lid.

Fig. 3.29a represents a second screwdriver function, that of pounding a paint can lid onto a paint can. The function POUND-CLOSED-PCAN (at label 1) illustrates a frame description for an entire device. POUND-CLOSED-PCAN is initiated by grabbing (GRASP) the screwdriver handle and pushing (PROPEL) the end of the handle (region END-SD) down (at 2) on top of the paint can lid (region TOP-SURFACE-PCAN-LID). POUND-CLOSED-PCAN is enabled by the can being open (state RESTRAINT-OPEN-PCAN, at 3), and by the placement (not shown) of the lid (PCAN-LID) above/onto the can (PCAN-VESSEL). POUND-CLOSED-PCAN is terminated by the state REST-CLOSED-PCAN (at 4) between the lid and the can.

The behavioral sequences shown in Fig. 3.28 and Fig. 3.29 describe general patterns

(functions) which can also be instantiated by other devices. For example, the function CRACK-OPEN using a nutcracker, shown in Fig. 3.28, can also be instantiated by a crank can opener as long as the can opener statics support the function, and as long as the device regions are properly matched. Similarly, a hammer could replace the screwdriver in the POUND-CLOSED-PCAN function of Fig. 3.29a.

The BPP sequences fleshed out in Fig. 3.28 and Fig. 3.29b, and the function frame presented in Fig. 3.29a can be described for any device, however, these representations place more emphasis on how a device behaves and less emphasis on a device's components and the roles they play in how the device behaves. In order to see why a device is constructed a particular way, its overall behavior should be represented in terms of the behavior of its components. Since a component may be used in other devices, recognizing its presence in a device enables its behavior to be inferred and applied as necessary. For example, consider the simple screwdriver shown in Fig. 3.30.

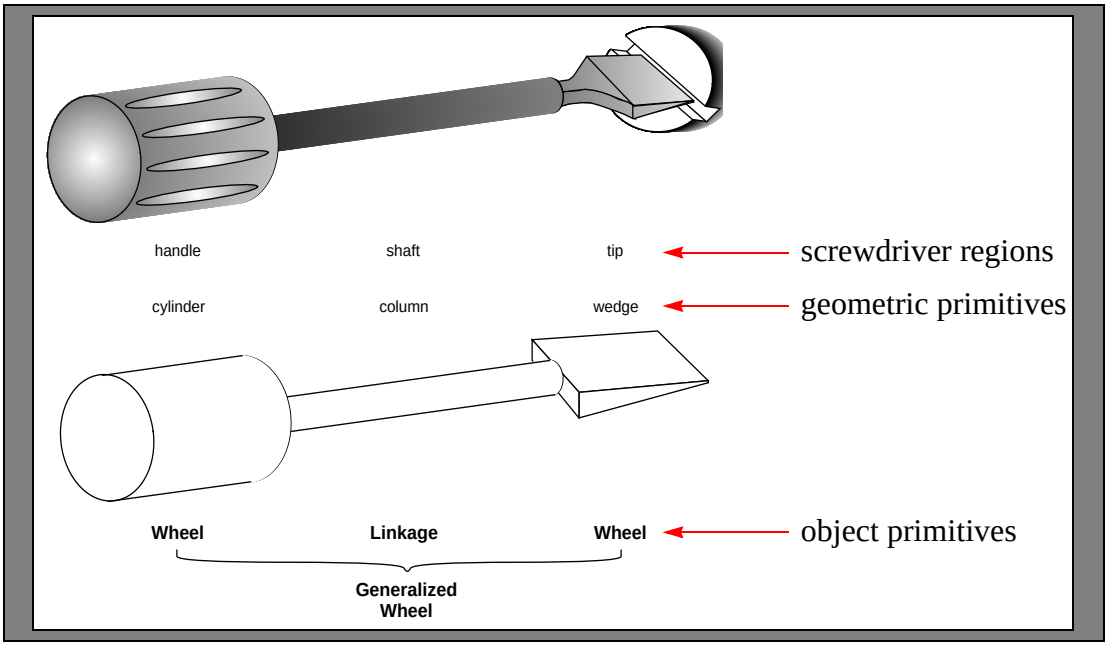


Figure 3.30 Illustration of a screwdriver and its idealized representation for driving screws.

The upper portion of Fig. 3.30 illustrates a screwdriver being used to drive a screw. The screwdriver handle, shaft, and tip components are labeled in the figure. The lower portion of the figure depicts the FONM idealization of the screwdriver as a simple device comprised of three geometric primitives: (1) a cylinder, (2) a column, and (3) a wedge. When the screwdriver handle is turned, as when driving a screw, a rotational force is applied to the handle. Because the handle and shaft are rigidly connected (see Fig. 2.63, Page 84), the rotational force is transmitted radially to the shaft, and, likewise, to the tip. The BPP sequence representations for the screwdriver components, for the driving function, are illus-

trated in Fig. 3.31-Fig. 3.33.

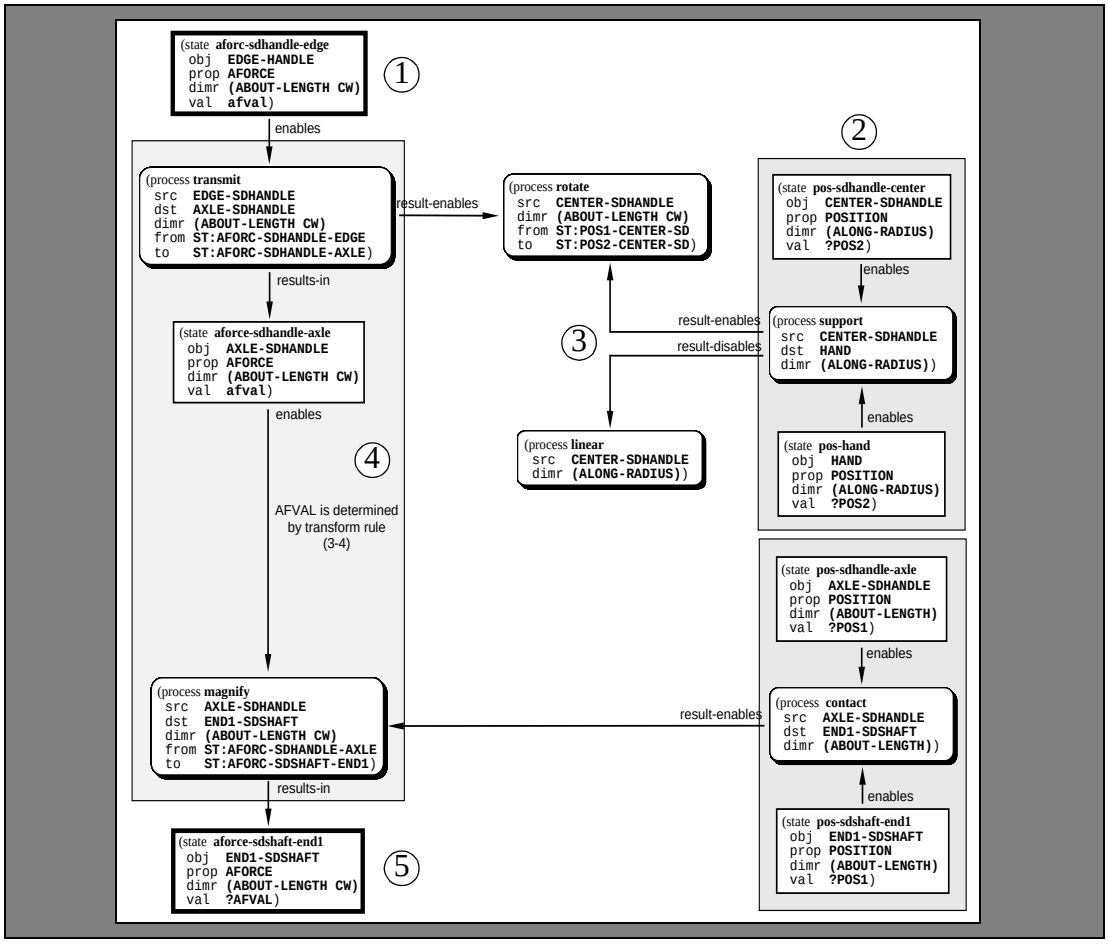


Figure 3.31 Frame representation for the screwdriver handle dynamics when the applied force is torsional.

Fig. 3.31 illustrates the effect of turning a screwdriver handle. The applied force caused by turning the handles is shown at (1). The hand holding the screwdriver enables a SUPPORT process (2), which is illustrated without the ABOVE placement between the screwdriver and the hand. The SUPPORT process provides the pivot which enables MOVE-ROTATE, and the restraint necessary to disable MOVE-LINEAR (3). The shaded box at (4) shows how the interactions between the handle and the shaft are combined into a BPP sequence describing screwdriver handle dynamics given the applied force and restraints. For this example, the handle rotates and the applied force is transmitted and magnified between the outer surface of the handle and the location where the handle and shaft are connected. The value of the force magnitude (AFVAL) resulting from this interaction is determined using transform relation (3.4). A similar set of processes is presented for both the screwdriver

shaft and tip.

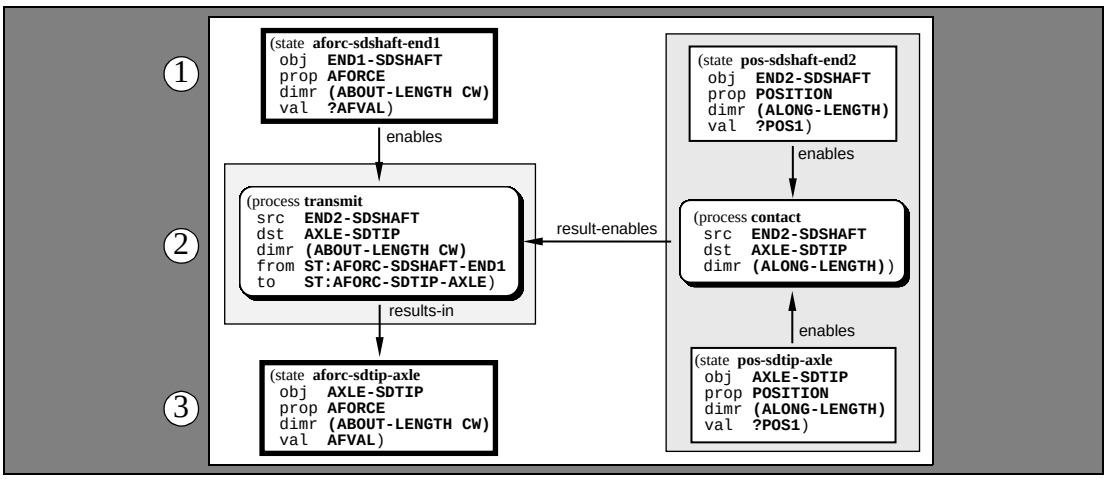


Figure 3.32 Frame representation for the screwdriver shaft dynamics when the applied force is torsional.

Fig. 3.32 illustrates the role the screwdriver shaft plays in the driving of screws. The shaft takes the force magnified by the handle and transmits it to the screwdriver tip. The state labeled (1) is identical to the state labeled (5) in Fig. 3.31. The shaded box (2) represents the BPP sequence associated with the screwdriver shaft. The single process is indicative of a very simple behavior, since the torsional force is only being relocated some distance away from where it was originally applied. Fig. 3.33 shows the role the screwdriver tip plays in the driving of screws. The BPP sequence begins (1) with the state labeled (3) in Fig. 3.32. The interactions shown are identical to those shown in Fig. 3.31, except that instead of a torsional force being applied at an outer radius and being transmitted to an inner radius, the reverse is occurring. The support relation still enables rotation, and the contact between the tip and the screw enables force transmission. The shaded box (4) represents the screwdriver

tip dynamics, but the pattern is virtually identical to that for the screwdriver handle.

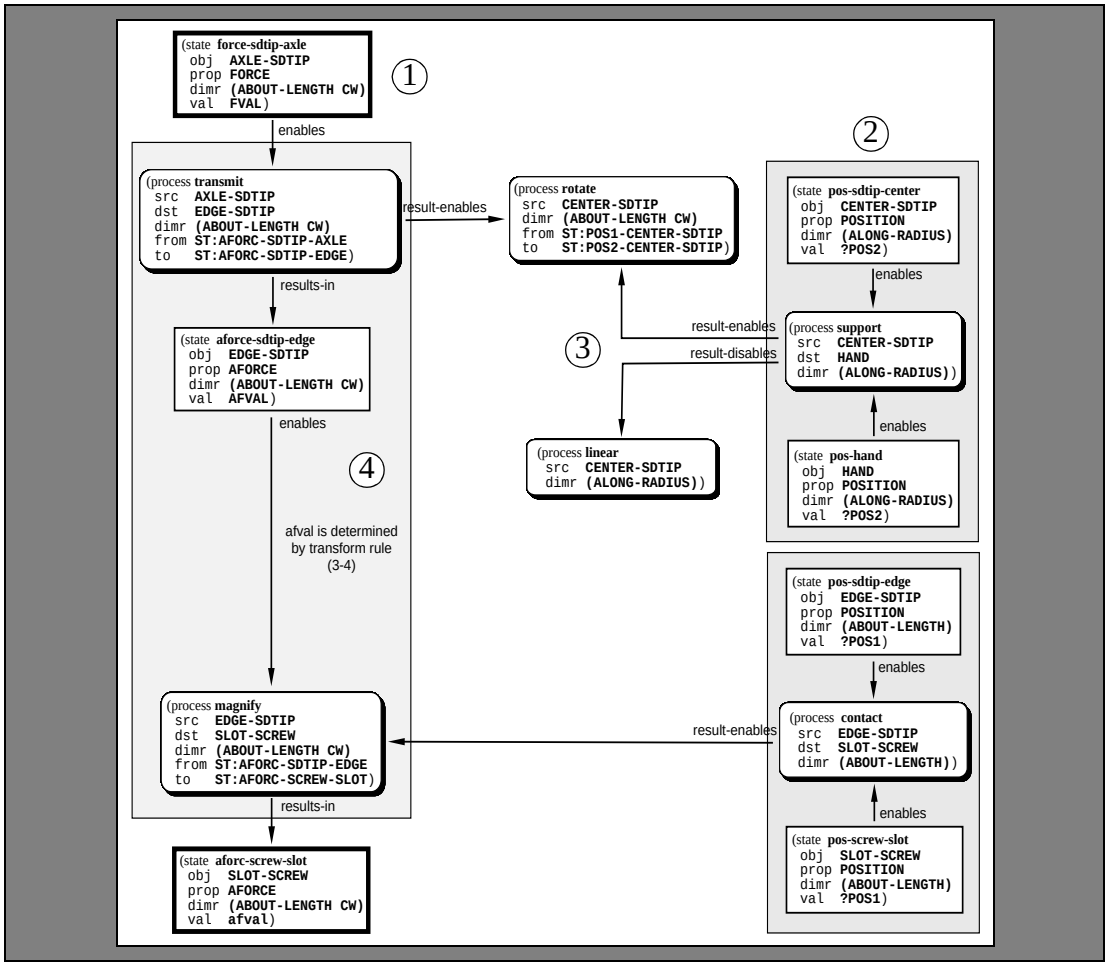


Figure 3.33 Frame representation for the screwdriver tip dynamics when the applied force is torsional.

The BPP sequences for the screwdriver components in these figures are dependent on the component statics and on the applied forces on the components, so they represent general patterns which can be used to describe the role of these components in other devices similarly configured. The shaded sequences on the left hand side of these figures represent component functions in FONM, while those on the right represent the supporting connectivity between components.

3.4 Low-Level Device Representations

This section presents the low-level dynamic representations of the 6 devices whose static and low-level dynamic representations were presented at the end of Chapter 2. The four device types presented: (1) simple devices, (2) simple compound devices, (3) multiple compound devices, and (3) complex devices are revisited with an emphasis on their low-level dynamics. The screwdriver, for example, is a simple device. Statically it is simple because all of its components are rigidly connected. Behaviorally it is simple because all of its components move together, so the behavior of the device is the same as the behavior the most specialized component.

Only the behavior for the intended function of a device is presented in the following sections. The value of these diagrams is in showing the complexity of mechanical device behavior while showing the similarities between devices provided by FONM behavioral primitives. The device representations will take the form of the first example in this section. An idealized illustration of a possible device geometry, based on FONM primitive objects, will be shown, along with the static representation diagram for the primitives. These will be followed by the behavioral sequence diagrams for each component of the idealized device pictured. Four caveats should be noted before presenting the devices:

Components of a device's static representation will only be presented when they directly affect the component's behavior. As such visual cues such as orientation and placement, which comprise a device's static representation, will be noticeably missing in these representations. In this discussion it is assumed that the FONM static representations resolve these issues. For example, the linkage-blade combinations in the fingernail clipper show a force being transmitted from along-depth of the linkage-object to along-length of the blade-object. This is made possible by the static structure, placement, which orients the axes of device components. In this device, the placement of the linkage-object and blade-object specifies that the depth dimension of the former is equivalent to the length dimension of the latter.

Second, there will also be an incomplete correspondence between the connection type illustrated in device SDDs and the associated RESTRAIN process in the behavioral sequence. This is for space considerations only. The dimension specified in the behavioral sequence will identify the most obvious dimension of the associated RESTRAIN process.

Third, some devices have specialized regions which are not depicted in the behavioral sequence. The most simplified version of the device is presented for space considerations, but alternatives will be specifically mentioned

Finally, the values associated with device behavioral states (position, aforce, iforce, and size) are left as variables throughout. Where appropriate, FONM behavioral rules which are used to specify values are noted, but otherwise value is assumed to be part of an actual simulation.

3.4.1 Simple Devices

The low-level dynamic representations of two simple devices: (1) a bottle opener, and (2) a screwdriver, are presented in this section. Each device is comprised of components which are rigidly connected, or regions of a single component. The low-level dynamic representations for three additional devices: (3) a carving knife, (4) a shovel, and (5) a spoon are presented in Appendix A.1.

Bottle Opener

A bottle opener is designed to open cans using one end and to open bottle caps at the other. Both ends make use of mechanical advantage, but one end is pointed so the magnified force is used to puncture. An idealized bottle opener is represented as two lever-objects and a

blade-object, as shown in Fig. 3.34 (redrawn from Figs. 2.58, 2.59) below.

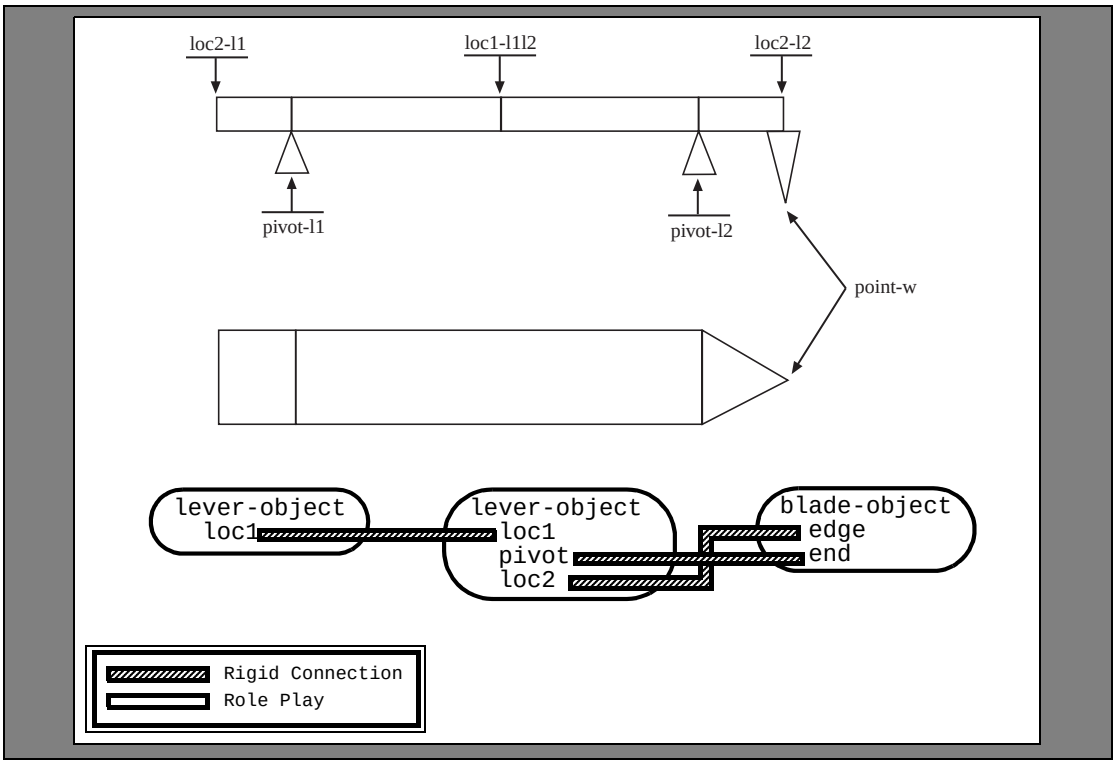


Figure 3.34 Bottle opener object primitives and static device diagram

The low-level dynamic representation for the bottle opener depicted in Fig. 3.34 is shown fleshed out in Figs. 3.35 - 3.37, one diagram for each component's behavior in the can-opening function. Fig. 3.35 illustrates the behavior of the handle (i.e., the blunt end in can opening). The effect of holding the can opener is to apply a force to the end of the device. Since the handle is represented with a lever-object that degenerates to a linkage-object, the

force which initiates the behavior is shown at (1). The function of the handle is simply to

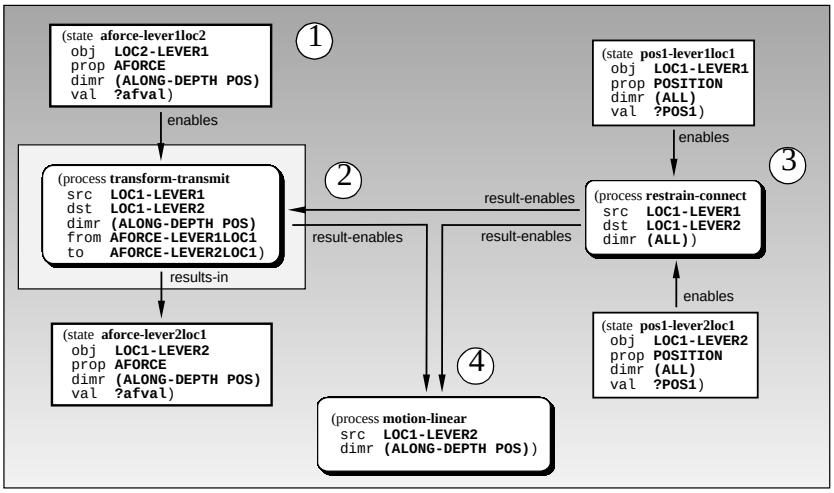


Figure 3.35 Behavioral sequence for bottle opener handle in the can-opening function.

transmit the lateral force to the other end (LOC1-LEVER2, at 2), which is enabled by the rigid connection between the two components (3). The transmission of force and the connection both enable linear motion of LEVER2 (4). The representation for LEVER2 behavior is illustrated in Fig. 3.36. LEVER2 transmits force to the can lip and to the upper lid. In so doing mechanical advantage is generated in the blade end of the can opener. Since the representation makes the blade-object an explicit component, the magnified force is transmitted to the end of the blade-object. The behavior is enabled by the applied force at PIVOT-LEVER2. In FONM, since no internal effects are represented, any applied force which is effected at one location is transmitted to other locations. The applied force which is transmitted from the handle to LOC1-LEVER2 is thus transmitted directly to PIVOT-LEVER2 (1). The pivot contact between PIVOT-LEVER2 and the substrate region2 (the can lip) en-

ables the transmission of force to the can lip. It also enables rotation since the contact point

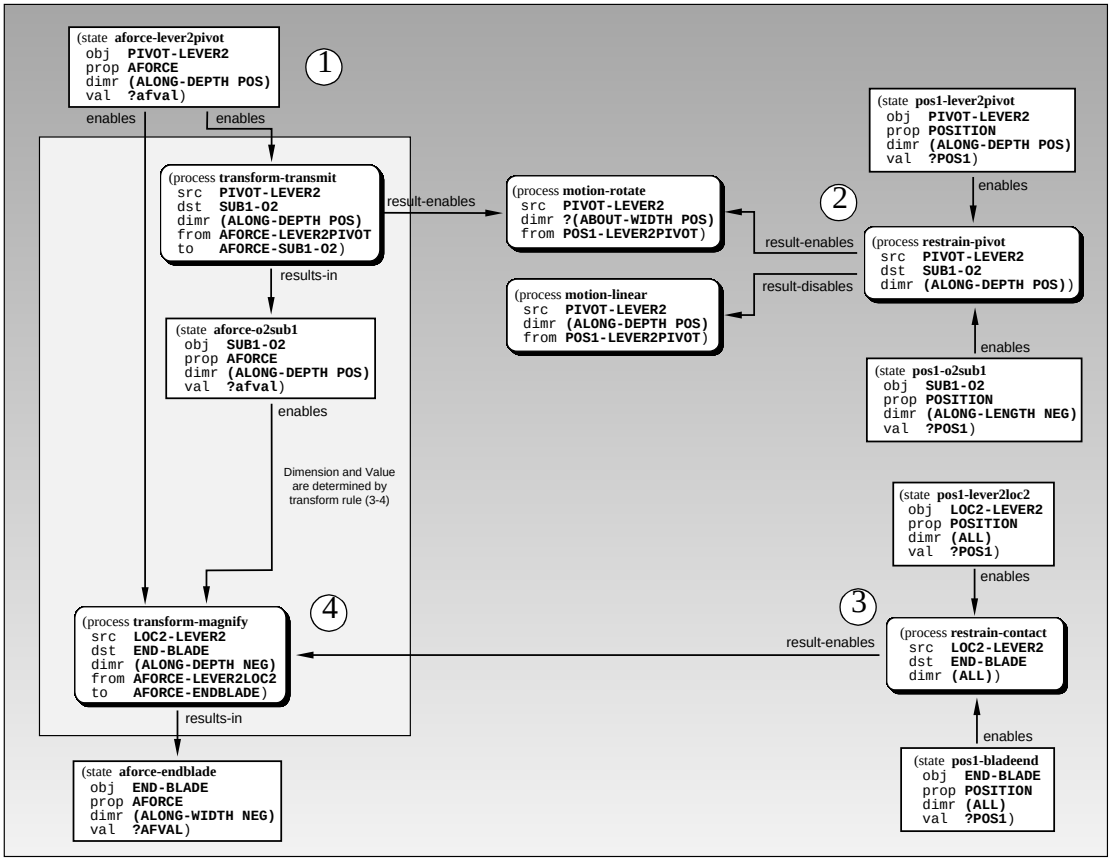


Figure 3.36 Behavioral sequence for bottle opener lever in the can-opening function.

is below the substrate region (2). The connection between LOC2-LEVER2 and the end of the blade-object (END-BLADE, at 3) enables the magnification of force to the blade according to the rules of mechanical advantage (3-4 in particular). The behavior for the blade is

shown in Fig. 3.37, starting with the applied force at the end of the blade-object (1).

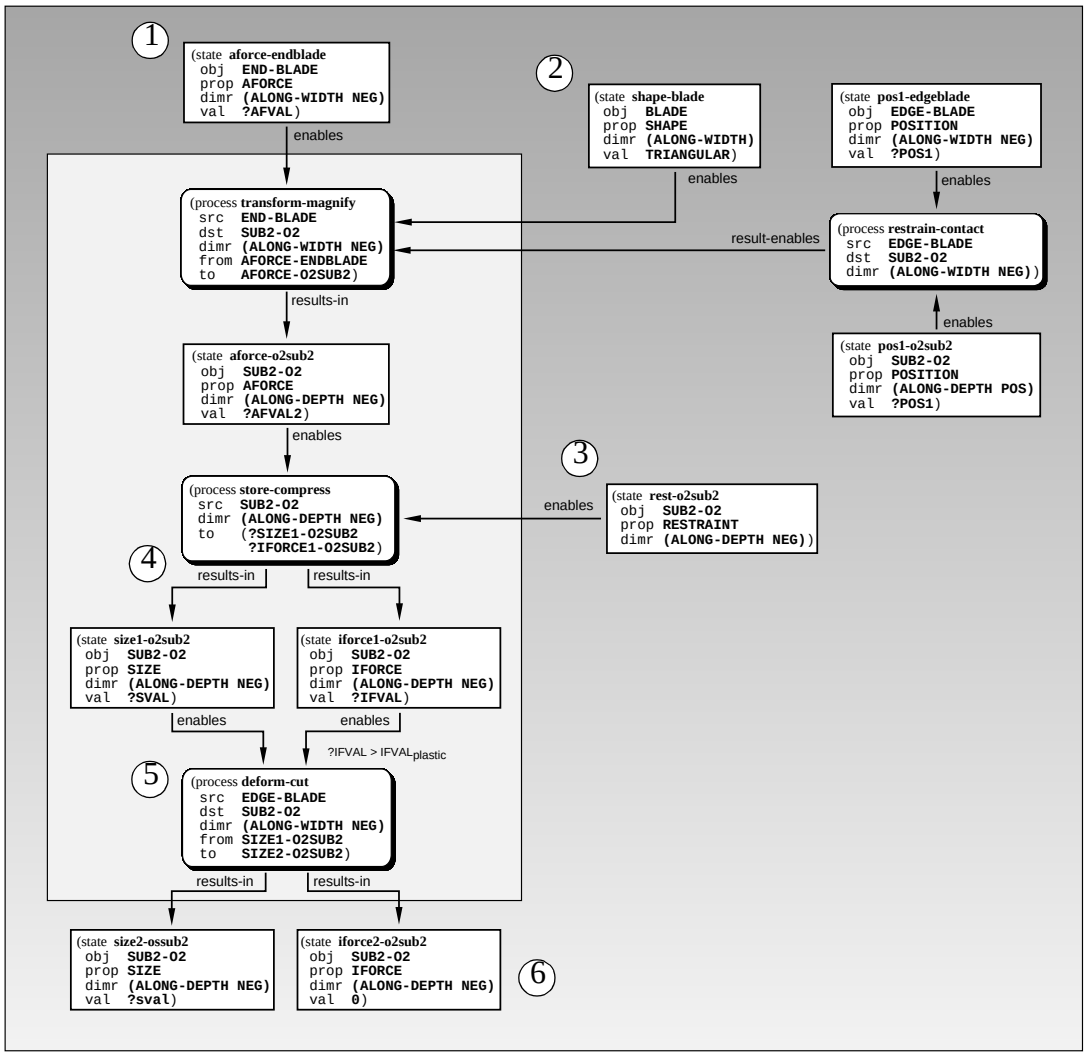


Figure 3.37 Behavioral sequence for bottle opener blade in the can-opening function.

Linear magnification is effected in the blade-object as a result of its triangular shape (2), its contact with the can lid, and the applied force at its end. It is assumed in this figure that the can is restrained along its length (3), although the representation of such would be a part of a pragmatic description of use. The combined result is a compression of the lid (SUB2-O2 at 4). Assuming that the force exceeds the plastic limit of the can lid ($IFVAL_{plastic}$), the lid is cut (5), which changes its size and releases the stored energy (6).

Screwdriver

The idealized screwdriver and its low-level dynamic representations are shown in Figs. 3.30 - 3.33. The idealize screwdriver and its static device diagram are depicted in Fig. 3.38

for continuity and will not be discussed further in this section.

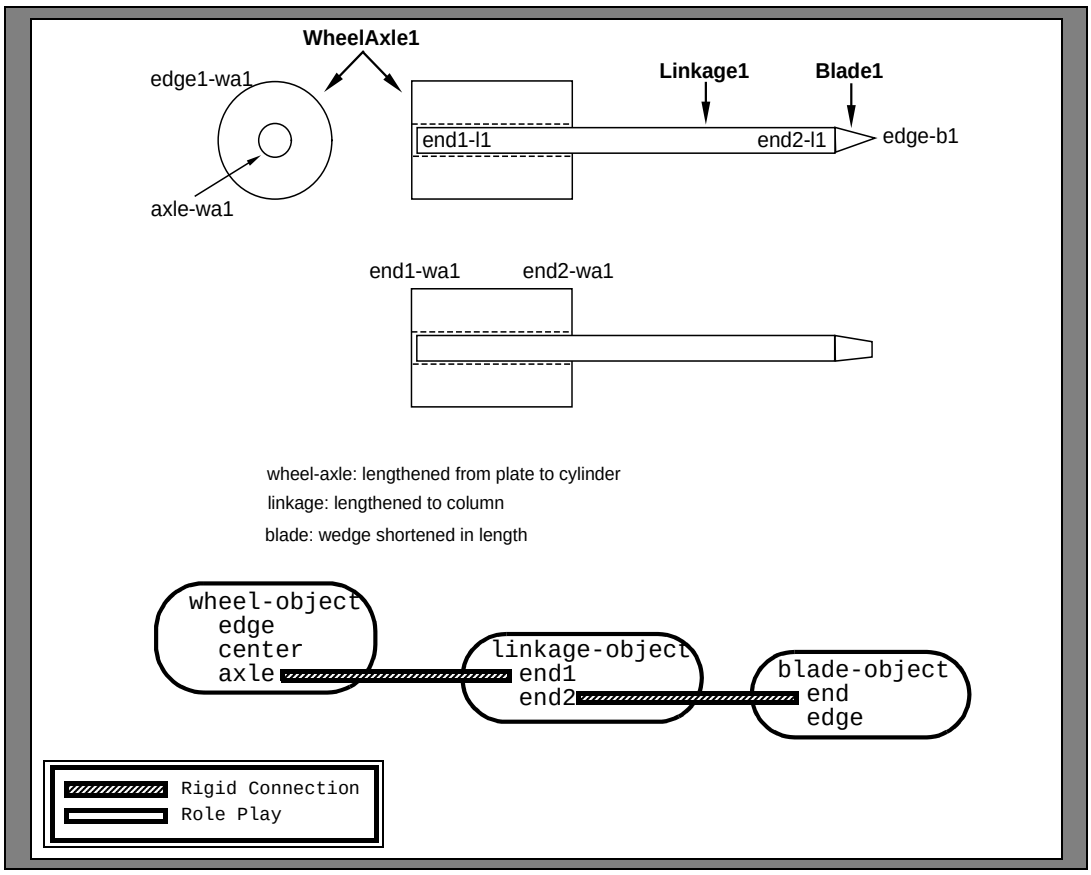


Figure 3.38 Screwdriver object primitives and static device diagram.

3.4.2 Simple Compound Devices

Devices in which relative motion between components is possible are *compound* devices. Compound device dynamics are representationally significant because components can behave independently. The behavioral representations of simple compound devices show striking similarities, because the types of relative motions for simple devices is limited. In this section, two devices: (1) a nutcracker, and (2) a clothes pin are presented.

Nutcracker

The intended function of a nutcracker is to restrain the motion of and crack the shell of a

nut. The idealized nutcracker and its static device diagram are illustrated in Fig. 3.39.

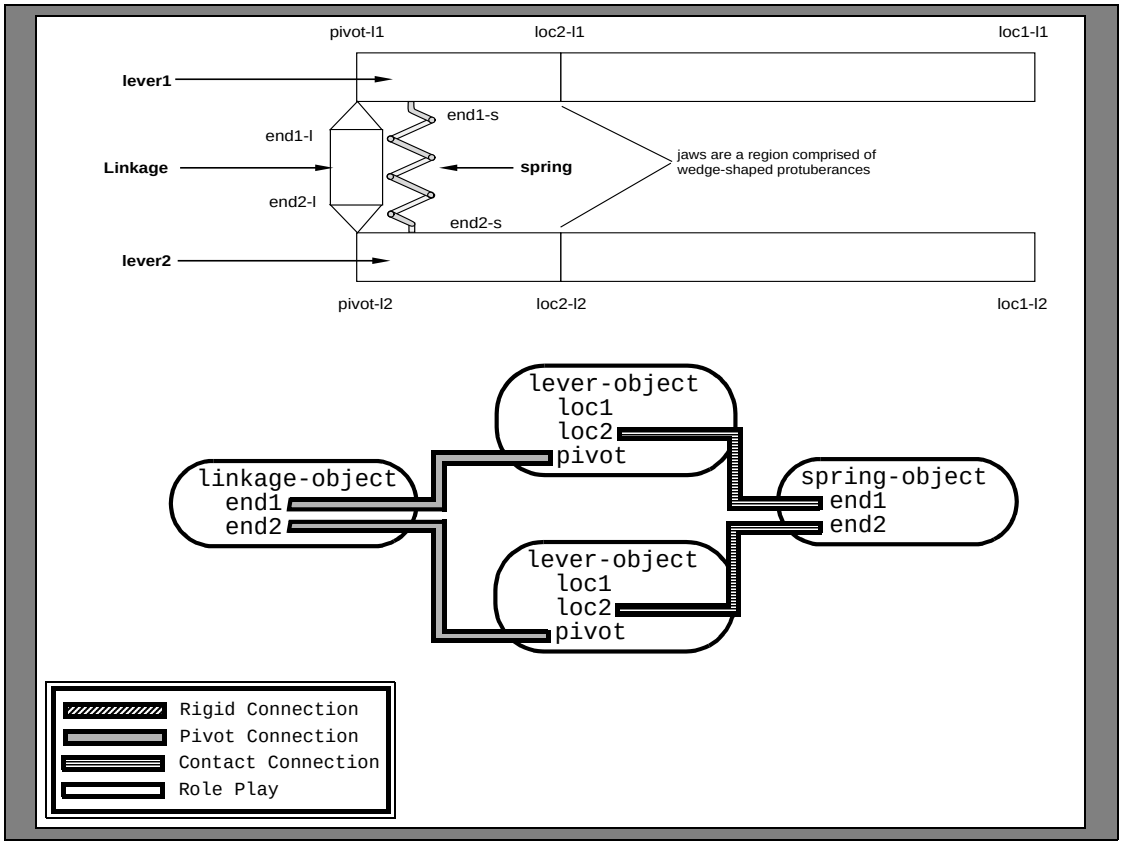


Figure 3.39 Nutcracker object primitives and static device diagram.

The behavioral sequences for the idealized nutcracker components in the cracking function are illustrated in Figs. 3.39 - 3.43. The device behavior is enabled by pressing the handles together. The applied forces are transmitted from the handles to the pivot location (at 1 in

Figs. 3.39, 3.41), as mediated by the connection labeled (2). The right side of the diagram

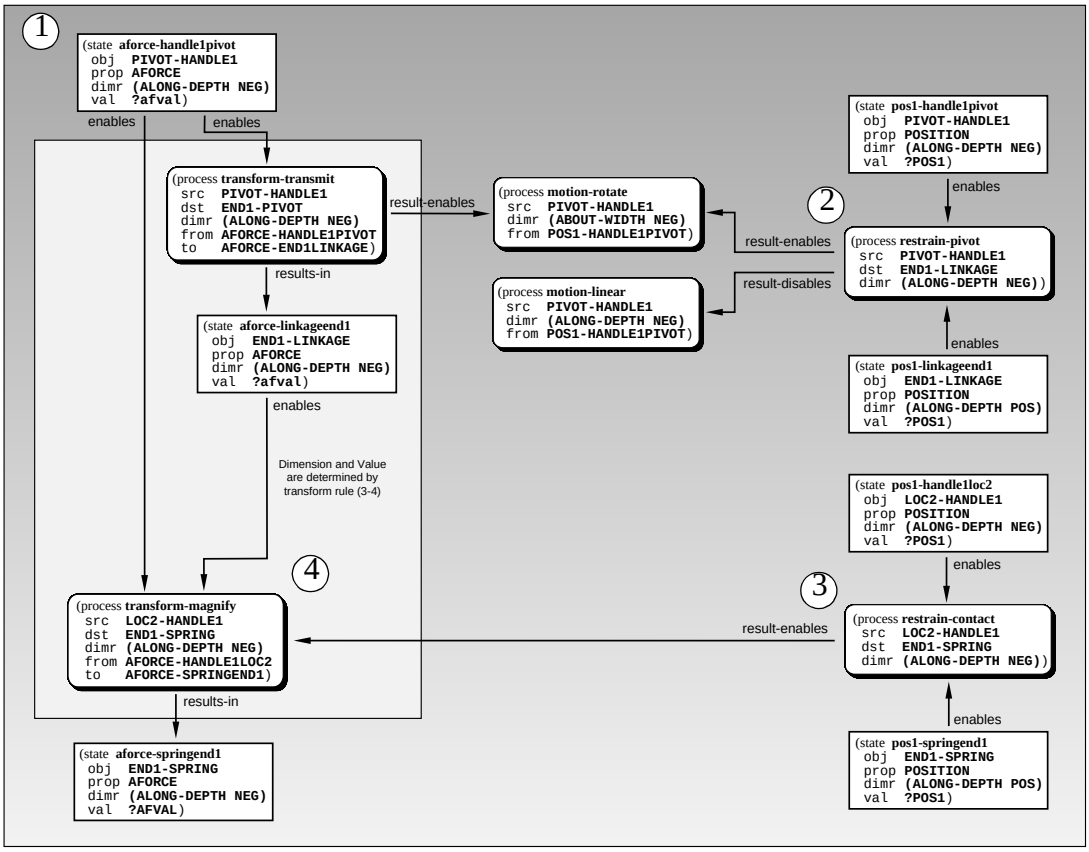


Figure 3.40 Behavioral sequence for upper nutcracker handle in the cracking function.

is similar to LEVER2 of the bottle opener, since two contacts are needed to produce mechanical advantage. The difference is in the location of LOC2-LEVER in the bottle opener and LOC2-HANDLE1 and LOC2-HANDLE2 on the nutcracker, which result in the type1

and type2 leverage, respectively. The resolution of forces in the linkage-object are shown

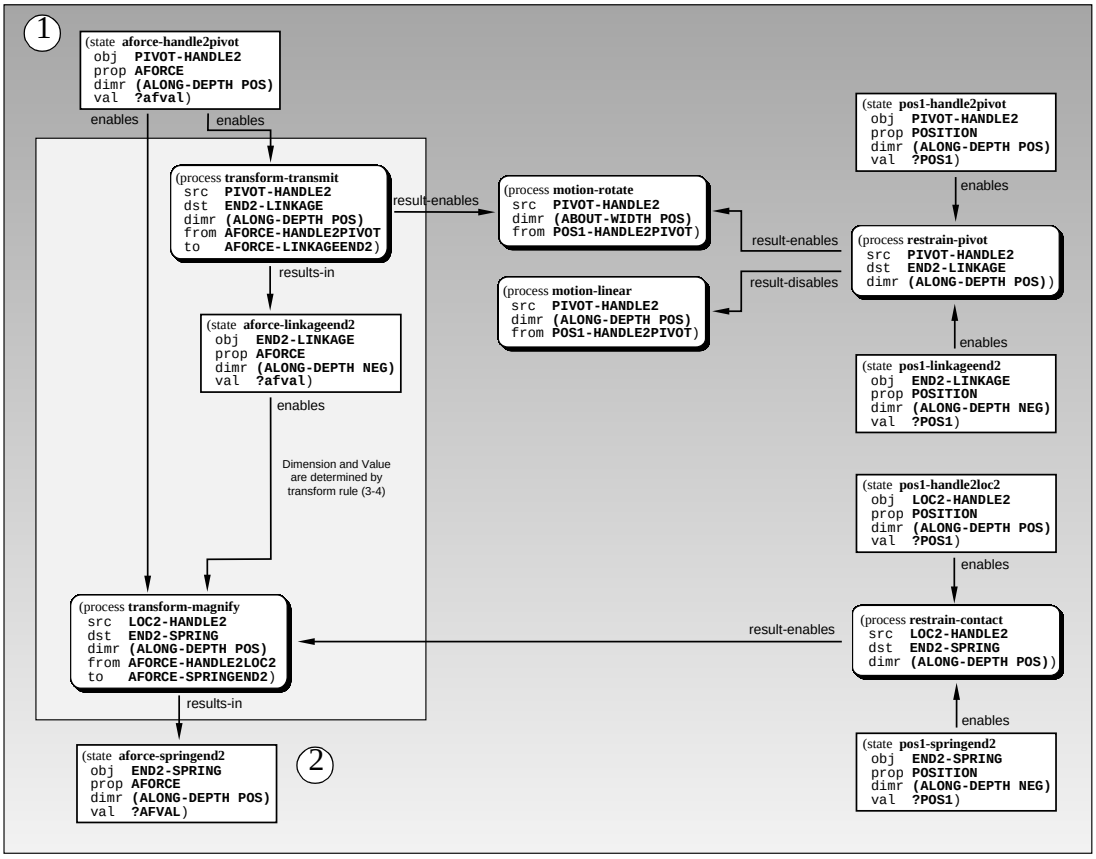


Figure 3.41 Behavioral sequence for lower nutcracker handle in the cracking function.

in Fig. 3.42. The two forces transmitted from HANDLE1 and HANDLE2 to the linkage-object (i.e., LINKAGE) are shown at (1). The force is transmitted through the linkage-object to the other lever-object (END1-LINKAGE → PIVOT-HANDLE2, and END2-LINKAGE → PIVOT-HANDLE1, at 2). The result is forces of equal value and opposite direction,

which cancel (at 4), and which collectively disable motion of the linkage-object (3). The

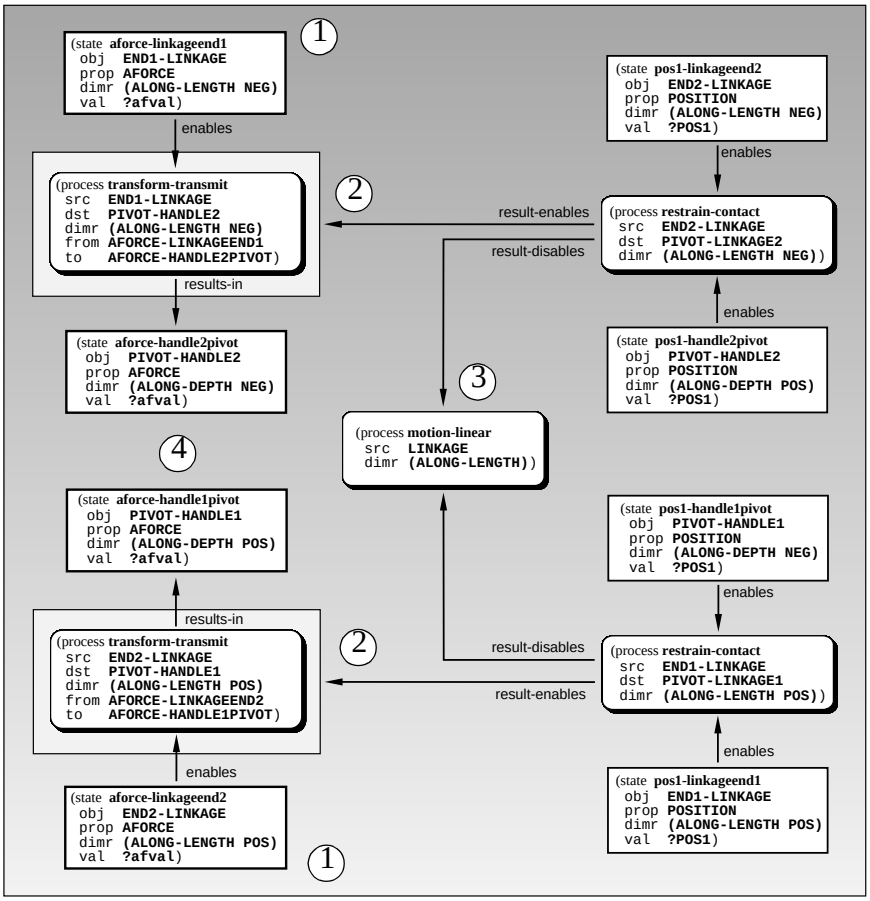


Figure 3.42 Behavioral sequence for nutcracker pivot in the cracking function.

final figure, for the spring-object, is shown in Fig. 3.43. The force which is transmitted from the handles in Figs. 3.39, 3.41 is shown at (1). For the spring-object to compress, its motion must be restrained. The contact between the *other* spring-object location and the *other* lever-object (`END2-SPRING` with `LOC2-HANDLE2`, for `END1-SPRING`, and `END1-`

SPRING with LOC2-HANDLE1, for END2-SPRING, at 2) achieves this effect (3). The

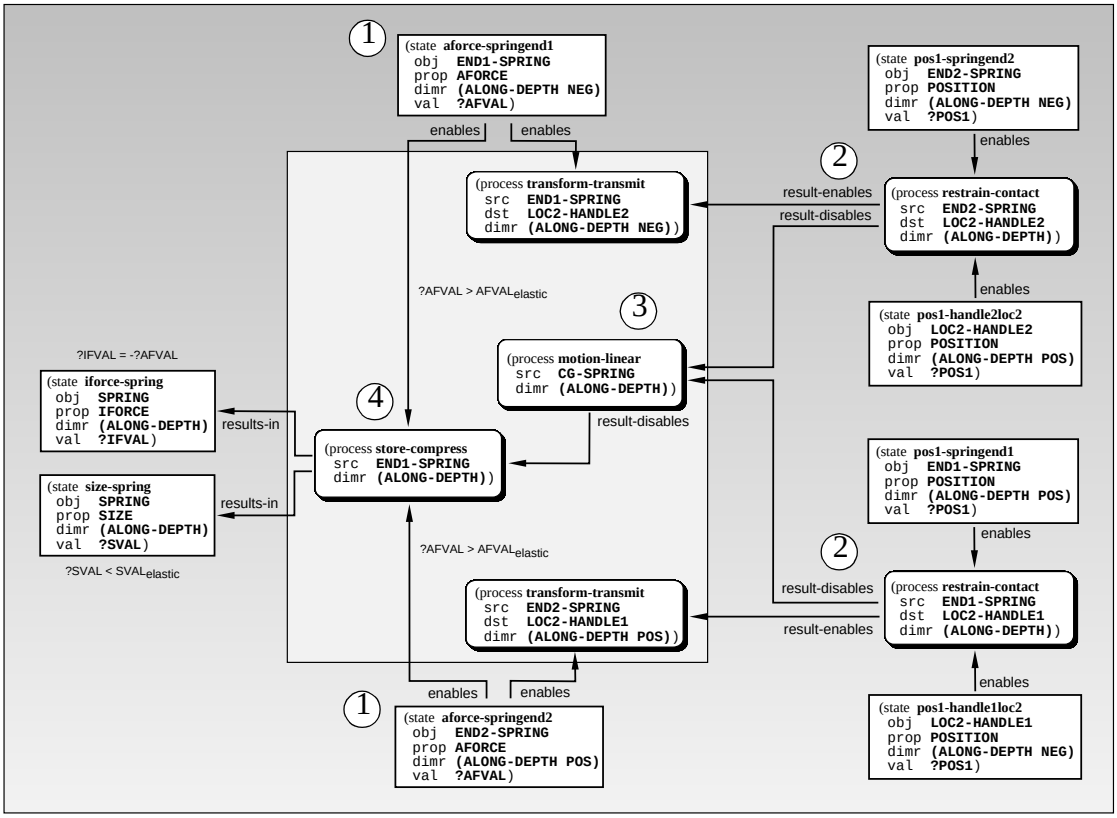


Figure 3.43 Behavioral sequence for nutcracker spring in the cracking function.

combination of applied force which exceeds the elastic threshold (4), and motion restraint, enables spring-object compression (4). To add the effect of containment in the nutcracker to these representations, two container-objects (or container-object combinations, as presented in Chapter 2, Page 88) could be added at LOC2-HANDLE1 and LOC2-HANDLE2. Two more behavioral diagrams would then be required.

Clothes Pin

A clothes pin holds objects together without manual control. The device is comprised of three components: two lever-objects and a spring-object. The spring-object is employed to keep the lever-objects together, as well as to keep the jaws closed, as shown in the idealized

clothes pin and static device diagram depicted in Fig. 3.44.

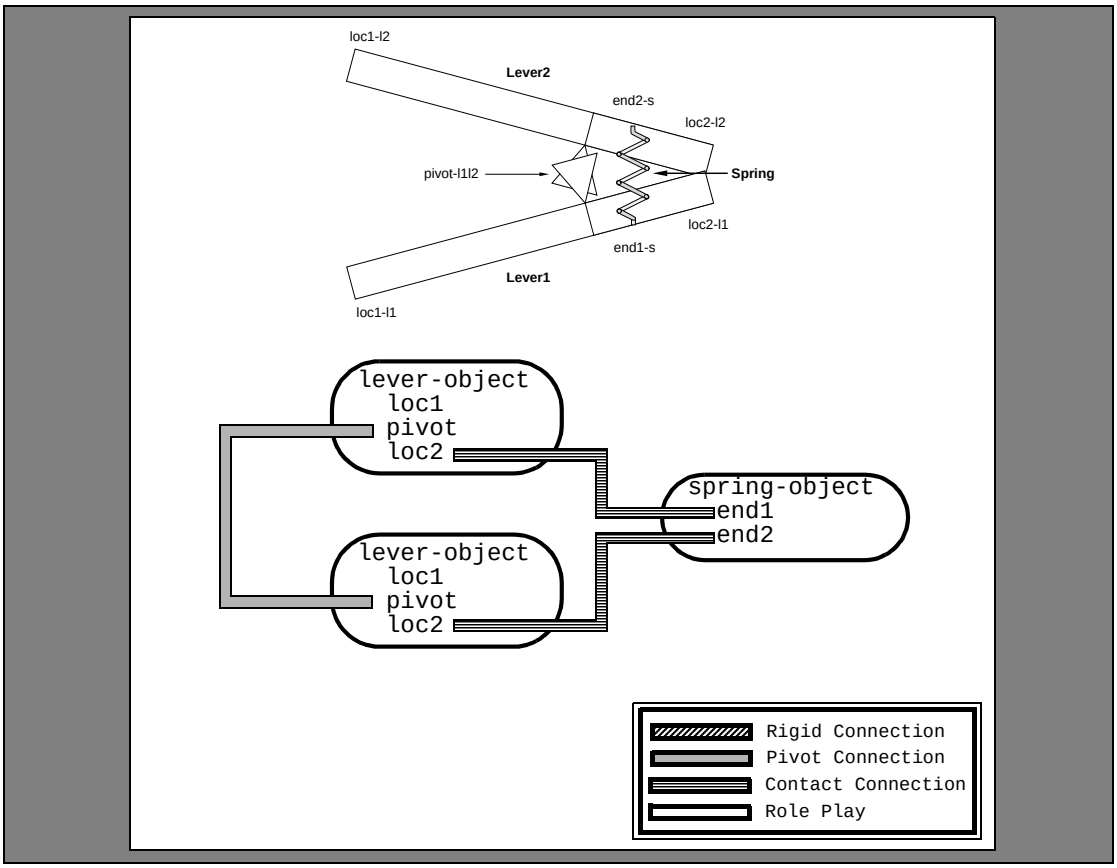


Figure 3.44 Clothes pin object primitives and static device diagram.

The static representation diagram for the clothes pin is almost identical to that of the nut-cracker in Fig. 3.39, but they behave differently under the same applied force because the lever-object pivot region is located away from the handle ends (it was at the handle ends in the nutcracker). The behavioral sequences for the idealized clothes pin components are shown for the clasping function in Figs. 3.44 - 3.47. The location of the clothes pin spring-object and the orientation of its contact (i.e., above) the handles results in the clothes pin being closed at rest, whereas the nutcracker is open at rest. In other respects the behavioral representations for the devices are identical. Figs. 3.44, 3.46 show the magnification of

force to the spring-object. In each figure, the applied force is shown at the lever-object pivot

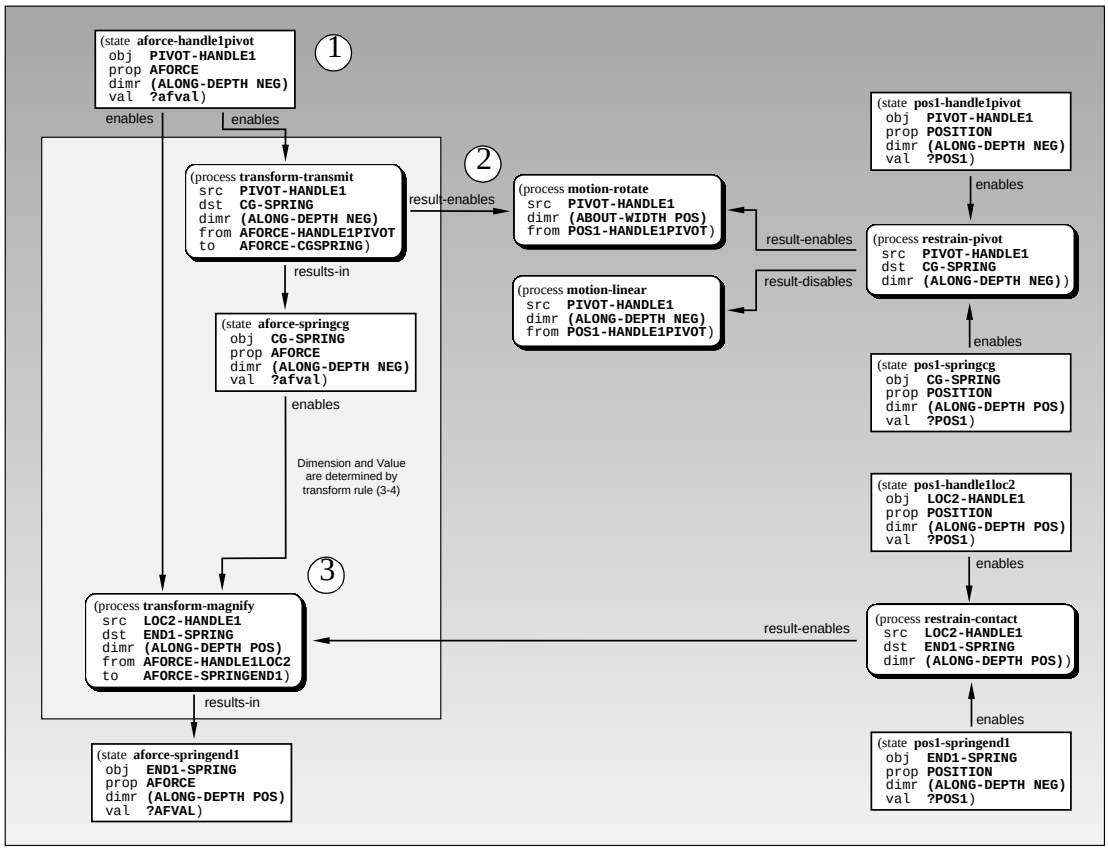


Figure 3.45 Behavioral sequence for upper clothes pin handle in the clasping function.

region (1). The transmission of force at PIVOT-HANDLE1, and PIVOT-HANDLE2, to CG-SPRING, is interesting, since the spring-object is effectively the fulcrum about which the lever-objects rotate. The magnification of force (3), thus works on two locations on the

spring-object. The spring-object behavioral sequence is illustrated in Fig. 3.47. The spring

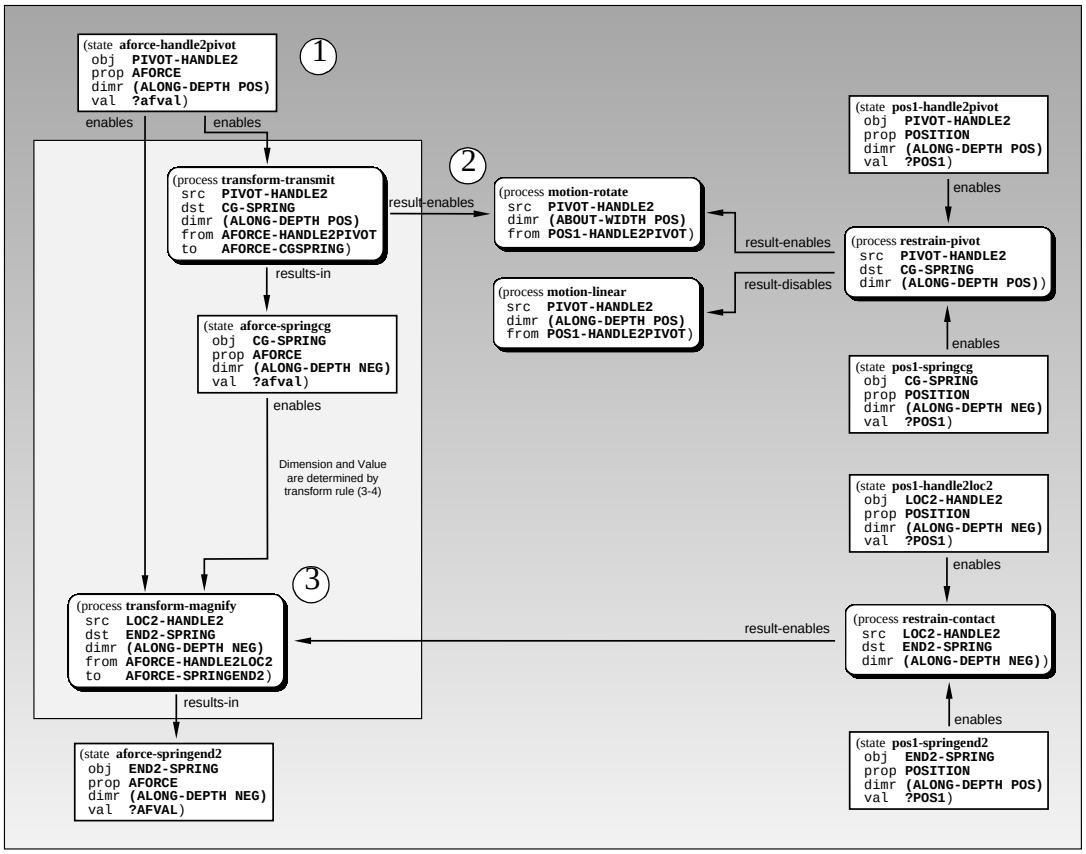


Figure 3.46 Behavioral sequence for lower clothes pin handle in the clasping function.

in the clothes pin is similar to that in the nutcracker inasmuch as both springs are connected to two handles which serve as levers. The detail shown in Fig. 3.43 for the forces propagating from the lever-objects to the spring-object has been omitted in Fig. 3.47. Only one half

has been shown, from the upper handle to the spring CG.

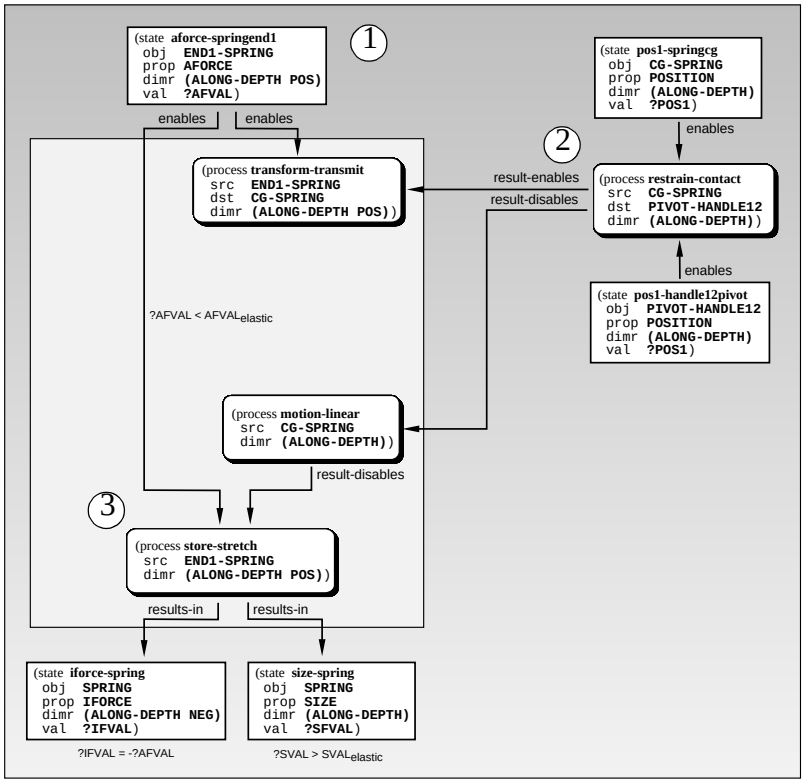


Figure 3.47 Behavioral sequence for clothes pin spring in the clasping function.

The other half is identical in form (see Fig. 3.43) to the nutcracker spring-object.

3.4.3 Multiple Compound Devices

Where a clothes pin has two components which can move independently, a multiple-compound device has more than one independently moving component. Although the statics and intra-object dynamics are fairly straightforward, the inter-object dynamics begins to become complex when fully elaborated. One such device is the fingernail clipper, whose behavioral representation is presented in this section.

Fingernail Clipper

A fingernail clipper clips nails by forcing two blades together. The force necessary to clip is achieved with a lever-object through magnification. The magnification results from a pivot connection between the lever-object and a pin (a linkage-object) which is restrained from motion by the lower linkage/blade. The lever-object has a fulcrum on it which presses against the upper linkage/blade. The magnified lateral force on the upper linkage/blade deflects it like a spring, forcing the two blade-objects together. An idealized fingernail clipper and its associated static device diagram are shown in Fig. 3.48. The linkage/blade objects are represented as separate objects rigidly connected. The associated linkage-objects are represented as being pivoted, and their function as springs is represented with a spring-object. The handle (lever-object) can move independently of the linkage/blade-objects, and the blade-objects can move with respect to each other. The handle is pivoted to the pin (at

1 in Fig. 3.48), so it can rotate about the pin in two radial dimensions.

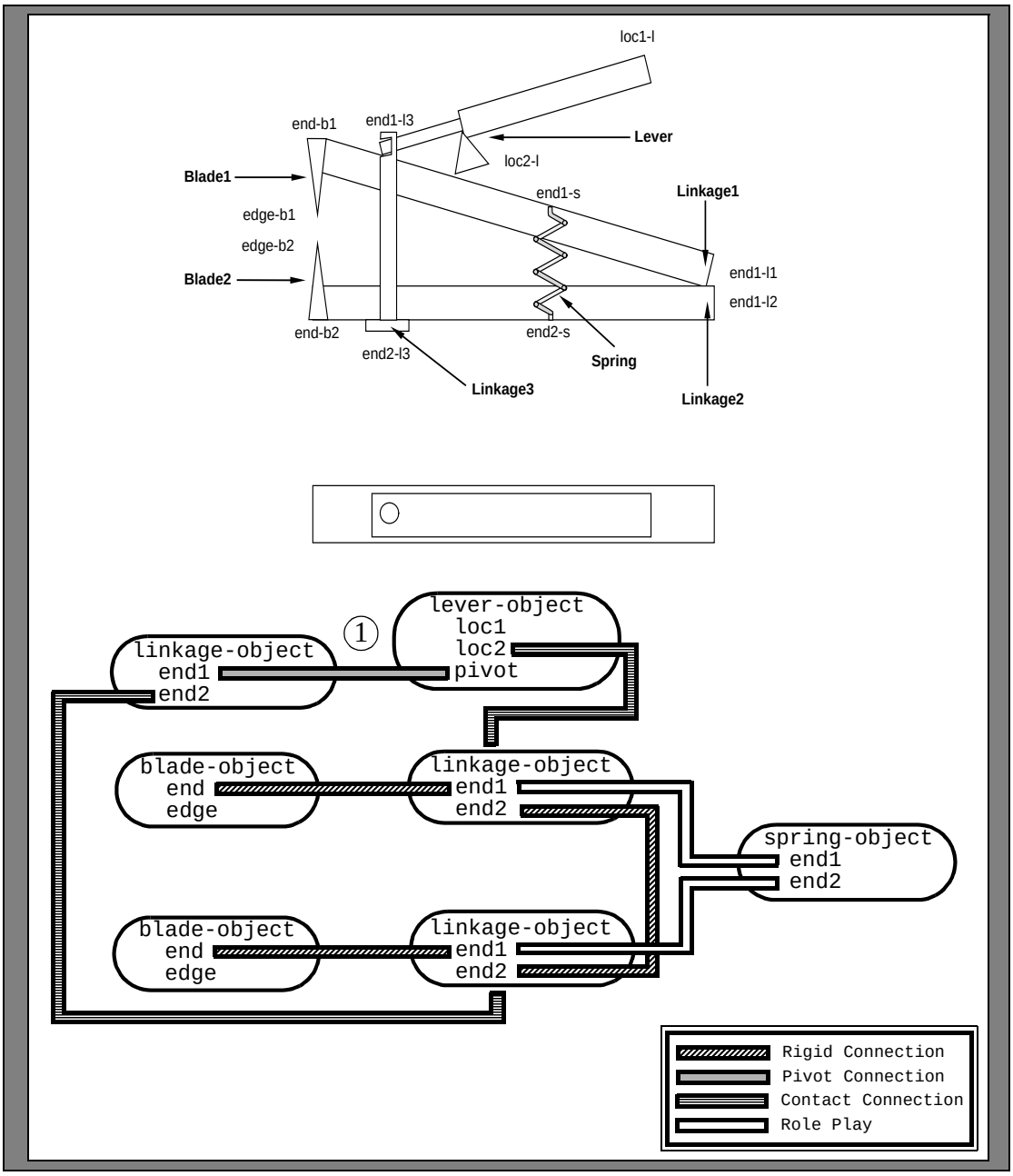


Figure 3.48 Fingernail clipper object primitives and static device diagram.

The behavioral sequences of the fingernail clipper components as they are used in the clipping function are shown in Figs. 3.49 - 3.55.

Fig. 3.49 shows the behavioral sequence associated with the fingernail clipper handle. The

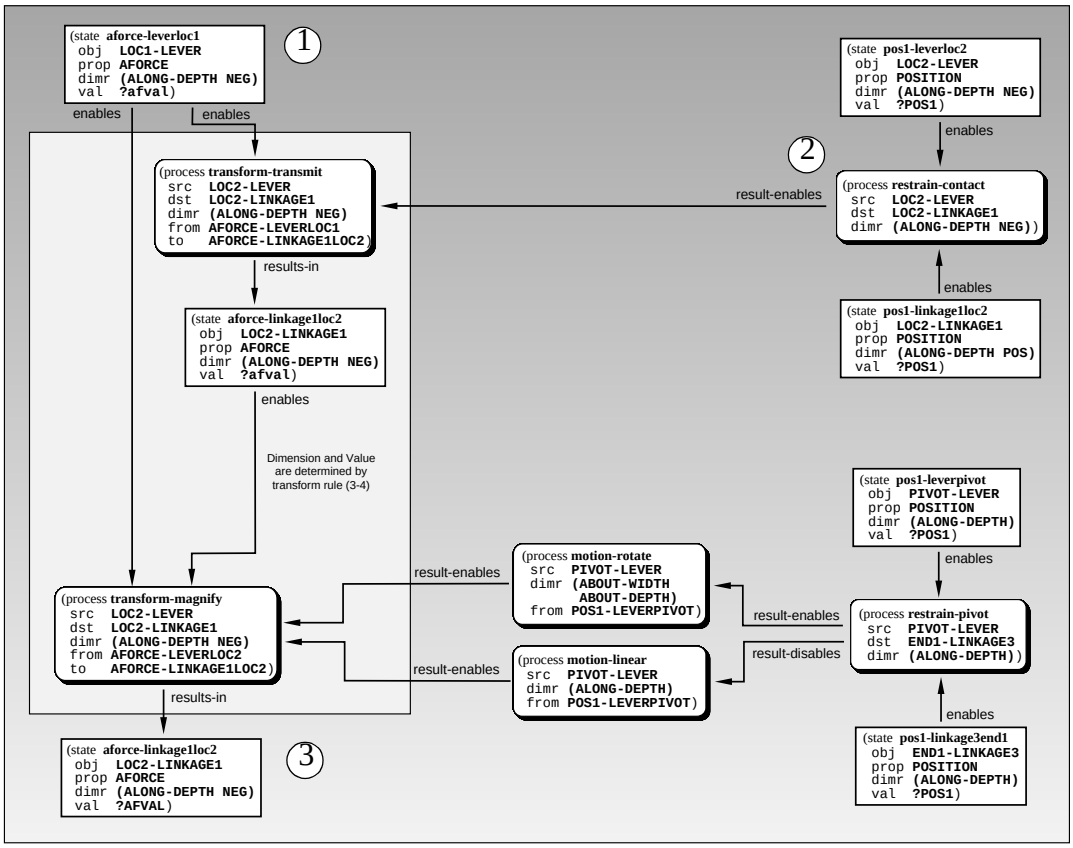


Figure 3.49 Behavioral sequence for fingernail clipper handle in the clipping function.

force resulting from pressing the handle (1) is transmitted to the upper linkage (`LOC2-LINKAGE1`) as a result of the contact shown at (2). The pivot relationship with the pin (`END1-LINKAGE3`) enables the force magnification (3).

The upper linkage-object (`LINKAGE1`) and blade-object (`BLADE1`) behaviors are shown

in Fig. 3.50 and Fig. 3.51. The upper linkage is connected to three components: SPRING

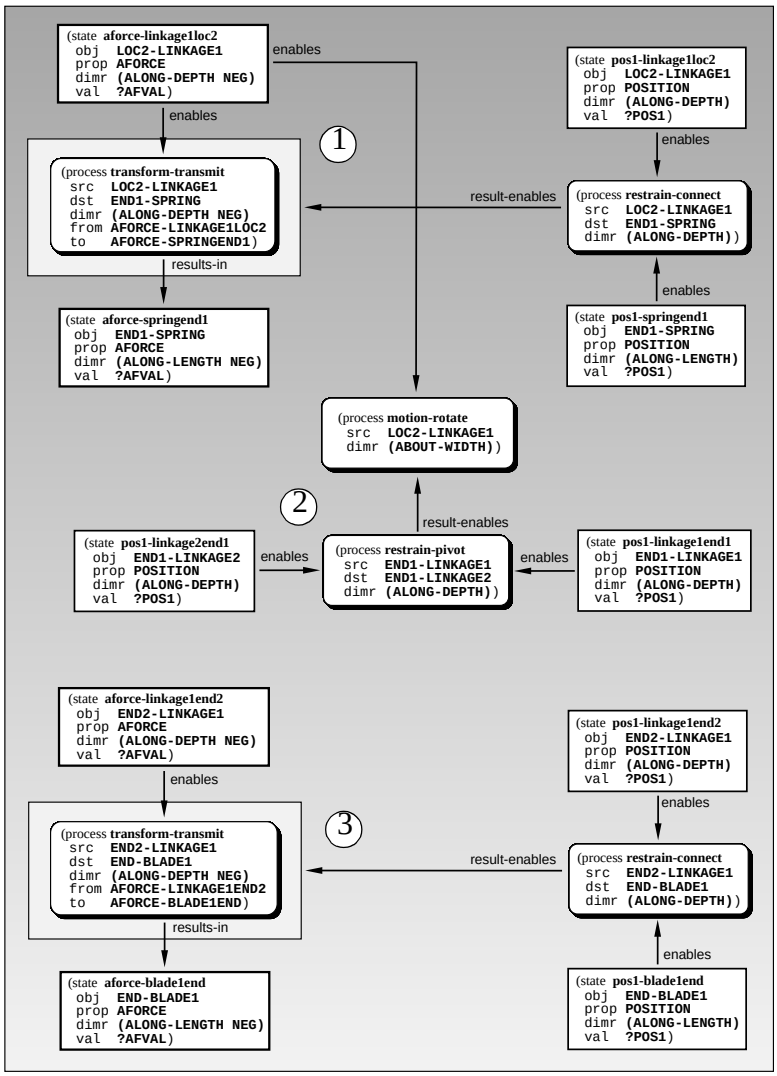


Figure 3.50 Behavioral sequence for the upper fingernail clipper linkage in the clipping function.

(1), LINKAGE2 (2), and BLADE1 (3), as noted by the labels in Fig. 3.50. The force which is transmitted from LEVER to LINKAGE1 is transmitted to these components. Fig. 3.51 shows how the magnified force (at 1) is again magnified (i.e., linear magnification) due to

the shape of BLADE1 (2). The upper blade is assumed to be in contact with a fingernail (FN)

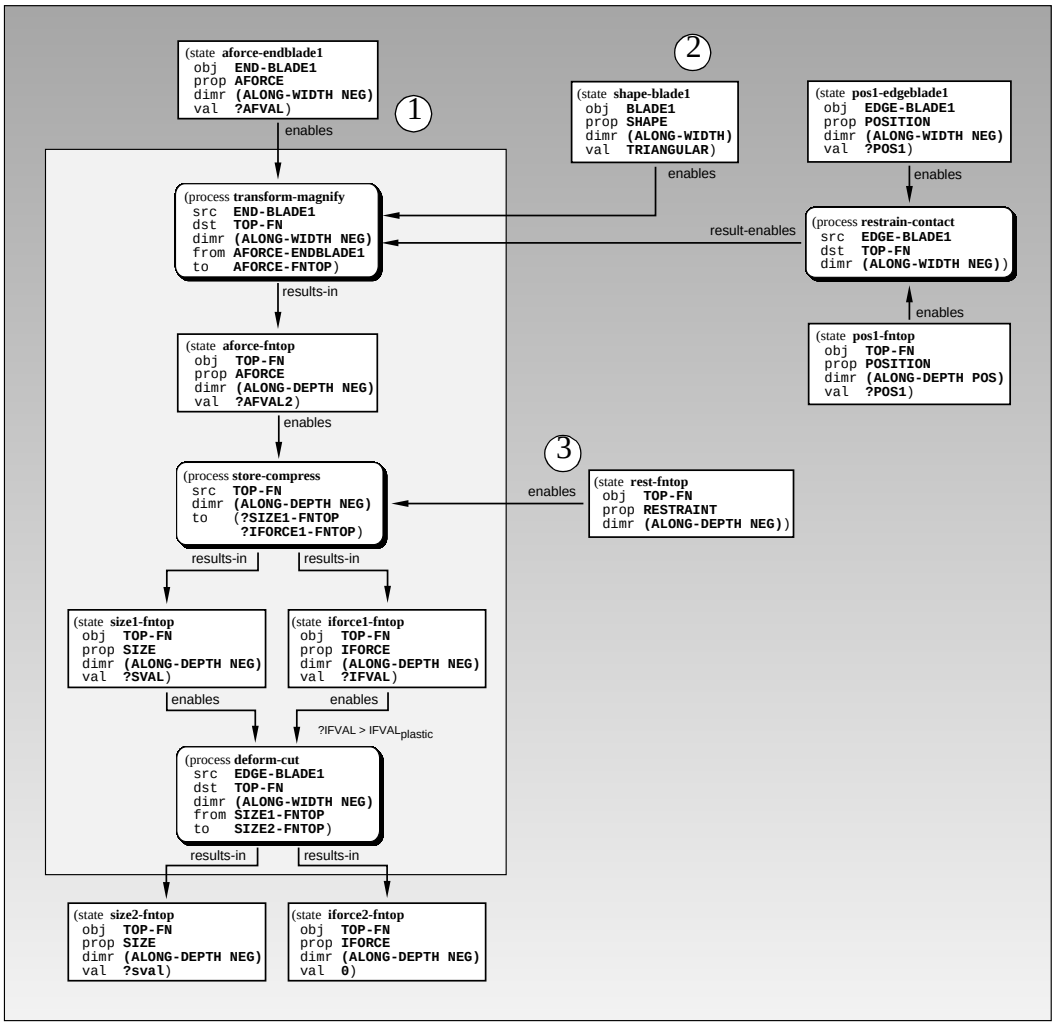


Figure 3.51 Behavioral sequence for the upper fingernail clipper blade in the clipping function.

for the example, and the fingernail is assumed to be restrained in the dimension and direction of BLADE1 movement (i.e., ALONG-DEPTH NEG, at 3). The lower linkage (LINKAGE2, Fig. 3.52) differs from LINKAGE1 inasmuch as the pin (LINKAGE3) is in contact with LINKAGE2 (at 1 in Fig. 3.55) and not with LINKAGE1, and because LEVER magnifies force to LINKAGE1, but that force is only transmitted to LINKAGE2 through LINKAGE3 (at 2 in Fig. 3.55) due to the pivot connection assumed between the upper and

lower linkage-objects in the idealization. Figs. 3.53, and 3.54 are otherwise identical to

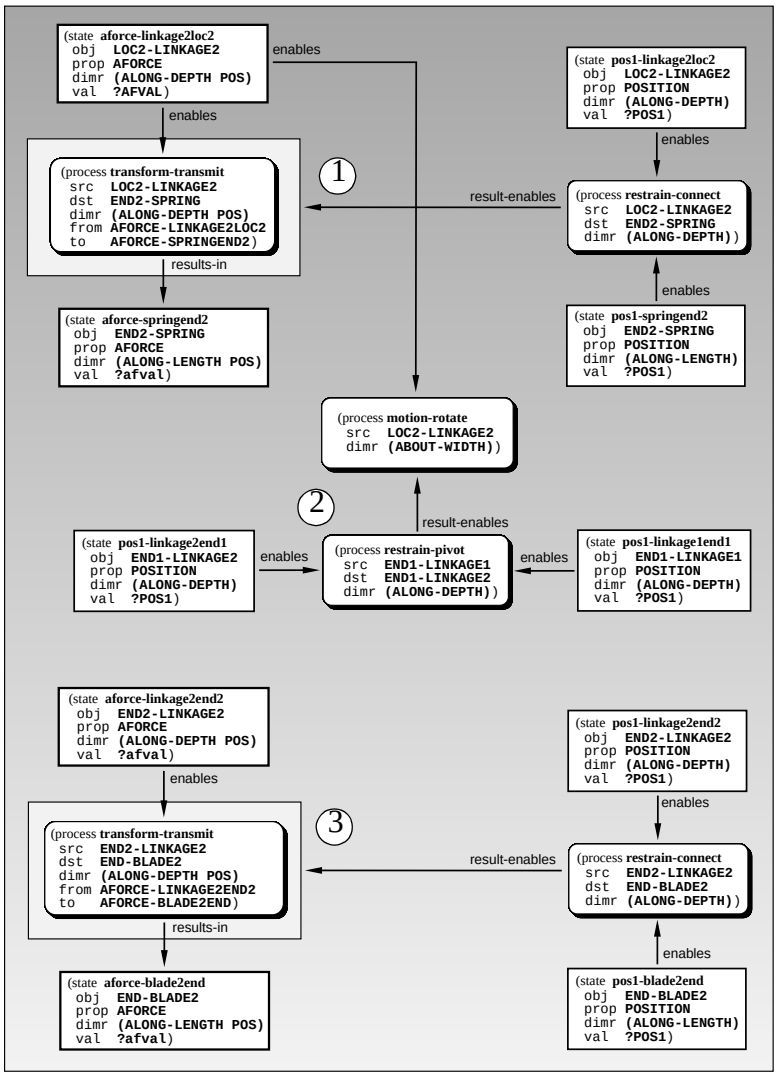


Figure 3.52 Behavioral sequence for the lower fingernail clipper linkage in the clipping function.

Figs. 3.50, and 3.52.

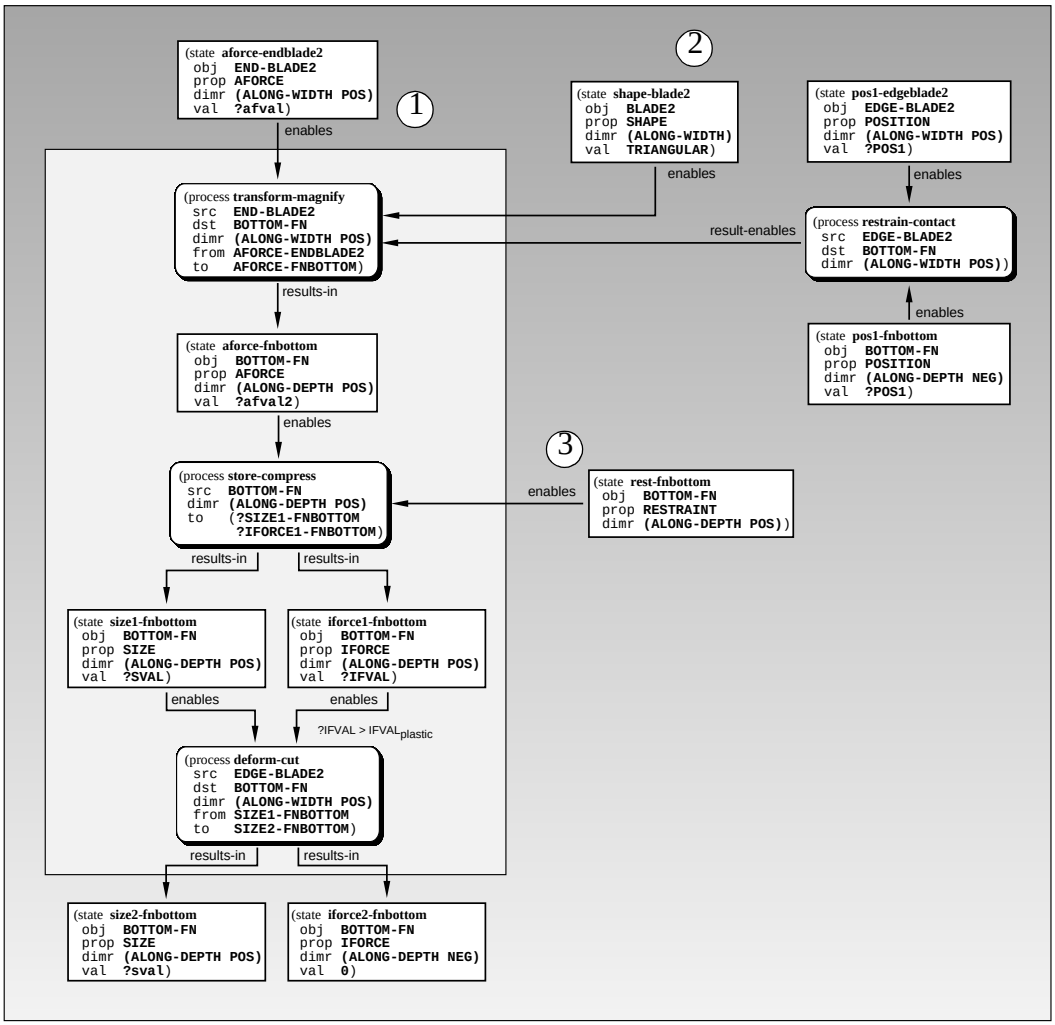


Figure 3.53 Behavioral sequence for the lower fingernail clipper blade in the clipping function.

The fingernail clipper spring-object behavior is illustrated in Fig. 3.54, and is identical

in behavior to that shown for the nutcracker device in Fig. 3.43.

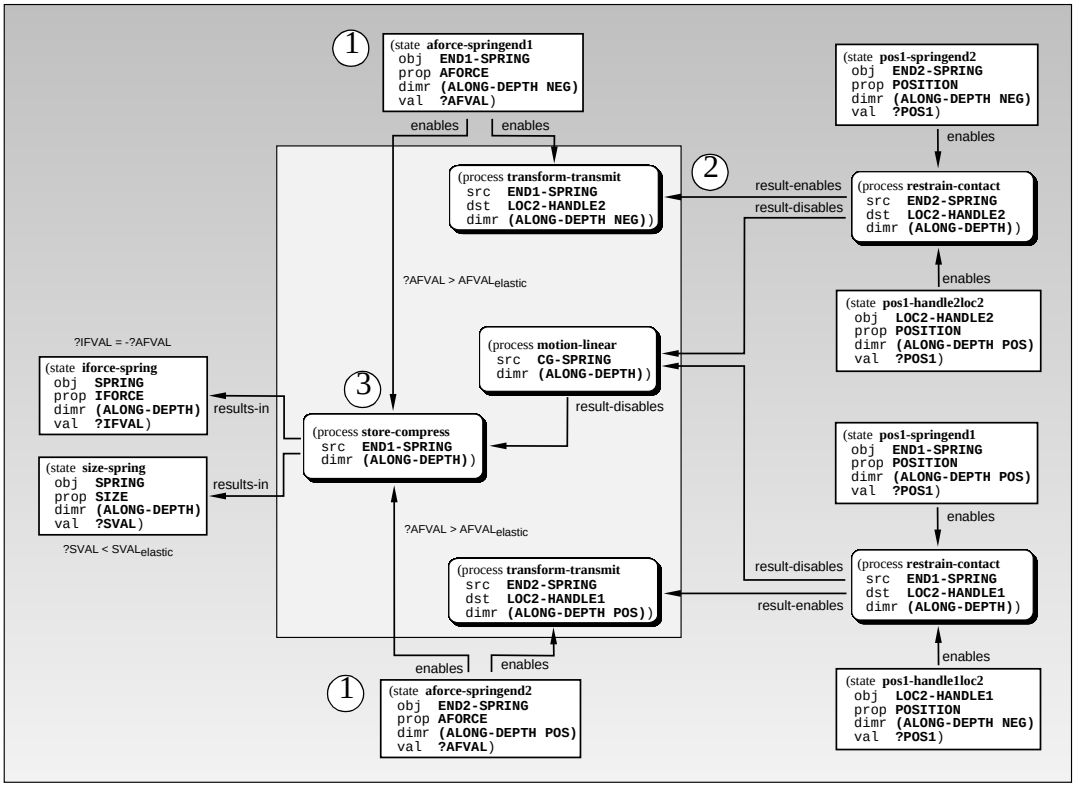


Figure 3.54 Behavioral sequence for the fingernail clipper spring in the clipping function.

The nutcracker pin is shown in Fig. 3.55.

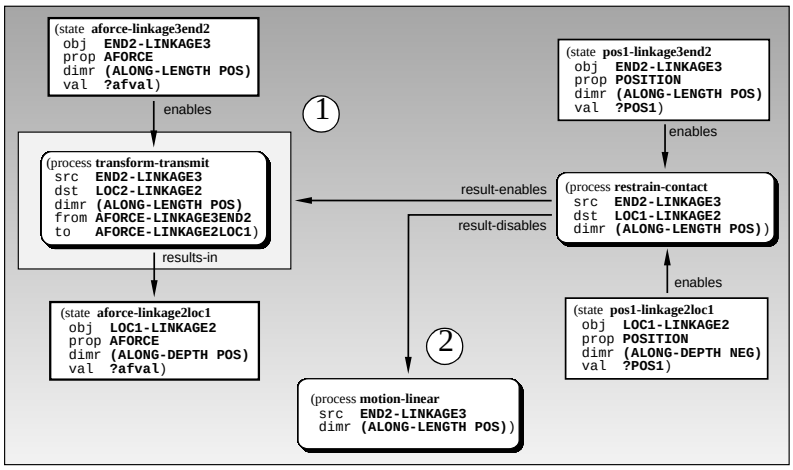


Figure 3.55 Behavioral sequence for the fingernail clipper pin in the clipping function.

3.4.4 Complex Devices

Complex devices involve objects which may or may not be permanent components of the

device, have complex motions, or have large numbers of components. One such example is a toy gun, which has a projectile (dart) which is only connected to the device when being armed and preparing to fire. Another complex device is a mousetrap, since the hook and baitplate motion are both confined to the limits imposed by their connectors, but free to move in every other respect. In this section, the toy gun's behavioral representation is presented.

Toy Gun

The idealized toy gun and static device diagram are illustrated in Fig. 3.56.

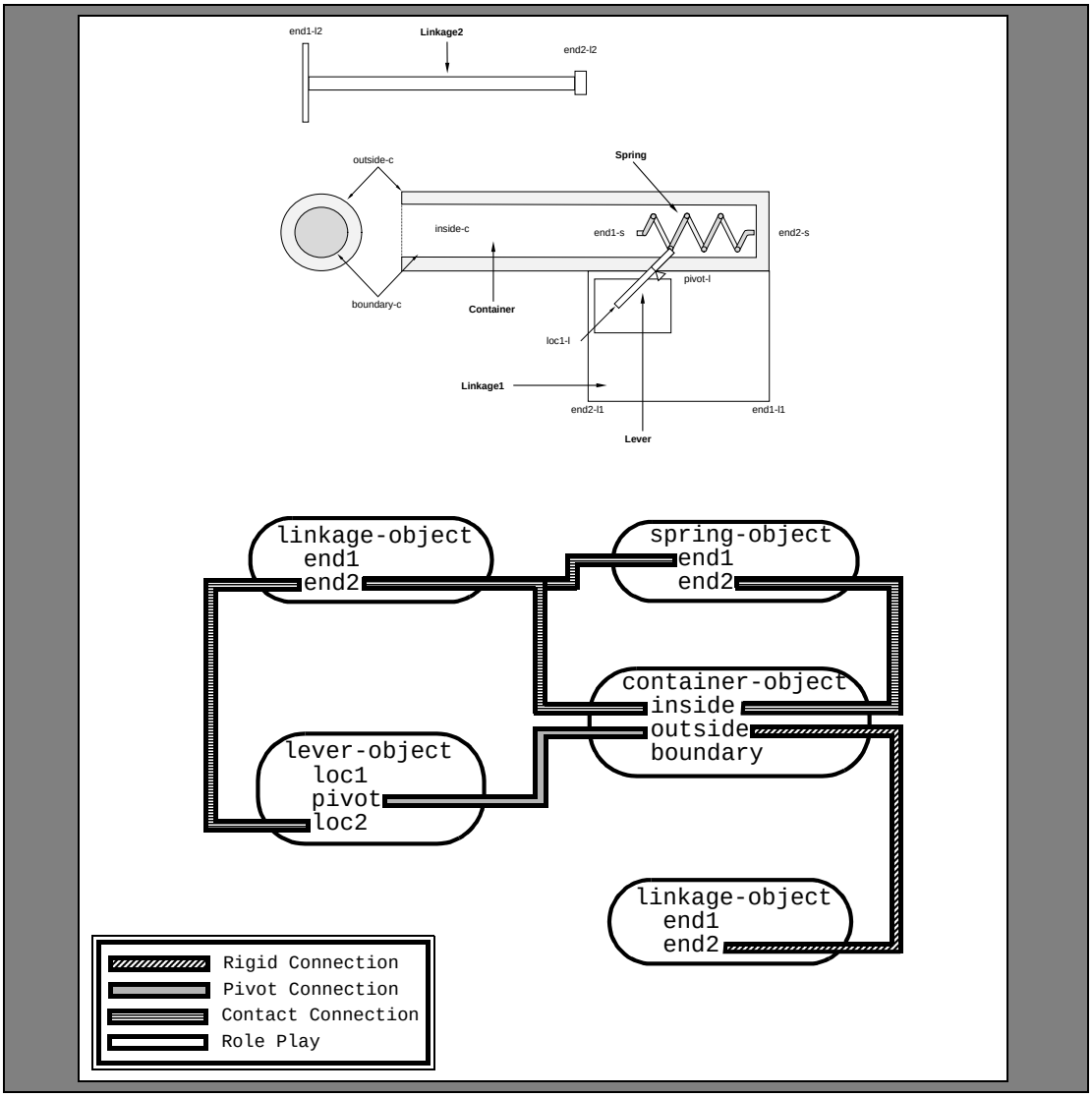


Figure 3.56 Toy gun object primitives and static device diagram.

The behavioral sequences associated with arming the toy gun are illustrated in Figs. 3.58 - 3.61. Fig. 3.58 illustrates the transmission of force from the end of the dart (`END1-`

LINKAGE1) to the spring inside the barrel (END1-SPRING).

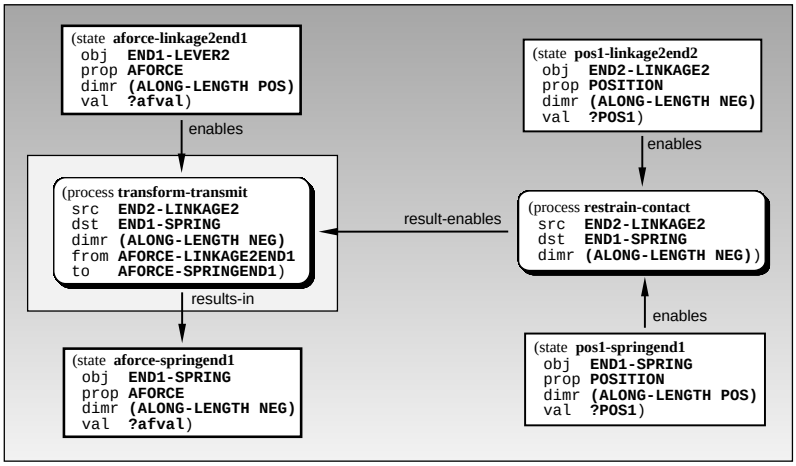


Figure 3.57 Behavioral sequence for the toy gun dart in the arming function.

Fig. 3.58 shows the handle and how force is transmitted from the handle (LINKAGE1) to the barrel (CONTAINER). An applied force at the trigger (LEVER) is assumed for the figure. The applied force is transmitted to the handle (LOC1-LINKAGE1), which is the location where the handle and barrel are connected (2). The second connection (rigid, as noted at 2), enables force transmission to the barrel (3). This connection also enables force transmission back through the barrel to the handle when the dart (LINKAGE2) is pushed against

the spring-object. The barrel (CONTAINER) behavior is represented in Fig. 3.59. The fig-

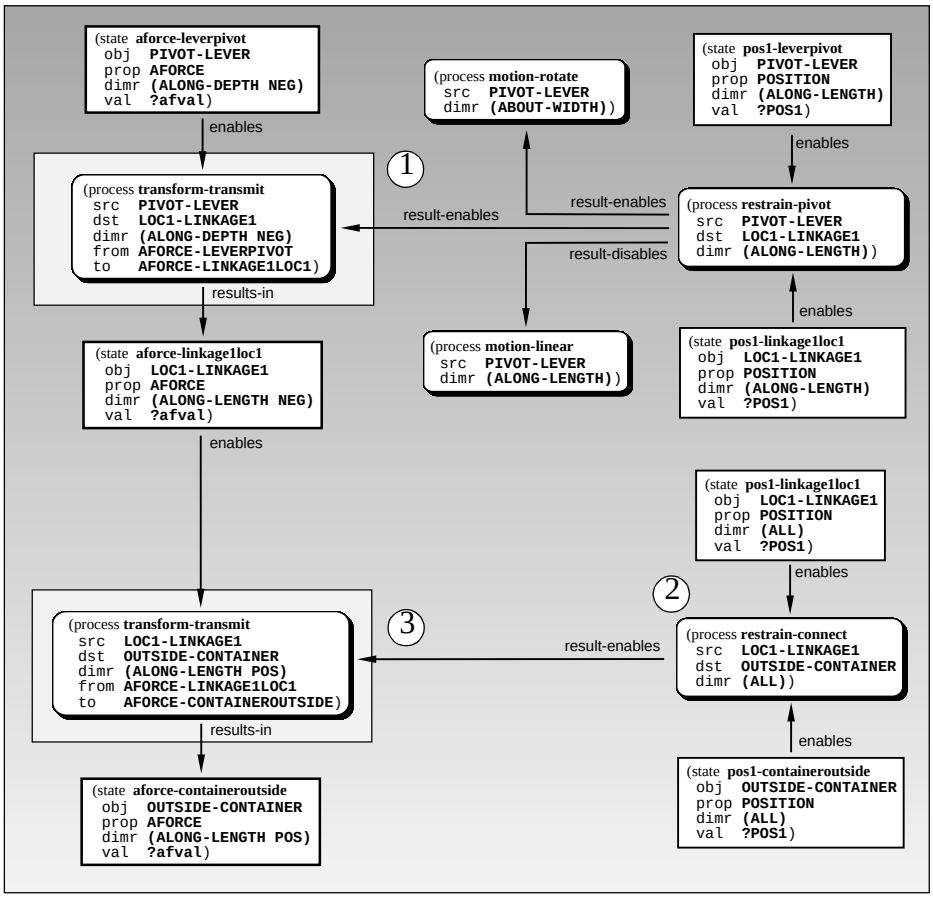


Figure 3.58 Behavioral sequence for the toy gun handle in the arming function.

ure is shown for the dart already contained by the barrel and represents how the container affects its ability to move. The enabling force is the applied force on the dart which resulted in its motion into the barrel. The RESTRAIN process (1), which represents the relationship between the dart and barrel radially, enables linear motion for the dart while disabling radial motion of the dart (2). The enabled linear motion results in a change in dart position which, if equivalent to INSIDE-CONTAINER (3), enables RESTRAIN-CONTACT (4) which, in turn, disables dart motion (5). The effect of RESTRAIN-CONTAIN is to define

the limits of motion and not to otherwise restrict, or care about, the position of the dart. The

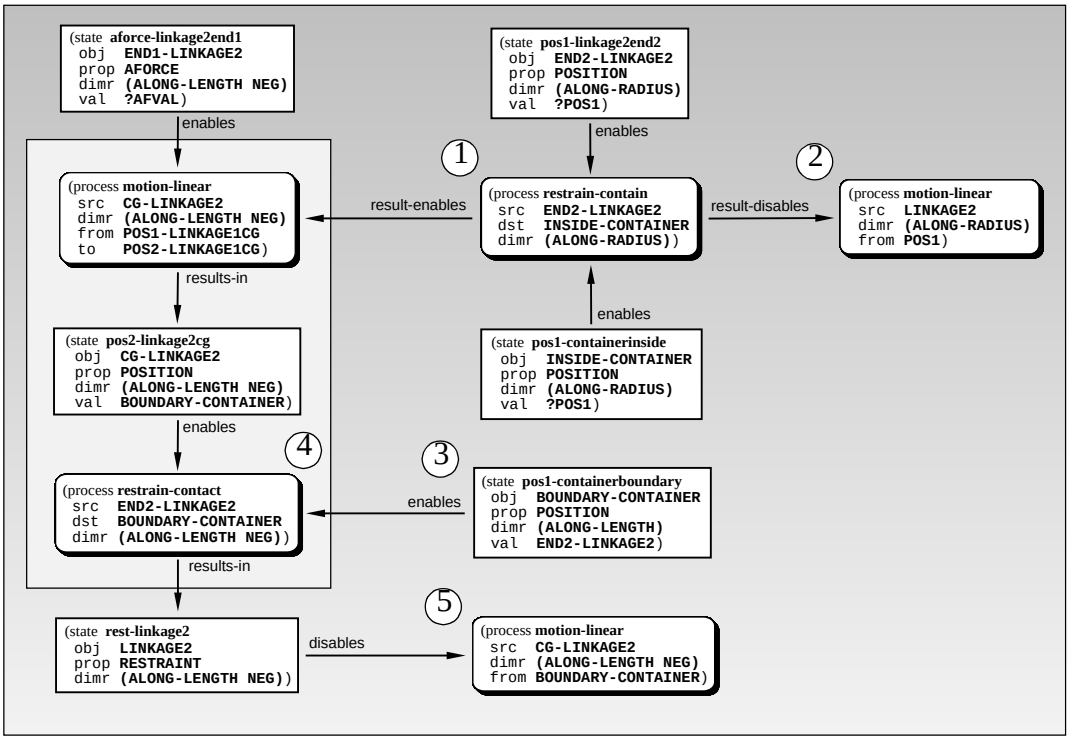


Figure 3.59 Behavioral sequence for the toy gun barrel in the arming function.

toy gun trigger mechanism is represented with a lever-object, shown in Fig. 3.60 instantiated for the dart in contact with the spring-object when the gun is armed. The behavior is enabled by pulling on the trigger, and the initial applied force in the diagram is the force at LOC2-LEVER. The force is transmitted to PIVOT-LEVER, which is translationally con-

nected (i.e., ALONG-ALL) at LOC1-LINKAGE1 (1) to the handle. The associated contact

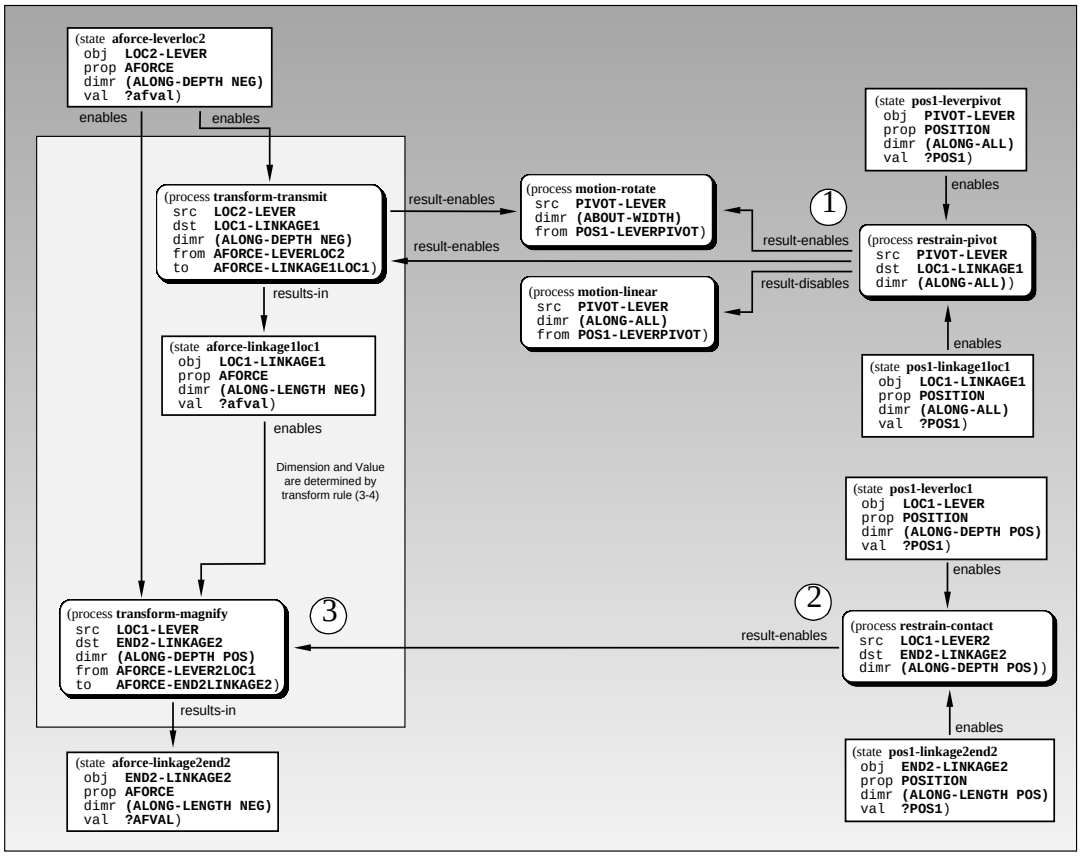


Figure 3.60 Behavioral sequence for the toy gun trigger in the arming function.

between LOC1-LEVER and END2-LINKAGE2 (i.e., dart, at 2), enables the magnification of force on the dart (3). The final diagram for the toy gun illustrates the behavior associated with the spring-object, in Fig. 3.61. As with other springs represented in this section, CG-SPRING must be restrained from motion or compression cannot be enabled. Unlike the springs in the nutcracker, the clothes pin, and the fingernail clipper, the toy gun spring is connected at one end (END2-SPRING) to the barrel (1), which disables spring-object motion ALONG-LENGTH (2) as long as the barrel is so restrained (as mentioned in the handle

discussion for Fig. 3.58). The force applied by the dart (END2-LINKAGE2) to the spring-

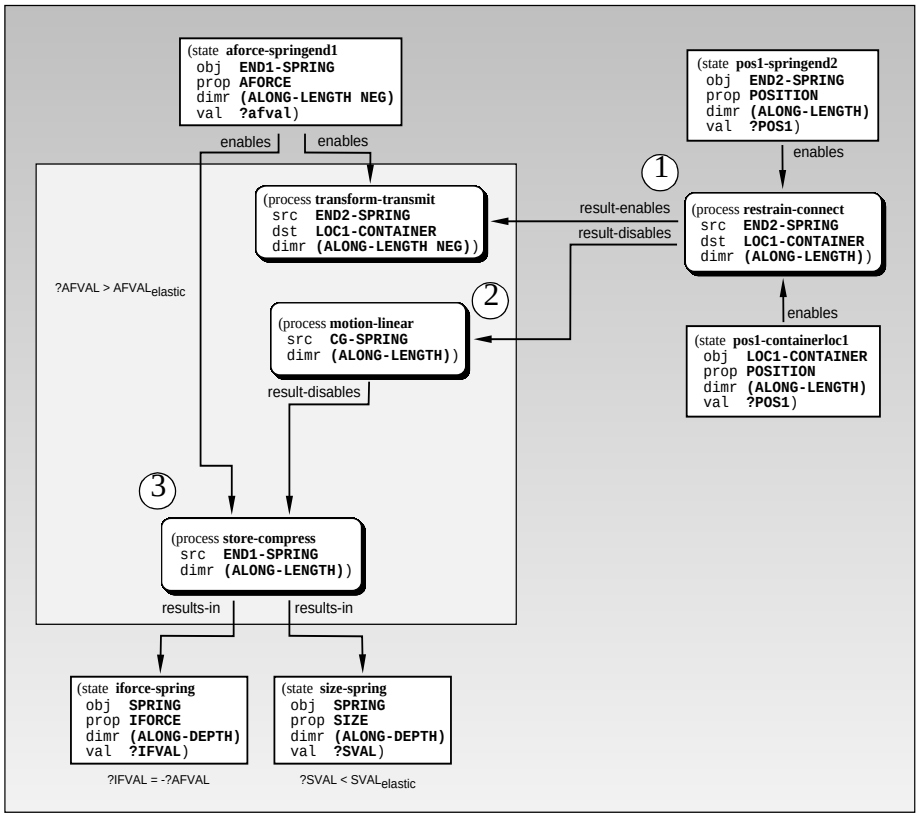
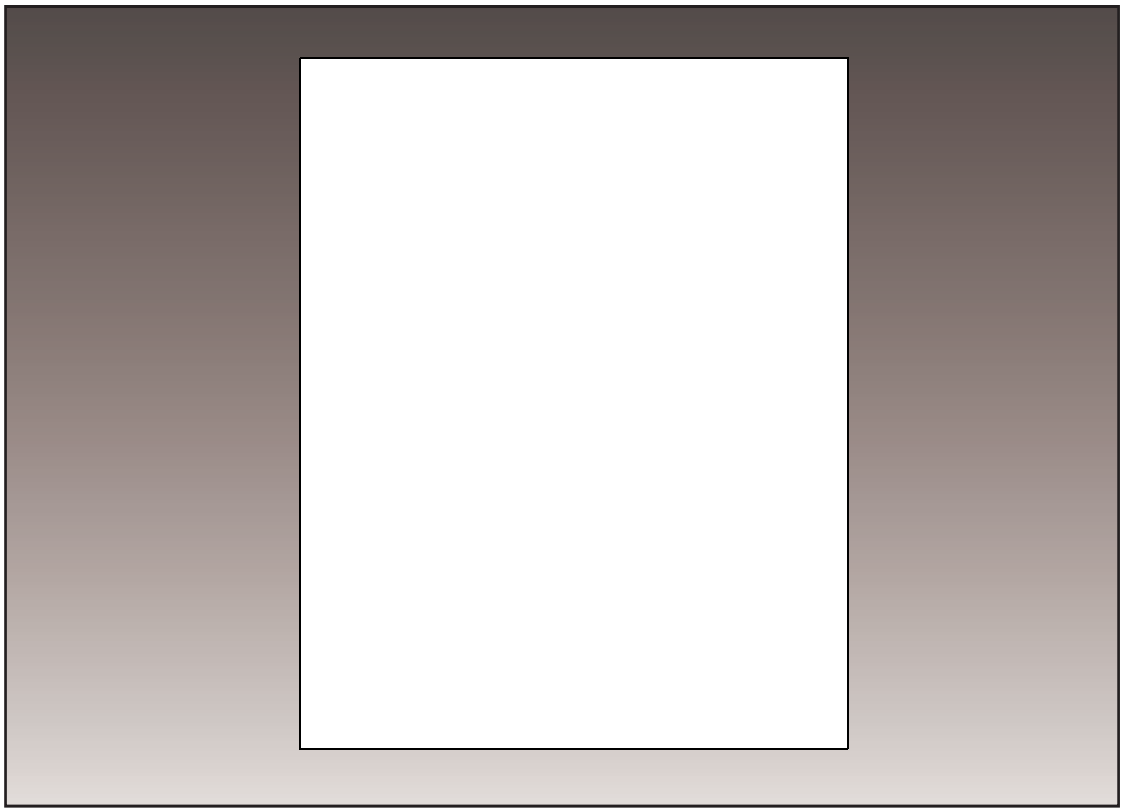


Figure 3.61 Behavioral sequence for the toy gun spring in the arming function.

object (END1-SPRING) thus contributes to enabling of STORE-COMPRESS (3) as long as the applied force exceeds the elastic threshold.



Chapter 4
High-Level Dynamics and Device Function

In this chapter, the representation constructs which are used in FONM to describe high-level device dynamics are presented. Every device and device function can be described in terms of a finite set of objects which have associated with them a specific function. These *machine primitives* comprise a set of functional building blocks for representing complex devices. The functions of these primitives are represented with behavioral sequences, so the discussion begins by defining a structure for function which is based on device behavior, and which is based on the physical requirements of objects which can instantiate them. The discussion ends with the presentation of dynamic representations for six devices of increasing complexity. Three topics are addressed in this chapter:

(1) Device function
(2) Machine primitives (MPs)
(3) MP combinations and mechanical device representation

4.1 Device Function

A *function* describes *what* a device does and is invoked by perturbing a device. For example, two boards can be connected with screws using a screwdriver. The function of the screwdriver is to provide the necessary force needed to drive the screw into the boards. The driving function is invoked by turning the screwdriver when it is in contact with a screw, and results in a magnification of the rotational force applied at the handle to the screw head. A device may instantiate different functions, depending on what force is applied, to what region of the device the force is applied, and the device statics. The screwdriver can be used to punch holes in oil cans, or to pry staples from documents. Driving, punching, and prying are each invoked by applying a different type of force to the same object. Each function makes use of different aspects of the device statics, and each produces a different result. In each case, the overall function of the device can be represented as a combination of the functions of its components, and they, in turn, can be represented as BPP sequences.

The shaded portions of Figs. 3.31 - 3.33 represent the underlying behavioral sequence of the screwdriver components. If the shaded boxes are darkened, so that the details of the component behavior are not visible, as in Fig. 4.1, then the boxes represent the individual functions of the screwdriver components in the overall driving function. The only information entering the boxes are the enabling states from the applied force and from the boxes on the right, which represent the connectivity relations between the components (device statics). The only information leaving the boxes are the resulting states. This is called a *black box* description because it describes what happens when the device is perturbed, but not how it happens.

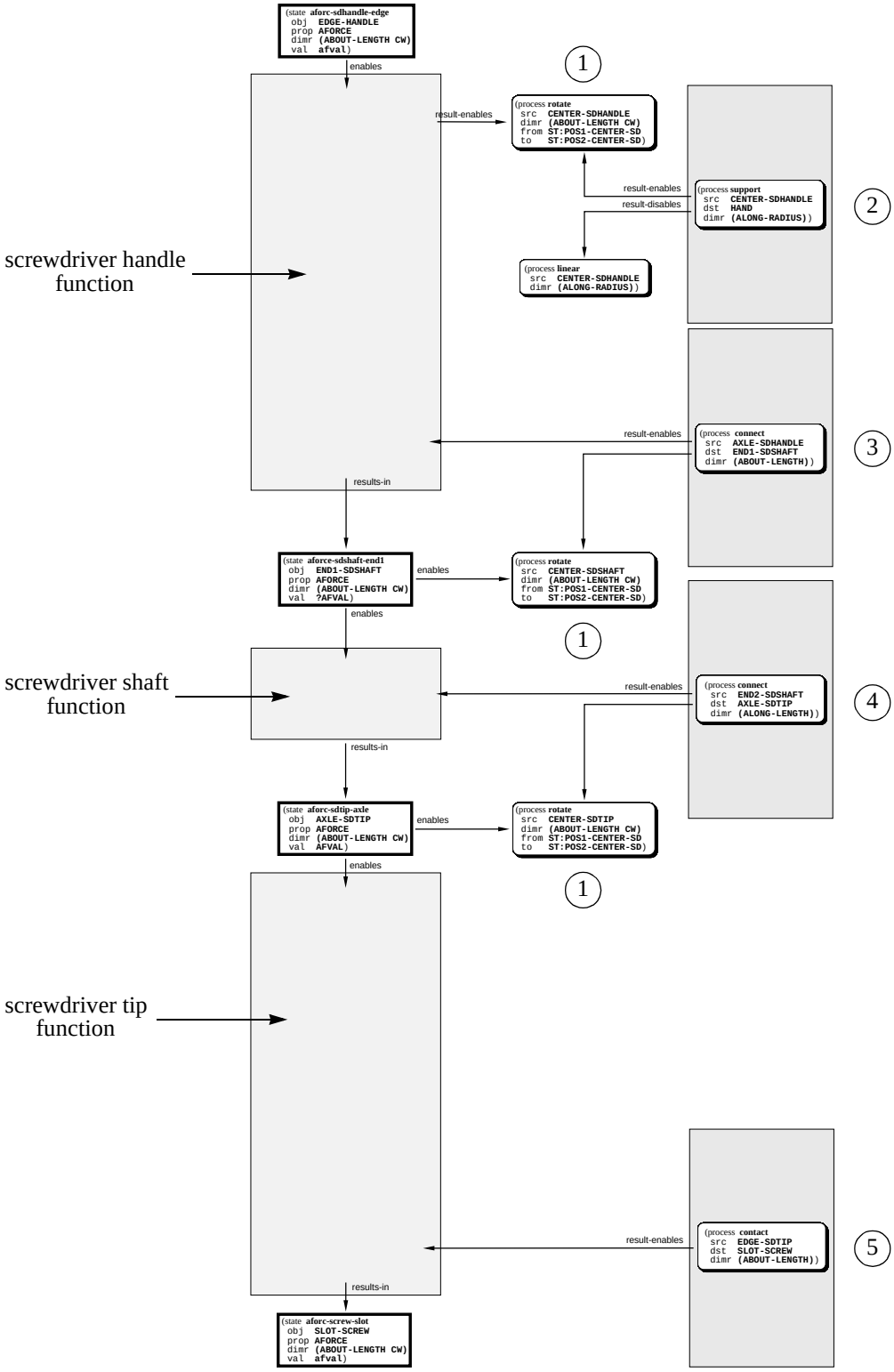


Figure 4.1 Screwdriver components in overall driving function, shown as black boxes.

The boxes representing the components functions describe *intra-object dynamics*, behavior

which may not be observable. The motions at label (1) in Fig. 4.1 describe *inter-object dynamics*, behavior which is observable. Since each of the screwdriver components is rigidly connected (3 and 4) to the others, the enablement of rotation for the handle (2) enables rotation for the other components. The rotations depicted at (1) all represent rotational motion of the device and are illustrated to show that each component enables motion of the screwdriver.

In FONM, device function is represented as a black box, with a frame that describes a BPP sequence by the associated device and by the states which enable and result from the BPP sequence. In Fig. 4.2, the function knowledge structure is depicted on the right (at 1) and the associated BPP sequence is depicted on the left (2).

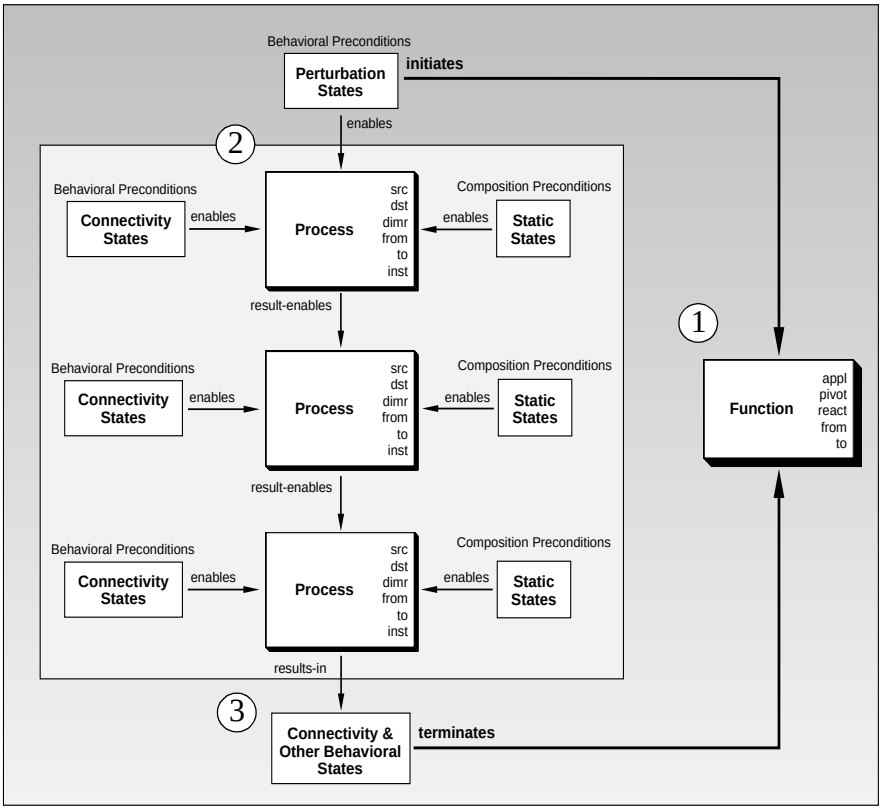


Figure 4.2 The relationship between device function and device behavior

The perturbation at 2 in Fig. 4.2 initiates the function. The connectivity between objects, and the static representation of the objects, enable the propagation of the perturbation states from one process to another. The behavioral state resulting from the BPP sequence (3) terminates the function. The black boxes depicted for screwdriver components in Fig. 4.1 have the same form as the left hand side of Fig. 4.2. Each box has an applied force which enables it, and a state resulting from it. These are identical to the initiating and terminating states of the function. Each box in Fig. 4.1 also has enablements based on the component's connectivity with the other components. In Fig. 4.1, the static enablements are separated from the component functions.

Device function is defined by the initial and final states, and not by the number of objects or processes needed to produce the final state. The function template in Fig. 4.2 is depicted as a three-process BPP sequence. A function can be comprised of a single process,

such as the transmission of force by the screwdriver shaft in Fig. 3.31b (Page 113). A function can also have a number of processes where each is instantiated for the same object. For example, the function associated with slicing tomatoes with a knife can be represented with the BPP sequence RESTRAIN, TRANSFORM, STORE, and DEFORM. All four of the processes in this sequence are instantiated by the same (knife) object.

Device function, as depicted on the right hand side of Fig. 4.2, is represented with a knowledge structure having five roles: appl, react, pivot, from, and to. The function *appl*, *react*, and *pivot* roles describe region locations on the device where the primary interaction takes place. The *from* role describes the applied force which initiates the function, and the *to* role describes the state which terminates the function. As with object behavior, terminating states are local states, associated with the region at which the process takes place. The screwdriver shaft depicted in Fig. 3.31 represents a very simple function called LINKAGE, which describes force transmission and translation from one location to another. The LINKAGE function is shown instantiated for a screwdriver shaft in Fig. 4.3a. Another instantiation of LINKAGE is shown, for a nutcracker handle, in Fig. 4.3b. These device components differ in how the force is applied (rotational in the first case and translation in the second) and how the components are connected to other components (rigid in the first and pivoted in the second). In other respects these components are doing the same thing.

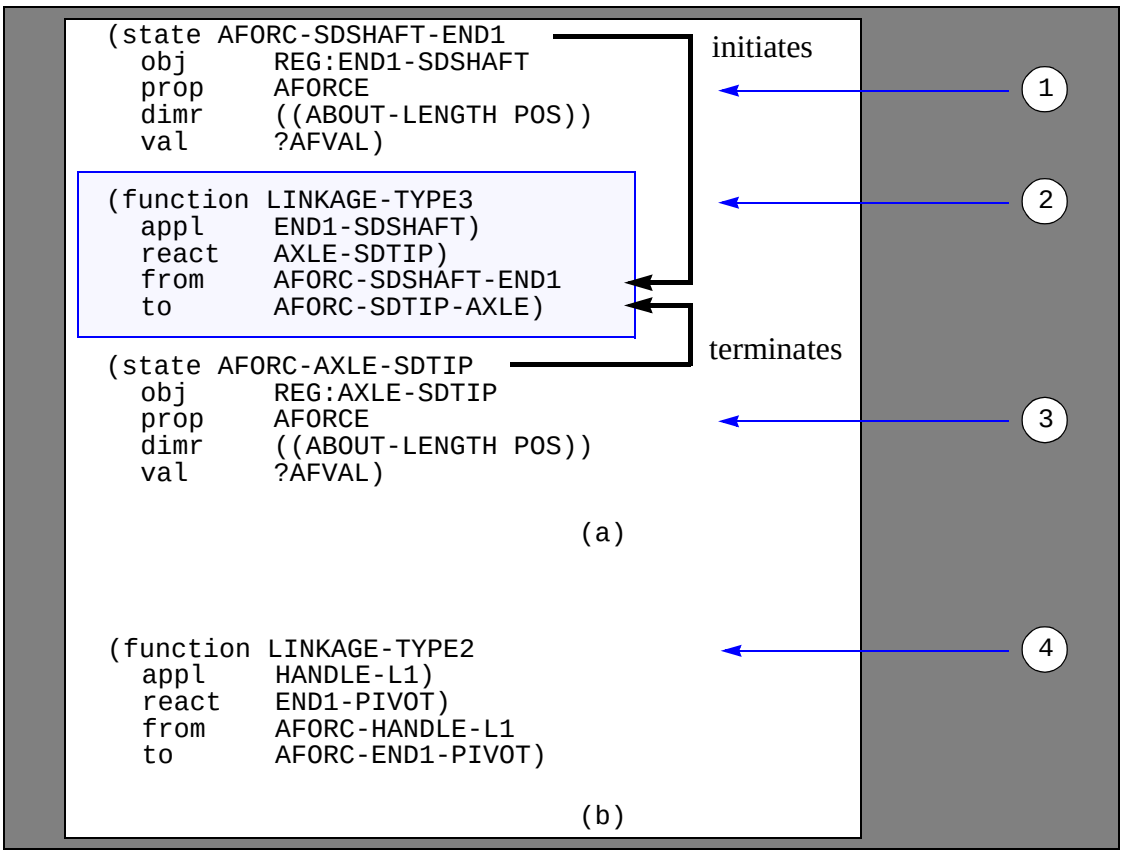


Figure 4.3 Instantiation of the LINKAGE function, (a) for a screwdriver shaft, (b) for a nutcracker handle.

LINKAGE represents the function associated with simple, uniaxial, force transmission: the extension of force from one location to another. In Fig. 4.3a, a rotational force AFORC-SDSHAFT-END1 is applied at region END1-SDSHAFT (1) and reacted at region AXLE-SDTIP (3). The function's *appl* role is filled by END1-SDSHAFT, because this is the loca-

tion where the force is applied to the screwdriver shaft. The *react* role is similarly filled by AXLE-SDTIP, because this is the location where the function's output is produced. The *from* role is filled by the perturbation state AFORC-SDSHAFT-END1, because this is the force that initiates the function. Finally, the *to* role is filled by the state AFORC-AXLE-SDTIP, because this is the state which terminates the function.

4.2 Machine Primitives

LINKAGE is called a *machine primitive* (MP). Machine primitives represent the functions associated with primitive objects, and form a set of building blocks from which the functions of complex devices can be described and reasoned about. For example, LINKAGE represents any object which instantiates the function of uniaxial force transmission, and the associated objects are called linkage objects. There are eleven machine primitives in FONM: MP-LINKAGE, MP-LEVER, MP-WHEEL-AXLE, MP-GEAR, MP-PULLEY, MP-BEARING, MP-PLANE, MP-BLADE, MP-SCREW, MP-SPRING, and MP-CONTAINER. Like LINKAGE, each primitive describes a single function and the class of objects which is commonly associated with that single function. Fig. 4.4 illustrates the relationships between machine primitives and the objects which instantiate them.

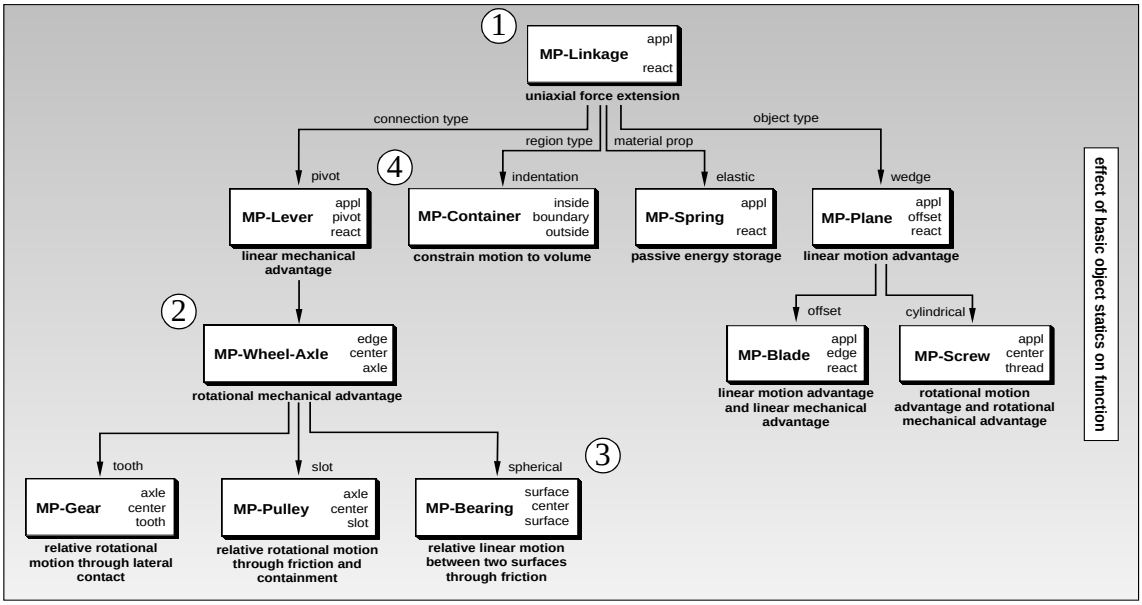


Figure 4.4 Machine primitive hierarchy, specialized by functional similarity and process role instantiation.

Fig. 4.4 is a hierarchy organized by the static differences between object primitives and the function classes they enable. Each box depicts the machine primitive and its role specializations as they instantiate the function appl, pivot, and react roles. For example, MP-LINKAGE (at 1) has only two roles, an appl and a react, which match the function appl and react roles. A stick can be used to extend and probe by transmitting force, and MP-LINKAGE can represent the stick and its function. Any object which can transmit force in at least one dimension and direction can instantiate MP-LINKAGE. The MP-LINKAGE *appl* role is instantiated by one stick end, and the MP-LINKAGE *react* role is instantiated by the other stick end.

MP-LEVER represents the function of magnifying or reducing applied force. The objects which instantiate MP-LEVER differ from those which instantiate MP-LINKAGE by the addition of a pivot, so they instantiate all three appl, pivot, and react function roles. When the force magnification is radial (2), the function/object is represented with MP-WHEEL-AXLE. If the wheel object's react region is toothed, then the function/object is represented with MP-GEAR, which describes motion control as well as force transmission. If the wheel object's react region is slotted, then the function/object is represented with MP-PULLEY, which describe transmission of rotational motion and force over distance. The primitive MP-BEARING (3) is used to represent relative motion between objects. The bearing object transmits force only on its outer surface, so the appl and react regions describe the same surface. The resulting bearing object must be cylindrical or spherical.

A linkage object with an indentation is capable of containment, which is represented with the primitive MP-CONTAINER (4). MP-SPRING is used to represent the function/objects capable of passive energy storage. Any object which is perturbed with force value large enough to enable the process STORE is represented with MP-SPRING.

Wedge shaped linkage objects are used to magnify or reduce the amount of motion in one dimension (and hence force) by transforming it into component motions in multiple dimensions. The wedge shape provides a built-in pivot in the form of the wedge offset angle. These function/object types are represented with MP-PLANE. When a person walks up a gangway, he or she is trading off the horizontal distance travelled in order to make the vertical distance travelled more enjoyable. Like MP-LEVER, the amount of work done remains the same. When the plane object is applied primarily along its edge region, instead of its react region, to enable DEFORM processes, then its function is represented with the primitive MP-BLADE. The plane object can be used linearly or rotationally. When used rotationally, the function/object is represented with MP-SCREW. Table 4.1: organizes the machine primitive classes shown in Fig. 4.4 by their physical characteristics.

Primitive Class	Parent Class	Physical Characteristics/ Parts	I/O Regions
MP-LINKAGE	Phys-Obj	Material: Elastic in dimension	Appl, React
MP-LEVER	MP-LINKAGE	Add Pivot Region	Appl, Pivot, React
MP-WHEEL-AXLE	MP-LEVER	Shape: Circular	Edge, Center, Axle
MP-BEARING	MP-WHEEL-AXLE	Shape: Spherical or Cylindrical	Surface, Center, Surface
MP-GEAR	MP-WHEEL-AXLE	Add Tooth Region; Parts: Gear or Chain	Tooth, Center, Axle
MP-PULLEY	MP-WHEEL-AXLE	Add Slot Region; Parts: Pulley and Belt	Slot, Center, Axle
MP-PLANE	MP-LINKAGE	Shape: Wedge (Offset Angle)	Appl, React
MP-BLADE	MP-PLANE	Attribute: Sharp	Appl, Edge, React

Table 4.1: Machine primitive (MP) classification.

Primitive Class	Parent Class	Physical Characteristics/ Parts	I/O Regions
MP-SCREW	MP-PLANE	Shape: Cylindrical	Appl, Center, Thread
MP-SPRING	MP-LINKAGE	Material: Elastic in dimr	Appl, React
MP-CONTAINER	MP-LINKAGE	Add Indentation Region	Inside, Boundary, Outside

Table 4.1: Machine primitive (MP) classification.

The objects which instantiate machine primitives and their function roles are presented in Table 4.1:. In the table, a parent describes the function class from which a primitive descends. MP-LINKAGE is dependent only on the material properties of an object. If the object is capable of transmitting force in at least the dimension and direction of perturbation, then it is a linkage object.

The physical characteristics and parts in column 3 of Table 4.1: describe the changes to the parent class objects which define the current object class. For example MP-PULLEY is instantiated by a pulley object which consists of a wheel object with a slot on its outer edge and a rope or belt. These physical characteristics are not invariant, but describe general classes which can instantiate the machine primitive. For example, MP-WHEEL-AXLE need not be instantiated by circular objects, as described in the table. The wheel object primitive is generally associated with circular objects, although a crank, which is not circular in general, transmits force radially and can be represented with MP-WHEEL-AXLE. The function enablement takes precedence over the physical appearance of the object. The gear object associated with MP-GEAR is instantiated by a combination of objects. MP-GEAR is enabled by two gears in tooth-contact. Likewise, the primitive MP-PULLEY is enabled for a pulley object which consists of a pulley and belt. Both MP-GEAR and MP-PULLEY describe the transmission of rotational force and motion from the region instantiating their appl role to that instantiating their react role, if the regions are the same. For example, a gear whose axle serves as a bearing, such as a pinion, transmits force from contact at one tooth region to that at another. The same primitives are used to represent force magnification from the region instantiating their appl role to that instantiating their react role if the regions are not the same. A drive shaft and gear are used to reduce force from the axle radius to that of the teeth.

4.2.1 MP-LINKAGE

MP-LINKAGE describes objects capable of, and used for, transmitting force to another object in the dimension and direction of an applied force. MP-LINKAGE is enabled by contact between the linkage object and another object, at the linkage object react region. MP-LINKAGE is initiated by an applied force at the linkage object appl region. An enabled MP-LINKAGE is terminated by a force on the second object at the region where it contacts the linkage object. MP-LINKAGE describes the input and output of the process TRANS-

FORM-TRANSMIT. These relationships are depicted in Fig. 4.5.

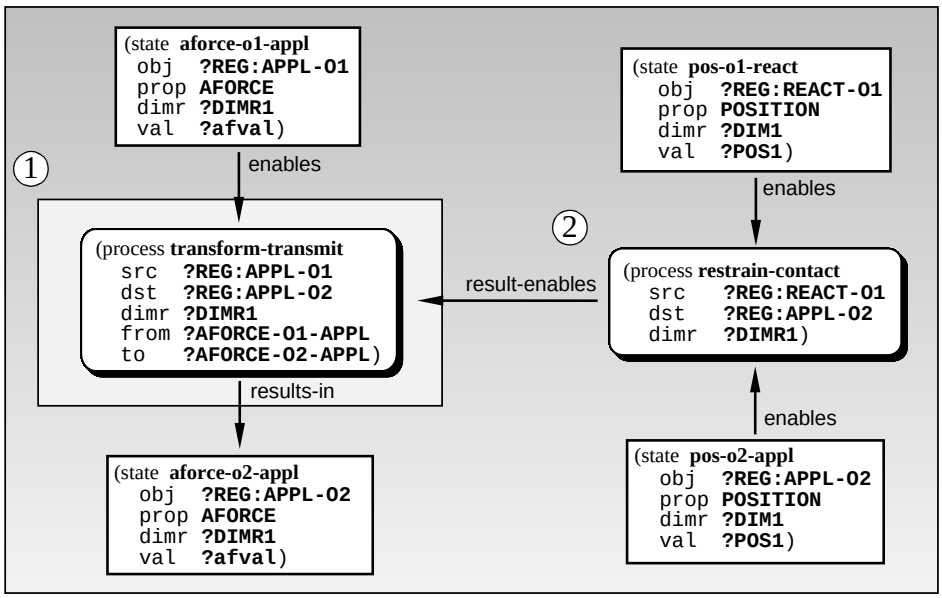


Figure 4.5 The behavioral sequence associated with the function of MP-LINKAGE.

Fig. 4.5 illustrates the behavioral sequence which represents the function associated with MP-LINKAGE. The shaded box (at 1) shows the function's primary process. The entire sequence is represented with two regions on the linkage object. These regions instantiate the appl and react roles of MP-LINKAGE. The appl role is instantiated by the region APPL-O1, and the react role is instantiated by the region REACT-O1. This representation is also shown in box form, instantiated for a stick linkage, in Fig. 4.6. The box labeled MP-LINKAGE (at 1 in Fig. 4.6a) is a shorthand form for the shaded box at (1) in Fig. 4.5. The box is shown with the states which initiate and terminate the function, and with the connectivity enablement between the stick and the amorphous blob depicted in Fig. 4.6a (at 2). The slot-

filler notation associated with the box (at 1 in Fig. 4.6a) is presented in Fig. 4.6b .

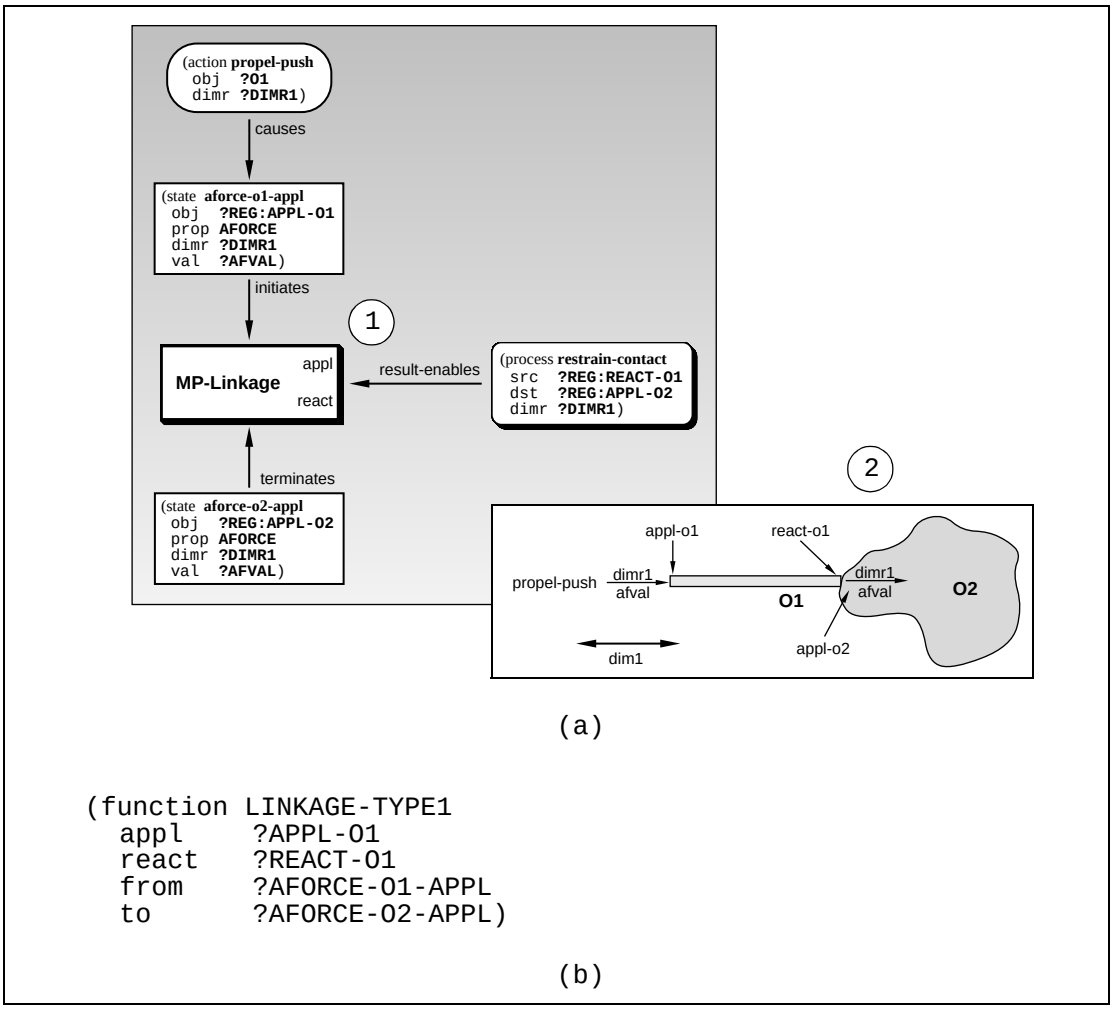


Figure 4.6 Block version of LINKAGE function, linkage object statics, and the associated slot-filler representation.

The graphic illustrates a stick linkage object (O1) being used to probe a general object (O2). The box labeled MP-LINKAGE (at 1) organizes the enablements, initiating, and terminating conditions associated with the sequence in Fig. 4.5. The process RESTRAIN-CONTACT between the stick (O1) and O2 enables the function. The function is shown being initiated by the force AFORCE-O1-APPL. The force is produced by pushing (PROPEL-PUSH) the stick. This causes a force at APPL-O1, which is transmitted internally to all contact locations. The state AFORCE-REACT-O1 is subsequently implied. The resulting state AFORCE-APPL-O2, which represents the force transmitted to O2, terminates MP-LINKAGE.

Machine primitives can be combined together as long as there are no discrepancies between their enabling conditions. The box used to depict the MP-LINKAGE in Fig. 4.6 can be combined to describe more complex devices. For example, consider the four compo-

nents of a nutcracker, as depicted in Fig. 4.7.

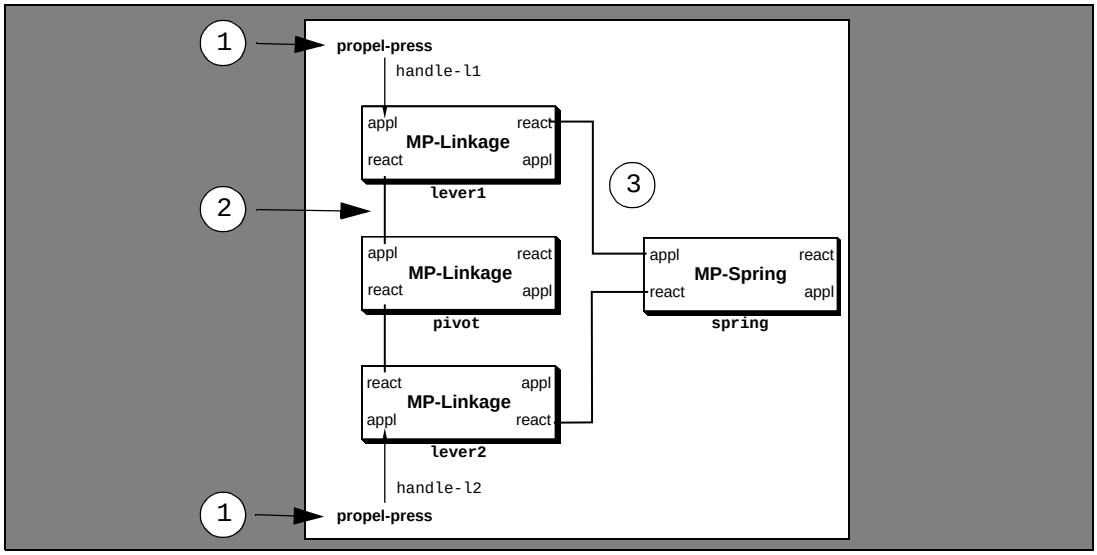


Figure 4.7 A partial MP diagram for a nutcracker without a nut.

Fig. 4.7 diagrams a nutcracker function. The blocks in the figure represent the functions which are enabled by the components of the nutcracker in the context of the action (at 1). In this figure, the components of the nutcracker are labeled below the block. Each component of the nutcracker instantiates the roles of a machine primitive. The figure shows only the blocks associated with the primitives. The actions applied during nutcracker operation are illustrated as thin lines with arrows. The solid links between the blocks (e.g., at 2) denote the static connection relations (inter-object dynamics) between the objects which instantiate the primitives. In the nutcracker, these relations are instantiated RESTRAIN-PIVOT (2) and RESTRAIN-CONTACT (3) processes. The states which initiate and terminate each function are assumed to be effected through these links. That is, the links represent where motion and force flow through the device. Each block has the primitive roles duplicated for ease in producing the drawing. Notice that the react role of the topmost MP-LINKAGE has two links emanating from it. When a link is associated with each region of the same primitive it simply means that the region engages in both interactions. In this case, the force which is reacted at the connection between the LEVER1 and the PIVOT is also reacted at the connection between the LEVER1 and the SPRING.

MP-LINKAGE is used to represent the nutcracker levers because, alone, they can only be used to transmit force. When a force is applied to the components of the nutcracker, as with the actions depicted at (1) in Fig. 4.7, it propagates along the links, initiating the respective machine primitive. In this example, when HANDLE-L1 and HANDLE-L2 are pressed, an applied force is generated in both LEVER1 and LEVER2. The linkage objects are connected at their other ends, to PIVOT, so the force is reacted (TRANSFORM-TRANSLATE) at PIVOT. When the local forces on PIVOT are considered together, the transmitted forces cancel. However, the local force is unbalanced at the handle regions, so the linkages are free to move (MOTION-ROTATE) about the pivot. The relative motion enables SPRING (STORE-COMPRESS) on SPRING, compressing the spring. Interobject motion has not been depicted in the figure.

Fig. 4.7 is called an *MP diagram* and is used to show how machine primitives interact to describe the function of arbitrary devices. Fig. 4.7 shows a single MP diagram for the

nutcracker device. Whereas a device has a single static representation, it has a unique MP diagram for each function it can instantiate.

There are three MP-LINKAGE class specializations in FONM: longitudinal, lateral, and torsional. Each is distinguished by the type of force applied.

LINKAGE-TYPE1 (*longitudinal*): The applied force is reacted, in the linkage object, in the same dimension shared by the appl and react linkage object regions. For example, the stick depicted in Fig. 4.5, has the force applied at APPL-O1 and reacted at REACT-O1. Both regions lie along the length of O1, so they share the dimension ALONG-LENGTH. As long as the force dimension is ALONG-LENGTH, the LINKAGE is longitudinal.

LINKAGE-TYPE2 (*lateral*): Describes the function of translating force. The perturbation force is reacted in a dimension not shared by the appl and react linkage-object regions.

LINKAGE-TYPE3 (*torsional*): The perturbation force is rotational rather than translational.

The nutcracker levers shown in Fig. 4.7 are represented as LINKAGE-LATERAL, since the force is applied in a dimension other than the one commonly shared by the lever regions.

4.2.2 MP-LEVER

MP-LEVER describes devices used to produce in-plane mechanical advantage by enabling the process TRANSFORM-MAGNIFY. Lever objects have regions which instantiate each of the appl, pivot, and react device function roles. MP-LEVER is enabled by two contacts between the lever object and other objects: one at the lever object region that instantiates the function pivot role, and the other at the lever object region that instantiates the function react role. MP-LEVER is initiated by an applied force at the lever object region which instantiates the function appl role. An enabled MP-LEVER is terminated by a force on the object in contact with lever object. MP-LEVER describes the enablements and resulting state of the process TRANSFORM-MAGNIFY, so the value of the force is magnified.

These relationships are depicted in Fig. 4.8.

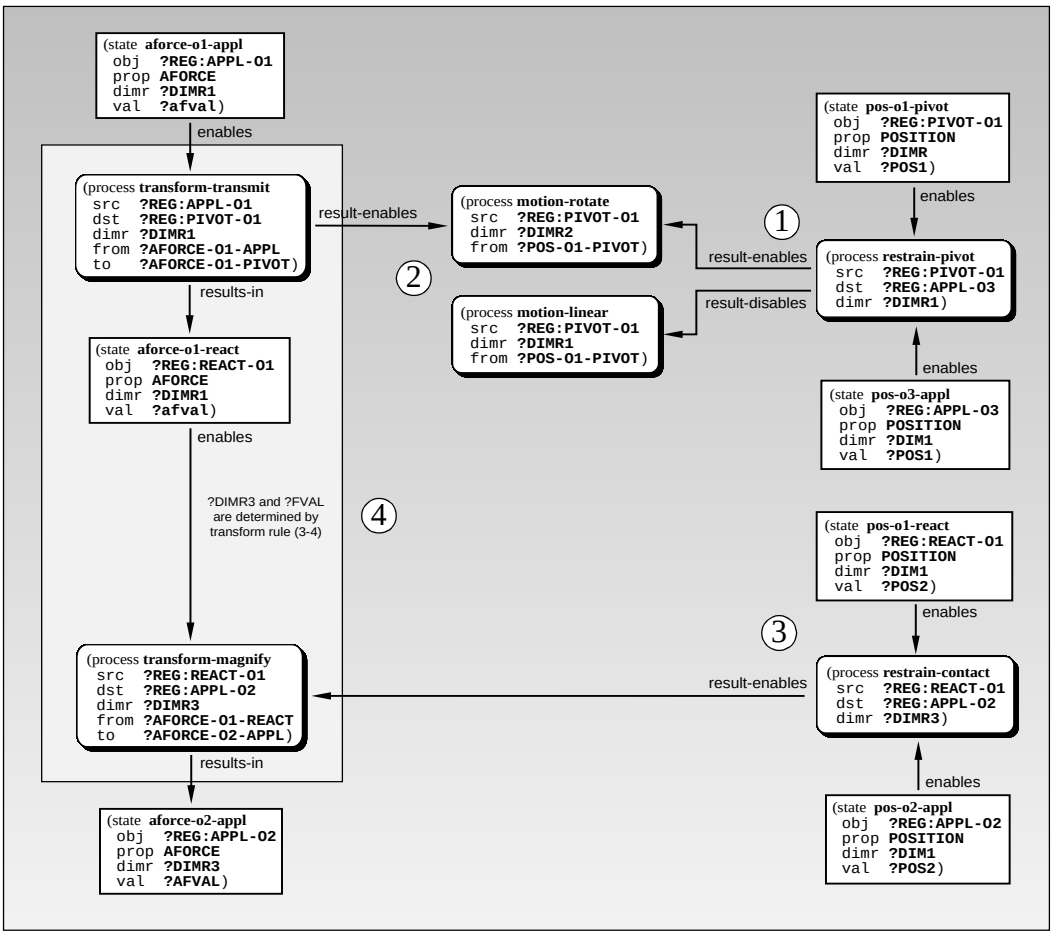


Figure 4.8 Process sequence associated with the LEVER machine primitive.

Fig. 4.8 depicts the sequence of behavioral processes which enable and describe MP-LEVER. The RESTRAIN-PIVOT process (1) enables MOTION-ROTATE and disables MOTION-LINEAR at the contact region of the lever object (2). The RESTRAIN-CONTACT process (3) provides the alternate force path which enables TRANSFORM-MAGNIFY, and the TRANSFORM rule TR-4 is used to determine the value of the magnified force. The shaded box on the left represents the intra-object dynamics of MP-LEVER. The representation for MP-LEVER in Fig. 4.8 is depicted in block form and slot-filler notation in Fig.

4.9.

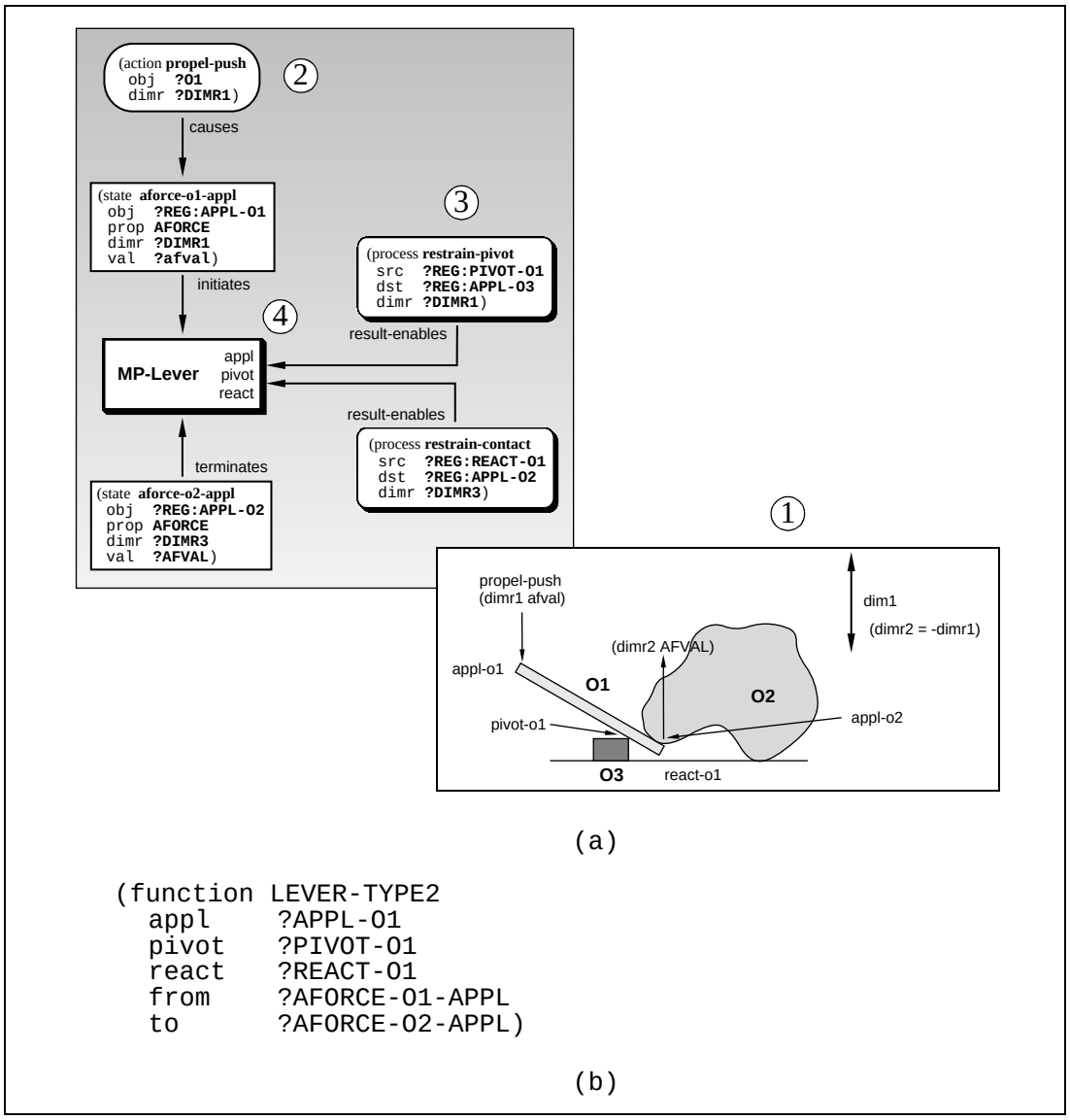


Figure 4.9 Block version of MP-LEVER function, lever object statics, and slot-filler representation

The graphic in Fig. 4.9a (at 1) illustrates a two-by-four (O1) and brick (O3) being used to lift a potato-shaped object (O2). O1 is propped up by an object O3, allowing O1 to rotate about the location PIVOT-O1 but disabling downward translational motion. This is a fulcrum pivot. The shaded portion of Fig. 4.9a illustrates the instantiation of MP-LEVER (shown as a block at 4) for the objects depicted in the figure. The action PROPEL-PUSH (2) causes the force AFORCE-O1-APPL which initiates MP-LEVER. The pivot (3) and contact processes defined by O1-O3 statics enable MP-LEVER, and the force AFORCE-O2-APPL terminates MP-LEVER.

Dimensions and Pivots

MP-LEVER causes *in-plane* force or motion magnification as a trade-off with the distance

the lever object moves. MP-LEVER is enabled by a restraint (caused by RESTRAIN-PIVOT) which disables linear motion in a translational dimension and enables rotational motion in a rotational dimension. Determination of the dimensions of motion disablement/enablement is thus an important consideration in representing MP-LEVER. Determining the translational dimension is straightforward: the pivot must restrict lever object motion in the dimension and direction of the applied force. In Fig. 4.9a, the applied force is in DIMR1 so the pivot restricts O1 linear motion in DIMR1. In MP-LEVER, the applied force is a lateral force, meaning that its dimension is not the dimension shared by the appl and pivot regions. Dimension *sharing* means that the two regions have the same locations values in one dimension but different location values on another. They share the second dimension. For example, if the forced dimension is the O1 ALONG-DEPTH dimension, then the dimension shared by the appl and pivot regions is ALONG-LENGTH. These two dimensions determine the plane of motion for the lever object. The rotational dimension is about the third (or out-of-plane) translational dimension. In the previous example, the rotational dimension is ABOUT-WIDTH. If the lever object appl and react regions share the same translational dimension as the appl and pivot regions, then the magnification is in-plane.

Instantiated Machine Primitives

The context of a device function can change without changing how the force is applied. In such cases the device is represented with a new function, but the function is a specialization of the former function. For example, when the nutcracker handles were pressed together in the example illustrated in Fig. 4.7, the device was represented as a pivoted combination of MP-LINKAGES. When a nut is placed between the levers, however, the second contact location enables TRANSFORM-MAGNIFY and the device can be represented as a combination of MP-LEVERs. The pivot region is defined by the connection between the nutcracker levers (LEVER1 and LEVER2) and the pivot (PIVOT). The pivot region of each nutcracker lever is located at the linkage end regions instead of somewhere along them. However, this only affects the direction of magnified force and not the concept of "leverness." Fig. 4.10 illustrates how the MP-Diagram of Fig. 4.7 is modified by the addition of the new contact between the nutcracker jaws and the nut.

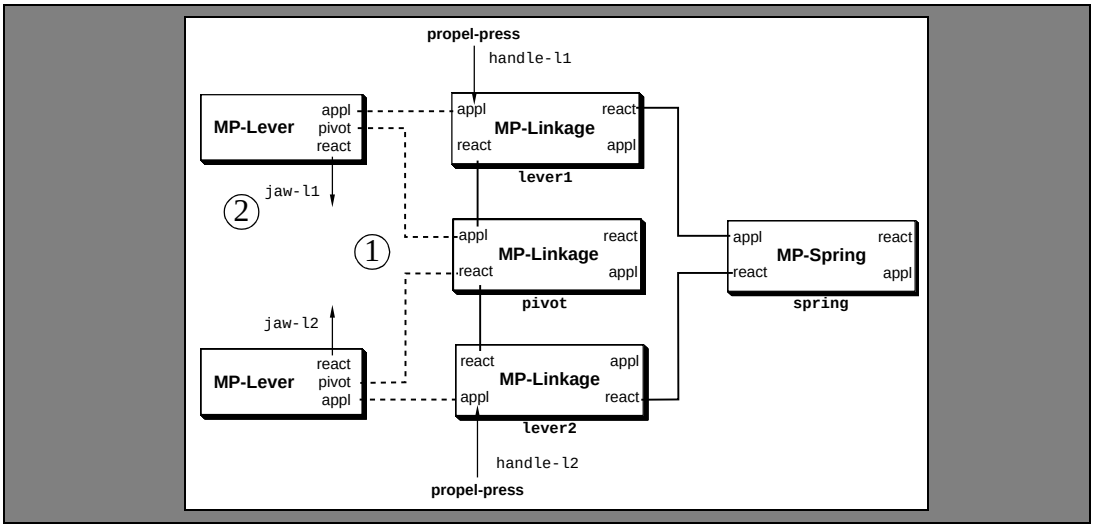


Figure 4.10 The full nutcracker MP diagram.

The LEVER1 region which instantiated the MP-LINKAGE appl role now instantiates the

MP-LEVER appl role. The linkage object react region is still connected to the PIVOT, which still instantiates an MP-LINKAGE, but the PIVOT region which instantiated the MP-LINKAGE appl role now instantiated the MP-LEVER pivot role (1). Finally, the JAW-L1 region (2) now instantiates the MP-LEVER react role. The nutcracker SPRING continues to instantiate MP-SPRING. The dotted lines in Fig. 4.10 (1) mean that the regions are the same rather than that they are connected. Thus the instantiation of MP-LEVER requires no different objects, just the addition of a contact location.

Lever Classes

The introduction of a second type of lever object, one in which the react region is located between the appl region and the pivot region, without adversely affecting the notion of 'leverness,' suggests an MP-LEVER class hierarchy differentiated by how and where force is applied, pivoted and reacted. There are three specialization classes of MP-LEVER, each defined by the relative location of the lever object's appl, react, and pivot regions: (a) teeter-totter-like, (b) wheel-barrow-like, and (c) crane-like. These lever types are commonly referred to as Type 1, Type 2, and Type 3 levers, respectively, in the literature (e.g., [Macaulay 1988]), as illustrated in Fig. 4.11.

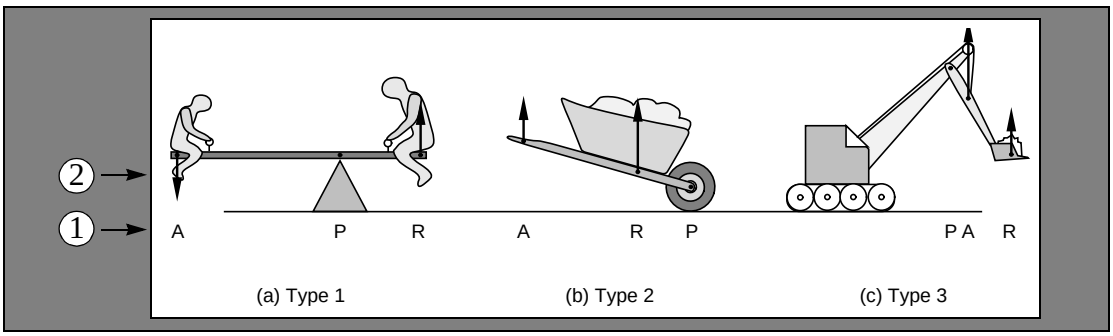


Figure 4.11 Three MP-LEVER classes: (a) a Type 1 lever, (b) a Type 2 lever, and (c) a Type 3 lever.

LEVER-TYPE1 (*Teeter-Totter*): Applied and reacted forces are located on opposite sides of the pivot region. Forcing regions move in opposite directions. Force is magnified.

LEVER-TYPE2 (*Wheel-Barrow*): Reacted force is located between the applied force and the pivot. Both forcing regions move in the same direction. Force is magnified.

LEVER-TYPE3 (*Crane*): Applied force is located between the reacted force and the pivot. Both forcing regions move in the same direction. Motion is magnified.

Fig. 4.11 depicts three different kinds of levers. The first two (a and b) both magnify force but differ in the direction of motion at the lever object react region. The A, P, and R labels beneath each device in Fig. 4.11 illustrates the relative positions of the appl, pivot, and react regions in each lever class, respectively (at 1). In (c), the use of the machine dramatically changes, since (c) has a force magnification less than unity. However, as in a human arm, the advantage gained with this design favors motion rather than force. Note that the only component of the crane used in this example is the last section, which is pivoted at the end and lifted from the center. The arrows shown in Fig. 4.11 (at 2) represent the applied and reacted forces at the appl and react region locations for their respective devices. The arrow length provides a measure of relative force magnitudes. The resultant force (dimension and value) is related to the applied force by comparing the relative appl-to-pivot and react-to-

pivot lengths. In FONM, the result can be calculated explicitly if the lengths are known, using transform rule 4 (TR-4), which was presented as relation (3-4). Otherwise, the result takes one of three values: less than nominal, nominal, or greater than nominal (<NOM, NOM, >NOM), where nominal is the applied force value. For example, if AP is the length from the appl region to the pivot region, and PR is the length from the react region to the pivot region, then the ratio AP:RP in Fig. 4.11a is >NOM. The reacted force value is greater than the applied force value. The direction is reversed. The same is true of both Type 1 and Type 2 levers, and the opposite is true of Type 3 levers. These relationships are described by three rules for levers in Table 4.2:

Lever Rule	Lever Class	Resultant Force Relation/Value	Resultant Force Direction	Resultant Distance Value
LR-1	Type 1 (a p r)	TR-4	-DIMR1	TR-4
LR-2	Type 2 (a r p)	TR-4, >NOM	DIMR1	<NOM
LR-3	Type 3 (r a p)	TR-4, <NOM	DIMR1	>NOM

Table 4.2 Rules for determining force and motion in levers (and wheels).

The resultant force and distance entries for LR-1 in Table 4.2 can only be determined by knowing the location of the appl and react regions (relative or absolute), since the values can be <NOM, NOM, or >NOM depending on the locations. In LR-2 and LR-3 the relative locations are fixed so the resultant values are known before apriori.

4.2.3 MP-WHEEL-AXLE

MP-WHEEL-AXLE describes rotational force magnification. MP-WHEEL-AXLE is identical to MP-LEVER except that the applied force dimension is rotational. The wheel object has an edge region which instantiates the MP-LEVER appl role, a center region which instantiates the MP-LEVER pivot role, and an axle region which instantiates the MP-LEVER react role. MP-WHEEL-AXLE is enabled by a RESTRAIN-PIVOT for the center region, and a RESTRAIN-CONTACT between the wheel object axle region and the affected object. The restraint state resulting from the pivot enables rotational motion about the wheel object's center. The wheel object is often (but not invariably) circular, but the wheel object motion is always circular. For example, the crank on an egg beater is a non-circular wheel object, but is turned about its axle center and transforms force from the handle to the axle. MP-WHEEL-AXLE is initiated by a force at the wheel object edge region. An enabled MP-WHEEL-AXLE is terminated by a force on the object in contact with the wheel object axle region. MP-WHEEL-AXLE describes the enablements and resulting state of the process TRANSFORM-MAGNIFY, so the value of the perturbation force is magnified. The behavioral sequences which describe the function of MP-WHEEL-AXLE are depicted in Fig.

4.12.

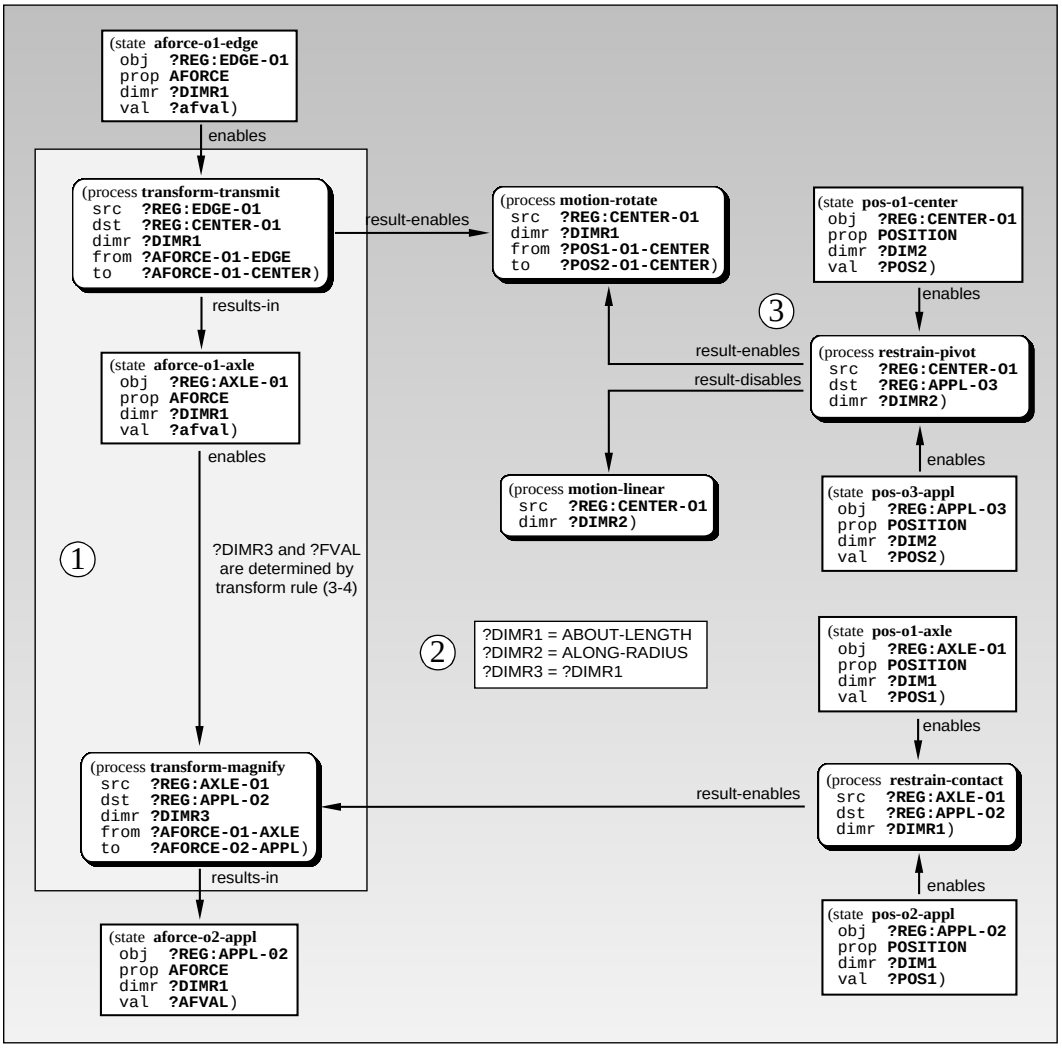


Figure 4.12 Behavioral process sequence associated with the WHEEl-AXLE machine primitive.

In Fig. 4.12, the shaded box (1) represents the intra-object dynamics of the wheel object. The figure is very similar to Fig. 4.5 (MP-LEVER), the differences between the two primitives are described in the inset box at (2). The wheel function dimensions are rotational and radial rather than translational. The representation for MP-WHEEL-AXLE in Fig. 4.12 is depicted in block form at (1) in Fig. 4.13a, and in slot-filler notation in Fig. 4.13b. Fig.

4.13a (2) illustrates a screwdriver (O1) driving a screw (O2).

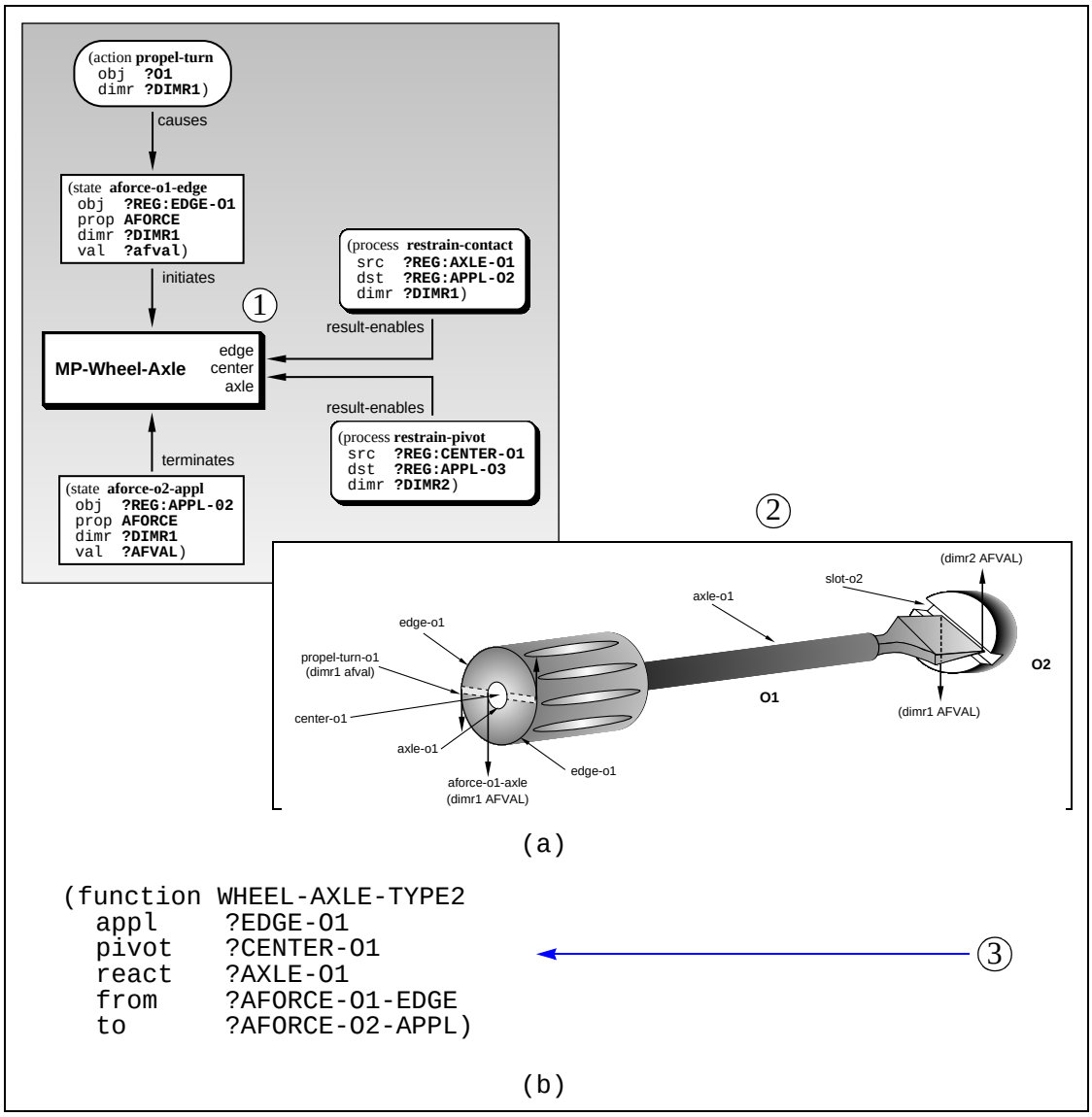


Figure 4.13 Block version of MP-WHEEL-AXLE function, wheel object statics, and associated slot-filler representation for a the Type-2 wheel illustrated

The screwdriver illustrated in Fig. 4.13 is considered a generalized wheel object rather than three individual components. The O1 center region (CENTER-O1) is supported by the user (O3), but not illustrated in the figure. When a force is applied at the screwdriver handle (i.e., outer radius), as a torque or moment, it is magnified at the axle radius. This force is translated along the axle length to the tip, where it is again magnified to the reaction location. The applied force dimension and direction at handle and tip are the same.

The upper portion of Fig. 4.13a illustrates the static preconditions (RESTRAIN-CONTACT and RESTRAIN-SUPPORT) which enable MP-WHEEL-AXLE. The process RESTRAIN-CONTACT between the wheel object (O1) and the screw object (O2) enables the process TRANSFORM-TRANSMIT to the screw. The process RESTRAIN-SUPPORT between O1 and O3 enables the process TRANSFORM-TRANSMIT to the supporting ob-

ject O3. Note that in Fig. 4.12 (3) the connection precondition is a RESTRAIN-PIVOT class, whereas in this example (1) it is a RESTRAIN-SUPPORT class. These are not different. RESTRAIN-PIVOT is a subclass of RESTRAIN-SUPPORT specialized to enable rotation. When the user is holding the screwdriver rotation is still enabled.

MP-WHEEL-AXLE is shown being initiated by the force AFORCE-O1-EDGE, produced in this instance by turning (PROPEL-TURN) the screwdriver handle (region EDGE-O1). The state AFORCE-O2-APPL, which represents the force transmitted to, and magnified at, APPL-O2, terminates MP-WHEEL-AXLE.

Fig. 4.13b shows the mapping between the roles of machine primitives and the function class. The function appl, pivot, and react roles are shown instantiated by the wheel object edge, center, and axle roles.

The screwdriver in Fig. 4.13 magnifies force from an outer radius to an inner radius, but the traditional notion of a wheel is one of rolling and motion transmission on it edges. Both are wheel objects, but the location of the react region on the traditional wheel is the same as the appl region so there is no force magnification. Fig. 4.14 illustrates rotational translation and magnification in wheels perturbed on their edges.

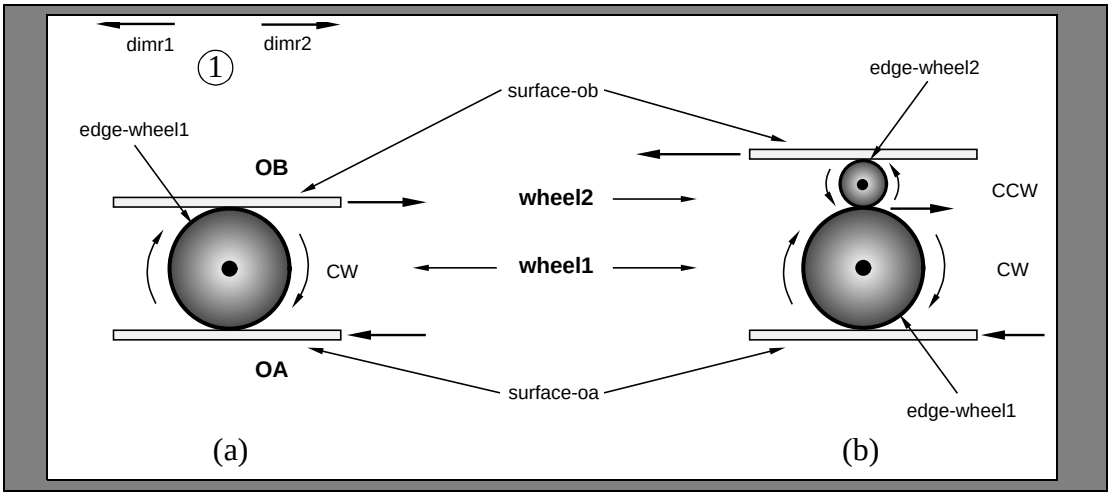


Figure 4.14 TRANSFORM processes in wheels.

Fig. 4.14a depicts a wheel (WHEEL1) in surface contact with two objects: at SURFACE-OA, and SURFACE-OB, respectively. SURFACE-OA moves in dimension/direction DIMR1 (1). The radial distance is the same from the center region of WHEEL1 (not illustrated) to any point on the edge. Applying lever rule LR-1, the resultant force value is the same (NOM) as the applied force, and the resultant force dimension is the opposite of the applied force (DIMR2 = -DIMR1), as noted in Table 4.2, Page 162:

Wheel Rule	Wheel Class	Resultant Force Value	Resultant Force Direction	Resultant Distance Value
WR-1	Type 1 (a p r)	LR-1, NOM	-DIMR1	LR-1, NOM
WR-2	Type 2 (a r p)	LR-2, >NOM	DIMR1	<NOM

Table 4.3 Rules for determining force and motion in single wheels.

Wheel Rule	Wheel Class	Resultant Force Value	Resultant Force Direction	Resultant Distance Value
WR-3	Type 3 (r a p)	LR-3, <NOM	DIMR1	>NOM

Table 4.3 Rules for determining force and motion in single wheels.

The result is force translation to SURFACE-OB by WHEEL1. WHEEL1 instantiates a wheel object, and MP-WHEEL-AXLE enables motion at SURFACE-OB in dimension/direction DIMR2. Rules WR-2 and WR-3 in Table 4.2 are repeated versions of LR-2 and LR-2 in Table 4.2.

The wheel in Fig. 4.14a translates force and motion. The wheels in Fig. 4.14b magnify force and motion. The motion applied to SURFACE-OA is transmitted directly to WHEEL1 in both examples. On the left, WHEEL1 is in direct contact with SURFACE-OB, so there is no mechanical advantage, but the direction of motion changes. On the right, WHEEL1 is in contact with a smaller wheel (WHEEL2) at region EDGE-WHEEL1. WHEEL2 is also in contact with SURFACE-OB at EDGE-WHEEL2. The direction of motion at the EDGE-WHEEL1/EDGE-WHEEL2 contact is DIMR2, so (applying LR-1 again) the direction of motion at SURFACE-OB changes back to DIMR1. In order to maintain contact with WHEEL1, WHEEL2 rotates faster, since WHEEL2 will rotate more than once for every rotation of WHEEL1. Mechanical advantage in wheel combinations is defined by the relative ratio of wheel radii, as represented in WR-1:

WR-4: dist(wh2) = dist(wh1) * (r1/r2) (4-1)

where dist is a relative measure of the amount of distance moved with respect to the wheel's own radius, wh2 is WHEEL2, wh1 is WHEEL1, r1 is the radius of WHEEL1, r2 is the radius of WHEEL2, and r1/r2 is the *radius ratio* of the two wheels. WR-4 says that if the radius of the second wheel is smaller than the radius of the first wheel, then the radius ratio is greater than unity, and the distance moved by a surface region on the second wheel will be greater (with respect to its radius) than the corresponding surface region on the first wheel. In qualitative values, the radius ratio of a larger wheel to a smaller wheel is >NOM, where NOM is the speed of the smaller wheel, so the motion at SURFACE-OB is >NOM. The direction of motion also changes with wheel combinations, since the applied force is on different 'sides' of the pivot for the different wheels. Rule WR-5 represents the change in direction.

WR-5: dimr(wh2) = -dimr(wh1) (4-2)

where dimr is the dimension and direction of motion, wh2 is the second wheel, and wh1 is the first wheel. WR-4 and WR-5 have the same form as rules TR-4, the difference being that the wheel rules apply to wheel compositions rather than single wheels. Gear and pulley combinations also produce mechanical advantage and motion direction changes the same way.

Wheel Classes

The screwdriver and wheel MP-WHEEL-AXLE examples illustrate two types of wheel function. As with MP-LEVER, there are three corresponding MP-WHEEL-AXLE specializations, defined by the relative location of the wheel object regions which instantiate the MP-WHEEL-AXLE edge, axle, and center roles: (a) gear-like, (b) screwdriver-like, and (c)

drive-shaft-like, as shown in Fig. 4.15. Type 1 MP-WHEEL-AXLE describes traditional wheel motion and force transmission along the wheel object edge, while Type 2 and Type 3 MP-WHEEL-AXLEs describe radial (outer-inner, inner-outer) force magnifications.

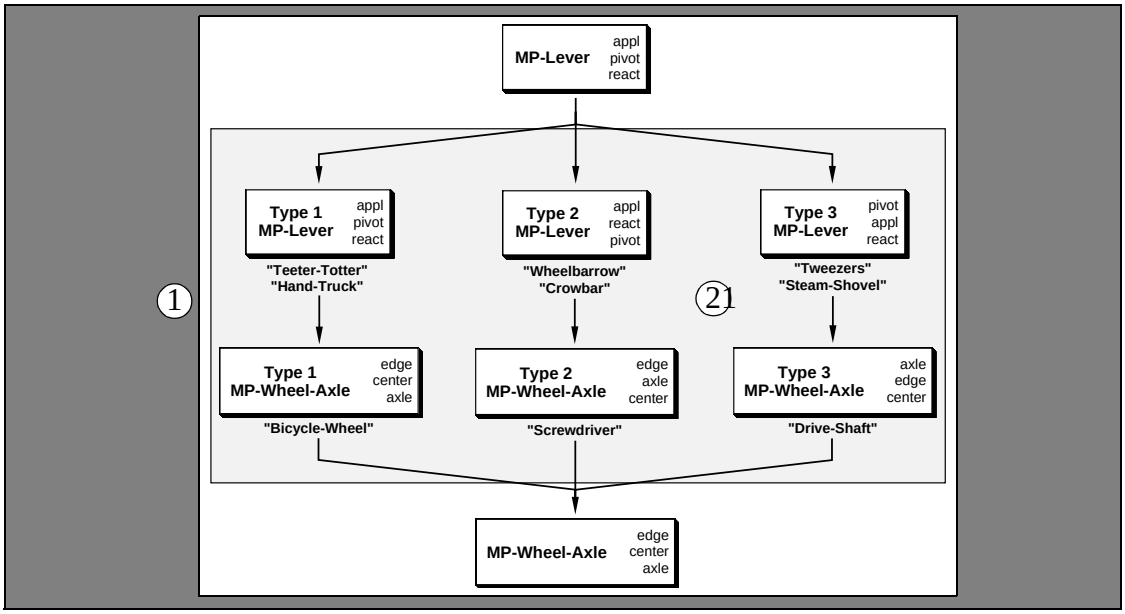


Figure 4.15 MP-LEVER and MP-WHEEL-AXLE classes.

Fig. 4.15 shows the correspondence between MP-LEVER and MP-WHEEL-AXLE classes. MP-WHEEL-AXLE also has three classes, referred to as Type 1, Type 2, and Type 3 wheels. Table 4.1: defined MP-WHEEL-AXLE as an MP-LEVER applied to a circularly shaped object. In this capacity, an MP-WHEEL-AXLE is specialized from a Type 1 MP-LEVER (1), as is the case for other *free-axle* devices: WHEEL, PULLEY, and GEAR combinations. MP-WHEEL-AXLE also describes *fixed-axle* leverage from outer-to-inner radius or vice-versa, corresponding to Type 2 and Type 3 MP-LEVERs (2). The wheel types depicted in Fig. 4.15 are illustrated below in Fig. 4.16.

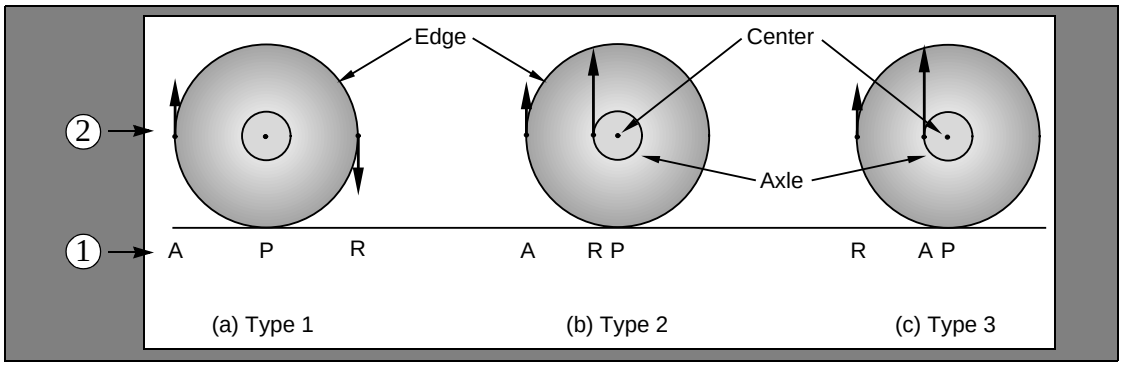


Figure 4.16 Three wheel device classes: (a) a Type 1 wheel, (b) a Type 2 wheel, and (c) a Type 3 wheel.

MP-WHEEL-AXLE-TYPE1 (*Gear*): Applied and reacted forces are both edge regions, on opposite sides of the center. Forcing regions move in opposite directions. Force is unchanged.

MP-WHEEL-AXLE-TYPE2 (*Screwdriver*): Force is applied at the edge region and reacted at the axle region, along the axle. Force dimension and direction is the same. Force value is magnified.

MP-WHEEL-AXLE-TYPE3 (*Drive Shaft*): Applied force is located at the axle region, and reacted at the edge region. Both forcing regions move in the same direction. Motion is magnified.

The A, P, and R beneath each device in Fig. 4.16 (1) represent the relative positions of wheel object edge, center, and axle regions with respect to the lever object function appl, pivot and react locations, respectively.

In Fig. 4.16, the arrows are shown at the edge and axle region locations for their respective objects (2). The arrow length provides a measure of relative force magnitude values. The value of resultant force is related to the applied force by comparing the relative edge-to-center and axle-to-center lengths, as with lever specializations. The resultant forces and distances are also identical to the lever classes identified in Fig. 4.2 except for Type 1 wheels. Type 1 wheels result in a NOM reacted force value because the fixed radii forces the edge-to-center and axle-to-center lengths to be equivalent.

4.2.4 MP-GEAR

MP-GEAR describes devices used to control the transformation of rotational force and motion by lateral interference between gear teeth. A gear object is a specialized wheel object with a toothed edge region, located at the wheel object surface1 and surface2 regions. The tooth region is a protuberance, so it can catch on objects as the gear rotates. MP-GEAR has tooth, center, and axle function roles. MP-GEAR is enabled by a RESTRAIN-PIVOT for the gear object region which instantiates the MP-GEAR center role, and a RESTRAIN-CONTACT between the gear object axle region and a second object. MP-GEAR is initiated by a force at the gear object axle region. An enabled MP-GEAR is terminated by a force state on the object in contact with the gear object tooth region. The teeth of both objects are the same size and shape. MP-GEAR describes the enablements and resulting state of the process TRANSFORM-MAGNIFY, so the value of the applied force is magnified. This relationship is depicted in Fig. 4.17, which illustrates the behavioral processes which describe MP-GEAR function. The shaded box represents the intra-object dynamics of the gear object. The static enablements for MP-GEAR are shown on the right of the box (a RESTRAIN-PIVOT at label 1 and a RESTRAIN-CONTACT at 2). The force (AFORCE-GEAR1-AXLE) which enables the TRANSFORM-MAGNIFY is applied at the gear object region which instantiates the MP-GEAR center role. The applied force and the pivoted center (1) enable rotation of the gear object. The contact with the second gear object (2) enables TRANSFORM-TRANSMIT to the tooth of the second gear (AFORCE-GEAR2-TOOTH1), which is the resulting state of MP-GEAR. The figure also illustrates the dynam-

ics of the second gear object.

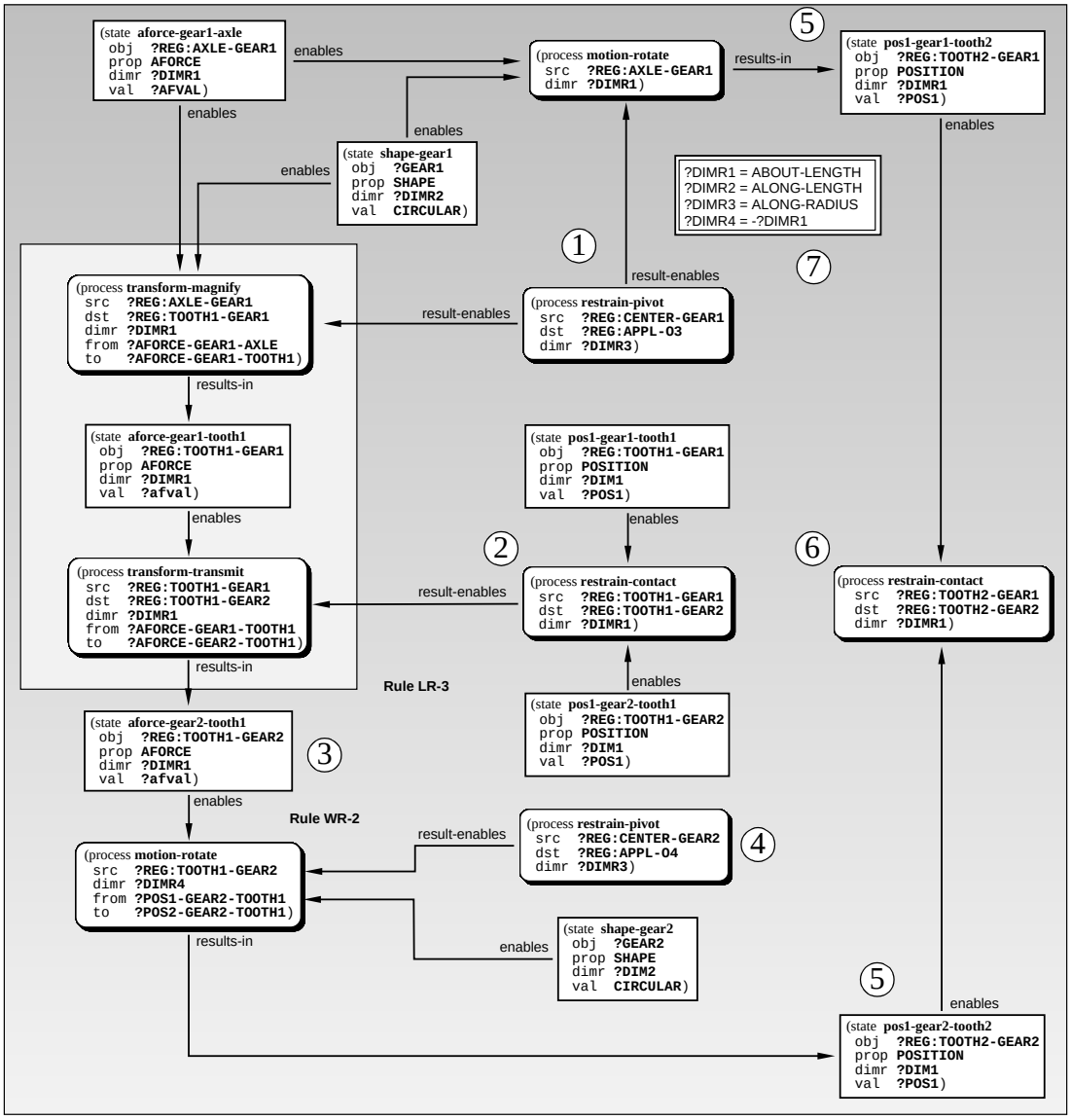


Figure 4.17 Behavioral process sequence associated with the GEAR machine primitive.

The force transmitted from the first gear (3), and the pivoted support of the second gear (4) enable its rotation, and the two gear objects advance by one tooth each (5). The advance places a second tooth of each gear in contact (6) at the same position which the first tooth of each gear were originally positioned. The process RESTRAIN-PIVOT between GEAR1 and O3, and between GEAR2 and O4, enables the process TRANSFORM-TRANSMIT to the supporting object (O3 and O4). These relationships have not been illustrated in the figure.

The representation for MP-GEAR in Fig. 4.17 is illustrated graphically at (2) in Fig. 4.18a. The machine primitive associated with the shaded box in Fig. 4.17 is also replaced with its

block equivalent at (1) in Fig. 4.18a.

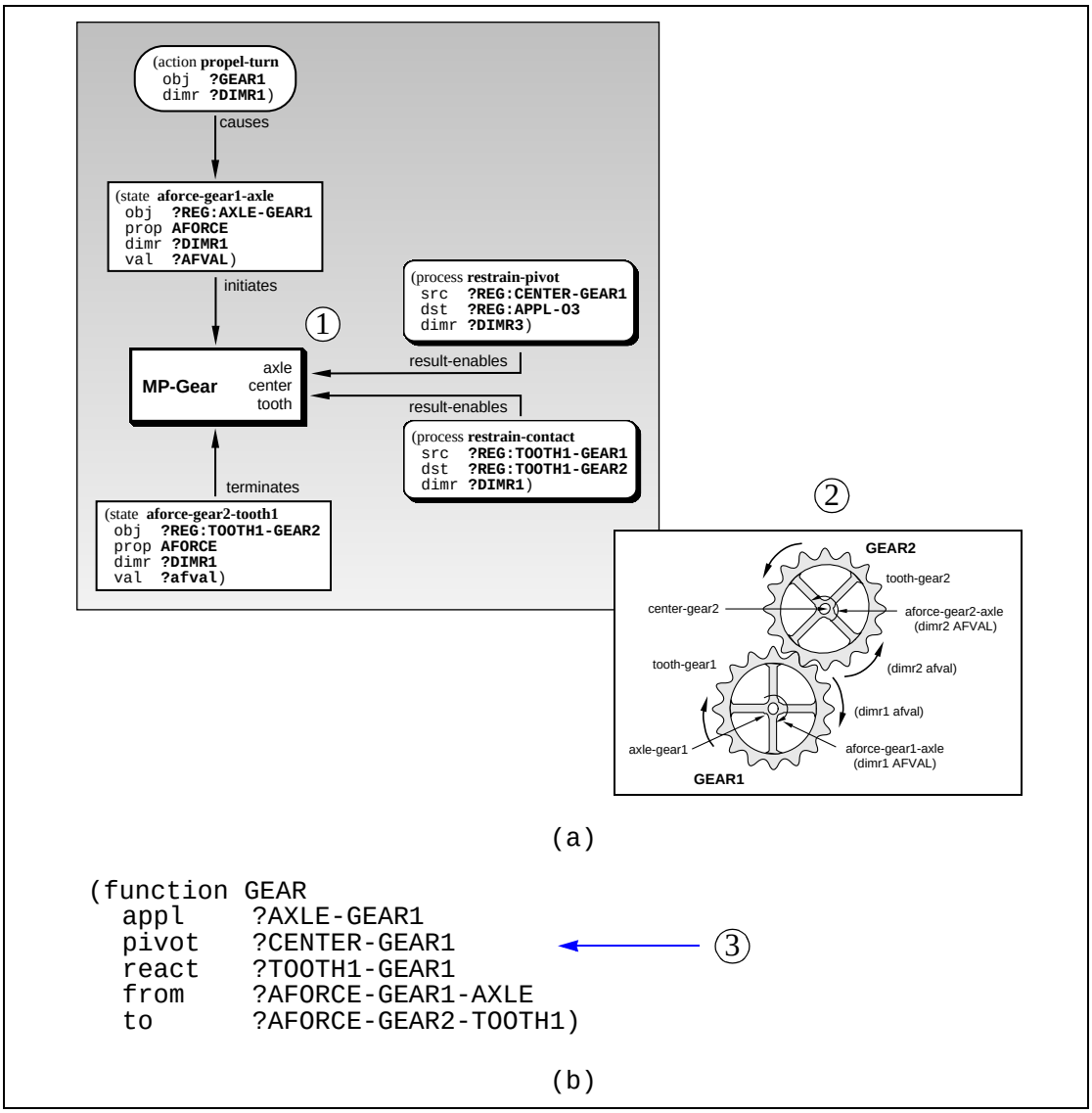


Figure 4.18 Block version of MP-GEAR function, gear object statics, and slot-filler representation.

The graphic illustrates two gears (GEAR1 and GEAR2) in motion. The size and shape requirement on tooth contact enables RESTRAIN-INTERFERE. Representing the motion of gear objects in tooth-contact as a one-tooth change in position has two consequences on the function of MP-GEAR. First, since there are a finite number of teeth on a gear, motion control is possible. Second, because the restraining process is lateral contact (using a protuberance), rather than surface contact, large force values can be transmitted because there will be no slipping.

Gear object statics can be specialized to control the direction and value of the force caused by MP-GEAR. For example, a gear object could be comprised of two gears with beveled edges. The result would be a change in the direction of the transmitted force/motion. A gear-object could be comprised of a gear and a chain, or a gear and a rack. The result would be the distance through which the force/motion are transmitted. The size of gears and

the number of teeth on their edge regions affect the degree of control and the amount of force magnification, respectively. The amount of resultant distance magnification is controlled by the rule WR-1, which is based on radius ratio between the gear objects. The same result can be obtained with the ratio of teeth between the gear objects, as noted in rule GR-1:

GR-1: dist(g2) = dist(g1) * (t1/t2) (4-3)

where dist is the relative distance travelled, with respect to the radius of the respective gear, g2 is the second gear, g1 is the first gear, t1 is the number of teeth on the first gear, and t2 is the number of teeth on the second gear.

4.2.5 MP-PULLEY

MP-PULLEY describes devices used to control the transformation of force and motion with edge-edge friction (surface-contact) and interference. The pulley object is a specialized wheel object with a slotted edge region. The slot is an indentation so it enables RESTRAIN-INTERFERE (it is not RESTRAIN-CONTAIN because the pulley belt center of gravity is outside that of the pulley). MP-PULLEY has slot, center, and axle function roles, which are associated with the MP-WHEEL-AXLE edge, center, and axle roles. MP-PULLEY is enabled by a RESTRAIN-PIVOT for the region which instantiates the center role, and a RESTRAIN-INTERFERE between the pulley object slot and belt inside-surface regions. MP-PULLEY is initiated by a forcing state at the pulley object axle region. An enabled MP-PULLEY is terminated by a force state on the object in contact with the pulley object slot region. MP-PULLEY describes the enablements and resulting state of the process TRANSFORM-MAGNIFY, so the value of the perturbation force is magnified. These

relationships are depicted in Fig. 4.19.

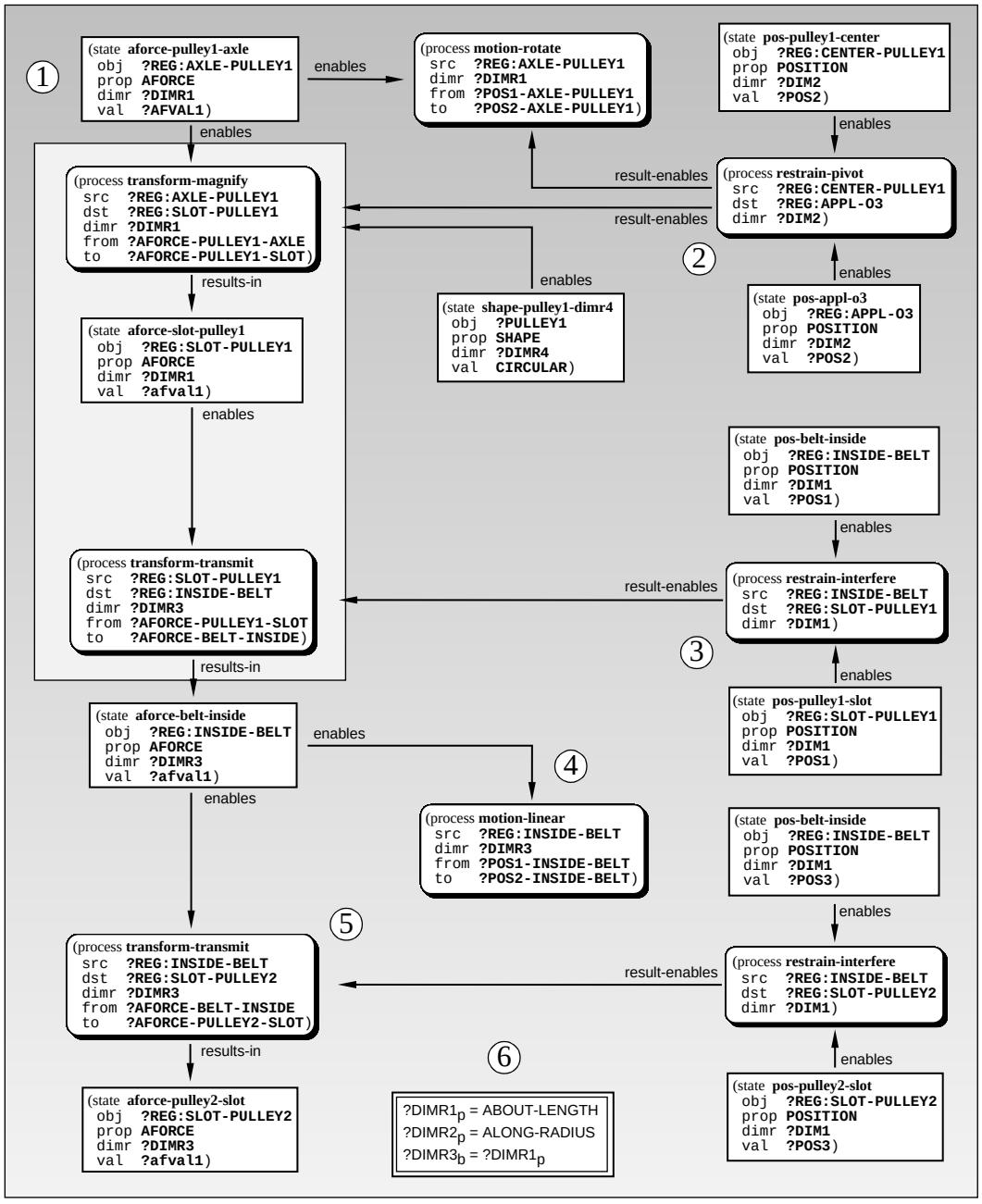


Figure 4.19 Behavioral process sequence associated with the PULLEY machine primitive.

Fig. 4.19 illustrates the behavioral processes which describe pulley function. The shaded box describes the intra-object dynamics of the pulley object. The applied force (AFORCE-PULLEY1-AXLE, at label 1) enables force magnification (TRANSFORM-MAGNIFY) from the pulley object axle to its slot contact. The RESTRAIN-PIVOT process (2) enables force magnification and rotation of the pulley object. The contact between the pulley object slot and the pulley belt (3) enables force transmission to the belt. The interference con-

strains the motion of the belt to the plane of the pulley object. Out-of-plane motion disablement is not illustrated in the figure. A second pulley object is also illustrated in the figure. The force resulting from MP-PULLEY (AFORCE-BELT-INSIDE) enables linear motion of the belt (4). If the belt is in contact with another gear (5), then the process is reversed. The legend labeled (6) is shows sample dimensions associated with the illustration of pulley object statics shown at (2) in Fig. 4.20a. The machine primitive associated with the shaded box in Fig. 4.19 is also replaced with its block equivalent at (1) in Fig. 4.20a.

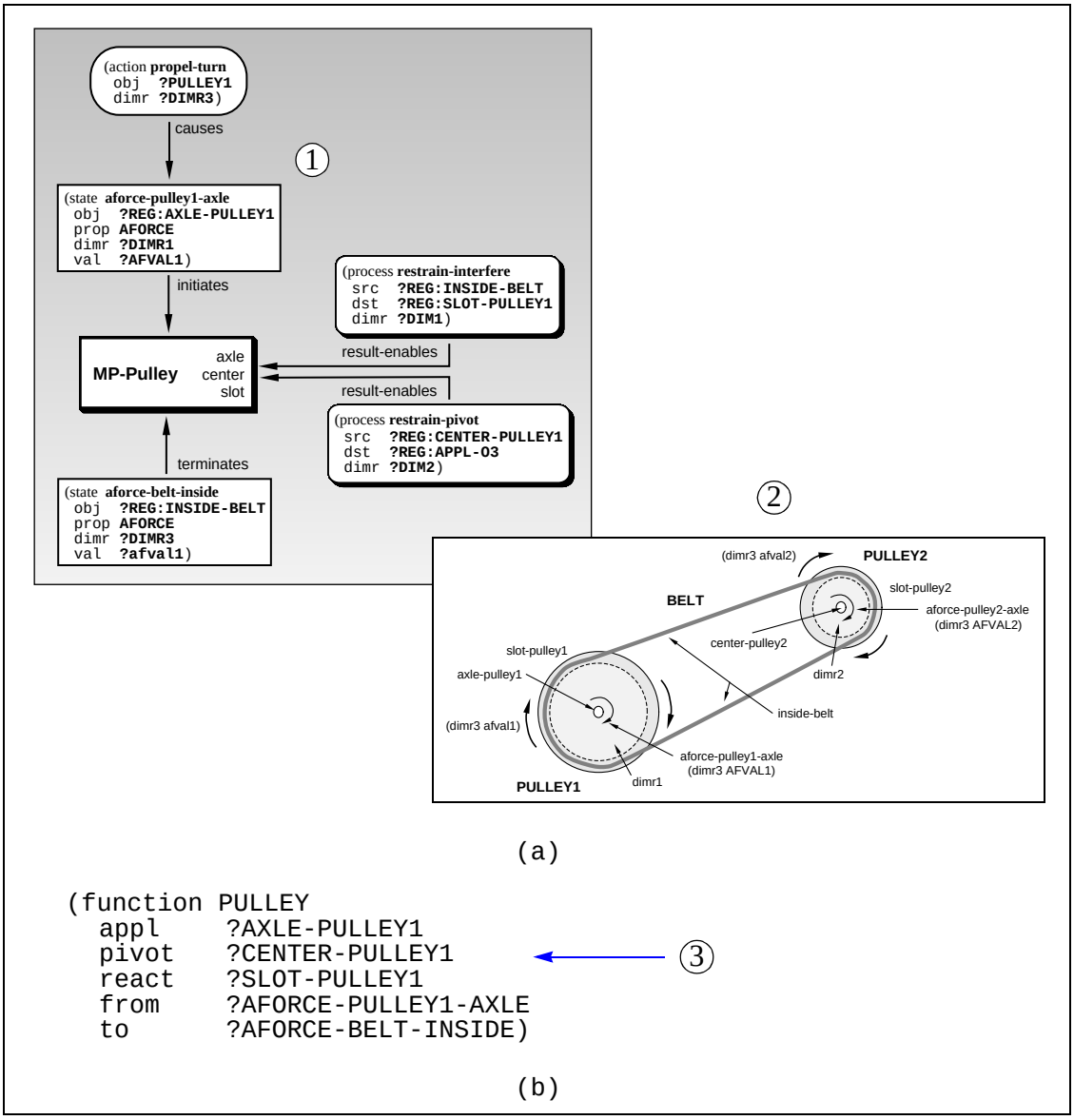


Figure 4.20 Block version of MP-PULLEY function, pulley objects, and slot-filler notation for MP-PULLEY.

Fig. 4.20a illustrates two pulley objects (PULLEY1 and PULLEY2), and their common belt (BELT), in motion. The process RESTRAIN-PIVOT between PULLEY1 and O3, and between PULLEY2 and O4, enables a TRANSFORM-TRANSMIT to the supporting object (O3 and O4), but these relationships have not been illustrated. The MP-PULLEY block

in Fig. 4.20a is shown being initiated by the force AFORCE-PULLEY1-AXLE, produced in this instance by turning (PROPEL-TURN) the pulley object at its axle (region AXLE-PULLEY1). The force (AFORCE-PULLEY2-SLOT), which represents the force transmitted to, and magnified at, SLOT-PULLEY2, terminates MP-PULLEY. The slot-filler notation for PULLEY function represented in Fig. 4.20b shows the mapping between MP-PULLEY roles and the function class roles.

Surface (or friction) contact between the pulley slot region (SLOT-PULLEY1) and the belt inner surface (INSIDE-BELT) has two consequences on the function of MP-PULLEY. First, since there is no way to guarantee relative position between a pulley and a belt, MP-PULLEY is not used as a motion control device. Second, because the restraining process is surface contact, transmitted forces are approximated by the amount of friction between the belt and the pulley slot.

Pulley object statics can be specialized to control the direction and value of the force caused by MP-PULLEY, similar to wheels and gears. For example, a pulley object could be comprised of a pulley and belt and, by twisting the belt, the dimension of motion can be arbitrarily changed. This cannot be done with MP-GEAR because a chain is usually fairly rigid so that forces can be transmitted. The force magnification produced by axle-to-slot transmission is determined with the rule WR-3 (or WR-2 in the reverse case). By cascading pulley objects, as depicted in Fig. 4.20a, mechanical advantage on the belt can be obtained without consideration of the axle forces whatsoever. For example, a block and tackle is a composition of many pulley objects sharing a single rope. Any multiple of input force magnification can be realized by increasing the number of pulley objects in the block. Force magnification in pulleys can be determined with radius ratios in the same way it was with wheels and gears, using WR-4. If the force of interest is the one in the belt, then the radii are not important but the number of pulleys is, and can be determined with the rule PR-1:

PR-1: afval2 = afval1 / N (4-4)

where afval1 is the applied load, afval2 is the resultant force value, and N is the number of pulley objects using the same belt in such a way that the belt travel is increased N-fold.

4.2.6 MP-BEARING

MP-BEARING describes devices that enable relative surface translation by reducing the friction associated with surface contact. The bearing object is a cylindrical or spherical object with a surface region that instantiates the wheel object edge and react regions. MP-BEARING has surface, center, and surface roles which play the role of the function appl, pivot, and react roles, respectively. MP-BEARING is enabled by a RESTRAIN-CONTACT processes between each surface and the surface regions of the bearing object. The bearing object shape and contact with the surface enables rolling. MP-BEARING is initiated by an applied force on a surface in contact with the bearing object surface1 region. An enabled MP-BEARING is terminated by a force on the second surface in contact with the bearing object surface2 region. MP-BEARING describes the enablements and resulting state of the process TRANSFORM-TRANSMIT, so the value of the applied force is un-

changed. These relationships are depicted in Fig. 4.21.

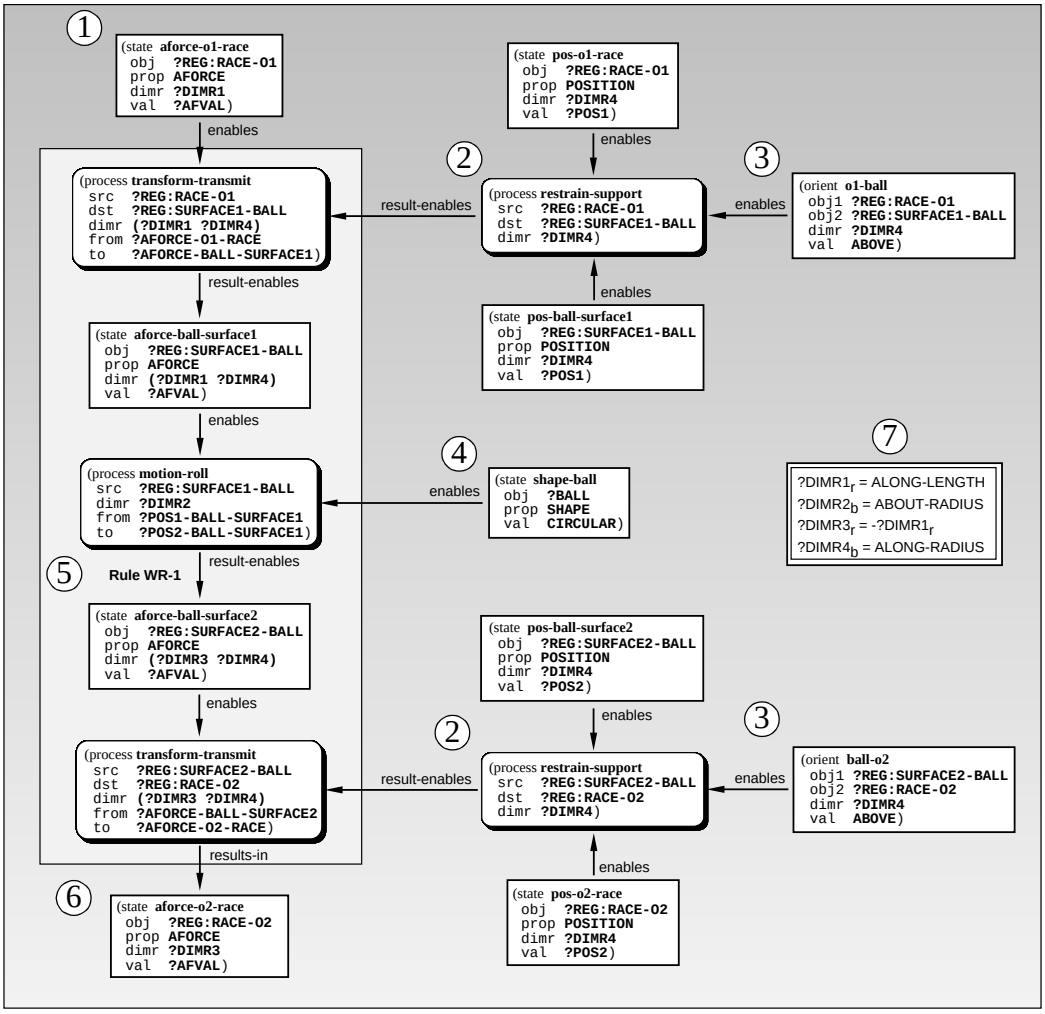


Figure 4.21 Behavioral process sequence associated with the BEARING machine primitive.

Fig. 4.21 illustrates the behavioral processes which describe bearing function. The shaded box represents the intra-object dynamics of the bearing object. The applied force (AFORCE-O1-RACE, at label 1) enables lateral force transmission (TRANSFORM-TRANSMIT) from the bearing race to the bearing object surface region, and motion of the bearing race. The RESTRAIN-SUPPORT processes (2 and 3) between the bearing object and the bearing races enable lateral force transmission from the bearing race to the ball, and radial force transmission to the other bearing race. The bearing object (BALL) supports radial load (DIMR4) because it maintains radial contact between one bearing race (O1), and the other (O2). The circular shape of the bearing object (4), the lateral applied force, and the surface contact between the bearing object and its races enable rolling (MOTION-ROLL) between the bearing object and its races. The force changes direction at the surface2 region on the bearing object according to WR-1 (5), which is then transmitted to the second bearing race (6). The legend labeled (7) shows sample dimensions associated with the illustration of bearing object statics shown at (2) in Fig. 4.22a. The machine primitive associated with the shaded box in Fig. 4.21 is also replaced with its block equivalent at (1) in

Fig. 4.22a.

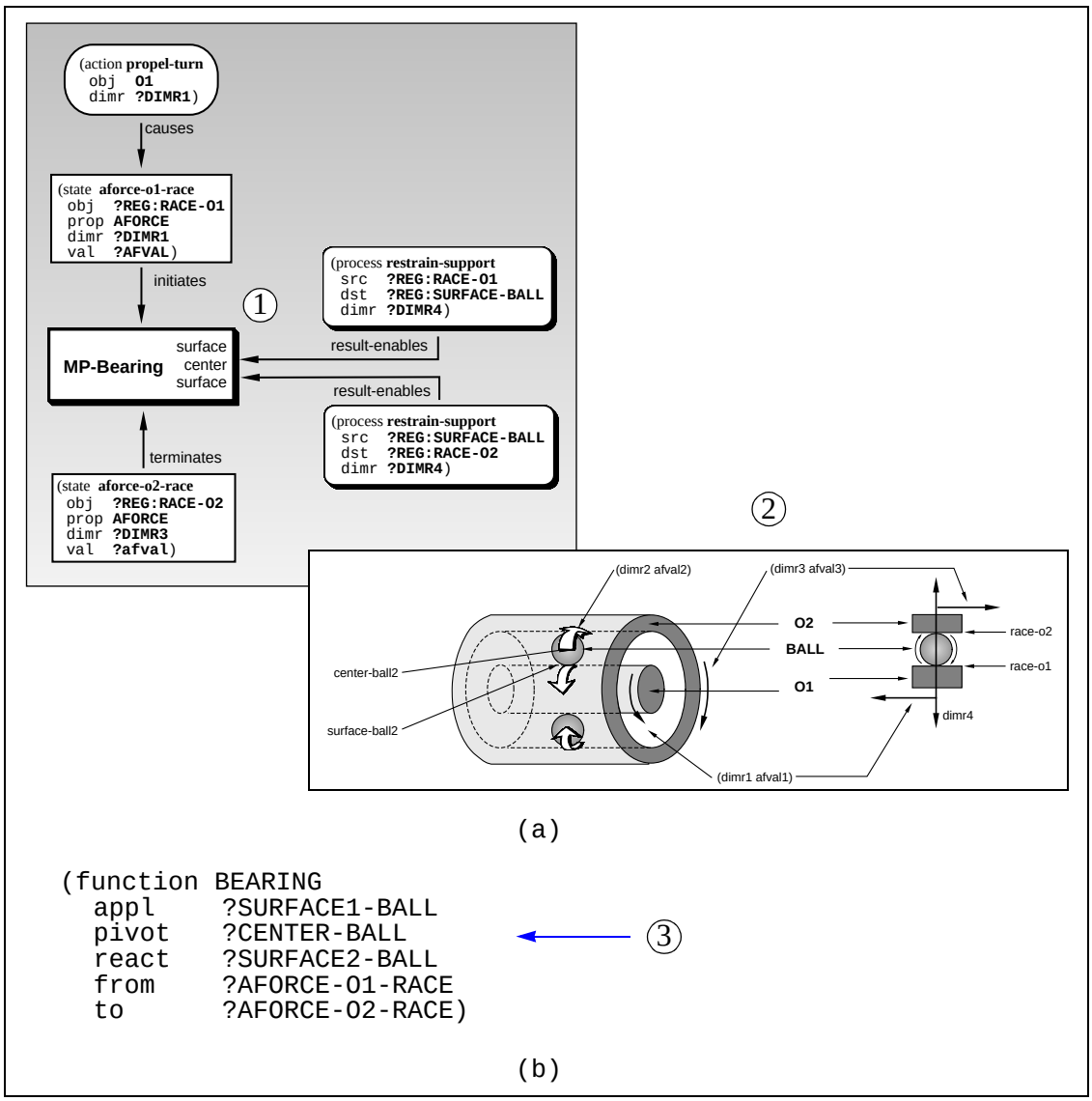


Figure 4.22 Block version of MP-BEARING function, bearing object statics, and associated slot-filler representation

The graphic in Fig. 4.22a illustrates two surfaces (O1 and O2) in relative motion, mediated by the rolling contact of the bearing object (BALL). The block version of MP-BEARING (1) is shown being initiated by the force AFORCE-O1-RACE. The state AFORCE-O2-RACE, which represents the force transmitted to RACE-O2, terminates MP-BEARING. The dimensions DIMR1, DIMR2, and DIMR3 are used in the figures to illustrate the relations which are maintained between the processes. DIMR1 and DIMR3 are simply opposing directions of the same dimension, while DIMR2 is the rotational dimension associated with DIMR1 and DIMR3. The force values AFVAL1, AFVAL2, and AFVAL3 are also used for clarity, and have the same value.

4.2.7 MP-CONTAINER

MP-CONTAINER describes devices used to restrict object motion to a specific volume by enabling the process RESTRAIN-CONTACT. MP-CONTAINER has three roles: inside, boundary, and outside which instantiate the function appl and react roles. The container object is a linkage object specialized by an indentation. The container object inside region (the indentation) describes a volume bounded by the container object's inside surface and the container object boundary region. The contained object is free to move within the container object inside region, but not beyond it, except via the boundary. The container object outside and boundary regions can instantiate the MP-LINKAGE appl and react roles. MP-CONTAINER is enabled by a RESTRAIN-CONTACT between the contained object and the container object inside region. MP-CONTAINER is initiated by a force on the container object, or the contained object, enabling the process MOTION for the forced object. An enabled MP-CONTAINER is terminated by a force on the contained object, opposing and disabling its motion. This relationship is depicted in Fig. 4.23.

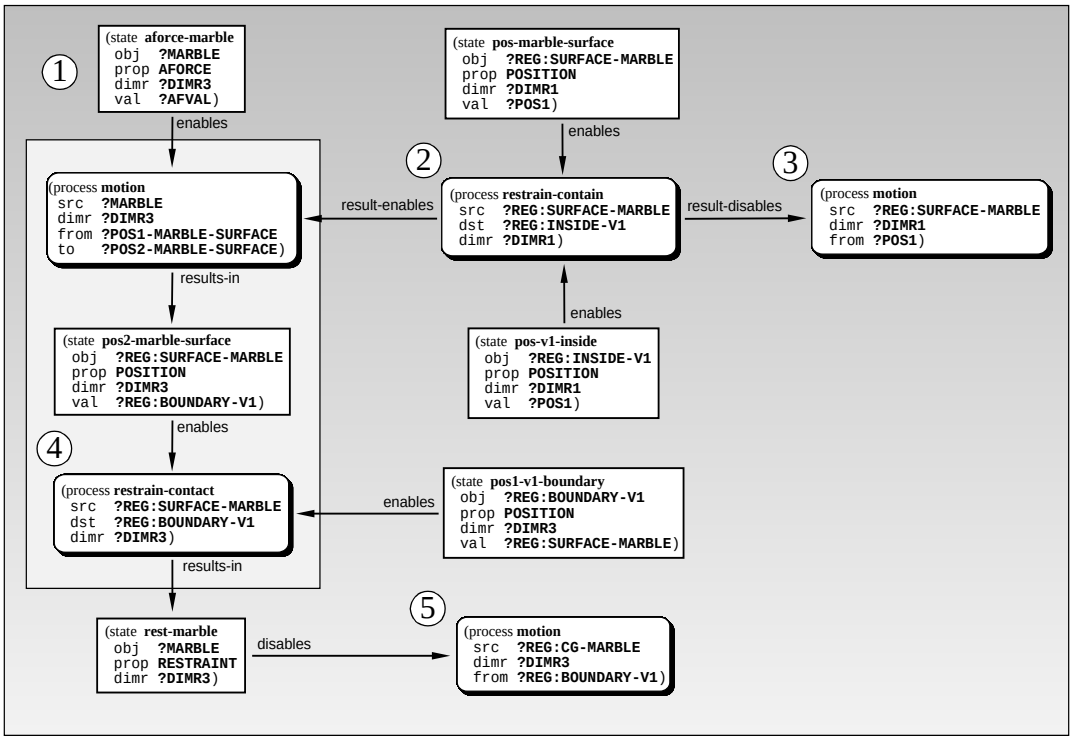


Figure 4.23 Behavioral process sequence associated with the CONTAINER machine primitive.

Fig. 4.23 illustrates the behavioral processes which describe container function, instantiated for the objects depicted in Fig. 4.24. The shaded box represents the intra-object dynamics of the container object. The force (AFORCE-MARBLE, at label 1) enables marble motion. The position of the marble at the same place as the inside of the vessel enables RESTRAIN-CONTAIN (2), which enables lateral motion but disables motion into the boundary (3). The lateral motion results in a new position, which eventually coincides with the vessel boundary (4). The new position enables RESTRAIN-CONTACT and the resulting restraint state disables marble motion (5). Fig. 4.24a (at 1) depicts the block version of the shaded box in Fig. 4.23. The accompanying illustration (at 2) depicts an example container

object and contained object statics.

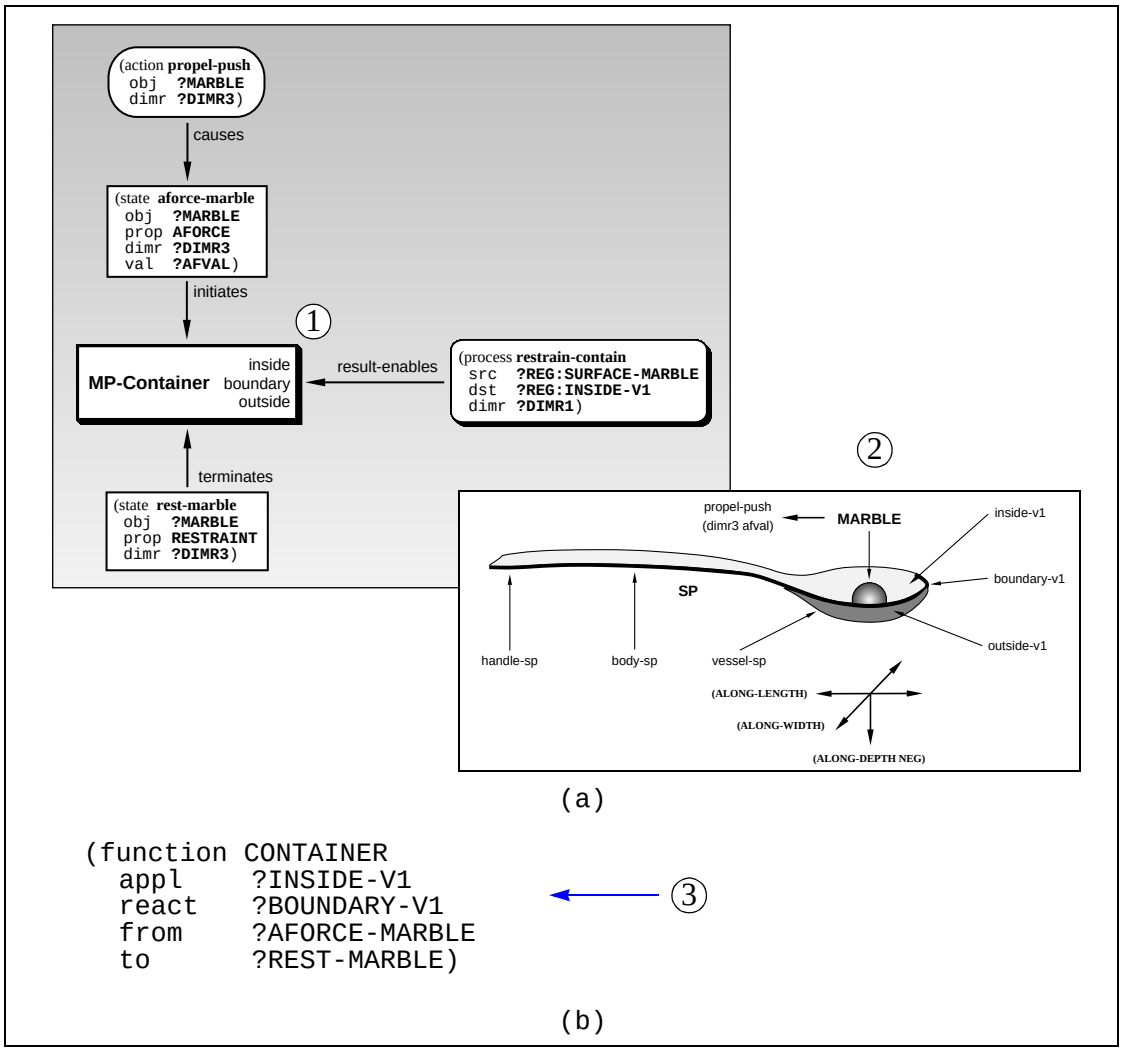


Figure 4.24 Block version of MP-CONTAINER function, container object statics, and the associated slot-filler representation

The graphic in Fig. 4.24a illustrates a spoon (SP) as a container object, and a marble (MARBLE) as the contained object. The spoon has three regions: HANDLE-SP, BODY-SP, and VESSEL-SP. VESSEL-SP region is the container object. MP-CONTAINER is shown being initiated by the force AFORCE-MARBLE in DIMR3. The connectivity precondition is that the contained object is within the boundary of the container object, which is represented as an enabled RESTRAIN-CONTAIN. The contact between the MARBLE and the vessel boundary results in a restraint state that terminates MP-CONTAINER.

In Fig. 4.24a, DIMR1 is instantiated by the spoon's thickness axis in the downward direction (ALONG-DEPTH NEG). The MP-CONTAINER behavioral enablement is position of, and contact between, MARBLE and INSIDE-V1 in the (ALONG-DEPTH NEG) dimension/direction. This relationship disables MARBLE motion downward, but not upward or laterally. If DIMR3 is instantiated by the spoon's longitudinal axis, toward its CG (ALONG-LENGTH POS), then MP-CONTAINER causes a motion disablement when

MARBLE comes in contact with INSIDE-V1 in the (ALONG-LENGTH POS) dimension/direction. The same result is caused if the forcing dimension/direction is (ALONG-LENGTH NEG), (ALONG-WIDTH POS), or (ALONG-WIDTH NEG). In FONM, MARBLE's exact location within the container-object's inside region unimportant.

Container-objects may take many static forms. For example, link chains, cables, and ropes instantiate MP-CONTAINER. The link of a link chain instantiates both the boundary and outside container object regions, while the link hole instantiates the inside container-object region. As long as one link is not touching another, it is free to move in any dimension. Cables and ropes can instantiate MP-CONTAINER, since their end region can describe an imaginary volume of containment. If an object is connected to the end region, then it can move freely within the imaginary volume of containment, but not beyond it. Leaks in container objects are represented as additional boundaries. If the size of the boundary is large enough for the contained object, then MP-CONTAINER is disabled because the RESTRAIN-CONTACT is disabled.

4.2.8 MP-SPRING

MP-SPRING describes devices used to produce passive energy storage, in the form of internal force, by enabling the process STORE. MP-SPRING functions by pushing back against a perturbation force, and by returning to its original size and shape, if allowed, when the perturbation force is released. MP-SPRING is a specialization of MP-LINKAGE where the force applied to the spring object is large enough to produce visible size and shape changes but doesn’t exceed the linkage object's material yield strength. MP-SPRING has the same appl and react roles as does MP-LINKAGE. MP-SPRING is enabled by a transmission of the applied force to another object which is, itself, restrained from motion in the forced dimension/direction. MP-SPRING is initiated by a forcing state at the spring object appl region. An enabled MP-SPRING is terminated by two states: an internal force which opposes the perturbation force state, and a size change which follows the perturbation force dimension/direction. An applied force toward the spring object CG results in shortening and an applied force away from the spring object CG results in a lengthening. The amount of internal force and size change is independent of the applied force direction. These rela-

tionships are depicted in Fig. 4.25.

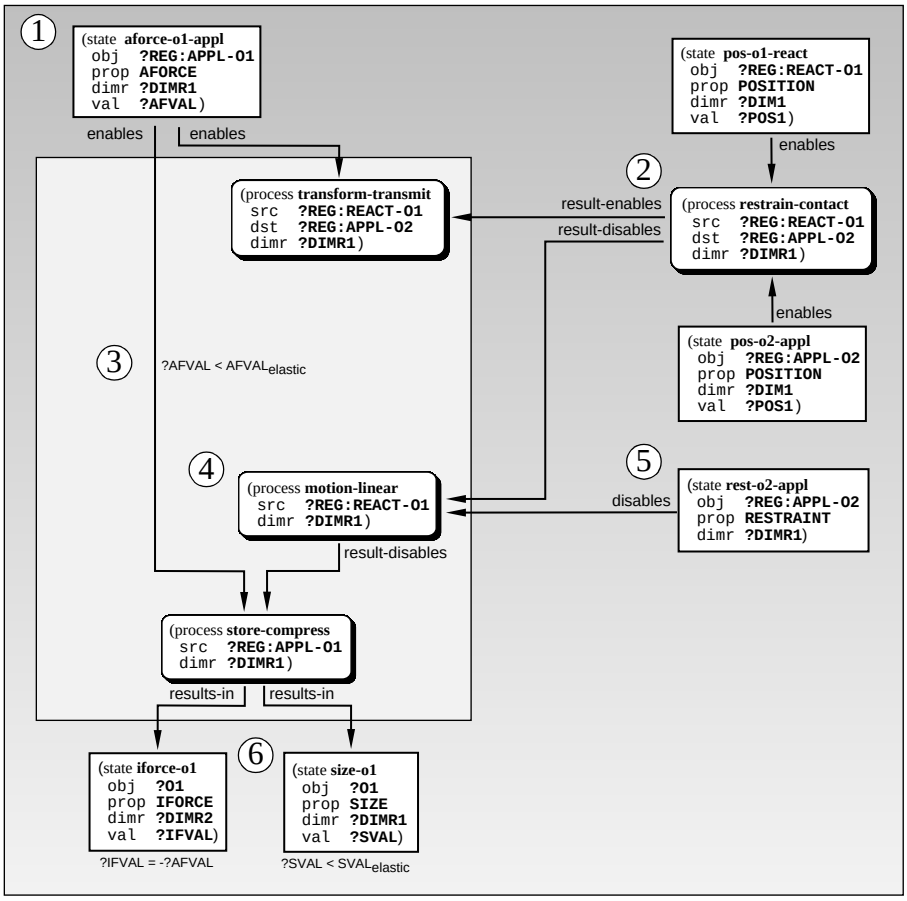


Figure 4.25 Behavioral process sequence associated with the SPRING machine primitive.

Fig. 4.25 illustrates the behavioral processes which describe spring function. The shaded box represents the intra-object dynamics of the spring object. The applied force (AFORCE-O1-APPL, at label 1) enables force transmission to the object (O2) which the spring object is in contact (RESTRAIN-CONTACT, at 2). The applied force also enables passive energy storage (STORE) as long as the force value is less than the spring object's material elastic limit (3). STORE is also enabled by disabled spring object motion (4), which is the combined result of contact with the base object (O2) and a restraint on O2 (5). The states resulting from STORE (IFORCE-O1 and SIZE-O1) are the effect associated with springs. The block version of MP-SPRING is shown in Fig. 4.26a (at 1), along with a example of a coil spring in compression, Fig. 4.26a (2). The slot-filler in Fig. 4.26b shows

the function role mapping (3) from MP-SPRING for the block function in Fig. 4.26a.

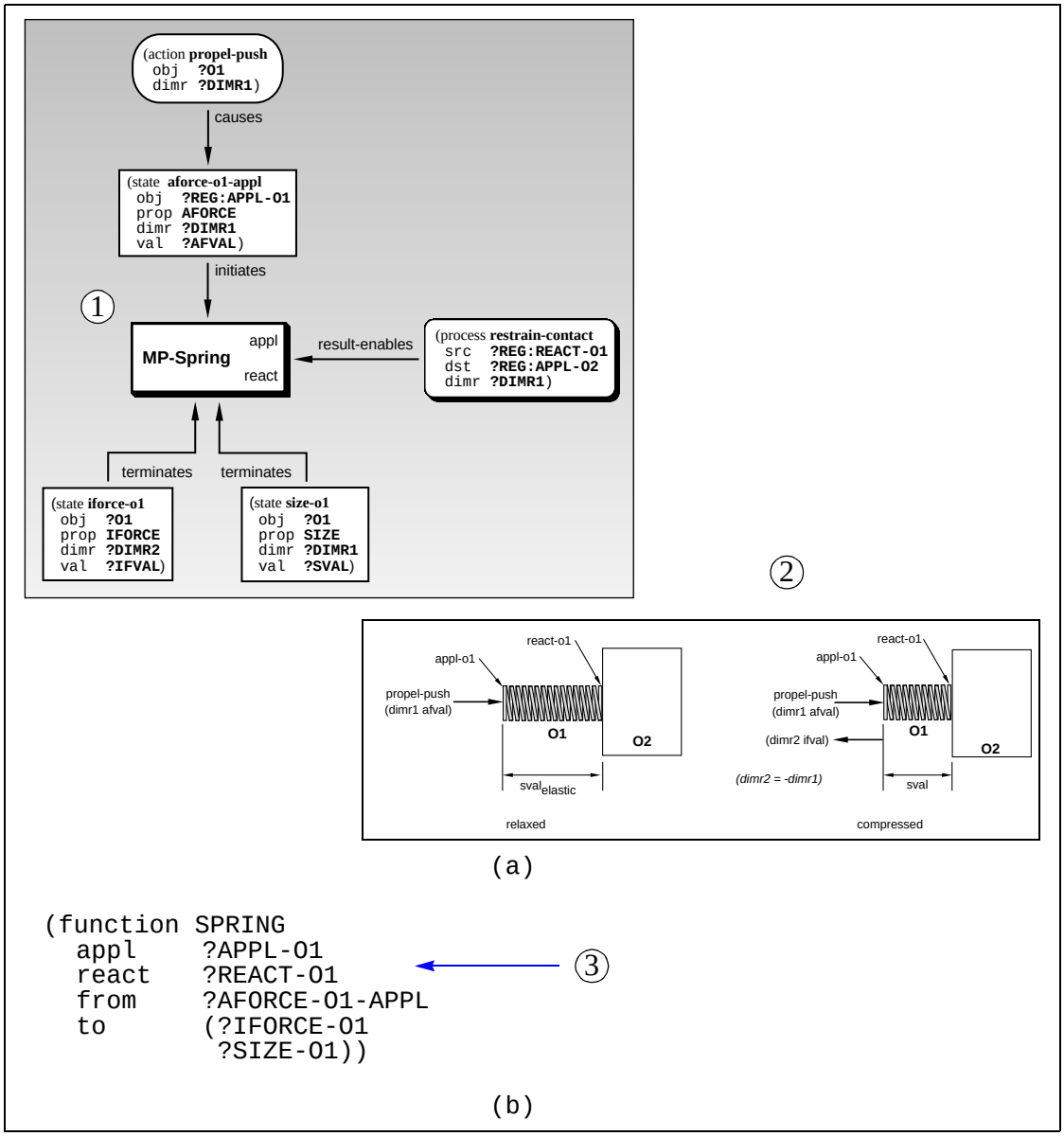


Figure 4.26 Block version of MP-SPRING function, spring object statics, and the associated slot-filler representation

Fig. 4.26a illustrates a helical-coil spring object (O1) in a relaxed state (SVAL = $SVAL_{elastic}$) and in contact with a block (O2). Once enabled, MP-SPRING causes a change in O1 length from $SVAL_{elastic}$ to (SVAL < $SVAL_{elastic}$), and an internal force in the applied force dimension, opposing direction (DIMR2 IFVAL).

4.2.9 MP-PLANE

MP-PLANE describes devices used to reduce the effort of moving an object in one dimension by moving it in a combination of dimensions. A person walking up a hill is effectively walking up an inclined plane. It is easier to ascend a hill by walking up an incline, or slope,

than by climbing straight up. As the hill's slope increases, the ease with which the person ascends the hill, as depicted in Fig. 4.27, decreases. From Applied Mechanics, the overall work is constant regardless of slope, however, lower slopes distribute the work over a larger distance. Therefore the inclined plane, like a lever, is a device which trades off the force applied and the distance through which it is applied.

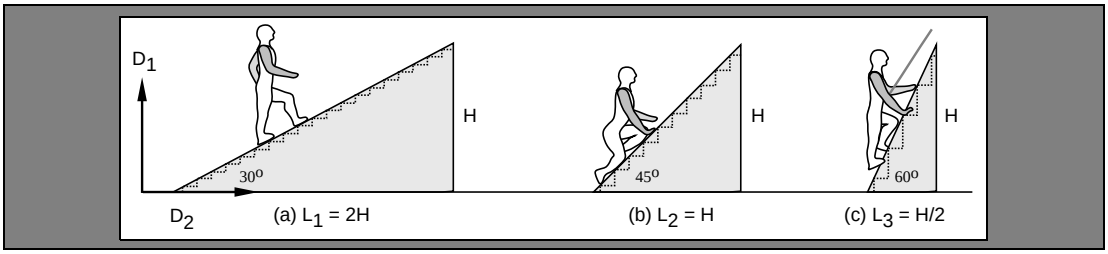


Figure 4.27 Hills as inclined planes.

Fig. 4.27 illustrates a person ascending three slopes, each defined by the ratio of the height he or she will ascend to the horizontal distance they travel to ascend that height. Walking up a shallow-sloped hill is much easier than crawling or climbing steeper slopes, all of which are easier than lifting oneself vertically the same distance. The lower the ratio of H to L, the easier the climb will be but the further the travel.

There is a difference between an inclined plane and a lever. In the lever, there is a trade-off between force value and the distance between the location of its application and reaction with respect to a pivot location. Neither the applied nor reacted force changes its location on the lever object. The inclined plane trades off the force required for an object to move a distance in dimension D1, with the distance traveled along another dimension D2 to do so. An analogy can be made using the number of steps required to climb a hill. To raise oneself from ground to the top of the hill in a single step takes the greatest single force. For each illustration, right to left, in Fig. 4.27, the number of steps taken increases and the height climbed in each step decreases. The only change is the slope of the hill. An infinite slope is a one-step climb, and requires the maximum force. A zero slope is an infinite-step climb, and requires no force. All slope gradations in-between represent fractions of the maximum force for each step.

The plane object is a linkage object specialized by is triangular shape. The plane object is described by two of the wedge surfaces and one of their edges. One surface is where the perturbation force is applied, and the other surface is where the perturbation force is transmitted to a second object. The edge formed by the intersection of these surfaces forms an acute angle, called an *offset* angle, which acts as a pivot in producing force magnification. MP-PLANE has three roles: appl, offset, and react, which correspond to the MP-LINKAGE appl and react roles, and to the MP-LEVER appl, pivot and react roles. MP-PLANE is enabled by a RESTRAIN-CONTACT between an object and the plane object region which instantiates the MP-PLANE react role. MP-PLANE is initiated by a force at the plane object region which instantiates the MP-PLANE appl role. An enabled MP-PLANE is terminated by a force on the second object in the dimensions and directions described by the plane object region which instantiates the MP-PLANE offset role. MP-PLANE describes the enablements and resulting state of the process TRANSFORM-MAGNIFY, so

the value of the resulting force is magnified. These relationships are illustrated in Fig. 4.28.

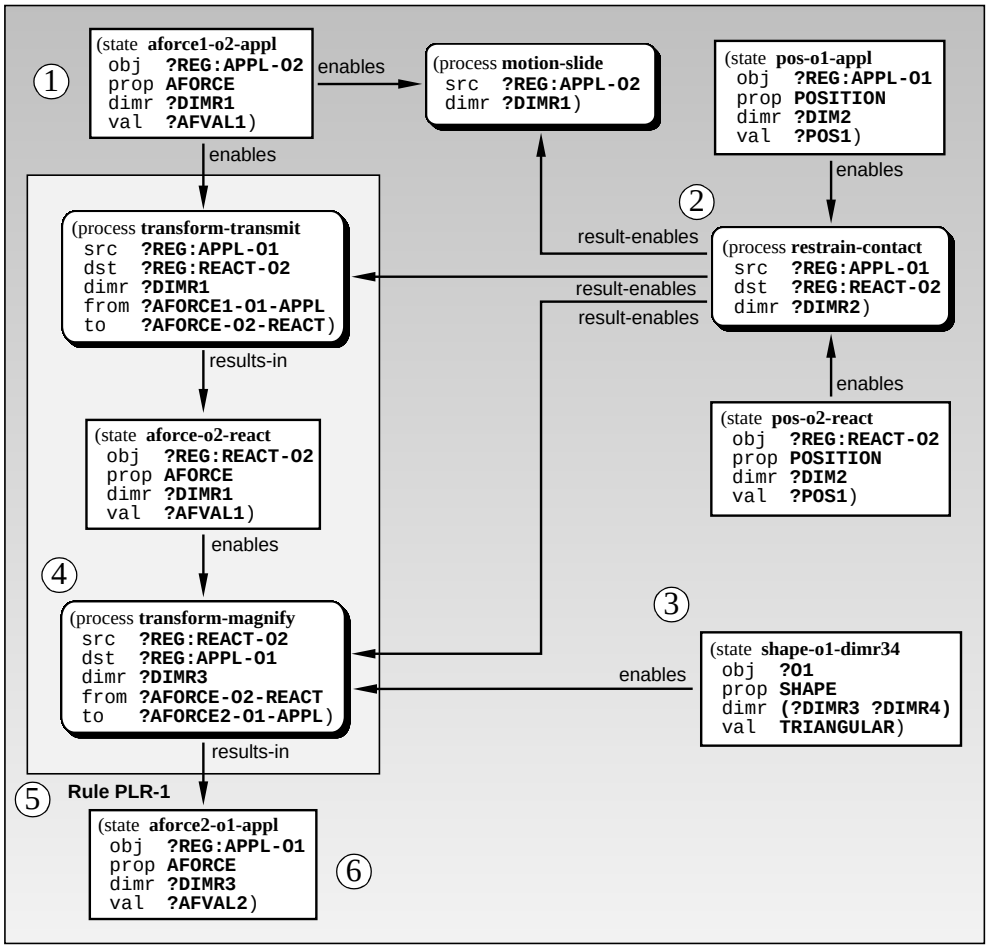


Figure 4.28 Behavioral process sequence associated with the PLANE machine primitive.

Fig. 4.28 illustrates the behavioral processes which describe plane function. The shaded box represents the intra-object dynamics of the plane object. The force (AFORCE-O1-APPL, at label 1) lies in the plane of the wedge (DIM1) and enables sliding along the surface because of the RESTRAIN-CONTACT in the normal dimension (DIM2). The force also enables transmission to the plane object (O2) react region (REACT-O2). The triangular shape of the plane object enables force magnification by dividing it into components in different dimensions (DIM3 and DIM4), resulting in a lower force in dimension DIM3 than if no plane were used. The block version of the shaded box representation of MP-PLANE in Fig. 4.28 is depicted at (1) in Fig. 4.29a. The accompanying illustration, at (2) in Fig. 4.29a depicts plane object statics. The function represented in Fig. 4.29b shows the role mapping

from MP-PLANE (3).

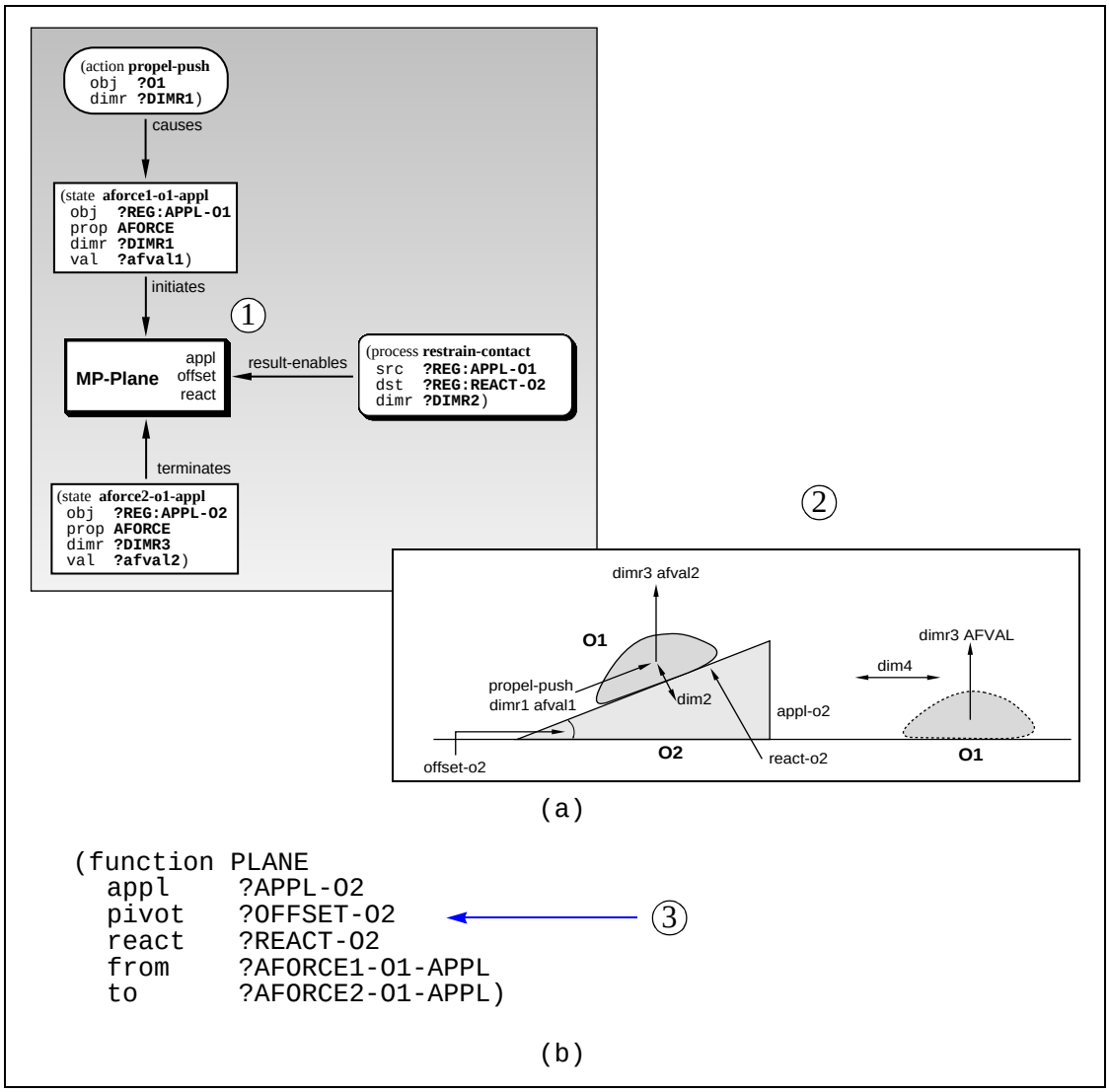


Figure 4.29 Block version of MP-PLANE function, plane object statics, and the associated slot-filler function representation

The graphic in Fig. 4.29a illustrates a wedge (O2) and a rock (O1). In this illustration, the rock is shown as the object in motion. As with all processes in FONM, there is no difference between the wedge moving toward the rock or the rock moving toward the wedge. The force AFORCE1-O1-APPL enables O1 sliding and initiates MP-PLANE. The contact between the rock and the wedge, and the shape of the wedge, enable MP-PLANE. The state AFORCE2-O1-APPL, which represents the force converted by the wedge into dimensional components, in DIMR3, terminates MP-PLANE.

When considered as an MP-LEVER pivot point, the plane object offset angle provides an effective measure of motion advantage. Consider a plane object with equal appl and base surface lengths. The ratio of these lengths (L_{APPL}/L_{BASE}), called an *offset ratio*, is 1. This value is defined as nominal, or NOM. This plane object is analogous to MP-LEVER-TYPE1, where the lever object pivot region is located halfway between the lever object

appl and react regions. Movement of the lever object appl region results in an equal and opposite movement of the react region. In MP-PLANE, the movement of the plane object O1 along DIMR1 results in an equal movements of O2 along DIMR3 and DIMR4, as shown in rule PLR-1:

PLR-1: $dist_{d3}/dist_{d4} = l_{appl}/l_{base}$ (4-5)

PLR-2: $afval_{d3} = afval1_{d3} * l_{appl}/(l_{appl} + l_{base})$ (4-6)

where dist is distance travelled, d2 is the dimension of the appl surface, d1 is the dimension of the base surface, l_{appl} is the length of the appl surface, and l_{base} is the length of the base surface. When the plane object offset ratio is less than nominal ($L_{APPL}/L_{BASE} < 1$), MP-PLANE is similar to MP-LEVER-TYPE2: it is most effective for moving an object in DIMR4, with a <NOM force value (rule PLR-2). When the offset ratio is greater than nominal ($L_{APPL}/L_{BASE} > 1$), MP-PLANE is similar to MP-LEVER-TYPE3: it is most effective for moving an object in DIMR3, with a greater rate than the nominal rate, and a resultant force approaching the original nominal force as the base length approaches zero.

4.2.10 MP-BLADE

MP-BLADE describes devices used to cut or otherwise permanently deform objects, by enabling the DEFORM process, through the application of linear mechanical advantage and linear motion advantage. MP-BLADE function results from the combination of two effects. First, the blade object is a wedge. The appl surface area of the wedge is much larger than the edge surface area, so forces applied there are magnified at the edge (see Fig. 3.15). This is linear mechanical advantage. Second, as a wedge the blade object is also a plane object. Once a cut is initiated the MP-PLANE enables the blade object to advance by pushing the material aside. This is linear motion advantage. The blade object has three regions that instantiate MP-BLADE roles: appl, edge, and react. The blade object edge region describes the intersection of the plane object react surface and base surface. MP-BLADE is specialized from MP-PLANE by perturbing the plane object toward the second object edge on, as illustrated in Fig. 4.30.

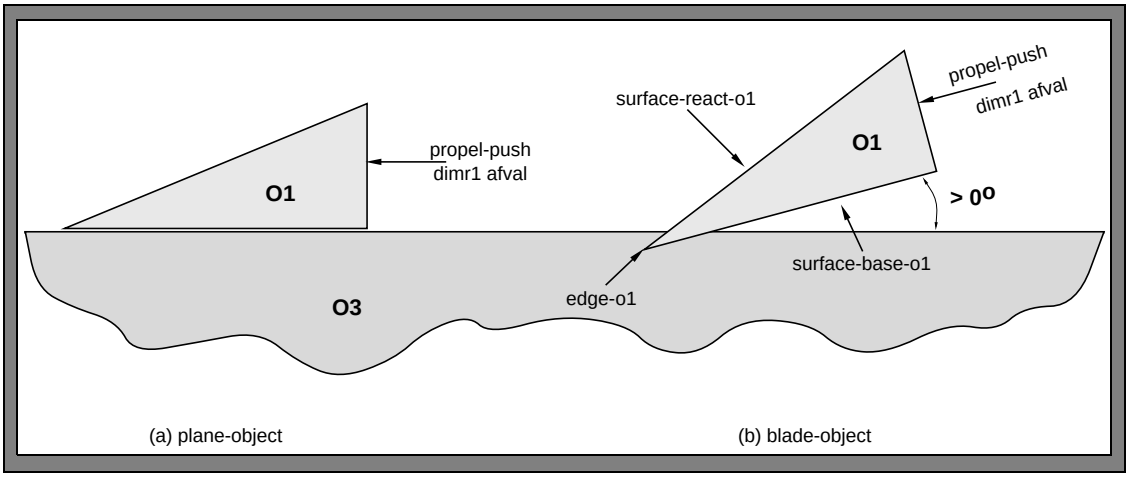


Figure 4.30 Application of (a) plane object versus (b) blade object.

Fig. 4.30 illustrates the difference in application that distinguishes the MP-BLADE from the MP-PLANE, depicted with plane objects. The term edge on refers to the orientation of

the blade object to the surface it contacts. In MP-PLANE, the plane object edge is parallel (angle = 0^{o}) with respect to the surface upon which it rests, so there is no edge contact, Fig. 4.30a. In MP-BLADE, the blade object itself is oriented at an offset to (angle > 0^{o}) substrate, so its edge impinges on the substrate as it moves. The blade object react region describes both top and bottom wedge faces.

MP-BLADE is enabled by a RESTRAIN-CONTACT between the blade object react region and the substrate. MP-BLADE is initiated by a force applied at the blade object appl region. An enabled MP-BLADE is terminated by a change in size of the substrate object. MP-BLADE describes the enablements and resulting state of the process TRANSFORM-MAGNIFY, so the value of the resulting force is magnified. These relationships are depicted in Fig. 4.31.

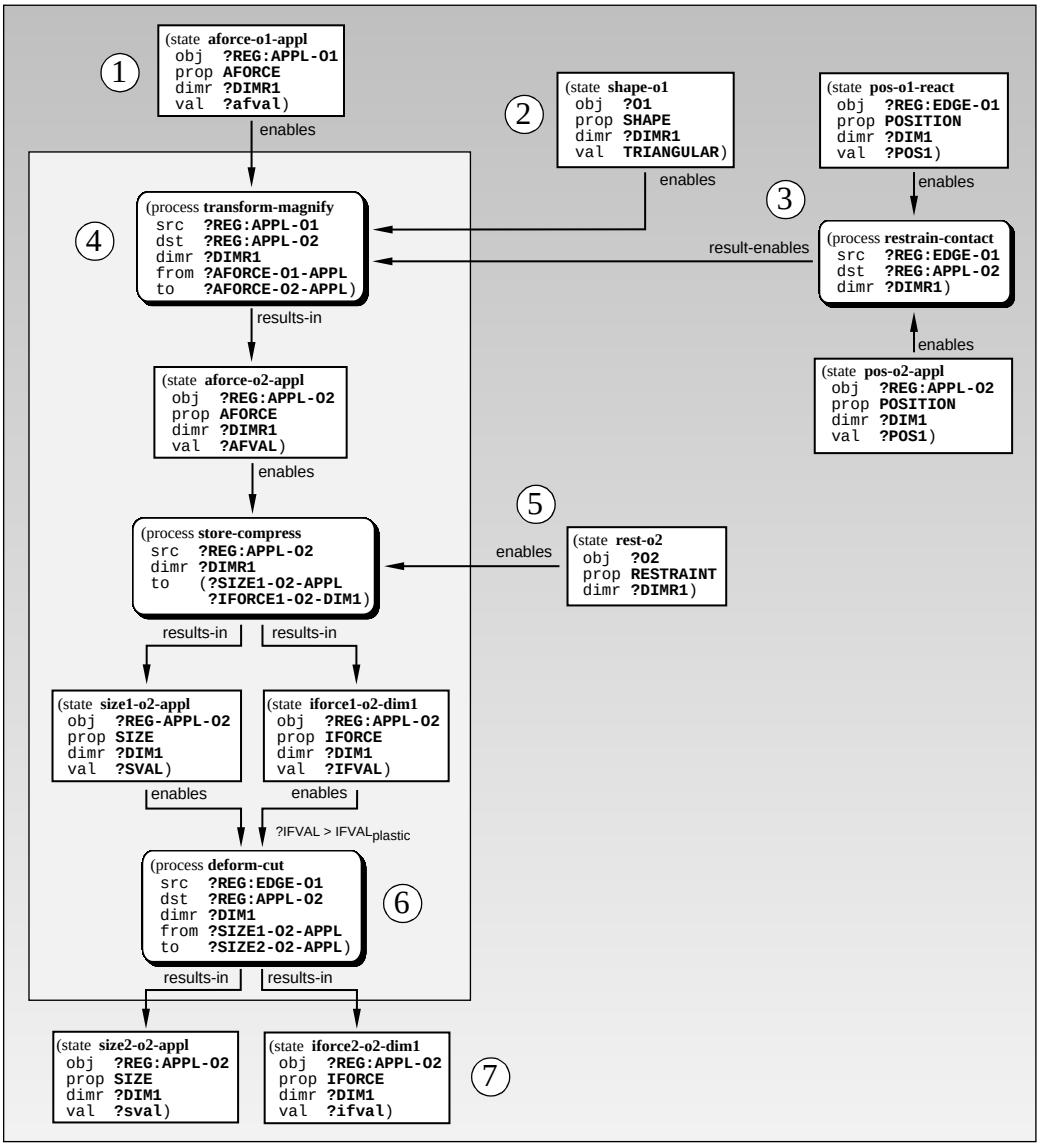


Figure 4.31 Behavioral process sequence associated with the BLADE machine primitive.

Fig. 4.31 depicts the behavioral processes which describe blade function. The shaded box

represents the intra-object dynamics of the blade object. The force (AFORCE-O1-APPL, at label 1), triangular shape (2) of the blade object (O1), and edge contact (RESTRAIN-CONTACT, at 3) with the substrate (O2) enable magnification of force (4). The restraint on O2 (5) motion enables compression, and, if the applied force exceeds the breaking strength of O2, deformation is enabled (6) and O2 undergoes local deformation (7). The block version of the shaded box in Fig. 4.31 is illustrated at (1) in Fig. 4.32a. An example depicting blade object statics, with knife slicing, is shown at (2) in Fig. 4.32a. A mapping from MP-BLADE roles to function roles is shown in Fig. 4.32b.

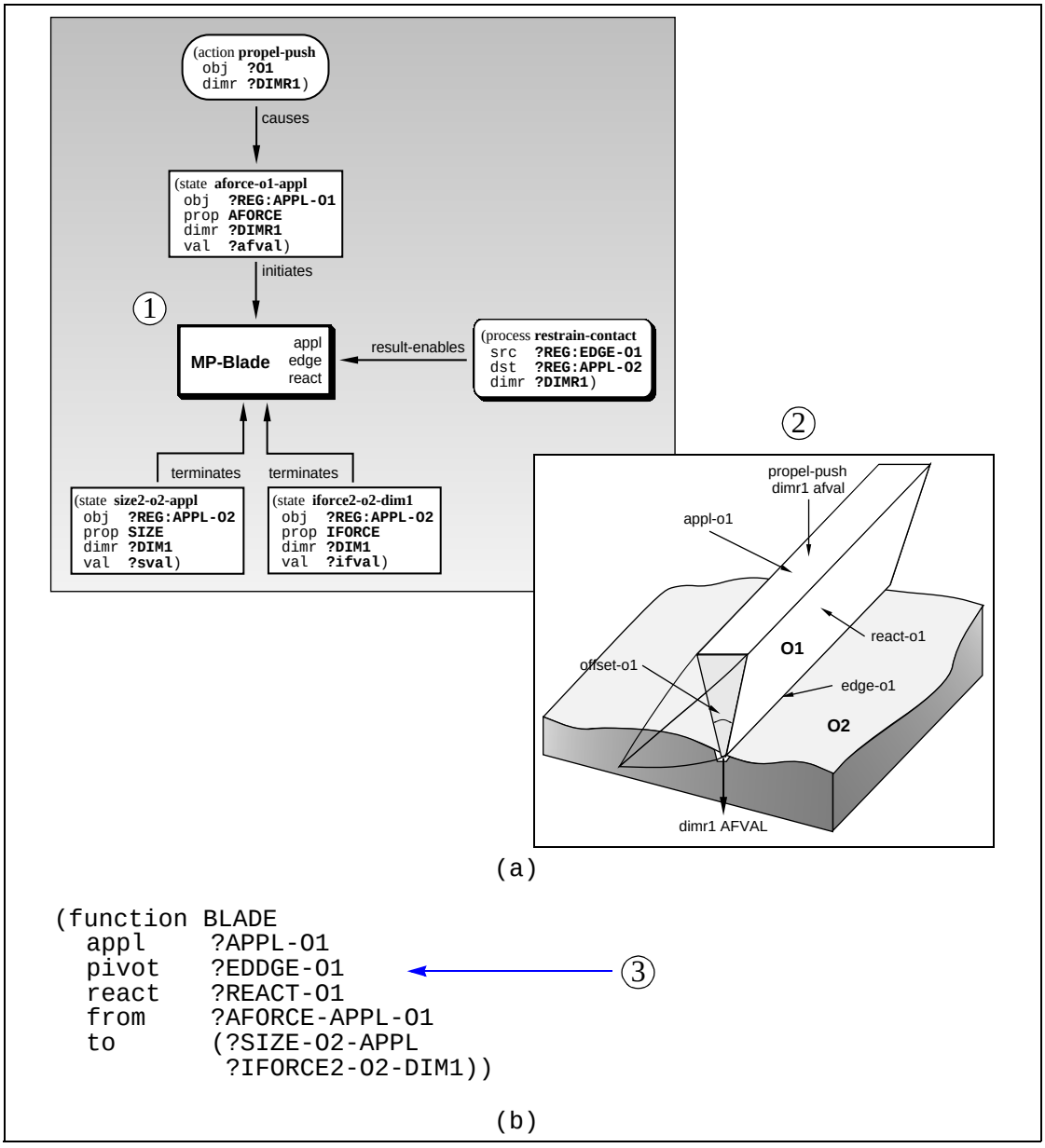


Figure 4.32 Block version of MP-BLADE function, blade object statics, and the associated slot-filler function representation.

The graphic inFig. 4.32a illustrates a knife (O1) and a substrate material (O2). The blade object (O1) is propelled (PROPEL-PUSH) in DIMR1 toward the surface of O2. The pro-

cess RESTRAIN-CONTACT between the O1 edge and O2 enables the process TRANSFORM-MAGNIFY to O2. MP-BLADE is initiated by the force AFORCE-O1-APPL. The state SIZE-O2-APPL terminates MP-BLADE.

4.2.11 MP-SCREW

MP-SCREW describes devices used to transform rotational motion to linear motion by applying rotational mechanical advantage and rotational motion advantage. A screw object is a plane object wrapped around a cylindrical or conical object, as shown in Fig. 4.33a. The screw object has three regions: appl, thread, and react. The appl and react regions are surfaces. The react region is shaped like a blade, called a *thread*, and, depending on whether the thread is sharp or blunt, the screw can be used to cut. The screw object cylindrical shape, and location of its appl region, instantiates a Type 2 MP-WHEEL-AXLE. MP-SCREW is enabled by a RESTRAIN-CONTACT between the screw object react region and the substrate material appl region. MP-SCREW is initiated by a forcing state at the screw object appl region. An enabled MP-SCREW is terminated by a force on the substrate object. MP-SCREW describes the enablements and resulting state of the process TRANSFORM-MAGNIFY, so the value of the resulting force is magnified. These relationships are depict-

ed in Fig. 4.33.

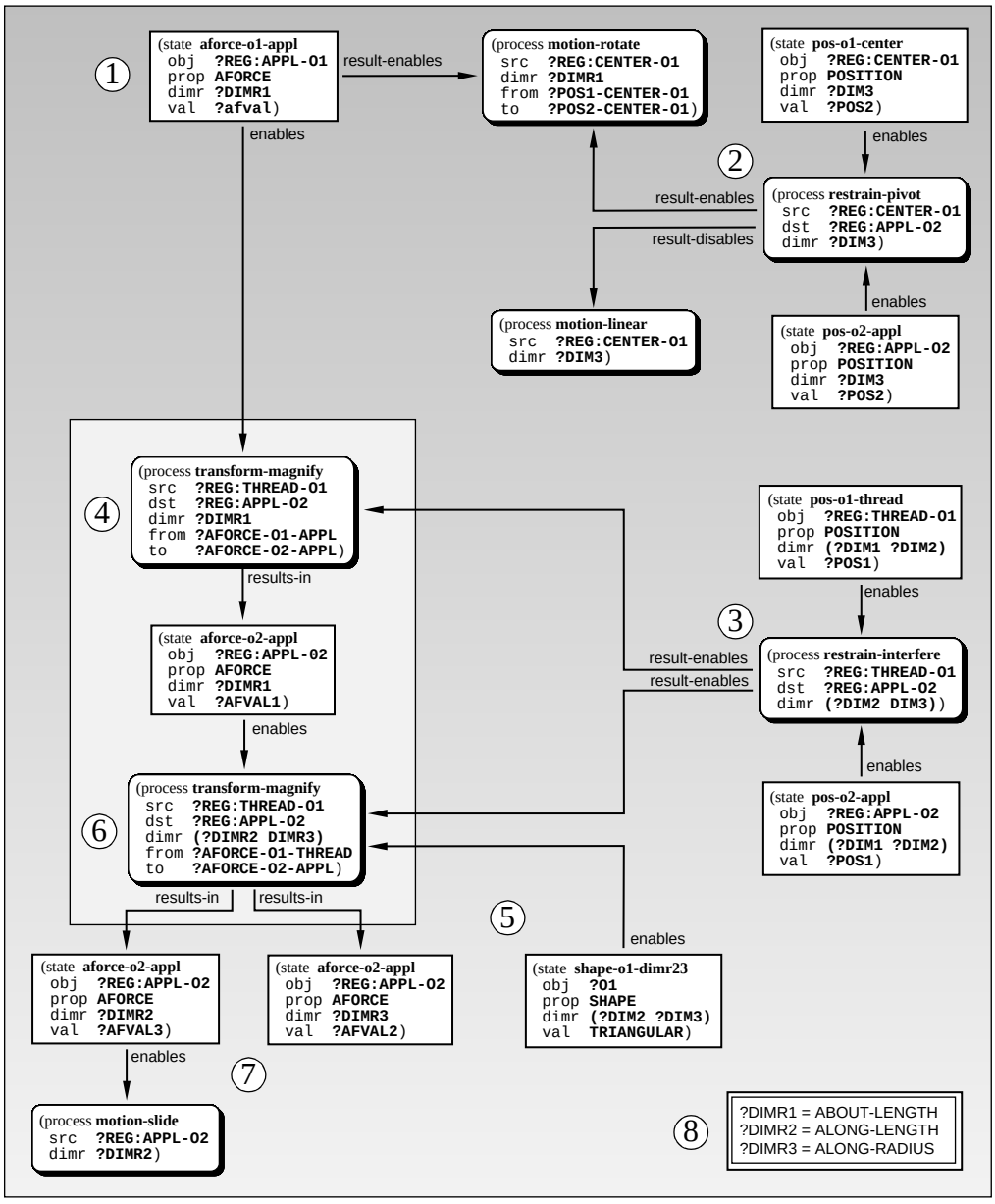


Figure 4.33 Behavioral process sequence associated with the SCREW machine primitive.

Fig. 4.33 illustrates the behavioral processes which describe screw function. The shaded box represents the intra-object dynamics of the screw object. The force (AFORCE1-O1-APPL, at label 1) enables rotation about the center, which is pivoted (RESTRAIN-PIVOT, at 2) by the substrate (02). The screw object thread (a protuberance) is in interference contact (RESTRAIN-INTERFERE, at 3) with the substrate. The contact relation at (2 and 3) are identical, separated in Fig. 4.33 to show the different effects they have on screw function. The pivot and applied force enable rotational mechanical advantage (TRANSFORM-MAGNIFY, at 4), whereas the applied force, interference, and triangular shape (6) enable rotational motion advantage (TRANSFORM-MAGNIFY, at 7). The result is applied force

in a translational dimension. The legend at (8) provides an example of the dimension dependencies between the processes in Fig. 4.33, and corresponds to those depicted at (2) in Fig. 4.34a. The block version of the shaded box in Fig. 4.33 is illustrated at (1) in Fig. 4.34a. The role mapping between MP-SCREW and the generic function class is shown with the slot-filler notation for MP-SCREW function in Fig. 4.34b.

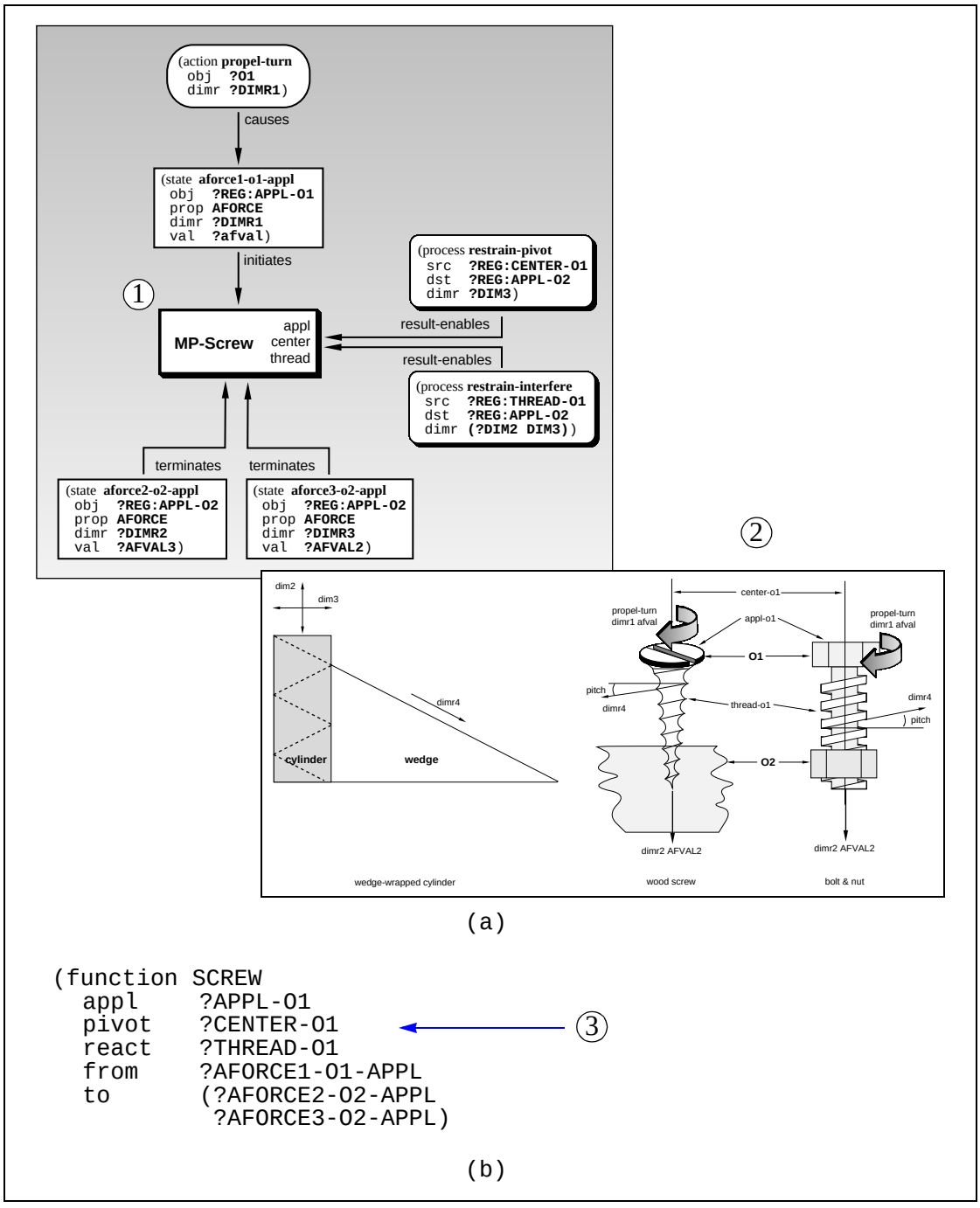


Figure 4.34 Block version of MP-SCREW function, screw object statics, and the associated slot-filler function representation.

The graphic in Fig. 4.34a illustrates the concept of screw objects and three screw objects: a wood screw, a bolt, and a nut (the bolt and nut are illustrated together). The wood screw object (O1) has a head and a point. Its threads (THREAD-O1) are wedge-shaped. This screw object is a combination of MP-SCREW and MP-BLADE, because the wedge-shaped threads and point both instantiate MP-BLADE.

4.3 MP Combinations and Mechanical Device Representation

A device has one static description at any point in time and that description can change based on the device's behavioral interactions with other objects. A device has one dynamic description for each function it can be applied to, but the function, being a dynamic description of the device, doesn't change with time. Behavioral processes can be used to describe an object's function, but they tend to be too low-level for most mechanical problem solving tasks, and are best suited for simulation and diagnostic analysis. Machine primitives can be associated with device components or the functions of device components, and can be combined to describe the functions of entire devices. Two machine primitives can be combined (e.g., the gears in Fig. 4.17 and the pulleys in Fig. 4.19) as long as their roles, enabling preconditions, and resulting states match. Machine functions can be conceptually put together, using the block versions of machine primitives (e.g., Fig. 4.10), like Tinker Toy pieces.

Consider a screwdriver like the one illustrated in Fig. 4.35a, which is designed to drive screws. The same screwdriver device can also be used to pry the lid off a can or to puncture materials and so it has more than one function, depending on the context in which it is applied. Fig. 4.35 illustrates the relationship between device statics and the representation of device functions using MP blocks. The screwdriver in Fig. 4.35a consists of three regions (or components): HANDLE-SD, SHAFT-SD, and TIP-SD. Fig. 4.35b shows the FONM idealizations of the screwdriver components and their static appearance. Fig. 4.35c illus-

trates the FONM static representation of the screwdriver.

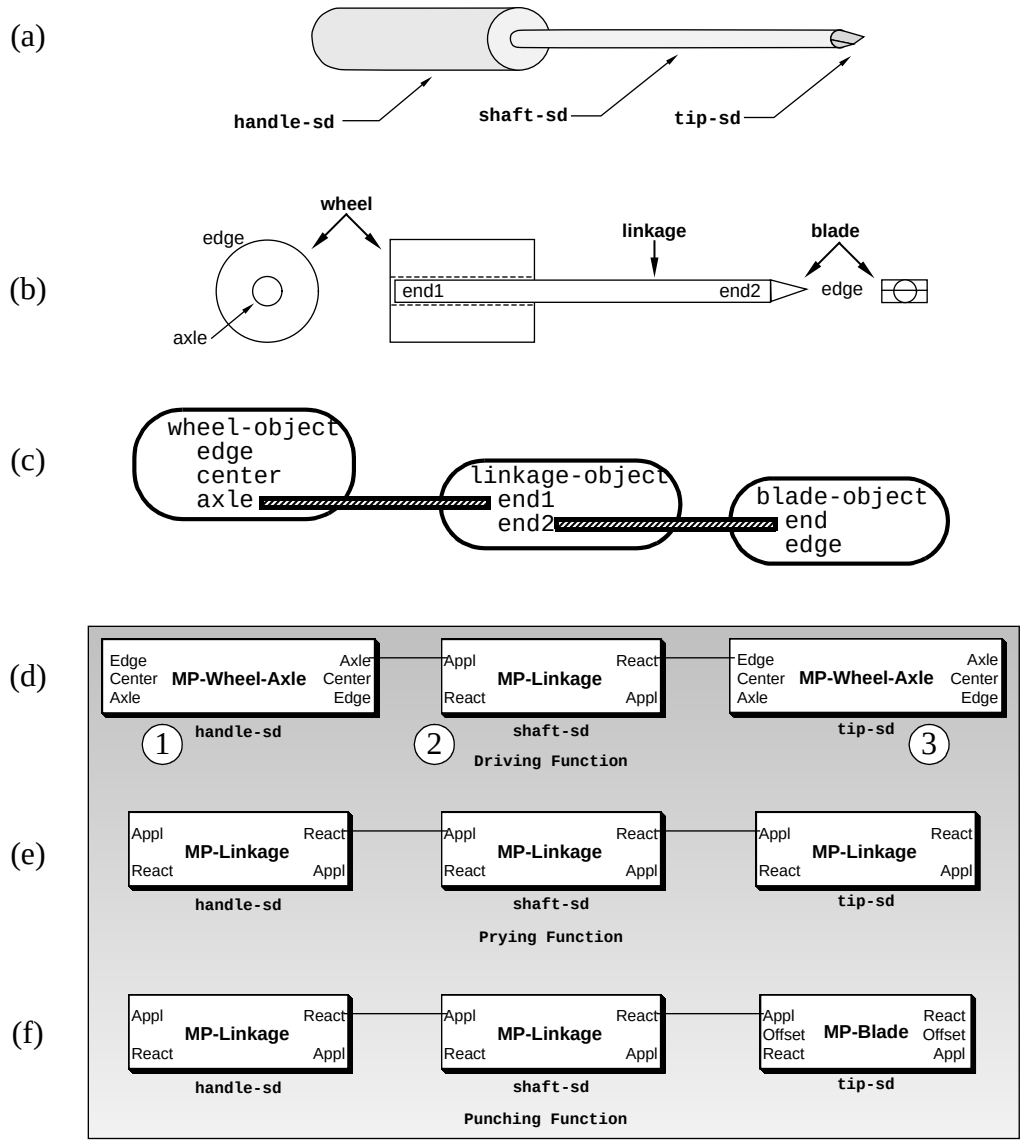


Figure 4.35 Screwdriver driving function as a combination of machine primitives.

A representation of the screwdriver driving function can be constructed from machine primitive blocks, as shown by the MP-Diagram in Fig. 4.35d. Using the same static representation, two alternate MP-Diagrams, one each for the prying and punching functions, can be constructed, and are illustrated in Fig. 4.35e and Fig. 4.35f.

The MP blocks in these figures are related to one another through links which represent the dynamic interaction of their instantiating components. For example, the static diagram in Fig. 4.35c shows that the screwdriver handle (HANDLE-SD) and shaft (SHAFT-SD) are rigidly connected. The links in Fig. 4.35d-f don't specify the type of connection, because the primitives must each satisfy the other's preconditions. Labeling the connection between components as a dynamic link indicates how force and motion are propagated throughout

the device. For example, MP-WHEEL-AXLE magnifies force at its axle, which is where the screwdriver shaft is connected. The force resulting from MP-WHEEL-AXLE both initiates MP-LINKAGE and specializes MP-LINKAGE to MP-LINKAGE-TYPE3 (torsional).

Each MP-Diagram in Fig. 4.35 (d-f) represents the dynamics of the screwdriver under a different applied force. In Fig. 4.35d, the force is applied by turning the handle, so it is a rotational force (or torque) that is applied. Fig. 4.35d represents the screwdriver driving function. The function of the handle component (HANDLE-SD) is represented with MP-WHEEL-AXLE (1), and is used to produce the necessary torque for driving screws. The function of the shaft (SHAFT-SD) is represented with MP-LINKAGE (2), and extends the torque force from HANDLE-SD to TIP-SD where the screw is located. The tip is wedge-shaped so that it will fit the screw slot, but magnifies the applied torque so its function is represented with MP-WHEEL-AXLE (3).

In Fig. 4.35e the applied force is lateral, and in Fig. 4.35f the applied force is longitudinal. Since machine primitives are represented with their initiating conditions, the effect of changing the type of applied force is to change the primitives which can represent what the associated object can be used to accomplish. The lateral force cannot instantiate MP-WHEEL-AXLE unless there is a different kind of pivot (as for a crank), but it can instantiate MP-LINKAGE-TYPE2. Similarly, the longitudinal force can instantiate MP-LINKAGE-TYPE1. The primitives in a combination are dependent on one another, to satisfy their enablements, so changing the primitives that can be instantiated by the applied force has an effect on the representation of the entire function. The functions illustrated in Fig. 4.35e and f show that the same static representation and a different applied force can instantiate different functions.

4.4 High-Level Device Representations

This section illustrates the dynamic representations of the 6 devices whose static and low-level dynamic representations were presented at the end of Chapters 2, and 3. The four device types presented: (1) simple devices, (2) simple compound devices, (3) multiple compound devices, and (3) complex devices are revisited with an emphasis on their high-level dynamics. The screwdriver, for example, is a simple device. Statically it is simple because all of its components are rigidly connected. Dynamically it is simple because all of its components move together, so the device function is effectively the function associated with the handle. That is, if one turns the handle, the entire screwdriver functions like an MP-WHEEL-AXLE, and if one pushes on the handle the entire screwdriver functions like an MP-LINKAGE. When an entire device functions as the machine primitive which one or more of its components instantiates, it is called a *generalized machine*. The screwdriver in the driving function is a generalized MP-WHEEL-AXLE.

The device representations presented in the following sections will take the form of the first example in this section. An idealized illustration of a possible device geometry, based on object primitives, will be shown, along with a static representation diagram for the primitives and one or more MP-Diagrams for the device pictured. It is assumed that the behavioral sequences associated with each MP block, and in particular the inter-object dynamics (e.g., motion) can be instantiated for each MP-Diagram, rather than fleshing them out explicitly. When appropriate, representation examples will be presented. As the static description becomes more complex, fewer independent MP-Diagrams will be presented, but different ways to perturb them will be discussed.

4.4.1 Simple Devices

Simple device dynamics are representationally significant because even simple devices can

be used to accomplish multiple tasks. The dynamic representation must make a device's different functions explicit with the same static representation. The distinction between device functions is often seen as a specialization on the type of input applied to a device, or on the physical properties of objects which comprise the device. Simple devices are also useful for showing that device descriptions which represent the same function, whether based on components or regions, are equivalent. The high-level dynamic representations of two simple devices: (1) a bottle opener, and (2) a screwdriver, are presented in this section. Each device is comprised of components which are rigidly connected, or regions of a single component. The high-level dynamic representations for four additional devices: (3) a carving knife, (4) a hammer, (5) a shovel, and (6) a spoon are presented in Appendix A.1.

Bottle Opener

A bottle opener is designed to open cans using one end and to open bottle caps at the other. Both ends make use of mechanical advantage, but one end is pointed so the magnified force is used to puncture. An idealized bottle opener is represented as two lever-objects and a blade object, as shown in Fig. 4.36 (redrawn from Figs. 2.58, 2.59) below.

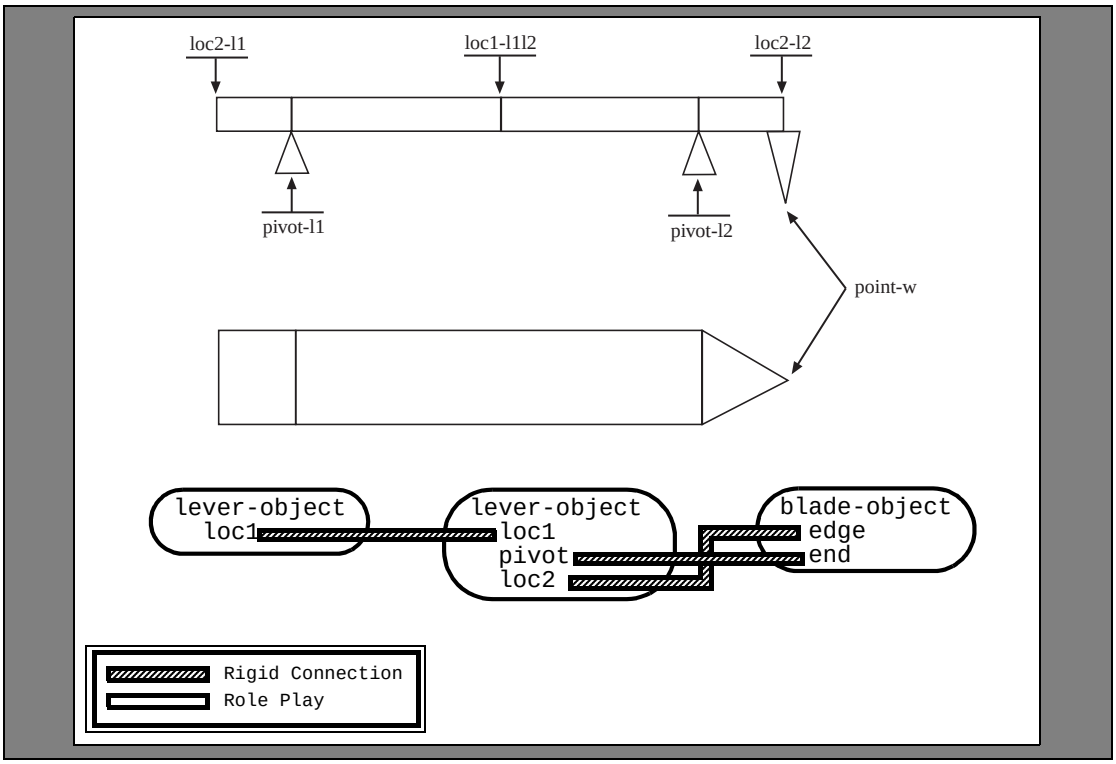


Figure 4.36 Bottle opener object primitives and static device diagram

The static representation for the bottle opener depicted in Fig. 4.36 can instantiate different functions depending on how and where the force is applied. Three bottle opener functions are represented as MP-Diagrams in Fig. 4.37: lid puncturing, lid prying, and a probing function. In each case, the regions shown in Fig. 4.36 and Fig. 2.59 are shown instantiating the roles of the machine primitives they instantiate in the respective function. The legend in Fig. 4.37 describes the arcs between the blocks, and will be replicated in all MP-Diagrams in this section. Each function depicted differs by where the force is applied and what type

of force is applied.

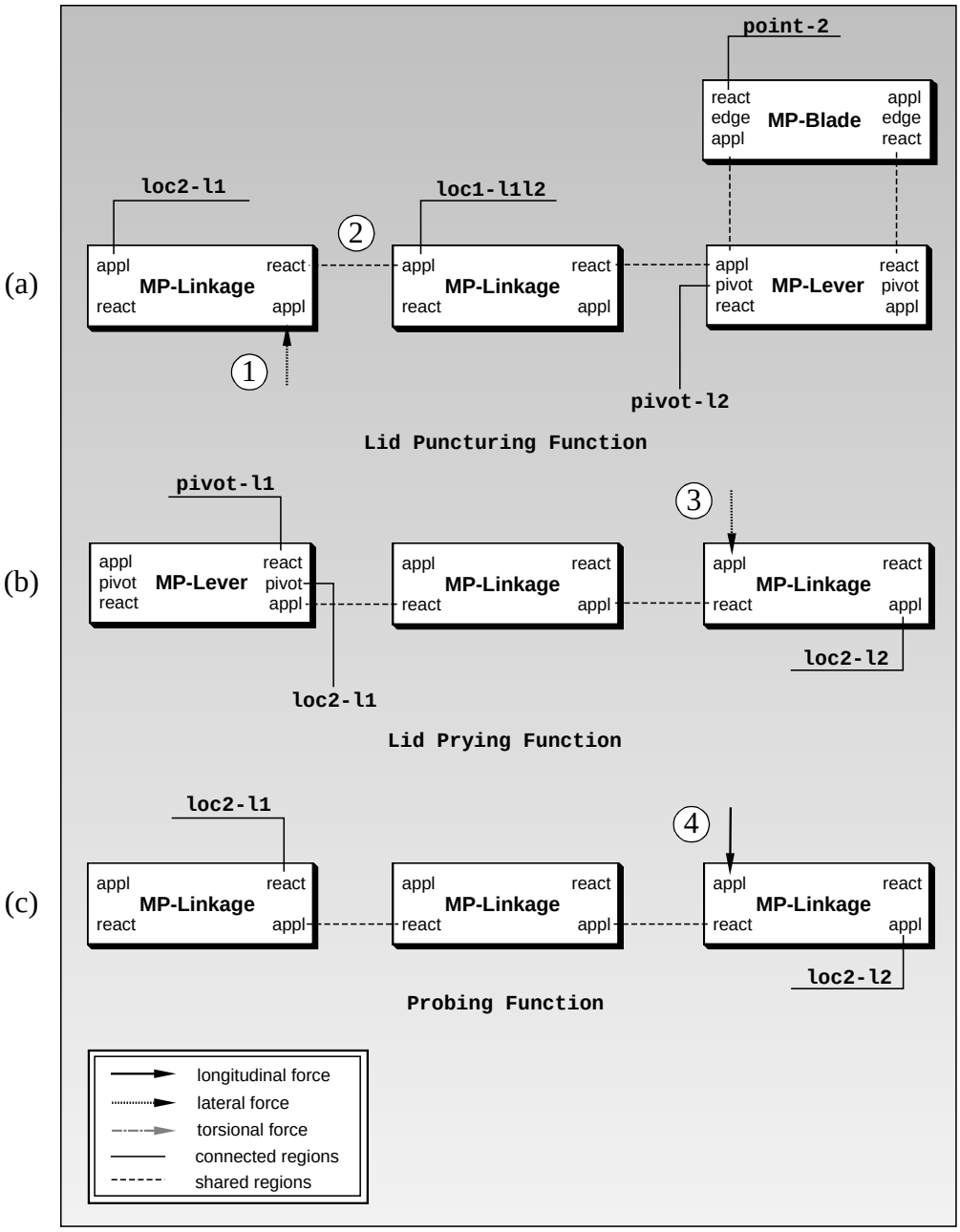


Figure 4.37 MP-Diagrams for bottle opener functions.

The input which identifies the bottle opener function for puncturing cans is a lateral applied force at the end opposite the wedge object (region LOC2-L1, shown at 1). The applied force is lateral, which, based on the static device representation, enables MP-LINKAGE. The connection between the objects is illustrated in the figure as a dotted line (2), which means that the roles of the respective machine primitives are instantiated by the same object region. In the idealized bottle opener, the device is represented as three objects connected together, so this arc could also be a solid line. The central component transmits the lateral

force from one end to the other, so it is represented with MP-LINKAGE also. The pointed end has a pivot location and, when the point is in contact with a can, can be represented with MP-LEVER-TYPE1.

The lid prying function illustrated in Fig. 4.37b is similar to the lid prying function, in that the same type of force is applied on the other end of the device (3), and the component functions are all represented with the same machine primitives. However, when applied in removing bottle caps, the end of the bottle opener becomes the region which instantiates the lever pivot role, and so the component is represented with MP-LEVER-TYPE2. The difference is in the shape of the components which instantiate the primitives. The pointed end of the bottle opener cannot be applied as a pivot without enabling the DEFORM process.

The last function illustrated shows the effect of changing to a longitudinal force (4). Similar to the functions illustrated in Fig. 4.37a and b, had the force been applied at the other end, another function would be instantiated, that of longitudinal puncturing. It is significant to note the graphical similarity between the lid puncturing function and the lid prying function. In general, devices which instantiate the same combination of machine primitives are functionally similar (but not necessarily functionally equivalent).

Screwdriver

The idealized screwdriver and its static device diagram are depicted in Fig. 4.35a-c. The functions illustrated in Fig. 4.35d-f are re-illustrated in Fig. 4.38 for continuity.

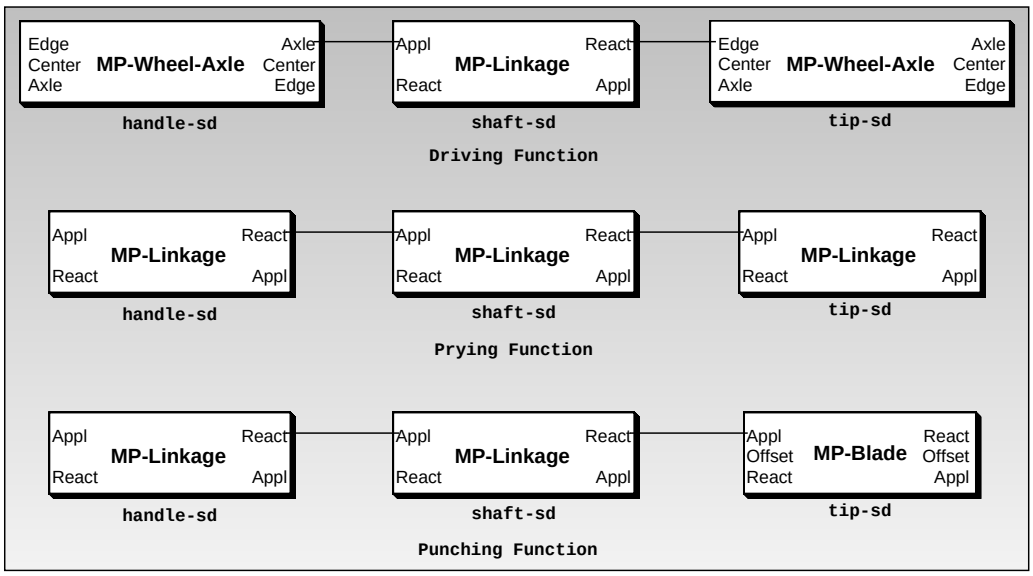


Figure 4.38 MP-Diagrams for screwdriver functions.

4.4.2 Simple Compound Devices

Devices in which relative motion between components is possible are *compound* devices. Compound device dynamics are representationally significant because components can function independently. This has two affects on the representation and interpretation of compound devices. First, if a device component is not rigidly connected to other components, then the functions it can instantiate can be invoked without considering the other de-

vice components. Second, if a machine primitive participates in a device's function representation, then its function can be considered independently of the function in which it participates. In this section, two devices: (1) a nutcracker, and (2) a clothes pin are presented. Appendix A.2 presents the MP representations of three additional compound devices: (3) a pair of scissors, (4) a pair of pliers, and (5) a door.

Nutcracker

A nutcracker is used to restrain the motion of and crack the shell of a nut. The nutcracker and pliers are similar in that they perform the same function in different ways. An idealized nutcracker and its static device diagram are illustrated in Fig. 4.39.

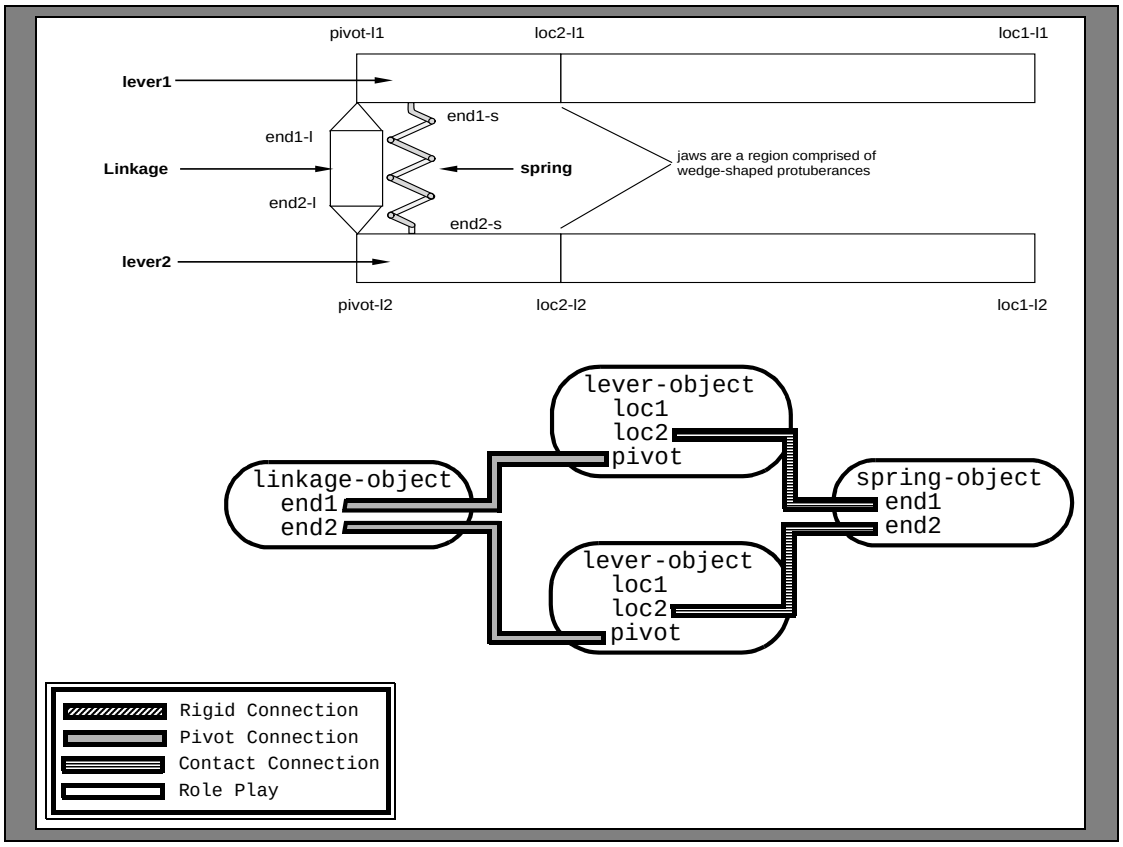


Figure 4.39 Nutcracker object primitives and static device diagram.

The cracking nutcracker function is depicted in Fig. 4.39. The function is initiated by pressing the handles together (at 1 and 2). The handles, being pivoted at their ends, can rotate, and independently instantiate MP-LEVER-TYPE2. When an object is placed between and in contact with each handle they work together to provide the leverage needed to exceed the object's breaking strength. The nutcracker can also instantiate the pliers clamping func-

tion, the hammer pounding function, and the bottle opener probing function.

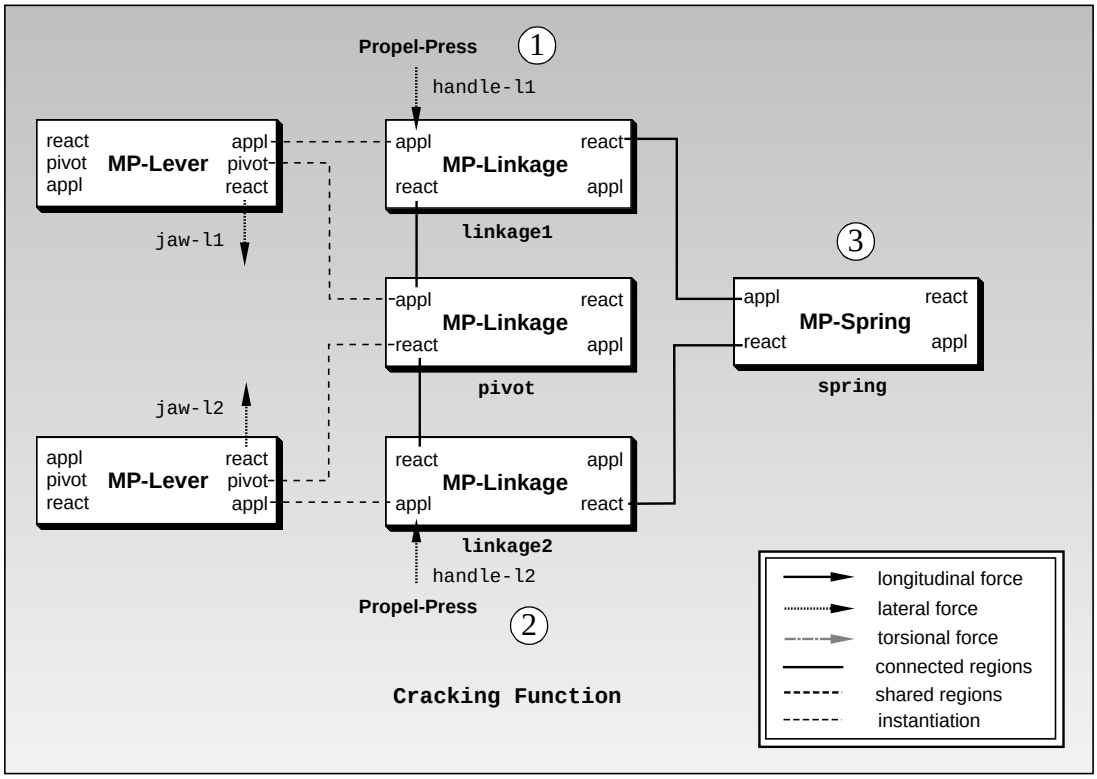


Figure 4.40 MP-Diagram for nutcracker cracking function.

In Fig. 4.39, the two handles are initially represented as linkage-objects. However, when an object is place in contact with the jaw regions, the linkage-object/pivot/object combination instantiates an MP-LEVER-TYPE2. To instantiate a hammering function, or the bottle-opener probing function, the nutcracker handles are used as linkage-objects directly.

Clothes Pin

A clothes pin holds objects together without manual control. The device is comprised of three components: two lever-objects and a spring-object. The spring is employed to keep the levers together, as well as to keep the jaws closed, as shown in the idealized clothes pin

and static device diagram depicted in Fig. 4.41.

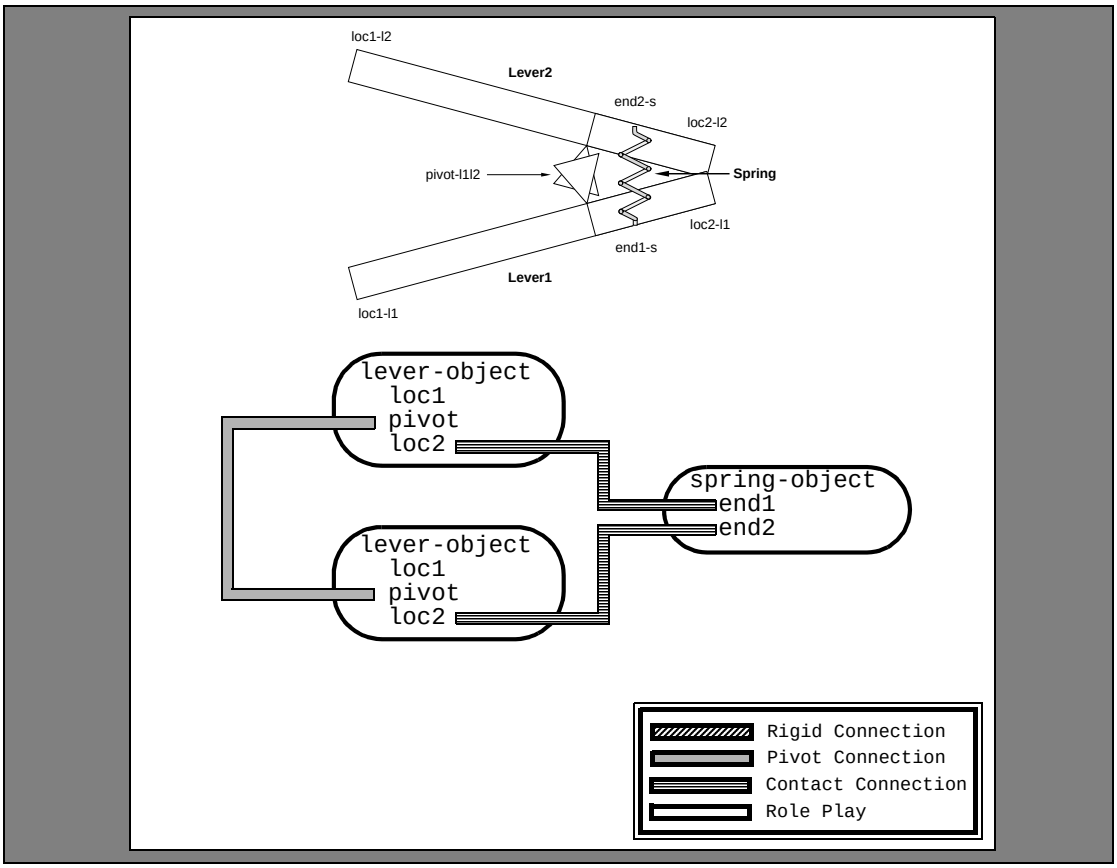


Figure 4.41 Clothes pin object primitives and static device diagram.

The static representation diagram for the clothes pin is almost identical to that of the nutcracker in Fig. 4.39, but their functions are dissimilar. In the clothes pin, the device is normally closed and the spring keeps it that way, whereas in the nutcracker the spring keeps the handles apart. In both devices the spring returns the device to its initial state, unlike scissors and pliers. In the clothes pin, the spring is part of the pivot and the lever object LOC2 regions are on the opposite side from the LOC1 regions. The effect is that LOC2 and LOC1 move in opposite directions, so having the spring in contact with this region provides opening resistance. The clothes pin clasping and probing functions are shown as MP-Diagrams in Fig. 4.41. In the clasping function, the handles are pressed together, which causes a lat-

eral force (at 1) in both handles.

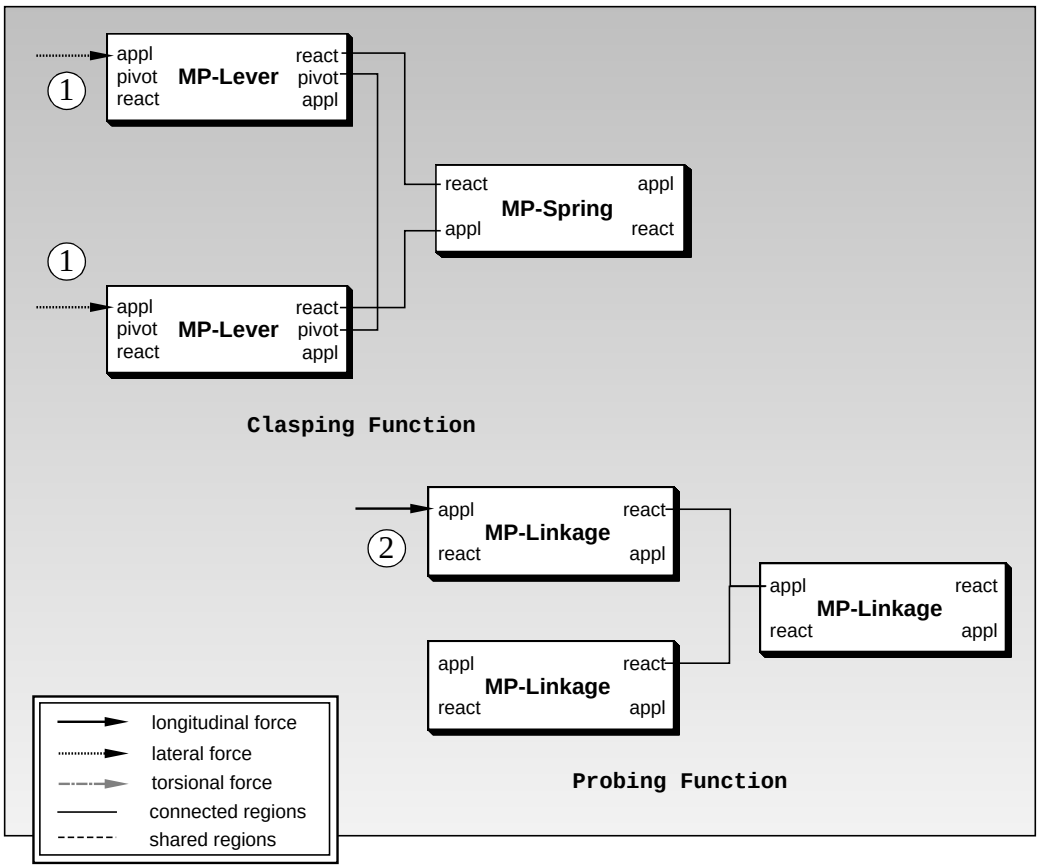


Figure 4.42 MP-Diagram for clothes pin function.

The lever pivot roles are shared, and the spring is connected to the lever react role regions. The probing function is initiated by a longitudinal force (2) and is represented with MP-LINKAGE.

4.4.3 Multiple Compound Devices

Whereas a pair of scissors or a clothes pin have two components which can move independently, a multiple-compound device has more than one independently moving component. Although the statics and intra-object dynamics are fairly straightforward, the inter-object dynamics begins to become complex when fully elaborated. One such device is the fingernail clipper, whose MP representation is presented here. Two additional devices, a press and a crank can opener are presented in Appendix A.3.

Fingernail Clipper

A fingernail clipper clips nails by forcing two blades together. The force necessary to clip is achieved with a lever, and the device is returned to its initial state with a spring. An idealized fingernail clipper and its associated static device diagram are shown in Fig. 4.43. The only components which are rigidly connected are the linkages. The handle can move independently of the blades, and the blades can move with respect to each other. The handle is

pivoted to the pin (at 1 in Fig. 4.43). The spring is illustrated as playing the role of `end1` in both linkage objects, rather than being connected to them, to emphasize that the linkage objects are being used as springs.

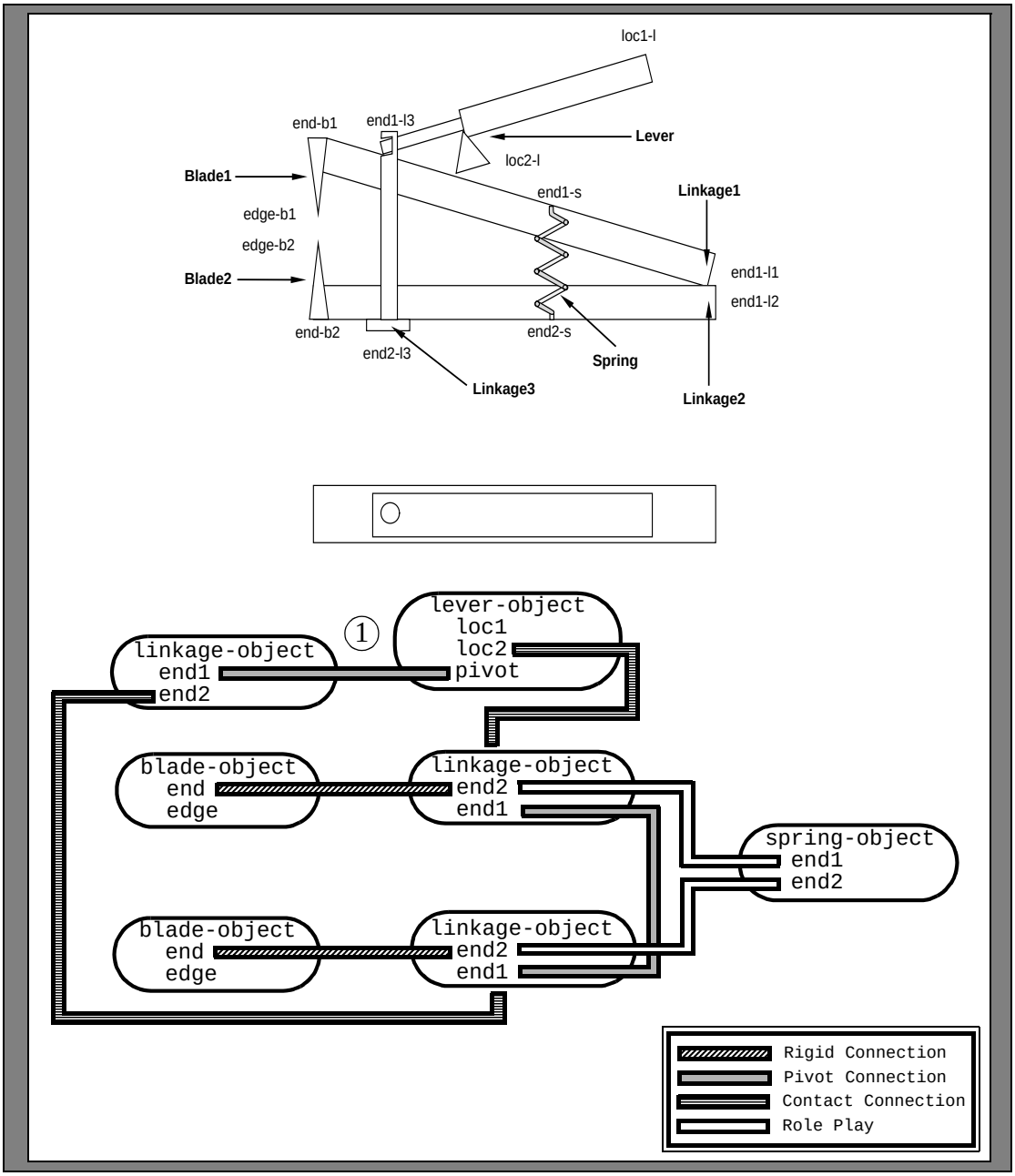


Figure 4.43 Fingernail clipper object primitives and static device diagram.

The fingernail clipper function is represented as an MP-Diagram in Fig. 4.44.

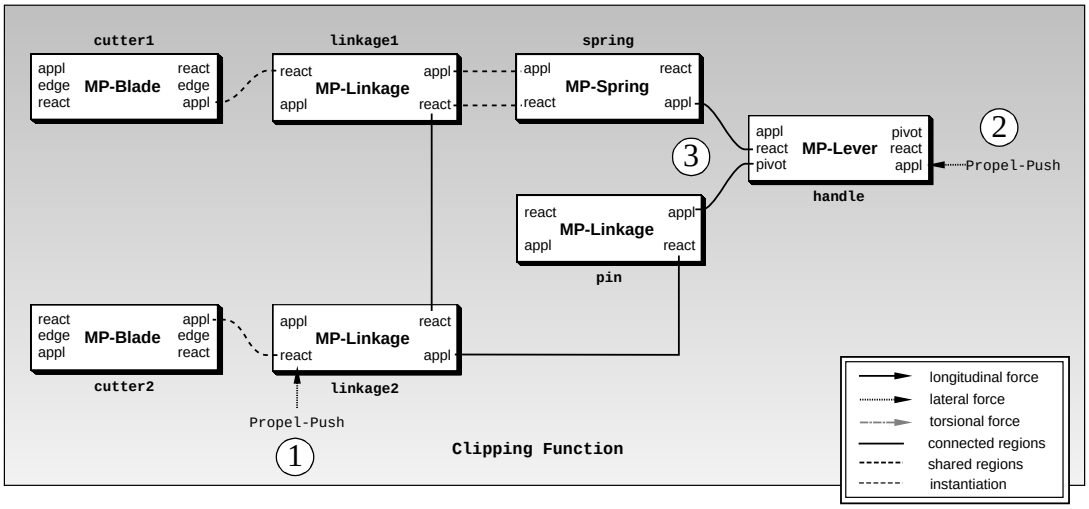


Figure 4.44 MP-Diagram for fingernail clipper function.

The clipping function is initiated by holding the lower piece (at 1) and pressing the handle (at 2). The handle is pivoted (see 1 in Fig. 4.43) to the pin and is also in contact with the upper piece (3). This is represented with MP-LEVER, which magnifies the applied force at the upper piece, while the pin acts as a fulcrum. As the handle rotates, the magnified force increases, and the blades move closer together. The shape of the blades enables further magnification with MP-BLADE. The resulting change in finger nail size terminates the function. The device is reset by releasing the handle (not shown in this figure), whereupon the blades open and the handle returns to its original position.

4.4.4 Complex Devices

Complex devices involve objects which may or may not be permanent components of the device, have complex motions, or have large numbers of components. One such example is a toy gun, which has a projectile which is only connected to the device when being armed and preparing to fire. Another complex device is a mousetrap, since the hook and baitplate motion are both confined to the limits imposed by their connectors, but free to move in every other respect. In this section, the toy gun's MP block representation is presented, while that of the mouse trap, a corkscrew, and an egg beater are presented in Appendix A.4.

Toy Gun

The idealized toy gun and static device diagram are illustrated in Fig. 4.45.

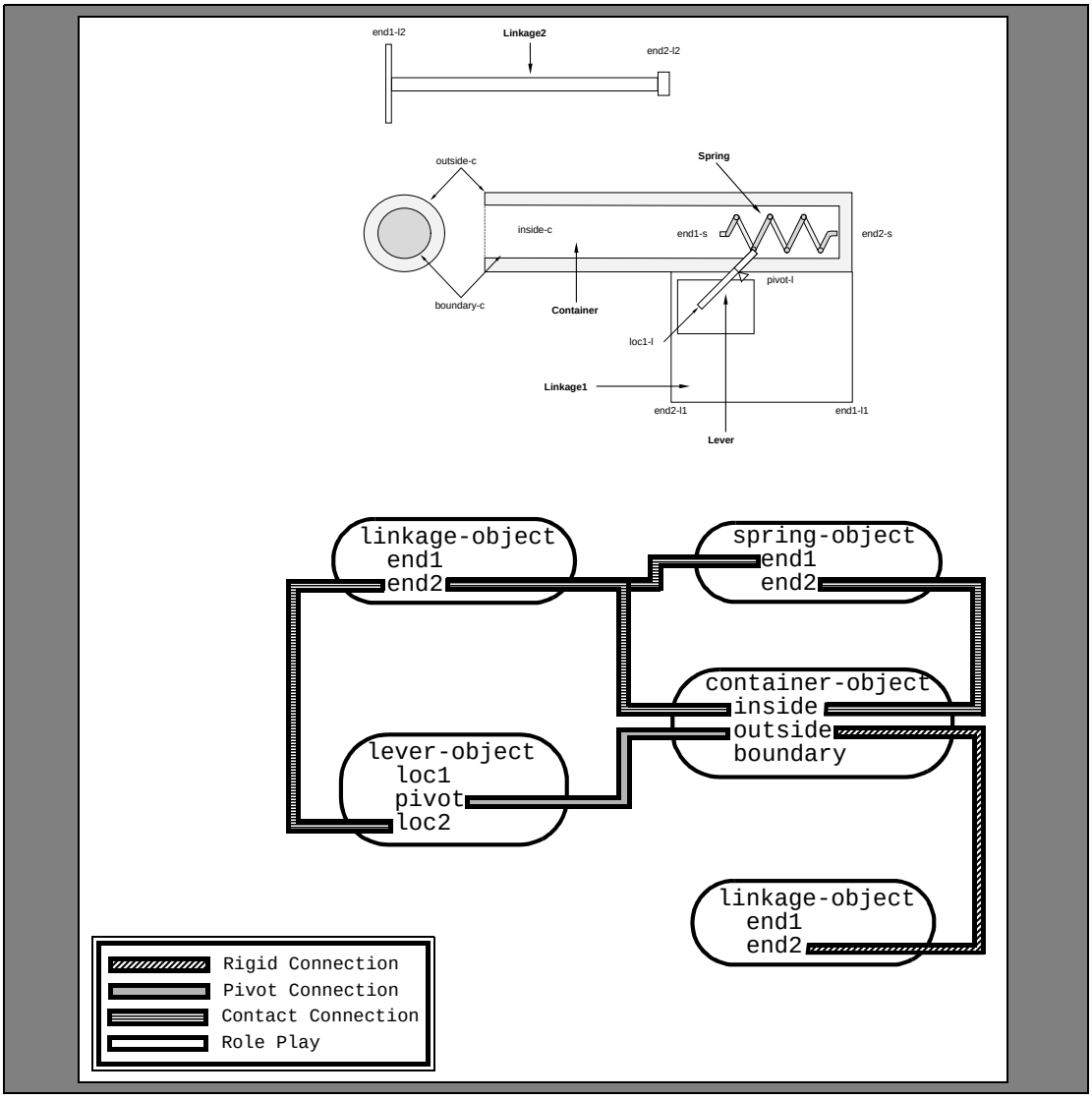


Figure 4.45 Toy gun object primitives and static device diagram.

The arming and firing functions of the toy gun are illustrated in Fig. 4.46. In both functions, the handle is held and does nothing more than transmit force to the barrel and other components. It is thus represented with MP-LINKAGE. In the arming function, the dart is pushed into the barrel. The dart is a separate object which comes into contact with the device during both functions. The force produced by pushing the dart initiates the arming

function.

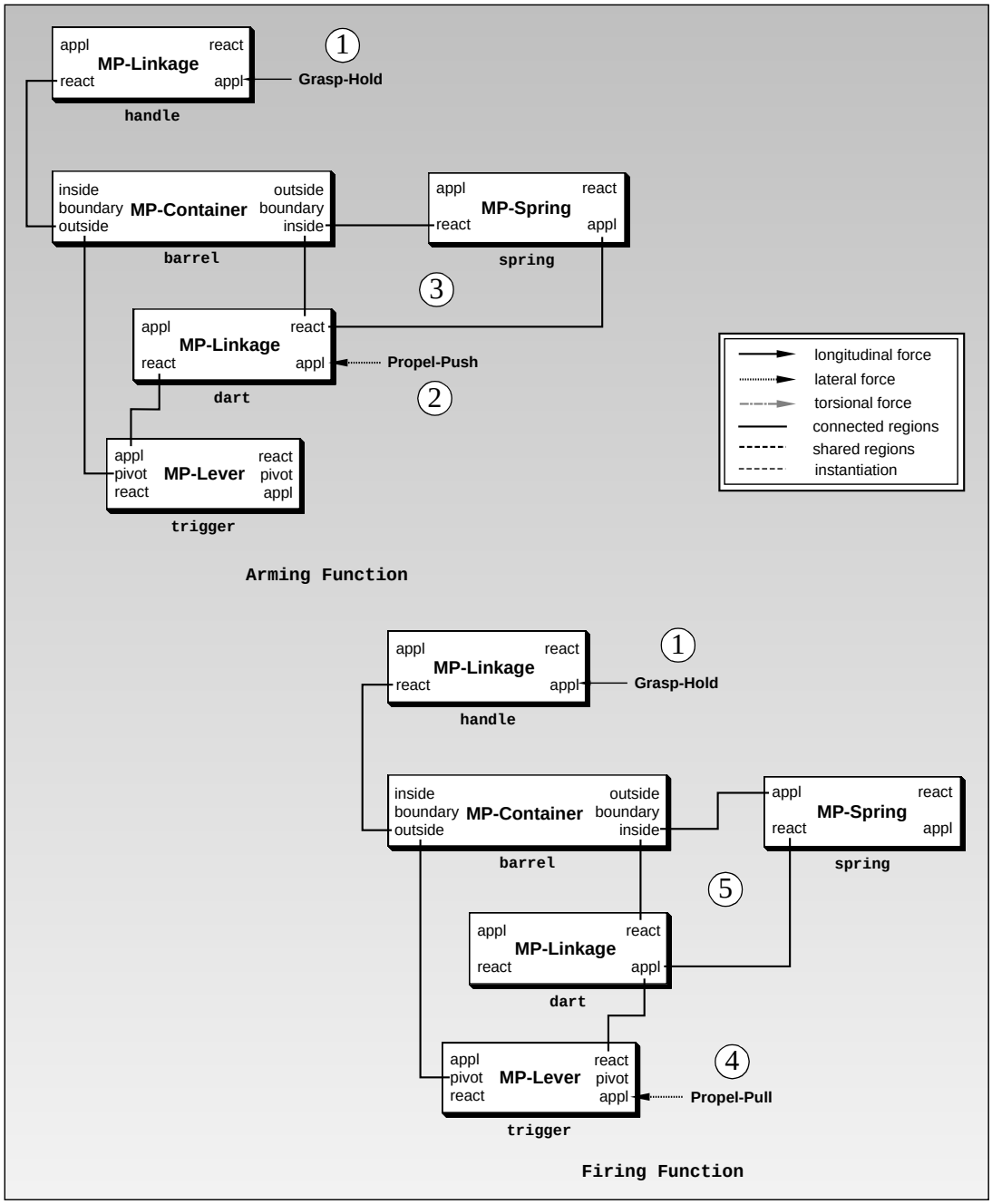
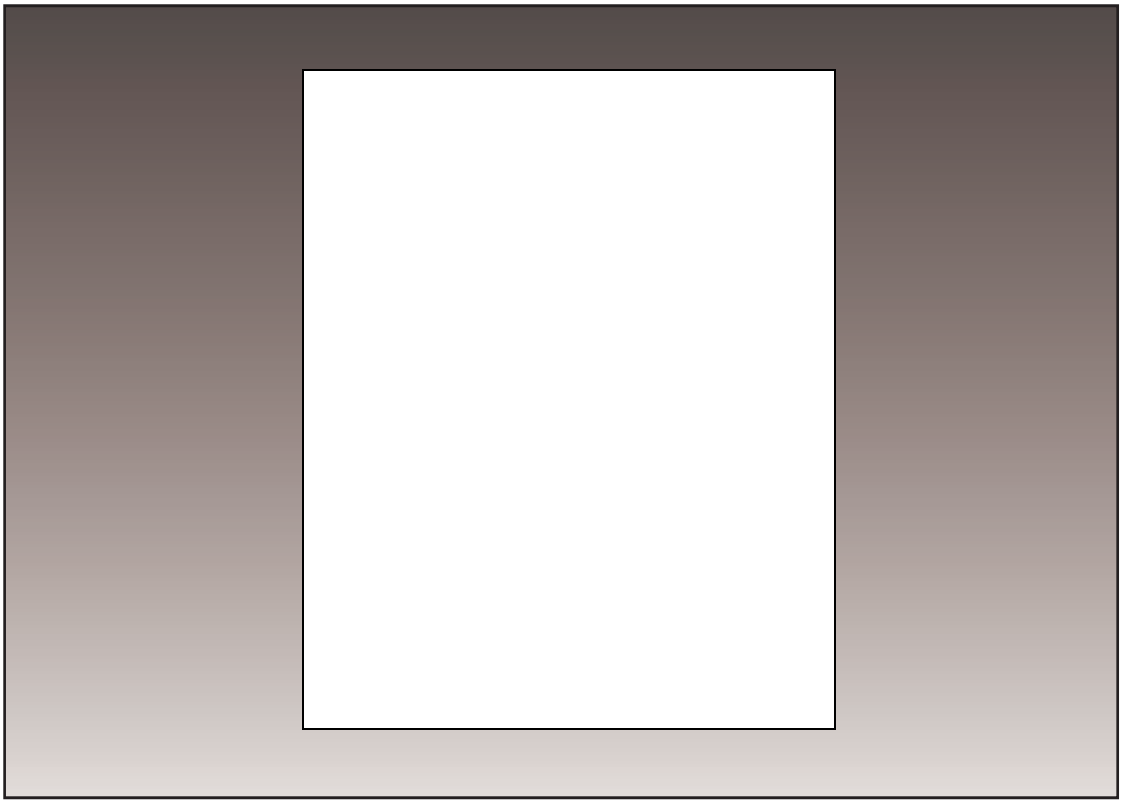


Figure 4.46 MP-Diagrams for toy gun functions.

The role of the spring and how MP-SPRING is instantiated differs in the two functions. In the arming function, the dart compresses the spring until the dart catches the trigger (3). In the firing function, the trigger is pulled (4), disengaging the dart, and the spring is released (5).



Chapter 5
Device Pragmatics

Devices are used to accomplish tasks. *Device pragmatics* represents the use of devices with respect to what the user's intentions and actions are and with respect to the device's function and behavior. Two topics are addressed in this chapter: (1) device use plans, and (2) device related subgoals.

The first section of this chapter introduces the notion of a *device-use* plan, which represents how a device is applied once the decision of what needs to be done has been made. The actions and plans associated with device-use enable a particular device function.

The second section of this chapter introduces the notion of device-related goals, which represent state changes a problem solver must effect that require device behavior and function. Device-use plans are comprised of subgoals, some of which are device-related goals, so these goals help to identify the need for device application, and to represent the causal relationship between the plans in which devices are used and the functions which they are used to produce.

The third section of this chapter presents pragmatic (device-use) representations for four device classes: simple devices, simple compound devices, multiple compound devices, and complex devices.

5.1 Device-Use Plans

A plan in which a device function is applied is called a *device-use* plan. The device-use plan is an adaptation of Schank and Abelson's USE(object) plan schema [Schank and Abelson, 1977]. Device-use plans have five roles: actor, inst, obj, from and to. The *actor* role is filled by the agent who performs the actions.The *inst* role is filled by the instrument used to produce the state. The *obj* role is filled by the object(s) to which the instrument is applied. The *from* and *to* roles are filled by the initial and final states described by the goal. For example, the use of a nutcracker to crack pecans is represented in Fig. 5.1.

```
(plan crack-open
  actor    EDISON
  inst     NUTCRACKER
  obj      PECAN
  from     CLOSED-PECAN
  to       OPEN-PECAN)
```

Figure 5.1 Slot-filler instantiation of device-use plan for nutcracking.

In device-use plans, a goal is achieved by actions, scripts, and plans which result in

physical control of the instrument and object. For example, the use of a nutcracker to crack pecans open requires the user to grasp the nutcracker handles (they start out spread apart), insert the pecan between the nutcracker jaws, and press the handles together, as shown in Fig. 5.2.

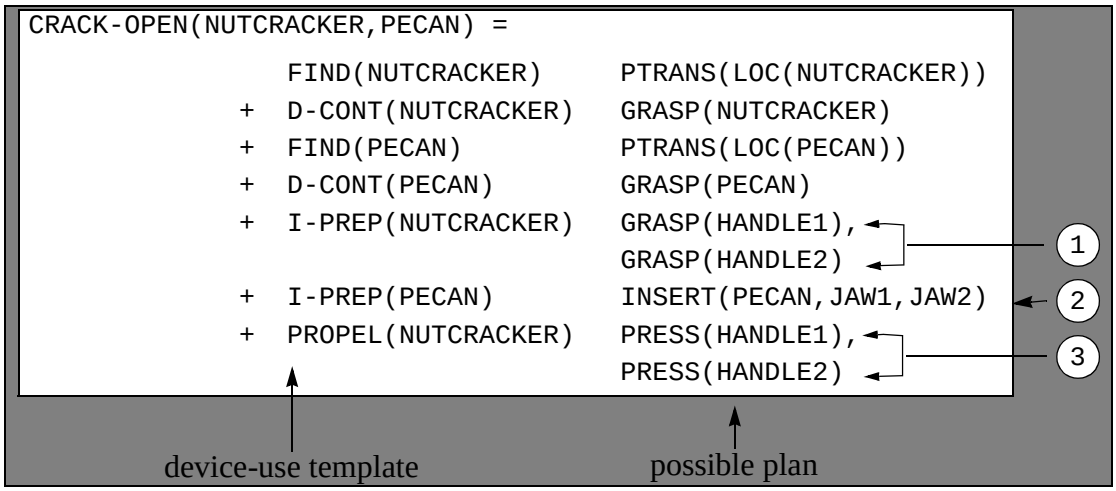


Figure 5.2 Slot-filler instantiation of device-use plan for nutcracking.

The CRACK-OPEN plan of Fig. 5.1, depicted in Fig. 5.2, is now similar to two interacting USE(object) plans, one for the instrument (NUTCRACKER), and the other for the object (PECAN). In Fig. 5.2, the FIND subgoal for both objects has been simplified to going to where the object is located (i.e., PTRANSing). The D-CONT (physical control) subgoal for each object is satisfied with a GRASPing action. The nutcracker preparation goal ensures that the handles are spread apart (i.e., to get the device OPEN), so that the pecan will fit between the jaws. A nutcracker has a spring between the handles, which is uncompressed when the device is open, so the user need not OPEN the nutcracker under normal use. However, the user must grasp the handles (shown at 1 in the figure). These actions provide a means to apply force to the handles and thereby enable the nutcracker CLOSEing plan. The preparation of the pecan involves inserting it between the two nutcracker jaws (2). INSERT is represented with an instrumental script which identifies the location and orientation of the components and results in interference between them. Thus, INSERT(PECAN,JAW1,JAW2) results in the necessary restraint state on the pecan which enables BPP-DEFORM-CRACK. Once these actions and plans have been successfully completed, the CRACK-OPEN plan is enabled, and pressing the levers together (3) initiates the nutcracker CRACK-APART function (MP-LEVER-TYPE2), resulting in the OPEN pecan state.

5.2 Device Use Goals

The representation for the CRACK-OPEN device-use plan (Fig. 5.2) assumes that there is an explicit causal relationship between the I-PREP(NUTCRACKER) goal of the user and the associated GRASP actions, and between the I-PREP(PECAN) goal and the INSERT(PECAN,JAW1,JAW2) script. Part of nutcracker use is that it must be open so that the pecan can be inserted between its jaws. In FONM these requirements specialize the preparation goals in terms of identifying the means to resolve them. In this section, the delta goals and instrumental goals which are associated with device-use plans are specialized to relate them with particular device behaviors and functions.

A device behavior or function is applied as a step in a plan to achieve a goal. Likewise, a goal may identify appropriate actions, scripts, and plans which produce specific behaviors and functions. The failure to achieve a goal with a known plan may motivate the user to try another approach rather than to abandon the original goal. Goals which represent changes in behavioral state are associated with blocked delta goals (with a "motivates" link, see Eqn. (1-11) on page 11). For example, suppose the user has an active goal to eat a pickle and that the pickle is in a closed pickle jar. The use(pickle) plan has a subgoal to gain physical control (D-CONT) of the pickle. The subgoal is blocked because the normal method for achieving control is to GRASP the pickle. One alternative is to abandon the goal and decide to eat something else. Another alternative is to open the jar. The latter choice represents a change in the jar's restraint state. In FONM, this change can only be effected by a RESTRAIN process, so the subgoal to change restraint states is directly related to plans which enable the RESTRAIN process. Each behavioral process primitive is similarly associated with a delta goal.

Now suppose that the method applied to open the jar is to GRASP the cap and then TWIST it, but that this plan fails because the lid is on too tight. Again, one option is to abandon the goal. Another is to try an approach which utilizes a device, such as running the jar under hot water, banging on the lid with an object, or finding some means to apply more torsional force to the lid. With these plans, the application of a device function may produce the desired state change indirectly, in which case its use is instrumental to subgoal achievement. In FONM, each machine primitive is associated with an instrumental goal.

5.2.1 Behavioral Delta Goals

In FONM, changes in behavioral state are represented using behavioral primitives, so the need to produce behavioral state changes implies a delta-type goal. There are five delta goal types associated with object behavior: D-POSITION, D-RESTRAINT, D-AFORCE, D-IFORCE, and D-SIZE. The pecan restraint state change needed for the pecan is represented with a delta-restraint (D-RESTRAINT) goal. Each of these delta goals describes necessary changes of a specific behavioral state and identifies the kinds of device-use and instrumental plans which can be applied to produce those state changes.

D-POSITION: A goal for producing object positional changes. Plans which achieve D-POSITION instantiate a MOTION process primitive.

D-RESTRAINT**:** A goal for changing physical restraints on an object. Plans which achieve D-RESTRAINT instantiate a RESTRAIN process primitive. D-RESTRAINT is motivated when a problem solver needs to open a bottle, for example, or to connect two engine components.

D-AFORCE: A goal for producing applied forces on an object when PROPEL is insufficient. Plans which achieve D-AFORCE instantiate a TRANSFORM process primitive. D-AFORCE is motivated when a problem solver needs to dislodge a frisbee from a tree, because an axial force applied at some distance may be sufficient.

D-IFORCE: A goal for temporarily changing object internal forces. Plans which achieve D-IFORCE instantiate a STORE process primitive. D-IFORCE is motivated when a person's watch runs down, or a child's windup toy ceases to run. In both cases, energy stored in a spring is dissipated as the device is used, and the user is motivated to retension the device to achieve a higher goal.

D-SIZE: A goal for changing size or shape of an object. Plans which achieve D-SIZE instantiate a DEFORM process primitive. For instance, D-SIZE is motivated when a problem solver needs to remove metal from objects which do not fit to-

gether, by shaving or grinding.

To illustrate the use of behavioral delta goals, consider the situation shown in Fig. 5.3.

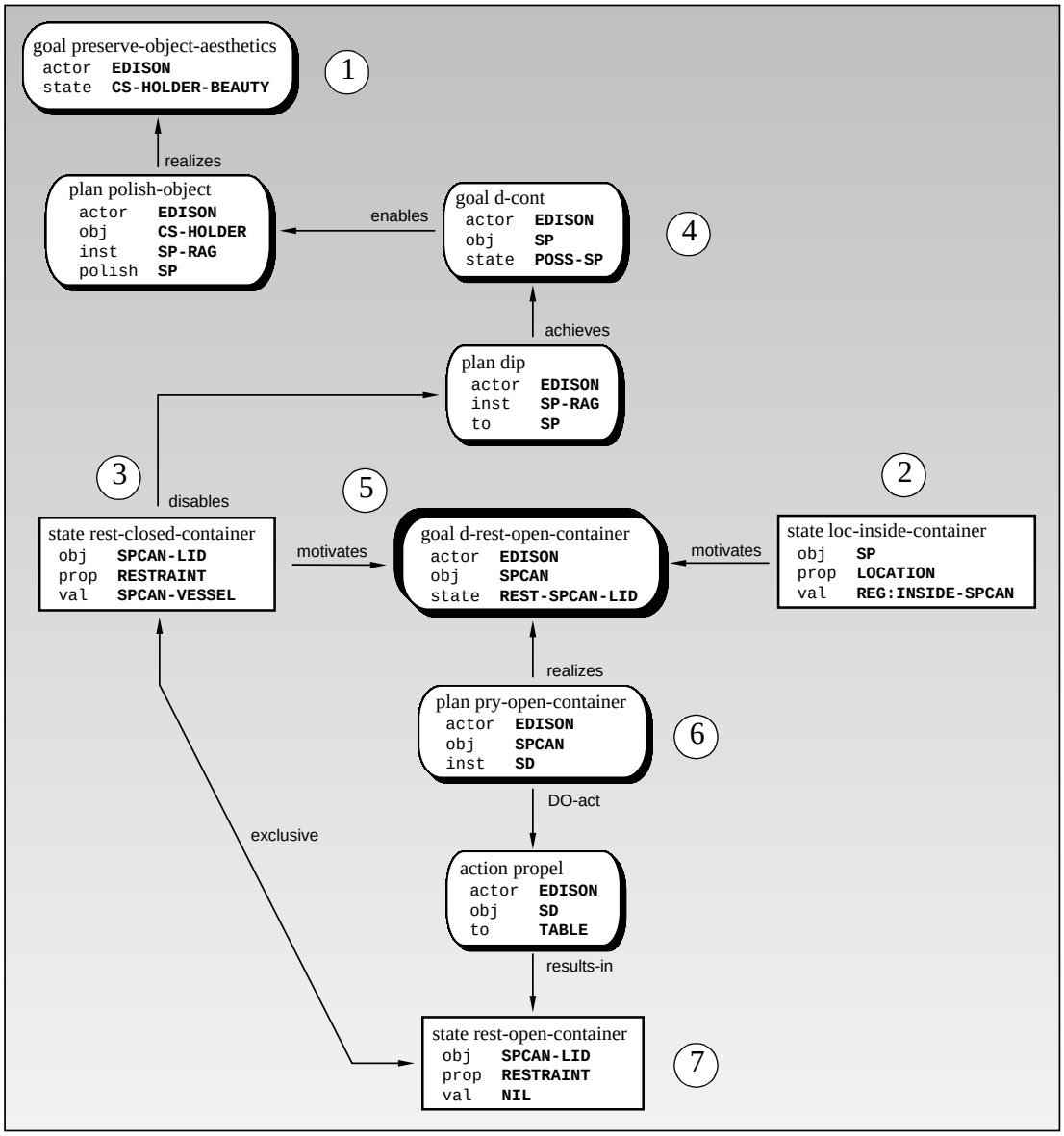


Figure 5.3 The application of behavioral delta goals for resolving goal conflict. The states at items 3 and 7 are named to coincide with the observable state, which is the same state as that mentioned in the d-rest goal at 5.

Fig. 5.3 depicts the goal/plan representation for a situation in which a problem solver wishes to preserve the beauty of a pair of candlesticks (CS-BEAUTY, at 1). One plan associated with such a goal is to polish the candlesticks. The preparation associated with polishing the candlesticks requires the possession of a rag (SP-RAG) and placing some silver polish (SP)

on the rag, as illustrated in Fig. 5.4. These subgoals can be represented with the delta goals,

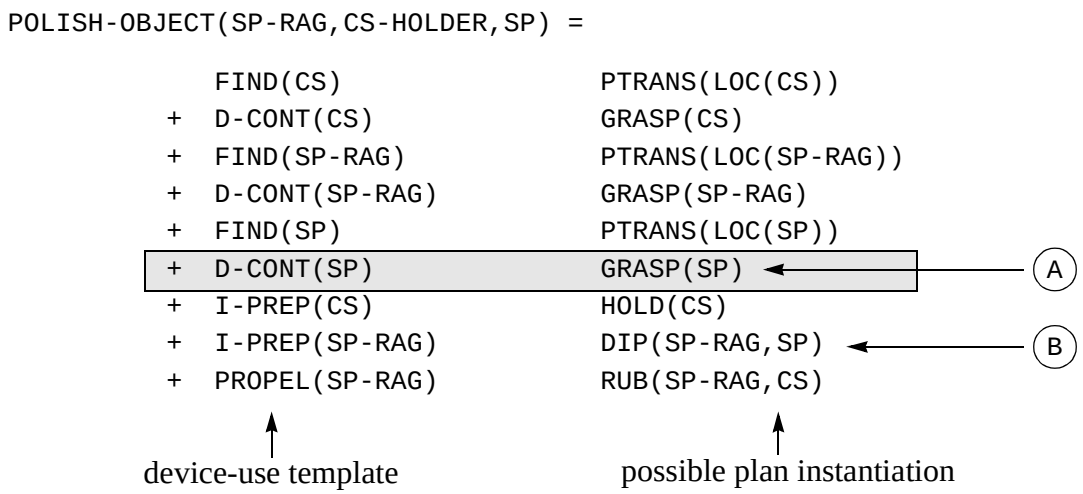


Figure 5.4 Device-use plan for polishing an object.

D-KNOW(LOC(SP)), D-PROX(LOC(SP)), and D-CONT(SP), with the action GRASP(SP-RAG), and with the plan DIP(SP-RAG,SP). However, prior to dipping the rag into the polish, the can which contains the polish must be open. Two states are given in the situation illustrated in Fig. 5.3: that the silver polish is located in the silver polish can (at 2), and that the silver polish can is closed (3). The D-CONT(SP) goal (at 4 in Fig. 5.3 and at A in Fig. 5.4) is thwarted because the closed silver polish can disables the DIP(SP-RAG,SP) plan (B). This motivates a behaviorally related delta goal to open the can by removing the lid (5), D-RESTRAINT(open-container). Lid removal is represented as a plan resulting in restraint removal by enabling a RESTRAIN process. A plan can be used to represent the method for can opening, such as using a screwdriver (SD) to pry the lid open (PRY-OPEN-CONTAINER(SD,SPCAN), at 6), but the motivation to open the can is a D-RESTRAINT goal.

The D-REST goal which is motivated by the inability to directly gain control over a substance in a closed container is associated with many possible plans for opening a container, depending on the container and the substance. Some alternative plans are detailed in Tab. 5.1 The identification of which plan is appropriate for the candlestick polishing sit-

D-RESTRAINT and Container Opening Plans	
Container Description	Open-Container Plan
threaded lid	twist-lid-off
friction-insert lid	pry-lid-off (1)
friction cap	twist-and-pull-cap-off
friction plug	pull-plug-out
crimped cap	pry-cap-off

Table 5.1: Container opening plans associated with D-RESTRAINT(remove-restraint).

D-RESTRAINT and Container Opening Plans	
Container Description	Open-Container Plan
release catch	release-catch + lift lid
lip-and-lid, fluid	punch-hole-in-lid
lip-and-lid, non-fluid	cut-lid-off

Table 5.1: Container opening plans associated with D-RESTRAINT(remove-restraint).

uation (at 1) requires a recognition of the container device and its static representation, specifically by identifying the friction interference between the lid and the can, and the insert orientation between the lid and the can.

5.2.2 Mechanical Instrumental Goals

A class of goals for generating mechanical advantage (I-MECH) is satisfied by devices which have functions whose main conceptualizations involve BPP-TRANSFORM (i.e., that can produce the necessary applied force). I-MECH is one of five *mechanical instrumental* goals in FONM which relate a desired function to the devices which can instantiate it: I-REACH, I-MECH, I-CNTN, I-RESIST, and I-WEDGE. Each instrumental goal type is associated with a particular machine primitive and its object type:

I-REACH: A goal for extending control beyond a component boundary in situations requiring mechanical assistance. The object of I-REACH is a linkage-object, which can instantiate MP-LINKAGE. For example, I-REACH is motivated when professor needs to point to something on a blackboard, in which case a ruler might be useful.

I-MECH: A goal for gaining purchase through mechanical force advantage. The object of I-MECH is a lever-object, which can instantiate MP-LEVER. For example, I-MECH is motivated when a problem solver needs to lift a box which is heavier than he can lift. In this case a crow-bar, or block and tackle, will suffice.

I-CONTN: A goal for gaining mechanical control of an object's range of movement. The object of I-CONTN is a container-object, which can instantiate MP-CONTAINER. For example, I-CONTN is motivated when a problem solver needs to keep tools from rolling down the driveway, or to keep a dog in the yard. In the first instance, the tools might be placed in a hubcap, and in the latter, the dog might be placed on a tether. Each device instantiates MP-CONTAINER.

I-RESIST: A goal for producing passive force that require mechanical assistance. The object of I-RESIST is a spring-object, which can instantiate MP-SPRING. For example, I-RESIST is motivated when a problem solver needs object movement in two directions but can only provide force in one, such as automatic door closing, which is satisfied by objects which instantiate MP-SPRING.

I-WEDGE: A goal for producing changes in object position that require mechanical assistance. The object of I-WEDGE is a plane-object, which can instantiate MP-PLANE. For example, I-WEDGE is motivated when a problem solver wants to change a lightbulb or get a refrigerator onto a moving truck. In the first case, a ladder can be used, while in the second, a ramp can be used, both of which instantiate MP-PLANE.

Five mechanical instrumental goals identify all machine primitives, because they identify the kind of states associated with all device classes. For example, MP-GEAR is used to obtain a specific kind of purchase, one in which position control is required, but it is still used, mechanically, to obtain purchase, so the appropriate instrumental goal is I-MECH.

Mechanical instrumental goals indirectly relate a desired state change to devices whose functions can produce the state change. The candlestick example illustrates the use of mechanical instrumental goals. The delta-restraint goal, which is motivated by a closed silver polish can, cannot be achieved without the use of a mechanical device to produce the necessary force. The PRY-OPEN plan can be used to achieve the D-RESTRAINT goal, but requires a device capable of prying. Consider the device function, PRY, shown in Fig. 5.5, and how it applies to the PRY-OPEN plan.

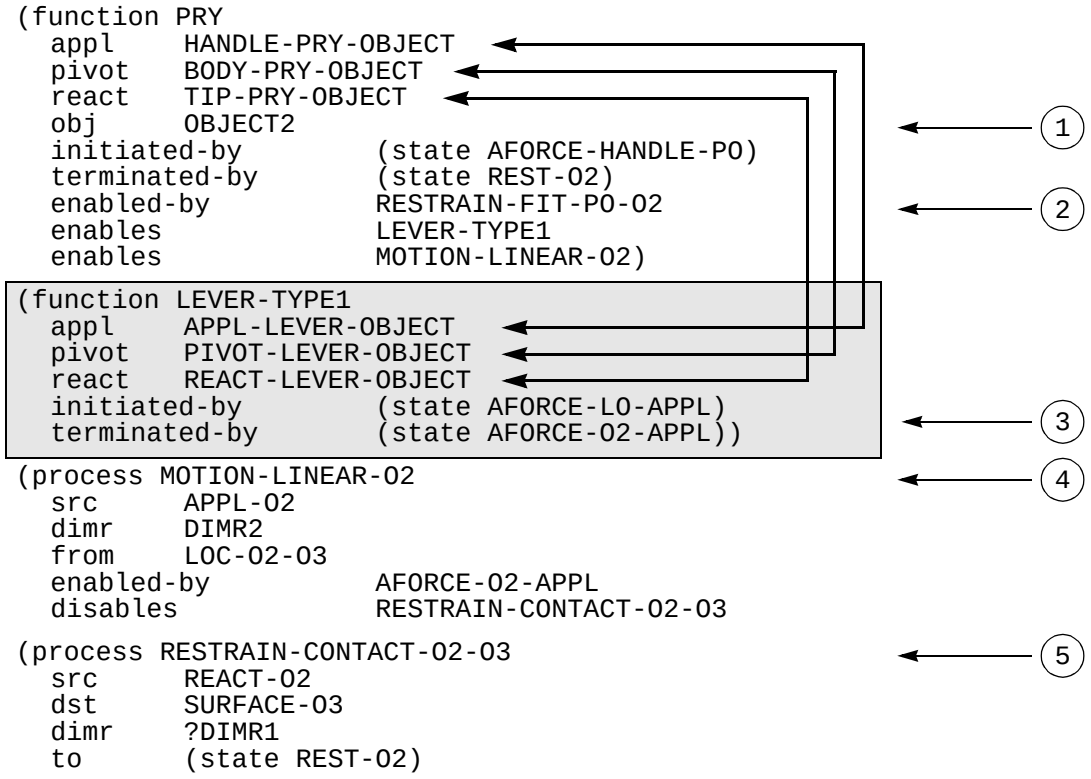


Figure 5.5 Slot-filler instantiation of the PRY function.

Fig. 5.5 illustrates the PRY function and how an object which can be used for prying objects apart enables the function MP-LEVER-TYPE1 (or MP-LEVER-TYPE2, but TYPE1 is represented in the highlighted area of this figure). The function roles instantiated by the pry-object regions also instantiate the roles of the lever-object, as noted by the double-arrowed lines in the figure. The pry-object fits under the object to be pried (RESTRAIN-FIT-PO-O2, at 2). The two contact locations (in this example, different regions/components of OBJECT2) provide a fulcrum and react location for leverage. The direction of applied force determines which MP-LEVER type is instantiated. Motion for the object (OBJECT2, at 1) being pried is enabled by the magnified force produced by the instantiated MP-LEVER.

The interaction between the LEVER machine primitive and the PRY function can be seen in the function bounding states (at 3). The initiating state for PRY is identical to that

of MP-LEVER-TYPE1, but the terminating states differ. The terminating state of PRY is the change in restraint on the object being pried, whereas the terminating state of MP-LEVER-TYPE1 is the magnified force. The representations for the MOTION-LINEAR process, and for one of the contact relationships enabling the RESTRAIN-FIT process, are shown in the figure, at (4) and (5), respectively.

To apply the function PRY in a device-use plan, the roles associated with the function and the function preconditions must be instantiated by the objects in the plan. For example, consider the PRY-OPEN plan illustrated in Fig. 5.6.

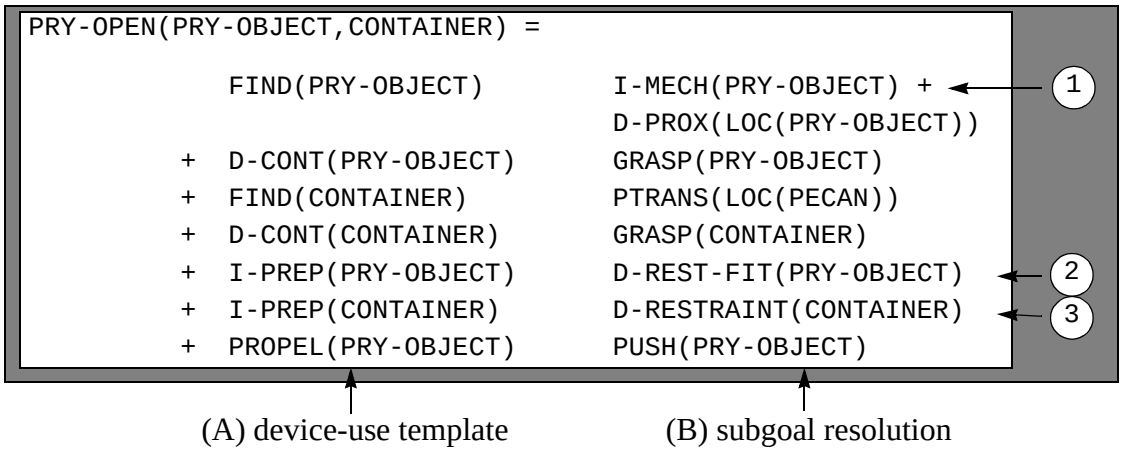


Figure 5.6 Resolution of subgoals in the prying device-use plan.

Fig. 5.6 illustrates a *resolution* (i.e., an elaboration or motivation) of the subgoals in the PRY-OPEN device-use plan template (at A) into mechanical instrumental and behavioral delta goals, actions, scripts, or plans (at B). In Fig. 5.6, finding a pry-object amounts to locating an object which can instantiate MP-LEVER-TYPE1 or MP-LEVER-TYPE2. The I-MECH goal (at 1) is thus an instrumental enablement to opening the silver polish can by prying. The specialization of the FIND subgoal with I-MECH concurs with the Schank and Abelson use of FIND (Eqn. (8-2), page 308), since D-KNOW(loc(pry-object)) implies knowledge that an object can be used as a pry-object. The mechanical instrumental goals specialize this knowledge to the type of device function needed. The delta-restraint goal D-REST-FIT (at 2) represents the restraint state necessary to achieve in order to instantiate the behavioral precondition for PRY (at 2 in Fig. 5.5). The goal D-RESTRAINT (3) represents the requirement that the container be restrained during the PRY-OPEN plan.

The manner in which the behavior associated with the PRY function is instantiated is illustrated in Fig. 5.7. I-MECH is achieved by the state (AFORCE-O2-APPL), which is

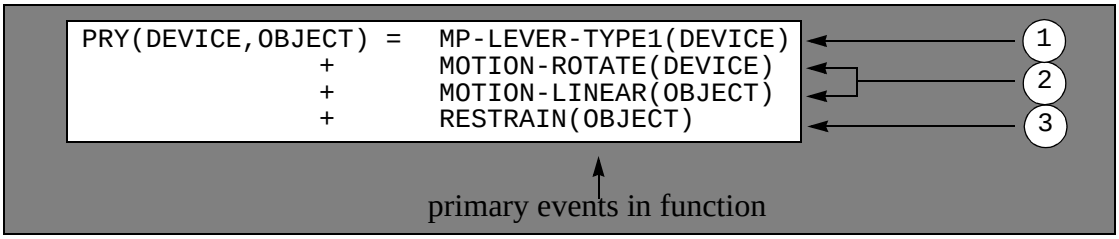


Figure 5.7 Main conceptualizations of the function PRY.

caused by the function PRY through the instantiation of MP-LEVER-TYPE1. Since the device which instantiates this process is being sought, the goal is to instantiate MP-LEVER-TYPE1 (at 1 in Fig. 5.7). The device, and object motion which result from enabling MP-LEVER (at 2), also enable the RESTRAIN (i.e., a disconnection, or removal of restraint, which is still a change in restraint). The RESTRAIN is the main conceptualization of PRY, because it achieves D-RESTRAINT. The device which instantiates MP-LEVER is an instrumental participant in producing this result.

The interaction between the delta goal which motivates PRY-OPEN, the instrumental goal which is motivated to locate a pry-object, and the PRY function which is applied is shown instantiated for opening a silver polish can (SPCAN) in Fig. 5.8.

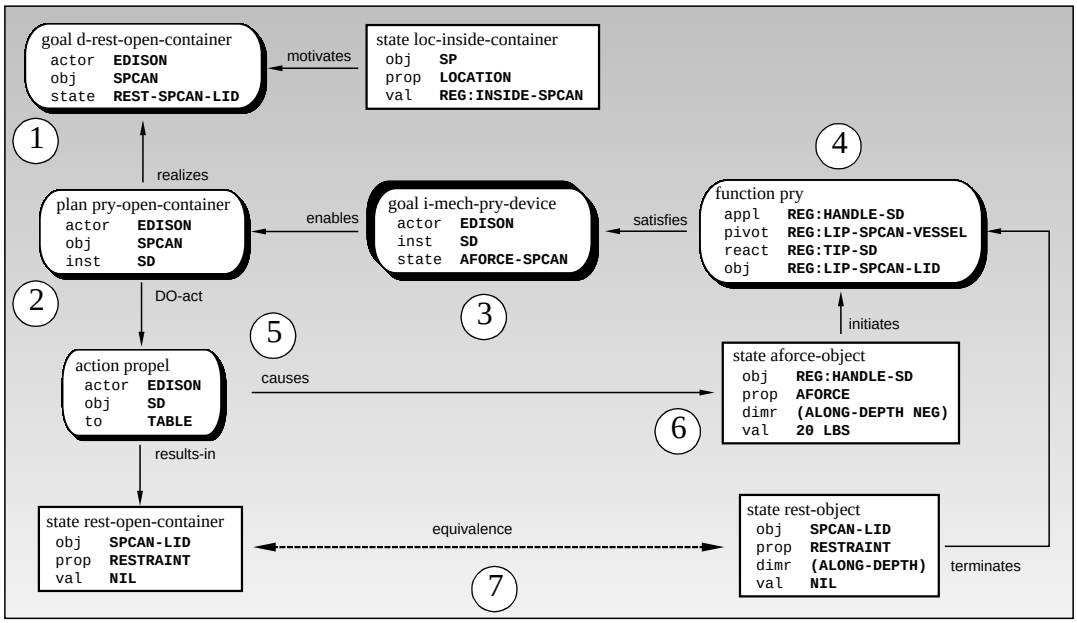


Figure 5.8 A mechanical instrumental goal, I-MECH, and its relationship to device-use plans and device functions.

The delta-restraint goal which was labeled (5) in Fig. 5.3 is labeled (1) in Fig. 5.8. Likewise, the PRY-OPEN-CONTAINER plan which was labeled (6) in Fig. 5.3 is labeled (2) in Fig. 5.8. The instrumental goal which assists in identifying the pry-object is highlighted in the figure (at 3). This goal is associated with devices which can instantiate the PRY function (4) and thus be used in the PRY-OPEN-CONTAINER plan. The object which instantiates the I-MECH goal in this example is a screwdriver (SD). The PROPEL-(PRY-OBJECT) action (5) taken in PRY-OPEN-CONTAINER causes the applied force (6) which initiates the PRY function. The restraint state which terminates the PRY function is identical to the OPEN state caused by the PRY-OPEN-CONTAINER plan (7).

Behavioral delta goals represent the causal interactions between device-use plans and device behavioral states which can be used to resolve plan subgoals. Mechanical instrumental goals represent the causal interactions between device-use plans, behavioral delta goals, and device functions. These goals provide a means to explicitly relate the machine primitives a device might instantiate with the types of goals and plans in which the device might be used.

A taxonomy of plans for resolving delta behavioral goals and instrumental mechanical goals has not been presented. Nevertheless, a class of eight general device-use plan types,

called *machine plans*, have been identified which resolve behavioral delta goals and invoke machine primitives through instrumental mechanical goals. The causal relationships between machine plans, device-related goals, and devices are shown in Tab. 5.2. The primary

MACHINE PLANS			
Plan Type	**BD Goals**	**MI Goals**	**Machine Primitive**
MAKE-CONTACT	D-RESTRAINT	I-REACH	MP-LINKAGE
TRADEOFF-FORCE	D-AFORCE	I-MECH	MP-LEVER
CHANGE-FORCE	D-AFORC	I-MECH	MP-LEVER
CONTROL-RANGE	D-RESTRAINT	I-CONTN	MP-CONTAINER
RESIST-FORCE	D-IFORCE	I-RESIST	MP-SPRING
TRADEOFF-MOTION	D-MOTION	I-WEDGE	MP-PLANE
SEPARATE-PIECES	D-SIZE	I-WEDGE	MP-BLADE
SHAPE-OBJECT	D-SIZE	I-WEDGE	MP-PLANE

Table 5.2 Machine plans and their goal/function relationships.

process which is enabled by the machine plan is identified by the behavioral delta (BD) goal in column 2. The machine plan requires a device which satisfies the mechanical instrumental (MI) goal in column 3. The type of machine primitive which satisfies the MI goal is shown in column 4. As an example, consider the class of CONTROL-RANGE plans, highlighted in Tab. 5.2. These plans are invoked when object motion must be controlled, such as containment, confinement, or connection. The delta goal involves object restraint, and plans which resolve the restraint problem create or remove object restraints. The CONTROL-RANGE plan identifies the primary device through an I-CONTN goal (i.e., a device which instantiates MP-CONTAINER). The plan chosen is specific to the type of container involved. The OPEN plans shown in Tab. 5.1 show a broad diversity of methods for removing restraints based on the type of container. In Tab. 5.1, the OPEN plans are associated with known methods associated with the respective container type. For example, the OPEN plan for a container which has a threaded lid involves twisting the lid.

5.3 Device Pragmatics Representations

A device's intended application can be represented with device use plans and device-related goals. In the following sections, the pragmatic representations of the design intended uses for several devices are presented and discussed. As in previous chapters, four device classes will be discussed:

(1) Simple devices
(2) Simple compound devices
(3) Multiple compound devices
(4) Complex devices

The device representations shown in the following sections are each specialized to the particular device and use, however, each device represented falls into the machine plan classification presented above. The full device-use template is presented for consistency, and the satisfaction of the instrumental mechanical goals for each device are discussed. Of

particular note in these representations are the similarities and differences between device uses, the types of subplans which are involved, and the implications for problem solving which they identify. Device static figures are duplicated for clarity; however, no device dynamics are replicated in this section. For full device representations see the Appendix.

5.3.1 Simple Devices

Simple devices have no moving parts, and all components move as a single object. As a consequence, the use of simple devices is represented and identified by the type of force applied to the object, and the static device representation plays a large role in their application. The pragmatic representations for two simple devices are presented in this section:

(1) Bottle Opener
(2) Screwdriver

Pragmatic representations for four additional devices: (1) carving knife, (2) shovel, (3) spoon, and (4) claw hammer are presented in Appendix A.1.

Bottle Opener

An idealized bottle opener is depicted in Fig. 5.9. The bottle opener has two design-intended uses: (1) can opening, and (2) bottle opening, both of which are TRADEOFF-FORCE type plans in support of the associated container's CONTROL-RANGE (OPEN) plans. In

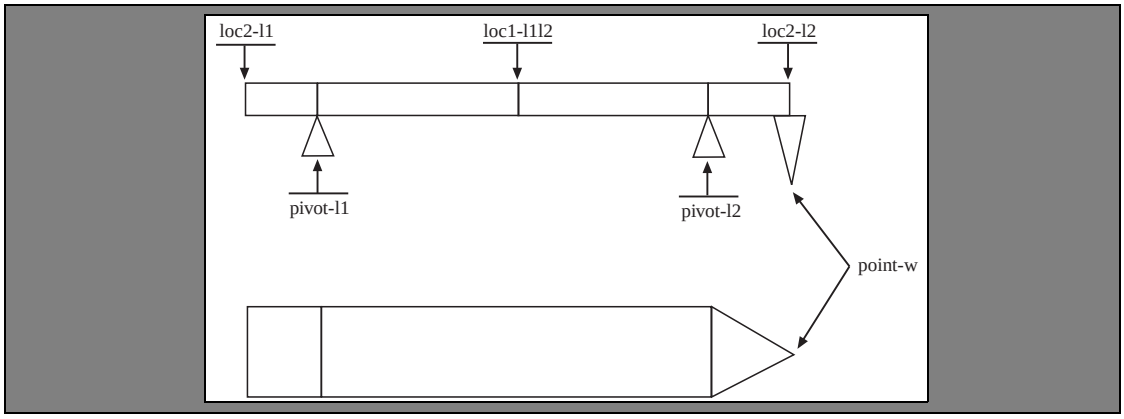


Figure 5.9 Idealized bottle opener.

both cases, the bottle-opener is a lever-object used for prying. As a simple device, using a bottle opener is primarily preparatory. For example, once one end of the bottle-opener is grasped, the intended use can be identified, because the use is specialized to the location on the bottle opener. Fig. 5.10 shows can puncturing as a device-use plan instantiated for a

bottle opener and can.

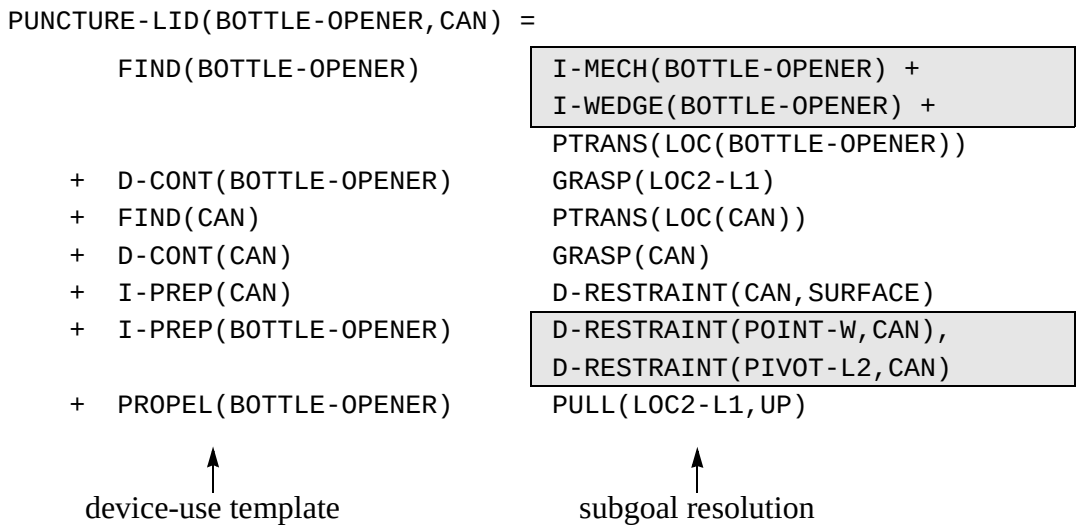


Figure 5.10 Pragmatic representation of bottle-opener can puncturing.

Fig. 5.10 is comprised of two components: (a) a device-use template, and (b) a possible resolution of the plan subgoals. A subgoal resolution is a template which satisfies the plan but isn't fully specified. The right hand side of Fig. 5.10 and the left hand side of the Fig. 5.11 are subgoal resolutions for the bottle-opener can puncturing plan. The plan to puncture a can with a bottle opener is enabled by achieving five subtasks. First, the user must locate and take control of a device which can: (a) produce the force necessary to puncture the can (I-MECH), and (b) cut the can (I-WEDGE). These are shown in the upper shaded portion of the figure. The bottle-opener satisfies the requirements of the can's CONTROL-RANGE plan, because it has a pointed end which will cut the can, and because it can generate the force necessary to do so with leverage. Second, the user must locate and take control of a can. Since no function is required of the can, no instrumental goals are involved. Third, the user must stabilize the can so that it cannot move relative to the bottle opener. Fourth, the user must make contact between the bottle opener and the can in the correct orientation to enable the function. Last, the user must initiate the function by pulling the bottle opener upward away from the can. These relationships are identified by the subgoal resolution in the figure. The FIND subgoal in Fig. 5.10 is resolved as a combination of knowing what functions are needed in the device and getting to where such a device is located. The I-MECH goal, when satisfied, will be associated with a device that can produce the mechanical advantage needed to open the can. Likewise, the same device will be capable of cutting. The pragmatic representation does not specify anything more about the physical characteristics of the device which is used in the plan, and all other references to regions are those associated with the devices found. The PROPEL action which is part of the device-use plan template is shown instantiated with a PROPEL specialization PULL. The subgoal resolution in Fig. 5.11 does not repeat actions which have been specified in Fig.

5.10, so the PROPEL-PULL is left out.

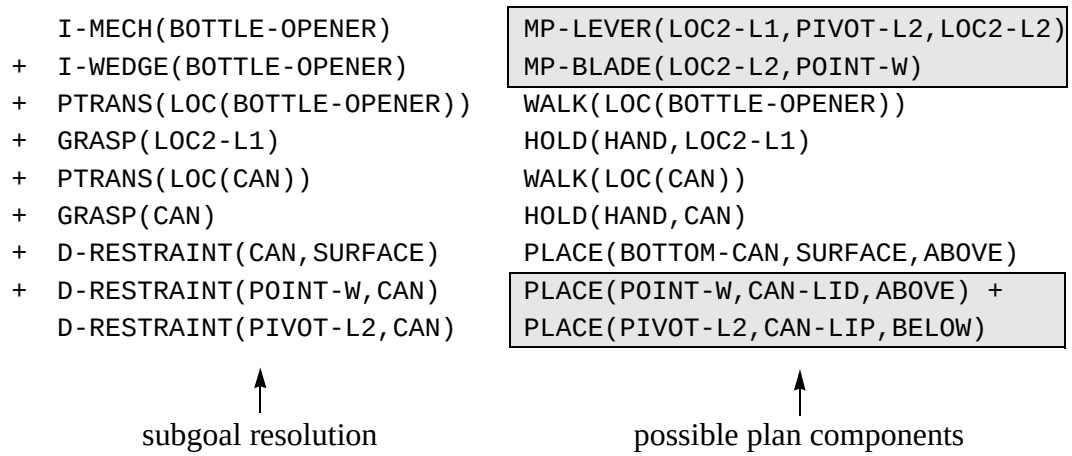


Figure 5.11 Subgoal resolution and possible plan components for bottle-opener can puncturing.

Fig. 5.11 is also comprised of two components: (a) the unspecified subgoal resolution in Fig. 5.10, and (b) a possible plan instantiation of the items in (a). The right hand side of Fig. 5.11 shows a possible instantiation of the bottle-opener puncture-lid plan. In the figure, role instantiations are related to the parts and regions in the idealized bottle opener pictured in Fig. 5.9. The upper shaded portion shows the bottle-opener regions which are instantiated to produce leverage, and to puncture, respectively. The same regions are shown in the two placements in the lower shaded portion of the figure. PLACE is an instrumental script (not presented here) which is used to relocate and orient the region of one object to the region of another object and results in contact between the regions of two objects. The main conceptualization of PLACE is the MOVE action. The first instance of placement, PLACE(BOTTOM-CAN,SURFACE,ABOVE), means that the result of this plan is the can being supported by the surface. The second and third instances of PLACE describe the placement of the bottle-opener on the can. The first of these places the pointed end of the bottle opener on top of the can's lid, and the second places the pivot under the can's lip. Each results in a local state of contact, and, combined, they represent the interference needed to produce leverage. As mentioned previously, the addition sign (+) between the two placements is a notation meaning that these plans are conjunctive. In these examples, conjunctive, but temporally sequential, plans will use the addition sign, to distinguish them from conjunctive, but temporally simultaneous, plans, which will use a comma (,).

Figs. 5.12 and 5.13 illustrate the same object being used to remove caps from bottles.

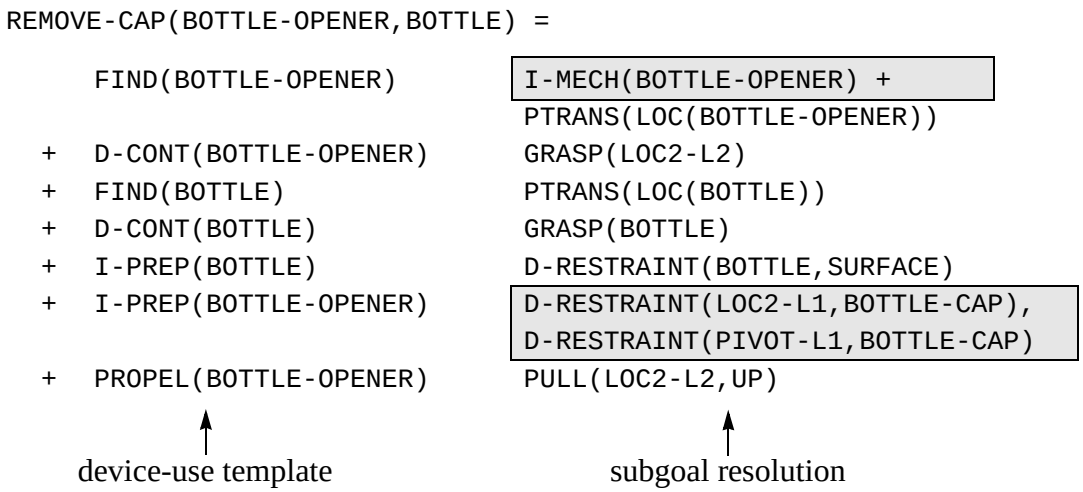


Figure 5.12 Pragmatic representation of bottle opener bottle cap removing.

The same plan template is used, and the subgoal resolution is very similar; however, a different target object is used, different regions of the bottle opener are used, and the resulting force location and direction are different. The subgoal resolution is different for the remove-cap plan because no cutting is performed, so only leverage (I-MECH) is needed to enable the plan. In this case, the pivot location is the end of the bottle-opener so the device instantiates MP-LEVER-TYPE2 instead of MP-LEVER-TYPE1.

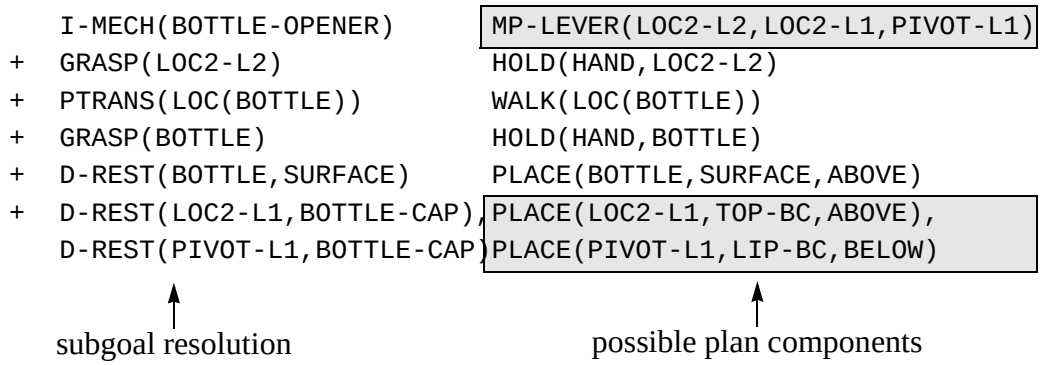


Figure 5.13 Subgoal resolution and possible plan components for bottle-opener cap removal.

In Fig. 5.13 (5), the roles of MP-LEVER are listed in (appl, pivot, react) order, so the bottle-opener regions for this use are (LOC-L2, LOC-L1, PIVOT-L1). Also, for space considerations, D-RESTRAINT is written as D-REST in the figure. As with can-puncturing, the lever regions are placed in their proper locations and orientations with respect to the bottle cap prior to pulling the bottle-opener. Note the different region instantiations of the same I-MECH and D-REST goals in the shaded areas.

Screwdriver

An idealized screwdriver is depicted in Fig. 5.14. The screwdriver has one design-intended

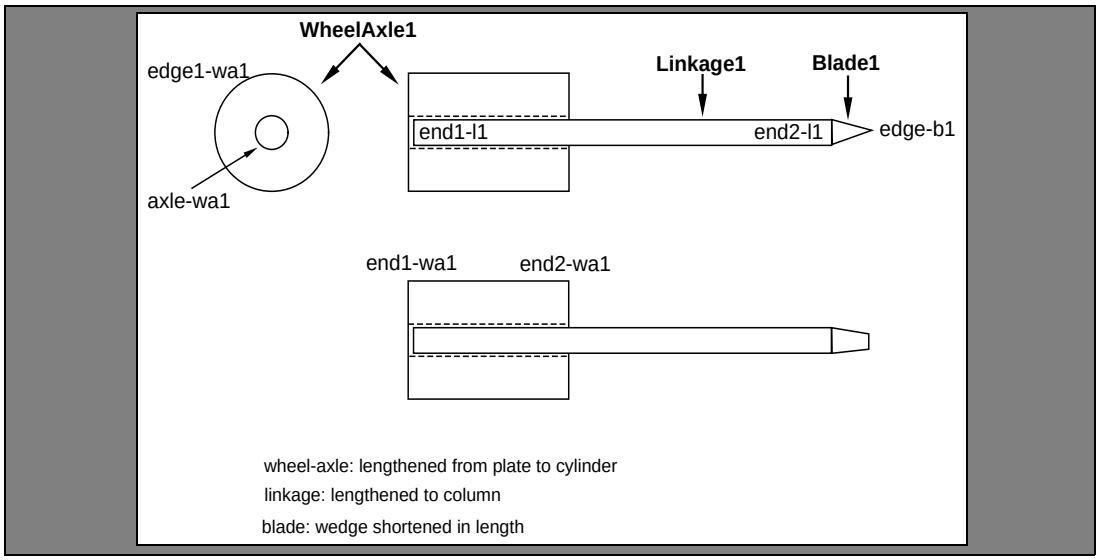


Figure 5.14 Idealized screwdriver.

use: (1) driving, which is shown in Fig. 5.15. Driving is used to create torque (i.e., a TRADEOFF-FORCE plan) in support of connecting objects (i.e., a CONTROL-RANGE plan). The driving plan is applied by (a) placing the screw point in contact with the proper substance, (b) inserting the screwdriver tip into the screw slot, and (c) turning the screwdriver handle. The driving plan makes use of a torque-generating device (MP-WHEEL-AXLE) function. When the handle is grasped and the tip inserted into the screw slot, the screwdriver instantiates the device function. When the screwdriver handle is turned, the function is initiated.

Fig. 5.15 illustrates a PROPEL specialization, TURN, which causes the production of torque on the screwdriver handle. The delta-restraint goal is also slightly different than that in the bottle opener use, since the screwdriver must be kept from turning so that it can transmit torque to the screw. This is accomplished by inserting it into the screw slot. INSERT,

like PLACE, is an instrumental script which results in restraint states. In the case of IN-

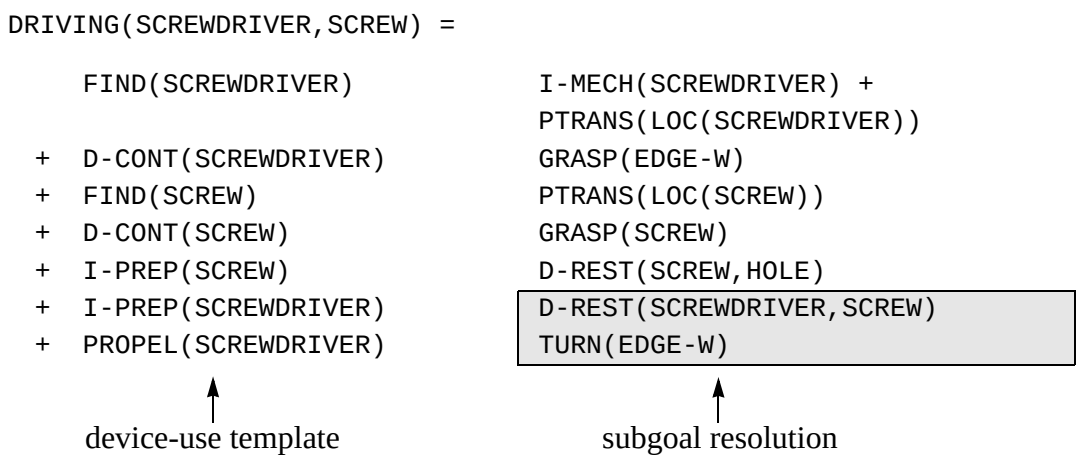


Figure 5.15 Pragmatic representation of screwdriver driving.

SERT, the result is confinement rather than contact.

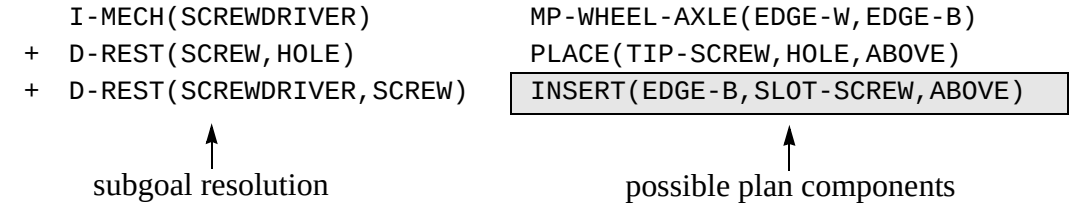


Figure 5.16 Subgoal resolution and possible plan components for screwdriver driving use.

5.3.2 Simple Compound Devices

Simple compound devices have two relatively moving components. As a result, the use of this class of devices requires control of both components, and the device has more than one recognizable state. The functions are thus identified by behavioral state as well as input type and static representation, generally resulting in subplans which apply component functions within the overall device-use plan. The pragmatic representations for two simple compound devices: (1) a nutcracker, and (2) a clothes pin are presented in this section. Three additional devices: (3) scissors, (4) pliers, and (5) door are presented in Appendix A.2.

As devices increase in complexity, their use becomes more specialized, and their application is more strongly associated with their intended function. For example, a pair of scissor is designed to cut a straight line in some material, which is a combination of the functions associated with MP-LEVER and MP-BLADE. The I-MECH and I-WEDGE subgoals combine to identify devices like scissors which have been used to perform such tasks, rather than to search for devices which can resolve each goal independently and collectively.

Nutcracker

An idealized nutcracker is depicted in Fig. 5.17. The nutcracker has a single design-intended use: (1) remove nut meats from nut shells (a CONTROL-RANGE plan), as illustrated in Fig. 5.18. The nutcracker is itself a confinement device, which both restrains the object

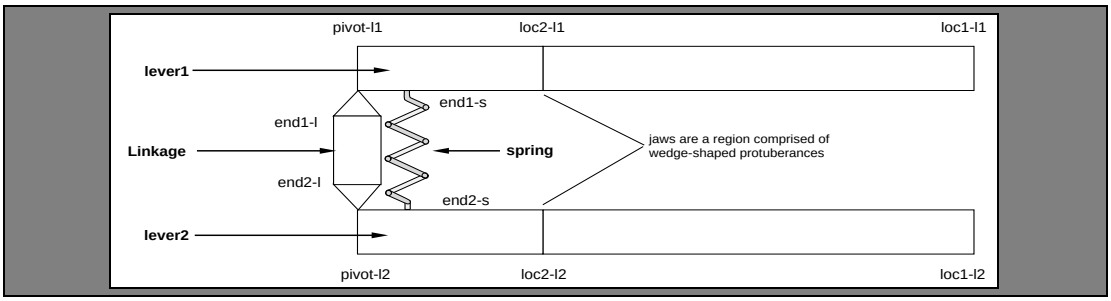


Figure 5.17 Idealized nutcracker.

and produces the force necessary to open it. The spring used in this device automatically prepares the device for use, so the use plan is simpler than that of a pair of scissors or pliers even though they are very similar. The only plan of import is the insertion/placement of the nut between the handles. In other examples of insertion, the object is associated with a containment volume. The nutcracker is associated with a space bounded by two objects and is a confinement space. The device is used to remove nuts by cracking their shells open. The

```
CRACK-OPEN(NUTCRACKER,PECAN) =

    FIND(NUTCRACKER)          I-MECH(NUTCRACKER) +
                              I-CONTN(NUTCRACKER) +
                              I-RESIST(NUTCRACKER) +
                              PTRANS(LOC(NUTCRACKER))
+   D-CONT(NUTCRACKER)        GRASP(NUTCRACKER)
+   FIND(PECAN)               PTRANS(LOC(PECAN))
+   D-CONT(PECAN)             GRASP(PECAN)
+   I-PREP(NUTCRACKER)        GRASP(LOC1-L1),
                              GRASP(LOC1-L2)
+   I-PREP(PECAN)             D-RESTRAINT(PECAN,NUTCRACKER)
+   PROPEL(NUTCRACKER)        PRESS(LOC1-L1),
                              PRESS(LOC1-L2)
```

device-use template — subgoal resolution

Figure 5.18 Pragmatic representation of nutcracker nut cracking.

procedure is initiated by locating an object which can: (a) produce enough force to overcome the strength of the shell which, (b) confine the nut during use, and (c) passively return the device to its starting state so that the device can be operated with one hand. In the case of a real nutcracker, this step degenerates to a PTRANS. The user must then gain control of the nutcracker and nut. Preparation of the nutcracker involves only holding the handles,

but preparation of the nut requires that it be confined by the nutcracker. Finally, the handles are pressed together. The subgoal resolution of the nutcracker device-use plan illustrates (a - c) as a combination of I-MECH, I-CONTN, and I-RESIST goals, each of which must be achieved using the same device. This plan identifies a spring with the device (I-RESIST), though the device would degenerate to a pliers-like device if the spring were to break. This would require manual opening and the device-use plan would then be identical to that for pliers or scissors.

Fig. 5.13 illustrates a possible specification for the items highlighted in Fig. 5.18. The

I-MECH(NUTCRACKER)	LEVER-TYPE2(LOC1-L1,PIVOT-L1L2,LOC2-L1),
	LEVER-TYPE2(LOC1-L2,PIVOT-L1L2,LOC2-L2)
I-CONTN(NUTCRACKER)	CONTAINER(LOC2-L1,LOC2-L2)
I-RESIST(NUTCRACKER)	SPRING(LOC2-L1,LOC2-L2)
D-REST(PECAN,NUTCRACKER)	INSERT(PECAN,LOC2-L1,LOC2-L2)

subgoal resolution — possible plan components

Figure 5.19 Subgoal resolution and possible plan components for nutcracker use.

nutcracker components and regions instantiate two instances of MP-LEVER (type2), MP-CONTAINER, and MP-SPRING. The INSERT script is instantiated by the nut and the nutcracker regions associated with the react MP-LEVER-TYPE2 role.

Clothes Pin

An idealized clothes pin is depicted in Fig. 5.20. The clothes pin has a single design-intended use: (1) holding two objects together, which is a confinement (CONTROL-RANGE) plan. The clothes pin has both opening and closing functions embedded in its application, as shown in Fig. 5.21. As with the nutcracker, a clothes pin's spring eliminates some of the

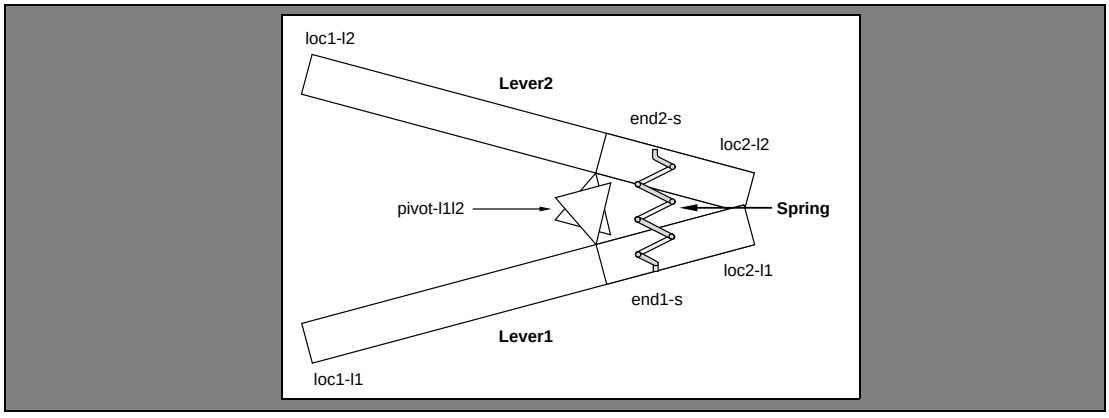


Figure 5.20 Idealized clothes pin.

preparatory work the user has to perform (1). Unlike the nutcracker, which returns the device to its initial (open) state, the clothes pin is found in and returns to its final (closed) state. The clothes pin has identical functional prerequisites to the nutcracker: (a) develop

force, (b) confine an object, and (c) use passive resistance to support one-handed operation. As with the nutcracker, the clothes pin components and regions achieve the I-MECH, I-CONTN, and I-RESIST subgoals; however, the clothes pin instantiates MP-LEVER (type1) instead of type2. Preparation of the clothes is illustrated in Fig. 5.21 with a HANG instrumental script.

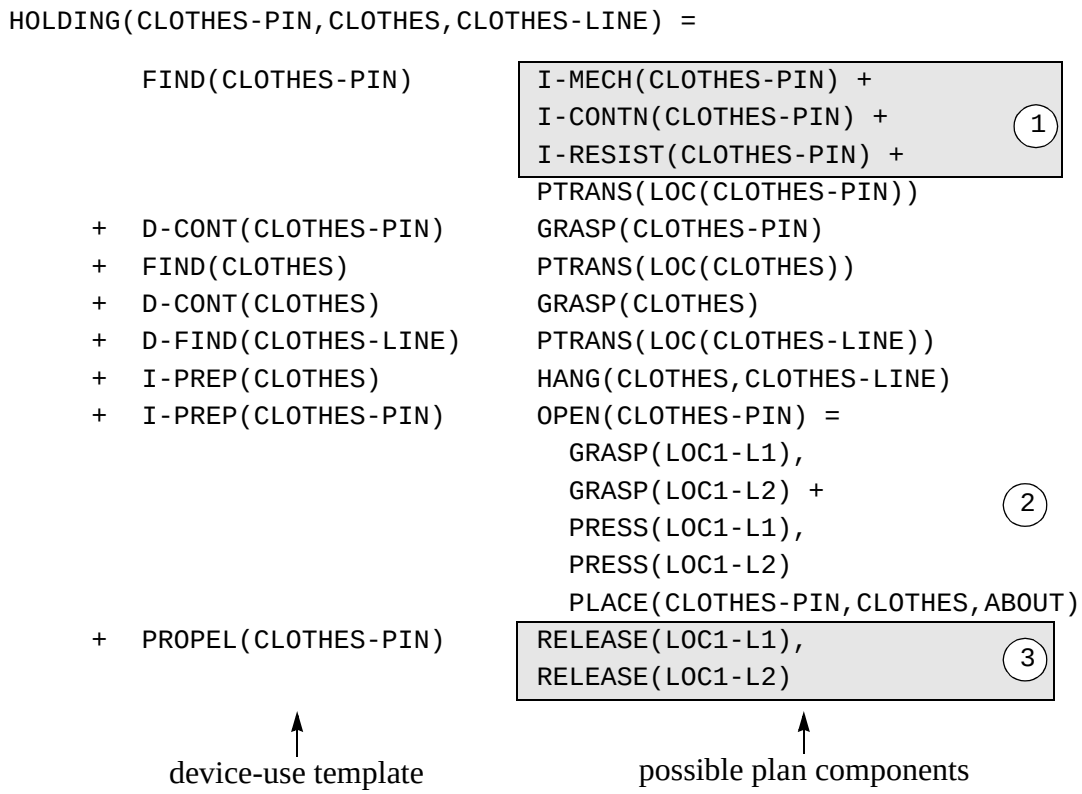


Figure 5.21 Pragmatic representation of clothes pin holding.

As with other such scripts mentioned, HANG is not represented here. Preparation of the clothes pin is represented as a separate OPEN plan (2), and is very similar to pliers (and scissors) devices. The ends are held in the hand and pressed together, and the clothes are placed between the ends. The other difference, which is seen at (3) in the figure, is that the DO action for the clothes pin, because of the spring and geometry, is simply a unGRASP (release) of the handles, instead of a PROPEL.

5.3.3 Multiple Compound Devices

Multiple compound devices may have multiple independently moving components. This increase in complexity has two consequences on the representation of device use for this class of devices. First, as the number of components which move independently increases, the number of design-intended uses tends to decrease toward 1. Second, as the number of components which move independently increases, knowledge of the functions affecting individual components, and how they interact, must be captured by the plan representation. The pragmatic representation for the finger nail clipper is presented in this section, and the representations for three other devices: (1) bottle lid loosener, (2) press, and (3) crank can opener are presented in Appendix A.3.

Finger Nail Clipper

An idealized finger nail clipper is depicted in Fig. 5.22. The finger nail clipper has a single

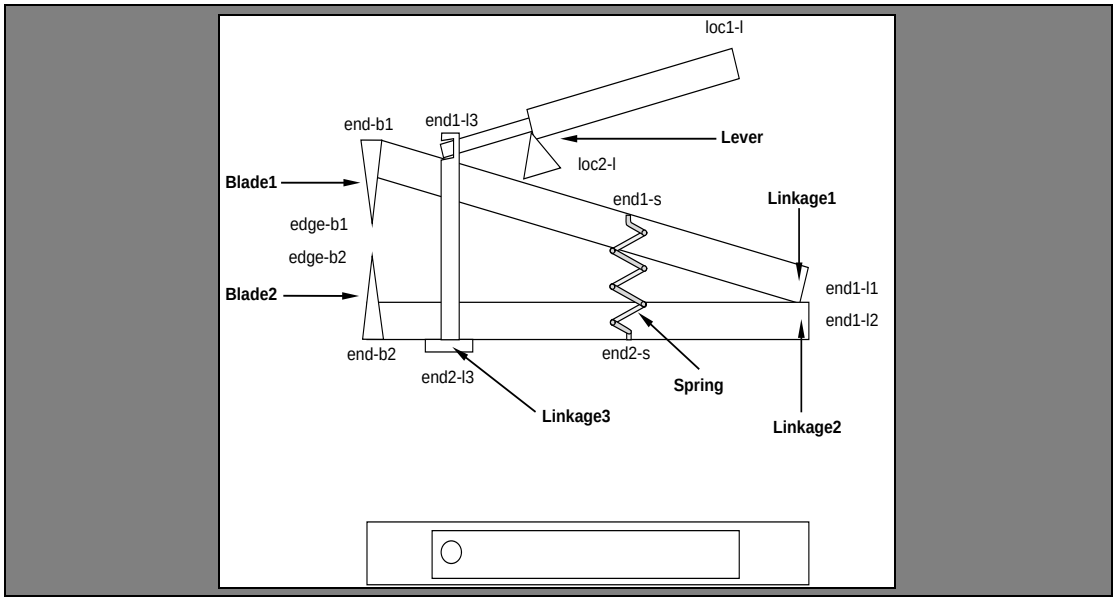


Figure 5.22 Idealized finger nail clipper.

design-intended use: removing finger nails, which is a separation (i.e., SEPARATE-PIECES) plan. Like the nutcracker and clothes pin, the finger nail clipper must constrain the motion of the finger nail prior to clipping it, so the components of a CONTROL-RANGE plan will be embedded in its use as well. The device-use plan for the finger nail clipper is shown

in Fig. 5.23.

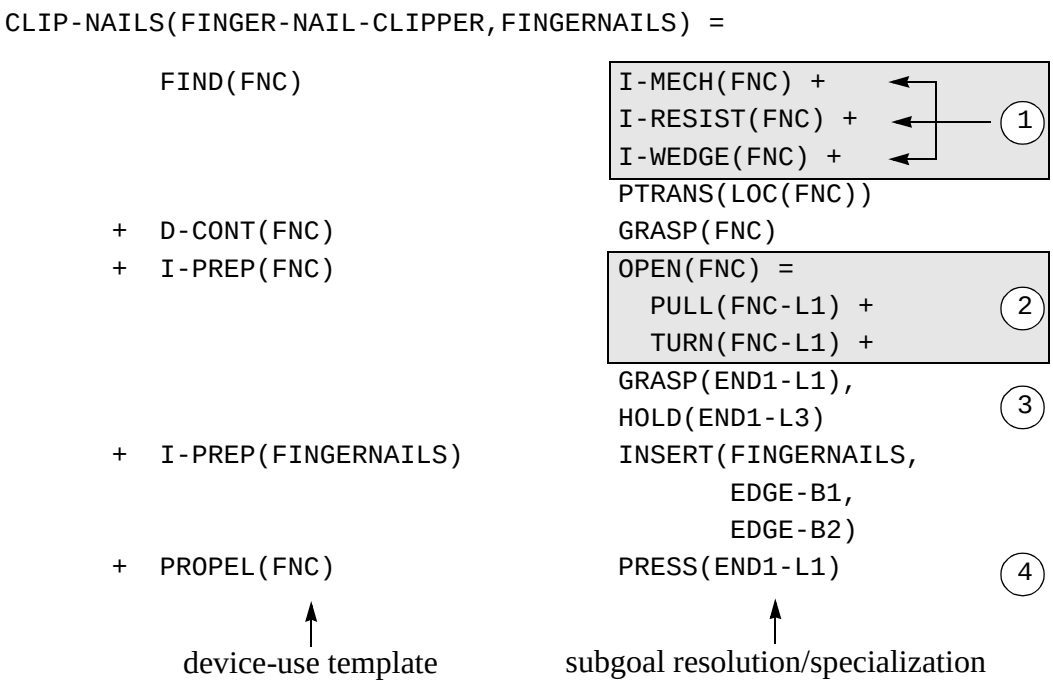


Figure 5.23 Pragmatic representation of finger nail removal.

The device needs three capabilities: (a) to provide enough force to cut, (b) to cut, and (c) to return to its initial state so that it can be operated with one hand. These requirements are represented with the three mechanical instrumental goals I-MECH, I-RESIST, and I-WEDGE, (at 1 in Fig. 5.23). The lever which is used to initiate the cutting function can move independently with respect to the pin linkage (FNC-PIN), and with respect to the linkage which is the upper blade (FNC-L2). The blades (FNC-B1 and FNC-B2) can move independently with respect to each other. The device is prepared in two stages. The first stage is to open the lever (at 2) and then rotate it into position. These actions enable the spring to relax, which enables the blades to move to their initial (OPEN) positions. The preparatory actions of grasping the lever and holding the lower linkage (at 3) enable: (a) the initiating action for the lever, and (b) the restraint of the device which will keep it from moving when the force is applied. By pressing on the lever (4), the device function is enabled and initiated. The finger nail clipper use is similar to the nutcracker use in that the spring returns the device to its initial position after use. This similarity is captured by the representations even though the devices have different functions.

5.3.4 Complex Devices

Complex devices have components which may themselves be multiple compound devices. As a result, the use of this class of devices requires integration of the functions of its components, each of which has more than one recognizable state. The use of one complex device, a toy gun, is discussed in this section. Appendix A.4 presents pragmatic representations for three other complex devices: (1) cork screw, (2) egg beater, and (3) mouse trap.

Toy Gun

A toy gun is a complex device which is used to propel a projectile into contact with a distant object (a MAKE-CONTACT plan). The device has arming and firing mechanisms, a housing, and a projectile, as shown in its idealized form in Fig. 5.24. The toy gun is used to

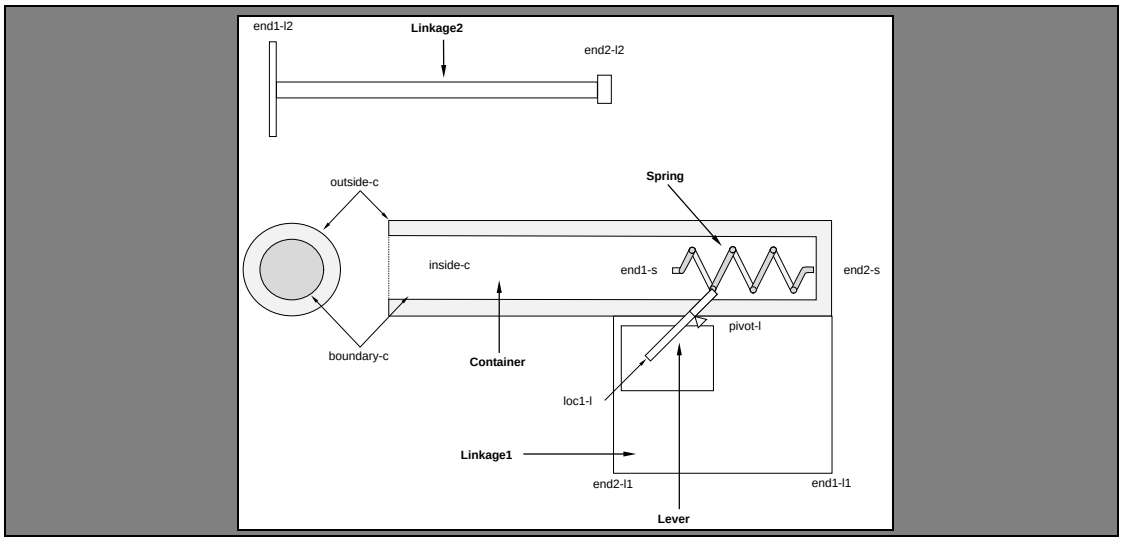


Figure 5.24 Idealized toy gun.

make-contact-at-distance using a shooting plan, as shown in Fig. 5.25. The projectile is the object which makes contact (at 1 in Fig. 5.25), and the rest of the device is designed to produce the necessary force (2), to control the projectile (4), and to control its firing (3). The housing serves to confine the projectile and to determine its path of motion. The arming and firing mechanism utilizes leverage to restrain the stored energy until the chosen time for release. The projectile is reusable and can be disconnected from the device during nor-

mal operation. The device is used by inserting the dart into the barrel. This action enables

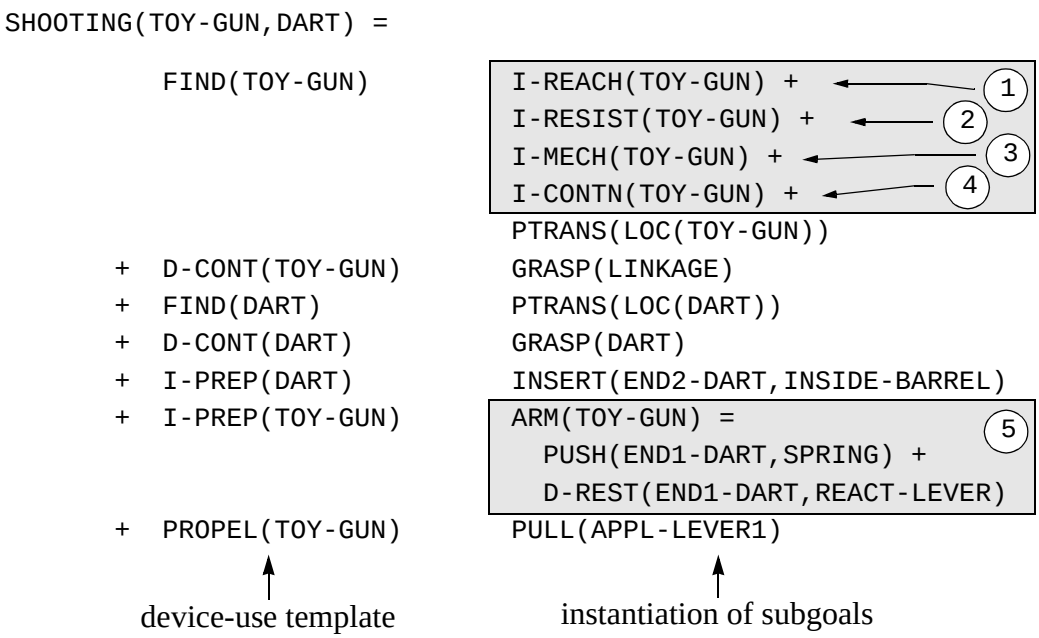


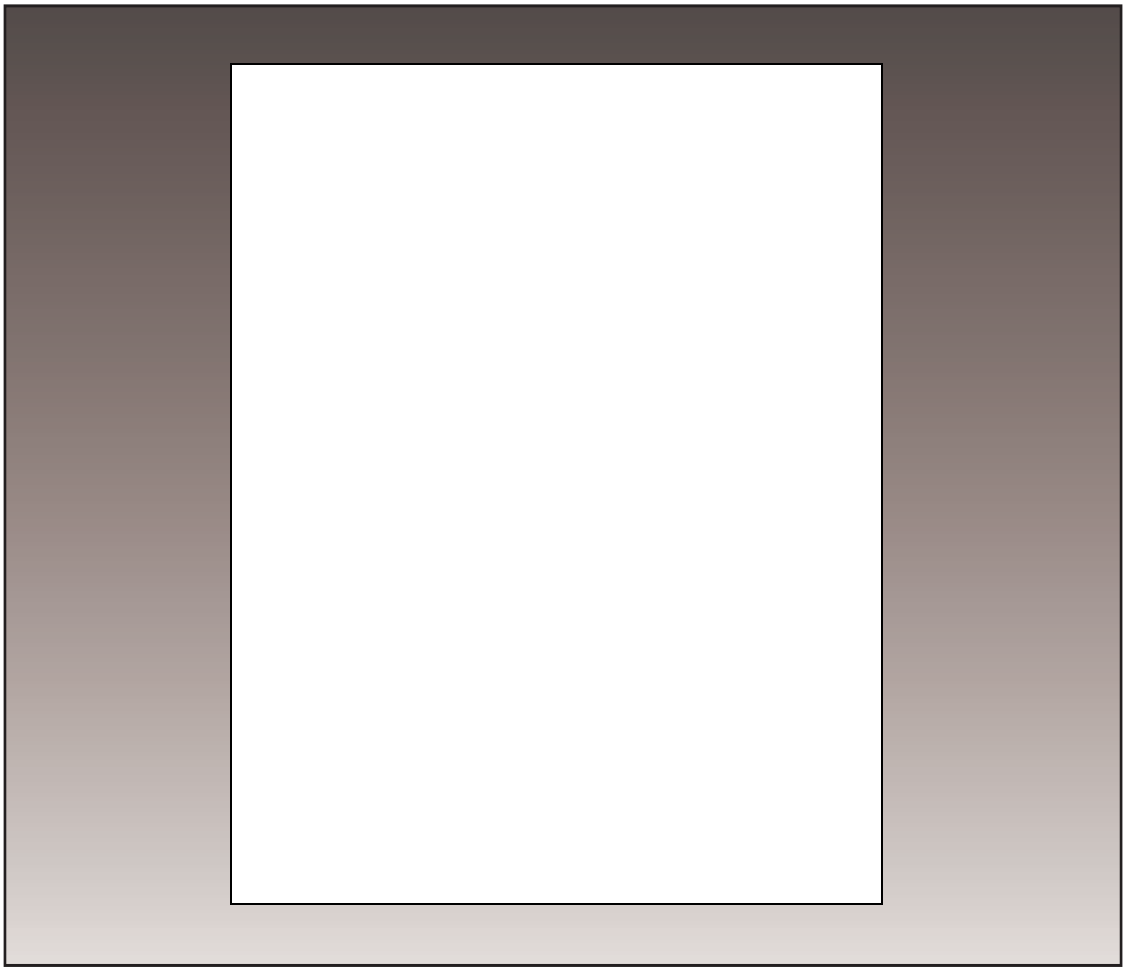
Figure 5.25 Pragmatic representation of toy gun shooting.

the arming by providing a vehicle for applying force that will compress the spring inside of the gun. When the gun is armed, it can be fired by pulling the trigger.

PART II

Reasoning About Mechanical Devices

The overall goal of the FONM representation theory is to support the design and implementation of computational models of mechanical reasoning, particularly improvisation and invention. As a creative problem-solving task, invention can be viewed as an integration of four reasoning tasks: (1) problem recognition, (2) problem interpretation, (3) problem evaluation, and (4) experimentation. This part of the dissertation presents two computational models constructed while developing FONM. Each model makes use of specific aspects of the FONM representation of devices, and demonstrates recognition of knowledge and inference-making capability in its reasoning domain.



Chapter 6
Mechanical Experimentation

This chapter presents a program which performs mechanical simulation. EDEXP (EDison EXPerimenter) is a computer program that takes the symbolic representation of a simple mechanical door, mutates the door by moving a hinge, simulates the capacity of the door to move, by moving the hinge to different locations, and identifies the relationship between hinge behavior and door rotation. To accomplish this task, EDEXP implements the FONM theory of MOTION and RESTRAIN behavioral primitives.

In this chapter, the reasoning processes associated with hypothesizing, executing, and interpreting the results of mechanical experiments are discussed with respect to their dependency on how knowledge of mechanical behavior is represented and manipulated. The EDEXP model is presented and applied to a door manipulation example.

6.1 Computational Architecture

The two models presented in this and the next chapter utilize the computational architecture from the EDISON project, but have been implemented using different versions of FONM and different computational approaches. As such, those components of the architecture which are specific to each program are presented with respect to the kind of I/O and behavior that the program produces.

6.1.1 EDEXP

EDEXP implements four components of the EDISON computational architecture (Fig. 6.1): (1) a planning demon interpreter, (2) rules for performing planning analysis, mutation, and recognition (3) a planning memory, and (4) a long term memory of abstract patterns

associated with devices, device plans, and device behaviors.

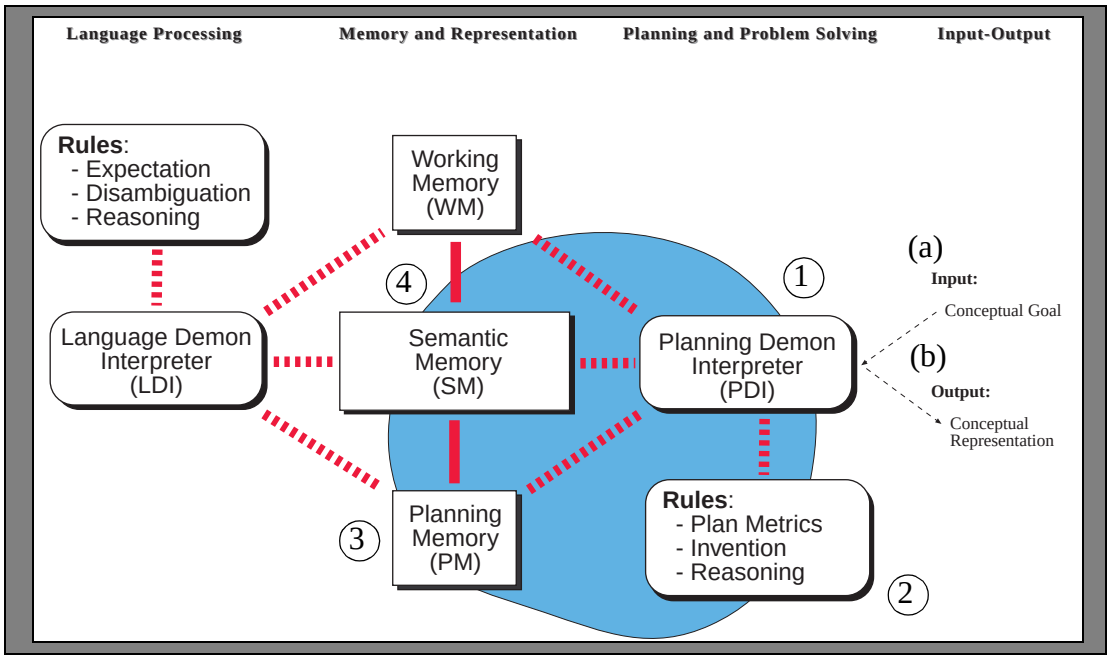


Figure 6.1 The components of EDEXP. Thick solid lines represent inter-memory access. Thick dotted lines represent control access to memories and rules from demon interpreters. Thin dotted lines represent input/output data flow.

- **Semantic Memory (SM)**. Semantic memory represents schematic knowledge of objects, events, and their causal interactions. In EDEXP, SM is comprised of frame-based rules which describe object behavior according to the FONM behavioral process definitions. During planning and behavioral simulation, input conceptualizations are compared to their counterparts in semantic memory, which are used to instantiate incomplete frames.

- **Planning Demon Interpreter (PDI)**. The planning interpreter uses demons to process goals and plans by recursively matching their subgoals to planning memory and comparing the subgoals to the overall task. The interpreter functions in a two modes: (1) experimentation, and (2) hypothesis testing. During experimentation mode, the plans associated with the experiment and device are instantiated with the intent of generalizing a hypothesis which relates the goal and device. During hypothesis testing mode, the planner applies the hypothesis to the chosen plan and device templates to attempt to prove the hypothesis.

- **Planning Memory (PM)**. During planning, EDEXP uses instantiated conceptual frames initially placed onto a memory called *planning memory*. Planning memory serves as a memory retrieval source for reasoning since the items accessed from memory represent the episodic input. Planning memory is maintained until a plan is analyzed.

- **Planning Metrics and Mutation Rules**. During planning, rules are needed to make choices and follow up on them. Three types of rules (implemented as demons) are used, for: (1) general planning, (2) hypothesis generation and testing, and (3) behavior recognition. The processes of planning and hypothesis generation and testing are implemented as demons. Demons associated with behavior recognition are associated with conceptual frames for devices in SM using procedural attachment. The PDI loads potential concepts into planning memory, and the demons associated with each concept are placed onto an associated agenda. The PDI cycles through the demons on the agenda and polls them with respect to the concepts resident in planning memory.

6.1.2 Communicating with EDEXP

EDEXP functions in a single mode. A symbolic object description and a symbolic goal (at 'a' in Fig. 6.1) are input to the system, and a conceptual representation of the analysis results is returned. The planning demon interpreter (PDI, at 1 in Fig. 6.1) matches the conceptual input (device and task) in planning memory (PM, at 2) to plans and devices in semantic memory (SM, at 3). The mutation metrics are rules (at 4) which modify the planning behavior. Symbolic output is returned at 'b'.

6.2 Object Representation in Task Driven Experimentation

Mechanical problem solving involves the retrieval and adaptation of plans and objects to suit a particular scenario. Adaptation may require using plans for connecting or separating objects, or modifying them in various ways. Experimentation is a fundamental method used to generate and test hypotheses, and thereby obtain new information that can be applied during problem solving. Experimentation requires the ability to (a) modify and simulate objects and plans, (b) recognize new behavior, and (c) form hypotheses regarding that behavior. Each of these requirements is a function of knowledge associated with an object, its behavior, and plans which invoke its function. EDEXP models reasoning tasks which use FONM representations for device statics and low-level device dynamics.

When an object is manipulated, it undergoes state changes which affect its behavior. The representation of objects and object behavior in EDEXP must be sufficient to support evaluations of the device with respect to a goal and plan. For example, consider the simple door illustrated in Fig. 6.2. To perform simple experiments involving the manipulation of this door's characteristics, the EDEXP representation must support the recognition of functional (i.e., can rotate) and disfunctional doors. The door at (a) is capable of rotation, mean-

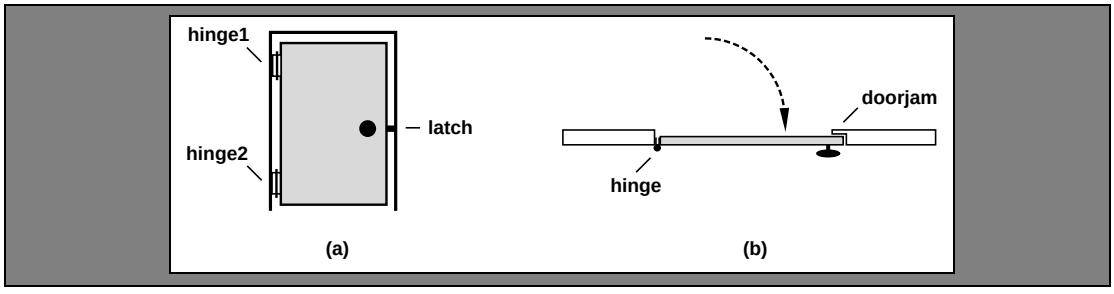


Figure 6.2 A simple functional (a) door and (b) application.

ing that it is *mobile*. The door at (b) is both mobile and perturbed. The door depicted in Fig. 6.3a represents the mutation of a single characteristic of the door depicted in Fig. 6.2a, where the upper hinge is relocated. In this case, the door is clearly incapable of rotation (i.e.,

immobile). The door in Fig. 6.3b illustrates a simple modification of the perturbation de-

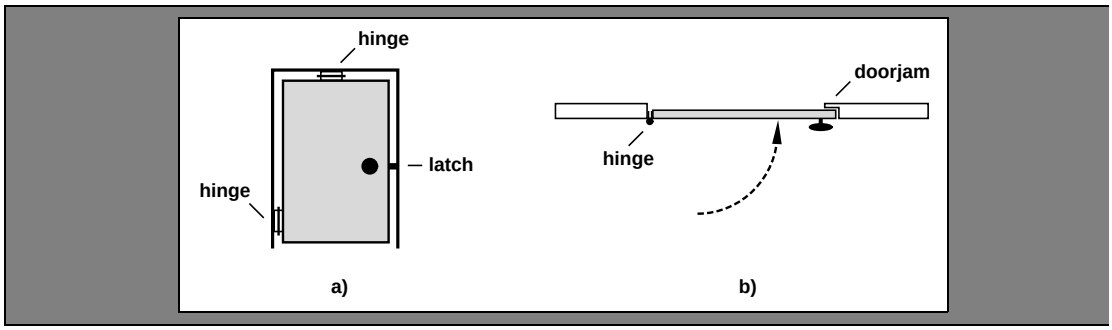


Figure 6.3 Disfunctional door (a) design and (b) application.

picted in Fig. 6.2b, resulting in a door which cannot rotate. The EDEXP door representation must describe the components of the door and their structural relationships. It must also describe the states of the door which change when a hinge is moved, or a force direction is changed, so that mobility and motion enablement and disablement can be recognized.

6.2.1 Object Statics in EDEXP

In EDEXP, objects are represented as a frame structure with six characteristics: (1) function, (2) components, (3) placement, (4) restraint, (5) mobility, and (6) connectivity. For the simple door depicted in Figs. 6.2, and 6.3, these characteristics are all related through object placement. The components, their connectivity, their mobility, and their placement are associated with portions of the FONM static representation. The processes in which the object is participating and its function are associated with portions of the FONM dynamic representation. Consider the door representation shown in Fig. 6.4, which is associated with the functional door depicted in Fig. 6.2. In this figure, the door has a single filled role (SFUNC-

```
(DOOR &DOOR.1
   SFUNCTION  ((&M-ROTATE.1 ABOUT-L (SIDE1 FAR_LEFT)))   <-- 1
   &PART <==> (&DOORWAY.1 &DOORSLAB.1 &HINGE.2 &HINGE.1))   <-- 2
```

Figure 6.4 Representation of the door device in EDEXP. The <==> symbol denotes a link between knowledge structures. The ampersand (&) is a reference mechanism for knowledge structures.

TION), which represents a *structural* function (as opposed to an intended function, such as to control access to a space). The role is filled (at 1) with the list (M-ROTATE.1 ABOUT-L (SIDE1 FAR_LEFT)), which represents a rotational motion, about the door's longitudinal axis, with respect to its far left side. The &PART role (at 2) identifies the components of the door (DOORWAY.1 DOORSLAB.1 HINGE.2 HINGE.1), each of which is represented with separate structures. The next four sections will describe the remaining components of the door static and dynamic representation applied to DOORSLAB.1.

6.2.2 Object Placement and Location

Because of the causal dependency between object connectivity and motion, the only infor-

mation needed to determine door behavior is hinge placement. In FONM, an object's placement is represented by locating its center of gravity with respect to another object. In EDEXP, this is accomplished by referencing region locations of a known object, such as the doorway, or the left side of the door, and using line intersection to locate the object. Consider the door re-illustrated in Fig. 6.5, which describes the representation for hinge1 placement. A region implicitly describes a cartesian axis, so reference to a region and value

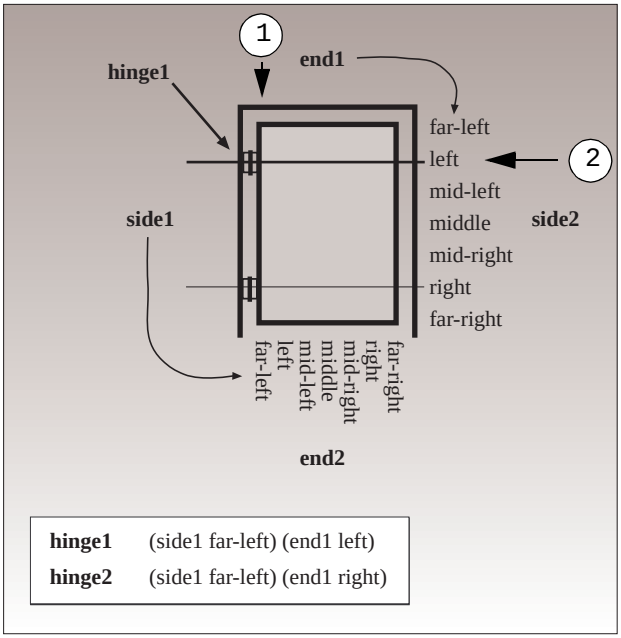


Figure 6.5 Object regions and location values in EDEXP.

locates an object axis. Fig. 6.5 depicts four regions for the doorslab: SIDE1, SIDE2, END1, and END2. In EDEXP, SIDE1 and SIDE2 refer to the doorslab width, so reference to SIDE1 in a location places an object somewhere on the ALONG-WIDTH axis of the doorslab. Similarly, any reference to END1 places an object on the ALONG-LENGTH axis of the doorslab, because END1 and END2 refer to the ALONG-LENGTH doorslab axis. In EDEXP, all sizes are referenced using a discrete scale with seven values: FAR-LEFT, LEFT, MID-LEFT, MIDDLE, MID-RIGHT, RIGHT, and FAR-RIGHT. Once a region is specified, a value from this list defines the location on the axis where the object is located. In Fig. 6.5, the value chosen to locate hinge1 on the ALONG-WIDTH dimension (w.r.t. SIDE1) is FAR-LEFT, so hinge1 is located somewhere on the left edge of the doorslab (shown at 1). By choosing a second doorslab region, which identifies another doorslab axis, a second line is identified and an intersection of these lines defines the location of hinge1. In Fig. 6.5, the second region is END1, which references the length of the doorslab, and the second value is LEFT, which specifies that hinge1 is located near the upper left edge of the doorslab (at 2). The representations for both hinge locations are shown in the lower left portion of the figure. Each location is comprised of two (region value) lists.

6.2.3 Object Restraint and Mobility

In the door representation, only one FONM behavioral state, that for restraint, is needed to represent the door's rotational capacity. As presented on Page 25, object restraint in FONM describes locations and dimensions in which the object cannot move as a direct result of

connection. Object *freedom* describes dimensions in which the object can move. Both have meaning locally, where two objects come into contact, and globally, with respect to how the object as a whole moves in space. In EDEXP, individual object restraints resulting from specific connections are represented with a list called RESTV, and their global combination is represented with a list called GRESTV. RESTV is used to identify which of the object's dimensions will support mobility. This information is also used to determine which components of a device can move and how, since connections between objects produce restraint states on the objects, and those restraint states can be analyzed individually and together to identify motion capabilities of each object. Consider the representation of the doorslab and hinge shown in Fig. 6.6.

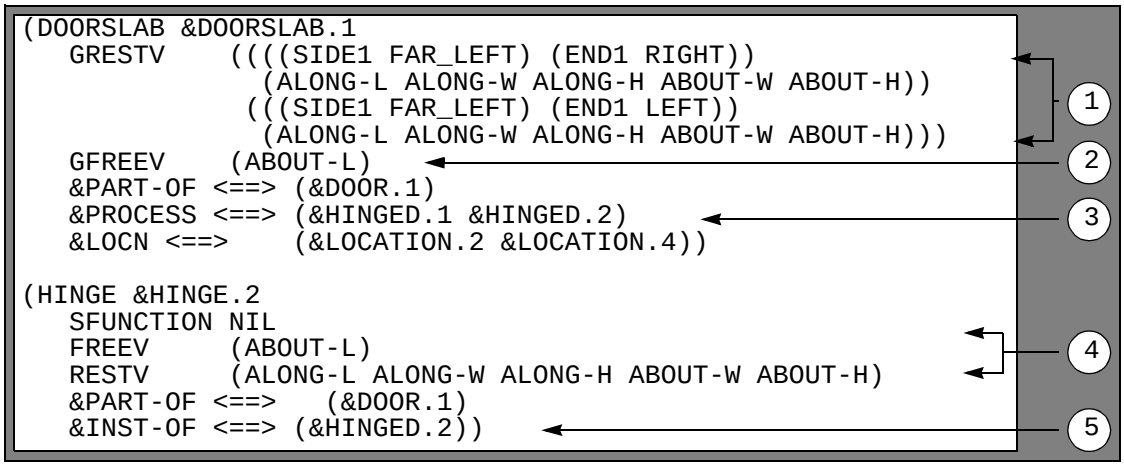


Figure 6.6 Restraint and mobility representation in EDEXP. In RHAPSODY, the double-headed arrow (<=>) identifies a semantic link between knowledge structures.

Fig. 6.6 illustrates representations for the door's doorslab and one of its hinges. The doorslab is the portion of the door that moves. Its GRESTV (at 1) is based on the rigid connections between the doorslab and its hinges, and describes two restraint states on DOORSLAB.1. The first is identified by the first sublist as a location, ((END1 FAR-RIGHT) (SIDE1 RIGHT)), and a list of dimensions which are restrained at this location (ALONG-L ALONG-W ALONG-H ABOUT-W ABOUT-H). This restraint means that DOORSLAB.1 cannot translate in any dimension at this location, and that it cannot rotate about is width or depth dimensions. The second restraint in GRESTV describes a second connection on the left side of DOORSLAB.1 which has the same dimensional restraints. The restraint on HINGE.1 is also shown in the figure as RESTV (4). The RESTV slot is used when there is only one connection. Although this is not exactly true, the doorway connection is ignored in this representation due to the rigid connection.

An object also has a slot called FREEV or GFREEV which represents mobility. An object with a single connection is represented with RESTV, and its mobility at that connection (FREEV) is the complement of RESTV. HINGE.1 is represented this way (at 4). When an object, such as DOORSLAB.1 has multiple connections whose resulting restraint states are represented with GRESTV, the global mobility is represented as GFREEV (at 2), which is the intersection of local mobilities.

6.2.4 Object Connectivity

Objects in EDEXP are related through their relative placement and their connectivity. Fig. 6.6 shows two connections, HINGED.1 and HINGED.2 (at 3 and 5), in which

DOORSLAB.1 and HINGE.1 participate. The representation for the HINGED.1 connection is shown in Fig. 6.7. HINGED.1 is a defined by the locations on participating objects,

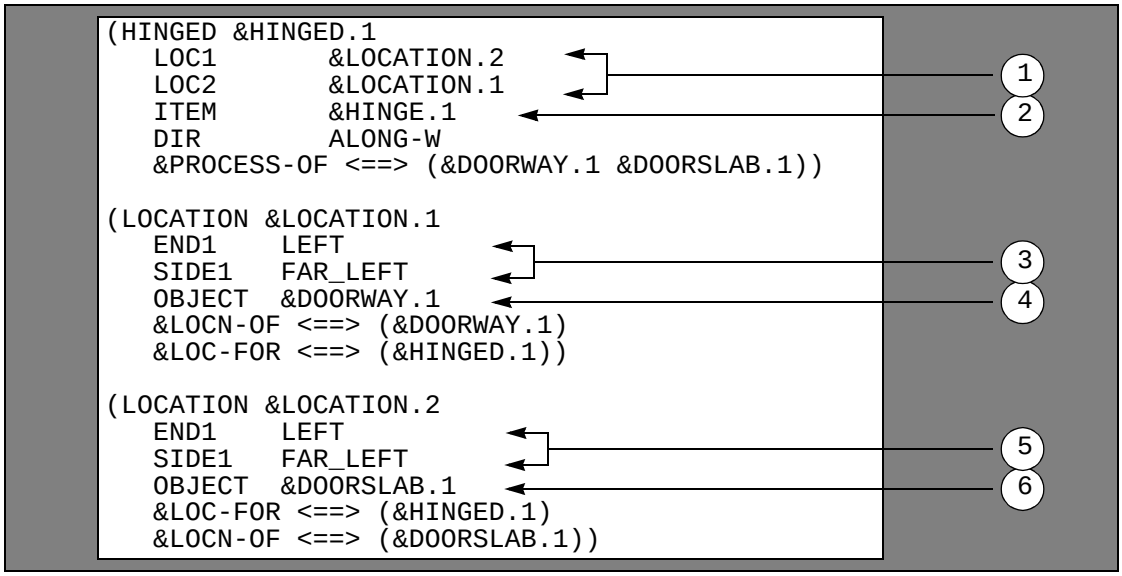


Figure 6.7 Object connectivity in EDEXP.

by the object which is identified with the process, and by the dimension/direction directly affected. HINGED.1 describes a FONM RESTRAIN process between HINGE.1 (at 2), DOORWAY.1 (at 4), and DOORSLAB.1 (at 6). The process is located at LOCATION.1 for DOORWAY.1 and LOCATION.2 for DOORSLAB.1. Each location is described as before, with two region/value pairs (at 3 and 5). As a RESTRAIN process instance, HINGED.1 results in the restraint (((SIDE1 FAR_LEFT) (END1 LEFT)) (ALONG-W)) in the doorslab GRESTV. The other dimensions in the restraint state are identified during program execution, when the combined effects of both hinge connections are put together into GRESTV. It should be noted that the same restraint state exists for the doorway, since the hinge is also connected to the doorway; however, the doorway is grounded (i.e., fixed) which subsumes the restraints produced by any local connectivity, so effects on the doorway are ignored.

6.3 Planning and Experimentation in EDEXP

The EDEXP planner uses rules to construct and connect knowledge structures. Demons are used to recognize motion capability in devices stored on planning memory. The planning

function of the program is illustrated in Fig. 6.8.

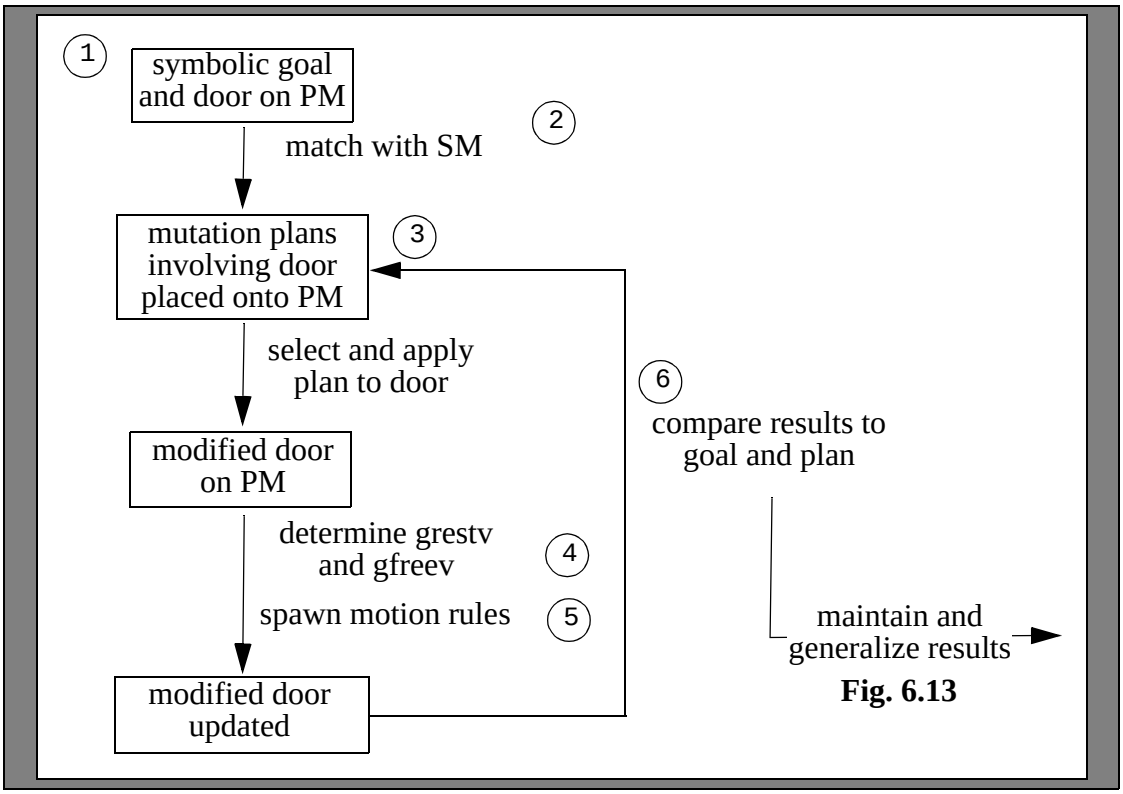


Figure 6.8 Flow chart for planning mode in EDEXP.

When the program is started, it is given a simple representation for a functional door (i.e., one the rotates) and a task to experiment with door function using mutation (at 1 in figure). These items are matched to mutation plans in semantic memory (2), producing a list of possible mutation plans on the door (3). One of the plans is selected for evaluation, and the resulting changes are effected on the door. Rules are then fired to determine the restraints and mobility of the new door as a result of changed connections (4), and demons associated with the door are generated to determine if the new door is capable of motion and, if so, to determine what type of motion is possible (5). The resulting information is used to evaluate the plan with respect to the original task (6), to select a continuing strategy, and is also used in the hypothesis generation phase of the program (shown in Fig. 6.13). The major components of planning phase: (1) plan choice, (2) plan application, and (3) plan evaluation will be described below.

6.3.1 Choice of Plans

In EDEXP, the goal is to experiment with door function (which is represented as rotational

motion) using mutation. Consider the goal given to EDEXP illustrated in Fig. 6.9. There are

```
(EXPERIMENT #{MOTION} . #{HOUSE-DOOR})
```

Figure 6.9 Goal representation in EDEXP. The #{symbol} notation refers to a representation object.

three components of this goal: EXPERIMENT, MOTION, and HOUSE-DOOR. The first component (EXPERIMENT) identifies the type of task to be performed. An experiment results in new information about the device, so appropriate plans involve mutation: (a) how the device is used by modifying a known plan, or (b) the device itself by modifying some characteristic of the device's static representation. EDEXP matches the EXPERIMENT task to a list of available plans and those shown in Table 6.1: are returned.

Feature Modification Plans for a Simple Door	
TYPE HINGE	Change function of hinge(s)
MATL HINGE	Change material of hinge(s)
MOVE HINGE	Change location of hinge(s)
SIZE HINGE	Change size of hinge(s)
NUMBER HINGE	Change number of hinges
SIZE DOORSLAB	Change size of doorslab
MATL DOORSLAB	Change material of doorslab

Table 6.1: Feature modification plans for door mutation in EDEXP.

Each plan modifies a single object characteristic at a time. For example, the MOVE HINGE plan (highlighted) selects a hinge, changes the location where it connects the doorslab and doorway, and modifies the appropriate connection in the door representation.

The remaining task components (MOTION and HOUSE-DOOR) define the context in which the goal is evaluated. MOTION identifies the device characteristic (i.e., FONM position state) of interest, and HOUSE-DOOR identifies the device of interest. The available plans are ordered with respect to how closely they match the goal. MOVE HINGE is the only plan in Table 6.1: which is directly related to door motion, since a change in connectivity (according to FONM device dynamics) can affect motion capability.

6.3.2 Plan Application

A feature modification plan is applied by modifying the door's static device representation. In the MOVE HINGE plan, plan application is broken into two steps: (1) a new location is selected for the hinge, and (2) the prior connection is updated to the new connection. In the current simulation, hinge locations are only allowed to be on the boundaries between the doorslab and the doorway. A new location is selected randomly from the list of unused locations available.

When a new/unique location is selected, the connection associated with the hinge being moved is updated to the new location on both the doorslab and the doorway, as shown in

Fig. 6.10. In this example, HINGE.1 is relocated to LOCATION.1 and LOCATION.2 (at 1

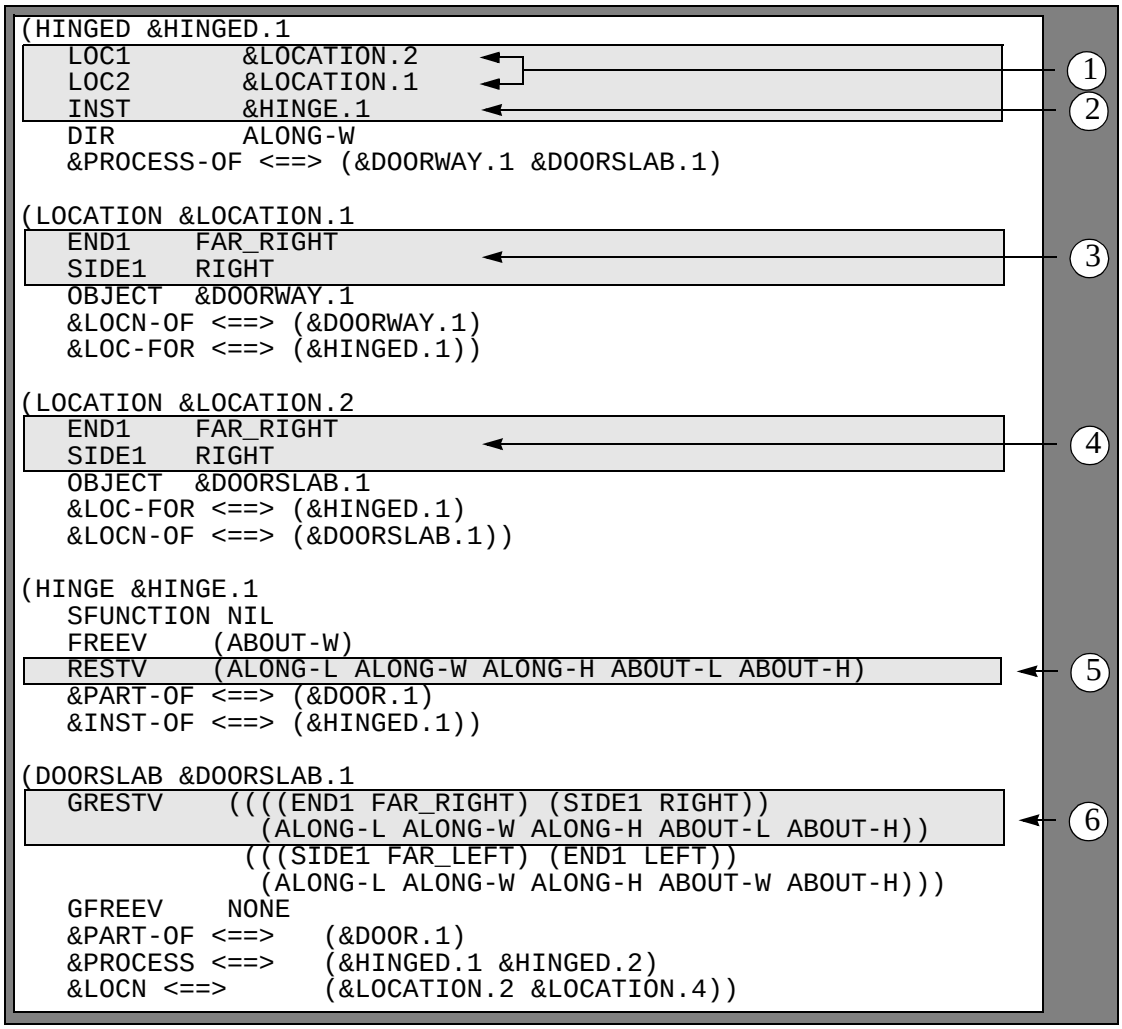


Figure 6.10 Change in representation for HINGED.1 when HINGE.1 location is changed, and the associated changes in object restraint for DOORSLAB.1.

and 2 in the figure). Both locations refer to the same place on DOORSLAB.1 and DOORWAY.1 (3 and 4), namely ((END1 FAR_RIGHT) (SIDE1 RIGHT)). The restraint state on DOORSLAB.1 is the intersection of restraints at each connection, and the union of restraints across connections. DOORWAY.1 is completely restrained, so the restraint state at LOCATION.1 and LOCATION.2 is the same as that for HINGE.1 (5). The global restraint vector (GRESTV) is shown at 6. This vector is used to determine the mobility of DOORSLAB.1.

6.3.3 Plan Evaluation

A plan is evaluated by first determining the effect of its application on device dynamics, and then by determining how changes in device dynamics affect the original goal. The MOVE HINGE plan produces a change in connectivity between DOORSLAB.1, DOORWAY.1, and HINGE.1. With respect to the FONM representation of device statics and dy-

namics, changes in connectivity result in local and global changes to device restraint states. When the MOVE HINGE plan is applied, the new restraint states must be identified and integrated with the existing restraints from HINGE.2 before any determination of motion capability (i.e., mobility) can be made. The rules for determining object mobility given its restraints in EDEXP are shown in Table 6.2. According to this table, an object which is

Object Mobility Recognition Rules		
Name	**Condition (Object state)**	**Freedom**
FRR1	OBJECT1 is grounded	NONE = RIGID = FIXED
FRR2	the union of restraints on OBJECT1 is same as the number of dimensions defined	RIGID, by inheritance
FRR3	OBJECT1 is connected at more than one location to grounded objects	RIGID
FRR4	OBJECT1 is connected to a single object in more than one location and the intersection of dimrs is empty	RIGID
FRR5	otherwise	at least one axis is free

Table 6.2 Object restraint recognition rules in EDEXP.

grounded (FRR1) has all degrees of freedom restrained. Grounded objects are *rigid*, or completely fixed to whatever frame of reference is involved, so they have no mobility (i.e., NONE). In the EDEXP demonstration, fixed means that DOORSLAB.1 is effectively welded to DOORWAY.1 and cannot move. The second mobility recognition rule (FRR2) combines all of the local restraints on the object. When taken as a union, they determine the object's overall, or global, restraint. If all global dimensions are restrained, then the object is rigid by virtue of being connected to objects which collectively eliminate its ability to move. The door shown in Fig. 6.3a is rigid by rule FRR2. FRR3 is similar, in that if the object is connected to grounded objects at any two points in different dimensions, then it is grounded. The same is true if the object is connected to a single grounded object at more than one location (FRR4), if there is no sharing of restraint locations. Any other object is mobile and covered by FRR5, which states that the object will have at least one dimension free to move and identifies the dimension(s). As an example of how object mobility is determined from the connections of components, consider the nonfunctional door depicted in Fig. 6.11. The door clearly cannot move, because the locations where the hinges are con-

nected produce restraints which work against each other.

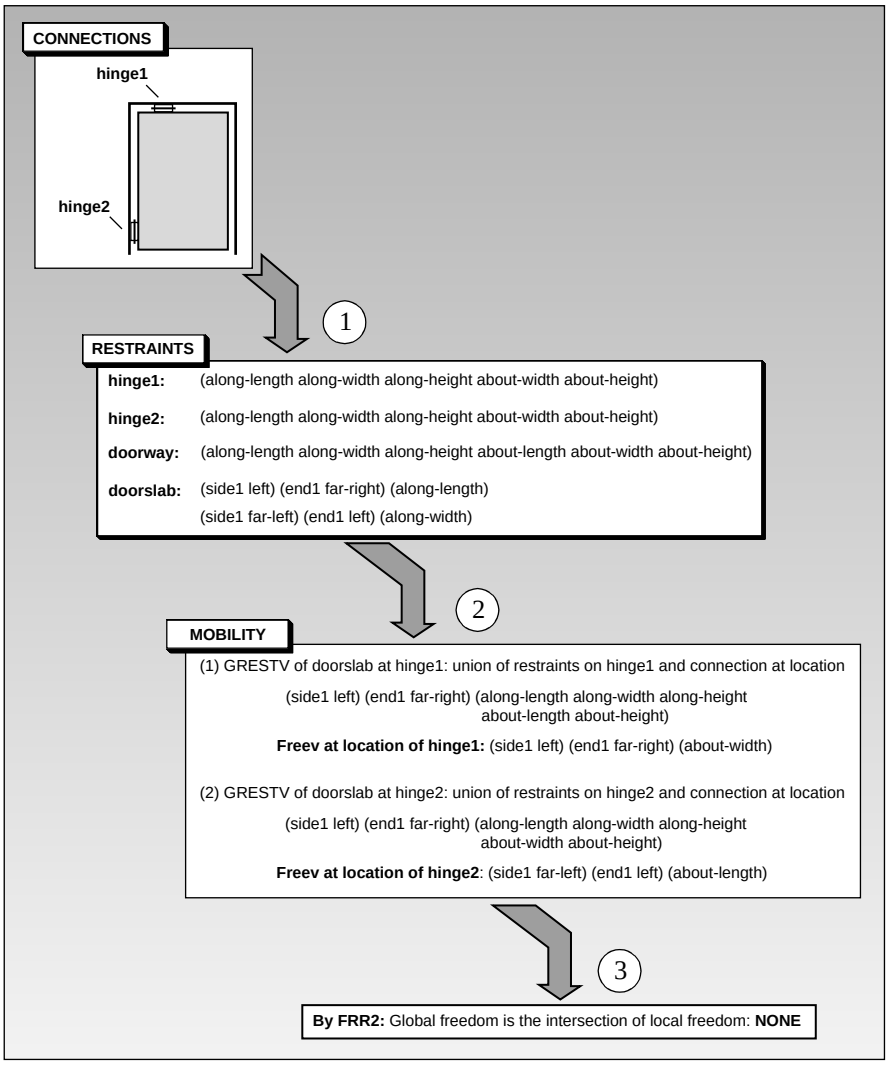


Figure 6.11 Determination of object mobility in EDEXP. Restraints are determined from connections in the door representation.

The restraints box (at 1) identifies the local restraints associated with the hinges, the fixed doorway, and the doorslab, as a result of their mutual connectivity. The mobility box (at 2) shows the combination of doorslab local restraints (i.e., GRESTV) at each hinge location. The mobility vector (FREEV) of the doorslab at each hinge location is the dimension complement of the associated restraint in GRESTV. The global mobility of the doorslab (at 3) is the intersection of the local doorslab mobilities. In this example, there is no intersection so the doorslab has a mobility of NONE and is effectively rigid.

Once the new object's mobility is determined, four motion rules are used to determine

if/how the new door can move. The four rules used in this model are shown in Table 6.3.

Motion Recognition Rules		
Name	**Condition (object state)**	**Action**
MOTION-LINEAR	OBJECT1 is mobile in at least one translational dimension and no rotational dimensions.	OBJECT1 is capable of LINEAR motion
MOTION-CURVI-LINEAR	OBJECT1 is mobile in at least two translational dimensions and at least one rotational dimension.	OBJECT1 is capable of CURVI-LINEAR motion
MOTION-ROTATIONAL	OBJECT1 is mobile in at least one rotational direction but no translational dimensions.	OBJECT1 is capable of ROTATIONAL motion
MOTION-SLIDING	OBJECT1 is mobile in at least one translational dimension, and OBJECT1 is in direct normal contact to that dimension with OBJECT2	OBJECT1 is capable of SLIDING motion

Table 6.3 Motion recognition rules in EDEXP.

When the MOVE HINGE plan is applied and the new restraint and mobility vectors are identified, the motion rules thus evaluate doorslab motion by identifying the value of the mobility vector. Consider the functional door shown in Fig. 6.12.

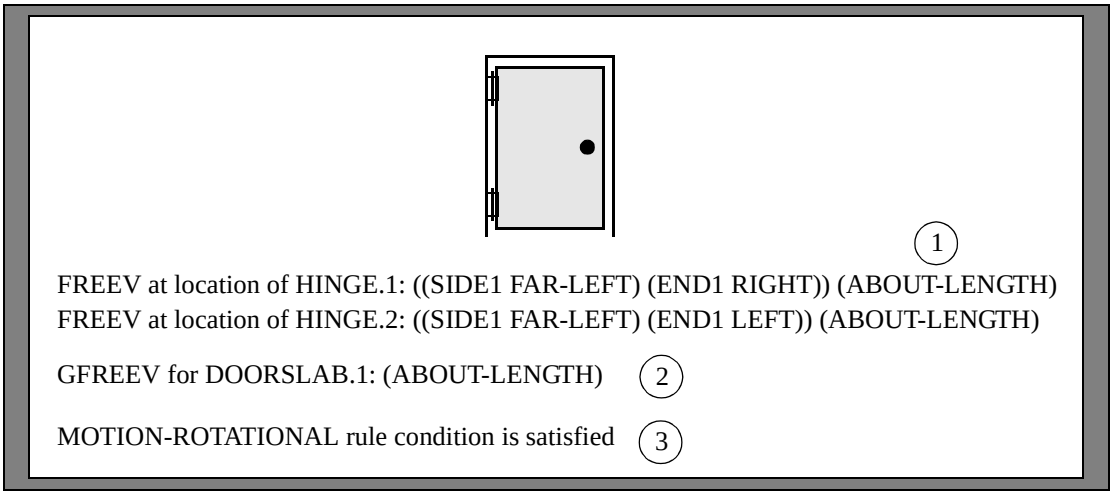


Figure 6.12 Functional door, local and global mobility, and motion rule which identifies rotational capability.

The doorslab pictured Fig. 6.12 has two local mobility vectors, one for each connection location. At each location the doorslab is free to rotate about its vertical axis (i.e., ABOUT-

LENGTH, at 1). The intersection of local mobility is the doorslab mobility (2), which is the single rotational (ABOUT-LENGTH) dimension. MOTION-ROTATIONAL is the only motion type shown in Table 6.3 which can be recognized given this mobility.

Any mutation to an object which changes its restraints results in an update of the object's mobility. This only happens if the object isn't already rigid. The local behavior of rigid objects is ignored in EDEXP. For example, DOORWAY.1 is grounded, so it is rigid. After the first simulation iteration, EDEXP no longer updates mobility on the doorway, because the changed restraints will not make the doorway any more or less rigid.

Once a determination of motion capability is made for the door, the plan can be evaluated with respect to the original task, since any change in motion capability is important in the (EXPERIMENT #{MOTION} . #{HOUSE-DOOR}) goal. In the EXPERIMENT task, the same plan is applied repeatedly until a rule which relates the hinge being moved in the MOVE HINGE plan, and the door, to door motion, can be generalized.

6.3.4 Hypothesis Generation and Testing

The hypothesis generation phase in EDEXP is shown in Fig. 6.13. When the MOVE HINGE plan is applied, it is evaluated by comparing the results to the original task. If the modified door can move, then the plan application is noted as a success. Otherwise it is not-

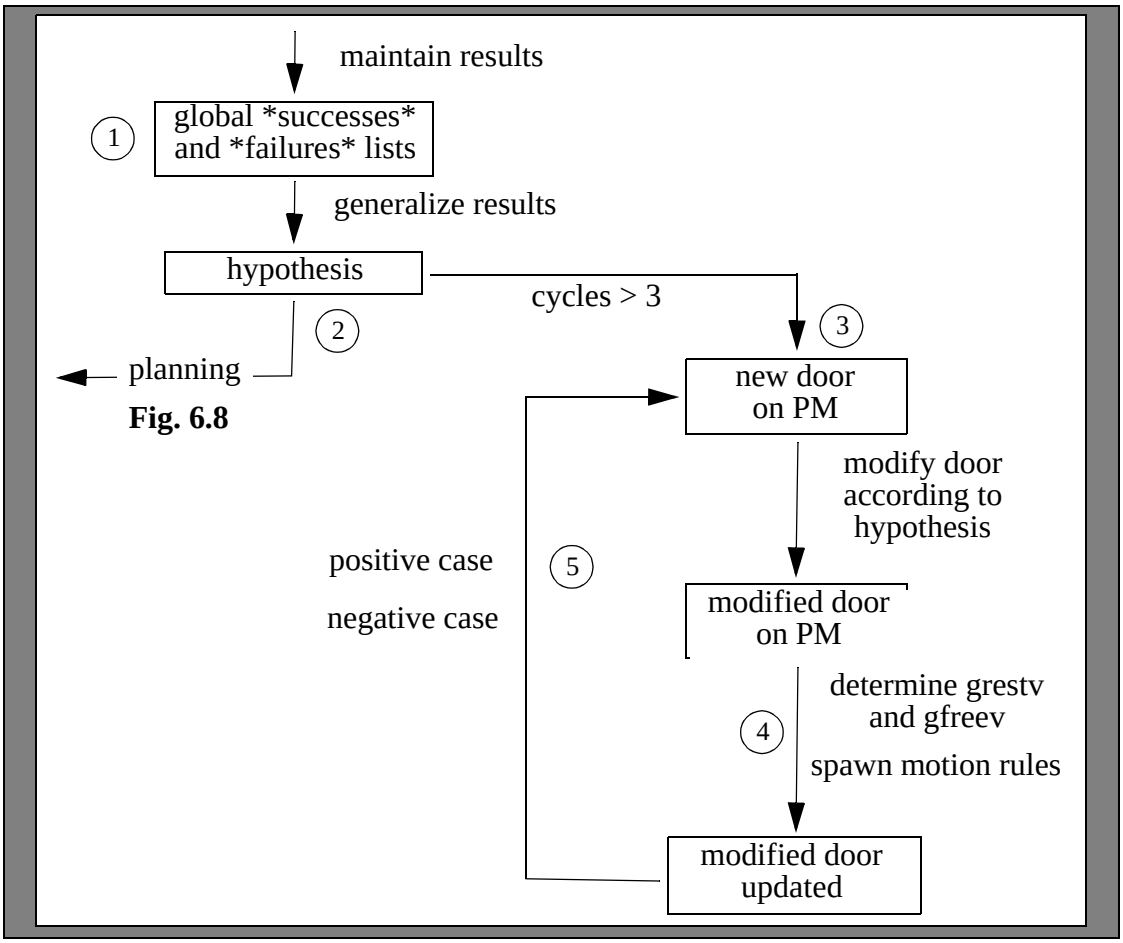


Figure 6.13 Flow chart for hypothesis generation in EDEXP.

ed as a failure. Planning results are maintained for generalization (at 1). The result is a copy of the associated door representation. When two planning iterations have been made, a hypothesis is generalized by comparing the locations of hinges and whether or not they resulted in door rotational capability (2). The hypothesis takes the form of a generalized hinge location, and is generated by looking at the results of successive plan applications. If the results of consecutive planning applications are the same (success or failure), then similarities between hinge locations of the associated doors are identified. If the results of consecutive planning applications are different, then differences between hinge locations of the associated doors are identified. Hinge locations which are on both lists (similarities and differences) are of little value, so the two lists are intersected after every iteration and the remains become the new hypothesis.

If the hypothesis remains consistent between successive plan applications, a counter is incremented, otherwise the counter is initialized. When the count (arbitrarily) reaches 4 (at 3), then planning applications are stopped, and a testing phase is entered. The testing phase amounts to taking the hypothesis and using it with the MOVE HINGE plan to locate a hinge on a new door, and then simulating the change to determine if the door can rotate (4). Two tests are performed: (1) a hinge placement which results in a rotational door, and (2) a hinge placement which results in a rigid door (at 5).

The hypothesis generation and testing phase is controlled by demons which perform the updating process while plan applications are made. When the count is reached, the demons switch EDEXP into hypothesis testing phase.

6.4 EDEXP Implementation

The basis for EDEXP is the RHAPSODY demon-based AI development tool written at UCLA [Turner and Reeves, 1987]. EDEXP and RHAPSODY were originally written in T [Reese 1984], a scheme-based Lisp dialect developed at Yale, and were ported to Common Lisp in 1988. RHAPSODY provides representational semantics, the McDypar parser, a generator, demon and agenda packages, and graphics facilities. In addition to the graphic display capabilities provided by RHAPSODY, the EDEXP model has its own representation to graphic parser and graphical planning interface. The graphic parser reads representation descriptions of a door and displays an icon representation of the door components. EDEXP makes use of 15 demons to analyze the house-door example described in this chapter. An annotated trace of the example follows. Fig. 6.14a depicts the door representation used in the simulation. The conceptual goal given to the system is represented in Fig. 6.14b.

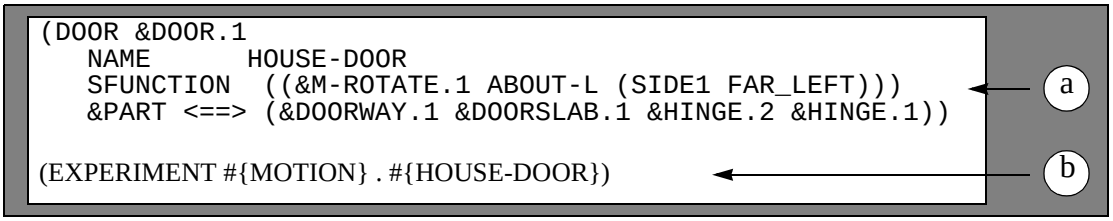


Figure 6.14 Conceptual input to the EDEXP **Doors3** experimentation simulation.

```
> (doors-demo)
```

```
   Spawning demon: EXP-PLAN-DEMON.1
   ========================================================
   TEST: IF   the goal is to EXPERIMENT and develop an
              understanding about some pattern on some template
              (device or part)
   ACT:  THEN try looking for a plan which will be useful
              toward this end
   ========================================================

running agenda #{AGENDA:TMP.2}:

   Running demon: EXP-PLAN-DEMON.1

        trying to find a plan for the current goal:

                EXPERIMENT
                #{MOTION}
                #{HOUSE-DOOR}

        matching the goal patterns to template functionality:

                #{MOTION}
                #{HOUSE-DOOR}

        evaluating experimental plans
```

*The planner is designed to choose among many possible plans, however, it currently prioritizes plans so that the **MOVE** plan is chosen.*

```
        choosing an element to manipulate
```

*Like the choice of plans, this demonstration selects **HINGE.1** as the element to manipulate, and object **location** as the mutation characteristic.*

```
        go find structurally allowable locations on
        #{DOORWAY.1} and #{DOORSLAB.1}

        moving #{HINGE.1}

        has ((SIDE1 FAR_LEFT) (END1 MID_RIGHT)) been tried?
```

*The model has a simple representation of location, based on **SIDE1/SIDE2** and **END1** and **END2.** The model has a correspondingly simple notion of value **(FAR_LEFT, MID_LEFT, LEFT, MIDDLE, RIGHT, MID_RIGHT, FAR_RIGHT)**. When the model selects a location for a hinge placement, a list of locations chosen is maintained so that the model does not continue to re-select the same location over and over again. The selection is otherwise random.*

```
        change physical locations of #{HINGE.1} on #{DOORWAY} to:

                ((SIDE1 FAR_LEFT) (END1 MID_RIGHT))
```

```
change physical locations of #{HINGE.1} on #{DOORSLAB} to:

        ((SIDE1 FAR_LEFT) (END1 MID_RIGHT))
```

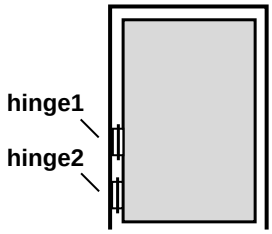


Figure 6.15 A functional door where both hinges have the same local freedom.

*Once **HINGE.1** has been relocated, the door has effectively been mutated into a new door, and the next step in the process is to evaluate the local mobility. This is accomplished with 4 demons. Mobility is determined both locally, at each hinge location, and globally. The global mobility is defined as the union of local mobilities, and a mobility is defined as a motion degree of freedom.*

```
what is the mobility on #{DOORWAY.1}?

        local freedom: NONE

what is the mobility on #{DOORSLAB.1}?

        global freedom: (ABOUT-L)
        local freedom:
  (1)     ((SIDE1 FAR_LEFT) (END1 MID_RIGHT)) (ABOUT-L)
  (2)     ((SIDE1 FAR_LEFT) (END1 LEFT)) (ABOUT-L)
```

*The new door has **HINGE.1** located at the position noted at (1) in the shaded portion above, and **HINGE.2** located at the position noted at (2). The dimension noted on the right (3) represents the local freedom for each hinge, and the dimension at (4) represents the intersection of local mobility on the doorslab as an object. At this point, the motion demons associated with this object are placed onto the PDI agenda to determine its motion capability.*

```
Spawning demon: MOTION-LINEAR.1
=======================================================
TEST: IF   an object is globally free in at least
           one linear axis
ACT:  THEN that object is capable of LINEAR motion
=======================================================

Spawning demon: MOTION-CURVI-LINEAR.1
=======================================================
TEST: IF   an object is globally free in all directions,
ACT:  THEN that object is capable of CURVI-LINEAR motion
=======================================================
```

```
Spawning demon: MOTION-SLIDING.1
=========================================================
TEST: IF   an object is globally free in at least one
           direction, and
      IF   the object is in direct contact normal to
           that direction to another object,
ACT:  THEN that object is capable of SLIDING motion
=========================================================

Spawning demon: MOTION-ROTATIONAL.1
=========================================================
TEST: IF   an object is globally free in at least one
           rotational axis
ACT:  THEN that object is capable of ROTATIONAL motion
=========================================================
```

*The four demons are placed onto the agenda **AGENDA:TMP.3**. The demons are polled once per cycle against the conceptual structures on PM. Since there is only one object being considered, demon interpretation amounts to seeing if the device enables any motion type and then instantiating a motion process of the appropriate type.*

```
running agenda #{AGENDA:TMP.3}:

   Running demon: MOTION-ROTATIONAL.1

        #{DOORSLAB.1} is capable of ROTATIONAL motion
```

*For this door representation and mobility, **MOTION-ROTATIONAL.1** is satisfied. The **+ACT** (positive test action) is part of the demon which tells what to do if its test condition is met, and, in the **MOTION-ROTATIONAL.1** demon, sets the sfunction slot of **DOOR.1** to **ROTATIONAL**. Part of the **+ACT** of **MOTION-ROTATIONAL.1** is to set a binary flag which is tested by all demons. During any cycle, if the flag is set, the demon will execute another component, its **-ACT** (negative test action). The **-ACT** of all four demons is to be taken off the agenda.*

```
   Killing demon: MOTION-ROTATIONAL.1 with kill value: +ACT

   Running demon: MOTION-SLIDING.1

   Killing demon: MOTION-SLIDING.1 with kill value: -ACT

   Running demon: MOTION-CURVI-LINEAR.1

   Killing demon: MOTION-CURVI-LINEAR.1 with kill value: -ACT

   Running demon: MOTION-LINEAR.1

   Killing demon: MOTION-LINEAR.1 with kill value: -ACT
```

Since a new device has been created, and its capacity to rotate has been identified, the ex-

periment has faithfully been executed (noted at 1 below) and the plan must now be evaluated. In experimentation, a plan is considered good as long as something happens. Any good plan is worth reapplying with new values to ascertain whether more information can be found. ***REPLAN-DEMON.1*** *is used to reapply the current plan.*

```
   Spawning demon: REPLAN-DEMON.1
   ========================================================
   TEST: IF   the experiment is completed due to a
              generalization goal, and
         IF   the plan is still considered a GOOD plan,
   ACT:  THEN re-plan using the current plan.
   ========================================================
```

```
   Killing demon: EXP-PLAN-DEMON.1 with kill value: +ACT   <-------- (1)
```

```
   Running demon: REPLAN-DEMON.1

        make a copy of #{DOOR.1} for similarity matching

        #{DOOR.1} succeeded on goal ((#{MOTION} . #{HOUSE-DOOR}))

        MOVE is still a good plan and is being retried
```

Here the plan evaluation metric says that if the new door is capable of the same kind of motion that the original door had, then note the change made and retry the plan with new values. A side-effect of ***EX-PLAN-DEMON.1*** *is the generation of a hypothesis for how the experiments and plans affect the overall goal.* ***SIMIL-DEMON.1*** *looks for simple consistencies and inconsistencies in the results of plans evaluated, and to see if a prediction of future behavior can be made. The demon locally keeps track of the number of plan applications which have resulted in a consistent hypothesis. Hypothesis verification is attempted when the value exceeds 3.*

```
   Spawning demon: SIMIL-DEMON.1
   ========================================================
   TEST: IF  similarities between successful planning
             attempts and unsuccessful planning attempts
             are consistent,
   ACT: THEN try to test if the rule can predict the
             outcome of a trial
   ========================================================

running agenda #{AGENDA:TMP.4}:

   Running demon: SIMIL-DEMON.1

        re-evaluating plan: MOVE

        moving #{HINGE.1}

        has ((SIDE1 FAR_RIGHT) (END1 RIGHT)) been tried?

        change physical locations of #{HINGE.1} on #{DOORWAY} to:
```

```
        ((SIDE1 FAR_RIGHT) (END1 RIGHT))

change physical locations of #{HINGE.1} on #{DOORSLAB} to:

        ((SIDE1 FAR_RIGHT) (END1 RIGHT))
```

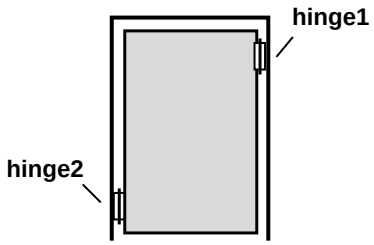


Figure 6.16 A nonfunctional door where both hinges have the same local freedom.

```
what is the mobility on #{DOORWAY.1}?

        local freedom: NONE

what is the mobility on #{DOORSLAB.1}?

        global freedom: NONE   <-------------------------------- (1)
        local freedom:
                ((SIDE1 FAR_RIGHT) (END1 RIGHT)) (ABOUT-L)
                ((SIDE1 FAR_LEFT) (END1 LEFT)) (ABOUT-L)
```

*The current door's mobility is evaluated with mobility rule **FRR2**. The local freedom of motion for the doorslab, at both connection locations, is **ABOUT-L**. However, when the location/mobility combinations are intersected, the result is a fixed door (1).*

```
Spawning demon: MOTION-LINEAR.2
Spawning demon: MOTION-CURVI-LINEAR.2
Spawning demon: MOTION-SLIDING.2
Spawning demon: MOTION-ROTATIONAL.2
```

*The text of these demons has already been shown, and described, as they appear during the simulation. In this and future occurrences they will be collapsed to save space. Since the global freedom of motion is restrained for the new object, none of the motion demons +**ACT**s will fire.*

```
running agenda #{AGENDA:TMP.5}:

Running demon: MOTION-ROTATIONAL.2
Killing demon: MOTION-ROTATIONAL.2 with kill value: -ACT
Running demon: MOTION-SLIDING.2
Killing demon: MOTION-SLIDING.2 with kill value: -ACT
Running demon: MOTION-CURVI-LINEAR.2
Killing demon: MOTION-CURVI-LINEAR.2 with kill value: -ACT
```

```
   Running demon: MOTION-LINEAR.2
   Killing demon: MOTION-LINEAR.2 with kill value: -ACT

   Spawning demon: REPLAN-DEMON.2
   ========================================================
   TEST: IF   the experiment is completed due to a
              generalization goal, and
         IF   the plan is still considered a GOOD plan,
   ACT:  THEN re-plan using the current plan.
   ========================================================

running agenda #{AGENDA:TMP.2}:

   Running demon: REPLAN-DEMON.2

        make a copy of #{DOOR.1} for similarity matching

        #{DOOR.1} failed on goal ((#{MOTION} . #{HOUSE-DOOR}))

        MOVE is still a good plan and is being retried
```

*In this case, the **MOVE** plan is being retried because the plan produced a design which no longer functions at all, let alone as it did originally. New versions of **REPLAN-DEMON** and **SIMIL-DEMON** are created for every planning cycle. The **SIMIL-DEMON** keeps track of the overall number of experimental iterations and the number of plans which have been evaluated successfully. When a device plan succeeds with respect to the task, EDEXP looks for similarities between the current and previous device (i.e., hinge locations). When a device plan fails with respect to the task, EDEXP looks for differences from the current and previous device. **SIMIL-DEMON** keeps track of the similarities and differences and slowly evolves a hypothesis of the experimental results.*

```
   Spawning demon: SIMIL-DEMON.2
   ========================================================
   TEST: IF  similarities between successful planning
             attempts and unsuccessful planning attempts
             are consistent,
   ACT: THEN try to test if the rule can predict the
             outcome of a trial
   ========================================================

running agenda #{AGENDA:TMP.6}:

   Running demon: SIMIL-DEMON.2

        re-evaluating plan: MOVE

        moving #{HINGE.1}

        has ((END1 FAR_RIGHT) (SIDE1 MID_LEFT)) been tried?

        change physical locations of #{HINGE.1} on #{DOORWAY} to:
```

```
            ((END1 FAR_RIGHT) (SIDE1 MID_LEFT))

        change physical locations of #{HINGE.1} on #{DOORSLAB} to:

            ((END1 FAR_RIGHT) (SIDE1 MID_LEFT))
```

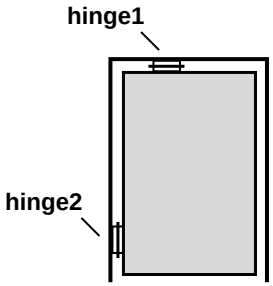


Figure 6.17 Nonfunctional door where hinges have dissimilar local freedom.

```
        what is the mobility on #{DOORWAY.1}?

            local freedom: NONE
```

```
        what is the mobility on #{DOORSLAB.1}?

            global freedom: NONE  <-------------------------------  (1)
```

*The current door's mobility is evaluated with mobility rule **FRR3**. The doorslab is connected at different locations, and in different dimensions, to a grounded object (DOORWAY.1), so there is no global mobility for DOORSLAB.1 (1).*

```
   Spawning demon: MOTION-LINEAR.3
   Spawning demon: MOTION-CURVI-LINEAR.3
   Spawning demon: MOTION-SLIDING.3
   Spawning demon: MOTION-ROTATIONAL.3

running agenda #{AGENDA:TMP.7}:

   Killing demon: MOTION-ROTATIONAL.3 with kill value: -ACT
   Killing demon: MOTION-SLIDING.3 with kill value: -ACT
   Killing demon: MOTION-CURVI-LINEAR.3 with kill value: -ACT
   Killing demon: MOTION-LINEAR.3 with kill value: -ACT

   Spawning demon: REPLAN-DEMON.3
   ========================================================
   TEST: IF   the experiment is completed due to a
              generalization goal, and
         IF   the plan is still considered a GOOD plan,
   ACT:  THEN re-plan using the current plan.
   ========================================================

running agenda #{AGENDA:TMP.2}:
```

```
   Running demon: REPLAN-DEMON.3

        make a copy of #{DOOR.1} for similarity matching

        #{DOOR.1} failed on goal ((#{MOTION} . #{HOUSE-DOOR}))

        MOVE is still a good plan and is being retried
```

Once again, the failure of the plan due to moving HINGE.1 to a location where the door cannot rotate is valuable, since the new location was unique and increases the number and type of locations which can be used for hypothesis generalization.

```
   Spawning demon: SIMIL-DEMON.3
   ========================================================
   TEST: IF  similarities between successful planning
             attempts and unsuccessful planning attempts
             are consistent,
   ACT: THEN try to test if the rule can predict the
             outcome of a trial
   ========================================================

running agenda #{AGENDA:TMP.8}:

   Running demon: SIMIL-DEMON.3

        re-evaluating plan: MOVE

        moving #{HINGE.1}

        has ((SIDE1 FAR_LEFT) (END1 RIGHT)) been tried?

        change physical locations of #{HINGE.1} on #{DOORWAY} to:

                ((SIDE1 FAR_LEFT) (END1 RIGHT))

        change physical locations of #{HINGE.1} on #{DOORSLAB} to:

                ((SIDE1 FAR_LEFT) (END1 RIGHT))
```

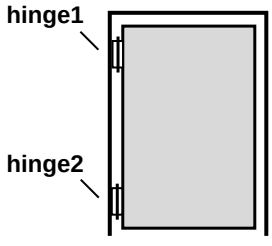


Figure 6.18 Normal functional door.

EDEXP tries a location which results in the original door because its attempt to construct a hypothesis does not maintain the original device hinge locations as part of the history, so

this set of locations is considered valid in this demonstration.

```
        what is the mobility on #{DOORWAY.1}?

                local freedom: NONE

        what is the mobility on #{DOORSLAB.1}?

                global freedom: (ABOUT-L)  <-------------------------- (1)
                local freedom:
                        ((SIDE1 FAR_LEFT) (END1 RIGHT)) (ABOUT-L)
                        ((SIDE1 FAR_LEFT) (END1 LEFT)) (ABOUT-L)
```

*The current door's mobility is evaluated with mobility rule **FRR5**. The doorslab is connected at different locations, but in the same dimensions, to a grounded object (DOORWAY.1), so there is both local and global rotational mobility for DOORSLAB.1 (1). As a result, the **+ACT** of **MOTION-ROTATIONAL.4** fires.*

```
   Spawning demon: MOTION-LINEAR.4
   Spawning demon: MOTION-CURVI-LINEAR.4
   Spawning demon: MOTION-SLIDING.4
   Spawning demon: MOTION-ROTATIONAL.4

running agenda #{AGENDA:TMP.9}:

   Running demon: MOTION-ROTATIONAL.4

        #{DOORSLAB.1} is capable of ROTATIONAL motion

   Killing demon: MOTION-ROTATIONAL.4 with kill value: +ACT
   Killing demon: MOTION-SLIDING.4 with kill value: -ACT
   Killing demon: MOTION-CURVI-LINEAR.4 with kill value: -ACT
   Killing demon: MOTION-LINEAR.4 with kill value: -ACT

   Spawning demon: REPLAN-DEMON.4
   ========================================================
   TEST: IF   the experiment is completed due to a
              generalization goal, and
         IF   the plan is still considered a GOOD plan,
   ACT:  THEN re-plan using the current plan.
   ========================================================

running agenda #{AGENDA:TMP.2}:

   Running demon: REPLAN-DEMON.4

        make a copy of #{DOOR.1} for similarity matching

        #{DOOR.1} succeeded on goal ((#{MOTION} . #{HOUSE-DOOR}))

        MOVE is still a good plan and is being retried

   Spawning demon: SIMIL-DEMON.4
```

```
   ============================================================
   TEST: IF  similarities between successful planning
             attempts and unsuccessful planning attempts
             are consistent,
   ACT: THEN try to test if the rule can predict the
             outcome of a trial
   ============================================================

running agenda #{AGENDA:TMP.10}:

   Running demon: SIMIL-DEMON.4

   Spawning demon: TESTGEN-DEMON.1
   ============================================================
   TEST: IF   strength is high enough to execute,
   ACT:  THEN execute a plan for trying to test
              the generalized concept by simulation
   ============================================================
        spawning a demon to try to test generalized concept
```

*The demon **SIMIL-DEMON.4** succeeds (highlighted below) when the number of iterations with a consistent hypothesis is at least 4 and the number of planning applications resulting in successes is at least 2. In this case, the successful planning applications alternate with the unsuccessful planning applications, but the hypothesis remains consistent. On the fifth iteration, the demon **TESTGEN-DEMON.1** is spawned. **TESTGEN-DEMON.1** tests the hypothesis generalized by **SIMIL-DEMON** by attempting to generate device instances which are both successful and unsuccessful using the hinge location hypothesis to select HINGE.1 locations.*

```
  Killing demon: SIMIL-DEMON.4 with kill value: +ACT
```

```
        re-evaluating plan: MOVE

        moving #{HINGE.1}

        has ((END1 FAR_RIGHT) (SIDE1 RIGHT)) been tried?

        change physical locations of #{HINGE.1} on #{DOORWAY} to:

                ((END1 FAR_RIGHT) (SIDE1 RIGHT))

        change physical locations of #{HINGE.1} on #{DOORSLAB} to:

                ((END1 FAR_RIGHT) (SIDE1 RIGHT))
```

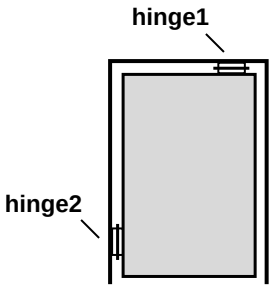


Figure 6.19 Disfunctional door where hinges have dissimilar local freedoms.

```
      what is the mobility on #{DOORWAY.1}?

              local freedom: NONE
```

```
      what is the mobility on #{DOORSLAB.1}?

              global freedom: NONE     <-------------------------  (1)
```

*The current door's mobility is evaluated with mobility rule **FRR3**. The doorslab is connected at different locations, but in the same dimensions, to a grounded object (DOORWAY.1), so there is both local and global rotational restraint for DOORSLAB.1 (1). None of the motion demons **+ACT**s fire.*

```
   Spawning demon: MOTION-LINEAR.5
   Spawning demon: MOTION-CURVI-LINEAR.5
   Spawning demon: MOTION-SLIDING.5
   Spawning demon: MOTION-ROTATIONAL.5

running agenda #{AGENDA:TMP.11}:

   Killing demon: MOTION-ROTATIONAL.5 with kill value: -ACT
   Killing demon: MOTION-SLIDING.5 with kill value: -ACT
   Killing demon: MOTION-CURVI-LINEAR.5 with kill value: -ACT
   Killing demon: MOTION-LINEAR.5 with kill value: -ACT

   Spawning demon: REPLAN-DEMON.5
   ========================================================
   TEST: IF   the experiment is completed due to a
              generalization goal, and
         IF   the plan is still considered a GOOD plan,
   ACT:  THEN re-plan using the current plan.
   ========================================================
```

*When **TESTGEN:DEMON** is spawned initially, with the idea that, when enough evidence exists, in the form of consistency in the hypothesis generated over multiple plan applications, an evaluation of the generalized device similarities and differences will be made. Since there should be competition between the desire to obtain more data and the desire to*

*test a hypothesis, the **TESTGEN** demon's priority is raised when there are successive planning attempts which produce the same hypothesis. This priority is a component of all demons, but is only manipulated in this demon. With each consistent hypothesis plan application cycle the demon's priority is incremented. When the number of consistent hypotheses reaches three, **TESTGEN**'s priority places it at the top of the evaluation queue. If successive planning attempts produce inconsistent hypotheses, then **TESTGEN**'s priority is returned to its nominal value. When **TESTGEN-DEMON.1** is fired, it performs two tests. The first is to generate a door with a hinge arrangement which will succeed according to the hypothesis. The second is to generate a door with a hinge arrangement which will fail according to the hypothesis.*

```
running agenda #{AGENDA:TMP.2}:
```

```
 Running demon: TESTGEN-DEMON.1
      try testing (SIDE1 FAR_LEFT) LOCATION for #{HINGE}
```

*The hypothesis generated is **(SIDE1 FAR_LEFT)**, which is a template for the location variables which have been generalized from the specific planning instances. The meaning of this hypothesis is that any hinge location is acceptable as long as it matches this template. Since no reference to any location along the side is specified, and value is acceptable.*

```
      moving #{HINGE.11}
```

```
      has ((SIDE1 FAR_RIGHT) (END1 MID_LEFT)) been tried?
```

```
      has ((SIDE1 FAR_RIGHT) (END1 MID_LEFT)) been tried?

      has ((SIDE1 FAR_RIGHT) (END1 LEFT)) been tried?

      has ((SIDE1 FAR_RIGHT) (END1 MIDDLE)) been tried?

      has ((SIDE1 FAR_RIGHT) (END1 RIGHT)) been tried?

      has ((END1 FAR_LEFT) (SIDE1 LEFT)) been tried?

      has ((END1 FAR_LEFT) (SIDE1 MIDDLE)) been tried?

      has ((END1 FAR_LEFT) (SIDE1 LEFT)) been tried?

      has ((SIDE1 FAR_RIGHT) (END1 LEFT)) been tried?

      has ((END1 FAR_RIGHT) (SIDE1 MID_RIGHT)) been tried?

      has ((END1 FAR_LEFT) (SIDE1 LEFT)) been tried?

      has ((END1 FAR_LEFT) (SIDE1 MIDDLE)) been tried?

      has ((END1 FAR_RIGHT) (SIDE1 LEFT)) been tried?

      has ((SIDE1 FAR_LEFT) (END1 MID_RIGHT)) been tried?
```

When hypothesis testing is begun, a copy of the door representation is made and placed onto planning memory. The new hinge1 instance is ***HINGE.11*** *In all previous* ***MOVE HINGE*** *plan applications, a list of locations for HINGE.1 that have been tried was maintained, but the list of locations tried was small compared to the list of available locations (7 on each side, 7 on each end). So random selection generally produced a new location on the first try. In the hypothesis testing phase, only those locations satisfying the constraint can be chosen, so the random selection and comparison takes more attempts. The reason for the identical attempt (highlighted above) is that only those locations which have been tried are maintained, so it is possible to ask the same question twice.*

```
change physical locations of #{HINGE.11} on #{DOORWAY} to:

        ((SIDE1 FAR_LEFT) (END1 MID_RIGHT))

change physical locations of #{HINGE.11} on #{DOORSLAB} to:

        ((SIDE1 FAR_LEFT) (END1 MID_RIGHT))
```

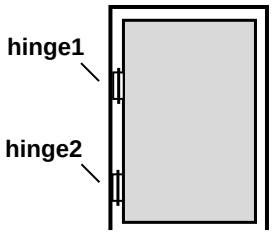


Figure 6.20 Generated functional door from hypothesis.

In the first try, a location for ***HINGE.11*** *is chosen,* **((SIDE1 FAR_LEFT) (END1 MID_RIGHT))***, using the hypothesis, which should yield a door which will rotate.*

```
what is the mobility on #{DOORWAY.6}?

        local freedom: NONE

what is the mobility on #{DOORSLAB.6}?

        global freedom: (ABOUT-L)
        local freedom:
                ((SIDE1 FAR_LEFT) (END1 MID_RIGHT)) (ABOUT-L)
              ((SIDE1 FAR_LEFT) (END1 LEFT)) (ABOUT-L)
```

Which results in a global mobility for ***DOORSLAB.6*** *in the dimension* ***(ABOUT-L).***

```
   Spawning demon: MOTION-LINEAR.6
   Spawning demon: MOTION-CURVI-LINEAR.6
   Spawning demon: MOTION-SLIDING.6
   Spawning demon: MOTION-ROTATIONAL.6

running agenda #{AGENDA:TMP.12}:
```

```
Running demon: MOTION-ROTATIONAL.6

     #{DOORSLAB.6} is capable of ROTATIONAL motion

Killing demon: MOTION-ROTATIONAL.6 with kill value: +ACT
Killing demon: MOTION-SLIDING.6 with kill value: -ACT
Killing demon: MOTION-CURVI-LINEAR.6 with kill value: -ACT
Killing demon: MOTION-LINEAR.6 with kill value: -ACT
```

```
     POSITIVE test succeeded with (SID FAR) LOCATION for #{HINGE}
     attempt NEGATED test with (SID FAR) LOCATION for #{HINGE}
```

*The first portion of the **TESTGEN-DEMON.1** test shows that the generalized hypothesis **(SIDE1 FAR_LEFT)** is valid. The second portion of the test does the same thing on a negative instance of the hypothesis applied to the new door.*

```
     moving #{HINGE.11}

     has ((END1 FAR_LEFT) (SIDE1 MID_LEFT)) been tried?
```

*It takes much fewer attempts to find a negative instance of the hypothesis, which is any location which violates the **(SIDE1 FAR_LEFT)** constraint.*

```
     change physical locations of #{HINGE.11} on #{DOORWAY} to:

          ((END1 FAR_LEFT) (SIDE1 MID_LEFT))

     change physical locations of #{HINGE.11} on #{DOORSLAB} to:

          ((END1 FAR_LEFT) (SIDE1 MID_LEFT))
```

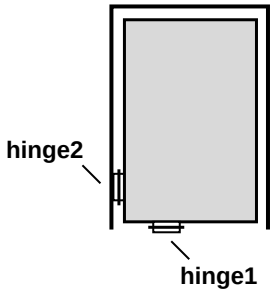


Figure 6.21 Generated disfunctional door from hypothesis.

```
     what is the mobility on #{DOORWAY.6}?

          local freedom: NONE
```

```
     what is the mobility on #{DOORSLAB.6}?

          global freedom: NONE
```

*The negative instance results in no global mobility for **DOORSLAB.6**.*

```
   Spawning demon: MOTION-LINEAR.7
   Spawning demon: MOTION-CURVI-LINEAR.7
   Spawning demon: MOTION-SLIDING.7
   Spawning demon: MOTION-ROTATIONAL.7

running agenda #{AGENDA:TMP.13}:

   Killing demon: MOTION-ROTATIONAL.7 with kill value: -ACT
   Killing demon: MOTION-SLIDING.7 with kill value: -ACT
   Killing demon: MOTION-CURVI-LINEAR.7 with kill value: -ACT
   Killing demon: MOTION-LINEAR.7 with kill value: -ACT

      succeeded test  <-------------------------------------------  (1)

      updating SFUNCTION of #{HINGE.1} to (LOCATION SID FAR)        (2)

   Killing demon: TESTGEN-DEMON.1 with kill value: +ACT
   Killing demon: REPLAN-DEMON.4 with kill value: +ACT
   Killing demon: REPLAN-DEMON.3 with kill value: +ACT
   Killing demon: REPLAN-DEMON.2 with kill value: +ACT
   Killing demon: REPLAN-DEMON.1 with kill value: +ACT
#T
>
```

*The result of **TESTGEN-DEMON.1** is a success (**+ACT** fired), because both the positive and negative hypothesis applications produce doors with the intended structural function (1). As a result, the hypothesis (2) is saved as an attribute of **HINGE.1**.*

The final representation for DOOR.1 is shown in Fig. 6.22 and Fig. 6.23.

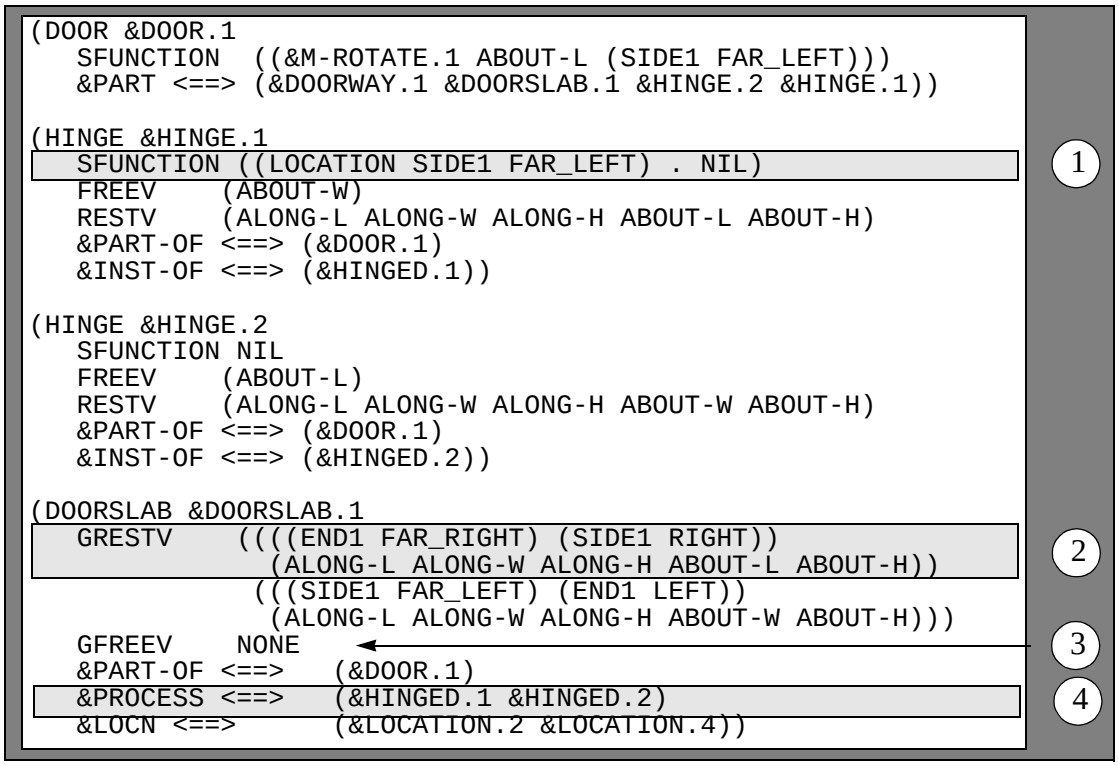


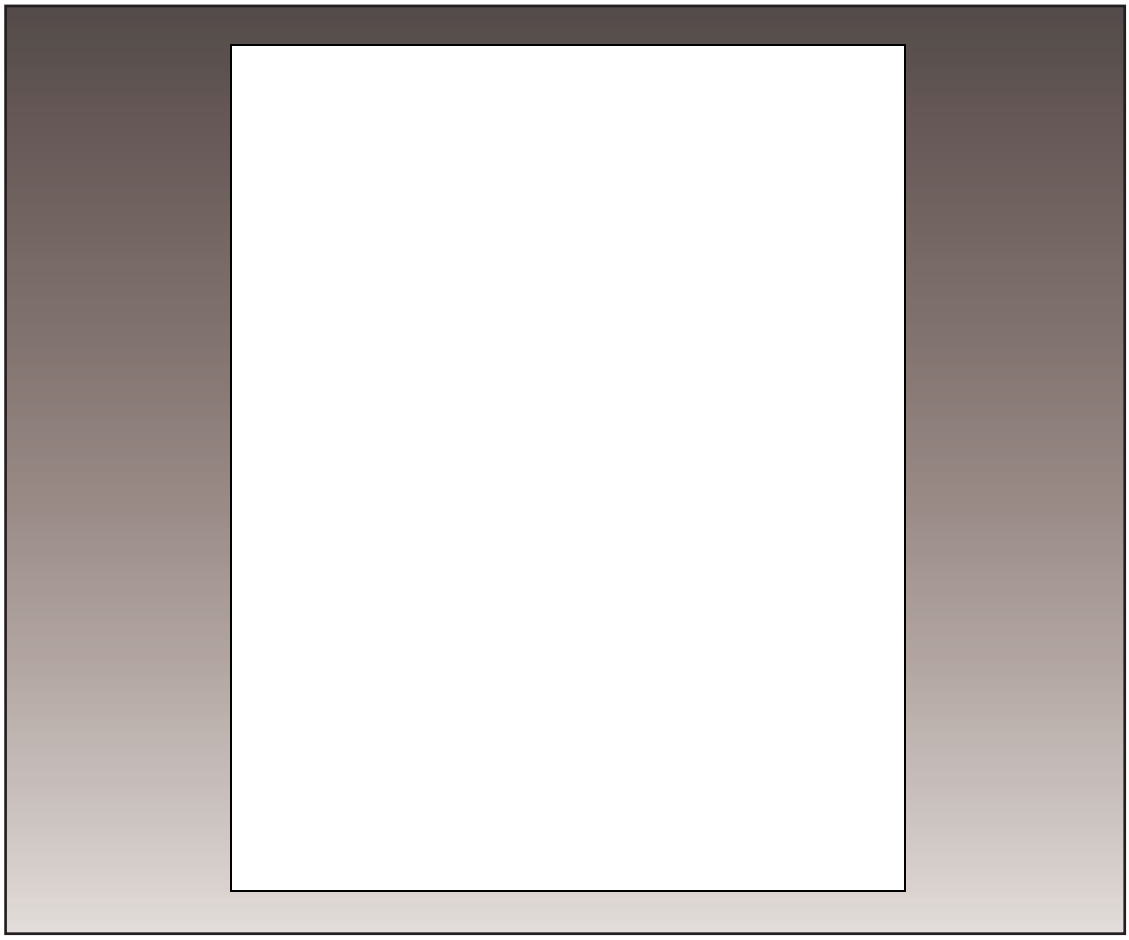
Figure 6.22 Doors demo final representation.

The function of the door illustrated, and its connectivity, is that of the door illustrated in Fig. 6.19, which was the last door representation generated in the planning phase. HINGE.1 has a structural function which has been updated from the hypothesis to include the location constraint (1). The DOORSLAB.1 global restraint vector (2) shows the location of HINGE.1 for that door ((END1 FAR_RIGHT) (SIDE1 RIGHT)) mutation, and DOORSLAB.1 has no mobility (3). The connections are represented as process frames (HINGED.1 and HINGED.2, at 4 in Fig. 6.22 and at 1 in Fig. 6.23). In EDEXP, these structures do not represent the dynamics associated with the associated FONM RESTRAIN process, but they are directly related to the local restraint states seen in the restraint vector GRESTV. The process has two locations, an object, and a dimension. The locations are defined with respect to regions on the object. The restraints are calculated procedurally in

EDEXP.

```
(HINGED &HINGED.1                                              (1)
   LOC1       &LOCATION.2
   LOC2       &LOCATION.1
   INST       &HINGE.1
   DIR        ALONG-W
   &PROCESS-OF <==> (&DOORWAY.1 &DOORSLAB.1)

(LOCATION &LOCATION.1
   END1    FAR_RIGHT
   SIDE1   RIGHT
   OBJECT  &DOORWAY.1
   &LOCN-OF <==> (&DOORWAY.1)
   &LOC-FOR <==> (&HINGED.1))

(LOCATION &LOCATION.2
   END1    FAR_RIGHT
   SIDE1   RIGHT
   OBJECT  &DOORSLAB.1
   &LOC-FOR <==> (&HINGED.1)
   &LOCN-OF <==> (&DOORSLAB.1))

(HINGED &HINGED.2
   LOC1       &LOCATION.4
   LOC2       &LOCATION.3
   ITEM       &HINGE.2
   DIR        ALONG-W
   &PROCESS-OF <==> (&DOORWAY.1 &DOORSLAB.1)

(LOCATION &LOCATION.3
   OBJECT  &DOORWAY.1
   SIDE1   FAR_LEFT
   END1    LEFT
   &LOCN-OF <==> (&DOORWAY.1)
   &LOC-FOR <==> (&HINGED.2))

(LOCATION &LOCATION.4
   OBJECT  &DOORSLAB.1
   SIDE1   FAR_LEFT
   END1    LEFT
   &LOCN-OF <==> (&DOORSLAB.1)
   &LOC-FOR <==> (&HINGED.2))

(M-ROTATE &M-ROTATE.1
   NAME      ROTATIONAL
   FUNC-VARS ((LOCATION 10) (DIR 10) (FREEV 40) (GRESTV 40)))
```

Figure 6.23 Doors demo final representation (continued).



Chapter 7

Comprehending Mechanical Device Descriptions

EDCA (EDison Conceptual Analyzer) is a computer program that reads mechanical device descriptions such as **Toy Gun**, in English, and constructs FONM representations of the device behavior being described.

Toy Gun

"An object is pushed into a barrel, against a spring, compressing the spring until it catches on a trigger. When the trigger is pulled, the spring is released, and the object is propelled from the barrel."

Three topics are addressed in this chapter, followed by an annotated trace of the **Toy Gun** example:

(1) Natural language comprehension and device descriptions
(2) The EDCA architecture
(3) The EDCA approach to constructing conceptual representations

7.1 Natural Language Comprehension of Device Descriptions

Natural language device comprehension is a process for dynamically constructing device representations from natural language input and incorporating these representations in a current understanding of the description meaning, called an episode or scenario. For example, in **Toy Gun**, EDCA must infer the parts associated with the trigger mechanism despite the fact that they are not stated in the story. To do so, EDCA must access known schemas concerning triggers, guns, toys, barrels, and springs. EDCA must disambiguate the meanings of "into" and "catches," which have different meanings in different contexts. EDCA must resolve anaphoric references such as which object is "it" in the first sentence, since "it" could be the barrel, the object, or the spring. As the analysis continues, EDCA has to dynamically recognize that "until" means that the previous process will be disabled by the next process, and relate the processes accordingly. Finally, EDCA must construct representations for the 'loading-a-toy-gun' and 'firing-a-toy-gun' functions from the description of its underlying behavior, and incorporate those representations, if necessary, in memory for future use.

7.2 EDCA - The EDISON Conceptual Analyzer

EDCA implements five components of the EDISON computational architecture: (1) a demon interpreter, (2) a lexicon and long term semantic memory, (3) a working memory, (4) rules for performing lexical analysis, disambiguation, and reasoning, and (5) an episodic memory, as illustrated in the shaded portion of Fig. 7.1. Words are input to the language

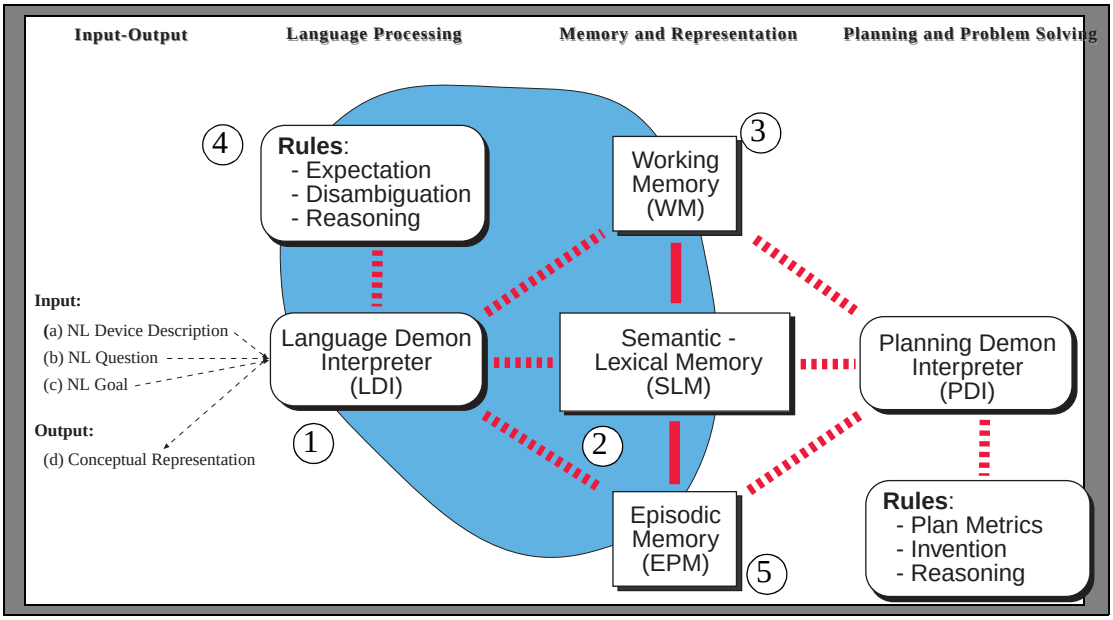


Figure 7.1 The conceptual analyzer components of the EDISON computational architecture. Dotted lines refer to processing control. Solid lines refer to data communication.

demon interpreter (LDI, at 1). A word is matched against entries in the lexicon (SLM, at 2), and candidate conceptual structures are placed onto working memory (WM, at 3). The demons associated with those structures (4) are placed onto an agenda and polled. When a phrase or sentence is fully input, the partially-instantiated structures in working memory are used for conceptual disambiguation. The overall result is integrated in PM (5) and returned as conceptual output.

- **Semantic and Lexical Memory (SLM)**. Semantic memory represents schematic knowledge of objects, events, and their causal interactions. Lexical memory represents the words associated with conceptual structures in semantic memory. During a parse, input words are compared to their counterparts in lexical memory, and the conceptual components associated with different meaning interpretations are maintained for disambiguation. During conceptual disambiguation, concepts in working memory are associated with semantic and lexical memory, which is used to instantiate incomplete patterns.

- **Language Demon Interpreter (LDI)**. The demon interpreter processes textual input word by word. Each word is potentially associated with many schemas in semantic and lexical memory, collectively called the lexicon. These potential structures compete to determine the word's meaning. During parsing, words are presented and disambiguated as well as possible from the local context using disambiguation rules specific to textual analysis. The results are placed in working memory. When a sentence is entirely parsed, the conceptual structures remaining in working memory are compared to schema in se-

mantic and situation memory to disambiguate the overall meaning using rules for reasoning about mechanical knowledge. When completed, the result is copied to episodic memory.

- **Working Memory (WM)**. During a parse, the conceptual analyzer uses textual input to access conceptual structures from a lexicon in semantic and situation memory. The intermediate conceptual meanings are placed on a list called *working memory*. Working memory serves as a source for disambiguation and reasoning since the lexical items accessed from memory represent the current text. These items also serve as memory retrieval cues. Working memory is maintained until a phrase is disambiguated.

- **Language Processing and Reasoning Rules**. During conceptual analysis, rules are needed to comprehend text. Two types of language processing rules are used, for: (1) word-sense disambiguation, and (2) clausal reference. Rules are associated with lexical entries and conceptual patterns in SLM. The conceptual analyzer loads potential concepts into working memory, and the rules associated with each concept are placed onto an agenda. The demon interpreter polls the rules with respect to all the concepts existing in working memory. When a partially constructed sentence is parsed, reasoning rules are used to identify relationships with conceptual patterns in long term memory, and use those patterns to fill in information missing in the text. Three types of reasoning rules are used: (1) recognition rules, (2) prediction rules, and (2) explanation rules.

- **Episodic Memory (EPM)**. Semantic disambiguation and reasoning begin when the phrasal input is complete. The conceptual analyzer copies the conceptualizations remaining in working memory to a memory list called *episodic process memory*. The concepts in episodic process memory represent EDCA's mental model of the text. The concepts in episodic process memory are used as retrieval cues to patterns in semantic and situation memory, to fill in existing gaps in causal relationships, and to perform reasoning tasks. When processing is complete, the patterns in episodic process memory are available to incorporate into semantic and situation memory.

7.3 Conceptual Analysis in EDCA

The demon interpreter (LDI) is the primary processing component coordinating text, lexical access, and memory. EDCA is a conceptual analyzer which uses McDypar [Dyer, 1983], a demon-based parser. Demon-based parsing involves the association of procedural entities with lexical entries. These procedural entities, called *demons* [Dyer, 1983], are context sensitive/specific, and are used to aid meaning disambiguation. McDypar processes words one at a time, or in phrases, places a token for the word onto working memory, and spawns any associated demons onto an agenda. The agenda is cycled through for every new word processed, during which all active demons are tested. Demon processing amounts to polling the demon against the currently instantiated working memory (WM) context. For example, if a word is read whose lexical entry has five possible meanings, then there may be five demons, say one for each meaning, placed on the agenda. When one demon's conditions are met, the associated word meaning is instantiated, and the demons associated with other meanings of the word are removed from the agenda. The meaning found is placed onto WM, which is a blackboard that all demons have access to. When an entire section of text is parsed in this way, the current concepts so associated are placed onto episodic process memory (EPM), where the current conceptualization is being constructed. It is in EPM that SLM is used to resolve and explain conflicts, and to make inferences necessary in filling gaps in device understanding.

7.3.5 Lexical Entries

Lexical entries are the link between words and their various conceptual patterns. The device (FONM) representation governs how devices and physical relationships are organized in semantic memory (SLM). In contrast, a lexical entry organizes the general conceptual patterns which a word might take. The pattern serves as a starting point for how the word, if a particular sense is chosen, will be interpreted. The manner in which meanings are chosen is controlled by the demon interpreter. Consider the first phrase in a second example, **Press** (from "How Things Work"; volume 1, page 38.):

Press

"A screw is rotated in a fixed nut, whereby a longitudinal force is developed in the spindle which thrusts against the part under it."

The lexical entries for the first phrase of **Press** are shown in Fig. 7.2. The notational semantics for lexical entries in the figure are defined as follows. The *def* slot (at 1) refers to the definition of the word, which is used when there is a single conceptual entry associated with the word. When there is more than one meaning for a word, the *m1*, *m2*, *m3*,...,*mn* notation (2) is used to define each meaning (e.g., "in" has three meanings, m1 - m3, represented in its lexical entry). Within a particular meaning definition, the *concept* slot (3) refers to the FONM representation associated with the concept. Fillers which are represented as *?bindingtoken* (e.g., ?part, at 4) are pattern variables which are filled during parsing. The *<== (EXPECT...)* notation (5) refers to a gap-filling demon which searches working mem-

ory looking for a concept with the class named and in the direction named.

```
(word:define screw
  (def    (part 'name 'screw)))

(word:define rotated
  (def    (concept    (process
                         'type  'motion-rotate
                         'name  'rotated
                         'src   ?part  <== (EXPECT 'physobj 'bef)
                         'pivot ?pivot <== (EXPECT 'region 'aft)
                         'realized-evs (list 'rotate.event1)))
          (demons     (note-item 'proc)
                      (new-event))))

(word:define in
  (m1     (concept    (orient
                         'type '&in
                         'name 'in
                         'part1 ?part1 <== (EXPECT 'physobj 'bef)
                         'part2 ?part2 <== (EXPECT 'physobj 'aft))))
  (m2     (concept    (event
                         'part ?part <== (EXPECT 'physobj 'bef)
                         'prop 'position
                         'name 'in)))
  (m3     (concept    (prepobj 'type 'in 'is 'in))))

(word:define fixed
  (def    (concept    (state
                         'type 'restraint
                         'name 'fixed
                         'part ?part <== (EXPECT 'physobj 'aft)
                         'prop 'gfreev
                         'val (quant 'val  'none)))
          (demons     (note-item 'state))))

(word:define nut
  (def    (concept    (part
                         'name 'nut
                         'type &nut))
          (demons     (search-for-ref))))
```

Figure 7.2 Lexical entries for "a screw is rotated in a fixed nut" within the McDypar parsing model and EDCA lexicon.

The *demons* slot (6) refers to demons which are associated with the concept-type in general, and are placed onto the parsing agenda when the lexical item is processed.

7.3.6 Textual Analysis and Disambiguation

The primary task of textual analysis is the disambiguation of word-meaning in working memory. Demons are used by EDCA to disambiguate text in four ways, by: (1) generating semantic (case) expectations, (2) resolving object references, (3) determining textually-related causal relations, and (4) structuring and focusing noun group references. The interaction of these demon types organizes enough content into WM so that in-depth reasoning between WM and the knowledge in SLM can occur. The description of **Toy Gun** demonstrates 1-3 of the demon processing types introduced above and is indicative of the difficulties arising in processing physical device descriptions.

Semantic Expectation

Physically-oriented text is often very specific in terms of word meaning. Words such as "barrel," "spring," "compressing," and "trigger" are words which map into specific conceptualizations. However, these words cannot be interpreted correctly without the surrounding context, because they have semantic dependencies which are resolved during processing. For example, "compressing" is represented as a process (BPP-STORE) which cannot itself be fully understood without its *src* and *from* roles being instantiated. Consider the lexical entry for "compressing" in Fig. 7.3:

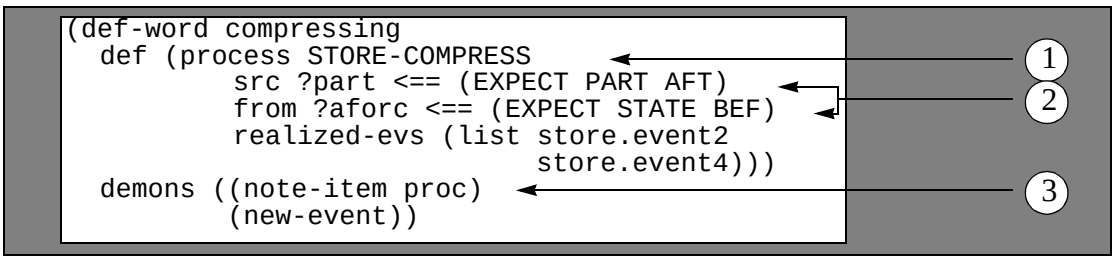
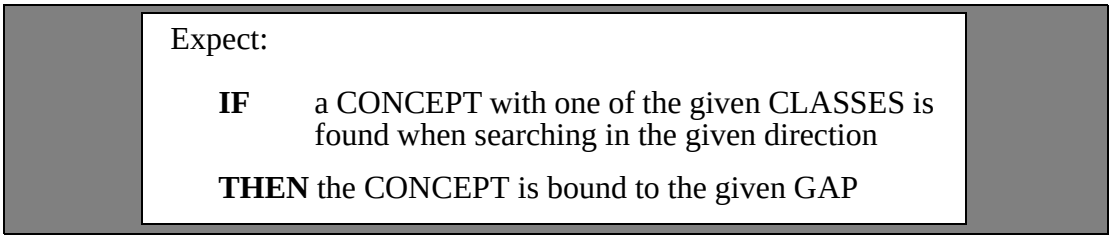


Figure 7.3 Lexical entry for the word "compressing."

If the word "compressing" is read but the roles are not identified, then the word can only be interpreted as a general reference to what is going on, because there is no direct way (through connections to other objects, which require knowing the regions where the behavior will occur) to relate the object to the associated context. The role fillers for a concept are resolved using a demon which searches for the role type in WM and indexes the two concepts appropriately. In the specific example shown, the EXPECT demon (2, also described below) matches the part and state roles necessary to understand "compressing" in **Toy Gun**:



The word "compressing" cannot be completely understood in context until the given roles are resolved, even though the expectations for what the process might mean from different perspectives can be spawned. When "compressing" is read, the EXPECT demons for the *src* and *from* roles are spawned. In this case, the *src* role is filled by "spring," and an instance of the STORE process with "spring" instantiating the *src* role is placed onto WM. In addition, when the word "compressing" is parsed into an instance of the process STORE-COMPRESS, a reference to the process is saved (the demon *note-item*, at 3) as a cue for later semantic disambiguation. A *parse diagram*, illustrating the word processed by EDCA

and how the expect demon works, is shown as Fig. 7.4.

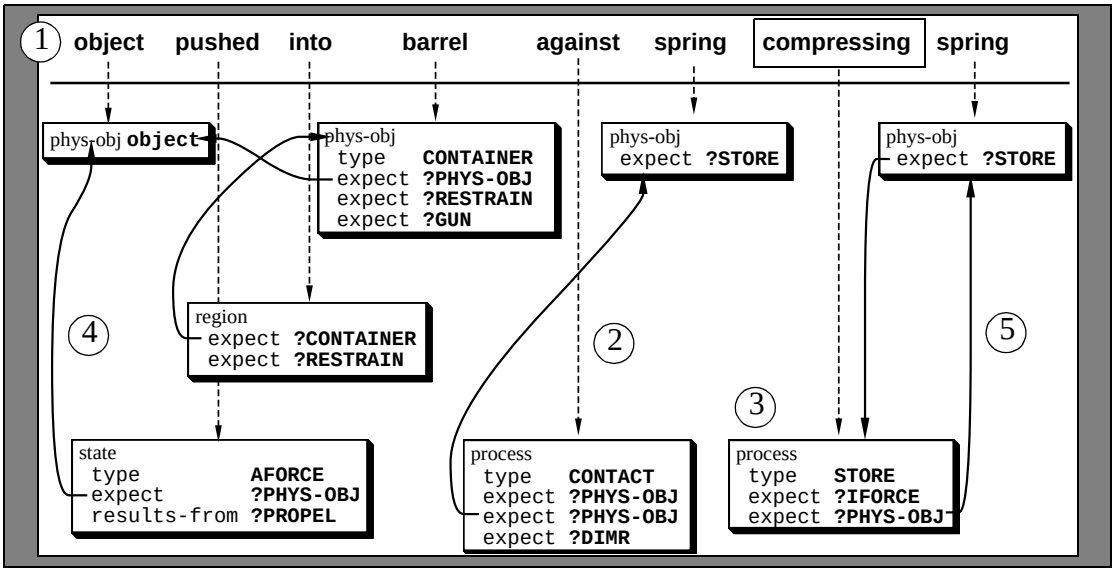


Figure 7.4 Partial parse diagram showing function of the EXPECT demon on "compressing."

Fig. 7.4 shows how the EXPECT demon is used during conceptual analysis to instantiate role fillers and hook conceptualizations together. The figure can be understood as follows. The words which are processed are shown above the solid horizontal line (e.g., "object" at 1). A dotted line is used to depict the correspondence between the word and the associated lexical entry in the EDCA lexicon (e.g., connecting the word "against" and the lexical entry `contact`, at 2). The conceptual representation of a lexical entry, e.g. the STORE process associated with "compressing" (3), shows the EXPECT demons which look into WM for appropriate role fillers. Finally, the solid curved lines represent where the demon eventually finds the filler for the concept and role it is associated with. One example is the line connecting the EXPECT demon for the object role in the AFORCE state with the PHYS-OBJ concept which preceded it (4). In this diagram, all role names, binding variable names, and direction identifiers associated with the EXPECT demon are left out for space considerations. Thus, the second EXPECT demon in the STORE process (5) refers to the demon associated with the *src* role illustrated in Fig. 7.3.

Reference Disambiguation

Object reference is the most common problem encountered in physical descriptions. **Toy Gun** has four object disambiguation instances of two types: (1) pronoun or backward reference, and (2) multiple object reference. The mechanisms used to resolve object references in EDCA are the same; an object is referenced and, depending on the context, it must somehow be remembered until it can be resolved.

1. **Object References:** Semantic expectation is used at the phrasal level to fill conceptual role expectations. However, often roles are not identified locally and must be retrieved from knowledge of previous phrases in EPM. This becomes a reference problem when multiple objects or processes are mentioned prior to the reference in question. In EDCA, this type of reference is associated with physical relations and processes, since they always depend on other processes and components. This problem is exemplified by considering the

second and third phrases of **Toy Gun**:

Toy Gun 2-3

"against a spring, compressing the spring"

The use of "against" and "compressing" in these phrases is indicative of a reference problem. If semantic expectation were to span phrasal boundaries, it would instantiate the first object that satisfied the search criterion. In this case, the *src* role for "against" would be filled by "barrel" while the *src* role for "compressing" would be filled by "object," indicating a breakdown in the use of expectation across phrasal boundaries. The resolution of this conflict has both syntactic and semantic elements.

Syntactic Object Disambiguation

In the first phrase of **Toy Gun 2-3**, the lexical item for the word "against" has an expectation for an object prior to its reference in the text. However, "against" is the first word in the phrase. Therefore, the reference object ("object") must be found from previously read phrases. This is accomplished using a syntactic rule for object reference, illustrated by **Rule OR1** below. Syntactic reference resolution is sometimes required when conceptual knowledge is unavailable or limited, such as in the current example. EDCA maintains a syntactic object focus using reference rules such as **OR1**, although semantic cues check, and can override, syntactic references, when they occur.

Rule OR1:

IF	the current process' subject cannot be identified from the current phrase, and if there is a current subject from a previous phrase
THEN	identify the current process' subject as the subject of the previous phrase

The application of syntactic rules in disambiguating object references is depicted in the parse diagram shown as Fig. 7.5. In this figure, the semantic expectations have been removed and the lexical items for the commas separating the phrases have been shown (e.g., at 1). When a comma or other lexical object is read, a number of demons are spawned (2) which are used to identify syntactic identifiers (such as subject and object) in the current conceptualization. The dotted line at (3) in Fig. 7.5 shows the effect of these demons on their respective phrases. For example, after the first phrase is parsed, the subject is remembered as "object," and the object is identified as "barrel" (3). When the second and third phrases are parsed, only the phrasal object ("spring") can be identified, so the syntactic rule for temporarily filling the syntactic subject identifies the local subject. Eventually this kind of reference is disambiguated by context, so the syntactic analysis provides placeholders

for the role fillers until more information becomes available .

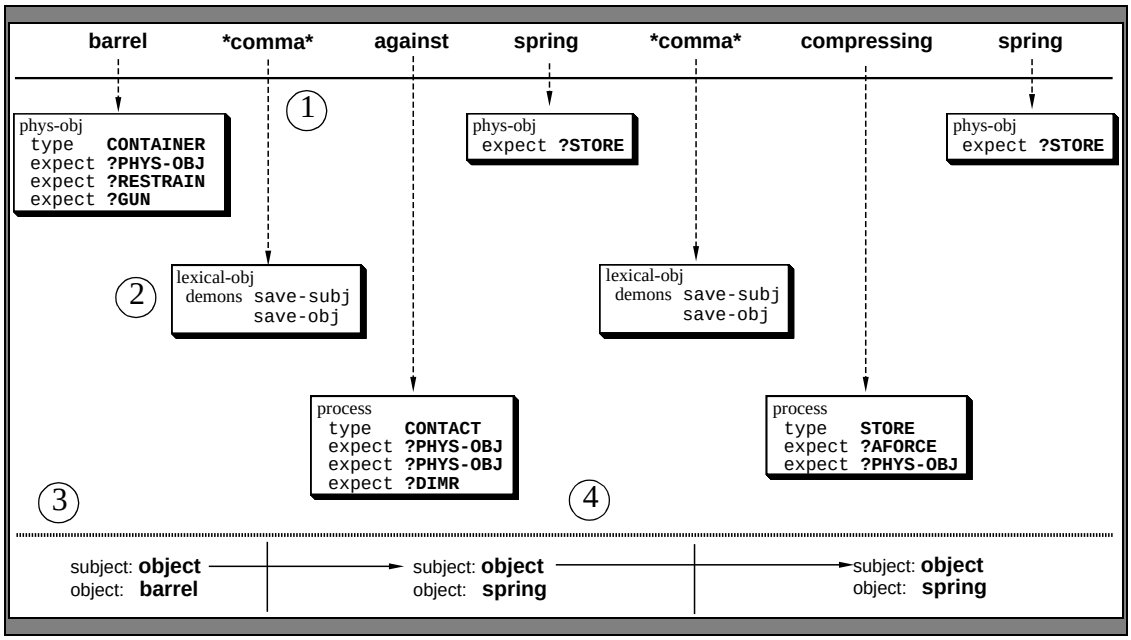


Figure 7.5 Syntactic rules and object reference disambiguation.

Semantic Object Disambiguation

The second phrase of **Toy Gun 2-3** can be understood from the sentence context, even though there is no explicit reference to the phrasal subject, because the physical relationships between the objects and their behavior provide semantic cues for disambiguating the phrasal subject. To understand the causal relationship which gives rise to "spring" compression, the "object" must possess some force that it can transmit to the "spring" in order to be instrumental in compressing the "spring." This knowledge is obtained as a result of knowing the enablement conditions for compression, and knowing what has already occurred in previous phrases. Since the STORE event is already realized, the STORE process enablements (applied force and contact) can be used to find the needed expectation ("object") using **Rule OR2**, thus disambiguating the part reference in "compressing."

Rule OR2:

IF the state resulting from an action (process) is recognized

THEN search memory for the corresponding satisfied enablements to the action (process) and index them into the episode.

This interaction is depicted in the parse diagram shown as Fig. 7.6. In this example, the thick horizontal dotted line distinguishes processing at the phrasal level (i.e., access to WM

only) and retrieval from SLM.

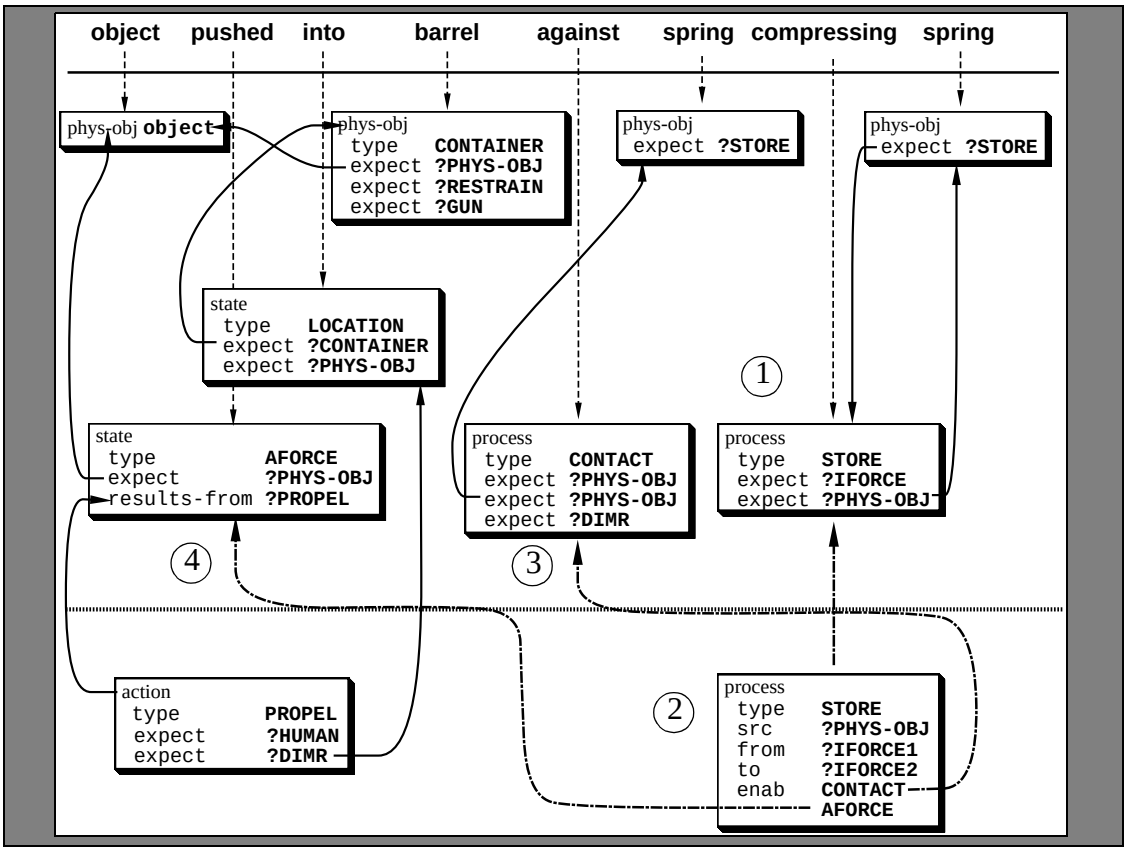


Figure 7.6 Role instantiation as a result of causal chain completion.

The STORE process which represents the lexical entry associated with "compressing" (1) is associated with the more general conceptual STORE schema in SLM (2). When "compressing" is parsed, a comparison to this schema is made possible. The dotted, arrowed lines emanating from the CONTACT and AFORCE process enablements, enable the filling of roles in STORE by finding the associated concepts in working memory (from 3 and 4) and mapping their role instantiations back to the current concept.

Anaphoric Object Disambiguation

The use of explicit pronoun references such as "it" as a reference to objects or processes poses additional problems to disambiguation. This is in part due to the lack of gender cues (such as he, hers, their) which aid in resolving pronouns. The remainder of the problem is that text describing objects and object interactions often references many parts. A reference to "it" may refer to any object or process seen in a discussion of objects and processes. The fourth phrase of **Toy Gun** (shown in **Toy Gun 4**) exemplifies this reference problem:

Toy Gun 4

"until it catches on a trigger"

In this phrase, "it" follows phrases which describe three parts: an "object," a "barrel," and a "spring." The object focus in EDCA combines the syntactic and semantic references as text is processed. However, there is no way to tell whether the part, that the RESTRAIN process associated with "catches" is expecting, is the subject or object of previous phrases. The "barrel" can be taken off the list of potential objects based on the syntactic subject and object bindings, but both the "object" and the "spring" must still be considered. EDCA currently has no mechanism for disambiguating this reference problem, so one of the objects is chosen with the anticipation that, downstream in the parse, semantic cues will help disambiguate the term.

2. **Multiple Object Reference:** Once again referring to **Toy Gun 2-3**, the word "spring" is introduced in the first phrase and reintroduced in the second. The reader has no difficulty understanding that both "spring" words refer to the same part. **Rule OR3** searches backwards for previous mention of objects and re-indexes current references to the original part.

Rule OR3:

IF this concept is mentioned previously in the text

THEN set the meaning of this node to that of the concept found

This methodology works as long as the words all refer to the same object.

Textually-Related Causal Disambiguation

Causal relationships are very prevalent in physical process descriptions. As a result, most every word is used to describe or infer aspects of the causality between parts and processes leading to a natural language conceptualization. One way in which causal ambiguity is resolved is through the use of functional determinants, called *causal keywords*. The first phrase of the second sentence in **Toy Gun** illustrates the use of "when" as a functional determinant:

Toy Gun S2-1

"When the trigger is pulled the spring is released..."

"When" is generally considered a conjunction. In the context of **Toy Gun S2-1,** "when" is a causal keyword implying that "trigger" pulling instrumentally causes "spring" release. Causal relationships are determined during lexical analysis by the use of words such as "when," "by," and "until" which directly imply functional causality. Three causal keyword groups have been identified in the texts analyzed by EDCA: (1) temporal enablement, (2) temporal disablement, and (3) instrumental support. "When" is represented as a temporal enablement conjunction, meaning that the second process (e.g., spring release in the example above) cannot occur until the first process has completed. The use of "until" in the fourth phrase of **Toy Gun** is an example of temporal disablement (see Fig. 7.7), while the

use of "by" in the following phrase from **Press** illustrates instrumental support.

Press

"Presses provide a means of compressing and shaping components *BY* exerting high pressure on them."

The state resulting from force transmission ("exerting high pressure") in **Press** is the instrumental support. The causal relationship is recognized from event states of actions or processes. Causal keywords set up an expectation for a antecedent/consequent relationship of which there are two kinds: (1) before, and (2) between. The designation *before* or *between* is based on the relative placement of the keyword with respect to the surrounding events. When a keyword is found before two event states, the causal direction is forward (processA causes processB). If the keyword is *between* the event states the causal direction is backward (processA caused-by processB). For example, in the phrase "The can is opened *after* being punctured." the causal is between the two physical processes and the inference is that "being punctured" precedes "opening can" temporally. In the sentence "*After* the can is punctured it can be opened." the causal is before the two physical processes and the inference is, again, that "punctured" precedes "opening." Although the two sentences chosen have the same meaning, the position and type of causal are a significant aid in interpreting the relationship between the physical events. In the case of process associations, the resulting state of the causing process can then be indexed to an enablement of the other.

In the **Toy Gun** example shown in Fig. 7.7, the causal disablement is between the two process events. This means that the causal direction is backward: the latter (catching) event is recognized as the causal antecedent, while the former (compressing) event is recognized as the causal consequent. Thus a conceptualization can be built indexing the two events.

The process of "catching" disables "spring compression" (Fig. 7.7).

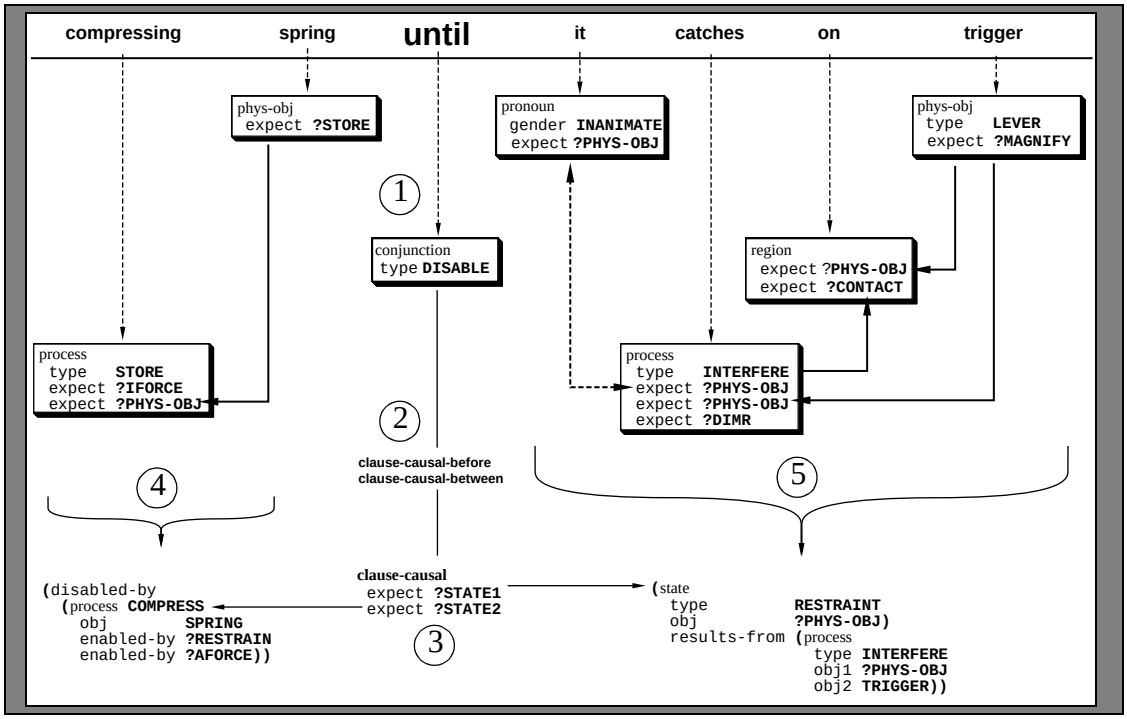


Figure 7.7 Causal enablement in **Toy Gun.**

Fig. 7.7 represents an example of textually-related causal disambiguation. In "compressing the spring until it catches on the trigger," the causal keyword is the conjunction "until," the processing of which creates expectations for the types of causal relationship between phrases previously described (before and between, enablement and disablement, at 1 and 2). As the text is analyzed, the phrasal events become known, and the causal expectations (antecedent and consequent, 3) are realized for the *between* case, resulting in a causal conceptualization (processB disables processA). The first phrase (processA) is recognized as the event resulting from the "compressing the spring" process. The second phrase (processB) is recognized as the event resulting from the "catches on the trigger" process. Some of the lexical entries and associated expectations have been left out of the figure.

WM Refinement via Expectation Failures

Instantiating a device in EPM requires an analysis of what is known about the device in question with respect to what is known about devices in general. Textual descriptions can describe devices which are slightly different, or broken, with respect to known devices in memory. Comparison of the known device to device schema in SLM helps to define the differences, which aids in their interpretation. The analysis associated with these comparisons leads to the reintegration of a new device into memory, and to rules describing device faults, both resulting in explanations.

Constructing Episodic Device Memory

The end product of reading a device description is a completely instantiated device episode. The episode reflects understanding gained through reading the description, comparing it to

known knowledge sources and reasoning about the two, by making connections between the related knowledge structures. In order to do this two things must happen: (1) the objects and processes must be indexed in the new episode, and (2) device functions must be constructed. Constructing device functions and reintegrating the device into SLM are learning tasks which, along with the actual organization of memory used in EDCA, have not been fully addressed in this dissertation.

7.4 EDCA Implementation

The basis for EDCA is the McDypar demon-based parser, which was used extensively at the UCLA Artificial Intelligence lab until 1988. The first version of EDCA was written in T [Reese, 1984], a scheme-based Lisp dialect developed at Yale. The model was implemented on top of an AI development tool called RHAPSODY [Turner and Reeves, 1987] written at UCLA. RHAPSODY provided representational semantics, the McDypar parser, a generator, demon and agenda packages, and graphics facilities. The EDCA model uses approximately 150 demons to analyze the **Toy Gun** and **Press** device descriptions illustrated in this chapter as well as a few others.

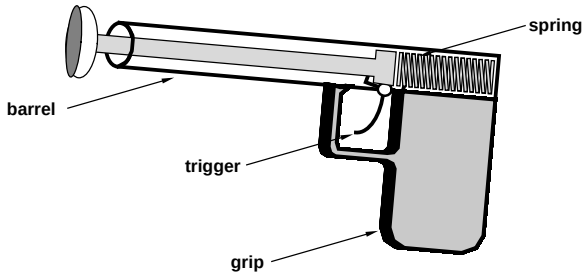
7.4.1 A Detailed Example: Toy Gun

The **Toy Gun** example has been used extensively to demonstrate various issues in device description comprehension and how they are addressed in EDCA from a conceptual standpoint. The processing of **Toy Gun** is described here in greater detail. The first sentence in Fig. 7.8a represents the *arming* function for a 'typical' toy gun, such as the one depicted in Fig. 7.8b. This example has been selected to illustrate the issues encountered while processing and understanding physical device descriptions. EDCA has demons used in lexical analysis as well as for reasoning about partially instantiated conceptual structures. The processing of these somewhat different demon types is accomplished with two agendas, one for the textually-specific demons (agenda.4 in the trace), and one for the reasoning and domain-specific demons (agenda.7 in the trace). Agenda.4 is run after every word is processed, while agenda.7 is run at clause boundaries. A large amount of processing is done in just the first sentence of this example. As a result, a number of demons have been left out of the trace, as well as redundant processing (throughout) once it has been reviewed once.

After EDCA is loaded into memory the following text is processed:

(AN OBJECT IS PUSHED INTO A BARREL *COMMA* AGAINST A SPRING *COMMA* COMPRESSING THE SPRING UNTIL IT CATCHES ON A TRIGGER *PERIOD*)

(a)



(b)

Figure 7.8 (a) EDCA input for first sentence of **Toy Gun** text, (b) toy gun illustration.

```
---------------------------------
    Processing word: AN
---------------------------------

   Spawning demon: (IGNOR.1 #{WMN.1})
RUNNING AGENDA: #{AGENDA:PARSE-AGENDA.4}
RUNNING DEMON: IGNOR.1
   Executing +act: (IGNOR.1)
   Killing demon: IGNOR.1
```

Articles are ignored by the conceptual analyzer

```
---------------------------------
    Processing word: OBJECT
---------------------------------

Adding to *wm*: #{WMN.2}
Created concept:
(PHYSOBJ &PHYSOBJ.19
 NAME OBJECT)

---------------------------------
    Processing word: IS
---------------------------------

   Spawning demon: (FIND-EVENT.1 #{WMN.3})
```

OBJECT

```
FIND-EVENT.1
=========================================================
TEST:   If the OBJECT or PREPOSITIONAL PHRASE (for one
        of the given preps) can be interpreted as an
        EVENT in a higher level memory structure event
ACT:    Let that event be the event for the sentence
=========================================================

Spawning demon: (TEST-OBJECT.1 #{WMN.3})

TEST-OBJECT.1
=========================================================
TEST:   Test the OBJECT of a verb to see if it is one
        of the given CLASSES
ACT:    Set the meaning for the node to the given
        meaning and execute the given procedure
=========================================================
```

*The two meanings of ``is'' represented are as a reference to an action or process (and particularly to states of same), and as a object reference. In this text the ``is'' refers to the action (**PROPEL**), and the resulting containment of the ``object'' in the ``barrel'' which is caused by the ``push'' of the ``object''.*

```
--------------------------------
    Processing word: PUSHED
--------------------------------

Adding to *wm*: #{WMN.4}
Created concept:

(ACTION &ACTION.1
 TYPE &ACTION.13   <------------------------------------  PROPEL
 NAME PUSHED
 OBJ ?PHYSOBJ)

   Spawning demon: (NEW-EVENT.1 #{ACTION.1})

   NEW-EVENT.1
   ===========================================================
   TEST:   A new event or state is parsed
   ACT:    Run the rule-agenda to see what domain knowledge
           has to say.
   ===========================================================

   Spawning demon: (EXPECT.1 #{WMN.4} #{ACTION.1} (OBJ) PHYSOBJ BEF)

   EXPECT.1
   =========================================================
   TEST:   A CONCEPT with one of the given CLASSES is
           found when searching in the given direction
   ACT:    The CONCEPT is bound to the given GAP
   =========================================================
```

```
RUNNING AGENDA: #{AGENDA:PARSE-AGENDA.4}

RUNNING DEMON: EXPECT.1
   Executing +act: (EXPECT.1)
  Slot "(PART)" in #{ACTION.1} <-- #{PHYSOBJ.19}
   Killing demon: EXPECT.1

RUNNING DEMON: NEW-EVENT.1
   Executing +act: (NEW-EVENT.1)

Adding to *wm*: #{WMN.5}
Created concept:
```

PUSHED *is assumed to be an object state in which* ***PROPEL*** *action has occurred and the associated goals for* ***PROPEL*** *can be inferred.* ***NEW-EVENT*** *integrates domain-specific (DS) knowledge with the lexical analysis by spawning DS demons which are later analyzed.* ***EXPECT.1*** *locates the object of the* ***PROPEL*** *in* ***WM****.*

```
--------------------------------
    Processing word: INTO
--------------------------------

Adding to *wm*: #{WMN.6}
Created concept:

(PROCESS CONTACT.3
 TYPE         RESTRAIN
 SRC          ?PHYSOBJ1
 DST          ?PHYSOBJ2
 DIMR         ?DIM
 REALIZED-EVS (&EVENT.1 &EVENT.2))

   Spawning demon: (EXPECT.2 #{WMN.6} #{CONTACT.3} (SRC) PHYSOBJ BEF)

   EXPECT.2
   ======================================================
   TEST:   A CONCEPT with one of the given CLASSES is
           found when searching in the given direction
   ACT:    The CONCEPT is bound to the given GAP
   ======================================================

   Spawning demon: (EXPECT.3 #{WMN.6} #{CONTACT.3} (DST) PHYSOBJ AFT)

   EXPECT.3
   ======================================================
   TEST:   A CONCEPT with one of the given CLASSES is
           found when searching in the given direction
   ACT:    The CONCEPT is bound to the given GAP
   ======================================================

   Spawning demon: (EXPECT.4 #{WMN.6} #{CONTACT.3} (DIMR) DIM AFT)

   EXPECT.4
```

```
   ========================================================
   TEST:   A CONCEPT with one of the given CLASSES is
           found when searching in the given direction
   ACT:    The CONCEPT is bound to the given GAP
   ========================================================

RUNNING DEMON: EXPECT.2
   Executing +act: (EXPECT.2)
  Slot "(SRC)" in #{CONTACT.3} <-- #{PHYSOBJ.19}
   Killing demon: EXPECT.2

RUNNING DEMON: FIND-EVENT.1
   Executing +act: (FIND-EVENT.1)
   Killing demon: FIND-EVENT.1
```

*One meaning of **INTO** is containment, which is a type of connection for object groupings. Expectations are spawned for the associated parts as well as the dimension of connection. Since this process is within the scope of ``is'' the restraint and orientation events are taken as realized by **FIND-EVENT.1***

```
---------------------------------
    Processing word: BARREL
---------------------------------

Adding to *wm*: #{WMN.7}
Created concept:

(PHYSOBJ &PHYSOBJ.20
 TYPE     &PHYSOBJ.21      <------------------------------------ TUBE
 NAME     BARREL
 PHYS-ATS &PHYS-ATS.5)
RUNNING DEMON: EXPECT.3
   Executing +act: (EXPECT.3)
  Slot "(DST)" in #{CONTACT.3} <-- #{PHYSOBJ.20}
   Killing demon: EXPECT.3
```

*An instance of &tube is placed onto **WM**. **BARREL** has a functional region HOLE which is oriented along the barrel length. A DS demon, **PHYSOBJ:BARREL-DIM**, for regions matches the need for a dimension with the dimensional characteristic of the **BARREL** hole.*

```
---------------------------------
    Processing word: *COMMA*
---------------------------------

Adding to *wm*: #{WMN.8}
Created concept:

(LEXICAL-ITEM &LEXICAL-ITEM.7
 TYPE COMMA)

   Spawning demon: (SAVE-SO.1 #{WMN.8})

   SAVE-SO.1
```

```
   ======================================================
   TEST:   There is no process for a clause.
   ACT:    The earliest object is the subject, and
           the most recent object is the object.
   ======================================================

   Spawning demon: (SAVE-EV.1 #{WMN.8})

   SAVE-EV.1
   ======================================================
   TEST:   There is a local process for the clause,
           and the clause is not a relative clause.
   ACT:    The process is the local process.  Otherwise
           we forward the existing local process.
   ======================================================

   Spawning demon: (PROCESS-CLAUSE.1 #{WMN.8})

   PROCESS-CLAUSE.1
   ======================================================
   TEST:   When a SENTENCE or CLAUSE BOUNDRY has been
           reached...
   ACT:    Fire demons to EXPLAIN the concepts that
           have been recognized
   ======================================================
RUNNING DEMON: SAVE-SUBJ1.1
   Executing +act: (SAVE-SUBJ1.1)
[Binding *MOST-RECENT-SUB*]
   Killing demon: SAVE-SUBJ1.1

RUNNING DEMON: SAVE-EV.1
   Executing +act: (SAVE-EV.1)
[Binding *MOST-RECENT-PROC*]
   Killing demon: SAVE-EV.1

RUNNING DEMON: PROCESS-CLAUSE.1
   Executing +act: (PROCESS-CLAUSE.1)
```

*The demons **SAVE-SO**, **SAVE-SUB1**, **SAVE-SUB2**, **SAVE-SUB3**, **SAVE-OBJ1**, **SAVE-OBJ2**, **SAVE-OBJ3** (not shown), and **SAVE-EV** are reference placeholders for objects in different contexts. In this case **SAVE-SO**, **SAVE-SUBJ.1**, and **SAVE-EVENT** retain ordinal knowledge for future reference problems.*

*When a motion of an unspecified device is recognized a demon, **EP:PROC**, is spawned to create a new device instance. Later the validity of this assumption is checked. **OBJECT**, **BARREL** and the forced-state of **OBJECT** are placed onto EPM for future device reference. Later these constructs are incorporated into the new device representation.*

```
Adding to *ep*: #{EPN.1}
Put concept in EPISODIC MEMORY:
```

```
(PROCESS &PROCESS.3
 REALIZED-EVS (&$MOTION.EVENT1)
 SRC          &PHYSOBJ.19)   <-------------------------  OBJECT

   Spawning demon: (EP:PROC.1 #{PROCESS.3})

   EP:PROC.1
   ======================================================
   TEST:   A mechanical process is parsed.
   ACT:    Create a new d-mop in episodic memory.
   ======================================================

Adding to *ep*: #{EPN.2}
Put concept in EPISODIC MEMORY:

(STATE &STATE.12
 TYPE &STATE.13
 NAME PUSHED
 PART &PHYSOBJ.19   <-------------------------  OBJECT
 PROP FORCE
 VAL  &QUANT.13)

   Spawning demon: (PART:BARREL-DIM.1 #{PHYSOBJ.20})

   PART:BARREL-DIM.1
   ======================================================
   TEST:   A BARREL part is recognized.
   ACT:    See if HOLE attributes can be used in memory.
   ======================================================

   Spawning demon: (PART:SCRIPT-REC.1 #{PHYSOBJ.20})

   PART:SCRIPT-REC.1
   ======================================================
   TEST:   A part is placed into episodic memory.
   ACT:    Check partially instantiated scripts/mops
           for activation.
   ======================================================

   Spawning demon: (EP:PART.1 #{PHYSOBJ.20})

   EP:PART.1
   ======================================================
   TEST:   A part is placed into episodic memory.
   ACT:    Make it a part of the current device.
   ======================================================

   Spawning demon: (PART:LINKAGE-RECOG.1 #{PHYSOBJ.19})

   PART:LINKAGE-RECOG.1
   ======================================================
        IF the part has rod or slab ptype
        THEN the part is a linkage
   ======================================================
```

```
RUNNING AGENDA: #{AGENDA:RULE-AGENDA.7}

RUNNING DEMON: PART:BARREL-DIM.1
   Executing +act: (PART:BARREL-DIM.1)
Adding to *ep*: #{EPN.3}
Put concept in EPISODIC MEMORY:

RUNNING DEMON: EP:PROC.1
   Executing +act: (EP:PROC.1)
Adding to *ep*: #{EPN.3}
Put concept in EPISODIC MEMORY:

(M-DEVICE &NULL-DEVICE
 DEVICE &DEVICE.1
 ACTOR  EDISON)

   Killing demon: EP:PROC.1
```

When a sentence or clause end is reached, DS demons are also fired to incorporate recognized knowledge into the current episodic representation for the unknown device. In this first phrase DS demons ***EP:PART.1****,* ***EP:PROC****,* ***PART:BARREL-DIM.1****, and* ***PART:LINKAGE-RECOG.1*** *are spawned (among others) to interpret what* ***OBJECT*** *is, and to aid in indexing the dimension from* ***BARREL*** *with* ***$MOTION.1****.*

PART:SCRIPT-REC.1 *is an example of one of many DS demons which are spawned onto the* ***RULE:AGENDA*** *when parts are placed onto* ***WM****. This demon looks for the processes and devices in which the part is a role filler and checks to see whether they are instantiated enough in* ***WM*** *to activate.* ***EP:PROC.1*** *is a demon which looks for new mentions of devices*

and creates dummy's for interpretive purposes.

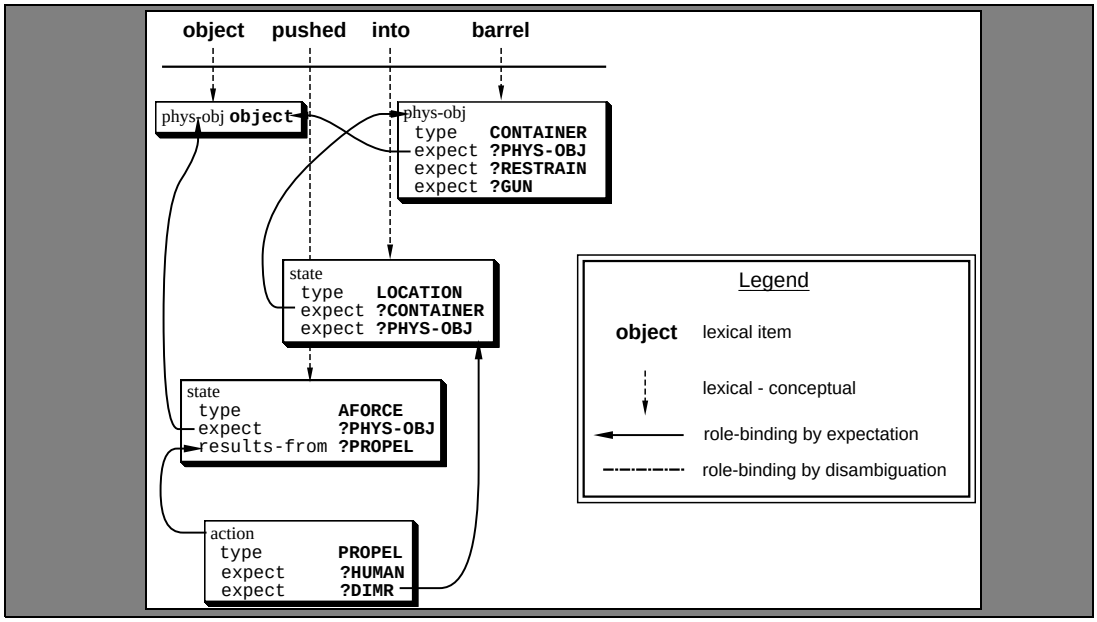


Figure 7.9 Parse diagram for the first phrase in **Toy Gun**, sentence 1**.**

```
---------------------------------
    Processing word: AGAINST
---------------------------------

Adding to *wm*: #{WMN.9}
Created concept:

(PROCESS &PROCESS.4
 NAME         AGAINST
 SRC          ?PHYSOBJ1
 DST          ?PHYSOBJ2
 REALIZED-EVS (&EVENT.5))

   Spawning demon: (EXPECT.2 #{WMN.9} #{PROCESS.4} (DST) PHYSOBJ AFT)

   EXPECT.2
   ========================================================
   TEST:   A CONCEPT with one of the given CLASSES is
           found when searching in the given direction
   ACT:    The CONCEPT is bound to the given GAP
   ========================================================

   Spawning demon: (EXPECT.3 #{WMN.9} #{PROCESS.4} (SRC) PHYSOBJ BEF)

   EXPECT.3
   ========================================================
   TEST:   A CONCEPT with one of the given CLASSES is
           found when searching in the given direction
```

```
   ACT:    The CONCEPT is bound to the given GAP
   ========================================================
```

*The **EXPECT** demons are spawned to locate the roles in the connection instance. The connection **AGAINST** satisfies the restraint expectation from the first clause. The realized event in **AGAINST** helps to satisfy the orientation between **OBJECT** and **BARREL**, as well as to fill an inference that the spring is also in the **BARREL** (orientation is transitive, i.e. if A is in C, and A is connected to B, then B is in C).*

```
---------------------------------
    Processing word: SPRING
---------------------------------

Adding to *wm*: #{WMN.11}
Created concept:

(PHYSOBJ &PHYSOBJ.22
 TYPE &SPRING
 NAME SPRING)

RUNNING DEMON: EXPECT.2
   Executing +act: (EXPECT.2)
  Slot "(DST)" in #{PROCESS.4} <-- #{PHYSOBJ.22}
   Killing demon: EXPECT.2
---------------------------------
    Processing word: *COMMA*
---------------------------------

   Spawning demon: (SAVE-SUBJ3.2 #{WMN.12})

   SAVE-SUBJ3.2
   ========================================================================
   TEST:   A subject of a process doesn't exist for the current phrase,
           and there is a preceding phrase which has a subject.
   ACT:    Remember the subject from the previous phrase as the most
           recent subject.
   ========================================================================

RUNNING AGENDA: #{AGENDA:PARSE-AGENDA.4}

RUNNING DEMON: SAVE-SUBJ3.2
   Executing +act: (SAVE-SUBJ3.2)
   Killing demon: SAVE-SUBJ3.2

RUNNING DEMON: SAVE-OBJ1.2
   Executing +act: (SAVE-OBJ1.2)
   Killing demon: SAVE-OBJ1.2

Adding to *ep*: #{EPN.4}
Put concept in EPISODIC MEMORY:
```

```
(PROCESS &PROCESS.4
 NAME          AGAINST
 SRC           &PHYSOBJ.19  <------------  OBJECT
 DST           &PHYSOBJ.22  <------------  SPRING
 REALIZED-EVS (&EVENT.5))
```

*In this phrase the use of multiple reference resolution techniques shows itself. In this case a process and local referents exist from the previous clause, as well as not existing (local referents) in the present clause. The result is that the previous subject is brought forward into the present one. However, since **AGAINST** was expected from the previous clause and instantiated in the present one, the subject of the previous clause is brought forward via context anyway. To reiterate a prior point, we have intentionally left out a number of processing items for the sake of brevity.*

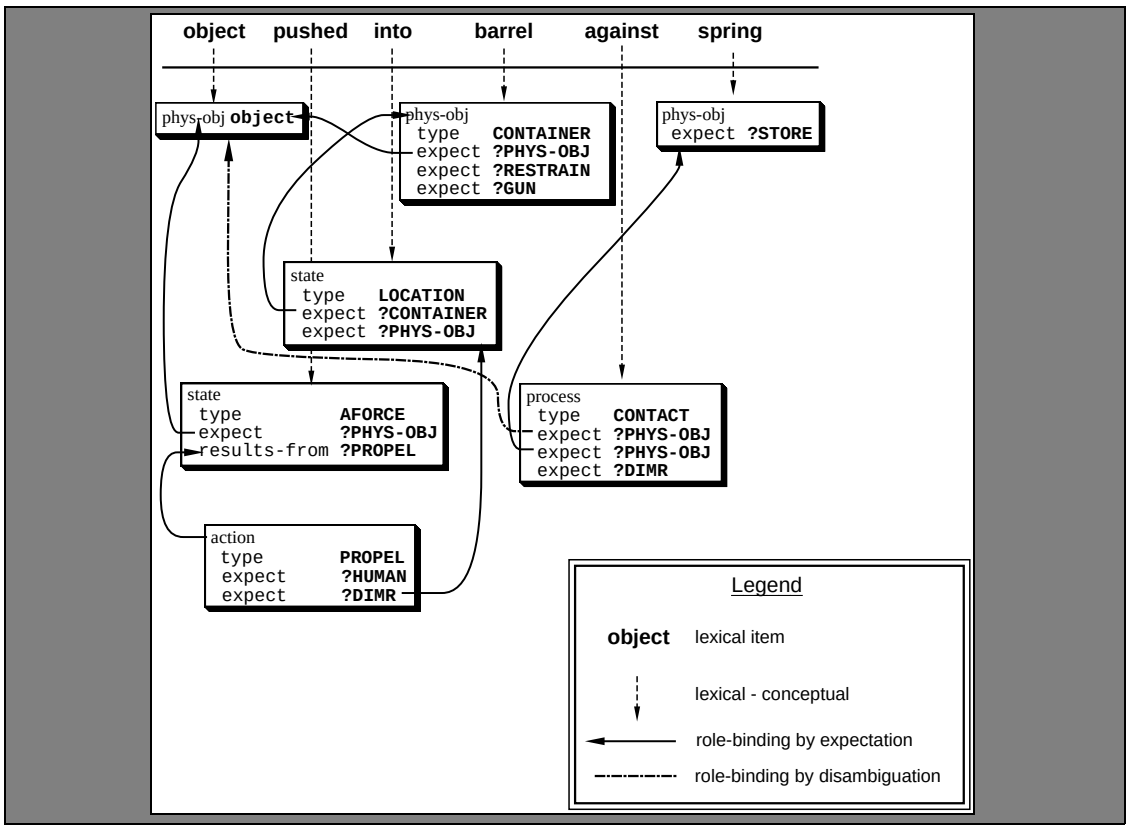


Figure 7.10 Parse diagram for the first two phrases of **Toy Gun**, sentence 1.

```
--------------------------------
    Processing word: COMPRESSING
--------------------------------

Adding to *wm*: #{WMN.13}
Created concept:

(PROCESS &PROCESS.5
 NAME         COMPRESSING
 SRC         ?PHYSOBJ
 REALIZED-EVS (&EVENT.6 &EVENT.7))

   Spawning demon: (EXPECT.5 #{WMN.13} #{PROCESS.5} (SRC) PHYSOBJ AFT)

   EXPECT.5
   ========================================================
   TEST:   A CONCEPT with one of the given CLASSES is
           found when searching in the given direction
   ACT:    The CONCEPT is bound to the given GAP
   ========================================================

--------------------------------
    Processing word: SPRING
--------------------------------

Adding to *wm*: #{WMN.15}
Created concept:

(PHYSOBJ &PHYSOBJ.23
 TYPE &SPRING
 NAME SPRING)

   Spawning demon: (SEARCH-FOR-REF.4 #{WMN.15})

   SEARCH-FOR-REF.4
   ========================================================
   TEST:   Search for a referent for this concept
   ACT:    Set the meaning for the node to the
           concept found
   ========================================================

RUNNING DEMON: EXPECT.5
   Executing +act: (EXPECT.5)
  Slot "(SRC)" in #{PROCESS.5} <-- #{PHYSOBJ.23}
   Killing demon: EXPECT.5

RUNNING DEMON: SEARCH-FOR-REF.4
   Executing +act: (SEARCH-FOR-REF.4)
   Spawning demon: (REPLACE-TO-BNDRY.1 #{WMN.15} #{PHYSOBJ.22})

   REPLACE-TO-BNDRY.1
   ============================================================
   TEST:   A concept is a forward reference to a previous concept
   ACT:    Replace all concepts which referred to the existing
```

```
          concept prior to its redesignation
   ================================================================

RUNNING DEMON: REPLACE-TO-BNDRY.1
   Executing +act: (REPLACE-TO-BNDRY.1)
   Killing demon: REPLACE-TO-BNDRY.1
```

*When the word **COMPRESSING** is read, a **STORE** process is instantiated when it finds the object **SPRING**. At the same time, **SPRING** has been mentioned before. **SEARCH-FOR-REF** is spawned to find and assimilate previous instances of spring (or any part) in ***WM***. Having found the previous instance the demon reindexes ***WM*** so that the second instance doesn't point to anything. This part is later removed altogether.*

*Likewise there is also a reference problem in this clause. The **STORE** process has no subject but cannot be initiated without some kind of contact and transmission. Because the **STORE** event is recognized, the enablement conditions must be met. A DS demon, **EP:PROC-PRECOND2**, is needed to survey the enablements to **STORE** and confirm. Since the **OBJECT** possesses force, is in motion, and is in contact with the **SPRING**, all of the **STORE** enablements are met. As a result **OBJECT** must be the cause of the compres-*

sion.

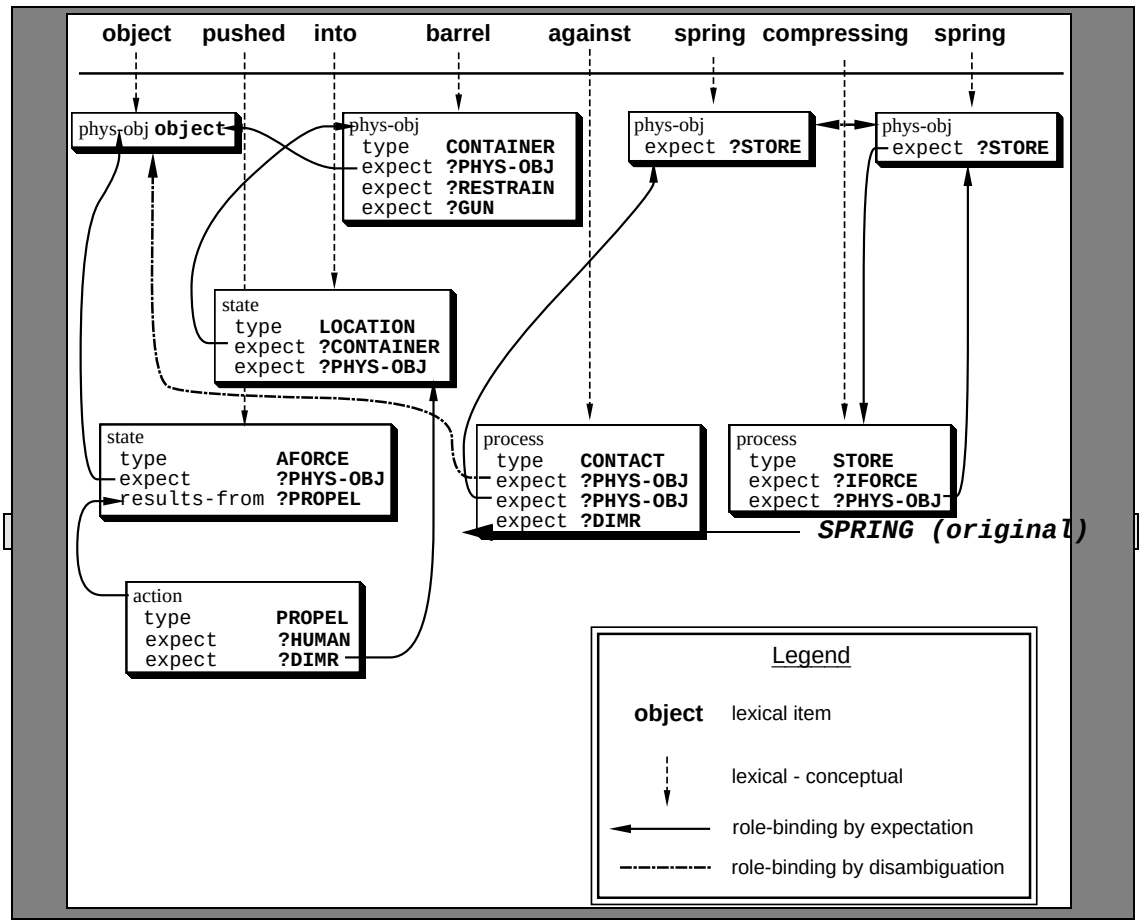


Figure 7.11 Parse diagram for the first three phrases of **Toy Gun**, sentence 1.

```
--------------------------------
    Processing word: UNTIL
--------------------------------

Adding to *wm*: #{WMN.16}
Created concept:

(PREPOBJ &PREPOBJ.8
 TYPE CAUSAL
 IS   UNTIL)

RUNNING DEMON: PROCESS-CLAUSE.3
   Executing +act: (PROCESS-CLAUSE.3)

Adding to *ep*: #{EPN.5}
Put concept in EPISODIC MEMORY:

(PROCESS &PROCESS.5
 NAME         COMPRESSING
 SRC          &PHYSOBJ.22
 REALIZED-EVS (&EVENT.6 &EVENT.7))

   Spawning demon: (CLAUSE:CAUSAL2-BET.1 #{WMN.16})

   CLAUSE:CAUSAL2-BEF.1
   ============================================================
   TEST:   A causal disablement keyword is found while parsing.
   ACT:    Search for an event preceding, an event
           following, and spawn demons to infer the
           relationship between the respective clauses.
   ============================================================
```

UNTIL *signals the potential for an* ***antecedent/consequent*** *relationship. Because these usually occur between high level knowledge structures they are preferred over process interpretation (and get some preference as a result).* ***CLAUSE:CAUSAL-BET*** *and* ***CLAUSE:CAUSAL-BEF*** *(not shown) are spawned from the functional-keyword meaning of* ***UNTIL****.*

```
(LEXICAL-ITEM &LEXICAL-ITEM.9
--------------------------------
    Processing word: IT
--------------------------------

Adding to *wm*: #{WMN.17}
Created concept:

(LEXICAL-ITEM &LEXICAL-ITEM.9
 TYPE PRON
 IS   IT)

   Spawning demon: (FIND-REF.1 #{WMN.17})
```

```
   FIND-REF.1
   ==================================================
   TEST:   OUR-NODE occurs late in a clause.
   ACT:    Set OUR-NODE's concept to the most
           recently mentioned object, otherwise to
           to the most recently mentioned subject.
   ==================================================

---------------------------------
    Processing word: CATCHES
---------------------------------

Adding to *wm*: #{WMN.18}
Created concept:

(PROCESS &PROCESS.5  <------------------------  simple lever mechanism
 NAME         CATCHES
 SRC          ?PHYSOBJ1
 DST          ?PHYSOBJ2
 REALIZED-EVS (&EVENT.8))

RUNNING DEMON: EXPECT.7
   Executing +act: (EXPECT.7)
  Slot "(SRC)" in #{PROCESS.5} <-- #{PHYSOBJ.19}
   Killing demon: EXPECT.7

---------------------------------
    Processing word: ON
---------------------------------

Adding to *wm*: #{WMN.19}
Created concept:

(ORIENT &ORIENT.5
 NAME  ON
 TYPE  ()
 PART1 ?PHYSOBJ1
 PART2 ?PHYSOBJ2)

   Spawning demon: (EXPECT.8 #{WMN.19} #{ORIENT.5} (PART2) PHYSOBJ AFT)

   EXPECT.8
   ======================================================
   TEST:   A CONCEPT with one of the given CLASSES is
           found when searching in the given direction
   ACT:    The CONCEPT is bound to the given GAP
   ======================================================
                          <------------------------  OBJECT
   Spawning demon: (EXPECT.9 #{WMN.19} #{ORIENT.5} (PART1) PHYSOBJ BEF)

   EXPECT.9
   ======================================================
   TEST:   A CONCEPT with one of the given CLASSES is
           found when searching in the given direction
```

```
   ACT:    The CONCEPT is bound to the given GAP
   ========================================================

RUNNING DEMON: EXPECT.9
   Executing +act: (EXPECT.9)
  Slot "(PART1)" in #{ORIENT.5} <-- #{PHYSOBJ.19}
   Killing demon: EXPECT.9

---------------------------------
    Processing word: TRIGGER
---------------------------------

Adding to *wm*: #{WMN.21}    <------------------------  OBJECT
Created concept:             <------------------------  TRIGGER
(PART &PHYSOBJ.24
 TYPE &PHYSOBJ.25
 NAME TRIGGER)

RUNNING DEMON: EXPECT.8
   Executing +act: (EXPECT.8)
  Slot "(PART2)" in #{ORIENT.5} <-- #{PHYSOBJ.24}
   Killing demon: EXPECT.8   <------------------------  SPRING

RUNNING DEMON: EXPECT.6
   Executing +act: (EXPECT.6)
  Slot "(DST)" in #{PROCESS.5} <-- #{PHYSOBJ.24}
   Killing demon: EXPECT.6

---------------------------------
Processing word: *PERIOD*
---------------------------------

Adding to *wm*: #{WMN.22}
Created concept:

(LEXICAL-ITEM &LEXICAL-ITEM.10
 TYPE PERIOD)

RUNNING DEMON: PROCESS-CLAUSE.4
   Executing +act: (PROCESS-CLAUSE.4)

Adding to *ep*: #{EPN.6}
Put concept in EPISODIC MEMORY:

(ORIENT &ORIENT.5
 NAME  ON
 TYPE  ()
 PART1 &PHYSOBJ.19
 PART2 &PHYSOBJ.24)

   Spawning demon: (EP:ORIENT.1 #{ORIENT.5})
```

```
   EP:ORIENT.1
   ====================================================
   TEST:   An orient is parsed onto wm.
   ACT:    Place it in working memory (ep),
           and put it into the local device.
   ====================================================

Adding to *ep*: #{EPN.7}
Put concept in EPISODIC MEMORY:

(PROCESS &PROCESS.5
 NAME         CATCHES
 SRC          &PHYSOBJ.19
 DST          &PHYSOBJ.24
 REALIZED-EVS (&EVENT.8))

Adding to *ep*: #{EPN.8}
Put concept in EPISODIC MEMORY:

(PROCESS &PROCESS.6
 NAME         COMPRESSING
 SRC         &PHYSOBJ.22
 REALIZED-EVS (&EVENT.6 &EVENT.7))
```

*There are two causal demons, a 'before' and a 'between', monitoring *wm* for process event sequences. In this example we show only the 'between' because that is the one that gets instantiated. The clausal finds the previous event (compressing the spring) right away, but keeps looking for the other.*

*The processing of **IT** occurs in parallel to the causal analysis. **FIND-REF.1** is spawned to look for referents to the word. What is known is that **IT** represents the subject of a connection process **CATCHES** (this is known because the local phrase is processed unsuccessfully first). Since both the **OBJECT** and **SPRING** are in motion it is difficult to resolve the problem. The **SPRING** is the object of compression in the last process read. However, the **OBJECT** is the subject of that phrase. In the present example the running object reference approach was used for the sake of a complete example. The other alternatives have not been fully addressed at the time of writing.*

*The remainder of the phrase is analyzed in much the same way as previous phrases and the **OBJECT**, **TRIGGER**, and **SPRING** are all indexed into **EPM** appropriately. At this point the second event sequence is available to the keyword search and processing continues.*

```
RUNNING DEMON: CLAUSE:CAUSAL2-BET.1
   Executing +act: (CLAUSE:CAUSAL2-BET.1)
   killing demon: CLAUSE:CAUSAL2-BET.1
```

*The +act of **CLAUSE:CAUSAL2-BET.1** is to check the resulting states of the antecedent process against the enablement states of the consequent process. In this example the resulting state of catching the object on the trigger is that the object is restrained from further motion and motion stops (processes watch for their disablement conditions). Since an en-*

ablement condition for compression is motion of the object, and since the restraint of the object transitively feeds through to the spring, the test is positive and the causal relationship can be constructed in EPM.

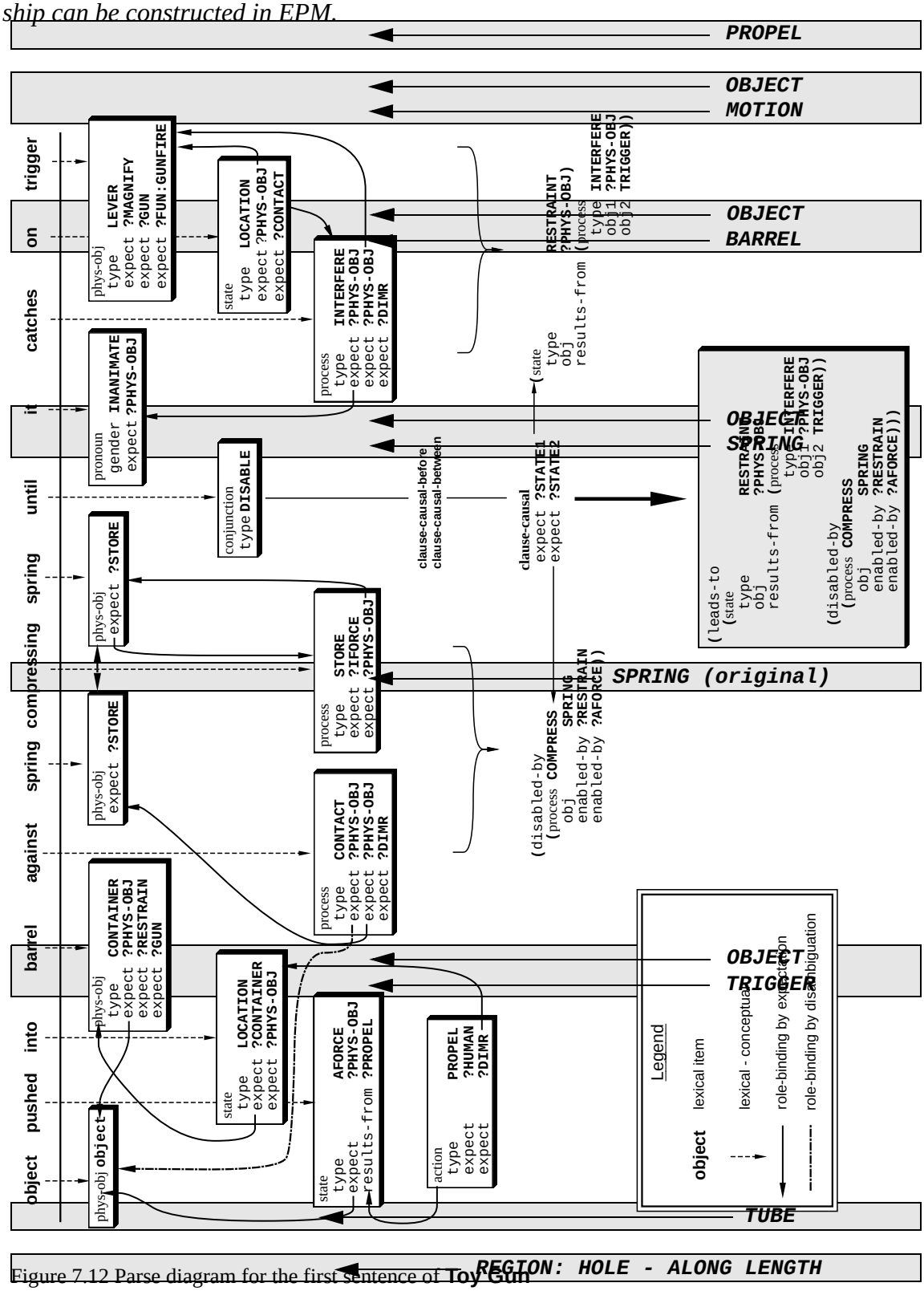


Figure 7.12 Parse diagram for the first sentence of **Toy Gun**

RESULT OF PARSE:

```
(ACTION &ACTION.1
 TYPE         &ACTION.13
 NAME         PUSHED
 OBJ          &PHYSOBJ.19 ◄──────────── simple lever mechanism
 REALIZED-STS (&STATE1))

(PROCESS &PROCESS.3
 NAME         CONTAIN
 SRC          &PHYSOBJ.19
 DST          &PHYSOBJ.20 ◄──────────── OBJECT
 REALIZED-EVS (&EVENT.4)) ◄──────────── TRIGGER

----------------------------

(PROCESS &PROCESS.4
 NAME         AGAINST
 SRC          &PHYSOBJ.19
 DST          &PHYSOBJ.22
 REALIZED-EVS (&EVENT.5))

----------------------------

SCENE1 of causal

(PROCESS &PROCESS.5
 NAME         COMPRESSING
 SRC          &PHYSOBJ.22
 REALIZED-EVS (&EVENT.6 &EVENT.7))

 (enabled-by process.4, action.1, and some unknown connection)

----------------------------

SCENE2 of causal disables process

(PROCESS &PROCESS.6
 NAME         CATCHES
 SRC          &PHYSOBJ.19
 DST          &PHYSOBJ.24
 REALIZED-EVS (&EVENT.8))

----------------------------

(PHYSOBJ &PHYSOBJ.19
 NAME OBJECT)

(PHYSOBJ &PHYSOBJ.20
 TYPE     &PHYSOBJ.21
 NAME     BARREL
 PHYS-ATS &PHYS-ATS.5)
```

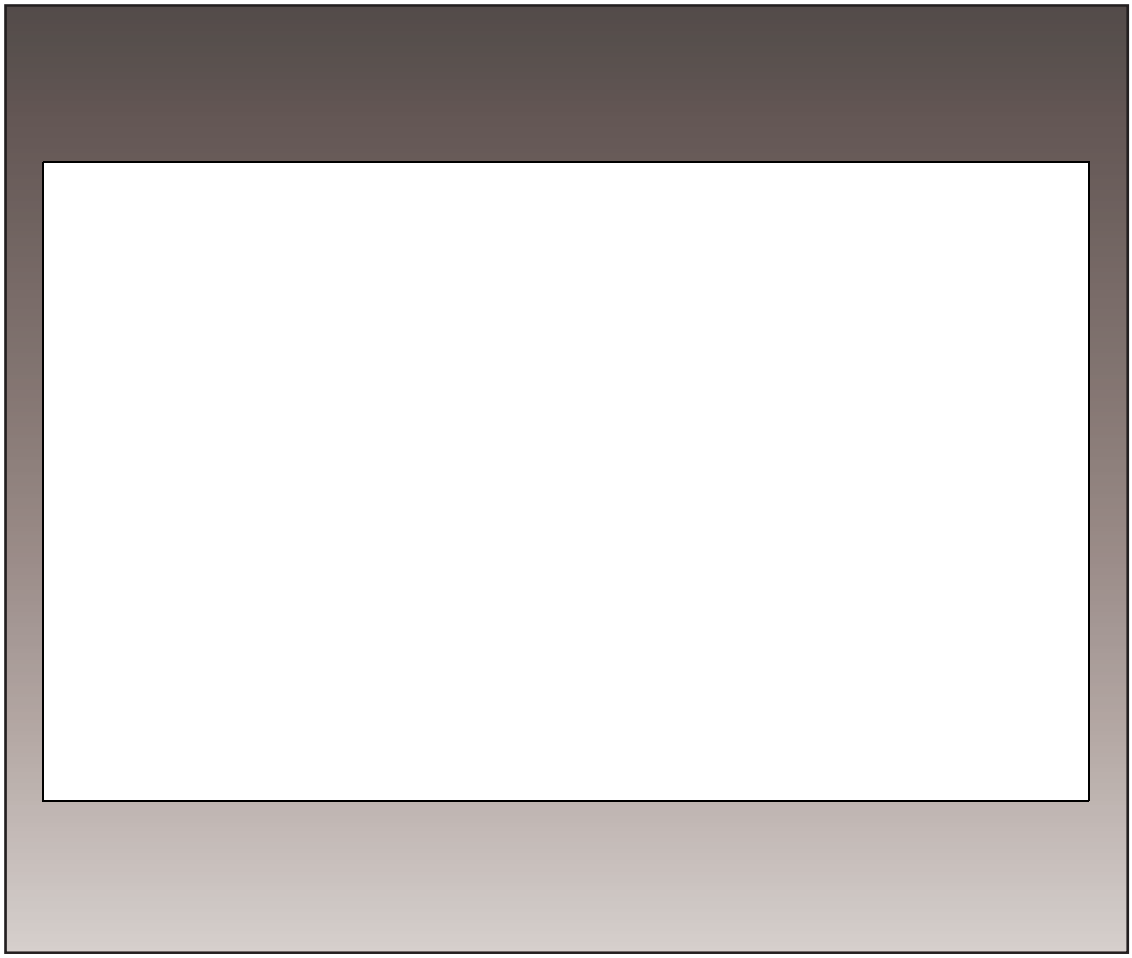
```
(PHYSOBJ &PHYSOBJ.22
 TYPE &SPRING
 NAME SPRING)

(PHYSOBJ &PHYSOBJ.24
 TYPE &PHYSOBJ.25
 NAME TRIGGER)

(ORIENT &ORIENT.5
 NAME  ON
 TYPE  ()
 PART1 &PHYSOBJ.19
 PART2 &PHYSOBJ.24)

---------------------------
```

Sentence interpretation results as follows. The primary structure is the causal which shows that the last connection (**CATCHES**) disables the elastic (**COMPRESSION**). The *compression* was analyzed as a success state but is known to be enabled by the action (**PUSH**) of the *object*, and some unknown restraint of **SPRING**. We also know that **OBJECT** is restrained (**CONTAINED**) by a **BARREL**. Since the primary process is elastic (**COMPRESSING**), the use of the device for this scene is taken to be the same use of the *compressing* scene, or energy storage.

PART III

Related Work, Scope, Extensions and Conclusions

The value associated with the FONM representation theory is determined as a function of how it relates to existing work, how well it addresses the problems it is intended to address, and to what extent it provides new knowledge in a rapidly evolving discipline. This part of the dissertation consists of three chapters which address the value of FONM.



Chapter 8
Comparison to Related Work

This chapter is organized in three sections as follows: First, the models and systems that were used as a starting point for the design and construction of FONM are briefly discussed. FONM is then compared to AI representation approaches that describe devices at the abstraction levels addressed by FONM. Finally, the FONM demonstration models are compared to AI systems that are similar in scope.

8.1 Background Work Supporting FONM and its Models

The approach used to develop FONM makes extensive use of Roger Schank's contributions to knowledge representation, that of (1) modeling naturally occurring language tasks and behavior, and (2) knowledge representation and organization in terms of sets of atomic primitives. The work of Schank and his students has been utilized in the following two areas.

- **Conceptual structures**: The Conceptual Dependency representation of actions [Schank 1973, Schank 1975], the script representation of event sequences [Schank and Abelson, 1977, Cullingford 1978], the representation of named plans, and the goal type taxonomy [Schank and Abelson, 1977].

- **Demon-based control**: Using demons and agendas to organize and control process knowledge as implemented in BORIS [Dyer 1983], IRON-FINDER [Reeves 1989], and THUNDER [Reeves 1991], and also by Lenat in AM and EURISKO [Lenat 1976, Lenat 1983, Lenat and Brown, 1984].

8.1.1 Device-Use

The design of FONM device-use plans is an extension of the Schank and Abelson USE(object) plan. In Conceptual Dependency theory, two methods are used to represent action sequences: scripts and plans. A *script* is a stylized template which describes prototypical or compiled action sequences. A *plan* is an abstract template which describes the method, or sequence of actions, as a collection of subgoals. Scripts are very stylized, and are recognized, and processed, quickly. The design-intended use of a device is readily recognized when the device is mentioned, and is very stylized in its application. A script has two drawbacks with respect to the representation of device use. First, if the device fails to work, for one reason or another, a script does not support recovery, so it is not a suitable structure for problem solving or creativity. Second, scripts represent known applications, so they cannot be used to represent unintended device functions. Plans are not processed quickly but are more general than scripts. Because of its general structure, if a portion of a plan fails, the entire plan doesn't automatically fail. Unlike a script, both an explanation for the failed por-

tion, and an alternative method for recovering from the failed portion, can be suggested. This is more in line with machine fault diagnosis and problem solving.

Schank and Abelson proposed two structures for addressing the strengths and weaknesses of scripts and plans when applied to device use: (1) named plans, and (2) instrumental scripts. The *named plan* was proposed as an intermediary between a script and a plan, and has characteristics of both. The named plan describes routine activities, and is represented as "a fixed sequence of subgoals which form the usual path to goal attainment." The *instrumental script* was proposed to represent "fixed and uninteresting actions" associated with device application, such as driving a car or working a keypunch, and was specifically associated with named plans [Schank and Abelson, 1977]. Being a fixed sequence, the named plan is scriptlike. Being represented as a sequence of subgoals, the named plan is planlike.

8.1.2 Object Use

Schank and Abelson's USE(object) is a named plan used to represent the application of objects toward the achievement of some goal. In their notation, USE(object) represents a sequence of four subgoals and an action, the template for which is shown in Eqn. (8-1):

$$\text{USE(object)} = \text{FIND(object)} + \text{D-CONT(object)} + \text{I-PREP(object)} + \text{DO} \qquad (8\text{-}1)$$

Each component on the right hand sides of Eqn. (8-1) and Eqn. (8-2), except DO, represents a subgoal to be achieved on the way to enabling the USE(object) plan. Finding an object means knowing that the object is appropriate, knowing where it is located, and getting there. Schank and Abelson defined the FIND(object) subgoal as follows:

$$\text{FIND(object)} = \text{D-KNOW(loc(object))} + \text{D-PROX(loc(object))} \qquad (8\text{-}2)$$

where the object may be a name of a specific object or a template for an object with a specific property. The DO in Eqn. (8-1) represents the action taken by the actor once the subgoals have been achieved. The addition signs (+) denote a conjunction between the subgoals and the actor's action. In the Schank and Abelson model, d-goals (e.g., D-CONT) are called delta goals, and i-goals (e.g., I-PREP) are called instrumental goals. D-goals and i-goals support the achievement of higher level goals (i.e., crisis, preservation, satisfaction, achievement, and entertainment) [Schank & Abelson, 1977]. For example, a high level goal to preserve one's health (a P-HEALTH goal) is associated with a number of plans, one of which might be to brush one's teeth. Three delta goals which support the achievement of preserving one's teeth are knowing (D-KNOW) the location of the toothbrush, getting to (D-PROX) that location, and gaining physical control (D-CONT) of the toothbrush. These goals can be achieved a number of ways, such as with the actions of remembering, walking, and grasping, respectively. There is a direct causal relationship between delta goals and the actions, scripts, and plans which are used to achieve them. An instrumental goal is the preparation (I-PREP) of the toothbrush for use. This goal is achieved by locating the toothpaste, opening the container, placing toothpaste onto the toothbrush bristles, and moving the toothbrush to one's mouth. Whereas the delta goal leads to new actions or plans, the instrumental goal generally leads to instrumental script application, where the causal relationship is indirect because the representation level needed to specify how the goal is achieved is beyond the scope of the Schank and Abelson approach.

With respect to device use representation, the USE(object) plan is problematic for four reasons. Consider the USE(object) plan applied to the task of brushing teeth, as illustrated

in Fig. 8.1 The first three subgoals (D-KNOW, D-PROX, and D-CONT) can be satisfied

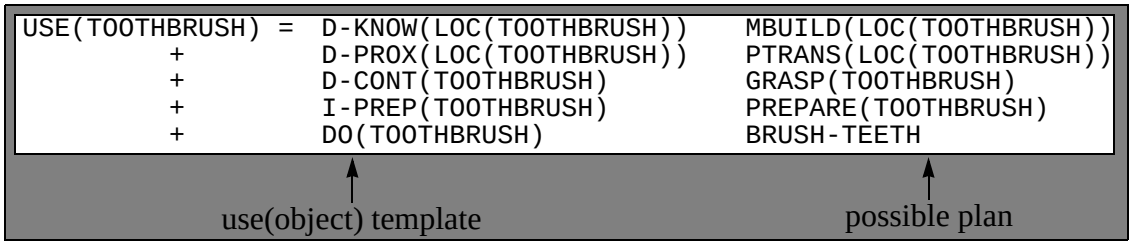


Figure 8.1 A possible instantiation for the USE(TOOTHBRUSH) plan.

with actions such as MBUILD (e.g., “remember”), PTRANS (e.g., "walk"), and GRASP (e.g., "pick up"). The remaining subgoals, however, should be resolved with new actions, scripts, or plans. Any of these which describe device application should be causally related to the device’s function and behavior. None of the Schank primitive acts, instrumental scripts, or plans are formally related to device function or behavior. Second, the preparation of the toothbrush for use involves a number of objects and actions. For example, the application of toothpaste implies a separate USE(TOOTHPASTE) plan. It is difficult to represent PREPARE(TOOTHBRUSH) with an instrumental script with a USE(TOOTHPASTE) plan embedded within it. Third, if the toothpaste container is found to be closed, a problem arises for the representation of PREPARE(TOOTHBRUSH) as an instrumental script, because the script implies that there is one way to prepare the toothbrush for use and does not allow for alternative plans, such as first opening the toothpaste tube. This is a general problem when scripts are underspecified. Finally, the procedure of brushing ones teeth involves complicated robotic movement, none of which is represented at the scriptal level. In order to effectively represent device use, the USE(object) plan must be modified to address these issues.

These considerations motivated the development of device-use plans. The FONM device-use plan extends the USE(object) plan by considering more than one object at a time, and by considering the dependencies between the different objects which are part of the plan. The development of device-use plans resulted in the parallel development of goals specifically associated with the application of device function.

8.2 Related Work in Representing Mechanical Devices

A primary motivation in developing FONM has been to provide an integrated mechanism for representing knowledge about devices at different levels of abstraction. Although many systems have been devised for representing and reasoning about physical systems (e.g., Hayes’ Naive Physics Manifesto [Hayes 1979], DeKleer and Brown’s conduits in ENVISION [DeKleer and Brown, 1983], Chittaro’s Multi Modeling model [Chittaro et al, 1993]), seven approaches share close similarities with the representational scope of FONM: (1) Forbus’ Qualitative Process theory [Forbus 1984], (2) Sembugamoorthy’s and Chandrasekaran’s Functional Representation [Sembugamoorthy and Chandrasekaran, 1986], (3) Rieger’s Common Sense Algorithm [Rieger 1975], (4) Lehnert’s Object Primitives [Lehnert 1978], (5) Lind’s Multilevel Flow model [Lind 1982], (6) Ulrich’s bond-graph model [Ulrich and Seering, 1987], and (7) the Finite Element Method [Clough 1962].

8.2.1 Qualitative Process Theory

Qualitative Process (QP) theory is a low-level, qualitative, approach for representing the behavior of physical systems, particularly thermodynamic systems and heat transfer [For-

bus 1984]. In QP theory, all object behavior is represented with processes. The process is associated with an individual view, or snapshot, of an object, and describes the interactions and influences which define what the object can and will do. QP theory enables object simulations to be performed, because a qualitative process provides the causal mechanism for describing changes in object properties with time, called an object state history. The strength of the QP approach is that it describes the causal nature of object behavior and interaction regardless of setting or intention. The Forbus model emphasizes the role of the process in making inferences about object relationships.

In QP theory, objects are represented as a collection of quantities, a substance, and a location, as shown in Fig. 8.2

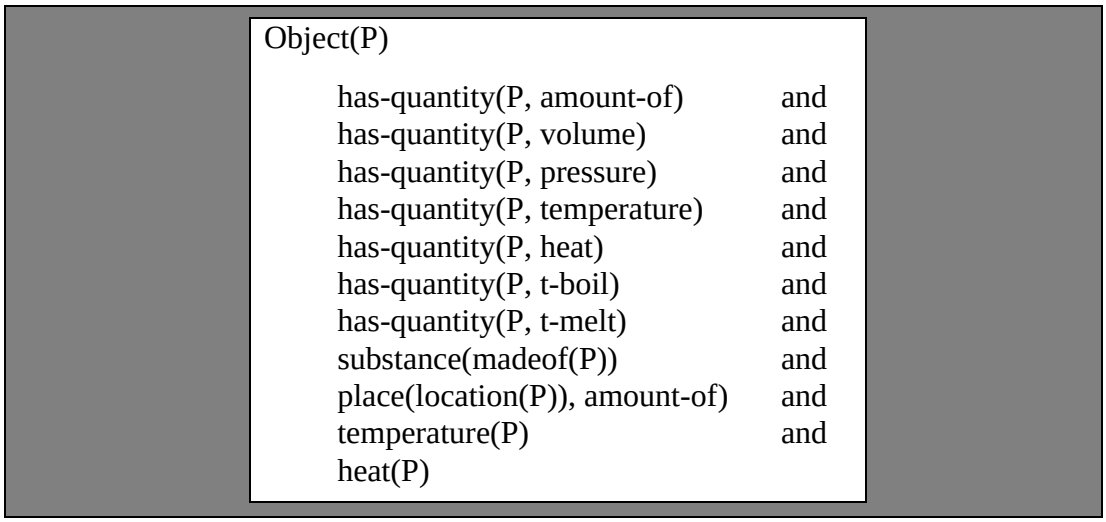


Figure 8.2 Object descriptions in Forbus' QP theory.

Although the quantities which describe objects, in Fig. 8.2, describe thermodynamic properties of objects, any quantities can be represented with this model. Objects can participate in what Forbus calls an *individual view*, which represents a collection of objects and the relations between their quantities. Individual views describe known states which can be used

to recognize behavior, for example for moving friction, as shown in Fig. 8.3.

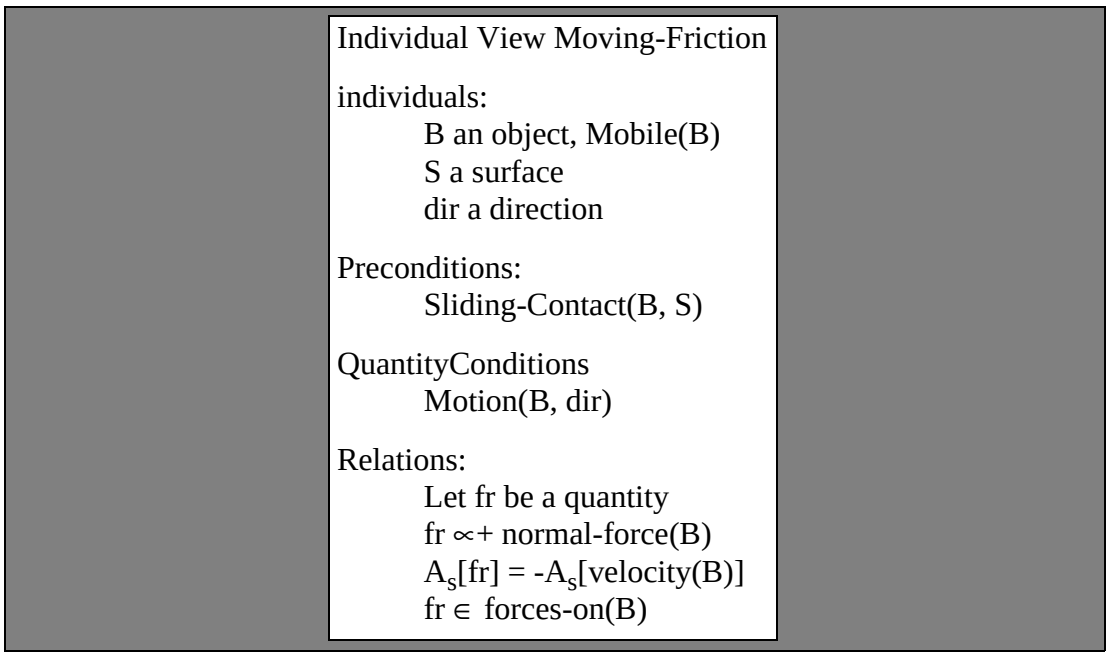


Figure 8.3 Forbus individual view for moving friction.

The individual view in Fig. 8.3 describes a static relation between an object (B) and a surface (S), each of which would be represented by Forbus as shown in Fig. 8.2. Objects which satisfy this definition are in a state of moving friction. The *preconditions* in the individual view describe predicates which must be satisfied by the object descriptions. The *quantity conditions* describe process requirements for the specified objects. In this example, the object (B) must be in motion in direction dir. The *relations* describe interactions between the objects and their quantities while this view is in effect. In this case, a new quantity, friction, is introduced, and its relation to the object's velocity and other forces on the object is defined. QP individual views represent a snapshot of an object in context with other objects and enables recognition of the types of behavior the object is either engaged in or can engage in. In QP theory, processes control the change of object state and individual views. Processes have 5 components: objects, behavioral preconditions, quantity preconditions, behavioral relations between objects, and the resulting influences or tendencies (states). For

example, in Fig. 8.4 the process associated with motion is illustrated.

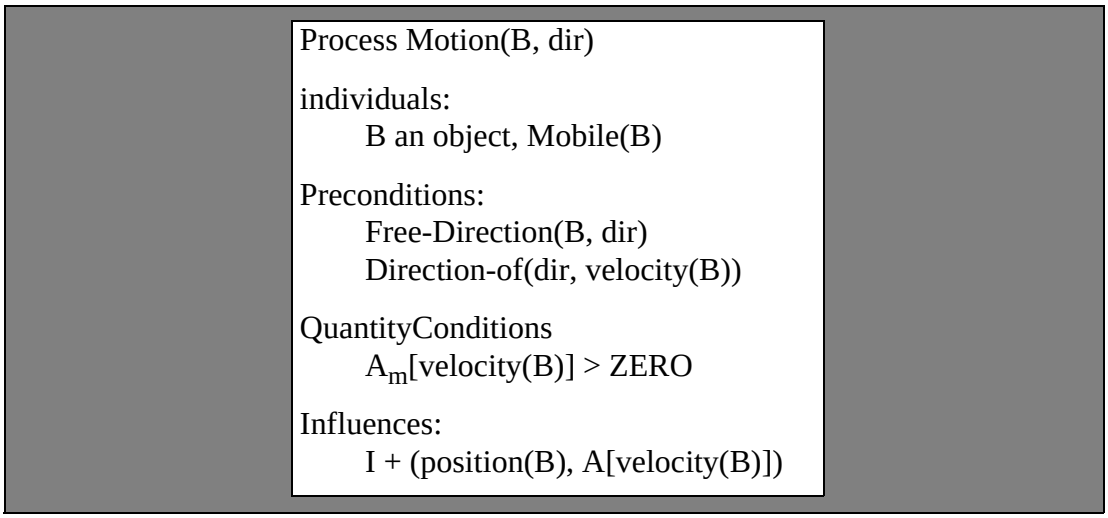


Figure 8.4 Qualitative process for motion.

The difference between an individual view and a process is that the process, through the *influences* component, describes changes to the individual views of objects, and thereby, the ability to generate behavioral histories (i.e., change) for an object. For example, the process for motion depicted in Fig. 8.4 describes a schema for objects which satisfy a mobile predicate in direction dir and have a velocity in dir. The motion influence affects the position and velocity of the object.

Objects in FONM are also represented as a collection of quantities and position. However, instead of a material type, material properties are also represented as quantities. The quantities which can be influenced by processes are distinguished from those which cannot. FONM also describes objects in terms of appearance. QP theory only describes object function in terms of object behaviors, so there is no need for representation of appearance and how it affects function.

The QP and FONM process representations are very similar. Both process models represent the same type and level of information. The FONM process identifies process roles and keeps them separate from the dynamics of the process definition, whereas the QP process combines static and dynamic process components. For example, the FONM process *src* and *dst* roles are represented in a QP process *individuals* role. The FONM process *behavioral preconditions* are equivalent to QP process *behavioral preconditions*. The FONM *perturbation preconditions* are equivalent to the QP *quantity preconditions*. The FONM *from* and *to* roles identify the change of state associated with the behavior, and the QP *influences* role provides a relation for determining the change of state. The algorithmic nature of the QP process influences relations is more detailed than the FONM *from* and *to* states, since it provides the information to calculate state histories.

The closest FONM analog to the notion of the QP *individual view* is a static device representation. The individual view is a construct which helps to recognize an object's behavioral status, and accounts for its dynamic dependencies with other objects and processes. In FONM, the active dynamic processes for an object would be instantiated as a collection of structures, but not as a single structure. The static representation would provide access to those dynamic structures, so some of the same information is represented. For example, the Moving-Friction individual view illustrated in Fig. 8.3 requires recognition that the object B is in motion (not just mobile) and to have access to its velocity for calculating the friction

force value. In FONM, friction is also a force, but would also be associated with a STORE process. As soon as an object is recognized as having surface contact and has having been in motion, the active TRANSFORM process would enable recognition of a STORE process and the effect of friction would be accounted for by distributing it into applied and internal forces (i.e., object motion is reduced because part of the applied force is now 'missing'). In this respect, the QP process and individual view representation has an equivalent FONM representation. By all accounts, the QP representation of friction is a more accurate representation of the behavior, because the relationship between velocity and force is described. In FONM it is only alluded to.

The intended domains of QP theory and Behavioral Process Primitives overlap. BPPs are not intended to support simulation to the same degree of granularity as QP processes, but to enable recognition of when a process is enabled or disabled, and to identify the type of state an enabled process results in. BPPs cannot perform detailed diagnosis, since there is no internal representation for space and time, or for rates of change such as velocity and acceleration. For example, in FONM, when an object stops moving, the object either ceased to be forced or its path was somehow blocked, because these are the two MOTION process enablements.

8.2.2 Device Functional Representation

Sembugamoorthy and Chandrasekaran have developed an approach for representing and reasoning about device function [Sembugamoorthy and Chandrasekaran, 1986]. Their claim is that a device is used to perform a task, and as long as its behavior can be predicted and diagnosed, the internal behavioral relationships can be ignored. The definition of device function is based on representing the function of devices on three levels: (1) causal, (2) temporal, and (3) interaction. The *causal* level identifies function in terms of the behavior which is produced by the correctly functioning device, such as the buzzing of a household buzzer. The *temporal* level describes the temporal dependencies between these behaviors, and the *interactive* level describes how information (material flow, force, etc.) is communicated between components. Within this scope, the *Functional Representation* (FR) recursively describes function in terms of the relations between its components. There are five components to the representation of functional knowledge: (1) the structural relationships between components, such as connectivity, (2) the function, or what the device does as a result of stimulus, (3) the behavior, or how, given a stimulus, the behavioral response is accomplished, (4) generic knowledge, such as rules for particular laws of physics, which contribute to the device behavior, and (5) assumptions which must be satisfied to support the behavior. These components are illustrated below (Fig. 8.5) for the household buzzer and

represented shown represented in Fig. 8.6

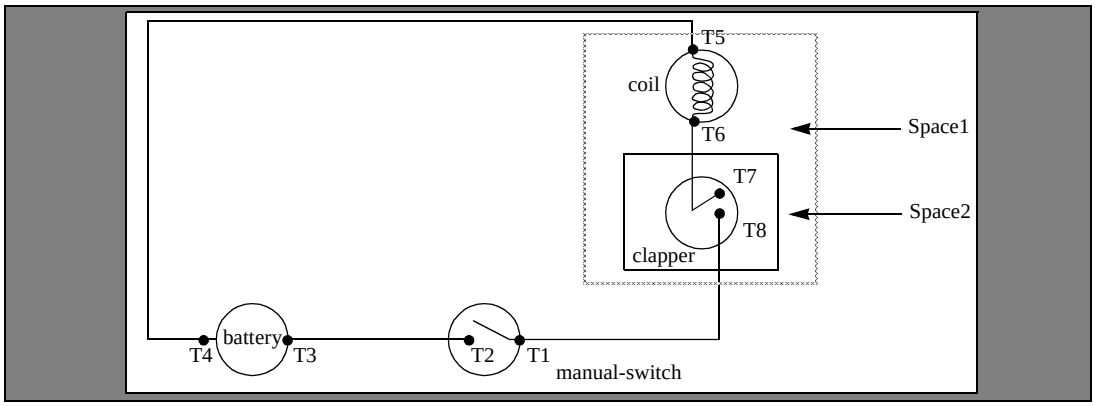


Figure 8.5 A schematic diagram of a household buzzer.

In Fig. 8.5, T1-T8 represent *terminals* of the components, and Space1 and Space2 represent locations in which relations can be effected. FR components of the buzzer representation

in shown in Figs. 8.6, 8.7

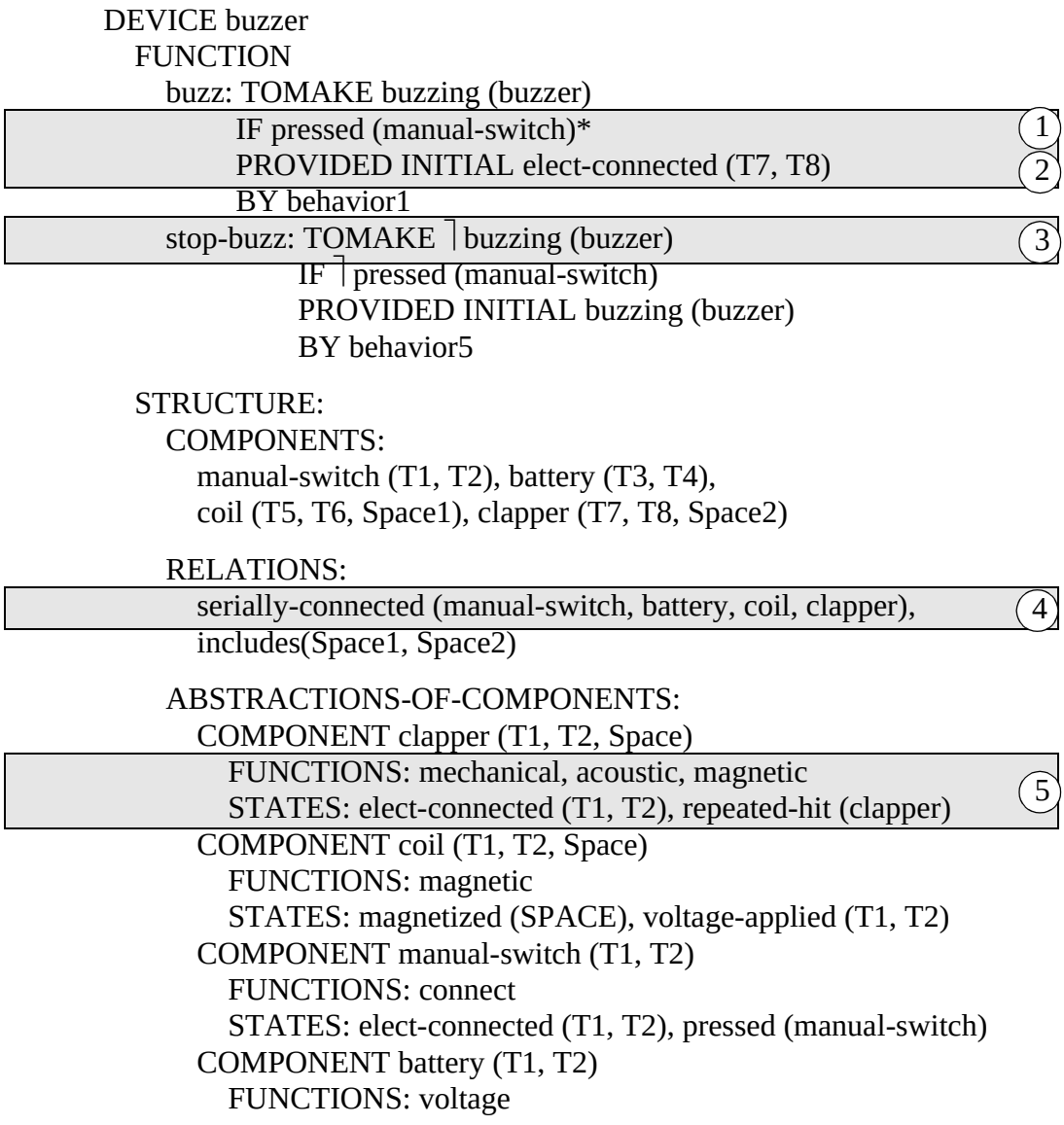


Figure 8.6 FR representation of buzzer.

Fig. 8.6 illustrates the function and structure components of the FR buzzer representation. The function aspect of the representation identifies what happens (TOMAKE buzzing, at 3), how the function is initiated (IF pressed, at 1), and its enablement conditions (PROVIDED INITIAL, at 2). It also identifies how to stop the device from buzzing. A primary assumption of FR is that function is directly associated with purpose. The function TOMAKE thus identifies the purpose of the device and what is produced.

The structure component of the representation identifies the physical components, their relations, and their abstract definitions. Behavioral relations such as *serially-connected* (4) and *elect-connected* (5) are described by the generic knowledge component of the representation. The abstract definition entails identifying affiliated FUNCTIONS and STATES associated with the component. Examples of the FR behavior and generic knowledge com-

ponents of the buzzer representation are shown in Fig. 8.7

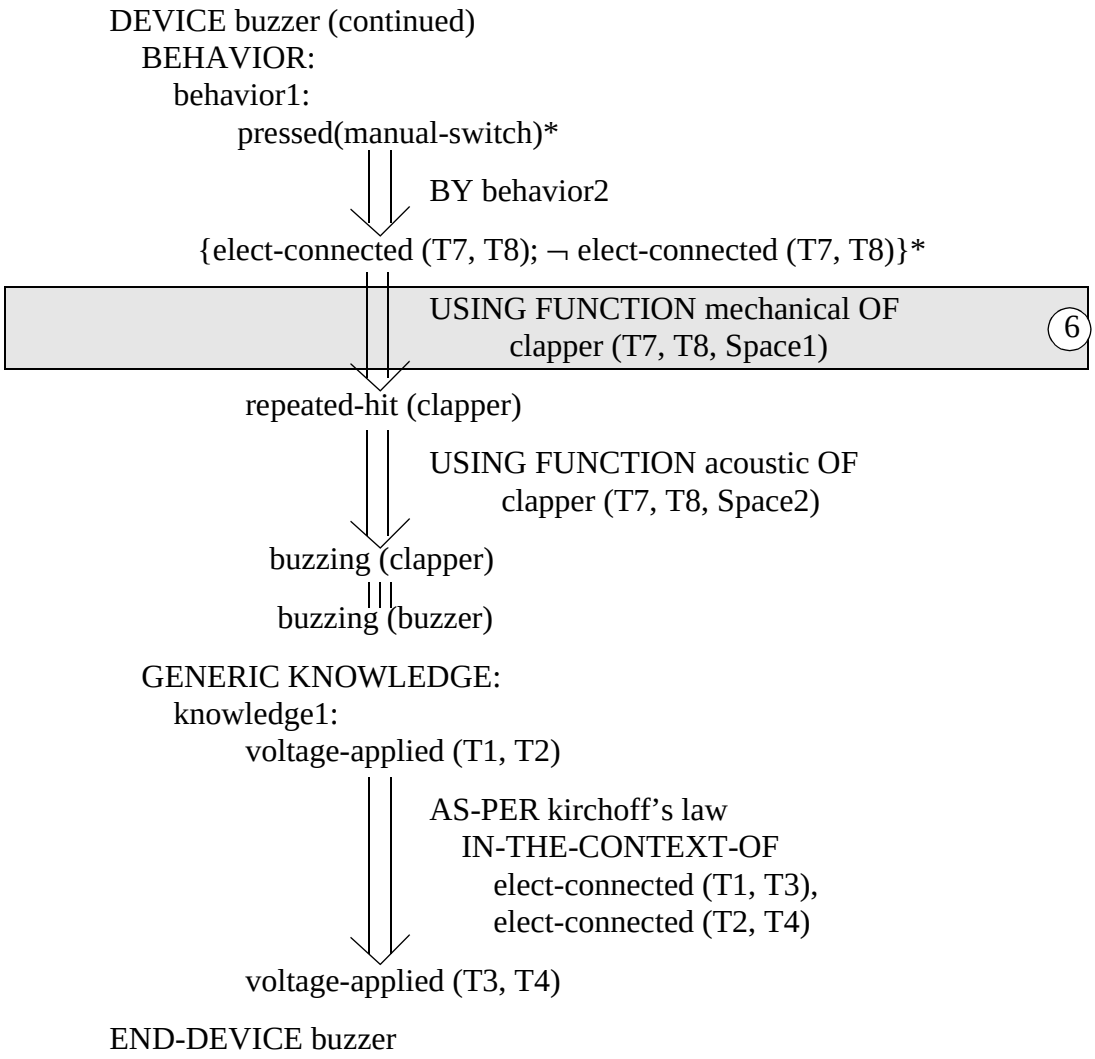


Figure 8.7 FR behavior and generic knowledge representation components of the electric buzzer.

The structure of the FR behavior component for a buzzer indicates that behavior is represented as the functions of the components of the buzzer, rather than their explicit behaviors as would be the case in QP or FONM. For example, in Fig. 8.7 the USING FUNCTION mechanical OF clapper statement (at 6) mediates the electrical connection and the clapping result. The behavior of the clapper: its position, movement, and striking are not specifically described, so the creation of sound is not specifically represented. Since the functions of objects in FR identify their behavioral enablements, the approach can be useful for malfunction diagnosis to the extent of identifying a disfunctional component, but not of explaining why or simulating how.

The FONM approach is basically a functional representation approach, since the interpretation and application of objects, in context, is based primarily on the functional nature of a problem-solver's experience with objects. There are three differences between the models. First, FONM makes the assumption that what an object does (its function), is not one and the same with its purpose but, rather, that a function is never invoked without a purpose. The purpose is itself represented as a goal, and the means whereby the goal is

achieved, by invoking the device function, is represented with a plan. The FONM approach provides access to the goal/plan representation level independently of the device function, behavior, and structural levels and vice versa. Likewise, causal relationships are defined between each of the different knowledge structures, and they can be combined and manipulated independently of one another. The FR device representation captures the same kinds of information but cannot make use of it independently of the device description.

Second, FONM proposes a taxonomy of 11 machine primitives for describing object functions, which help to classify and recognize devices. The Functional Representation provides a representation structure for describing function but, like Qualitative Process Theory for behavior classification, makes no claims for a closed set of device functions. Each device function can be compared along metrics of TO MAKE, etc., but there is no classification of TO MAKE functions which aid in comparing similar devices. The FONM representation for a LEVER function can be applied to comparing any two objects which can instantiate the function and thus potentially be considered as replacements for one another.

Third, the knowledge associated with FR device representations takes the form of rules which are identified with the device. FONM device representations have device-specific rules which are identical in nature to the FR rules. FONM knowledge structures also have rules associated with them which can be used to make inferences about devices which are not specifically associated with a device representation. The FR representation is useful for prediction and diagnosis as long as the intended device function is appropriate to its context. However, no mechanism is provided for relating the device to why it is used, what it was expected to do, or how it was expected to do it outside of the intended context.

In other respects, the FR structure component is equivalent to the FONM static representation. The FR behavioral component is similar to the FONM combination of machine primitives to describe device function in terms of the functions of its components. FR has no equivalent for the behavioral sequences which describe component functions. Nor does FR have any representation for object shape or appearance and their affect on device function and behavior.

8.2.3 CSA and the Representation of Device Function and Application

Rieger's *Common Sense Algorithm* (CSA), integrated Schank's action primitives and object functionality to describe commonsense algorithmic knowledge, or the causal interactions between actors and objects and between objects and other objects [Rieger 1976, Rieger and Grinberg, 1977]. Rieger introduced the notion of an "agentless" action, called a tendency, with the capacity to describe influences such as gravity. He also introduced a family of causal links which were used to describe various causal (temporal) relations and their effects. For example, using CSA, Rieger was able to describe the use of various devic-

es, including a bicycle, a flush toilet, and the bulb bicycle horn detailed in Fig. 8.8.

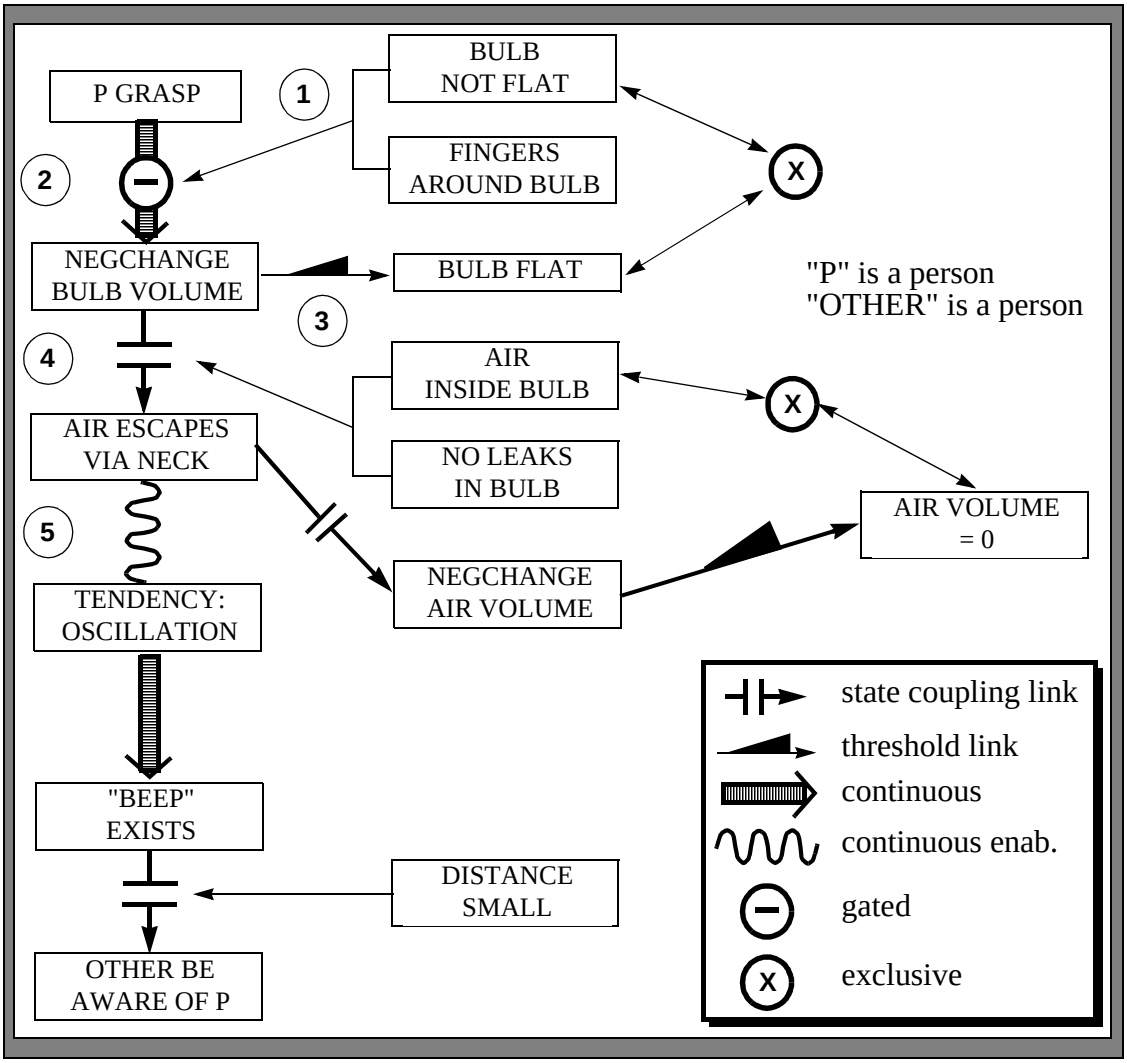


Figure 8.8 Rieger's CSA applied to a bicycle horn.

Figure 8.8 shows how a bulb bicycle horn works using Rieger's CSA. The central aspect of the approach was to identify 5 event types (action, state, statechange, tendency, and want), and to introduce 26 links which described causality, enablement, concurrency, iteration, intention, gating, and thresholding among those event types. In the figure, actions such as "P GRASP" and "OTHER BE AWARE OF P" are events which define the initial and final states of the horn's function. The GRASP event represents the input applied to the horn which initiates the function. The AWARE event represents a perceptual effect produced by the horn, which terminates the function. The actual function of the horn, in this context, can be considered the production of sound, and that OTHERS become aware of the sound and P, is related to the goals of P. For the most part, the boxes in the diagram represent states of various components of the bicycle horn. For example, "NEGCHANGE BULB VOLUME" describes a new state of bulb volume. The tendency "OSCILLATION" also describes a change of state, even though it appears to be a dynamic structure. The dynamics in this diagram/representation are captured by the links. For example, in the presence of P grasping the bulb, fingers being around the bulb, and the bulb not being flat (at 1), a one-shot, continuous state change is effected (2), that of the negative change in bulb

volume. The one-shot aspect of the relation means the events are sequential and non-repeating. The circle in the link is called a gate, and identifies additional enabling conditions for the event. The link at (3) represents the associated intended state that P has, and that, eventually, the negative change in bulb volume will lead to the bulb being flat. It is called a threshold link because the goal is being approached, and is suddenly achieved. The link at (4), called a state coupling link, relates two states through an unknown causal relation. It makes an equivalence between the negative change in bulb volume and the escaping of air through the neck of the horn. The relation at (5) is called a continuous enablement, and relates the escaping air to the oscillation tendency.

CSA emphasizes how objects interact, and how object functions are applied and perceived by problem solvers. It identifies the types of relationships which problem solvers associate with object use. The notion of causality is identified at the object function level and the attempt is to relate observed states with object behavior (tendencies). CSA and FONM can both be used to represent the function of a device and how function is applied and perceived by its user. CSA identifies object function with actions by the observable states which are produced by the function when the action is applied. FONM does this, but also identifies object function with the plans and goals associated with why the object function is being invoked. Thus a FONM representation supports the application of devices used in dissimilar situations as long as the goals or plans are similar. CSA's dynamic links provide a notion of temporal causality as interpreted by a device user, and has no direct equivalent in FONM. In FONM, temporal causality is captured by the sequencing of behavioral processes, which are enabled or disabled by the states of other objects. There is no FONM notion of a threshold, for example, because it implies the combination of causal effects (one thing happens, a value is reached, and then something else happens). In FONM each causal effect is associated with a particular process, so the thresholding value in CSA is an enabling condition in FONM. The same is true of gating conditions. The notions of one-shot and continuous enablement in CSA are related to FONM by how the behavioral process is defined. A FONM process, once enabled, remains enabled unless disabled by another state or process, so the same causal relationships are represented without defining new forms of causality.

8.2.4 Object Primitives and the Representation of Device Use

Lehnert's *Object Primitives* were proposed to assist in making inferences about objects in natural language processing, such as for the sentences shown below [Lehnert 1978]:

S1: John drank from the faucet.

Q1: What did John drink?
A1: Water.

S2: John filled his canteen at the spring.

Q2: What did John get at the spring?
A2: Water.

Lehnert's notion was that the faucet and spring both serve (from a naive perspective) as sources of water and should share representational similarities which enable a program to recognize this and make similar inferences. Lehnert proposed a set of 7 primitives for describing similarities and differences between objects based on how they are used: (1) setting, (2) gestalt, (3) relational, (4) source, (5) consumer, (6) connector, and (7) separator.

The use of these primitives are illustrated in the examples shown in Figs. 8.9 - 8.13.

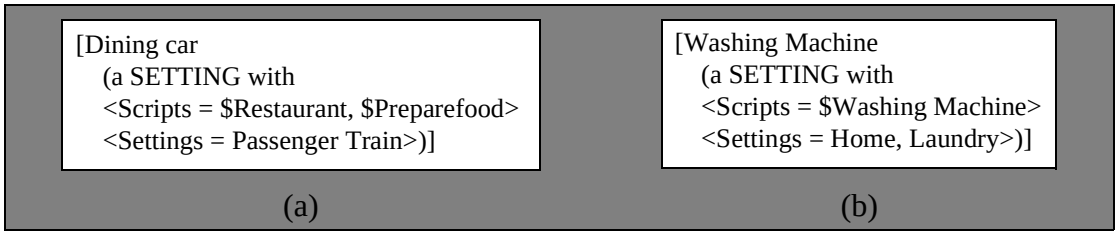


Figure 8.9 Examples of SETTINGS, (a) location type, (b) object type.

Fig. 8.9 illustrates the two types of SETTING in the Lehnert approach. In the first case, a setting is a location where something happens, and can be associated with scripts and additional settings. In the second case, a setting is identified by a particular object in terms the tasks which can be undertaken with it. This latter designation is similar to FONM REGIONs and machine primitives, which relate an object to its known functions, and with the plans which invoke a particular function. A device's intended function and its associated plan are then similar to the Lehnert functional setting.

Lehnert's notion of GESTALT is used to describe object compositions which are functionally related, whether or not they are connected together. Two examples are illustrated in Fig. 8.10

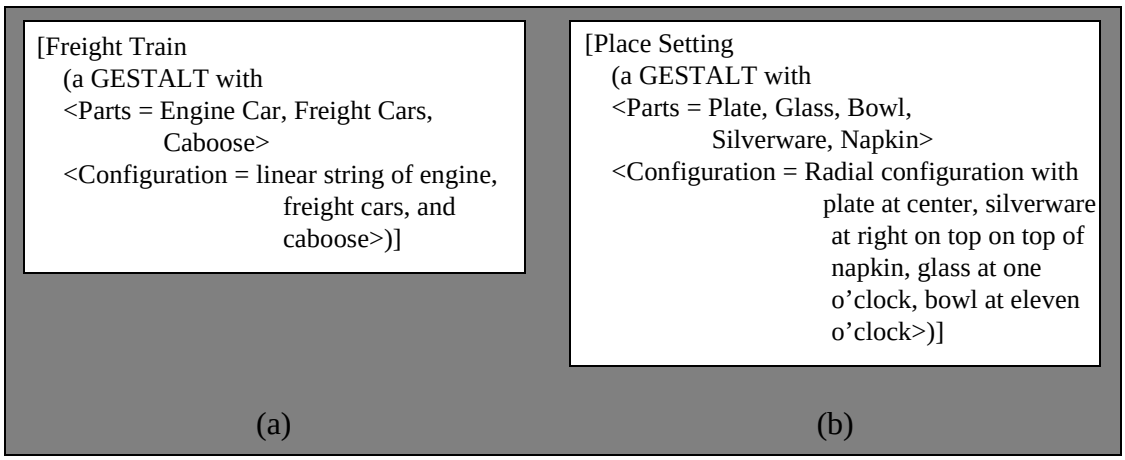


Figure 8.10 Examples of the GESTALT primitive, (a) connected objects, (b) functional cluster.

The GESTALT examples in Fig. 8.10 point out the need to identify an object's components and their orientation to one another. Unlike most representation approaches for objects, the Lehnert model has no specific representation for placement or connectivity. For example, in Fig. 8.10b, the manner in which orientation and value are defined is left unclear. In FONM, one object is selected as a focus, and the placements of the other objects are represented with respect to that object. This is an important consideration if component placement is important to how the device functions as a whole.

The RELATIONAL primitive describes an object's relationship with, and dependence upon, other objects, as shown in Fig. 8.11. In this figure, the RELATIONAL is used to represent object support (Fig. 8.11a) and object hanging (Fig. 8.11b). The representation provides the participants, and a link identifying their behavioral dependency (e.g., On-top-of and Stuck-to), but no representation for how these relationships are achieved. In FONM

support and hang are represented with RESTRAIN process instances. The relationlink 'On-

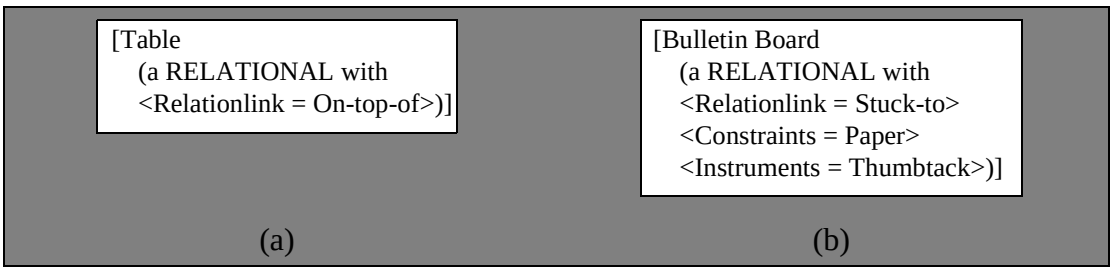


Figure 8.11 Examples of RELATIONAL, (a) table support, (b) bulletin-board and thumbtack.

top-of' is represented with an ORIENT structure which identifies the dimension of comparison and the value which defines the tables as below the object. A RESTRAIN-SUPPORT then represents the restraint state which defines the object's support dimension and location. Similarly, the 'Stuck-to' relationlink is represented, in FONM, with a RESTRAIN process which results in a restraint state on the paper in all dimensions accept rotational about the thumbtack axis. By not representing the dynamics associated with these relations, it is impossible to say anything about the behavior of the objects when the relation is removed. For example, if the thumbtack is removed in Fig. 8.11b, the RELATIONAL doesn't say whether the paper will fall. Also, the RELATIONAL doesn't show the similarities between the two relations (On-top-of and Stuck-to), which reduces the ability to share inferences between the two relations.

The SOURCE and CONSUMER primitives are used to infer context of object use by defining how the object is related to the context in which it is used. For example, a sponge can be represented in terms of providing a source of fluid when squeezed, and as a consumer of fluid when used to wipe (Fig. 8.12). Recognizing what action is being performed then enables an inference of which object function is being applied. In FONM a sponge would be represented as a container. The container can be filled or emptied, each of which has an analog with the notions of source and consumer.

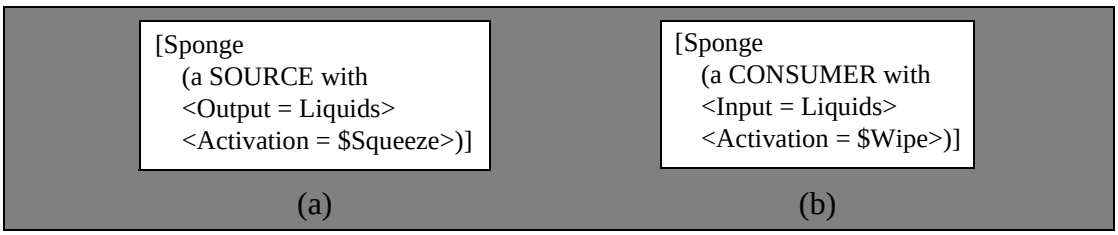


Figure 8.12 Example of a sponge as a SOURCE and CONSUMER

The SEPARATOR and CONNECTOR primitives enable and disable actions a user can engage in with respect to the object, and also assist in recognizing how the object is being used. A separator disables actions between spatial regions, and a connector enables actions between spatial regions. Consider the window represented in Fig. 8.13. When the window is open, communication (MTRANS), perceptual stimulus (SPEAK), perceptual reception

(ATTEND), and movement (PTRANS) are enabled. When the window is closed, both the

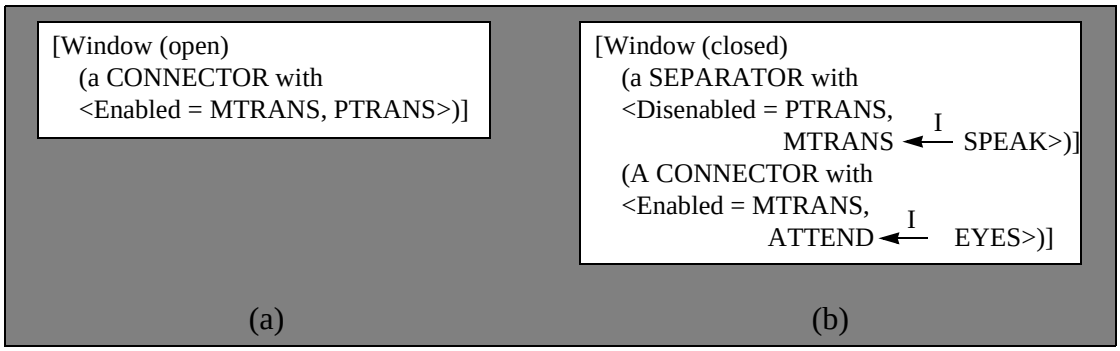


Figure 8.13 Examples of window as CONNECTOR and SEPARATOR, (a) open, (b) closed.

SPEAK and PTRANS actions are disabled. In FONM the window is represented as a container with opening and closing functions. The states which initiate and terminate the respective functions enable and disable PTRANS type actions, while the physical characteristics of the window enable and disable MTRANS and ATTEND type actions. The function is recognized by associating the goals of an actor with the existing states of the device. The difference between the approaches, for SOURCE/CONSUMER and CONNECTOR/SEPARATOR is that FONM associates functionality with the object and object primitives associate functionality with intent.

8.2.5 The Multilevel Flow Model and the Representation of Systems

The Multilevel Flow Model (MFM) has been proposed by Lind [Lind 1990, Lind et al, 1992] to model the function and control of complex dynamic systems. The MFM model is intended as an integration of system function and the intentions (i.e., goals) of the users who maintain and supervise its operation. The model describes a physical system as being comprised of: (1) goals, (2) functions, and (3) physical components. The overall system is hierarchically defined to describe subsystems and their causal integration. The model provides formalized concepts that describe causal mappings between these system components. In MFM, three types of goals are defined: (1) maintain, (2) prevent, and (3) achieve. These goals can be applied to one of three categories: (1) safety goals, (2) production goals, and (3) economy goals. Goals are represented as a 2-tuple: an expression to be met, such as an inequality, and a priority for ordering. The use of these goals is applied to an example

of a central heating system (Fig. 8.14):

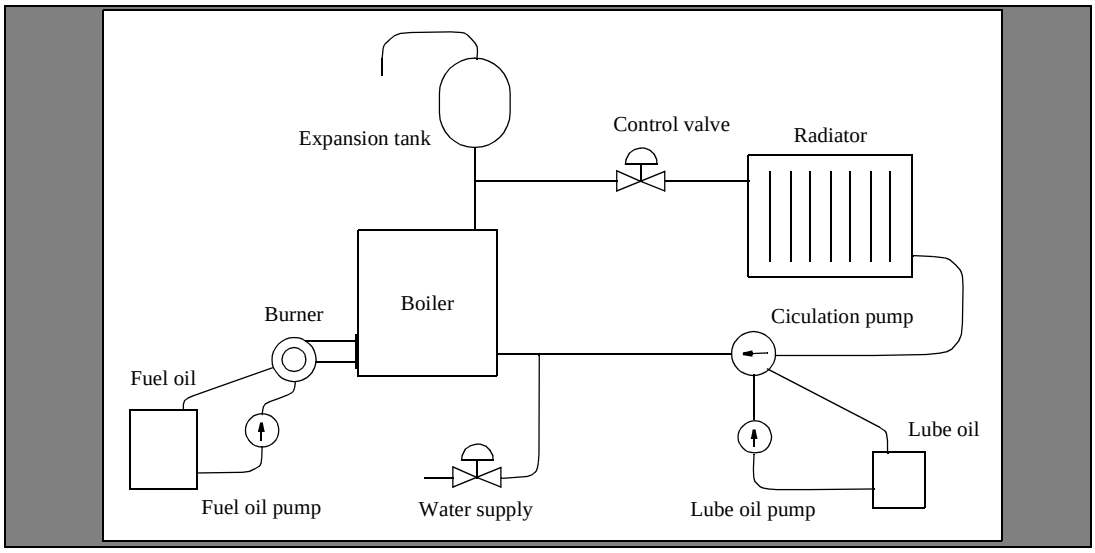


Figure 8.14 Central heating system example for Multilevel Flow Modeling.

Lind identifies 5 goals (G11, G21, G22, G31, G32) associated with the operation of this system:

G11: Maintain room temperature within defined limits

G21: Minimize heat losses

G22: Optimize fuel combustion

G31: Keep water temperature below boiling point

G32: Keep water inventory below upper limits

These goals can be categorized as production goals (G11), economy goals (G21 and G22), and safety goals (G31 and G32). The approach then defines the functions associated with the goals of the system. MFM has eleven functions associated with the achievement of these goals in the central heating system:

F1: Energy supply

F2: Storage of fuel

F3: Air-gas path in the boiler

F4: Distribution of heat in the boiler

F5: Circulation of water

F6: Transfer of heat from boiler to radiator

F7: Control of room temperature

F8: Lubrication of the circulation pump

F9: Storage of lubrication oil

F10: Circulation of lubrication oil

F11: Water supply

In MFM, function is also represented as relations to be met. Three types of function are

described: (1) mass and energy functions, (2) information functions, and (3) organizational functions. The mass and energy functions describe the functional relationships between components and systems, while the other two function types describe how the information associated with device and system function is propagated or controlled. Each of the functions in F1-F11 represent mass and energy functions associated with fluid flow in the central heating system model.

At the lowest level are the physical components themselves. There are fourteen components in the central heating system, most of which are shown in Fig. 8.14.

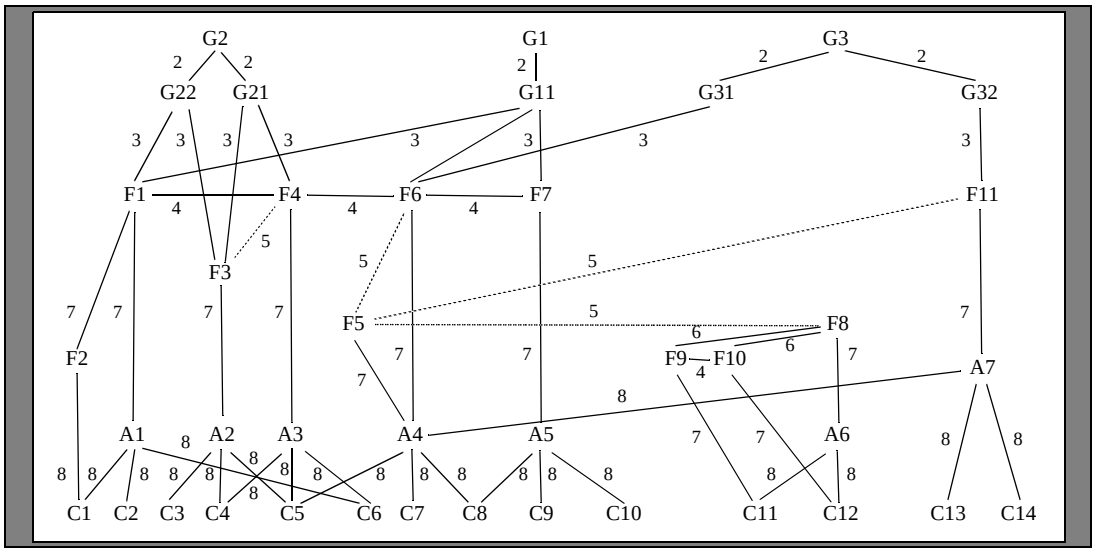


Figure 8.15 MFM mappings between goal, function, and component levels for a central heating system.

The model describes causal mappings between each of these component levels, as shown in Fig. 8.15. The items `G1-G32` represent goals, `F1-F11` represent functions, `A1-A7` represent component aggregates (like mechanisms), and `C1-C14` represent the low-level components. The arcs are labelled with numbers which describe four types of relationships between component levels: (1) ordering, (2) decomposition, (3) relations, and (4) connectivity between the components as detailed below.

1) **Ordering** of goals according to priority
2) **Decomposition** of goal into subgoals
3) **Relation** between a goal and the functions which can achieve the goal
4) **Connection** of two functions belonging to the same level
5) **Relation** between a function and its supporting function
6) **Decomposition** of function into subfunctions
7) **Relation** between a function and its physical implementation
8) **Decomposition** of an aggregate into subaggregates or components
9) **Connection** between two components

In MFM, these mappings are expressed graphically. For example, the graphical repre-

sentations for goals and functions for mass and energy are shown in Fig. 8.16:

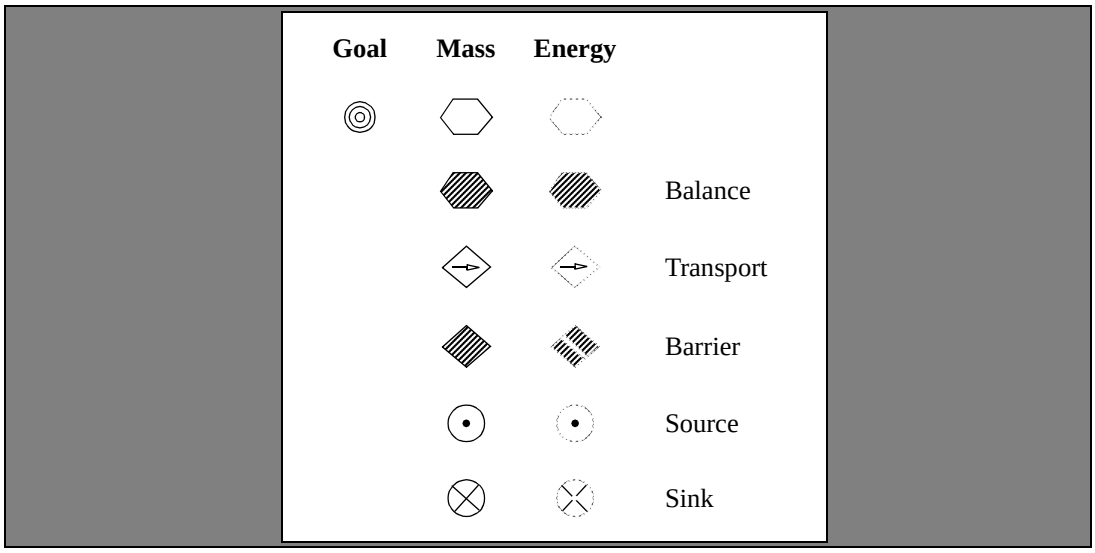


Figure 8.16 Graphical representations of MFM mass and energy flow functions.

These functions represent the functional building blocks for constructing MFM flow models. The functions are related to goals (and components) with links. An ’A’ link represents the achievement of a goal with a function, and an ’A-C’ link represents the achievement of a goal which has a controlling influence on a function (e.g., the addition of a controller unit). There are rules which define which functions can be combined and how they can be combined to maintain physical consistency.

The MFM model of the central heating system is shown in Fig. 8.17. The major func-

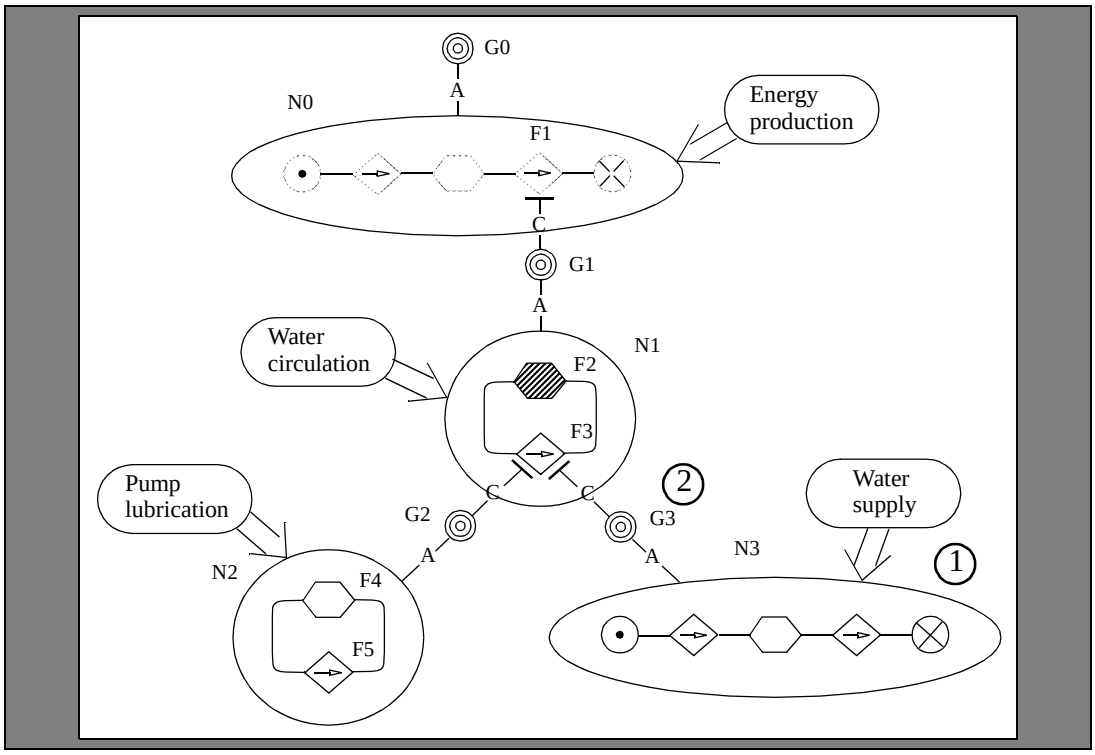


Figure 8.17 MFM model for the central heating system.

tional components can be seen to comprise their own flow subunits (`N0-N3`). The functions of these aggregates achieve subgoals of the overall system's control. For example, the `water supply` function (at 1) achieves the goal `G3` (2), which itself is a condition for the `water circulation` (transport) function. Likewise, the `pump lubrication` function achieves the goal `G2`, which itself is also a condition for water transport (i.e., the pump must be lubricated in order to function). The function `water circulation` then achieves the goal `G1` to provide water flow in the system, without which temperature cannot be controlled. `G1` is a condition on the function of `energy production`, which achieves the overall goal `G0`.

MFM and FONM are very similar in design philosophy. Both models approach the representation of function from the standpoint of how the device is used by individuals and by how the device behaves in lieu of individuals when perturbed. FONM uses device-use plans to provide an intermediate step between user goals and the functions which, when invoked, aid in their achievement. Device-use goals satisfy the control and organization functions in MFM which, in both models, can be achieved by a human or a device. The MFM model does not represent the actual behavior of devices, but, rather, the relations which are met when the device is functioning properly. As such, the diagnostic capabilities of the model are limited. The notion of a flow module is very similar to that of a machine primitive. That the functions represented in MFM are based on mass and energy, and, in particular, with transport and storage, suggests that FONM and MFM models could be merged to describe mechanical and fluid mechanical devices.

8.2.6 Bondgraphs and the Construction of Aggregate Behavioral Models

A *bondgraph* is a graph model for describing energy exchange between components and was developed as an approach for constructing device models for conceptual design [Ulrich and Seering, 1987]. A bondgraph is comprised of ports and bonds, where a port is an object type and a bond represents a path of energy flow between ports. Each bond has associated with it two variables: (1) effort, and (2) flow. In mechanical systems effort represents force, and flow represents velocity. There are four port types in this model: sources, 1-ports, 2-ports, and n-ports. A source represents a device which specifies a force or velocity. A 1-port describes devices which, when a force is applied, have a behavior which is based on force. A lever and a spring would be considered 1-ports, because the type of input and output are basically the same. A 2-port describes objects which transform or convert their input. A screw is a 2-port, because it converts rotational motion into translational motion. N-ports represent bond junctions. There are two types of n-ports: one junctions and zero junctions. A one junction represents a junction of bonds which share a common energy flow, while a zero junction represents a junction of bonds which share a common effort. The sum of the efforts at a one junction is zero, while the sum of flows at a zero junction is zero.

Bondgraphs are constructed by connecting ports and bonds, and can be combined by connecting bondgraph chunks between n-ports. There are four ways to construct a bondgraph:

(1) A source connected to an n-port

(2) A 1-port connected to an n-port

(3) A 2-port connected to an n-port at each port

(4) Two bondgraphs connected by bonds between n-ports

A partial table illustrating some of the bondgraph elements proposed by Ulrich and

Seering is shown in Fig. 8.18:

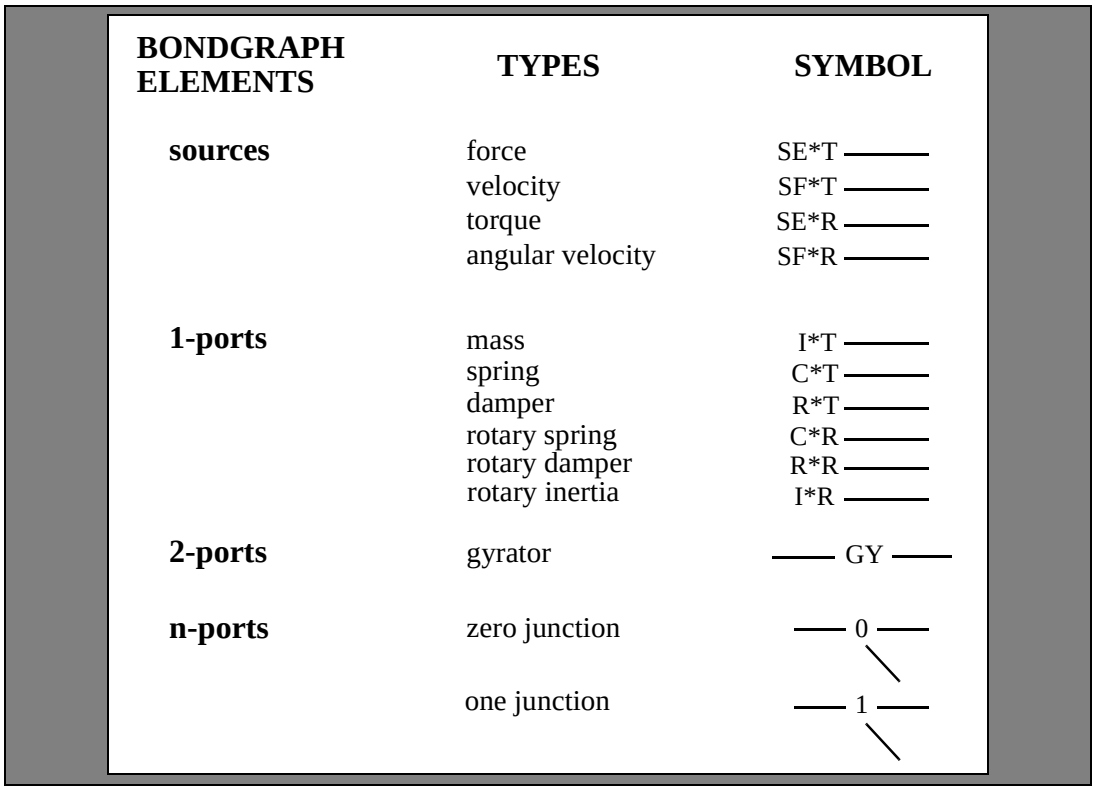


Figure 8.18 Bondgraph elements for simply mechanical objects.

A simple illustration of the use of bondgraphs for representing a mass, spring, damper

system is shown in Fig. 8.19. The spring is moving with a velocity SF*T (at 1), which is

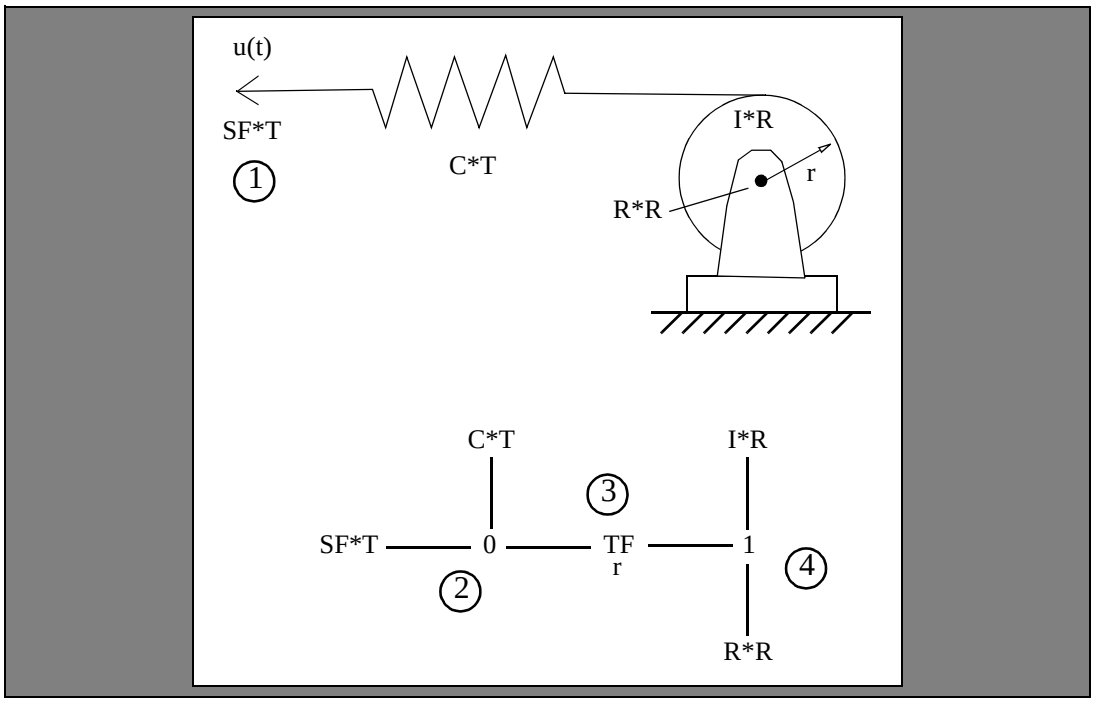


Figure 8.19 Bondgraph representation of mass, spring, damper system.

represented as a source. The spring constant of the spring is represented as a 1-port describing the storage of internal energy, `C*T`. The source and spring 1-port are connected to a zero junction (at 2), as well as to a transformer (2-port, at 3). The zero junction describes the conservation of flow. The transformer is also connected to a one junction (at 4), which is connected to two 1-ports: one representing rotational inertia (`I*R`) of the wheel, and the other representing the rotational damping (`R*R`) of the wheel.

The bondgraph approach is a behavioral analog to FONM machine primitives. The bondgraph can be used to construct an overall behavioral description of the device, by constructing device models using the rules for combining bondgraph segments and knowing something about the desired input/output behavior. The machine primitive decomposes to a behavior representation of the related device function; however, the primitive is intended to be used in constructing devices where the behavioral description is not deemed as important as what the device can do and what goals it can be applied to, so the behavioral aspect of the FONM representation is used for explanation rather than generation. Although the bondgraph approach is appealing in its ability to describe the dynamic relationships between components, it has no model of use embedded in it, so the devices which are constructed can only be compared using structural and behavioral characteristics (i.e., MP-Equivalence).

8.2.7 The Finite Element Method

The Finite Element Method (FEM) is an approach for quantitatively representing complex nonlinear systems as a combination of discrete linear constructs [Clough 1960], where the causal mapping between constructs mandates compliance with engineering mechanics. FONM is similar to FEM in four ways: (1) design of discrete individuals, (2) type of ele-

ment reference, (3) coordinate systems, and (4) definition of behavior, and behavioral consistency/continuity.

Discrete Individual Objects

In FEM, a one dimensional element (such as a beam) may be represented as an idealized column (illustrated as a line) whose ends are called *nodal points*, as shown in Fig. 8.20.

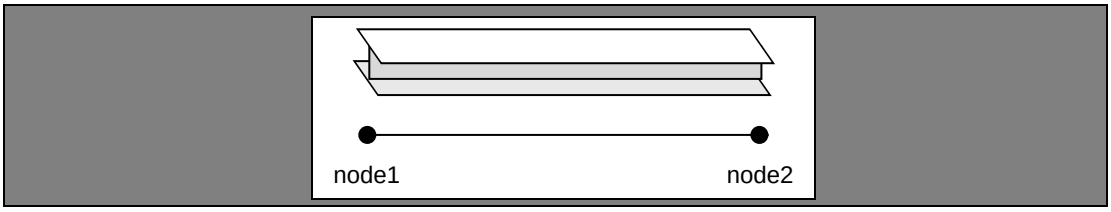


Figure 8.20 A single element and external nodes 1 and 2.

These nodal points, labeled node1 and node2 in the figure, are called *external nodes* because they represent connecting points to adjacent elements [Desai and Abel, 1972]. In the finite element method, behavior of an object is predicted as the combined behavior of the set of discrete elements, and the behavior of each element is effected only at its external nodal locations. FONM object primitives are similar to FEM elements in that (apart from the object CG) only external/observable locations are represented. In this respect, the FONM object primitive can be likened to a qualitative finite element although the granularity is much more coarse.

Nodal Reference System

The nodal displacements, rotations, and/or strains necessary to specify, completely, the deformation of a finite element are defined at the element *degrees of freedom* [Desai and Abel, 1972, page 84]. Element degrees of freedom differ from generalized coordinates in that each is specifically identified with a single nodal point and represents a displacement, rotation, or strain. In the FEM, degrees of freedom at external nodes are referred to as *joint* or *nodal degrees of freedom.* FONM degrees of freedom and FEM nodal degrees of freedom are equivalent.

Local and Global Coordinate Systems

A *local coordinate system* is one that is defined for a particular element and not necessarily for the entire body or structure; the coordinate system for the entire body is called the *global system* [Desai and Abel, 1972, page 88]. The FONM approach is based on a local coordinate system for objects; however, the notion of local is associated with each region (nodal point) rather than with an element, since in FONM an element and an object are one and the same. Object properties are described for an entire individual object. Forces and restraints are local (nodal) effects and are described locally, at object regions, and globally with respect to the entire object.

In FEM, the summation of nodal forces and moments at an arbitrary point (all degrees of freedom) is zero [Archer et al, 1972]. In FONM, the summation of nodal forces is represented as a union of local forces. The forces are resolved into the six Cartesian dimension equivalents used in FONM, and compared to local restraints to determine whether the object can move.

Object Behavior, Consistency and Compatibility

The FONM representation of object behavior is restraint-based. Object displacements and continuity relations can be described by the consolidation of nodal degrees of freedom. Degrees of freedom which are restrained cannot move, while those which are unrestrained are free to move.

FONM and FEM both describe objects as compositions of discrete elements. At some descriptive level, both approaches must agree with a continuous object model. The difference is in the type of model and the granularity of the analysis. *Granularity* refers both to the size of the elements and to the values of the resulting behavior. The finite element approach is a quantitative model, and is distinguished from continuous mechanical models by the discretization of objects into discrete linear entities. As the element size decreases (i.e., the number of elements increases), the model behavioral approximation should approach the closed-form mechanical solution. Although the individual elements may not describe the behavior of the whole very well, their combined effects may. In order to predict object behavior, two requirements are imposed on the combination of finite elements: (1) displacements must be continuous with elements, and (2) displacements must be compatible between adjacent elements [Desai and Abel, 1972].

The FONM approach describes discrete objects; however, they are discrete on a much larger scale than the finite element approach. Although the consistency and compatibility requirements for behavioral process primitives are based on the requirements imposed on finite elements, each object represents what in the FEM would be considered a large set of elements. FONM representations utilize the notion of a finite element to describe sequences of behavior associated with object function. By enforcing continuity between behavioral processes in a sequence, individual behaviors may not describe the function very well (like the discrete FEM element), but the overall behavioral representation is a viable account of the observed function.

For example, the FONM gear object is represented as an object with a center, an axle, and a tooth region which spans its entire outer edge. In FEM, the gear would be represented as many smaller plate elements, as shown in Fig. 8.21 (re-illustrated from Fig. 2.35).

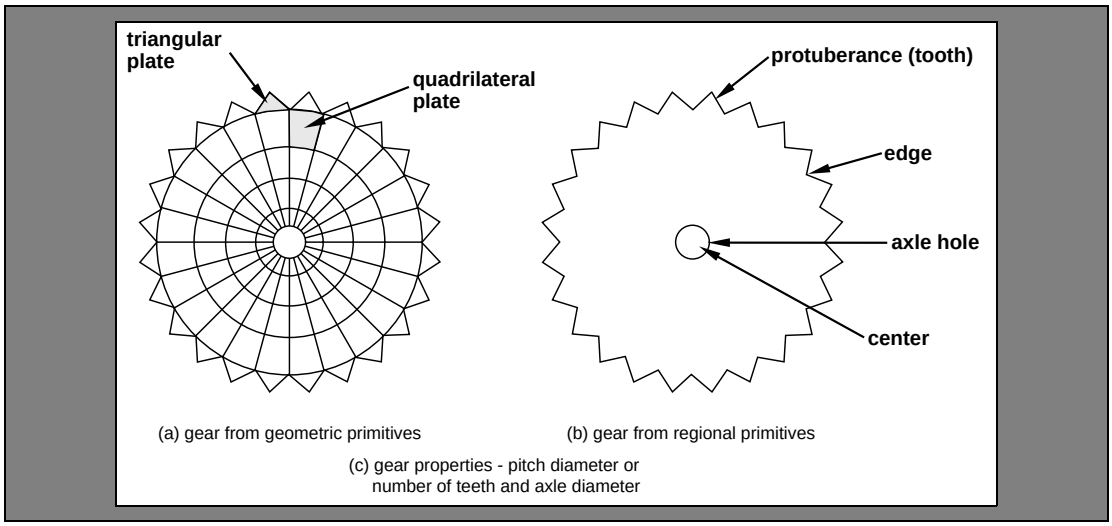


Figure 8.21 Comparison between FEM and FONM gear representations.

In FONM, because of the granularity of the object, only motion type and direction can be determined, whereas in the FEM gear internal stresses and displacements can be determined. Although the results are quite different, the approach is basically the same. Despite the difference in granularity, FONM dynamics is based on the assumption that applied forces and changes in position (FEM nodal displacements) will be continuous between connected objects. The FEM approach has an advantage in being able to predict the internal forces and moments with a high degree of accuracy, but suffers from size and complexity. The simple gear illustrated in Fig. 8.21a has 5 rows of 24 elements, or 120 elements. Each element has 3 or four nodes and 6 degrees of freedom per node, or 2,736 degrees of freedom for one gear. In contrast, the FONM gear in Fig. 8.21b has a single element, 4 regions (the edge is actually represented with two opposing regions), and 6 degrees of freedom per region, or 24 degrees of freedom. This representation size reduction (114:1) is a strength for FONM representations, supporting the representation of larger systems.

8.3 AI Systems for Reasoning about Objects

Two research areas of computational modeling have been particularly relevant in defining the scope of FONM and its demonstration models: (1) design and creativity, and (2) natural language understanding.

8.3.1 Design and Creativity Systems

One aspect of the design process is the effect of design constraints on the creation of an artifact once the constraints have been identified. Sriram proposed a calculus for describing and manipulating constraints in the ALL-RISE model, [Sriram and Maher, 1986]. ALL-RISE is an expert system for preliminary structural design. The system takes as input an architectural building plan, represented as an object (or frame), and the output is a ranked set of feasible alternative structural systems, also represented as objects. Constraint schema represent the causal dependencies between components and influences. There are five constraint types in the system: (1) synthesis constraints, which affect the construction of feasible configurations; (2) interaction constraints, which describe dependencies between components; (3) causal constraints, which describe the stress-strain equations on components; (4) parametric constraints, which describe requirements on component physical characteristics such as strength; and (5) evaluation constraints, which are used to rank alternatives. Constraints are represented as schema which are procedurally attached to structural components. ALL-RISE functions by performing a depth-first search through a hierarchy of predefined structural systems. A structural subsystem is selected and its constraint set is evaluated against the design criteria. When an interaction constraint is found, the affected subsystem is marked. When the current pass is completed, control shifts to evaluating the constraint on the other component, rather that waiting until all constraints on the first component are evaluated. The search is complete when all of the constraints, interaction and otherwise, have been evaluated for all subsystems. The result of the search is a set of systems which satisfy both the individual and combined structural constraints.

ALL-RISE and EDEXP are both based on behavioral object models, both implement objects and constraints as schema, and both utilize procedural attachment during simulation. ALL-RISE implements its behavioral relations as simple expressions, whereas EDEXP implements behavioral relations with frames. The design philosophies differ inasmuch as ALL-RISE identifies systems which meet a specific set of criteria (i.e., constraint satisfaction), while EDEXP identifies the constraints between structural subsystems (i.e., constraint formulation). Neither model implements a complete behavioral model.

JACK is a program which formulates hypotheses about hidden mechanisms within devices based on an input/output (i.e., functional) description of the device [Doyle 1989a,

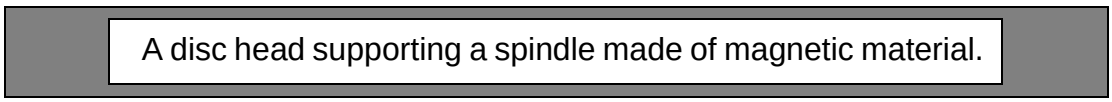
Doyle 1989b]. JACK is able to determine how devices such as a tire pressure gauge, a toaster, and a refrigerator function by generating all the potential combinations of mechanisms which will produce the observed behavior. JACK takes two inputs: (1) a description of a device's externally observable behavior, and (2) a set of mechanisms. The output is a set of plausible combinations of those mechanisms which explain the behavior. The problem is characterized as a graph, where the nodes correspond to device events and the arcs correspond to the mechanisms which produce state changes. The observable behavior represent the edges of the graph, and the solution is a connected graph between edges. Mechanisms are represented as combinations of 9 constraint types: (1) type constraint, which controls quantities; (2) delay constraint, which relates the occurrence of events; (3) sign constraint, which describes value direction; (4) direction constraint, which describes spatial value; (5) magnitude constraint, which defines values of physical quantities; (6) alignment constraint, which describes positioning requirements between components before and after an event; (7) bias constraint, which affects directions of change; (8) displacement constraint, which controls physical location of objects, and (9) medium constraint, which relates object connectivity. Each of these constraints can take on a value or values. Device events are also represented in terms of the same 9 constraints as mechanisms. There are three types of causal interactions in JACK: (1) enablement, (2) disablement, and (3) equilibrium. A mechanism is hypothesized from available events. Once enabled, the constraints associated with the mechanism are propagated to construct a new (i.e., expected) event. The representation in JACK is purely behavioral. There is no particular representation of device structure, nor of device function; only device states and means to produce them. The mechanism representation, being comprised of 9 constraints, has no classification scheme, nor any motivation for why the 9 constraint types were chosen. There is no way to compare two devices which use the same mechanism, or to substitute one device for another based on the mutual ability to effect a mechanism. For example, JACK cannot be used to construct a model for a light bulb switch, even though it can construct a model for a toaster or a refrigerator, which are each comprised of switch elements. This is because JACK is not defined by a set of object, behavior, or function primitives, as in FONM. One limitation this has on JACK's performance is that JACK must enumerate the possible mechanism combinations in order to find a plausible one. A knowledge-based approach, such as FONM, aids in reducing the combinatorics inherent in JACK by classifying behavior and function types and their dependencies. Although JACK isn't capable of determining which of the potential functional descriptions is the correct one, the model is interesting because many non-intuitive functional models may be proposed. JACK can be considered a creativity model, because it could potentially be given the behavioral description for a device that one *wants* to build and then suggest functional models which might aid the designer in implementing the device.

KRITIK is a problem-solving system which designs devices using design knowledge represented as cases and organized in an episodic memory of cases [Goel 1989]. Devices in KRITIK are based on the Functional Representation (structure-behavior-function, or SBF) approach. Cases are represented by combining the models associated with different representation levels (e.g., functional). Similarities in devices are represented in the cases by device functional specifications. KRITIK takes as input a specification of the constraints on a design, and produces as output a structural specification that satisfies the constraints. The problem-solving approach in KRITIK is based on three subtasks: (1) retrieval, (2) adaptation, and (3) storage. The retrieval task matches the design specification to cases in memory and retrieves the most similar one. The adaptation task modifies the retrieved design to satisfy the design constraints. The storage task stores the current design task in memory for future problem-solving scenarios. KRITIK uses an integrated approach to indexing cases for retrieval, based on the goals which are achieved by the design and by the features which are relevant to the goal. The KRITIK system is a useful experiment in design scenarios in which the design constraints are known, and in which successful design cases

are available. It is not intended to, nor can it support general problem solving, for two reasons. First, the representation approach, being based on the intended, or expected, functions of devices does not allow for a device to be used outside of its intended context. The device can only be applied to a design scenario when the context is matched. Second, the representation accounts for all the structure-behavior-function device characteristics, but they are not represented as independent structures. Problem solving thus cannot adapt a plan for achieving a goal, with a device, rather than adapting the device to suit a plan. A general problem-solving model should be able to use the design case, or experience, in such a way that knowledge at any abstraction level can be applied to retrieval and adaptation of designs. Moreover, KRITIK doesn't appear to take into consideration the possibility of combining designs which partially satisfy the design constraints, rather then trying to adapt a single design to the task. The application of knowledge structures such as FONM device-use goals and machine primitives to the KRITIK model could enable it to perform its design task with the flexibility described above.

8.3.2 Natural Language Systems

RESEARCHER is an experimental model for reading, remembering, and generalizing patent abstracts in English [Lebowitz 1985, Wasserman and Lebowitz, 1983]. RESEARCHER represents objects in three ways: (1) by their component hierachies, (2) by the physical and functional relations between parts, and (3) by the properties of parts. The model reads and understands device components and their behaviors. An example of the kind of text that RESEARCHER reads is shown below:



When the description is parsed, information about its components and their physical characteristics, their composition, and of their behavior is resident in memory. RESEARCHER disambiguates the text by using memory access. For example, RESEARCHER must access knowledge about "support" in order know which object is made of magnetic material. Otherwise the reference to composition cannot be disambiguated.

TAILOR is a natural language generation program which has been designed to generate descriptions of the devices analyzed by RESEARCHER. TAILOR is based on the premise that a users knowledge level is important in answering questions they might have about an object description, and that a generation model should be able to choose the right level to answer a question based on knowledge about the user [Paris 1987, Paris 1989]. TAILOR uses information about a user's level of expertise to select and combine discourse strategies for replying to queries about an object. TAILOR uses two discourse strategies: (1) constituency schema, and (2) process trace. Constituency is used when the user is expert about a particular aspect of an object, and then presents detailed information about the object and its characteristics when describing the object. Process trace is used when the user is naive about a particular aspect of an object, and presents a behavioral account of how an object works. For example, consider the text below, which is a description of a telephone generated by TAILOR based on the user knowing how a loudspeaker works. TAILOR selects the constituency strategy for most of the description, but, because the user doesn't know about

transmitters, switches to a process trace for that component:

> The telephone changes soundwaves into soundwaves. The telephone has a various-shaped housing, a transmitter that changes soundwaves into current, a curly-shaped cord, a line, a receiver to change current into soundwaves, and a dialing_mechanism. The transmitter is a microphone. A person speaking into the microphone causes the soundwaves to hit the diaphram of the microphone. The soundwaves hitting the diaphram causes the diaphram to vibrate. The diaphram vibrating causes the current to vary. The current varies like the the intensity varies. The receiver is a loudspeaker with a small aluminum diaphram...

The textual component associated with TAILOR's application of the process trace is highlighted. It follows the portion of the description where the device components are enumerated.

TAILOR selects which strategy to apply based on what it knows about the user's understanding of the object. The user is deemed naive when he/she does not know about the basic concepts associated with the domain, such as how an engine works when describing a tune-up procedure.

RESEARCHER and EDCA are both models which represent knowledge at the structural, compositional, and behavioral levels. RESEARCHER has been extended well beyond what EDCA can currently analyze, because its behavioral model includes more domains than the purely mechanical domain used in descriptions parsed by EDCA. Neither system analyzes devices based on a functional level which is separate from the behavioral level, let alone about object use and experience. The FONM knowledge constructs support representation and description of device function and use, but have not been implemented in that capacity. The TAILOR model points out the need to describe objects at multiple abstraction levels and demonstrates that need with description of objects, such as a telephone, which are pertinent to different levels of expertise. Often a person's level of expertise is lower, even, than the causal, compositional, and structural levels which TAILOR has access to from RESEARCHER. In such cases the discourse strategy should be extended to support descriptions which make use of FONM machine primitive-like representations and device-use goals.

Granacki and Parker, and, more recently, Arens have developed a natural language front end, PHRAN-SPAN for their Advanced Design And Manufacturing (ADAM) system [Granacki and Parker, 1986, Granacki et al, 1989]. Object descriptions are based on a modified Conceptual Dependency representation model, and the domain is digital systems design. For example, PHRAN-SPAN reads the following sentence:

> The cpu transfers the block of data bytes from the disk to the control store.

To parse and understand this sentence, a Design Data Structure (DDS) was introduced to represent design data. The DDS has four components: (1) data flow, which represent data dependencies; (2) timing; (3) structural, which represents the composition of a circuit; and (4) physical, which represents the components and their properties. In addition, five conceptual structures are introduced to represent domain-specific knowledge: (1) information-transfer, (2) temporal-activities, (3) temporal-constraints, (4) control, and (5) declarations. For example, to parse the sentence above, a conceptual structure called VTRANS was proposed:

```
(uni_dir_vtrans
  (source  (a_component *unspecified*))
  (sink    (a_component *unspecified*))
  (info    (df_val command1))
  (control (a_component cpu1)))
```

The `a_component` fillers are unspecified when the concept is initially encountered, and are filled during the processing task. This structure is similar to the FONM BPP, although it is not apparent whether the ADAM project has identified a taxonomy of DDS primitives specific to digital design.

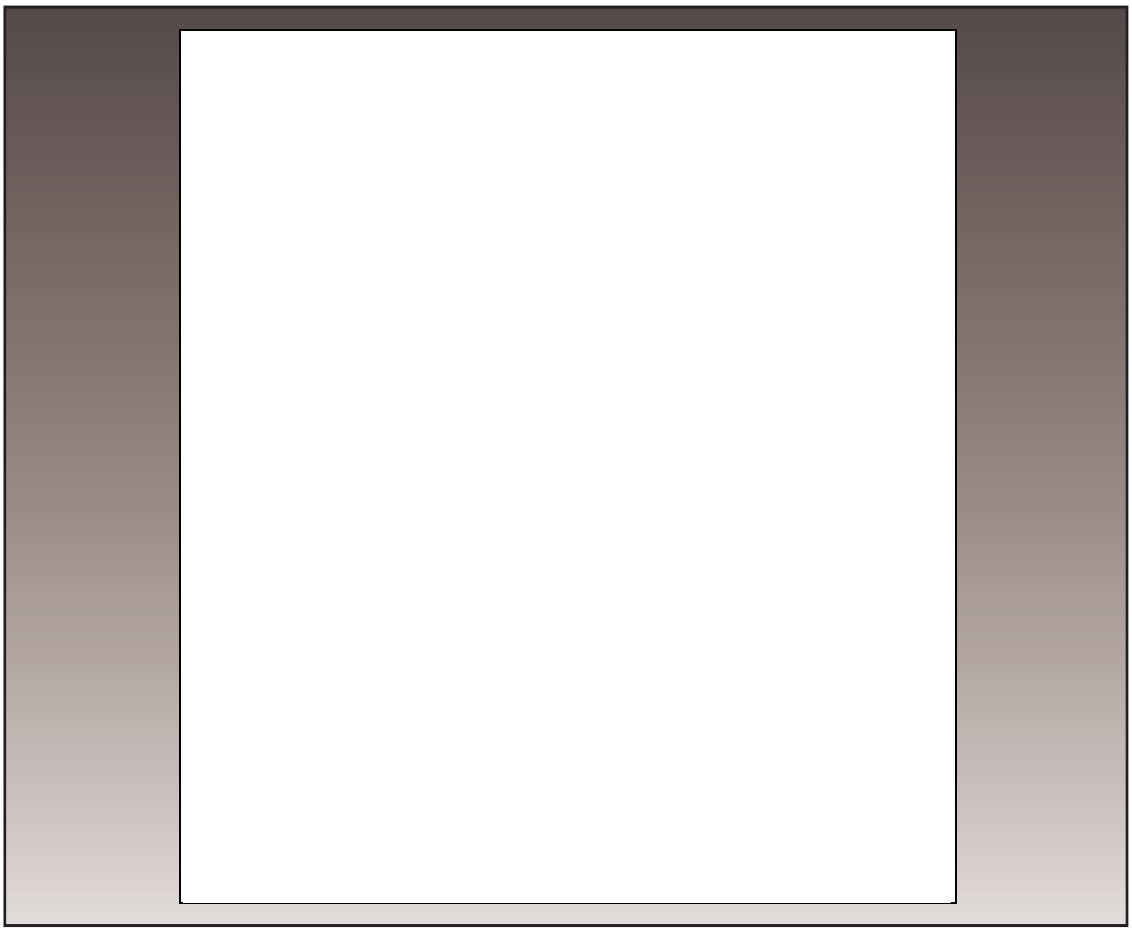
PHRAN reads and analyzes sentences one at a time, and word by word [Arens 1986]. The model represents words as pattern-concept pairs, where the patterns can be phrasal constructs, literals, or syntactic. A word is read, looked up in the lexicon, matched and then retained or eliminated from further consideration. The pattern match can be lexical, semantic, or syntactic. The knowledge domain in which PHRAN-SPAN is used, digital design, is much more dependent on timing relations than the domain of simple mechanical devices. As such, the temporal model used much more extensive than that in FONM. Both models focus on constructing explanations of descriptions which have elements of static and dynamic relations between components. Computationally, the models differ inasmuch as EDCA is a demon parser and PHRAN is a phrasal parser.

Recent work has extended the behavioral approach to language comprehension modeled in RESEARCHER/TAILOR, PHRAN-SPAN, and EDCA to incorporate functional modeling from the KRITIK system. The KA model integrates the acquisition, understanding, and analysis of text design cases using a Functional Representation approach [Pittges et al, 1993]. KA addresses the issues of ambiguity, underspecification, and information relevance at the functional, rather than behavioral, representation level. KA is being designed to take design specifications in English (like PHRAN-SPAN) and to produce a design which meets the specification. The underlying representation uses a structure-behavior-function (SBF) model that specifies how the structure achieves its function (where function is represented as the device purpose). The figure below illustrates the kind of text the KA model is supposed to analyze: One problem with this text is that no reference is made to a

> Consider a flashlight circuit. The function of the circuit is to produce light. The input is a small force on the switch. The output is light of eighteen lumens intensity and blue color.

power source; KA must infer that a circuit has a power source or it cannot function proper-

ly, and so the design must have a power source. KA must also recognize that an "input" generally refers to a function's initialization, that the force is being applied to the switch, and that switch movement results in the closed circuit since no other description, other than the output specification, is provided in the description. Although KA, when combined with KRITIK, is a fully integrated approach to conceptual analysis, it fails at being comprehensive in the same ways that RESEARCHER and EDCA fail, but from a different perspective. As pointed out in the TAILOR system, descriptions of an object must be available at any level of abstraction, and KA only provides a function-level description of the object. By performing the type of analysis that EDCA would perform with access to FONM machine primitives, KA extends the comprehension of objects.



Chapter 9
Limitations of FONM

This chapter describes short term (minor) and long term (major) limitations of FONM and the EDEXP and EDCA demonstration models.

9.1 Representational Limitations

FONM is comprised of three levels of abstraction, each of which has a particular set of assumptions which define its scope and its limitations. The following sections address the limitations of FONM (1) statics, (2) dynamics, and (3) pragmatics representations.

9.1.1 Limitations of FONM Statics

There are six basic limitations to the representation of device statics in FONM:

- Spatial and temporal knowledge are not represented in detail
- Material and behavioral properties chosen are not exhaustive
- No support for property derivatives such as velocity and acceleration
- Complex shapes cannot be represented with regions
- Geometric primitive classification is redundant
- Object primitives constrain behavioral reasoning

There is no particular representation of space in the model. An adhoc symbolic method for describing relative position and value based on scales was used for the purposes of parsing representations into illustrations; however, this method was not intended to be robust because spatial analysis was not intended to be the focus of attention. The same approach was used to represent dimension and direction. As a result, FONM static object representations cannot represent position or direction well enough to perform detailed spatial simulations. The same limitations are true to a greater extent with temporal knowledge. When an event actually occurs, and its duration, are not modeled in any way in FONM, so the only temporal inferences which are supported are those which are either explicitly sequential or explicitly simultaneous.

The material properties chosen to represent objects are not claimed to be an exhaustive set of properties, but to represent the types of properties which can be used to support some behavioral inferences about what an object can do. For example, the stiffness of an object helps to determine whether it can transmit force. But the material properties do not support any notion of state change, such as from ice to water to steam, or the tensioning of rope by twisting, which dramatically affect the mechanical properties of an object. For example, there is no mechanism in the FONM approach for the idea that, if one twists rope a lot, it has some capacity to transmit forces in the `NEG` direction (i.e., one can push on rope if it is wound tightly enough).

The behavioral properties which have been selected to describe object behavior (position, restraint, applied force, internal force, and size) support the recognition and prediction of simple mechanical behaviors. They do not represent an exhaustive list of behavioral properties for all physical domains, nor even in the mechanical domain. For example, the notion of internal force is used as an abstraction to describe energy absorption, which is not explicitly true. As an approximation it is adequate at the representation level being described because, regardless of its representation as force or energy its effect, both in FONM dynamics and physically, is the same.

A fundamental limitation of the FONM object representations is lack of support for property derivatives. The qualitative reasoning community (e.g., Forbus, DeKleer and Brown, Kuipers, etc.) have developed calculi for symbolically representing velocity, acceleration, momentum, conservation of energy and mass, and so on which are not represented with FONM. This limitation makes it impossible to use FONM for any transient analysis. Since much of dynamics is associated with transient behavior, particularly for diagnosing machine and system faults, this limitation is problematic. All of these limitations are based on the granularity of the FONM statics. Each of these problems could be resolved by extending the scope of the static object model, because the dependencies between object statics, dynamics, and pragmatics are distinct in FONM.

The FONM region taxonomy was chosen to simplify the representation of shape and the number of locations on an object that needed to be explicitly represented. As such, the size of representations is reduced, but the type of simulations that can be performed using this kind of spatial model is compromised. For example, a FONM gear object has only four regions: a center (or rotation), an axle, and a tooth region on the outer edge (actually represented with two opposing regions for consistency with cylinder, wheel, etc.). Representing the gear with four regions enables representation of force transmission and magnification, direction of motion, and relative magnitude. It does not enable the representation of position along the edge, since the entire edge is represented with a single region. As a result, the FONM gear object cannot be used to simulate a timing sequence in a clock beyond half gear revolutions, because position of one tooth to another is not represented. The FONM object can support diagnosis of malfunction in a clock. Because gear contact is required for force transmission, it could be inferred that malfunction at the gear level was caused by a problem with the tooth region.

Regions also do not directly support the representation of truly complex shapes. For example, the region taxonomy allows the representation of a nutcracker jaw as a tooth region, where each tooth is represented as a wedge-shaped protuberance. Each protuberance has the capacity to restrain motion, and the wedge shape supports the possibility of cutting, but there is currently no mechanism to describe many teeth in a jaw or many teeth on a gear. The model simply makes the assumption that the entire region has the same behavioral properties as a single entity (in this case a tooth), and so the area encompassed by the region need not be represented explicitly in any other capacity.

The FONM object classification is a superset of volumetric primitives, selected to provide a basic set of uncomplicated shaped objects. The representation of the geometric primitives is limited to four characteristics: (1) center of gravity, (2) observable regions, (3) relative sizes in different dimensions, and (4) shape viewed along dimensional axes. The taxonomy of geometric primitives chosen is clearly redundant, since there are a number of ways to describe the same object and many of the objects are specializations of other objects. The filament and film, for example, are specializations of column and plate, which, in turn, are specializations of cylinder and block, which are true volumetric primitives. The limitations in this approach are the number of the objects in the set and how that number affects processing performance, though no specific processing based on geometric primitives is currently done in models which use FONM representations.

The notion of object primitive is used to describe an object which instantiates a machine primitive, and is not intended to take on a particular shape. The appearance of the object primitives was chosen to identify the required regions of the object, rather than its shape. By representing object primitives only in terms of those regions required to describe the FONM behavioral primitives and machine primitives, the approach is limited in the types of behavior which can be simulated. For example, a gear can be instantiated in a MOTION process, but since all gear teeth are represented as a single region of the object primitive, the inferences which can be drawn about a particular gear tooth are limited to the direction of motion with respect to the center and with respect to another point on the other side of the center. Detailed simulations are not supported by this description of objects. On the other hand, this level of description eliminates large amounts of information which are not required for a functional level of analysis.

9.1.2 Limitations in FONM Dynamics

The FONM notion of object dynamics is based on the assumption that it is important from an object use point of view to know how a behavior is enabled, and what type of effect it produces, but not necessarily how it is produced or to support its simulation. This assumption is based on the cognitive work of McCloskey [McCloskey 1983] and DiSessa [DiSessa 1983], which have shown that naive reasoners are able to recognize low-level behavior but are not able to predict low-level object behavior. Within this scope, FONM dynamic representations suffer from limitations of six kinds:

- BPPs do not describe all dynamic behavior
- BPPs granularity is coarse
- Complex behaviors not represented
- MPs are not true dynamic primitives

The motion-restrain-transform-store-deform BPP taxonomy is only valid for the intended scope of the representation and is not, otherwise, complete. For example, it is not possible to represent the momentum of objects using FONM, even though force, mass, and position can be represented (to some extent). As such, object collisions cannot be effectively modeled using FONM dynamics. There are two reasons. First, position rates of change are not directly represented. Second, the interaction of mass and other properties has not been addressed in FONM. The recognition that an object is in motion implies that the object has a velocity, so it must have a momentum, but FONM has no structure which represents it. One could claim that, at some level, momentum has the same type of effect as internal stored energy. Then it could be represented as an internal force and its behavior could be represented with the STORE process. This would require a rethinking of the role of STORE and its interactions with the TRANSFORM and DEFORM processes, and has not been attempted in this research.

The claim of primitives in FONM is based only on mechanical behavior, and not on behavior in other physical domains, despite the fact that many mechanical behaviors have analogs in other physical domains, and despite the fact that mechanical devices are often used in conjunction with other physical domains. For example, the RESTRAIN process is used to represent interactions which affect object motion mechanically, and could potentially be used to represent resistance in electrical circuits and viscosity in fluid mechanics. Likewise, the STORE process is used to represent internal energy in mechanical objects, and could potentially be used to represent capacitance in electrical circuits. In order to make claims about the general applicability of FONM behavioral primitives, each physical domain would have to have 5 behavioral classes which were analogs with the 5 processes in FONM, and their structures would have to be the same as the structure of behavioral processes proposed in FONM.

This ‘level of representation’ assumption, which defines the scope of low-level dynamics in FONM, also limits the applicability of FONM representations in mechanical diagnosis and simulation. The FONM representation of behavioral process identifies those aspects of a device’s static representation which either enable a form of behavior or are changed as a result of that behavior, but the behavioral model does not specify how much change is produced nor does it represent the actual change. Thus, the FONM MOTION process can be recognized as being enabled, but it will only be possible to determine what kind of motion the object is engaged in, what the dimension and direction of its motion are, and the type of path it is on.

The behavioral primitives proposed in FONM have specialization hierarchies (e.g., Figs. 3.6, 3.11) which illustrate exemplars for their class of mechanical behavior. Examples which have been presented in this dissertation describe the basic specializations of the class but not complex behaviors which can be represented by combining them. For example, the FONM representation for BPP-MOTION was presented with basic specializations for translation, falling, sliding, rotation, rolling, and swinging. It is also possible to combine these motion types to describe complex motions, such as bouncing, which is a combination of falling, translating, and spinning. This type of combination has not been performed in this research.

Machine primitives represent primitive object/function relationships. Their abstraction layer in FONM enables access to how devices are used in a plan, and to how devices behave. They are not true representation primitives because they do not form the lowest inference level in FONM, which is represented with processes. The taxonomy of machine primitives has been chosen to encompass the true machines (lever and inclined plane), the primitive mechanisms as defined by the U.S. Navy [Navy 1972], and those which have been motivated by adhoc protocol studies. There is no evidence to support the particular choice of machine primitives, nor their organization in Fig. 4.4.

FONM dynamic representations can make the task of identifying role associations between objects difficult when the objects have a similar functional capacity but are applied in different ways than their static representations would suggest. Consider a broken fan belt, which is used as a component of the pulley-object, a confinement device. It has been suggested in [Rubinstein 1986] that a silk tie might be used for a makeshift fan belt, because of its elasticity, size, and shape. The tie ends would have to be tied, to form a belt of the right size, and the knot would have to be very tight (i.e., small) to fit the pulley channel. If a reasoning model makes comparisons between devices at the function level, then there is a possibility that the tie will be considered a potential replacement because of its role as confinement device. The tie confines the shirt collar and neck around which it is tied. But there is no mapping of regions between the fan belt and the tie when they are applied in this manner, so the tie would not be considered further. If the function level is bypassed when making comparisons, then a comparison could be made at the static level, and the tie could be tried on that basis, but so could many other objects which are not appropriate to the situation. In FONM, device-use plans are more closely associated with device functions than with device structure, so a model which made use of FONM representations might not be able to make the correspondence between the fan belt and the tie.

9.1.3 Limitations in FONM Pragmatics

The FONM representation of device pragmatics does not address two important issues which must be addressed by a problem-solving model:

- Device malfunction and misuse
- Property measurement in problem solving

Currently there is no representation for damage (such as a broken gear tooth) in FONM,

only disabled behavior. If two gears must be in gear contact in order to function, and the gear function fails, then one of the inferences that can be made from a FONM representation is that gear to gear tooth contact has been disabled. Since a protuberance is required for gear contact, it could be inferred that the protuberance no longer exists, but no such inference is supported in any of the demonstration models which have been constructed in this research, nor is there an explicit representation for the broken tooth. This is a limitation to FONM pragmatics because most problem solving situations provide the problem solver with perceptual information about the objects on hand, which must then be interpreted with respect to knowledge about how they should function/behave. This type of analysis cannot be performed unless the representation contains information about possible failure modes in terms of geometry changes to the objects.

Property Measurement and Problem Solving

A problem solver may know that to open a varnish can, the prying object chosen must fit the can slot. He can satisfy this requirement by: (1) measuring each object before trying it, (2) trying only objects which he already knows will fit the slot, or (3) trying an object and see if it fits. He is more likely to try using the object first, than to measure its size or the sizes of every potential pry object, simply because in trying the object he is effectively measuring it. Naive problem solving is based on experience and experimentation. An object's property values play a greater role in planning failure and learning than they do in predicting whether the object is suitable for a task. The construction of problem-solving models requires a representation which associates the process of experimentation with an object, its function, and the goals which its application either achieves or fails to achieve.

9.2 Limitations of the EDEXP Demonstration Model

The EDEXP program simulates simple door mutation and creation through the relocation of a hinge, recognizes whether new doors can rotate, and constructs a rule for locating hinges that controls door rotation. The model has four limitations:

- limited application of planner
- limited use of mutation and problem-solving heuristics
- only static graphics were implemented
- limited device and behavioral representations

Although EDEXP was designed with a general recursive descent planner and problem solver, the demonstrator only addressed a single type of goal, used only one plan (MUTATE), and one metric (MOVE COMPONENT). Replanning always used the same metric, so the demonstration model never took into account the effect of moving more than one hinge, or moving another component (like the doorknob). The hinge was not allowed to be moved to a location on the doorslab at all, which would have been interesting. Selecting another object (e.g., the doorknob), or another metric (e.g., increase the size of a hinge, or change the metal hinge into a rubber hinge) would also have been interesting. The plans are not implemented declaratively, so the planner executes them procedurally when their use is indicated.

The original intent of EDEXP was to devise a computational model capable of retrieving, combining, and modifying device representations toward specific design and invention goals (e.g., design a door for dog entry and exit). Except for the MOVE COMPONENT metric used in EDEXP, problem-solving and invention heuristics identified in the EDISON project were not implemented.

A simulation is a much better aid to user comprehension when it has a graphical interface. EDEXP had an interface which translated the current device representation into a

graphical display using icons. When a new device was created by the program, the illustration changed, so that the user could see what was going on and compare it to the demons which were firing during the simulation. The EDEXP graphical interface was not a dynamic simulation. It simply displayed the current device, and the interface was never extended to graphically display behavioral or functional simulations. Recently a number of graphics packages which simulate device behavior have been developed (e.g., Interactive Physics). The innovative idea is the object-oriented approach to constructing the device and having them interact properly. Although these packages are based on quantitative, rather than qualitative, behavioral models, they underscore the usefulness of graphics in aiding users comprehension of device mechanics.

EDEXP would never have been developed if a general model of all the knowledge needed in mechanical experimentation was necessary before constructing a model. Only those structures which were directly applicable to reasoning about door function were described in detail, while others were simplified to account for their later inclusion in FONM. For example, in EDEXP the basic notions of motion and restraint are fairly well defined, but the structure and taxonomy of behavioral primitives were not yet fully developed. The notions of goals and plans as device pragmatics, and their role in the experimentation process, were not yet developed at all, even though the planner was based on goals and plans.

9.3 Limitations of the EDCA Demonstration Model

The EDCA program parses English descriptions of mechanical device behaviors underlying the functions of a toy dart gun, a press, and a door. The model has four limitations:

- No object statics or machine primitives are used
- No means for disambiguating independent multiple object references
- No reasoning support
- No question/answering or generation
- No learning support

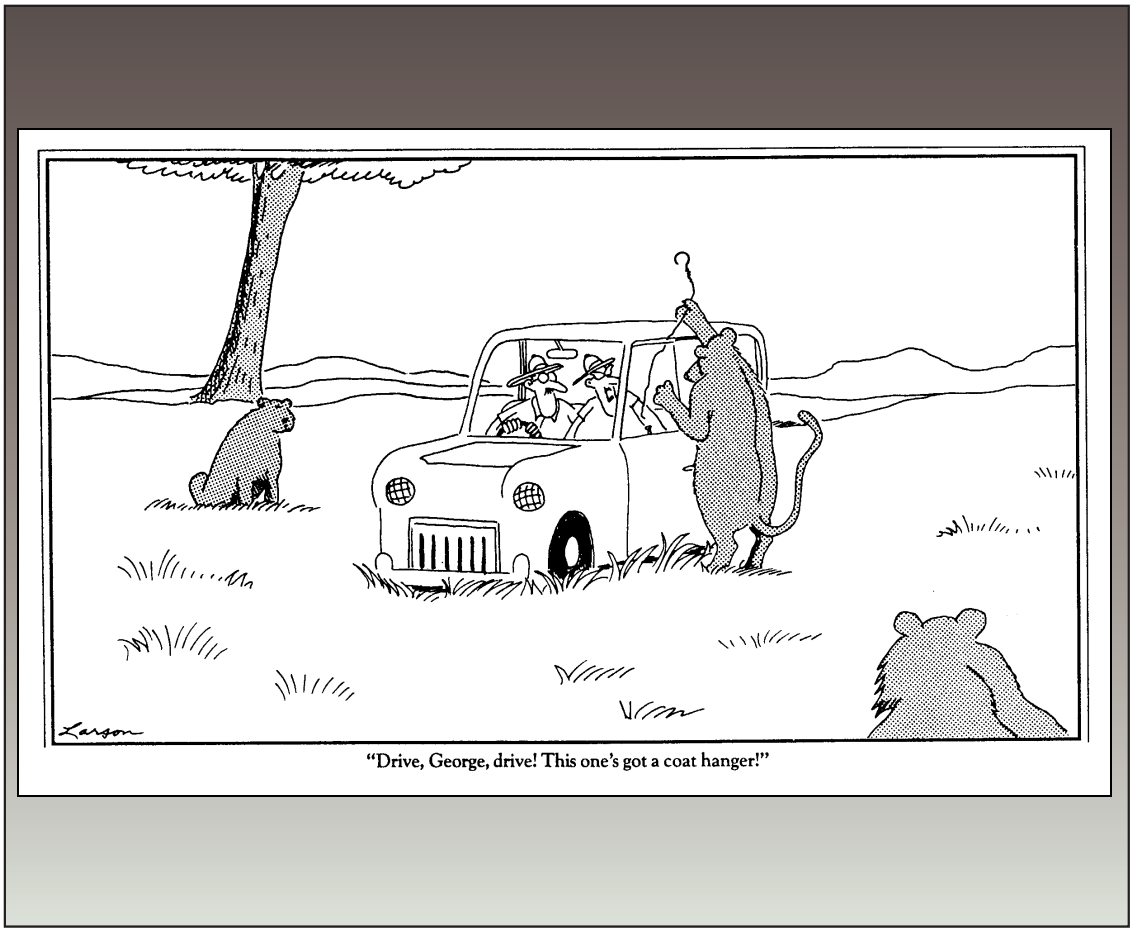
The EDCA model implements the FONM behavioral primitives, but neither the object static descriptions nor the machine primitives. The model can infer the presence of a behavioral primitive by finding the states which enable it or result from it, but not from the properties or regions of the associated objects or from recognizing an object and using its function to help infer its behavior. This is particularly evident in the parsing of Toy Gun, when the object reference for “it” cannot be disambiguated. With a memory which includes different toy dart guns, this reference could possibly be understood.

EDCA can disambiguate multiple references to a single object; however, if the same word is used for different objects, such as the gears in a transmission, or the screws used to connect a hinge to a door, then EDCA will erroneously associate all of them to a single object.

Although reasoning is used to construct the conceptual representation of the description, EDCA doesn’t support any reasoning about the device knowledge constructed by the conceptual analyzer. Questions cannot be asked and then traced through the representation in either direction.

There is no support for question-answering or generation in the EDCA implementation.

Although a primary use of a natural language model capable of reading and comprehending device descriptions would be to develop a device library from how-to books, no learning mechanism was ever implemented in EDCA.



"Drive, George, drive! This one's got a coat hanger!"

Chapter 10
Extensions, Applications and Conclusions

This chapter is organized as follows. First, the status of FONM and the implementation status of demonstration models are presented. Second, the future work necessary to extend the FONM approach to support EDISON project goals is discussed and possible approaches are presented. Third, potential applications based on FONM and its process models are discussed. Fourth, a summary of contributions and significance of this research are discussed.

10.1 Status of FONM and its Process Models

FONM independently supports the two processing tasks presented in Chapters 6, 7: (1) simple mechanical experimentation, [Dyer et al, 1986], (2) comprehension of three short behavioral descriptions [Dyer et al, 1987], (3) recognition of low-level MOTION, RESTRAIN, and TRANSFORM processes and their specializations [Hodges, 1988], and (4) hand coded solutions to three improvisation scenarios [Hodges, 1989, 1992a, Hodges et al, 1992b]. The approach currently supports a library of 19 device statics, dynamics, and pragmatics representations in four device classes. The parser lexicon currently consists of 75 entries for the three device descriptions currently parsed. There are 460 demons in the EDCA, EDEXP, and related models combined.

10.2 Future Directions

Full FONM device representations have not been implemented in a single process model which can perform the separate reasoning tasks illustrated in the EDEXP and EDCA demonstration programs, nor have the reasoning tasks associated with these models been significantly developed. The development of such a model would enable researchers to compare simulations using FONM representations to data gathered in experiments on human subjects. The results could be used to determine its suitability for representing the knowledge associated with naive mechanics. By constructing an integrated reasoning model, it could be determined to what extent the approach can be merged with lower level and higher level representation approaches, and to what extent the approach supports simulation of device behavior, function and use.

The first step toward constructing an integrated processing model would be to develop the demonstration programs into systems that utilize the full representation and perform the appropriate reasoning tasks. EDEXP could be extended to utilize both mutation and combination plans when experimenting, instead of the mutation plan alone. The mutation types used could be extended to include all the FONM state types (material, position, restraint, applied force, internal force, size and shape). The model would have to be updated to associate its experimentation with the behavioral and mechanical goals proposed in Chapter 5. Those goals in turn would have to be associated with higher level metrics for making choices between

plans. For example, we developed a set of six invention metrics early in the EDISON project which were never implemented in EDEXP: (1) simplicity, (2) efficiency, (3) performance, (4) cost, (5) novelty, and (6) elegance. The application of these problem-solving metrics, combined with a planner with access to FONM device representations, and based on mechanical goals and plans would make a significant improvement in the program.

Currently no FONM-based model performs top-down simulation at either the behavioral or functional level. The EDEXP model could be extended by incorporating a perturbation and inference mechanism which would support top-down prediction of behavior. Using the same method in reverse, an explanation component could be developed.

EDCA could be extended by incorporating object statics, machine primitives, and device pragmatics into the lexicon and into the disambiguation and reasoning demons. These changes would enable EDCA to utilize both top-down and bottom-up reasoning during conceptual analysis. The efforts of Pittges et al to include functional reasoning capabilities into language comprehension models illustrates the need for this type of extension [Pittges et al, 1993], and the generation of device descriptions based on user expertise [Paris 1985, Paris 1987] illustrate the need to represent and reason about device descriptions over a broad spectrum, from very naive to expert. The FONM representation of function, and its representational distinction from goals and plans makes it a logical choice for this type of language extension. Another extension would be to develop a question-answering component to the system.

An improvisation model using FONM device pragmatics has never been implemented as a computational model, even though hand coded situations have been represented [Hodges 1989, Hodges 1992, Hodges et al, 1992]. The general idea is to analyze a situation by identifying potential goal failures and then associating them with affected plans and devices which can be used in those plans. An important capability for this type of program would be to utilize entire experiences organized in memory to look for similar goals, plans and devices, and to reincorporate the current scenario back into memory when problem solving is complete. The implementation of this model would require a memory model and a theory for assimilating new experiences into memory. I have done some preliminary work on organizing experiences in memory by the goals which are achieved or failed and by the devices which are used [Hodges 1989]. The model should also be capable of associating naive experimentation with device characteristics.

10.3 Potential Applications of FONM

A number of applications of FONM have been identified by constructing process models for each of the two processing tasks demonstrated: (1) experimentation and (2) conceptual analysis. The potential application of a representation approach which can support general reasoning about mechanical devices is limitless; however, a few specific applications have been identified below.

10.3.1 Experimentation and Invention Applications

Device experimentation could allow an intelligent system to learn about the behavior of new devices without assistance, or acquire knowledge through experimentation. Device invention could be used to rethink design scenarios and support the solution of problems remote to human access (e.g., Hubble telescope and Mars satellite problems). A design understanding and invention system capable of reasoning and brainstorming about physical devices, their functions and behavior, and capable of communicating that reasoning to a designer.

10.3.2 Simulation Applications

The simulation of object behavior has broad practical use for helping people to understand how objects function and behave. Two applications can readily be discerned: (1) machine diagnostic analysis, and (2) inferring and learning device mechanisms.

A mechanical reasoning system could analyze device behavior and function for arbitrarily complex machines. It could be used to simulate and diagnose behavior, or suggest ways to find out what is wrong with a disfunctional component or machine.

A system that has access to knowledge about mechanisms and the objects which effect them could be used to recognize the components and mechanisms of new objects from a description of their I/O behavior.

10.3.3 Language Processing Applications

Device comprehension could lead to the implementation of systems capable of communicating knowledge about device behavior and function to humans. Two examples are: (1) tutoring systems, and (2) database generation systems.

On-line tutors could replace written manuals. Manuals are difficult to read because they require the reader to supply necessary index keys, and manuals are only able to answer anticipated questions and rarely completely enough for naive reasoners. A system capable of behavioral and functional reasoning about devices could answer questions about devices and answer 'why' questions at a level understandable by the naive mechanic.

With the capability to understand natural language, a system could take descriptions from a designer or a text and interpret them (i.e., generate a conceptual representation). This representation could then be indexed into the conceptual database for later use, specifically for the types of applications already described.

10.3.4 Improvisation Applications

Improvisation is a capability well suited to people or machines placed in new and unfamiliar environments, or highly constrained situations. A system capable of interpreting object function based on context could be applied to application types requiring horizontal reasoning, such as conceptual design. A system that could reason about object use in context could be used to suggest ideas during design formulation. Given a particular status of a project, and a memory of other projects, the system could compare the new goals and constraints against earlier designs. This could lead to the resolution of design 'ruts,' and perhaps to new approaches to known problems.

10.4 Summary and Conclusions

FONM is a representation theory for describing simple mechanical devices and supporting the kinds of reasoning associated with naive mechanics. The approach to developing FONM representation structures was to: (1) identify the knowledge that is associated with day-to-day mechanical reasoning, (2) develop structures and processes which describe those knowledge types, and (3) show how those structures and processes are used in various tasks requiring mechanical knowledge.

To create a computer program that models a naive mechanic's understanding of mechanical devices for use in problem-solving tasks, the program has to implement four knowledge theories:

- A theory of machines.
- A theory of function and behavior.
- A theory of device interpretation and use.
- A theory of reasoning and inference about device behavior, function, and use appropriate to the processing task.

This dissertation has presented a theory for representing mechanical device structure, behavior, function, and use. The FONM model of naive mechanics has addressed these requirements in six ways:

- **Abstraction levels**: 4 abstraction levels for representing mechanical devices have been identified.
- **Device statics**: Knowledge structures and taxonomies were introduced for representing device characteristics from which mechanical device components and configurations can be described.
- **Low-level device dynamics**: A behavioral process primitive knowledge structure and a taxonomy of 5 behavioral primitives were introduced for representing complex behavior and function.
- **High-level device dynamics**: A machine primitive knowledge structure and a taxonomy of 11 machine primitives were introduced for representing complex mechanical devices.
- **Device pragmatics**: Taxonomies for goals and plans which are specific to mechanical device use

and application were introduced.
- **Knowledge dependencies**: The knowledge dependencies which support reasoning between the representation levels were identified.

At each abstraction level, the knowledge necessary to describe an object and its causal interactions was identified, a canonical structure for representing the knowledge was developed, and a taxonomy of related knowledge structures was proposed. The usefulness of FONM representations was shown by constructing 2 experimental process models which demonstrate the application of these knowledge structures on reasoning tasks associated with mechanical problem solving. Although the community is divided on the definition of function, there is widespread agreement on the use of abstraction levels and which levels are useful in representing and applying device function (e.g., [Chittaro et al, 1993, Lind et al, 1993, Qian and Gero, 1993]).

Introduction of Device Regions

The introduction of device region and its taxonomy in Chapter 2 provides a mechanism for simplifying the geometric description of devices and for recognizing device function. Regions provide for a means to reduce the size and complexity of device representations because only those regions which are associated with machine primitives are identified in the device's physical description. Regions provide a means for reducing processing effort by providing a metric for describing object appearance along with object function and behavior. Researchers in the area of artificial vision are now beginning to use regions to help identify objects (e.g., [Rivlin et al, 1993, Stark and Bowyer, 1990]).

Introduction of BPPs and MPs

The notion of a behavioral process is not a new concept and has been used since the inception of naive physics [Hayes 1978, Forbus 1984, DeKleer and Brown, 1984, Kuipers 1984]. However, the FONM behavioral process primitive proposed in Chapter 3 captures a naive reasoning level while retaining much of the structure and content of existing approaches. In addition, FONM a taxonomy of behavioral primitives is proposed. The taxonomy results in an overall savings both representationally and computationally: the small number of structures which can represent a behavior means that there are fewer unique structures to keep track of, and the small number of primitives means that there are fewer unique inferences that can be made.

The notion of a function-based knowledge structure is relatively recent in qualitative reasoning (e.g., [Dyer et al, 1986, Sembugamoorthy and Chandrasekaran, 1986, Ulrich and Seering, 1987, Lind 1990]). In addition to the representational and computational benefits of behavioral primitives, Machine primitives provide a mechanism for constructing device models based on a skeleton which is causally related to, and providing access to, device use and behavior through the common device and its structure. The taxonomy of machine primitives proposed in Chapter 4 provides a finite set of structures from which to compose representations of mechanical device function.

Introduction of Device-related Goals and Plans

Other researchers have addressed the issue of how humans relate to objects and how objects are used to achieve goals [Lehnert 1978, Rieger 1978]. FONM contains a set of device-specific goals and plans (Chapter 5) which directly relate intention with device function and behavior. In each, a taxonomy of the related knowledge structure was introduced.

Combination of Object Use and Function

Previous approaches to representing and reasoning about devices have addressed the issue of interpreting object use and function; however, FONM represents relationships between device use and function, in Chapter 5, which are consistent with (a) how devices are used by humans, and (b) how devices behave and function. Recently there has been interest in this level of reasoning within the functional modeling and design community, with an emphasis on identifying how device function relates to how the device is inter-

preted by its user (e.g., [Qian and Gero, 1993, Kannapan, 1993]).

Appendix
Mechanical Device Representations

This appendix is devoted to an in-depth presentation of the static, dynamic, and pragmatic representations for each of the devices that have been used in this research. Fig. A.1 illustrates the four mechanical device classes which are presented in this appendix: (1) simple devices, (2) simple compound devices, (3) multiple compound devices, and (4) complex devices

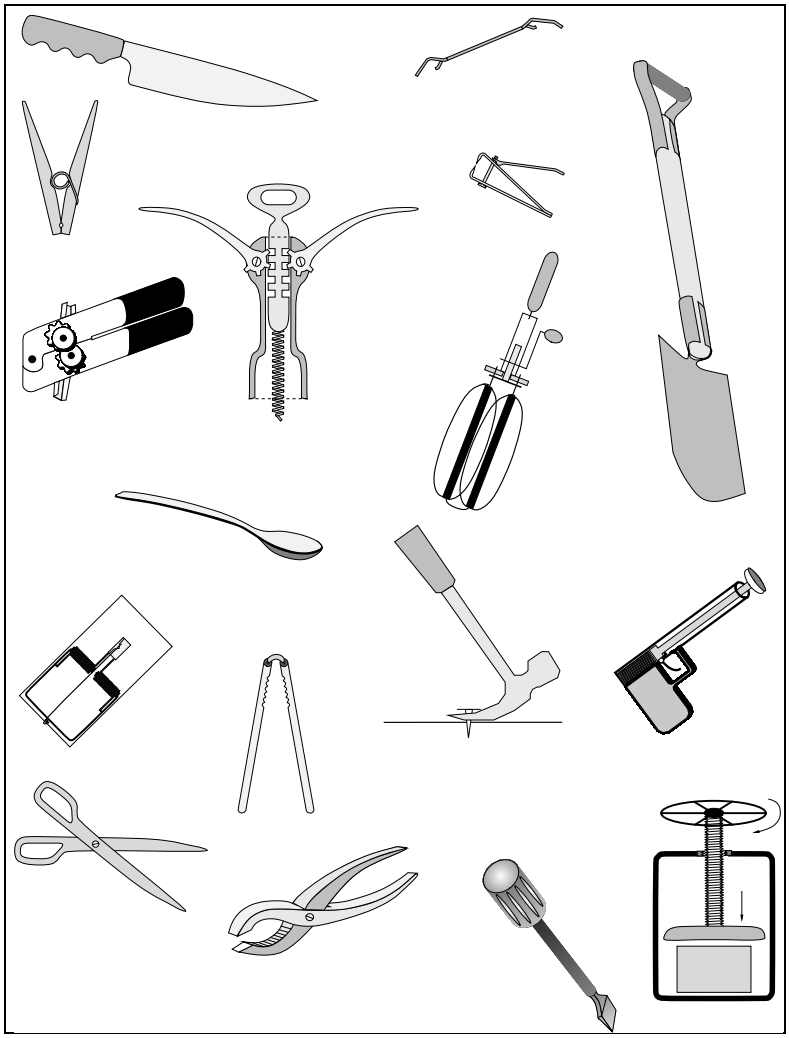


Figure A.1 FONM Devices.

A.1 Simple Devices

Simple device dynamics are representationally significant because even simple devices can be used to accomplish multiple tasks. The dynamic representation must make a device's different functions explicit with the same static representation. The distinction between device functions is often seen as a specialization on the type of input applied to a device, or

on the physical properties of objects which comprise the device. Simple devices are also useful for showing that device descriptions which represent the same function, whether based on components or regions, are equivalent. The simple devices presented in this section all consist of components which are rigidly connected, or regions on a single component. As in Section 2.7.1, five devices will be addressed in this section:

(1) carving knife
(2) hammer
(3) shovel
(4) spoon

A.1.1 Carving Knife

A carving knife is designed for slicing and puncturing. One end is blunt and used as a handle, while the other is sharp and pointed.

Carving Knife Statics

A knife is comprised of a handle and a blade. The blade has an edge and a pointed tip, as illustrated in Fig. A.2 below.

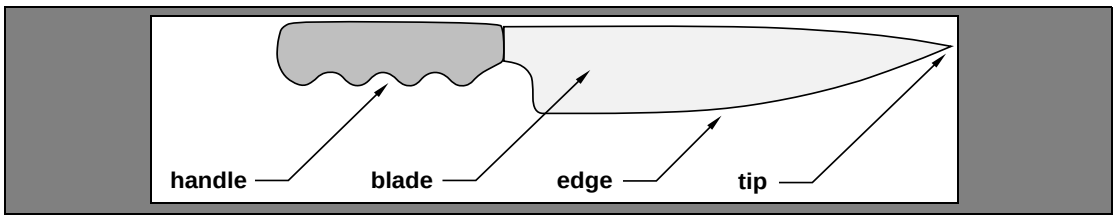


Figure A.2 Carving knife illustration.

An idealized carving knife and its static device diagram are illustrated in Fig. A.3.

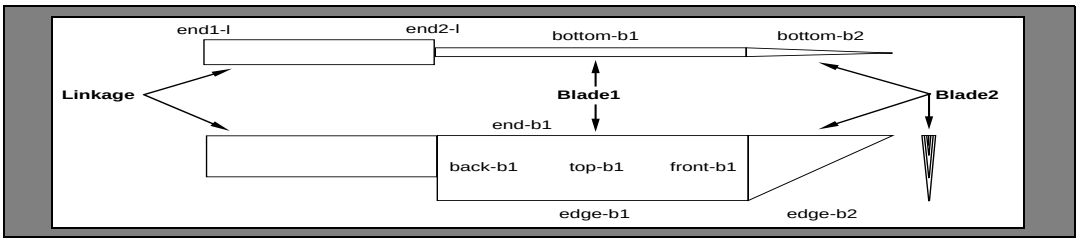


Figure A.3 Carving knife object primitives and static device diagram.

The linkage-object is rigidly connected to the wedge-shaped blade-object, which is itself rigidly connected to the tetrahedron-shaped blade-object. The SDD shown in Fig. A.4

illustrates these connections.

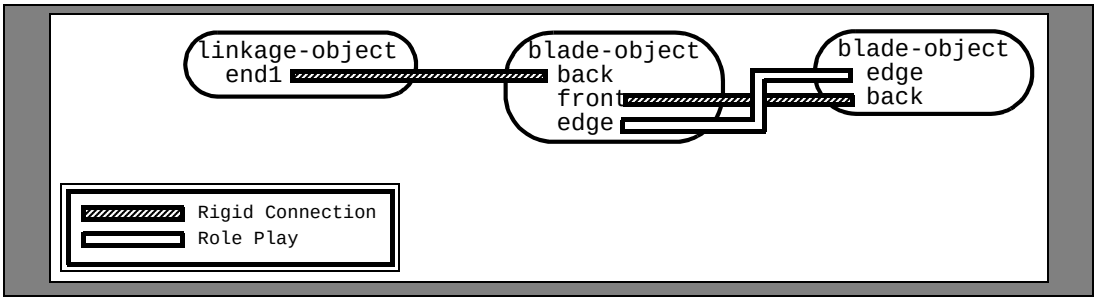


Figure A.4 Carving knife static representation diagram.

The two blade-object edges are actually the same edge, so there is a role-play link between the two objects. Role-play links identify which frame roles are related between representation schema, so that inferences about connectivity and force transmission are continuous. The carving knife objects share the colinear orientation of objects that the bottle opener has, so their representation will be skipped here.

Low-Level Carving Knife Dynamics

The behavioral sequences for knife components in the cutting function are shown in Figs. A.5, A.6. Only one of the blade sequences is shown. Note the similarities between the knife handle and the screwdriver handles, and between the can-opener blade and the knife blade..

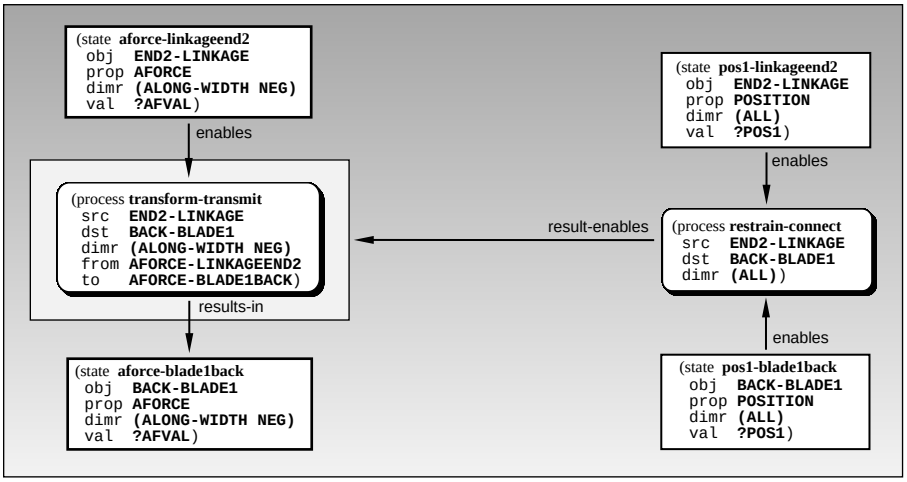


Figure A.5 Carving knife handle dynamics for cutting function.

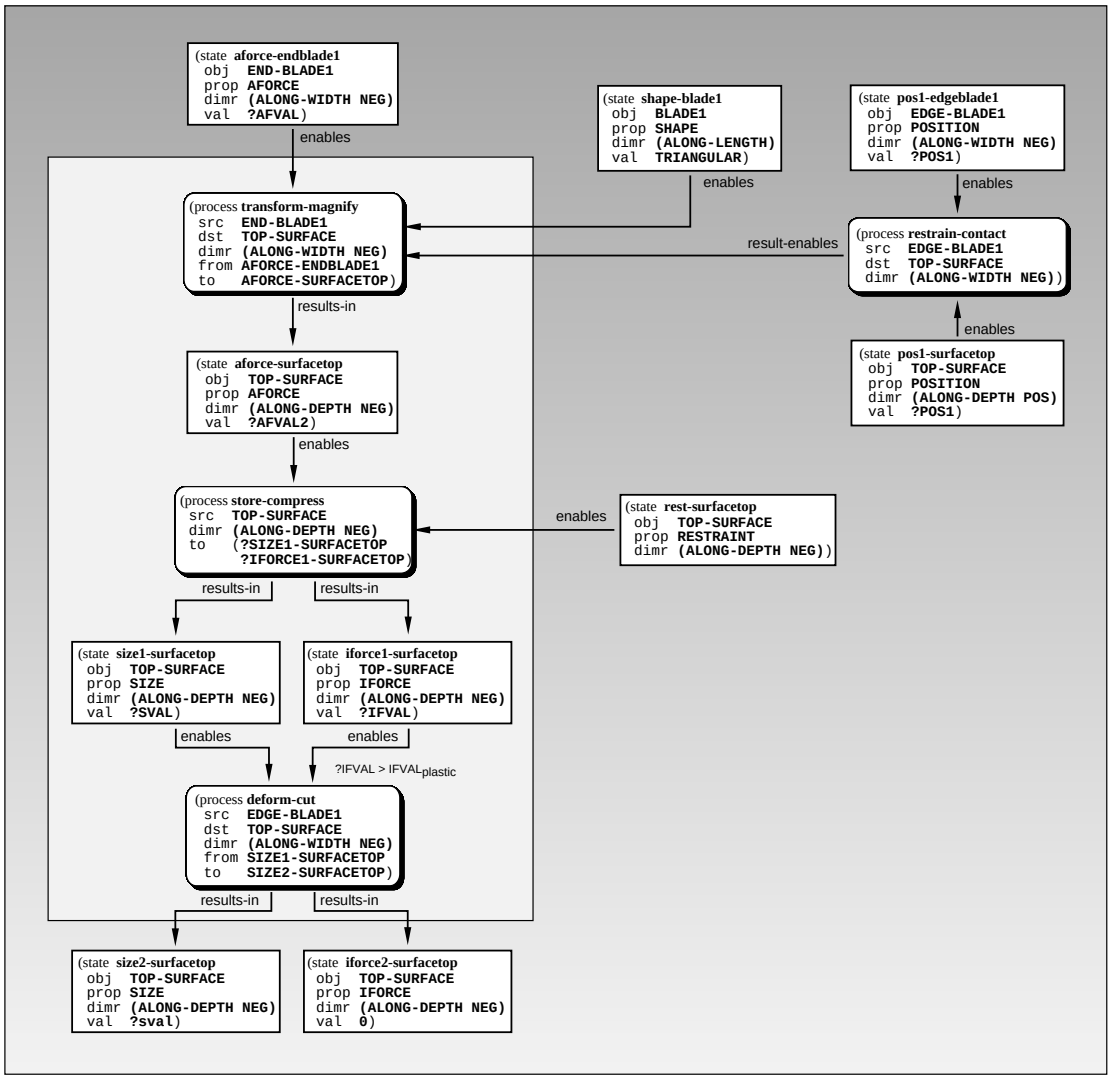


Figure A.6 Carving knife blade dynamics for cutting function.

High-Level Carving Knife Dynamics

Two functions which the carving knife statics instantiate are depicted in Fig. A.7. When the applied force is lateral (as shown at 1), the knife handle instantiates the function of force

translation (MP-LINKAGE-TYPE2).

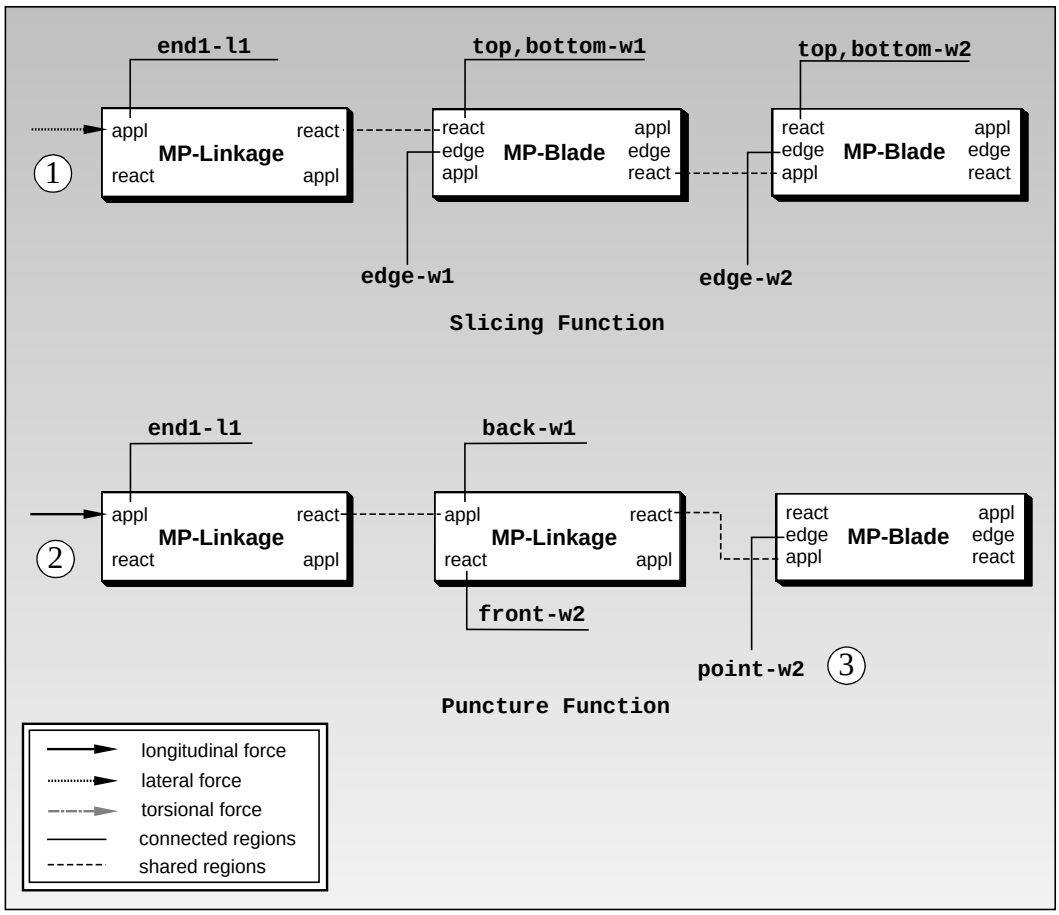


Figure A.7 MP-Diagrams for carving knife functions.

The resulting force and shape of the knife blade enable MP-BLADE and the cutting function at the wedge edges. When the applied force is longitudinal (2), MP-LINKAGE-TYPE1 is enabled, and the force is propagated through the knife to its end (3). In this case, the end of the blade is wedge-shaped both longitudinally and laterally, so MP-BLADE also represents the function, however, the edge role is now instantiated by the knife point.

Carving Knife Pragmatics

The carving knife has a single design-intended use: (1) slicing, as shown in Fig. A.8. Slic-

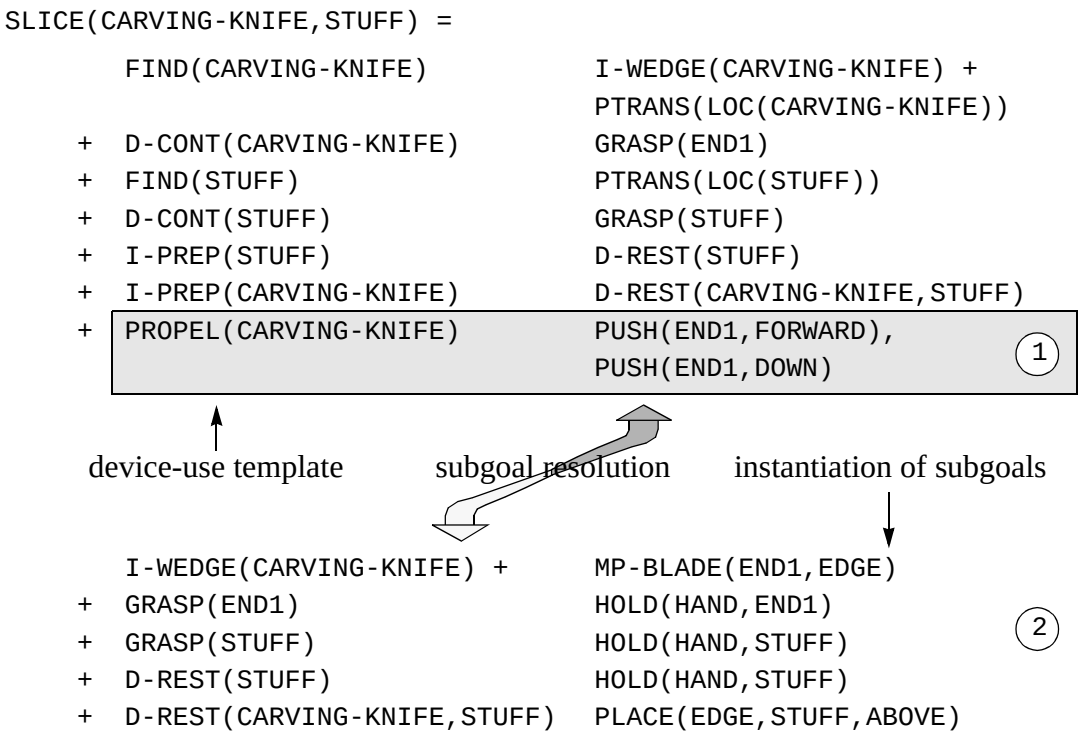


Figure A.8 Pragmatic carving knife representation.

ing is an SOP type plan, in which some material STUFF is removed from a body of STUFF. The SOP plan requires a device which can instantiate MP-BLADE. The HOLD action is used in this plan to produce the necessary restraint on STUFF so that it will not move with respect to the carving-knife, as well as to control the carving-knife. When illustrating actions such as HOLD, the body part will be identified first, and the object second. HOLD(HAND,STUFF) means that the hand is holding the stuff. It is assumed that a hand can only hold one object, so the two HOLD actions (2) must support the inference that different hands are being used to perform these tasks. The PUSH actions use a similar notation. The region where the application is made, as well as the direction, are identified in the figure. In this case, two actions are depicted to clarify the directions of application. No notion of cyclic motion is shown in this representation. For example, the PUSH action would enable MOTION-LINEAR, and the carving-knife could move forward and downward into STUFF. What is missing in this figure is the known MP-BLADE bounding state that, when the end of the blade is reached during slicing, motion must be reversed.

A.1.2 Shovel

A shovel is depicted in Fig. A.9. This shovel is comprised of three components: a handle,

a shaft, and a scoop.

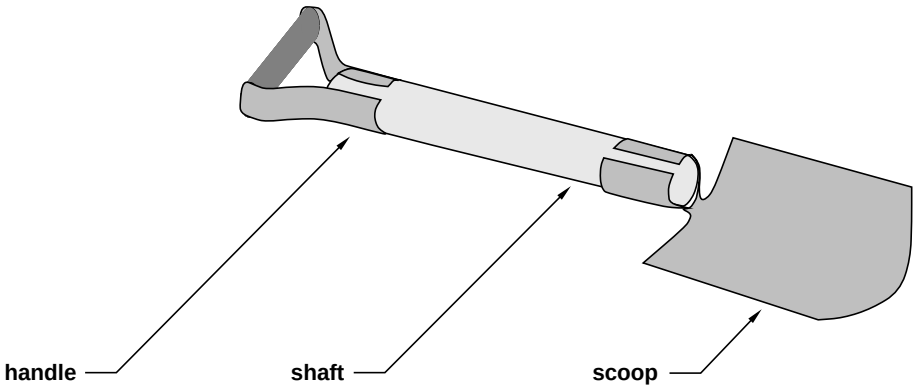


Figure A.9 Shovel illustration.

A shovel has a scoop or container at one end, a long shaft, and a handle at the other end.

Shovel Statics

The scoop is often sharp. An idealized shovel and its static device diagram are depicted in Fig. A.10.

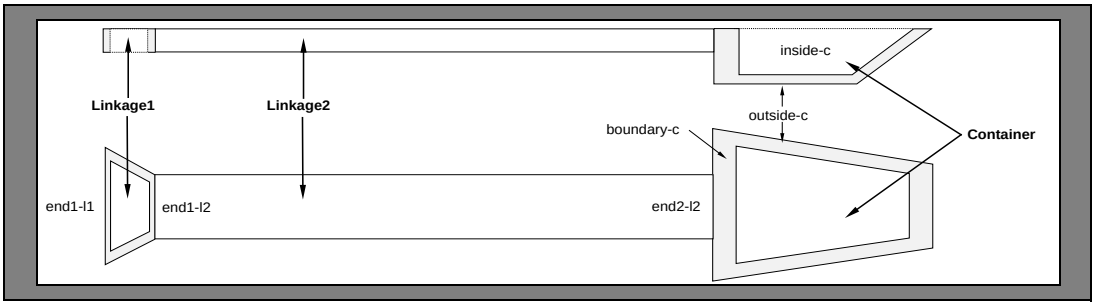


Figure A.10 Shovel object primitives and static device diagram.

The linkage-object which represents the handle has a hole in it, and the hole is approximately the size of the object. The hole is not represented in the SDD shown in Fig. A.11, since the handle can also be blunt..

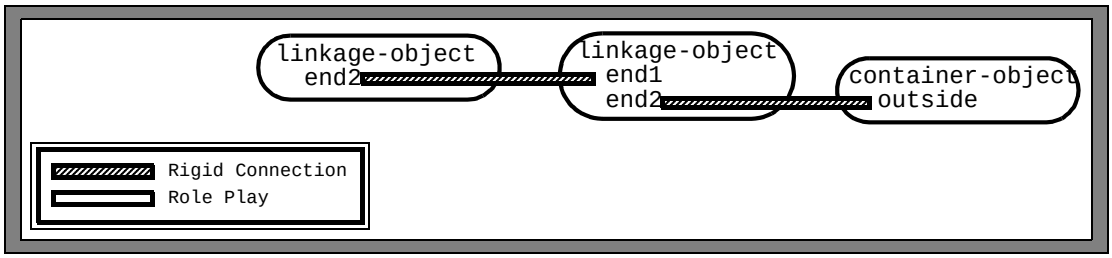


Figure A.11 Shovel static representation diagram.

The container-object is wedge shaped. These relationships are detailed in the accompanying representation (Fig. A.12). In this figure, the shovel is shown having three components

and two spatial relations. The shovel handle is represented as a linkage-object (1) with a hole region and two end regions. It also has a local restraint state (4). The restraint is rigid, meaning that all local dimensions are fixed.

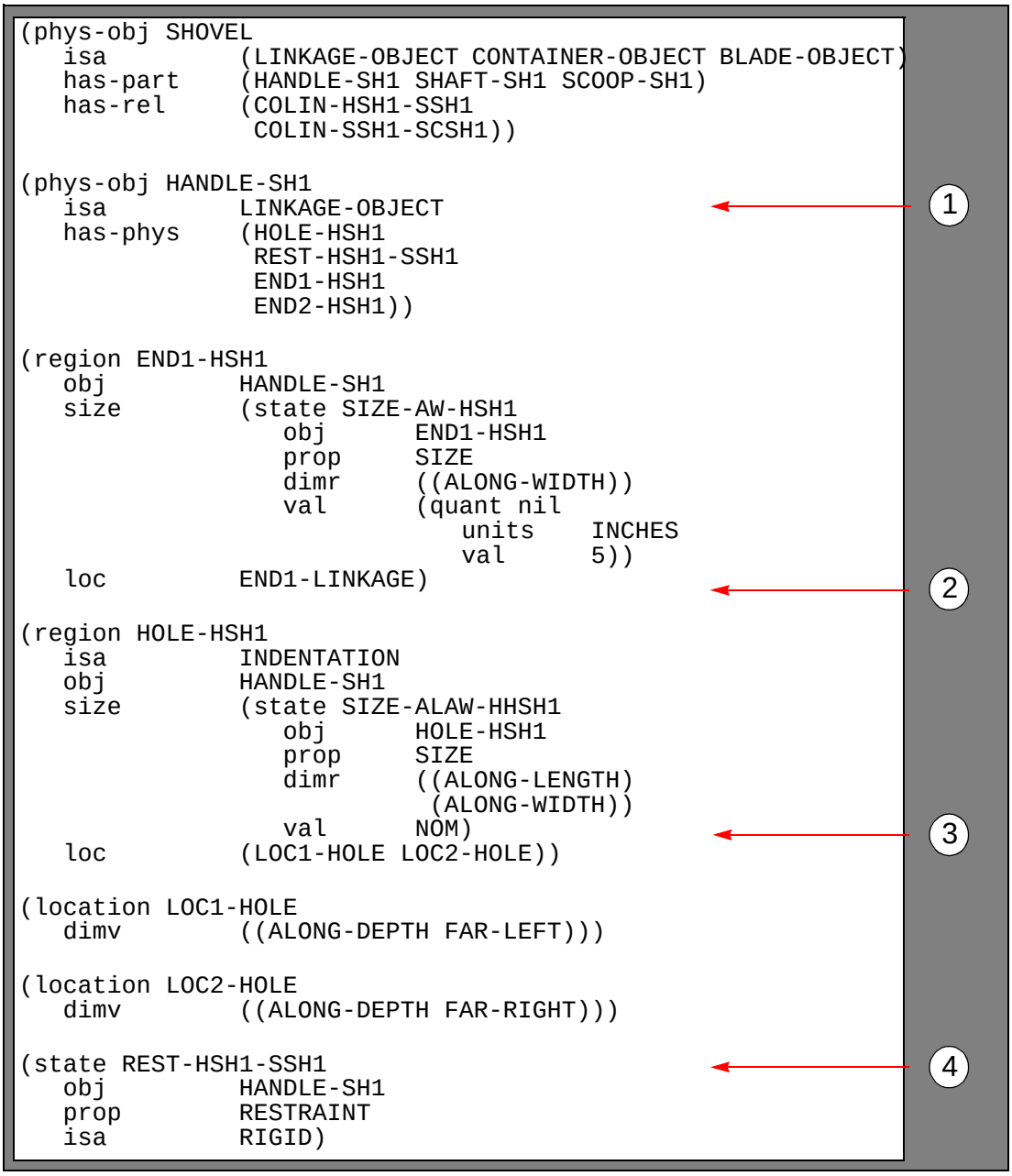


Figure A.12 Physical and relational representation for the shovel handle and its rigi connection to the shovel shaft.

The handle's left end is located at the same place as the linkage-objects left end (2), as is the handle's right end with the linkage-object's right end. The hole is represented as an indentation which is located at both extremes of the handle, and which has approximately the same area (NOM, at 3, in length and width) as the handle.

Low-Level Shovel Dynamics

The behavioral sequences for the shovel lifting function are illustrated in Figs. A.13 - A.15.

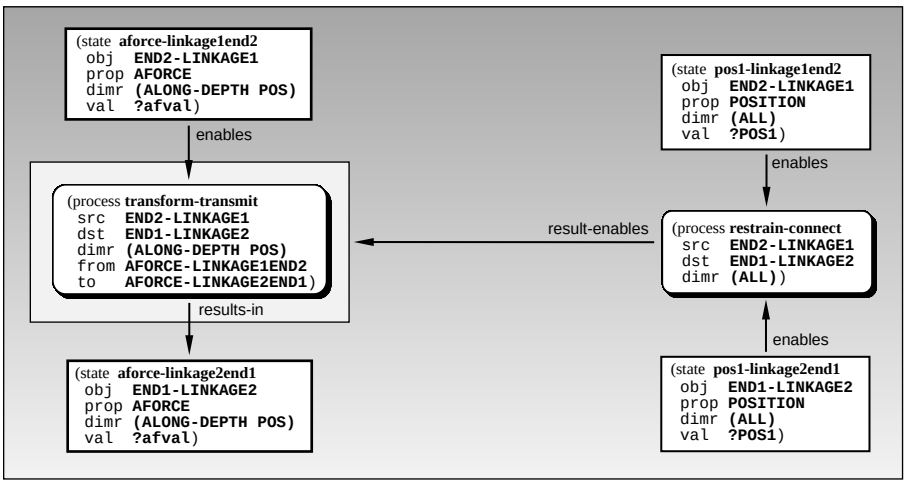


Figure A.13 Low-level shovel handle dynamics for lifting function

These two diagrams, except for their instantiations, are identical to the screwdriver handle and shaft low-level dynamics..

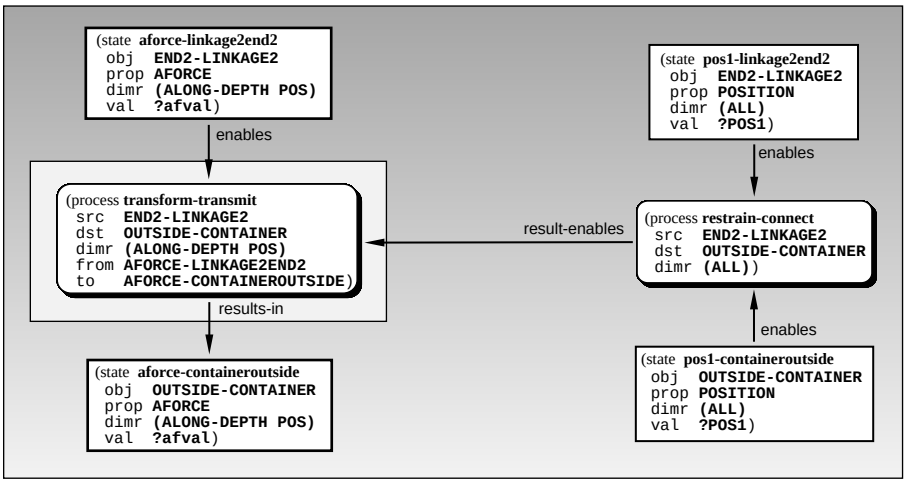


Figure A.14 Low-level shovel shaft dynamics for lifting function

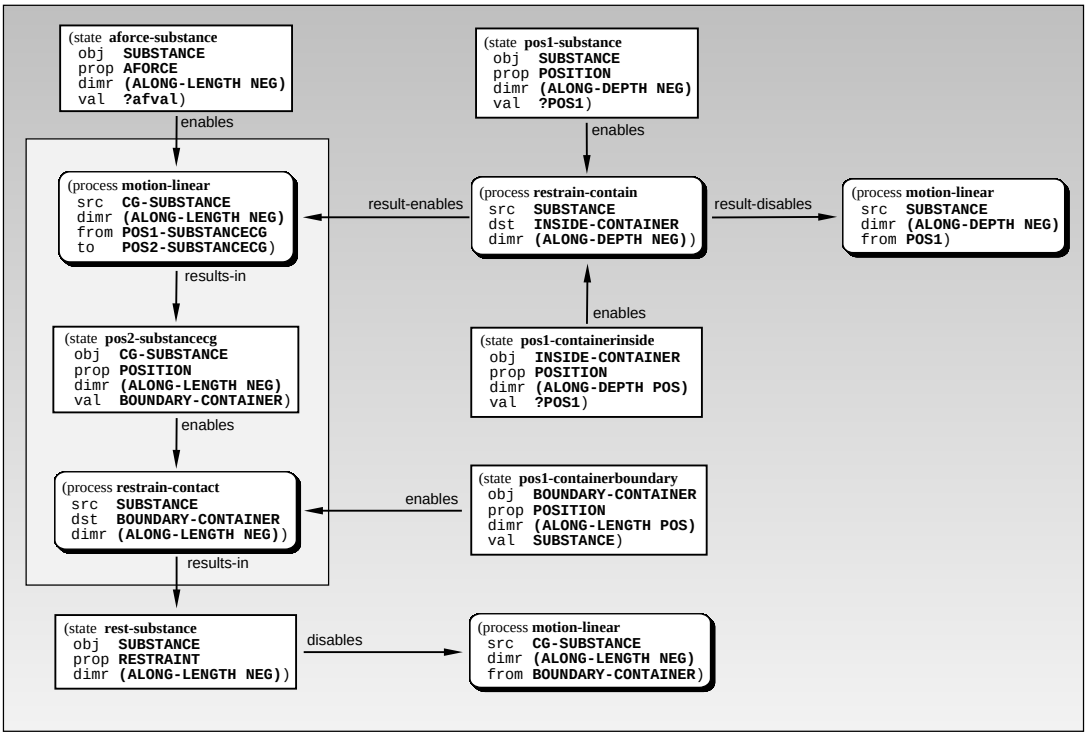


Figure A.15 Low-level shovel scoop dynamics for lifting function

High-Level Shovel Dynamics

A shovel can be used in many ways: for cutting, leveraging material, carrying, and throwing. The general use of a shovel involves four functions: (1) cutting the soil with the blade, (2) leveraging the soil from the new hole, (3) lifting and transporting the soil, and (4) flinging the soil. Each function is instantiated by applying force to different locations on the shovel while constraining its motion at others. These functions are shown as MP-Diagrams in Fig. A.16. The shovel functions add the complexity of multiple inputs to the representation of device function. When the shovel is used for digging, the handle is held (at 1) and and a longitudinal force is applied to the outside of the scoop (2). Since the scoop illustrated is wedge shaped, MP-BLADE is used to represent the cutting function. In the second function illustrated, the shovel handle is held and pushed (3) and the body is used as a fulcrum on the edge of the hole (4). In the diagram, the shovel body is shown instantiating the MP-LEVER pivot role (5), however, the instantiation is really the contact between the body and

the hole.

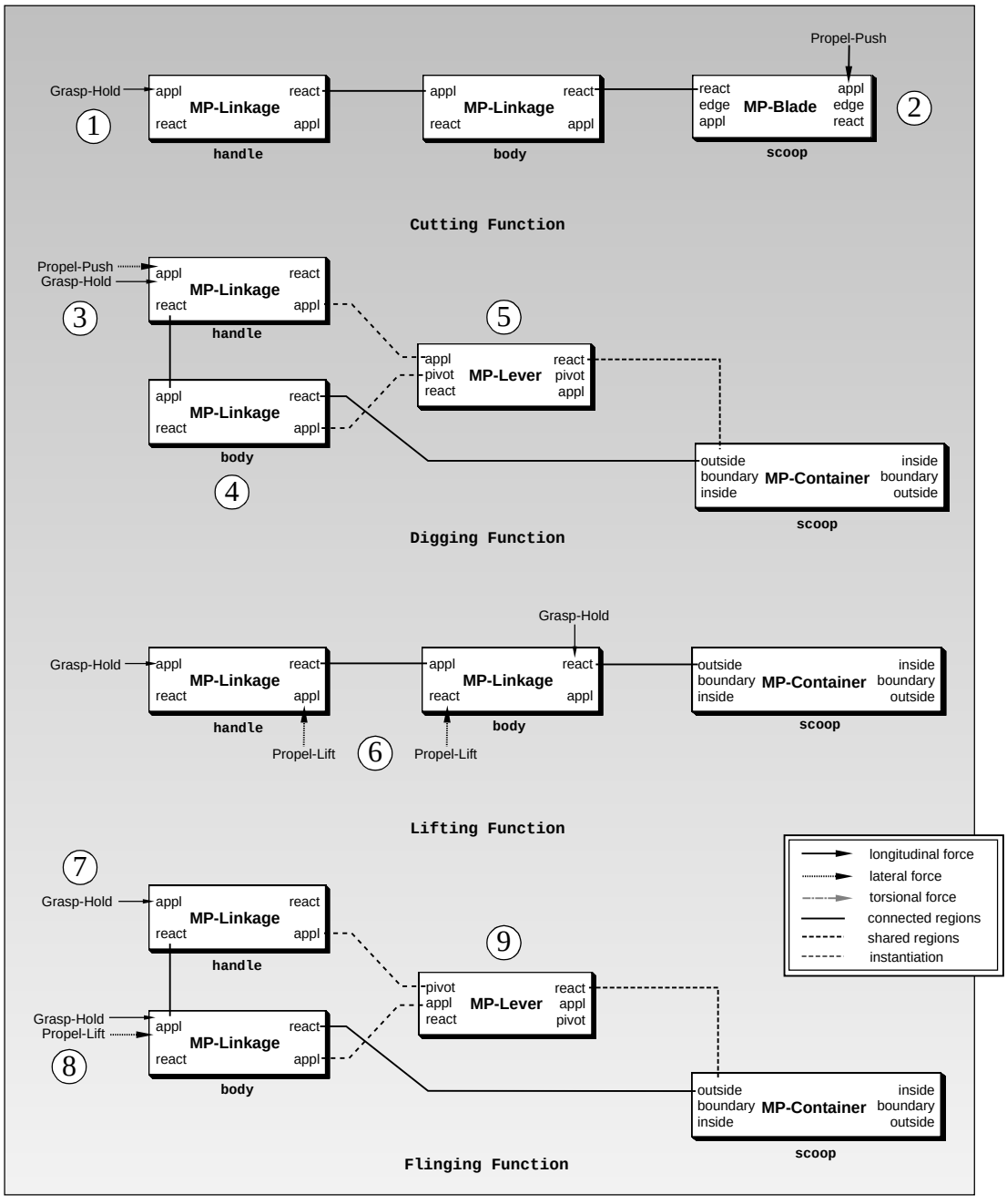


Figure A.16 MP-Diagrams for shovel functions.

The light dotted lines depict the instantiation of MP-LEVER roles. The third function shows two lateral forces being applied to different regions (6), multiply enabling MP-LINKAGE-TYPE2. The last function illustrated shows the handle being held (7) and the body being lifted (8), which instantiates MP-LEVER-TYPE3 with the handle instantiating the pivot role. The lever is instantiated in two of these functions, but the difference in how and where the force is applied changes how it is instantiated and, thus, what subclass is used to represent the function. Each of the last three functions is represented using MP-

CONTAINER, since the soil is in contact with the container's inside surface.

Shovel Pragmatics

The shovel has two design-intended uses: (1) create holes, and (2) transport material. Hole creation is an SOP type plan, and makes use of the cutting edge on the shovel container boundary, while transport is a CROM type plan and makes use of the container itself.

The hole-digging use is shown in Fig. A.17. Being a large device, the shovel requires control in more than one location by more than one appendage. In this application, the hand(s) are placed on the handle and a foot is placed on the edge of the scoop (at 2 in the figure). A hole can also be created by placing a second hand on the shovel body, depending on the nature of the shovel and stuff. The grasping actions are not noted as being either sequential or simultaneous, since the shovel can be used either way. Another new item is that the applied force is also located at two regions. The foot presses the container edge into the stuff while the hands push against the handle (1). Depending on the shovel and stuff, lever-

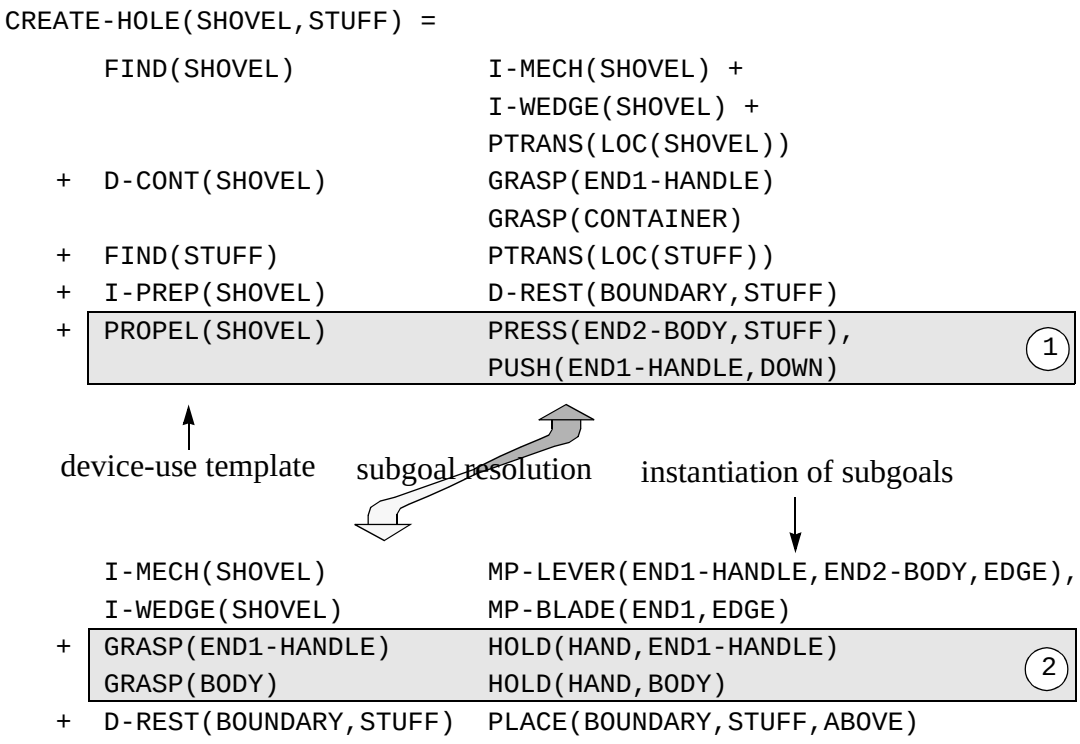


Figure A.17 Pragmatic representation of shovel hole creation.

age is required to create a hole, in which case the device chosen must be used in such a way that it instantiates MP-LEVER-TYPE1. All digging plans require the application of an object which instantiates MP-BLADE (2).

The second application, that of transporting stuff with the shovel, is shown in Fig. A.18. Transport is a CROM type plan, since the control over the substance is the primary concern. Most of the subgoals in the device-use plan are redundant from the previous plan, so they have been left out of the illustration. There is no additional preparation of the stuff required. In this example, force is applied at both locations where the shovel is being controlled, and

the DO action of the plan is a D-PROX rather than a PROPEL.

```
TRANSPORT-SUBSTANCE(SHOVEL,STUFF) =

                +  PROPEL(SHOVEL)          LIFT(END2-SH-BODY,UP)        (1)
                                           LIFT(END1-SH-HANDLE,UP)
          (2)   +  D-PROX(SHOVEL)          PTRANS(SHOVEL,LOC2)

                   ↑                              ↑
            device-use template          instantiation of subgoals
```

Figure A.18 Pragmatic representation of shovel substance transport.

A.1.3 Spoon

The spoon and shovel are very similar objects. The shovel depicted in Fig. A.9 is comprised of three components, whereas the spoon depicted in Fig. A.19 is a single component with three regions of the same names as those in the shovel.

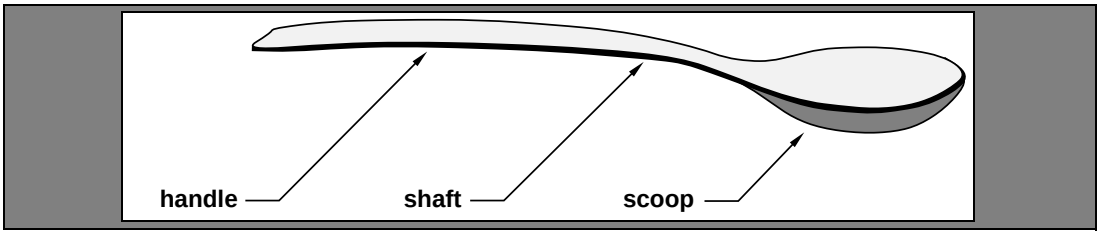


Figure A.19 Spoon illustration.

The spoon regions are associated with the same kinds of object primitives as they are for the shovel. The spoon is shorter than a shovel, but has a container or scoop at one end and a handle at the other. All components are scaled down in size.

Spoon Statics

The spoon and shovel have similar static representations, except that the spoon handle and body both comprise regions of a single component, while in the shovel they are represented with two components. An idealized spoon and its static device diagram are illustrated in Fig. A.20 and Fig. A.21.

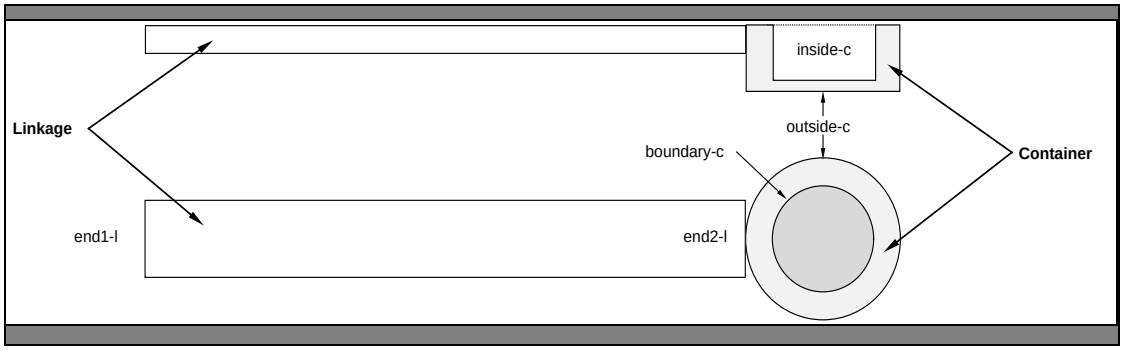


Figure A.20 Spoon object primitives and static device diagram.

The similarities between this figure and Fig. A.10, and between Fig. A.20 and Fig. A.11, are obvious. The two linkage-objects used to represent the shovel can be considered a single linkage-object in comparison to that of the spoon. In other respects, except size, the two objects have identical representions.

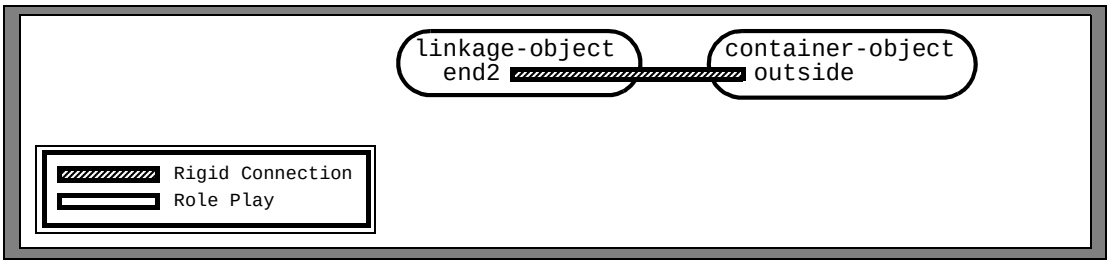


Figure A.21 Spoon static representation diagram.

Low-Level Spoon Dynamics

The behavioral sequences for the spoon are included here for continuity and comparison to the shovel diagrams. The only difference between the two devices at the behavioral level is magnitude: size of object, volume of containment, magnitude of applied and reacted forces..

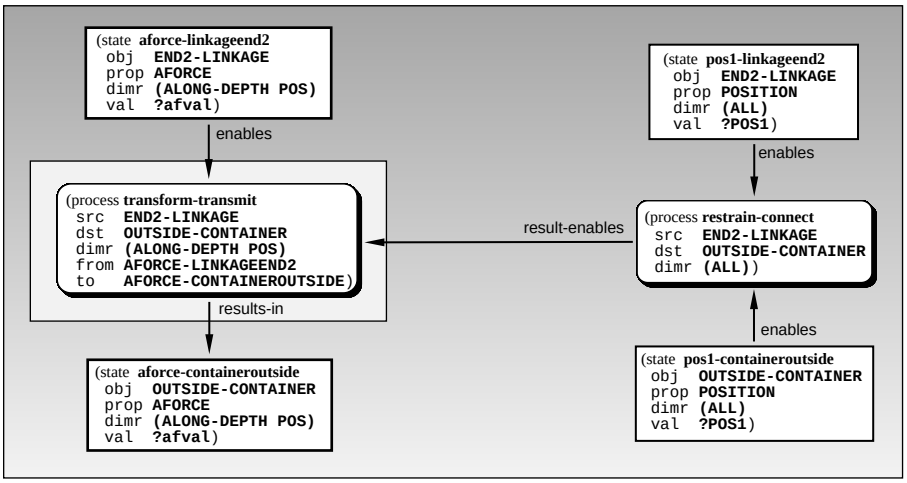


Figure A.22 Low-level spoon handle dynamics for lifting function

A spoon has the same kind of function as a shovel, and the representation of its behavior

and function should capture their similarity.

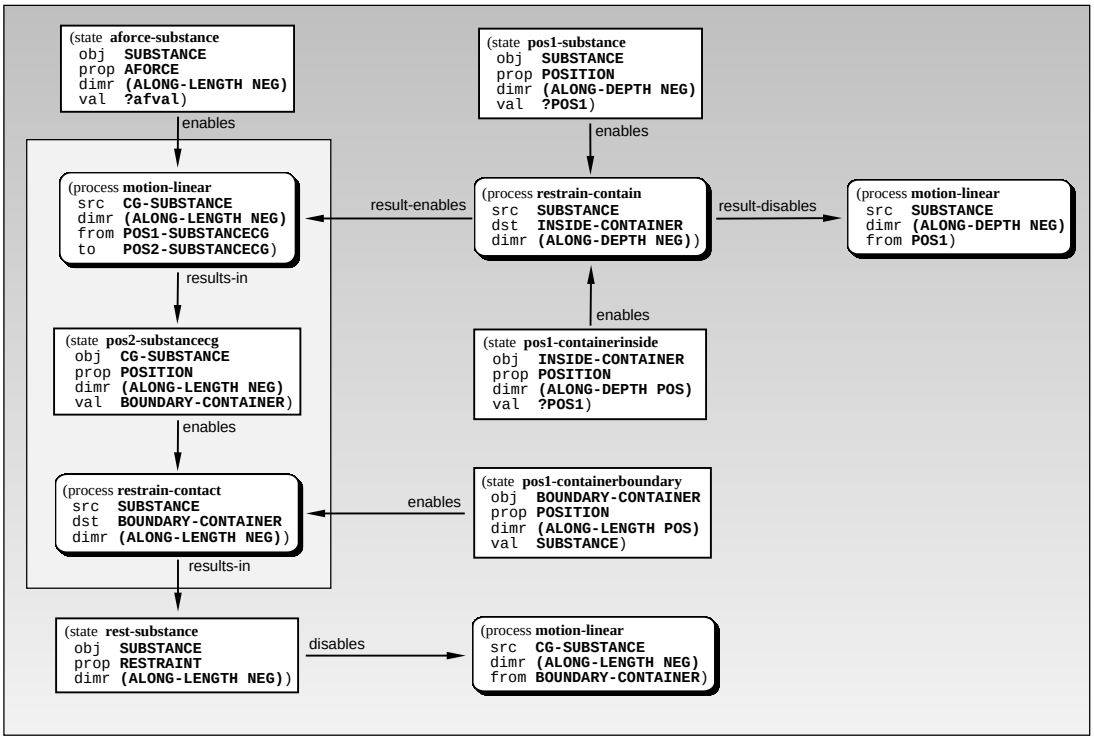


Figure A.23 Low-level spoon scoop dynamics for lifting function

High-Level Spoon Dynamics

Two spoon functions, dipping and lifting, are shown in Fig. A.24. The dipping function (1) has an identical MP-Diagram to that of the shovel cutting function, and performs an iden-

tical task.

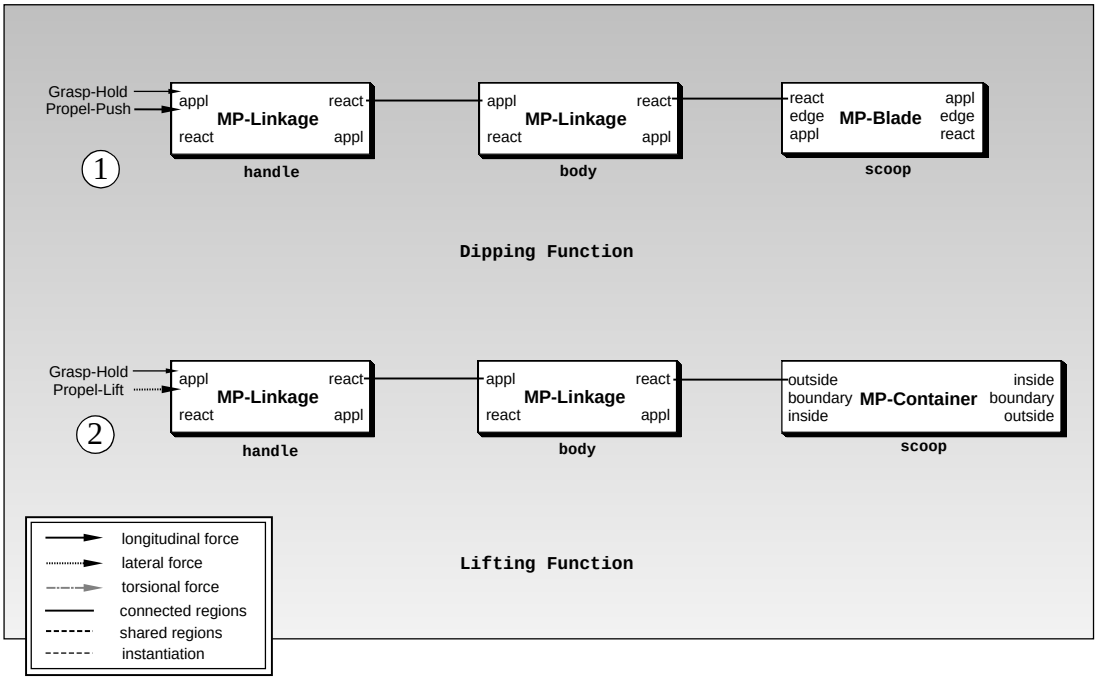


Figure A.24 MP-Diagrams for spoon functions.

Likewise, the lifting function (2) is represented using the same primitives as the shovel lifting function, the difference being the absence of the second applied force. Physically, the spoon and shovel differ in size more than composition, and these function representations illustrate their dynamic similarities. The digging and flinging shovel functions also have spoon counterparts, not illustrated in Fig. A.24.

Spoon Pragmatics

The spoon has two design-intended uses: (1) dipping, and (2) transport. The dipping application is illustrated in Fig. A.25. Dipping is an SOP plan like digging a hole with a shovel. Two new plans are involved in dipping. The spoon edge must be inserted into the stuff so that the inside of the spoon's container is in the stuff. This requires a second new plan, DI-

RECT, which points the edge of the spoon toward the stuff, before it is actually relocated.

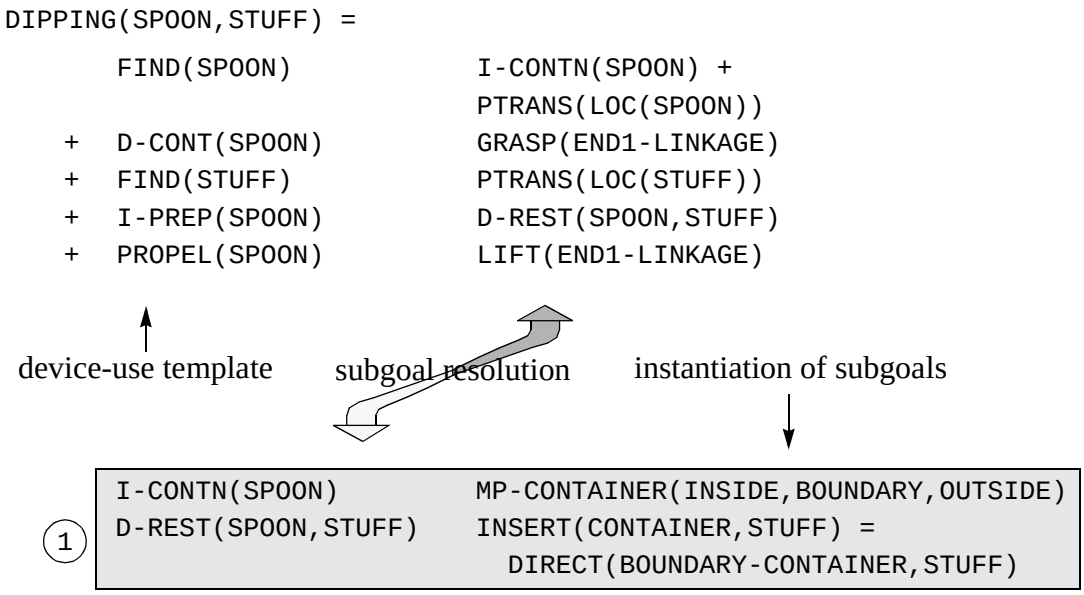


Figure A.25 Pragmatic representation of spoon dipping.

In both plans, the second argument in the plan represents an implicit direction (toward). For example, DIRECT(BOUNDARY-SPC,STUFF) means that the edge of the spoon container is aimed at the stuff. Since the edge is the entire container boundary, nothing is nor need be said about which portion of the spoon container is dipped into the stuff. The transport use of the spoon is identical to that of a shovel (i.e., a CROM type plan), and is not illustrated.

A.1.4 Claw Hammer

A claw hammer is illustrated in Fig. A.26. The claw hammer has a handle, a shaft, a head, and a claw. The claw is curved toward the handle and has a wedge-shaped notch in it

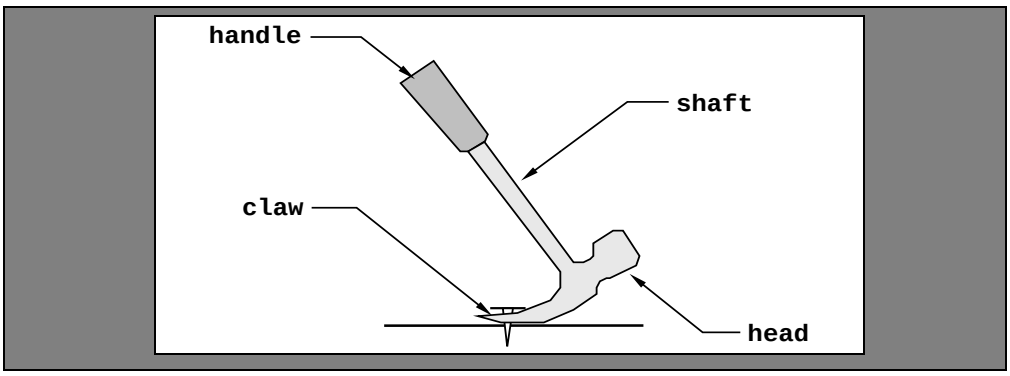


Figure A.26 Hammer illustration.

A claw hammer is used for pounding and prying.

Claw Hammer Statics

It has a long handle, and a head component which is heavy. One end of the head is flat, and the other is wedge-shaped and slotted. An idealized claw hammer and its static device diagram are depicted in Fig. A.27.

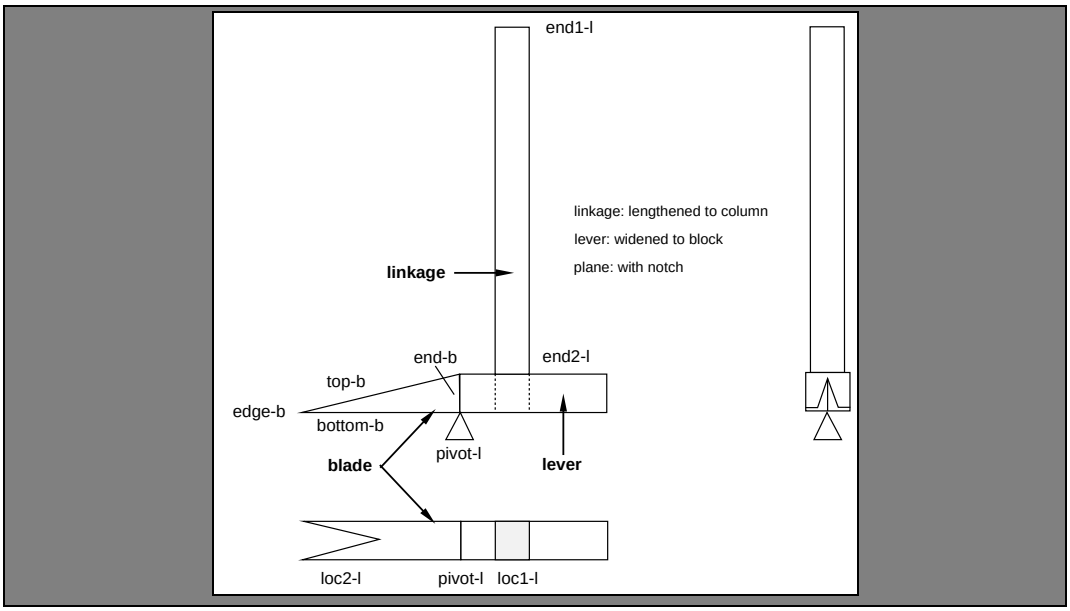


Figure A.27 Hammer object primitives and static device diagram.

The object is represented with a linkage-object, a lever-object, and a blade-object with an indentation for representing the slot. The lever-object `loc2` region is represented with a blade-object, and the lever-object pivot region represents the claw curvature. The accompanying SDD is shown in Fig. A.28.

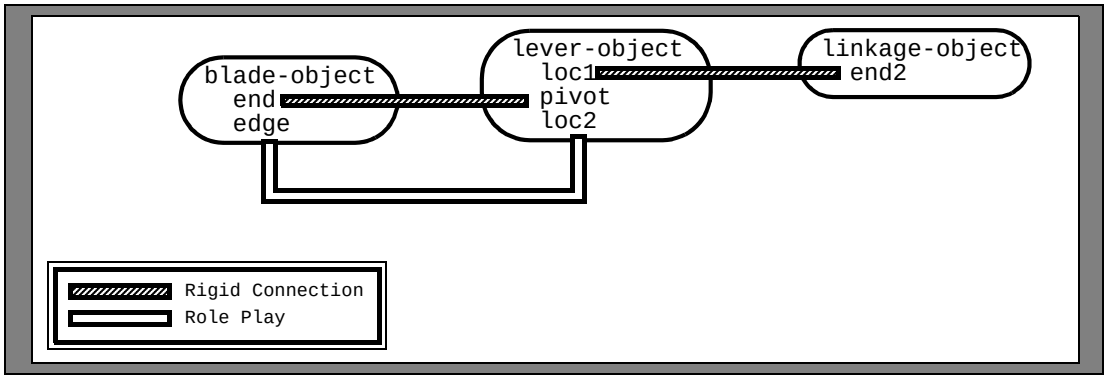


Figure A.28 Hammer static representation diagram.

In this figure, the `end` region of the plane-object is fixed to the `pivot` region of the lever-object. The `end2` region of the linkage-object is also fixed to the `loc1` region of the lever-object. A role-play link associates the `loc2` lever-object region with the `edge` blade-object region. The representation for the slot indentation on the claw, the perpendicular orientation of the handle and head, and the rigid connection between the head and the claw are represented in Fig. A.29. The blade-object edge region (`edge-cach1`, shown at item

1) has an indentation located at along its width and length. The slot is sized less than either dimension. The locations `loc1-slot` and `loc2-slot` place the slot ALONG-DEPTH at extremes (2), so that the slot goes all the way through the claw in this dimension.

The orientation between the hammer handle and head is represented with an orient knowledge structure (3). The two components, HANDLE-H1 and HEAD-H1, are oriented with respect to their longitudinal axes, which are 90 degrees offset. This relationship is represented with a single element size, where the dimensions are both ALONG-LENGTH.

Finally, the connection between HEAD-H1 and CLAW-H1 is represented with a fixed restraint state (4).

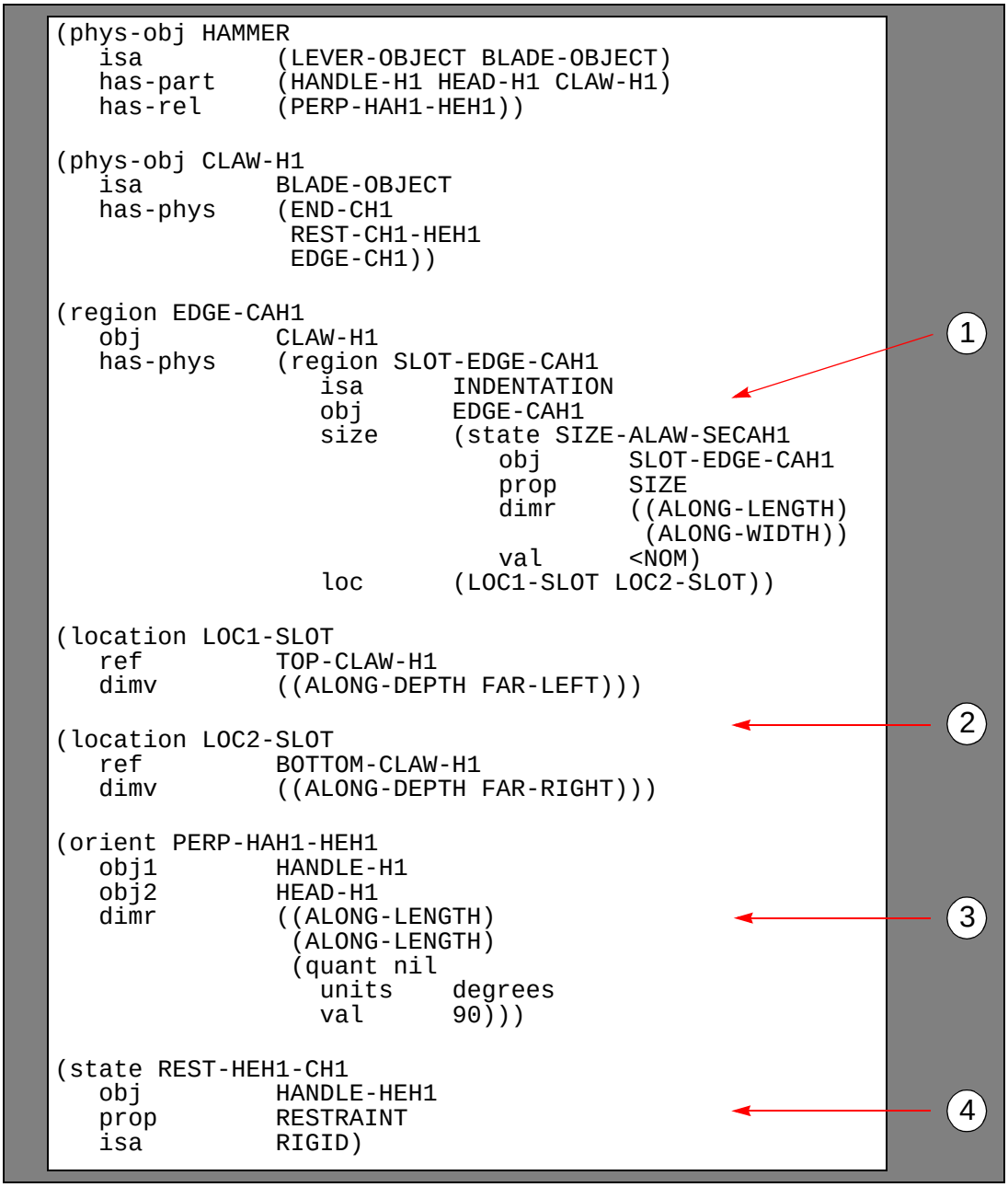


Figure A.29 Representation of claw hammer orientation and claw slot.

High-Level Claw Hammer Dynamics

Two hammer functions, those for prying and pounding, are depicted in Fig. A.30. The significance of this example is the representation of two different MP-LEVER classes with the same force type (lateral, at 1 and 3) and location (end1). In the prying function, the pivot is the head of the hammer (end2 of the handle), and MP-LEVER-TYPE1 represents the function (2). In the pounding function, the pivot is not a component of the device but, rather, the person wielding the device, and MP-LEVER-TYPE3 represents the function (4).

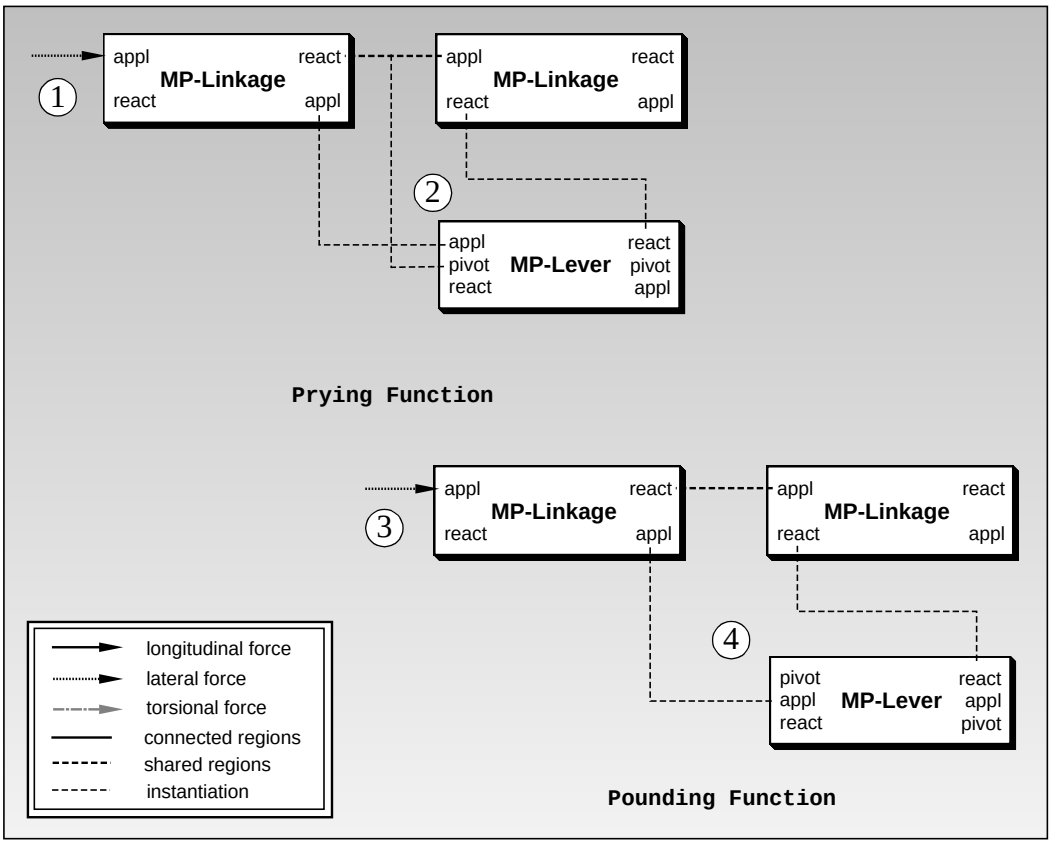


Figure A.30 MP-Diagrams for hammer functions.

Claw Hammer Pragmatics

The claw hammer has two design-intended uses: (1) nail pounding, and (2) nail removal. Nail pounding is a TFD plan, and nail removal is a CROM plan. Both plans support CROM plans for connecting and disconnecting objects. The pounding use is illustrated in Fig. A.31.

This use introduces the possibility of altering the device-use subgoals. No preparation of

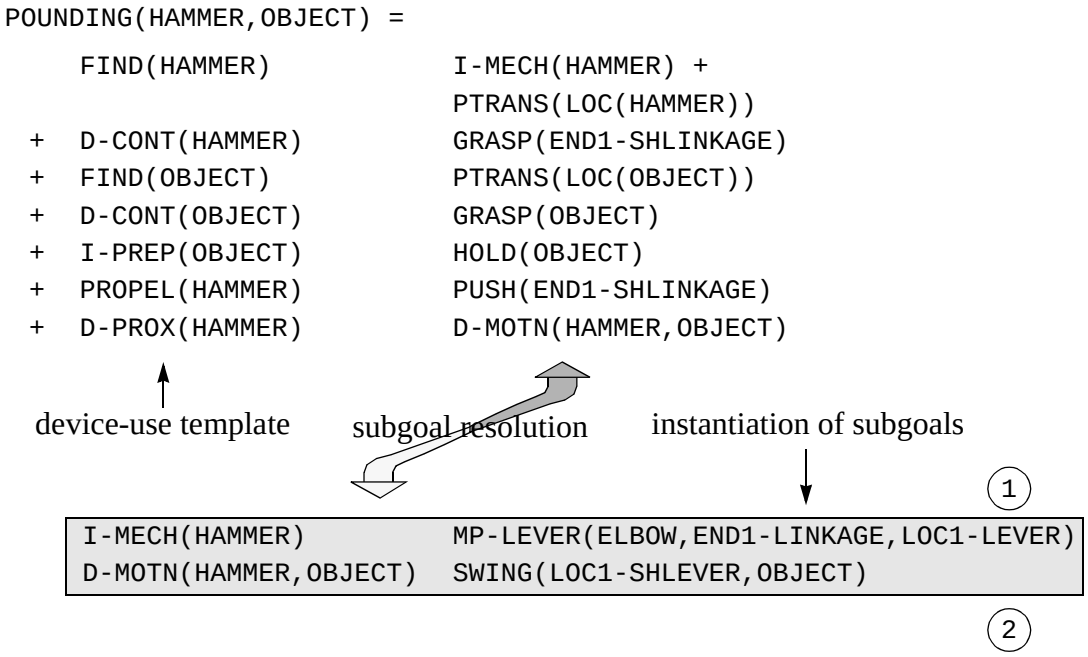


Figure A.31 Pragmatic representation of hammer pounding.

the hammer is required once control is effected, however, both an applied force, and movement of the hammer head into position, are required (i.e., PROPEL and D-PROX). The applied force is effected by swinging the hammer handle toward the object, where SWING is a plan which instantiates the MOTION-SWING process. The SWING plan also instantiates MP-LEVER-TYPE3 with the arm-hammer combination.

The nail removal plan is a confinement plan (CROM) which utilizes leverage, as illus-

trated in Fig. A.32.. In this figure, a new plan is introduced, called SLIDE Fig. A.32. In this

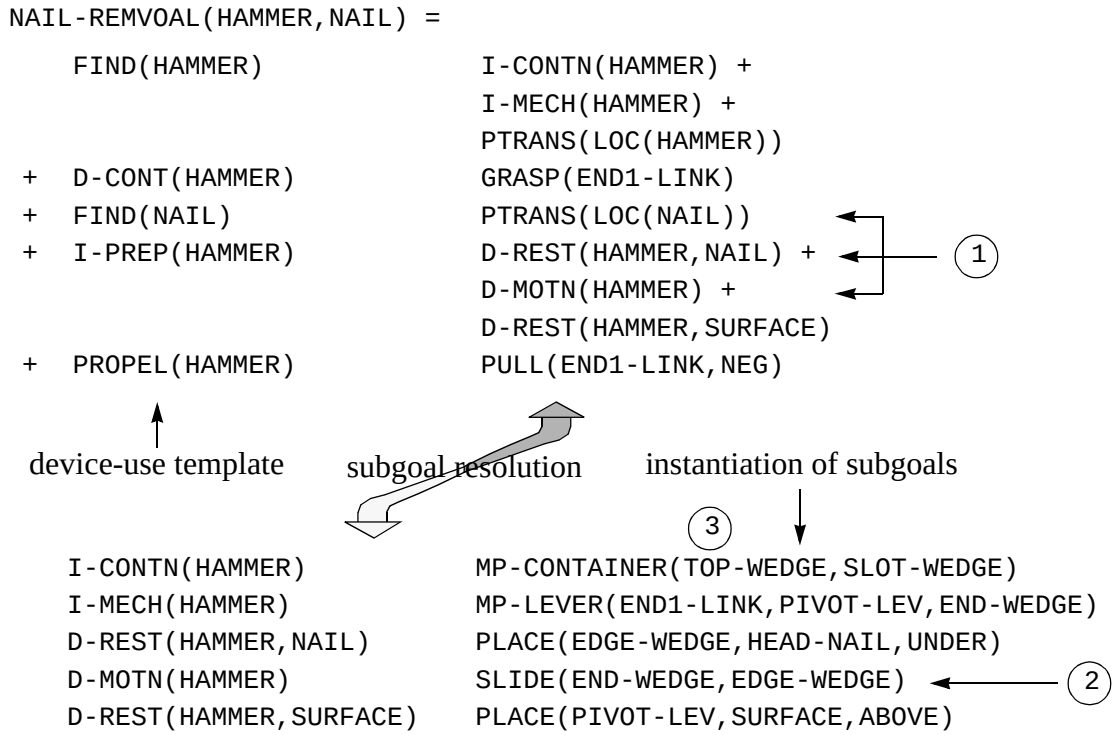


Figure A.32 Pragmatic representation of hammer nail removing.

figure, a new plan is introduced, called SLIDE To remove a nail with a claw hammer, the user (a) positions the hammer claw under the nail head, (b) slides the hammer toward the nail, (c) creates a fulcrum with the hammer head, and (d) pulls the hammer away from the nail. These tasks are represented with the behavioral delta goals D-REST and D-MOTN at (1) in Fig. A.32. SLIDE is used to describe the movement of the hammer after being positioned under the head of the nail, so as to wedge the nail in the claw slot (2). When the hammer is pulled, the direction is identified implicitly as LOC1-HLEVER (toward the head) to indicate away from the claw.

A.2 Simple Compound Devices

Devices in which relative motion between components is possible are *compound* devices. Compound device dynamics are representationally significant because components can function independently. This has two affects on the representation and interpretation of compound devices. First, if a device component is not rigidly connected to other components, then the functions it can instantiate can be invoked without considering the other device components. Second, if a machine primitive participates in a device's function representation, then its function can be considered independently of the function in which it participates. In this section, three devices are presented which support these notions insofar as two components can be considered somewhat independently:

(1) scissors
(2) pliers
(3) door

A.2.1 Scissors

A pair of scissors is illustrated in Fig. A.33. A pair of scissors has two handles, a pivot, two

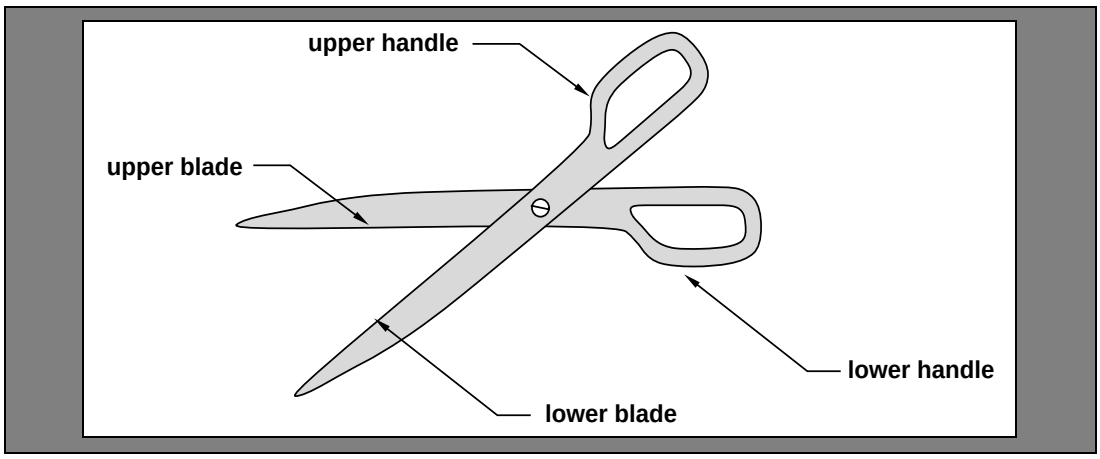


Figure A.33 Scissors illustration.

blades and two pointed tips. The device is comprised of two components connected together at a common location. Like the shovel, the scissor components have a hole region called a handle.

Scissors Statics

A pair of scissors is a simple compound device in which two components are pivoted to one another. The second component becomes the fulcrum for the first, so either can function independently of the other. Each component consists of a handle at one end and a pointed blade at the other. An idealized pair of scissors and its associated static device di-

agram are illustrated in Fig. A.34.

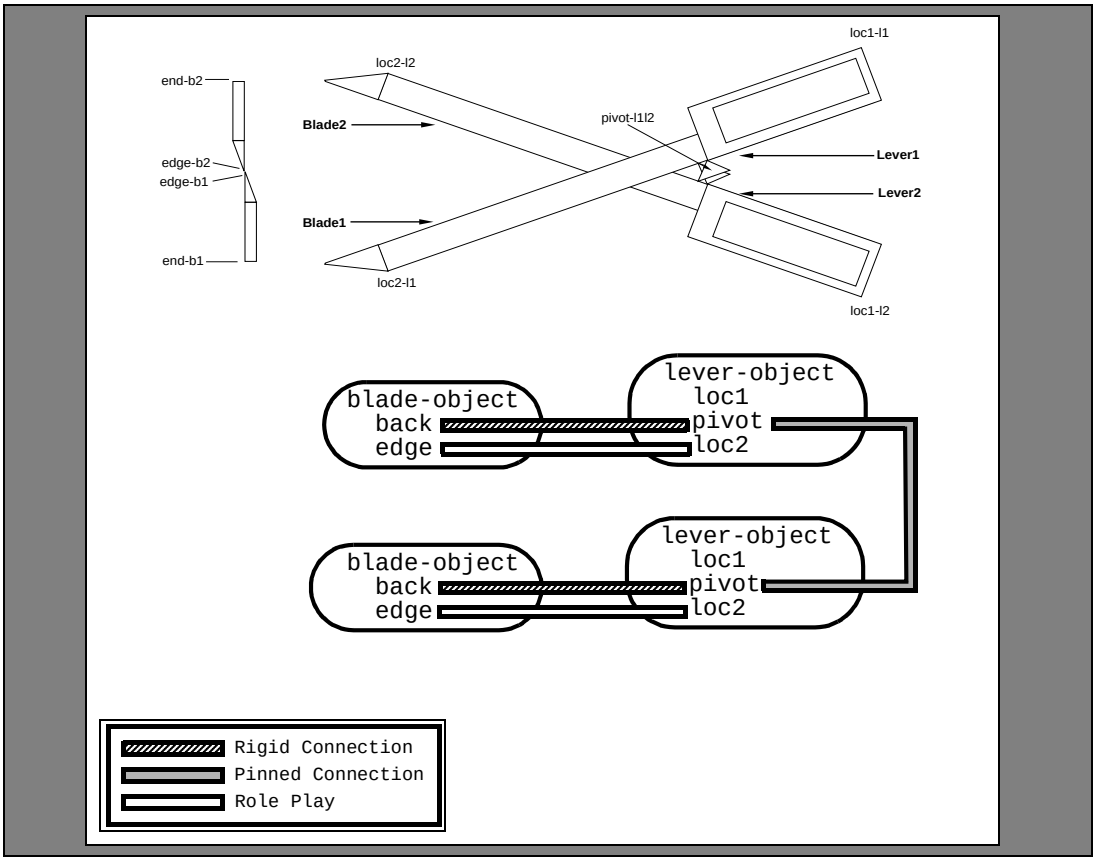


Figure A.34 Scissors object primitives and static device diagram.

Because of the pivot in the static description, the components are represented with lever objects. Because of the shape of the components, they are represented as blade objects. Since each component can be a single device the lever and blade objects are depicted as rigidly connected with regions being shared. The shaded arc between the lever object pivot regions represents a pinned connection.

The components are each represented with a lever-object whose `loc2` region is instantiated with a blade-object. The `pivot` regions are shared between the two objects, as a pinned connection. The SDD for the device is shown in Fig. A.34.

In the figure, the `pivot` region for both lever-objects is represented as the other object. The handle regions, represented in Fig. A.35, are rigidly connected to the lever-object `loc1` regions. Similarly, the `back` region of either blade-object is rigidly connected to the lever-object `pivot` regions. The blade-object `edge` regions play the role of the lever-object `loc2` regions.

Fig. A.35 represents the scissor device (1) as a combination of two components in parallel (PARL-CSC1-CSC2). The handle of one of the scissor components (COMP1-SC1) is represented at (2). The handle is a hole region located at the lever-object `loc1` region (at the end). Like the shovel, the hole is approximately the size of the component, and goes through the component (3). The pivot region of COMP1-SC1 is represented at (4). It is located at the lever-object pivot region, and has similar characteristics to the handle hole. COMP1-SC1 is restrained (5) in all three translational dimensions and two of the three ro-

tational dimensions. This is a pinned connection from Table 2.11.

The components of the scissor pair, unlike other devices represented so far, are connected in a nonrigid manner which enables independent motion of one component with respect to the other. This type of connection is characteristic of compound devices.

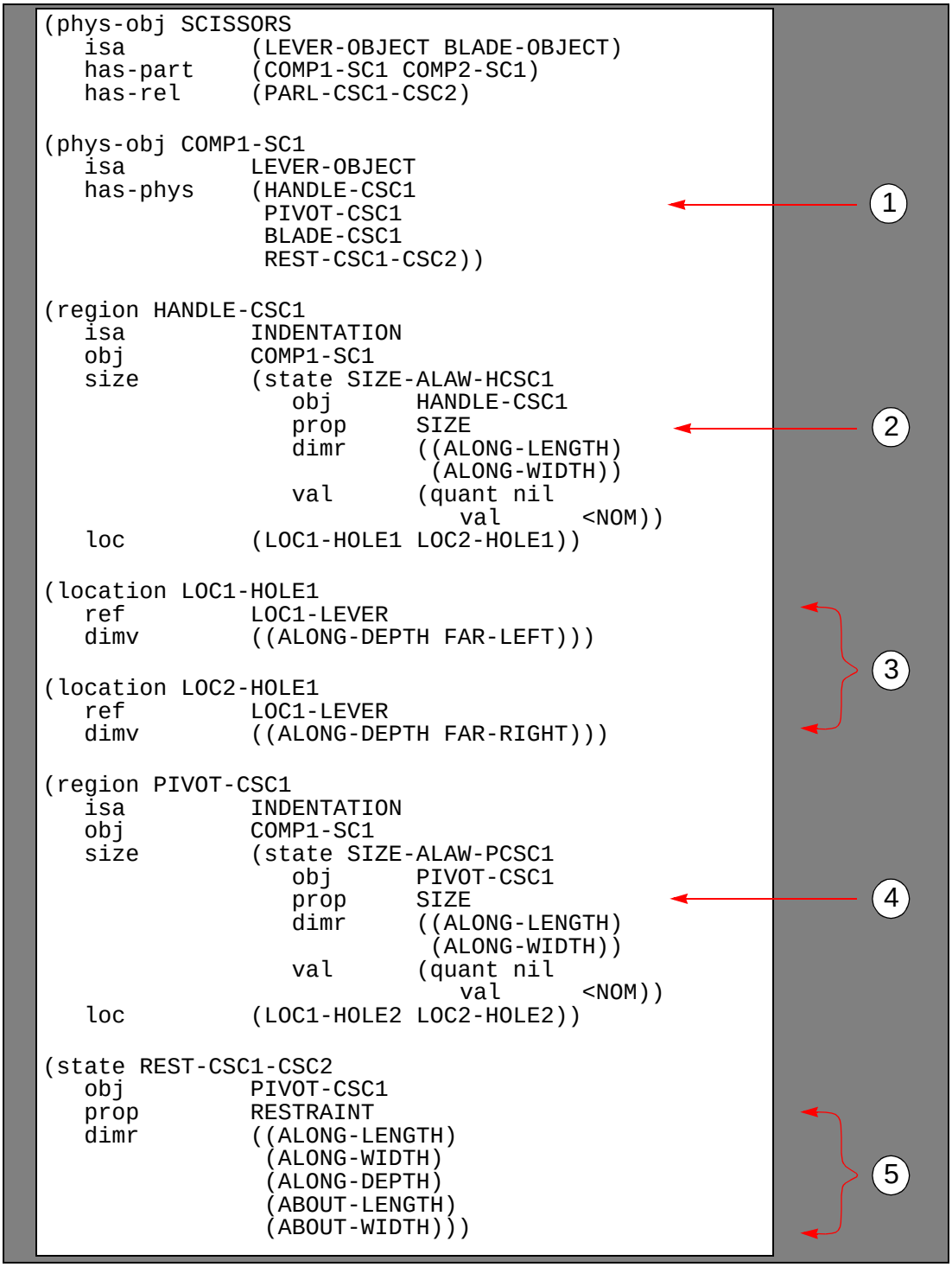


Figure A.35 Partial static representation for scissors.

High-Level Scissors Dynamics

Two scissors functions are illustrated in Fig. A.36. The cutting function is initiated by applying lateral force to the handles.

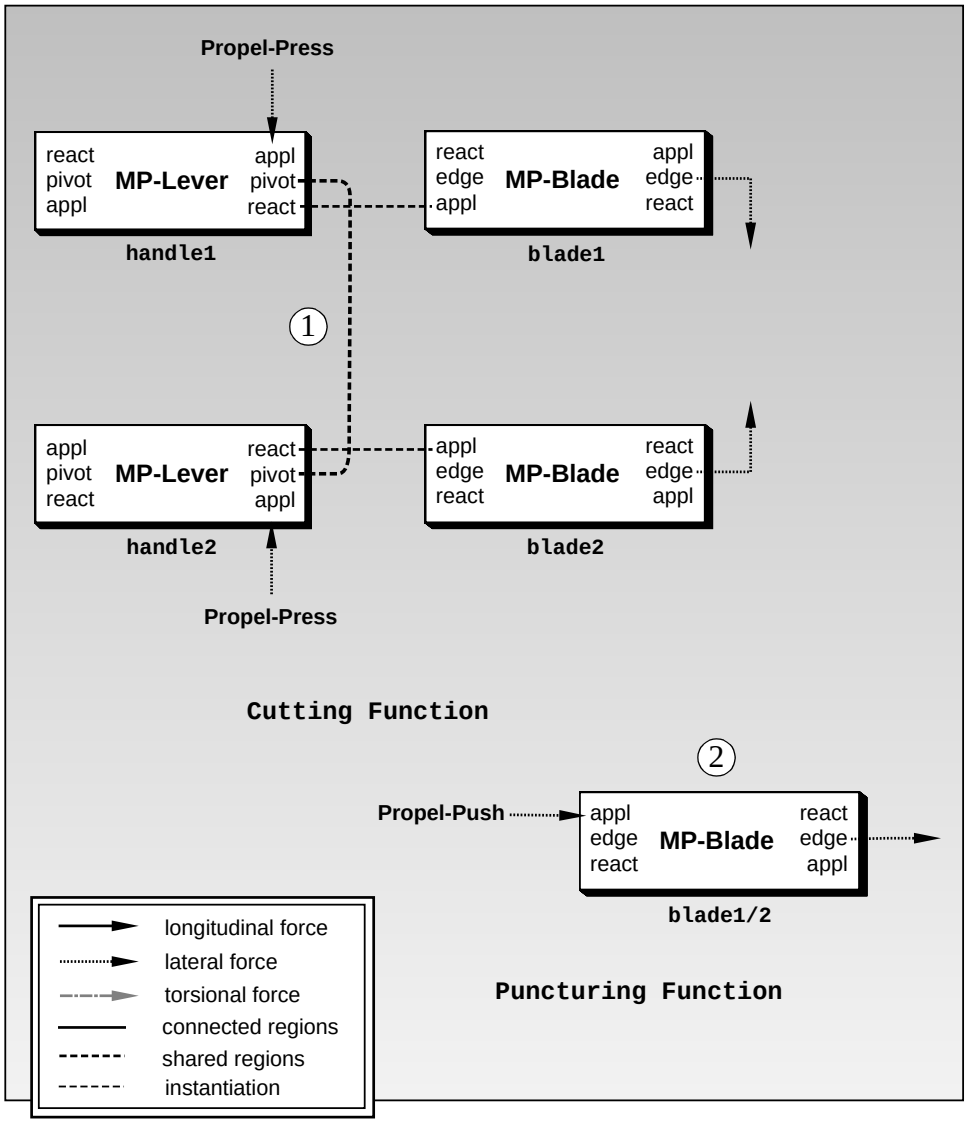


Figure A.36 MP-Diagrams for scissor functions.

The pivot role of MP-LEVER-TYPE1 is shared by the two lever objects (1), and is represented statically as a pinned connection. Thus either pair of MP-LEVER/MP-BLADE primitives can be applied for cutting if the other pair is unable to move. The second function (2) also shows that the entire device can be used as a generalized MP-BLADE, or that one component can instantiate MP-BLADE, for puncturing, if the applied force is longitudinal.

Scissors Pragmatics

The pair of scissors has two design-intended uses: (1) poking holes, and (2) shearing, both

of which are SOP plans. As a simple compound device, using a pair of scissors requires the repetitive application of two subplans: opening, which enables the shear-apart function; and closing, which initiates the shear-apart function, as shown in Fig. A.37. The device is used by (a) opening the blades, (b) inserting the material between the blades, and (c) closing the blades on the material. As the blades impinge on the material, a cut is made which propagates along the length of the blade. The preparation (OPENing) of the scissors involves grasping the two handles and then pulling them apart, as shown in the shaded portion of the figure, at (2). These actions are represented individually, so that their directions can be shown. The UP and DOWN directions are not very effective in this example, since neither UP nor DOWN is at all necessary. The insertion of material is labeled (3). Unlike the directions shown for pulling the handles apart, when the handles are pressed together (at 4), the directions are well-defined, since they are implicitly defined by the other handle.

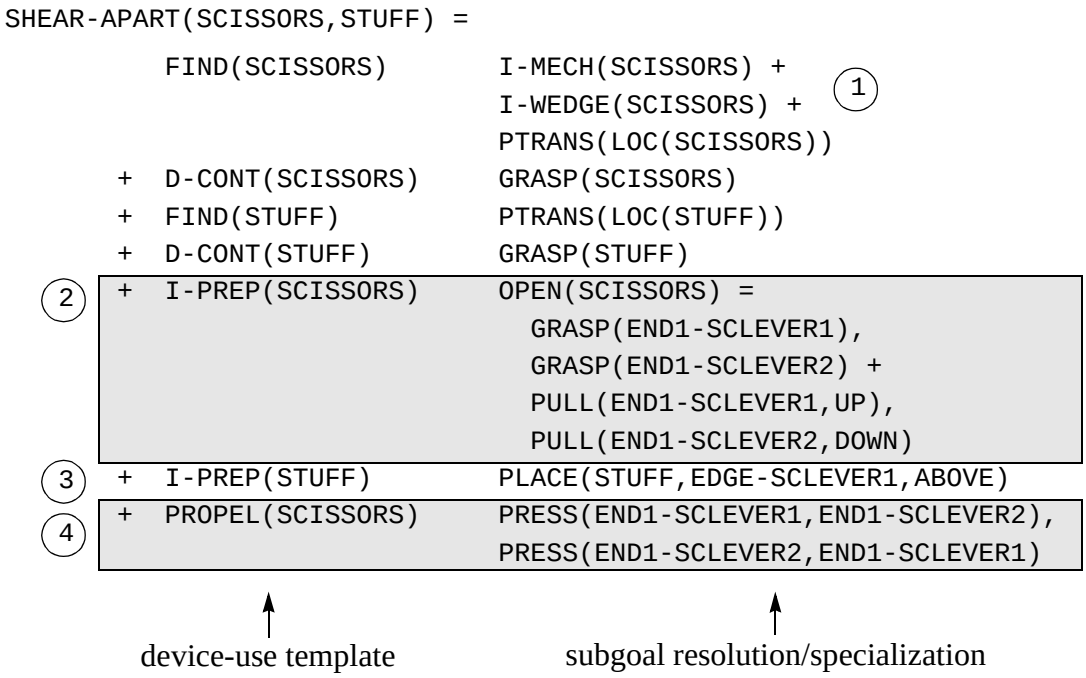


Figure A.37 Pragmatic representation scissors shearing.

In this example, both the comma (,) and plus (+) notation are used to describe simultaneous and sequential events, respectively As mentioned above, the instrumental mechanical goals (1) assure retrieval of a device which can both apply the force and cut the material.

The second use of a pair of scissors, to poke a hold in some stuff, is illustrated in Fig. A.38. In this latter example, the application is very straight forward, because the pair of scissors is being used as a simple device. The poke-hole plan makes use of the scissors blades being pointed and not on the device's ability to produce mechanical advantage. The one distinction is that the device must be pointed in the correct direction, toward the stuff

(6), in order to puncture it.

```
POKE-HOLE(SCISSORS,STUFF) =
    FIND(SCISSORS)              I-WEDGE(SCISSORS) +          (5)
                                PTRANS(LOC(SCISSORS))
  + D-CONT(SCISSORS)            GRASP(SCISSORS)
  + FIND(STUFF)                 PTRANS(LOC(STUFF))
  + D-CONT(STUFF)               GRASP(STUFF)
  + I-PREP(STUFF)               HOLD(STUFF)
  + I-PREP(SCISSORS)            PLACE(PT-SCISSORS,STUFF)     (6)
  + PROPEL(SCISSORS)            PUSH(SCISSORS,STUFF)
```

device-use template — subgoal resolution/specialization

Figure A.38 Pragmatic representation scissors poking.

A.2.2 Pliers

A pair of pliers is illustrated in Fig. A.39. Like the scissors, the pliers is comprised of two components pinned together. There are two primary static differences between the devices. First, instead of blades the plier components have jaws at one end.

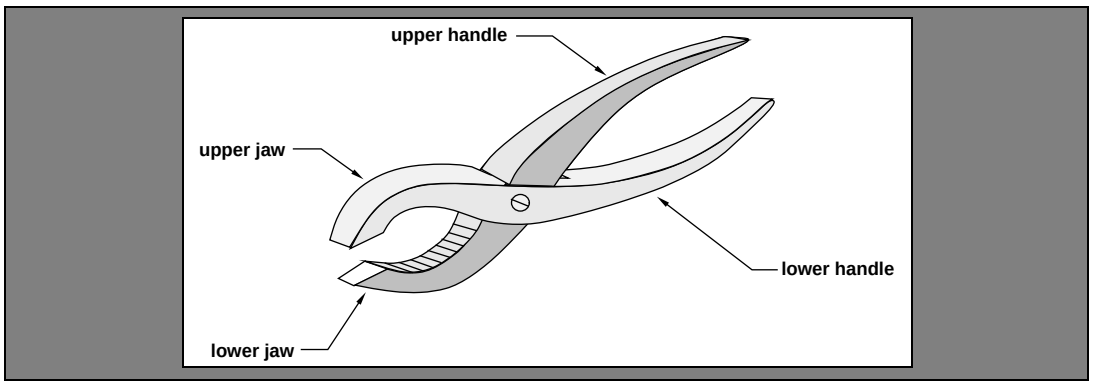


Figure A.39 Pliers illustration.

Second, the plier jaws are opposed rather than parallel.

Pliers Statics

The idealized version of the pliers is shown in Fig. A.40.

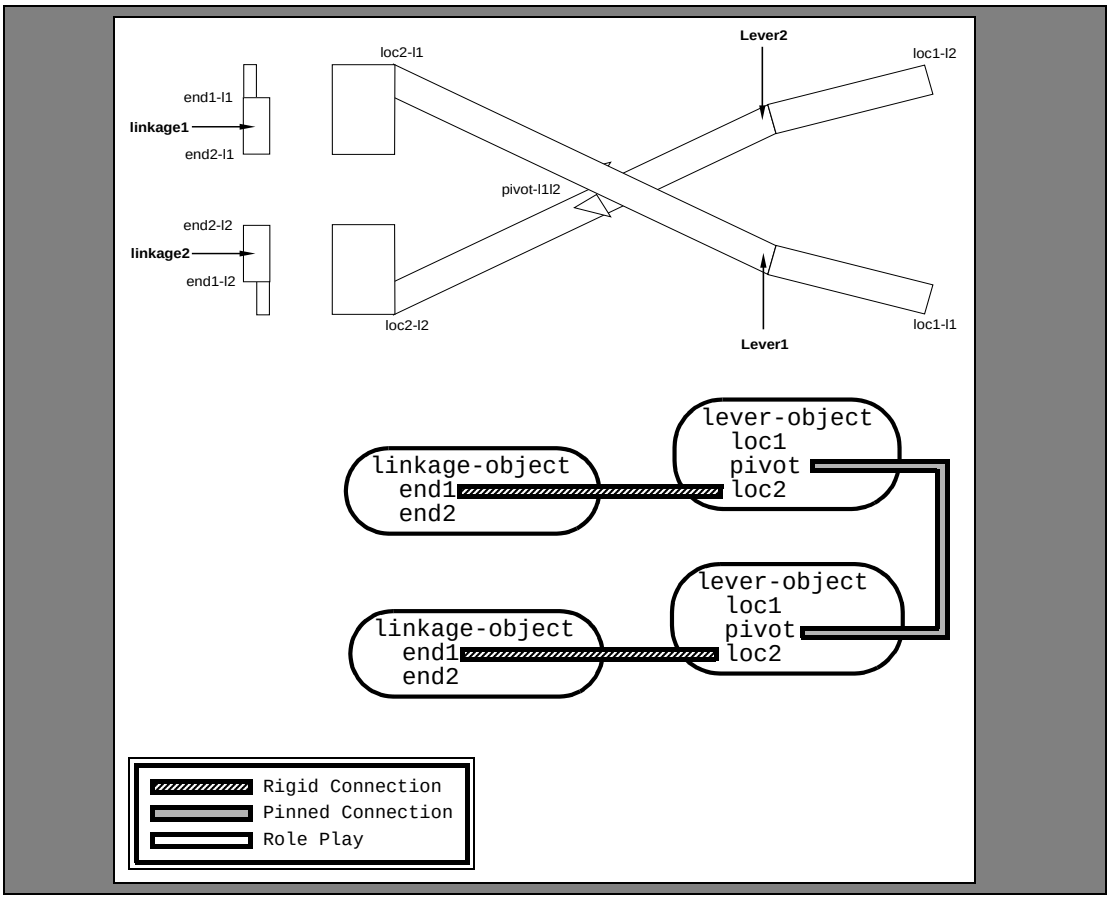


Figure A.40 pliers object primitives. Two levers and two linkages.

Each component is represented as a lever-object rigidly connected to a linkage-object, as shown in the accompanying SDD of Fig. A.40.

In Fig. A.40, the linkage-object is used to represent the plier component jaw region and is rigidly connected to the lever-object `loc2` region. The linkage-object and lever-object are colinear in orientation, but offset so that the jaws of the different components can make contact. This orientation difference, and the simplification of the handle representation (no holes for hands), are the primary static differences between the pliers and scissors. The ori-

entations are represented in Fig. A.41 below.

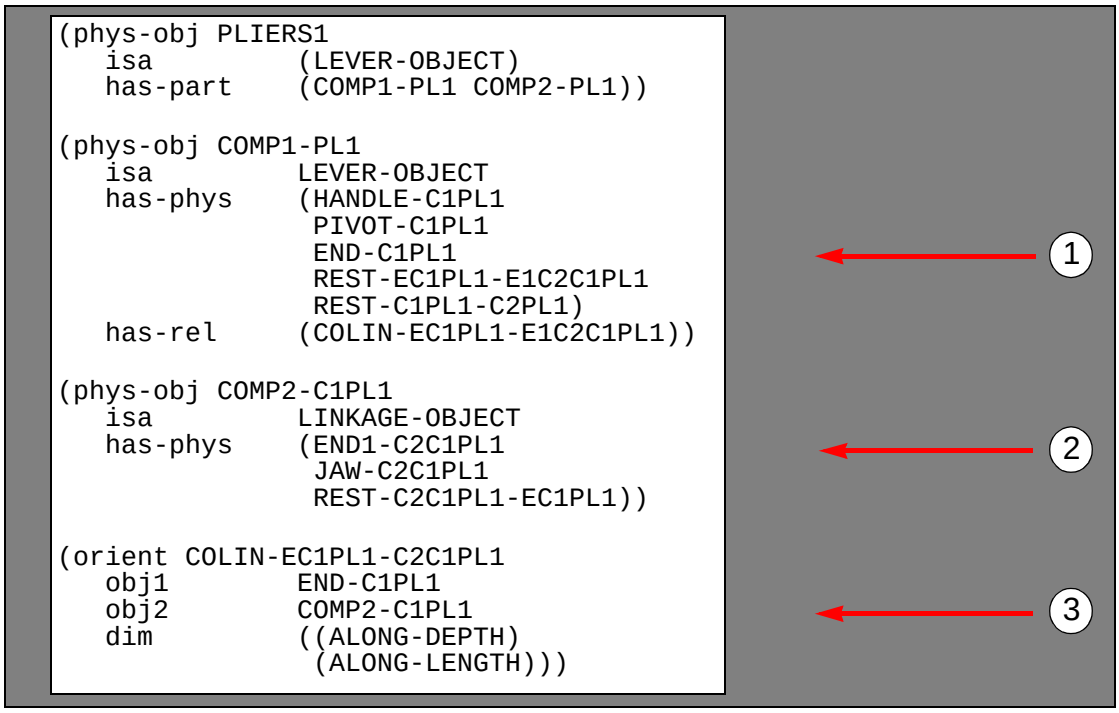


Figure A.41 Static representation for plier jaw orientation.

To represent the orientation between the lever-object and the linkage-object, the loc2 lever-object region and the linkage-object dimensional axes must be represented. The connection between the two objects provides a dimension of comparison. The two plier components each have a handle, a pivot, and a jaw region (1), and restraint states with their respective linkage-object and the other lever-object. The linkage-object (COMP2-C1PL1) is represented (2) with an end region and a jaw (end) region. The orientation is defined (3) between the lever-object ALONG-DEPTH dimension and the linkage-object ALONG-LENGTH dimension. The notion of colinearity arises when the two objects are connected in the proximity of the orientations definition. That is why both colinear and parallel orientations can be represented with similar instantiations.

A pair of pliers is used to hold or restrain objects and employ mechanical advantage to do so. Pliers are physically similar to a pair of scissors, the difference being that the plier components flat rather than sharp.

High-Level Pliers Dynamics

The clamping pliers function is depicted in Fig. A.42. The pliers and scissors are very similar physically, as were the spoon and shovel, and their MP-Diagrams demonstrate that their functions are also similar. The physical difference between the shape of the scissor blades and the plier hammers shows up in the whether MP-BLADE is used to represent the function or whether MP-LINKAGE is used to represent the function. In both devices/func-

tions, the pivot region is shared by the two components.

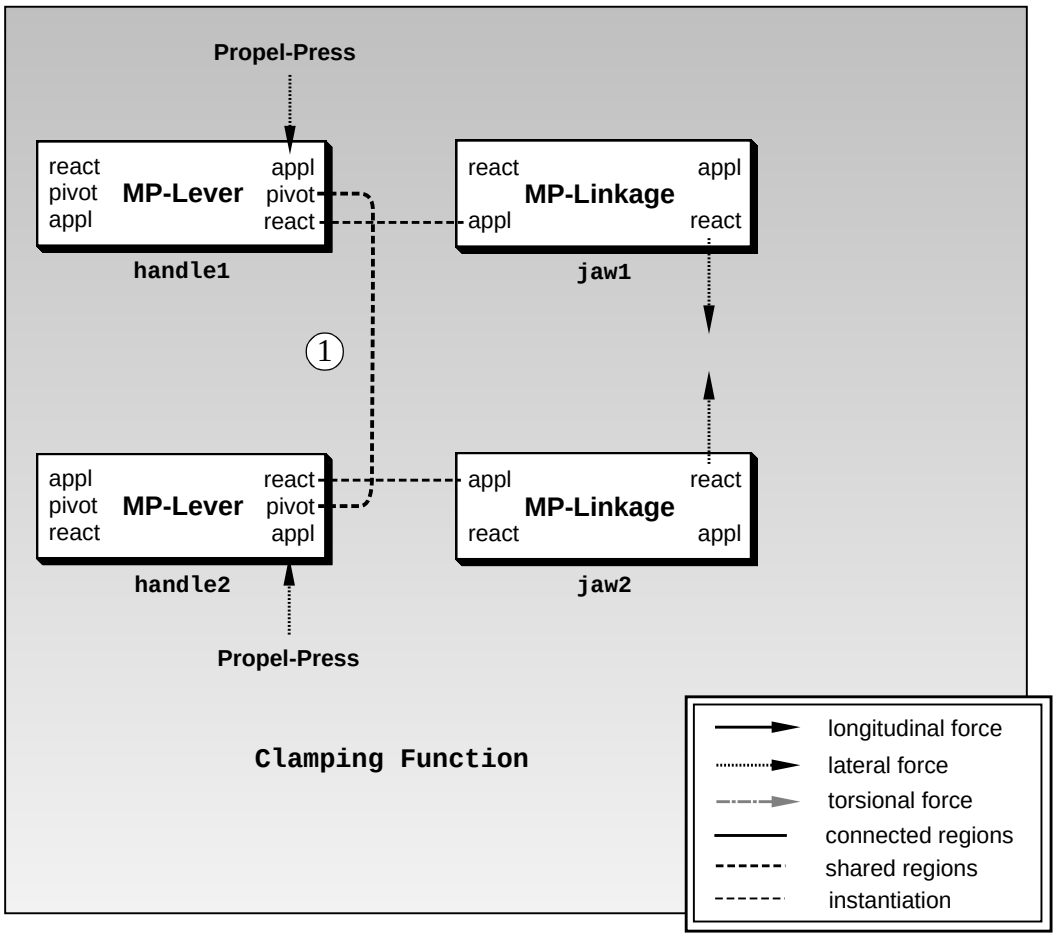


Figure A.42 MP-Diagrams for plier functions.

Pliers Pragmatics

The pair of pliers has a single design-intended use: (1) gripping objects together, as shown in Fig. A.43. The gripping plan is a CROM plan used to confine and control an object. Despite their static and dynamic similarities, a pair of pliers and a pair of scissors are used to achieve different ends. These differences show up in the types of plans they are intended

to address, and in the types of machines they instantiate in enabling these plans.The scis-

```
GRIP-TOGETHER(PLIERS,STUFF) =
(1)    FIND(PLIERS)          I-MECH(PLIERS) +
                             I-CONTN(PLIERS) +
                             PTRANS(LOC(PLIERS))
     + D-CONT(PLIERS)        GRASP(PLIERS)
     + FIND(STUFF)           PTRANS(LOC(STUFF))
     + D-CONT(STUFF)         GRASP(STUFF)
(2)  + I-PREP(PLIERS)        OPEN(PLIERS) =
                               GRASP(END1-PLLEVER1),
                               GRASP(END1-PLLEVER2) +
                               PULL(END1-PLLEVER1),
                               PULL(END1-PLLEVER2)
(3)  + I-PREP(STUFF)         PLACE(STUFF,END2-PLLINKAGE1,ABOVE)
(4)  + PROPEL(PLIERS)        PRESS(END1-PLLEVER1),
                             PRESS(END1-PLLEVER2)
```

device-use template

instantiation of subgoals

Figure A.43 Pragmatic representation of pliers gripping.

sors use is to separate objects whereas the pliers use is to control objects. In Fig. A.43, the difference is identified by the I-CONTN goal, as opposed to the I-WEDGE goal for the scissors. Both devices restrain their object (at 3), however, the application of force is used differently in each. In other respects the plans are identical.

A.2.3 Door

A door physically blocks the motion of objects into or out of a container.

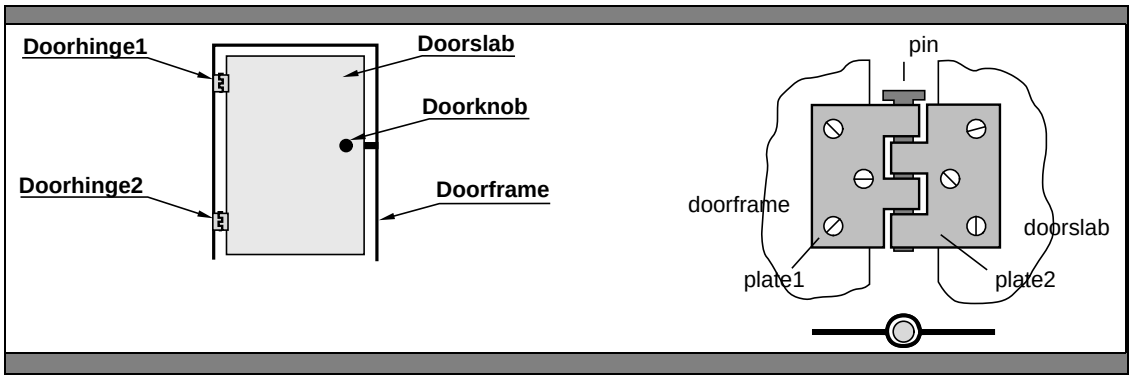


Figure A.44 Door amd doorhinge illustration.

Door Statics

A simple door is comprised of four components: a piece which moves, called a doorslab, a doorframe, and two hinges. When the door is open, motion past the door is enabled. When the door is closed it isn't. The doorslab is generally a solid object which functions to trans-

mit force and motion, so the doorhinges are significant components in the device. An idealized doorhinge and its static device diagram is shown in Fig. A.45.

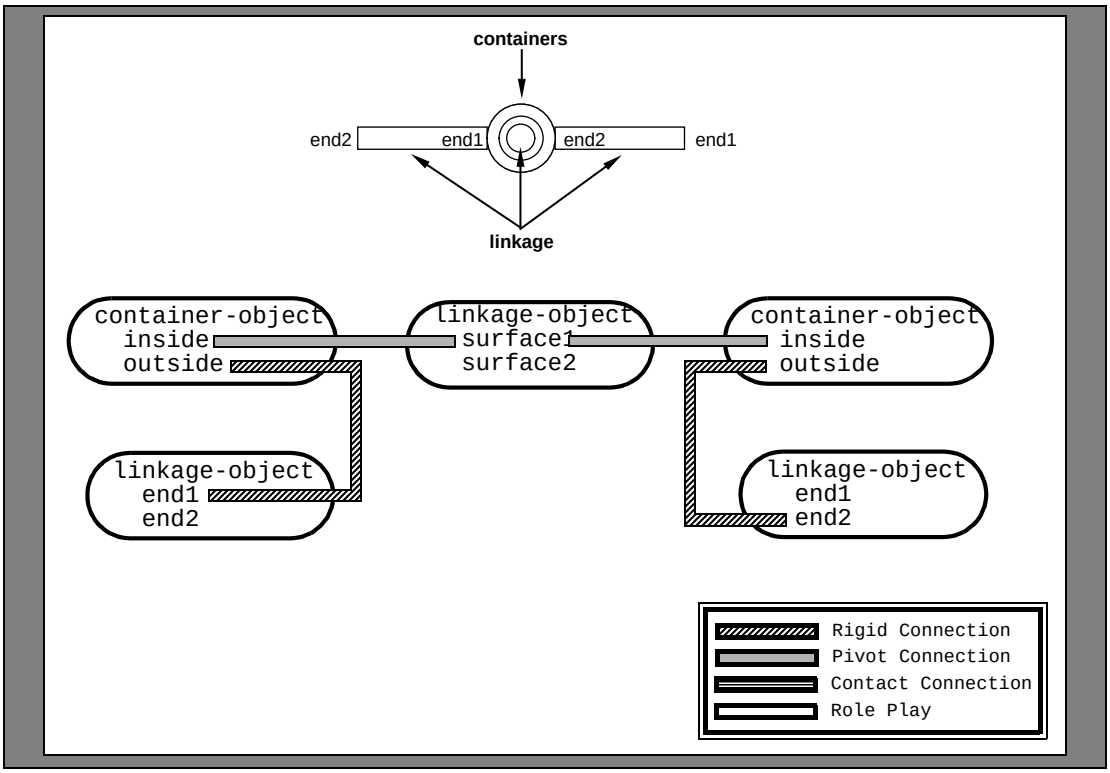


Figure A.45 Doorhinge object primitives and static device diagram.

The doorhinge is comprised of three components: two plates and a pin. The pin is contained by containers from each plate, which restricts translational motion of each plate, but allows rotational motion of each plate. Thus the connection is represented as a pinned connection.

High-Level Door Dynamics

The representation of doorhinge function is shown in Fig. A.46.

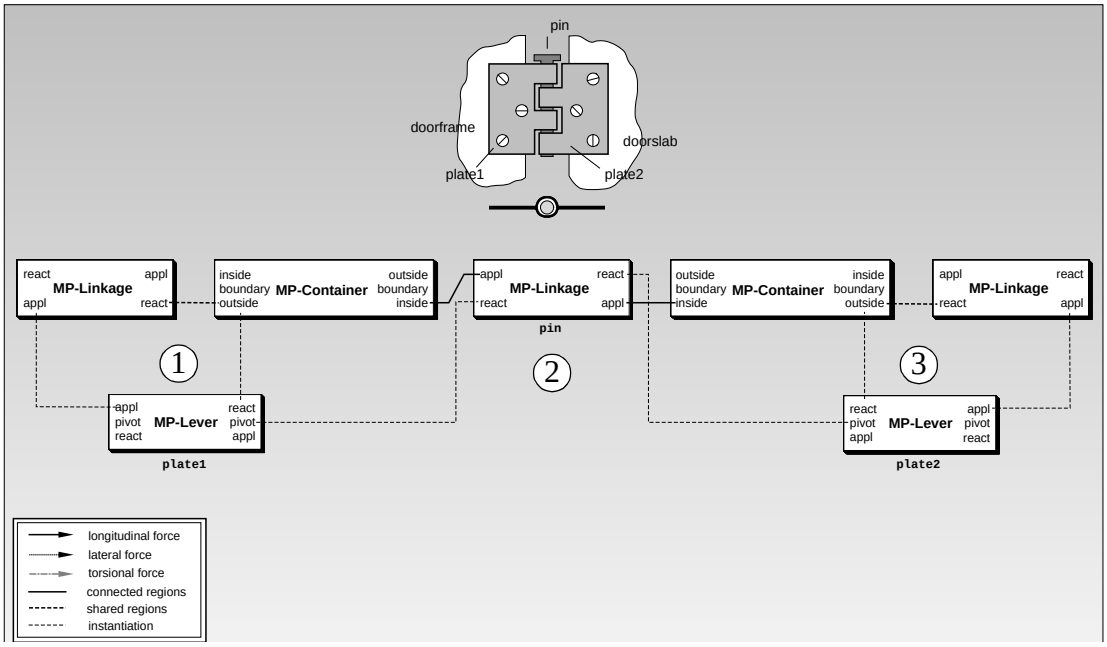


Figure A.46 MP-Diagram for doorhinge function.

Alone each doorhinge plate function is represented with MP-LINKAGE (at 1 and 3). However, by sharing the common hinge pin (PIN, at 2), the two plates (PLATE1 and PLATE2) instantiate mutually dependent MP-LEVER-TYPE2 devices. Without the other plate, neither hinge plate's motion is contrained by the hinge pin and cannot function as a MP-LEVER. However, with this caveat, i.e., as long as the plates remain pinned, both PLATE1 and PLATE2 can be considered as levers independently. This representation can be used in the overall door dynamic representation by replacing the linkage/container object with the lever object. An idealized version of the simple door and its static device diagram are illustrated in Fig. A.47. In this figure, the doorhinge is idealized to a single lever

object, since the second plate is rigidly connected to the doorframe.

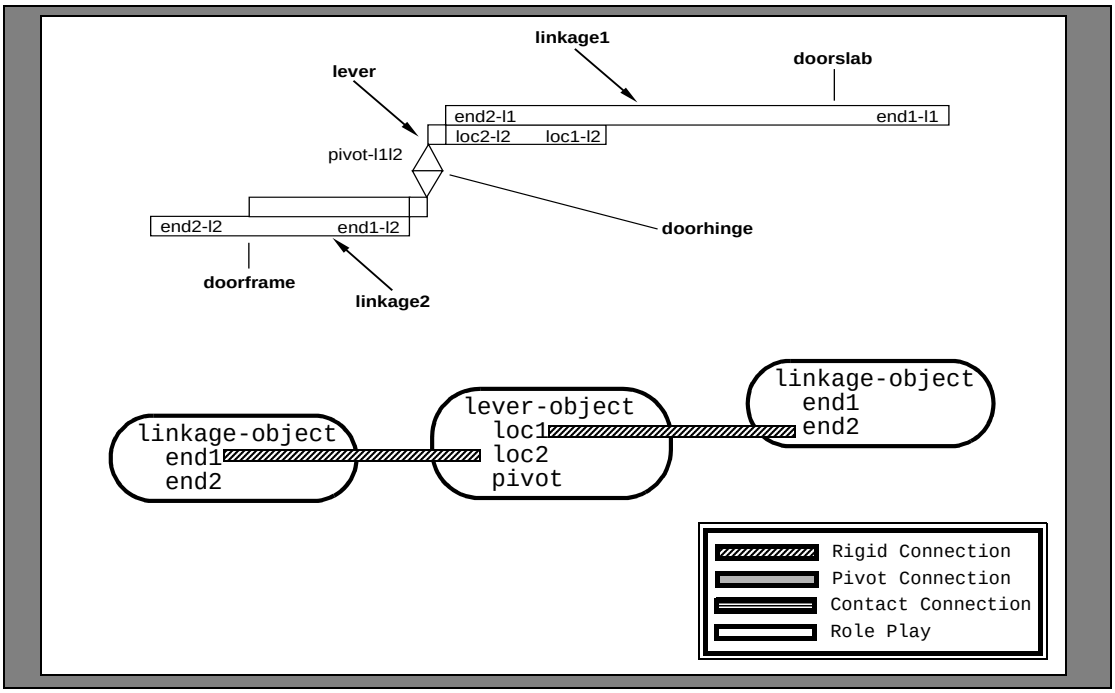


Figure A.47 Door object primitives and static device diagram.

The door opening function illustrated in Fig. A.48 presents both doorhinge plates, and also represents the doorknob function. In this figure, the doorknob function of releasing the latch is represented with MP-WHEEL-AXLE-TYPE2. This function is initiated by turning the doorknob. When the latch is released, the door opening function is initiated by pulling the doorknob, in which case the force is transmitted directly to the doorslab, which is represented with MP-LINKAGE (2). The doorslab is connected to the hinges, so the resulting force is transformed via MP-LEVER (3) to the doorframe.The function of the slot in the doorframe is to restrict the motion of the door, so it is represented with MP-CONTAINER

(4).

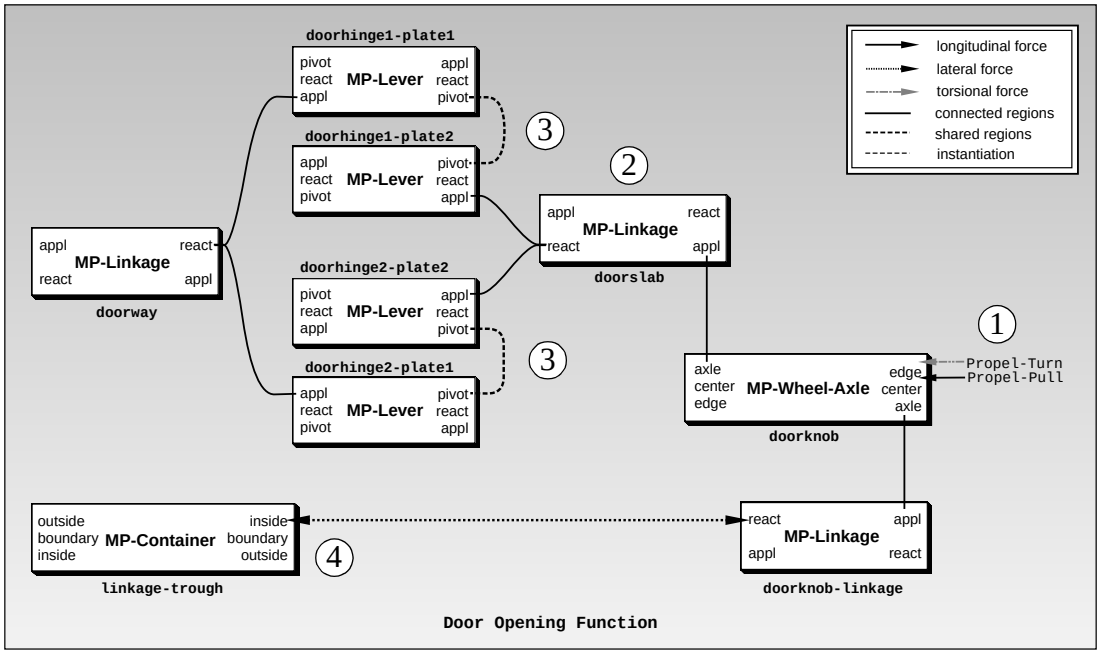


Figure A.48 MP-Diagram for door opening function.

Door Pragmatics

The door has two design-intended uses: (1) secure an environment, and (2) access an environment, both of which are containment (i.e., CROM) plans. Access makes use of the door opening function, while security and containment use the door closing function. The containment use plan is illustrated in Fig. A.50.

Access means that we activate movement of the door so as to create an opening through which objects may pass. This is accomplished by grabbing and turning the doorknob and then pushing the door. Because the door is part of the environment, the environment need

```
ACCESS-ENVIRONMENT(DOOR,ENVIRONMENT) =

      FIND(DOOR)              PTRANS(LOC(DOOR))
+     D-CONT(DOOR)            GRASP(KNOB-DOORSLAB)
+     I-PREP(DOOR)            DISENGAGE-LATCH(DOOR) =
                                TURN(KNOB-DOORSLAB)
+     PROPEL(DOOR)            PULL(KNOB-DOORSLAB)

      device-use template     subgoal resolution/specialization
```

Figure A.49 Pragmatic representation of door environmental access.

not be found or controlled. The access-environment(door,environment) plan thus degenerates to a use(door) plan, but makes use of the DISENGAGE-LATCH plan, which is also a containment plan.

Door use for environmental containment is illustrated in Fig. A.50. Environmental containment means that by closing the door, the means of passage is removed, disabling entry and exit. This plan also degenerates to a use(door) plan.

```
CONTAIN-ENVIRONMENT(DOOR,ENVIRONMENT) =

      FIND(DOOR)              PTRANS(LOC(DOOR))
+     D-CONT(DOOR)            GRASP(END1-DOORSLAB)
+     PROPEL(DOOR)            PUSH(END1-DOORSLAB)

        ↑                          ↑
device-use template      subgoal resolution/specialization
```

Figure A.50 Pragmatic representation of door environmental containment.

A.3 Multiple Compound Devices

Whereas a pair of scissors or a clothes pin have two components which can move independently, a multiple-compound device has more than one independently moving component. Although the statics and intra-object dynamics are fairly straightforward, the inter-object dynamics begins to become complex when fully elaborated. One such device is the fingernail clipper, which is the first of three multiple compound devices represented in this section.

(1) press
(2) crank can opener

A.3.4 Press

A press modifies the shape and size of objects with a screw. The press functions by disabling motion in a container and then reducing the volume the object can occupy.

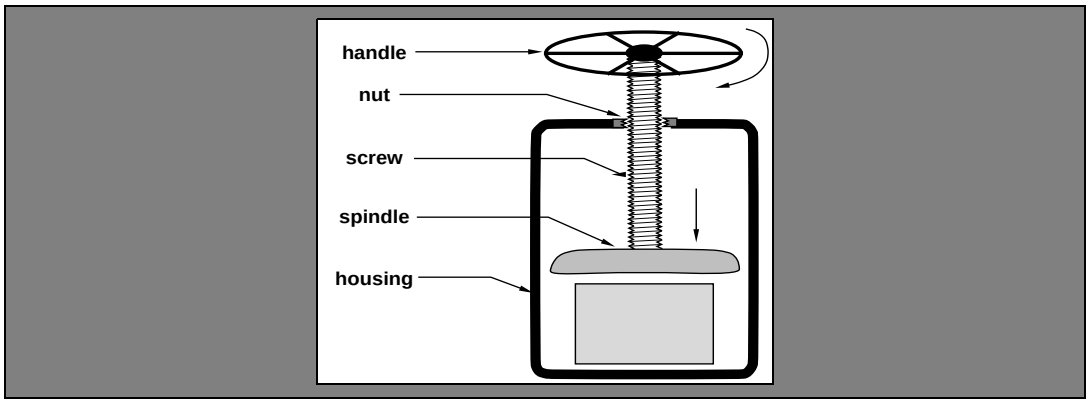


Figure A.51 Press illustration.

Press Statics

An idealized press and static device diagram are illustrated in Fig. A.52.

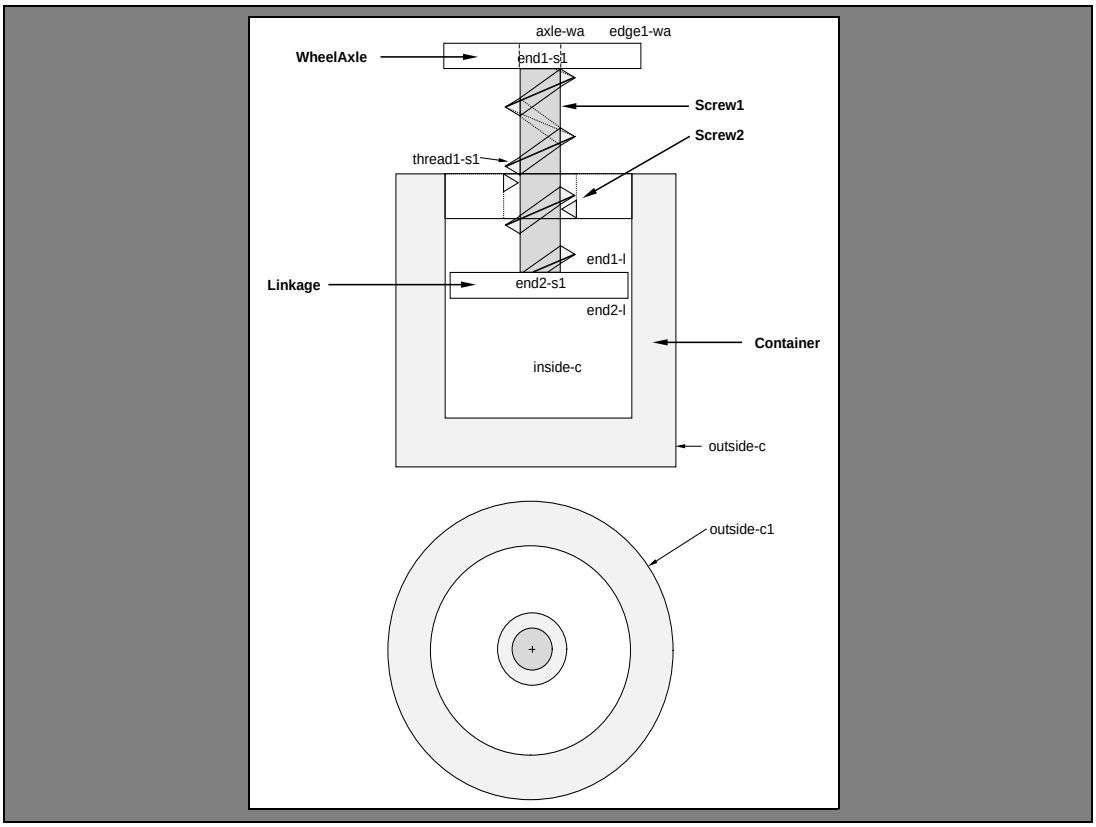


Figure A.52 Press object primitives.

.

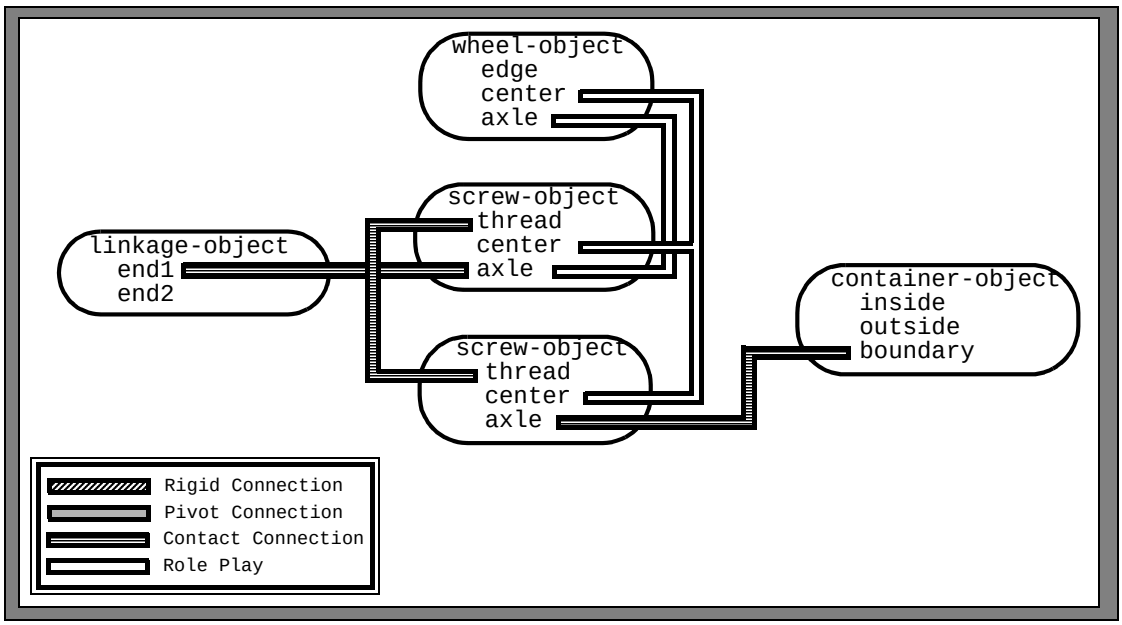


Figure A.53 Press static device diagram.

High-Level Press Dynamics

The pressing function is shown as an MP-Diagram in Fig. A.54. The device function is initiated by a torsional force on the handle rim (EDGE). This is represented with MP-WHEEL-AXLE. The wheel's axle is shared with that of a screw (2), which magnifies force

and transforms it to longitudinal force using MP-SCREW.

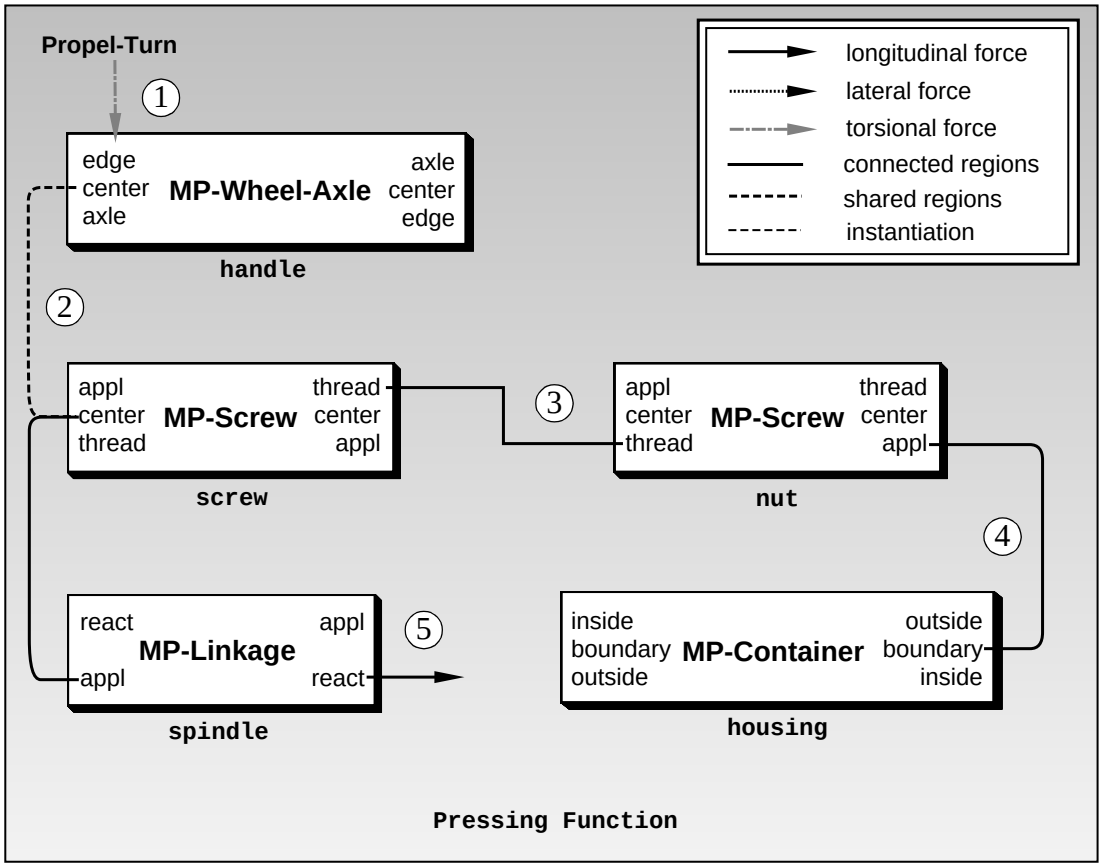


Figure A.54 MP-Diagram for press function.

The screw threads are in contact with the threads of the nut (3), which is represented with another MP-SCREW. The nut is connected to the housing (4), and the spindle which is connected to the screw is within the housing boundary, the it is represented with MP-CONTAINER. As the wheel is turned back and forth, the spindle moves up and down in the housing.

Press Pragmatics

A press has two design-intended uses, to: (1) compress objects together or apart (SOP plan), and (2) extrude material (SO plan). The press illustrated is a container object which has as a component a plane object (a screw). Each of the plans makes use of the fact that a press can apply great force on an object, because the object is confined, and because the screw can magnify force. The press-together plan is illustrated in Fig. A.55. In this plan, the press must be opened so that the objects can be placed inside. Once inside, the shape of spindle (linkage in Figure B.69), container, and objects determines whether the objects are forced together, apart, or extruded. For example, if the objects are a ring and a bearing, and the bearing is resting on the edge of a hole in the ring, and the spindle is flat, then the bearing will be pressed into the ring's hole. If, on the other hand, the bearing is already in the ring, the ring is supported, and the spindle is the shape of the bearing but smaller, then the

bearing and ring will be pressed apart. The plan is invoked in basically the same way for

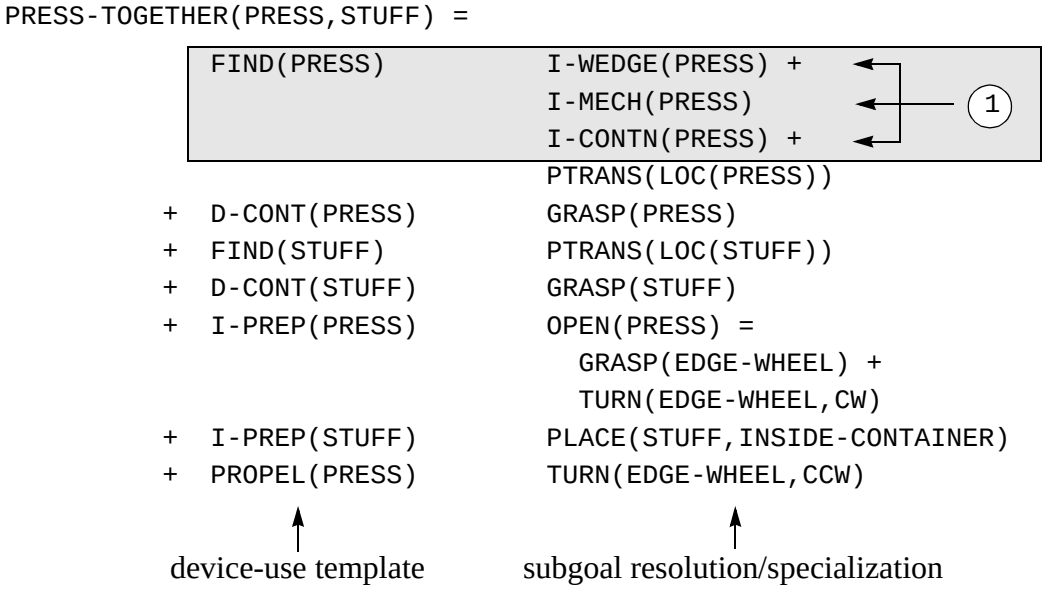


Figure A.55 Pragmatic representation of press together.

both. The press-together plan is similar to the extrude-stuff plan in that motion of an object under pressure occurs, but in the former plan, motion is limited to the confines of the container. In the extrude plan, there is an opening in the container which allows a pressure release as long as the object changes shape and moves through the opening.

A.3.5 Crank Can-Opener

A crank can-opener is illustrated in Fig. A.56.

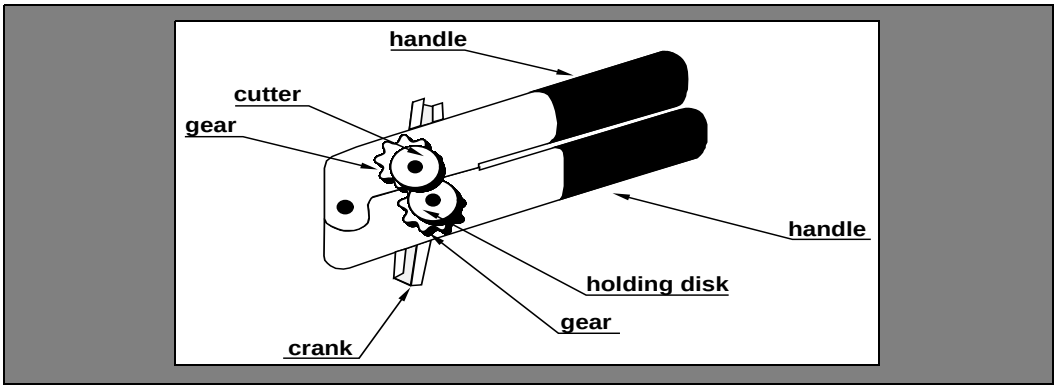


Figure A.56 Crank can opener illustration.

Crank Can Opener Statics

An idealized can opener and static device diagram are depited in Fig. A.57.

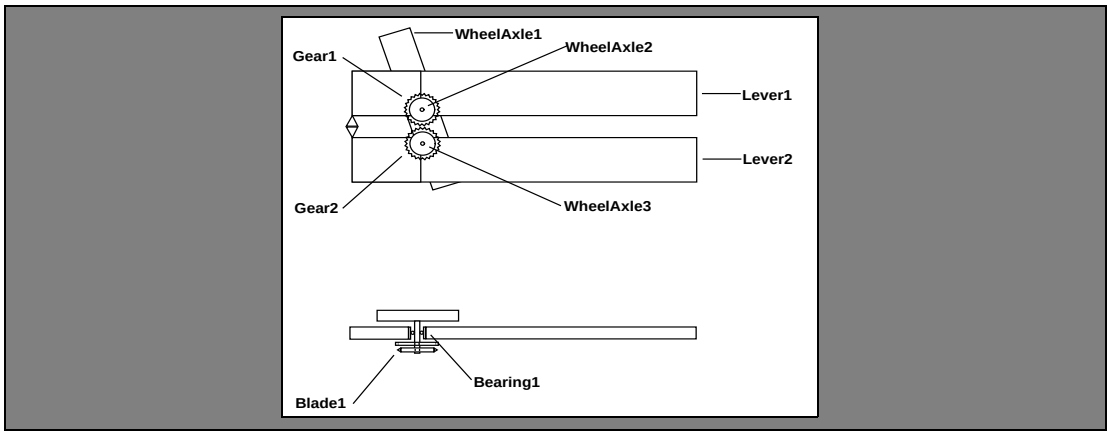


Figure A.57 Crank can opener object primitives.

The crank can-opener SDD is shown in Fig. A.57:

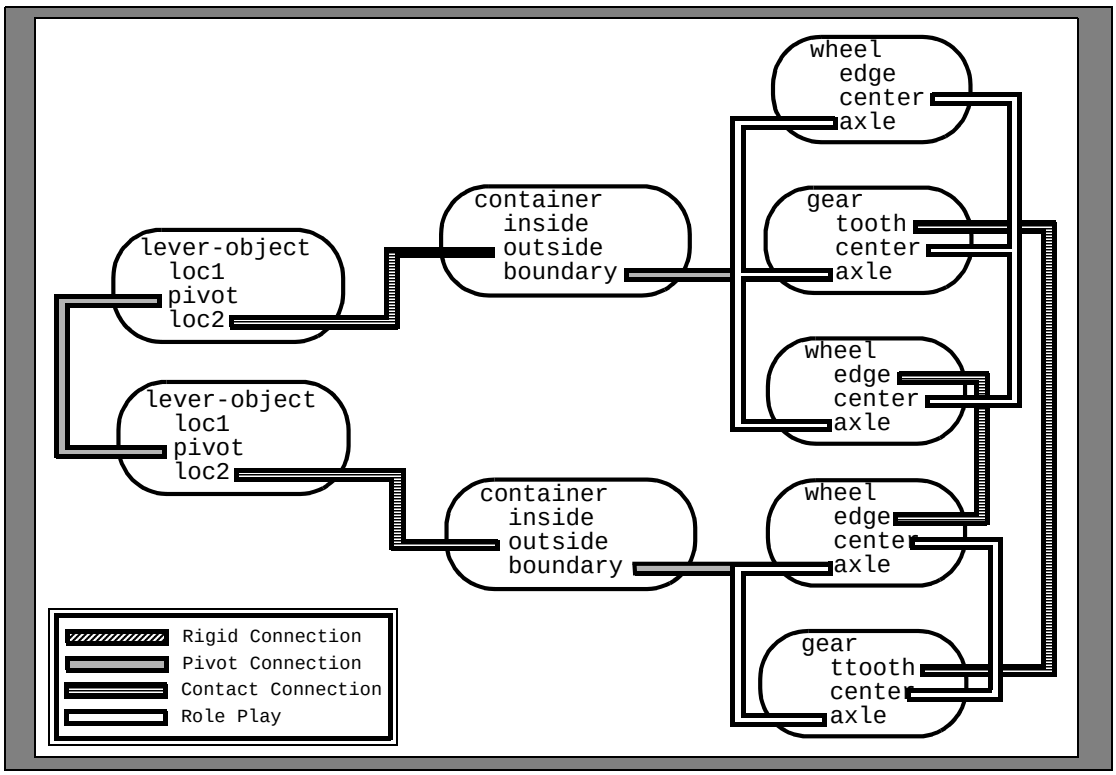


Figure A.58 Crank can opener static device diagram.

High-Level Crank Can-Opener Dynamics

The crank can-opener slicing function is represented as an MP-Diagram in Fig. A.59. The LINKAGE1 and LINKAGE2 components are represented with MP-LEVER-TYPE2, be-

cause of the common rivet. The crank assembly (CRANK, HOLD-GEAR, and HOLD-DISK) axle and the cutter assembly (CUT-GEAR and CUT-DISK) axle also share axle roles of MP-WHEEL-AXLE (at 3 and 4). Rotational force is applied to CRANK (2), whose center is supported, so it is represented with MP-WHEEL-AXLE. Because of the shared axle, the resulting force is transmitted to HOLD-GEAR, and the toothed disk HOLD-DISK. The gears, HOLD-GEAR and CUT-GEAR, and toothed disk HOLD-DISK are represented with MP-GEARs, which enable position control and force transmission. Finally, the cutting disk CUT-DISK is represented with MP-BLADE, which enables the puncturing and slicing.

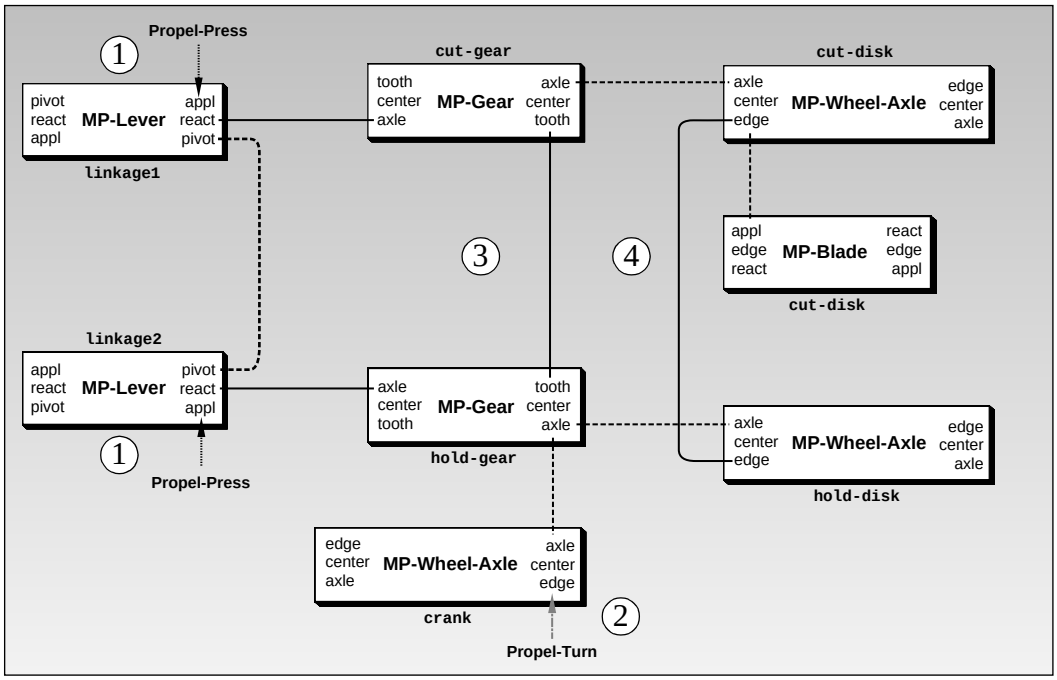


Figure A.59 MP-Diagram for crank can opener function.

The CRANK-CAN-OPENER can also be used to pound nails or crack nuts. These applications can be represented as generalized machine primitives. When the CRANK-CAN-OPENER is used to pound nails, the entire device is associated with a Type 3 MP-LEVER. When the CRANK-CAN-OPENER is used as to crack nuts, the entire device is associated with a Type 2 MP-LEVER. The CRANK-CAN-OPENER will always be associated with the subset of machine primitives applied to produce the desired effect, regardless of the design-intended device function. This notion is identical to what Schank and Abelson called the *main conceptualization* (maincon) of a script or plan. In FONM, a device's maincon is associated with a single behavioral process primitive for a machine primitive, such as DEFORM for MP-BLADE.

Crank Can-Opener Pragmatics

A crank can opener is designed for removing lids from cans (i.e., an SOP plan in support of a CROM plan). The crank can opener has many independently moving components, which form small systems which perform specific functions. For example, the handles provide leverage to initiate the cut in the can, and they house the remaining components. The crank, cut-gear and cut-disk transmit force to the can. The hold-gear and hold-disk restrain the can’s motion and transmit motion to the can. Each of these components and systems is

activated in the crank can opener function and enabled through its use, as shown in Fig. A.60.To prepare and use the crank can opener, five subtasks must be performed: (a) the

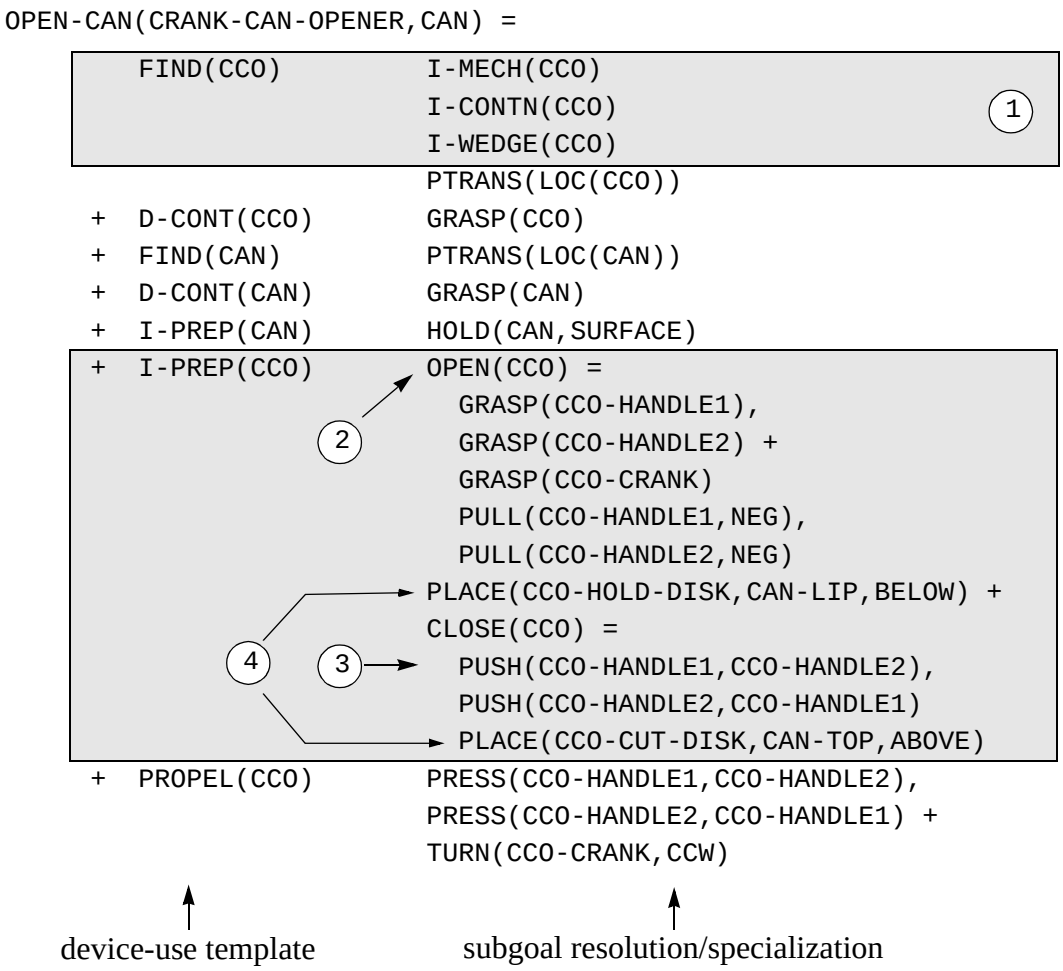


Figure A.60 Pragmatic representation of crank can opener can opening.

handles must be opened (2), (b) the HOLD-DISK must be oriented under the can lip (4), (c) the handles must be closed (3), so that the two gears engage and so that the CUT-DISK contacts the top of the lid (4), (d) the handles must be pressed together so as to puncture the lid, and (e) the crank must be turned so as to force the cut (i.e., SLICE) around the lid. The notation used to describe the direction the handles are pulled is based on the direction being away from the handle CG. The placement of the can opener about the can is represented as a preparation of the device rather than the can, since, once the can is held stable, it is prepared for cutting. The actual application of the device involves two kinds of input: (a) to initiate the cut, and (b) to continue the cut. The first enables the behavior BPP-DEFORM-PUNCTURE, and the second enables the behavior BPP-DEFORM-SPLIT, both conceptualization of MP-BLADE.

A.4 Complex Devices

Complex devices involve objects which may or may not be permanent components of the device, have complex motions, or have large numbers of components. One such example is a toy gun, which has a projectile which is only connected to the device when being armed and preparing to fire. Another complex device is a mousetrap, since the hook and baitplate motion are both confined to the limits imposed by their connectors, but free to move in every other respect. In this section, these two devices and two others are presented. The intent is to address some of the issues in representing such devices, rather than to represent them explicitly.

(1) corkscrew
(2) egg beater
(3) mouse trap

A.4.6 Corkscrew

A corkscrew is illustrated and labeled in Fig. A.61:

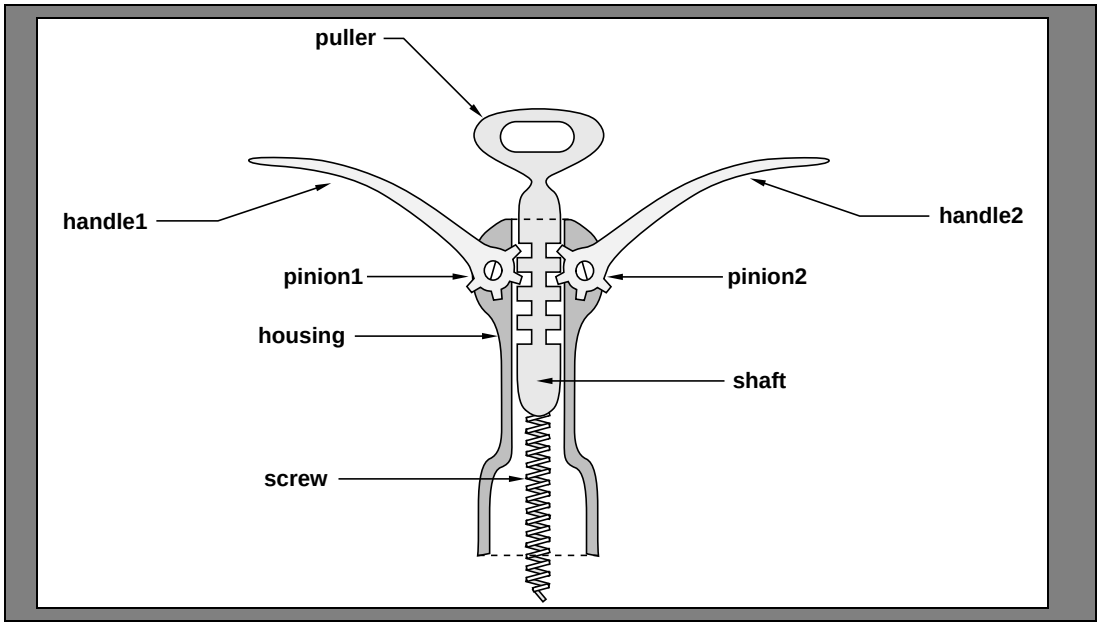


Figure A.61 Corkscrew illustration.

Corkscrew Statics

An idealized corkscrew and its static device diagram is illustrated in Fig. A.64.

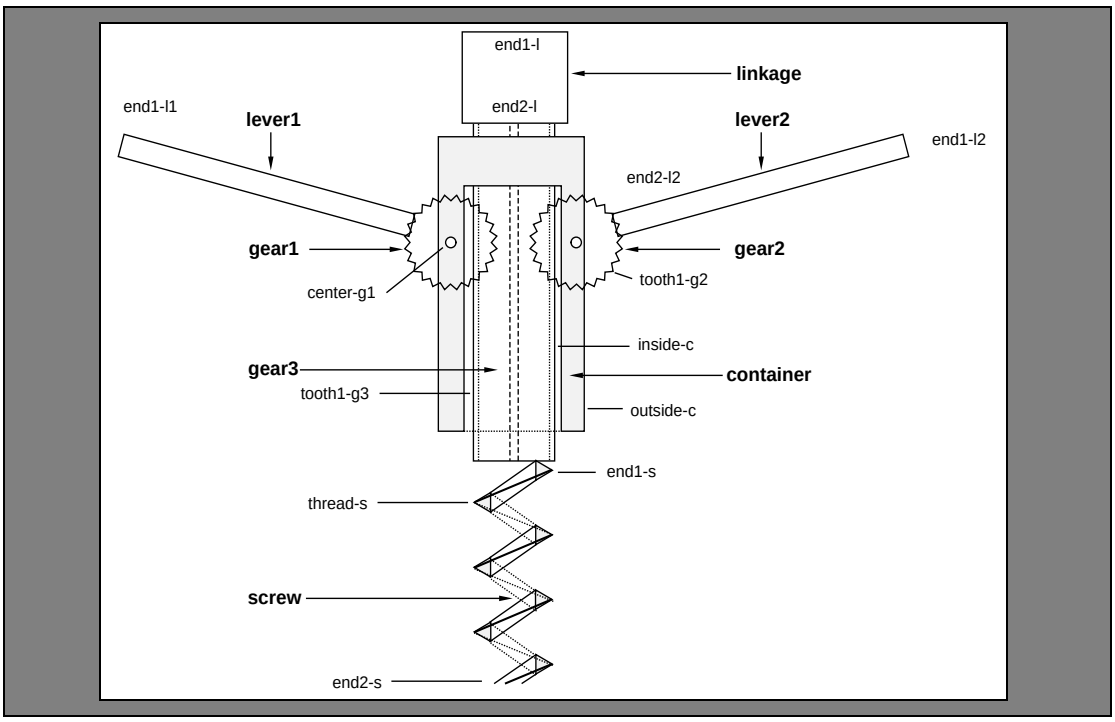


Figure A.62 Idealized corkscrew representation.

The corkscrew SDD is shown in Fig. A.63:

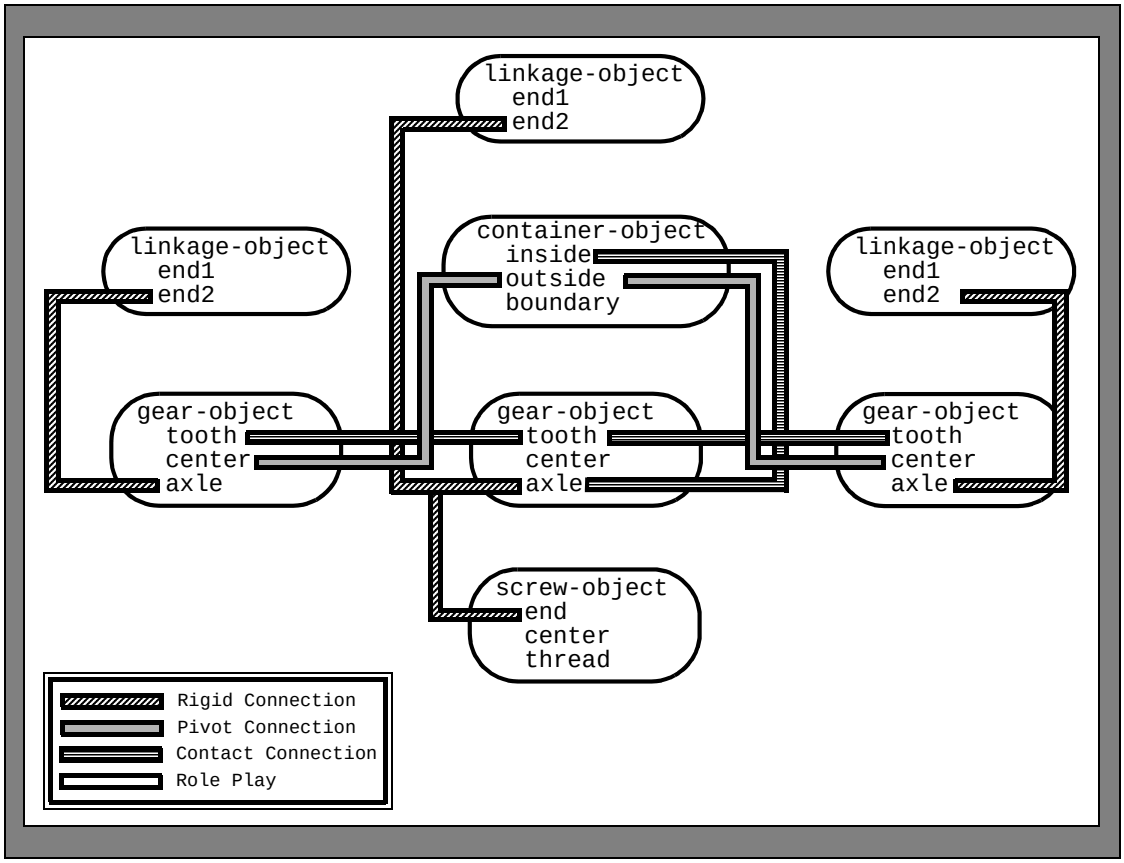


Figure A.63 Corkscrew static device diagram.

High-Level Corkscrew Dynamics

The corkscrew has two normal functions: a screwing function and a pulling function, both

of which are represented in Fig. A.64.

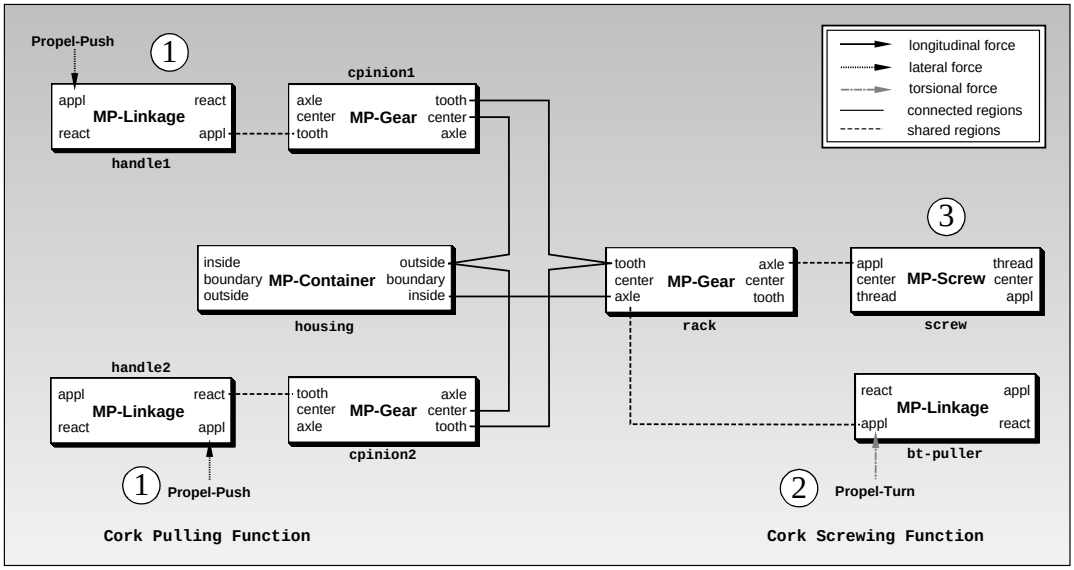


Figure A.64 MP-Diagrams for corkscrew functions.

Corkscrew Pragmatics

The cork screw has a single design-intended use: (1) remove corks, which is a control-range-of-motion (CROM) plan. The corkscrew is able to control the cork and open the bottle, because it utilizes a screw to burrow into and hold the cork, and then uses leverage to

pull the cork from the bottle, as shown in Fig. A.65.

```
OPEN-CORKED-BOTTLE(CORK-SCREW,BOTTLE) =
    FIND(CORK-SCREW)        I-CONTN(CORK-SCREW) +                    (1)
                            I-WEDGE(CORK-SCREW) +                    (2)
                            I-MECH(CORK-SCREW) +                     (3)
                            PTRANS(LOC(CORK-SCREW))                  (4)
+   D-CONT(CORK-SCREW)      GRASP(CORK-SCREW)
+   FIND(BOTTLE)            PTRANS(LOC(BOTTLE))
+   D-CONT(BOTTLE)          GRASP(BOTTLE)
+   I-PREP(BOTTLE)          PLACE(BOTTLE,SURFACE)
+   I-PREP(CORK-SCREW)      OPEN(CORK-SCREW) =
                              PUSH(CS-LEVER1,DOWN),
                              PUSH(CS-LEVER2,DOWN) +
                              PLACE(END2-CSBODY,TOP-BOTTLE,ABOVE) +
                              PLACE(CS-POINT,CORK,ABOVE) +
                              TURN(CS-HEAD,CCW)
+   PROPEL(CORK-SCREW)      PUSH(CS-LEVER1,DOWN)
                            PUSH(CS-LEVER2,DOWN)

   device-use template                instantiation of subgoals
```

Figure A.65 Pragmatic representation of cork pulling.

A.4.7 Egg Beater

An eggbeater is depicted in Fig. A.66:

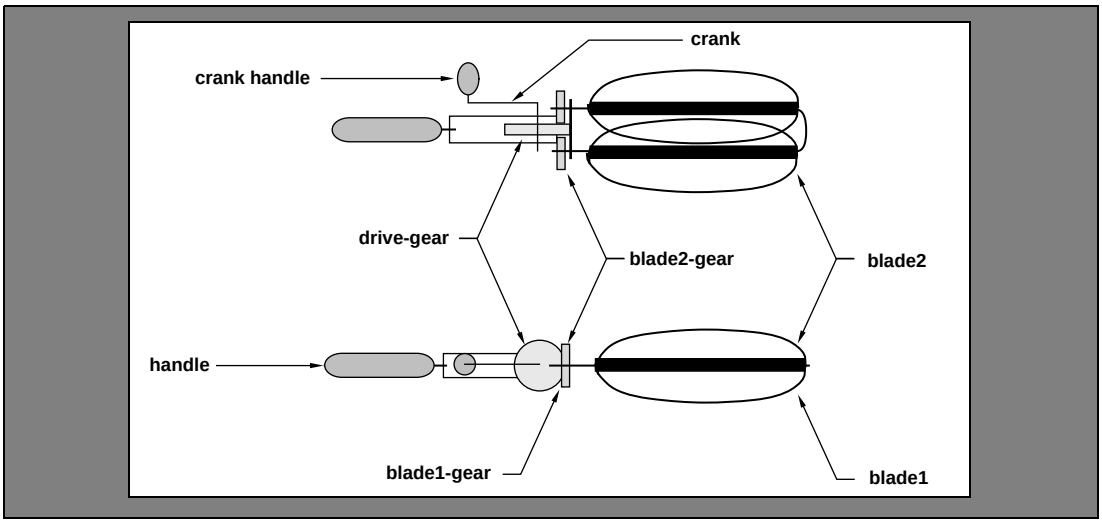


Figure A.66 Egg beater illustration.

Egg Beater Statics

An idealized egg beater is illustrated in Fig. A.67.

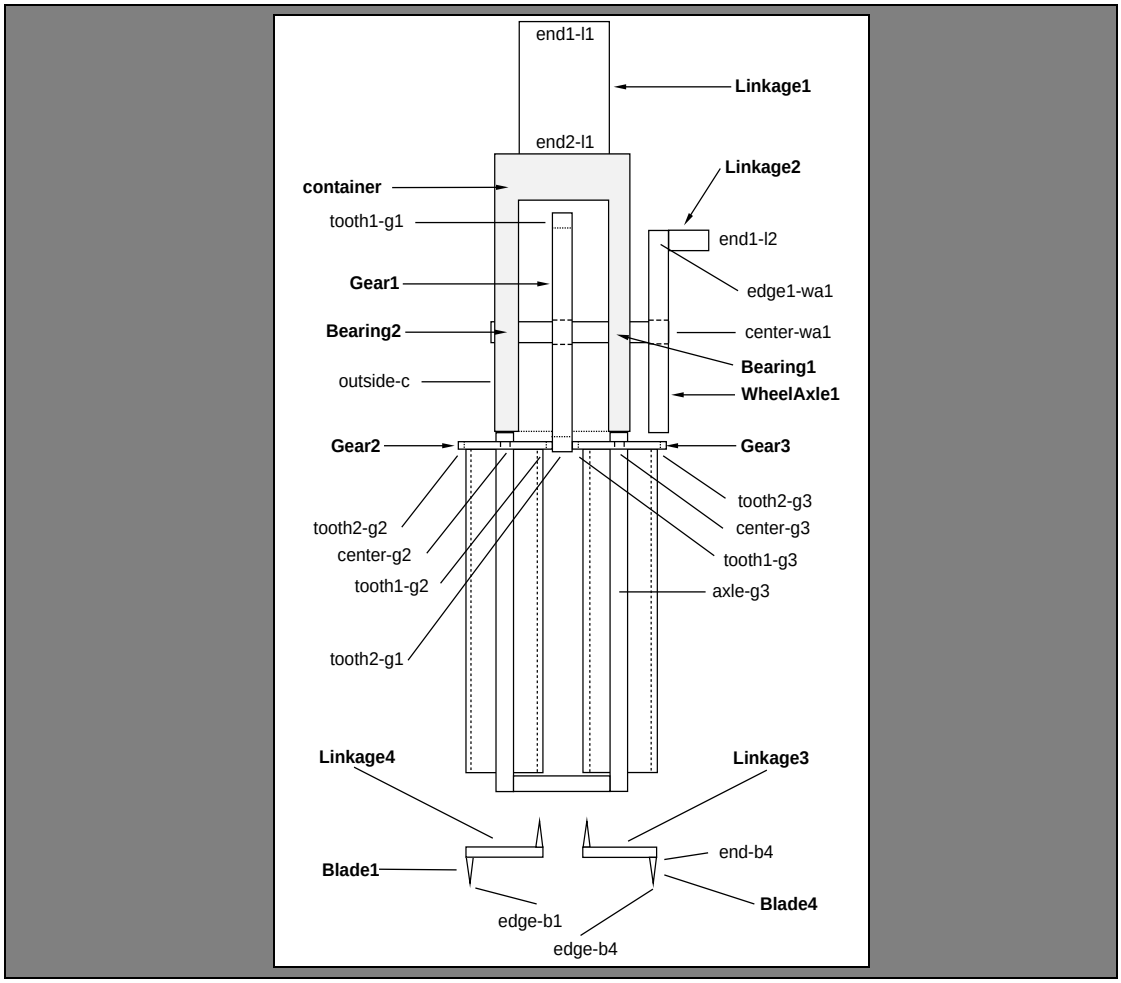


Figure A.67 Idealized representation of egg beater.

The egg-beater is a complex gear composition where the gears move in different rotational planes and effect the motion of counter-rotating blade-whips. The static device diagram for

the egg beater is shown in Chapter A.68.

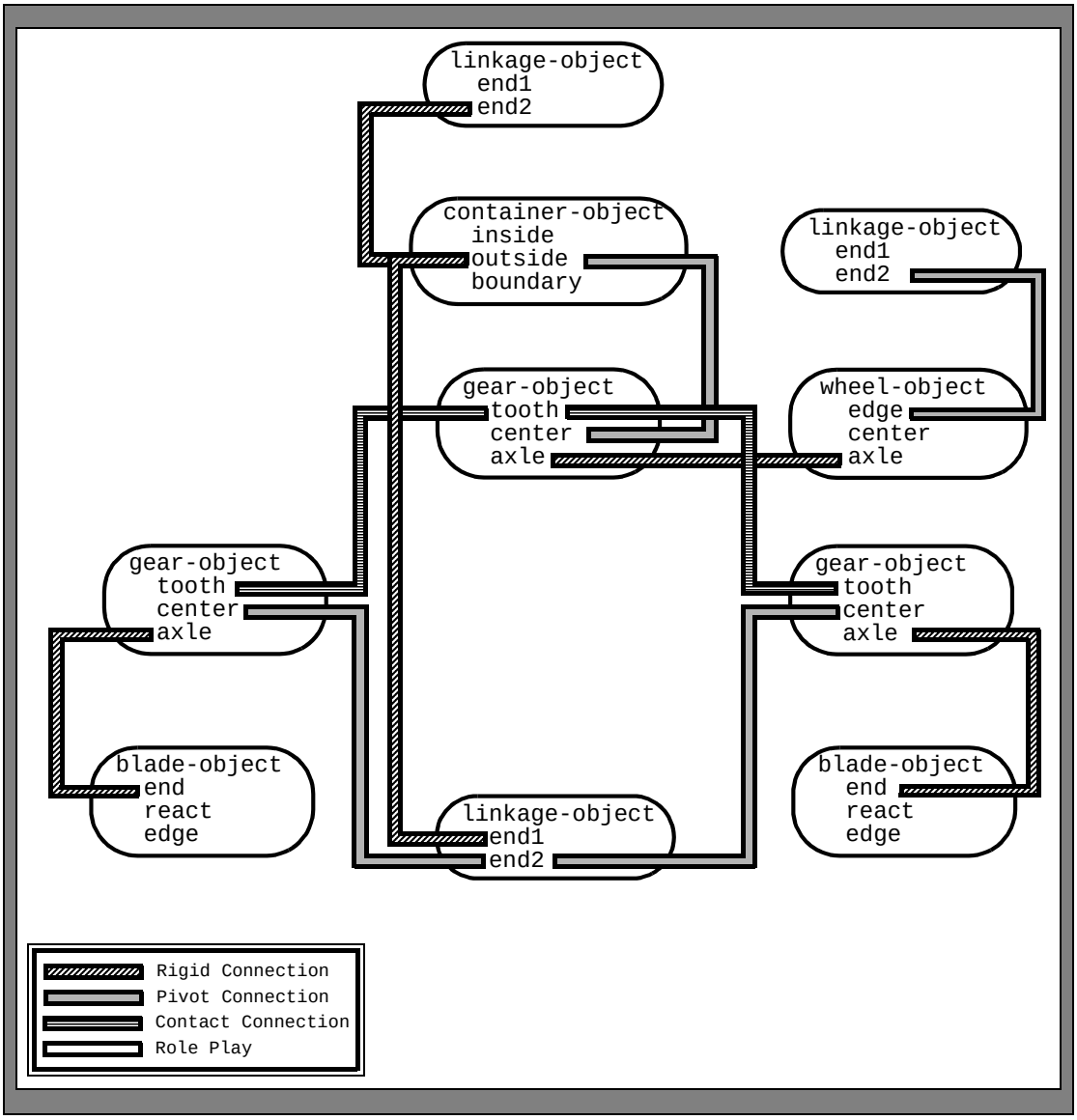


Figure A.68 Static device diagram for idealized eggbeater.

High-Level Egg Beater Dynamics

The MP-Diagram which represents the egg beater whipping function is shown in Fig. A.69.

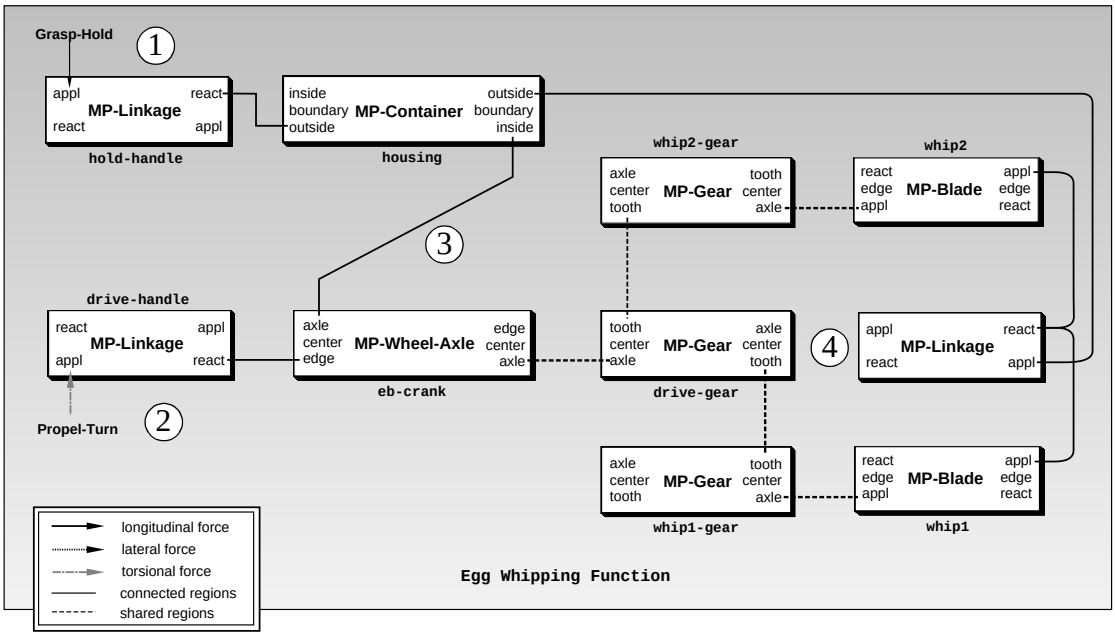


Figure A.69 MP-Diagram for egg beater function.

The egg beater presents a number of problems for mechanical representation. First, the counter rotation of the whips, and the complex shape, make geometric modeling and simulation of the device difficult. However, the MP-Diagram is straight forward. This implies that the function is simple, but that implementing it might take many forms. Since the primary notion of whipping eggs is nothing more than separating them and adding air, it is clear than even a fork can implement the function.

The egg beater combines MP-GEARs and MP-BLADEs differently than in the CRANK-CAN-OPENER. The former device uses gears both to drive the cutter and to drive the soupcan. In the egg beater the blades, and not the stuff, are turned. By using a single MP-GEAR, driving a separate MP-BLADE on either side of its center, results in counter-rotating MP-BLADE motions.

Egg Beater Pragmatics

The egg beater has a single design-intended use: (1) whip eggs, as shown in Fig. A.70. Egg

whipping can be seen as a separation or a combination, but both are SOP plans.

```
WHIP(EGG-BEATER,EGGS) =
          FIND(EGG-BEATER)          I-WEDGE(EGG-BEATER) +
                                    I-MECH(EGG-BEATER) +
                                    PTRANS(LOC(EGG-BEATER))
        +  D-CONT(EGG-BEATER)       GRASP(EGG-BEATER)
        +  FIND(EGGS)               PTRANS(LOC(EGGS))
        +  D-CONT(EGGS)             GRASP(EGGS)
        +  I-PREP(EGGS)             CRACK-OPEN(EGGS,BOWL)
        +  I-PREP(EGG-BEATER)       HOLD(EB-HANDLE1) +
                                    PLACE(EB-BLADES,BOWL) +
                                    GRASP(EB-HANDLE2)
        +  PROPEL(EGG-BEATER)       TURN(EB-HANDLE2,CCW)
```

device-use template

instantiation of subgoals

Figure A.70 Pragmatic representation of egg beater whipping.

A.4.8 MouseTrap

A mouse trap is illustrated in Fig. A.71:

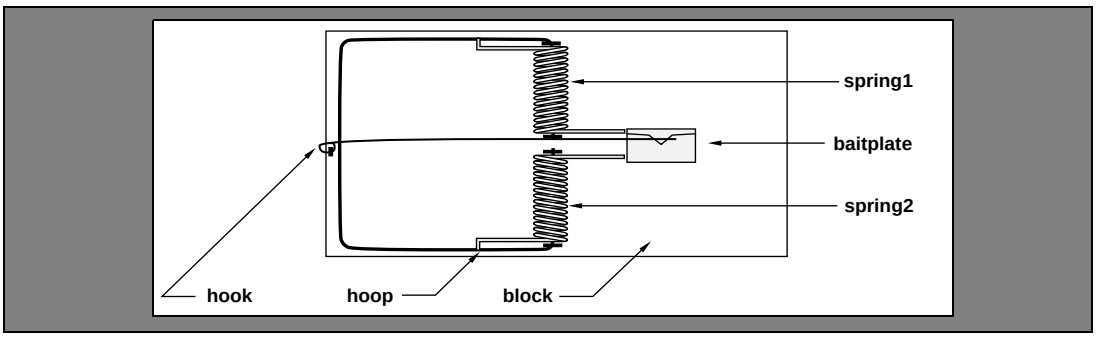


Figure A.71 Mouse trap illustration.

Mousetrap Statics

An idealized mousetrap and its static device diagram are depicted in Fig. A.72.

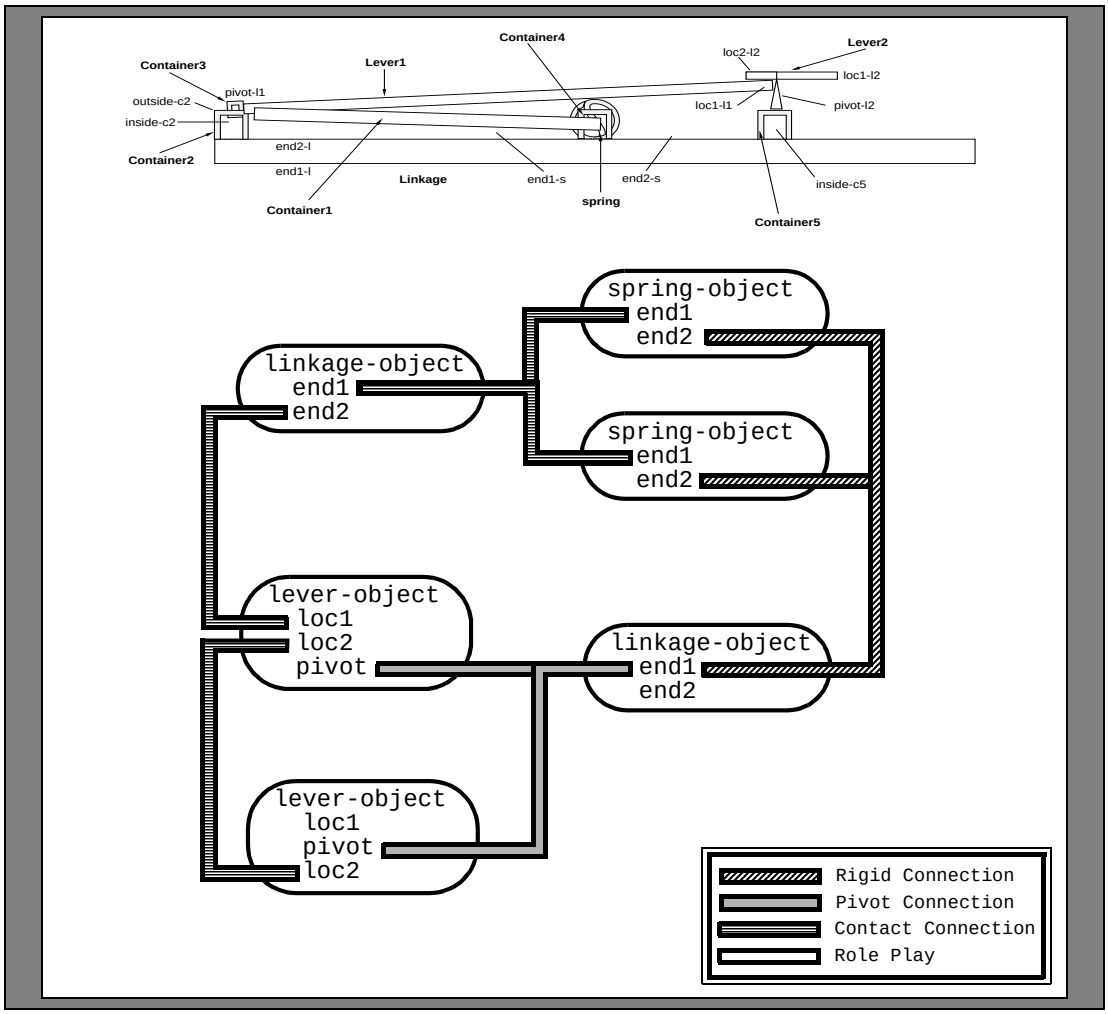


Figure A.72 Mousetrap static device diagram.

High-Level Mousetrap Dynamics

The mouse-trap has two primary functions, arming and firing, like the toy gun. These functions make use of the same primitives, and can be combined into a single MP-Diagram with

the flow of force and motion traced through the affected primitives, as shown in Fig. A.73.

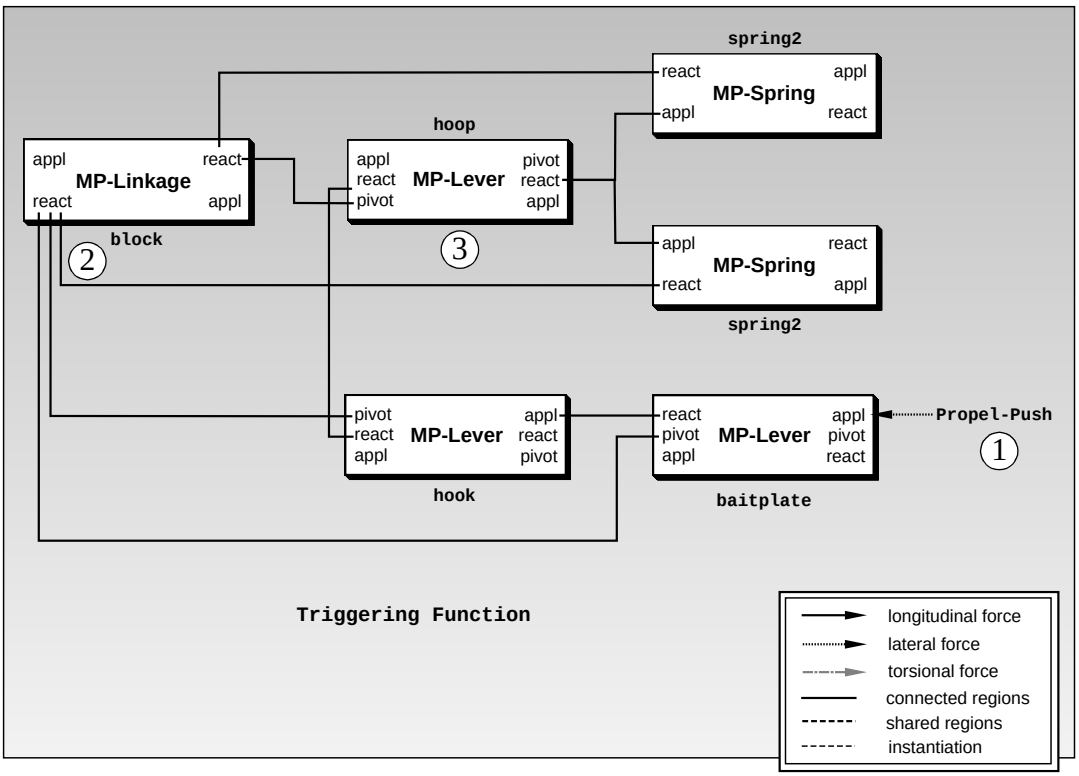


Figure A.73 MP-Diagram for mousetrap function.

Mousetrap Pragmatics

The mouse trap a single design-intended use: (1) to kill mice, which is a CROM plan. The

use is illustrated in Fig. A.74

```
TRAP-MOUSE(MOUSE-TRAP,MOUSE) =
        FIND(MOUSE-TRAP)          I-CONTN(MOUSE-TRAP) +
                                  I-MECH(MOUSE-TRAP) +
                                  I-RESIST(MOUSE-TRAP) +
                                  PTRANS(LOC(MOUSE-TRAP))
    +   D-CONT(MOUSE-TRAP)        GRASP(MT-BLOCK)
    +   FIND(BAIT)                PTRANS(LOC(BAIT))
    +   D-CONT(BAIT)              GRASP(BAIT)
    +   I-PREP(BAIT)              PLACE(BAIT,CLAW-MTBAITPLATE,ABOVE)
    +   I-PREP(MOUSE-TRAP)        ARM(MOUSE-TRAP) =
                                    HOLD(MTBLOCK) +
                                    GRASP(MTBAR) +
                                    PULL(MTBAR,MTBAITPLATE) +
                                    HOLD(MTBAR,MTBAITPLATE) +
                                    GRASP(END1-MTLOOP)
    +   PROPEL(MT-LOOP)           PUSH(MT-LOOP,CATCH-MTBAITPLATE)
```

device-use template

instantiation of subgoals

Figure A.74 Pragmatic representation of mouse trap use for trapping mice.

Glossary of Terms

A

Applied force: An object behavioral property associated with motion and force transmission. (22)

B

Behavioral delta goal: A goal which describes a state change needed to recover from a failed goal. (209)

Behavioral preconditions: BPP preconditions which are based on behavioral object state changes. (79)

Behavioral Process Primitives: A knowledge structure which represents fundamental mechanical behaviors and can be sequenced to represent object function. (3, 78)

Black box: An object whose function is known but nothing about how the function is achieved is known. (147)

Blackboard: A memory which is available to a variety of processing mechanisms. In the case of conceptual analysis, the working memory is a blackboard since it can be accessed by all the demons. (271)

Bondgraph: A representation approach which describes the ways that a device can be connected to other devices and by what behavioral interactions each connection can support. (327)

Boundary region: Describes the junction of surface regions. An edge is a boundary between two faces, between a face and an indentation, or between a face and a protuberance. (28)

C

Canonical form: A structure into which knowledge can be captured in such a way that two concepts which have the same meaning are represented with the same structure. (8)

Cartesian axis dimensions: The three translational dimensions - X, Y, and Z and their rotational equivalents - P, Q, and R. (22)

Causal keywords: Words which causally connect physical events and enable inference about intervening relationships. For example, "X until Y," or "when X Y." (279)

Clamped connection: A state in which both objects are fully (i.e. all dimensions) restrained. (61)

Complex device: A device comprised of systems, each of which can be a multiple compound device. (62)

Compound device: A device where at least two components can move relative to one another. (62)

Continuity consistency: A model which predicts continuous behavior when applied mechanics would predict continuous behavior. (79)

Cylindrical axis dimensions: The dimensions associated with cylindrical bodies, namely radius and length. In addition, the rotational equivalents for these dimensions complete the set of cylindrical degrees of freedom. (22)

D

Degree of freedom: A dimensional reference. There are six degrees of freedom in space, one each for the three translational axes and one each for the three rotational axes. (23, 330)

Demon agenda: A stack onto which demons are placed and polled against a current or working memory of concepts. (271)

Demons: Delayed procedures with test-act parts that implement rules. (271)

Device appearance: The shape and size of a device's components and their features. (21)

Device construction: The composition and connectivity of a device's components. (21)

Device dynamics: The combination of low-level and high-level behavioral interactions between objects. (77)

Device function: What a device does, with respect to perturbation and perceived results, as opposed to how the result comes about. (147)

Device pragmatics: How a device functions, how it shares similarities with other devices, and how the application of device function is affected by the environment in which it is used. (207)

Device statics: Composition, appearance, orientation and connectivity of a device's components. (21)

Device-use plan: A plan in which a device function is applied. (207)

Dynamic consistency: When a model is based on the same behavioral principles upon which applied mechanics is based. (78)

E

Elastic deformation: Changes in object size which are reversed when the applied force is removed. (98)

Elastic limit: In Applied Mechanics the elastic limit is 0.2% strain on the stress-strain curve. Beyond the elastic limit some strain becomes permanent deformation. (97)

Episodic memory: A memory of experiences and events. (271)

External nodes: Boundary nodes of a finite element. (330)

F

Face: A region representing a flat surface and associated with support. (27)

Finite element: An engineering theory which represents objects as compositions of discrete objects with linear properties. A finite element object appears like a wire frame. (44, 47)

Fixed connection: A state (identical to clamped) in which both objects are fully (i.e. all dimensions) restrained. (61)

Fixed-axle device:A device whose axle region location hasadifferent radius than the appl region location, so force magnification is produced on the device itself. (167)

Free-axle device: A device which rotates about its axle without axle rotation. Force magnification is generated by cascading free-axle devices of different radii. (167)

G

Generalized machine: A device which takes on the functional characteristics of one or more of its component functions. (193)

Generic process: The minimum definition which allows recognition of a process class. (83)

Global behavior: High-level object behavior associated with device function. (77)

Grounded object: An object which is anchored to the inertial frame and cannot move in any dimensional axis. (245)

H

High-level behavior: Behavior which is observed as the function of an object. (77)

I

In-plane: Force or Motion resulting from a process is in the same dimension as the one which enabled the process. (159)

Indentation: A region representing a concave volume and associated with containment. (27)

Individual object: A single object and its regions. An individual object is the primitive building block in the FONM representation (see object primitive). (21)

Individual View: A collection of objects and the relations between their quantities. Generally a bounding state for recognizing behavior or function. (310)

Internal energy: An object which is perturbed but unable to move must absorb, store, or internalize the force. The internal force is used to change the size and shape of the object and, when the applied force is removed, to return the object (98)

Internal force: An object behavioral property associated with energy storage. (22)

Inter-object dynamics: Interactions which are not explicitly a part of an object primitive but, rather, with a device. Usually motion of a device component or mechanism. (149)

Intra-object dynamics: Dynamic interactions which are associated with an object primitive. (148)

K

Knowledge Dependency Graph: A diagram illustrating the class dependencies of a KR approach. (9)

Knowledge primitive: Represents the lowest level of inference in the knowledge domain, and is the inferential building block of the system. (8)

L

Linkage: An elastic object, capable of transmitting mechanical force in at least one dimension and direction. (30)

Local behavior: Low-level behavior associated with device components and their interactions. (77)

Location: A knolwedge structure with two roles -- ref and dimv for representing region locations on objects. (28)

Location value: Locations on an object are described with a discrete set of values (FAR-LEFT, LEFT, NEAR-LEFT, MIDDLE, NEAR-RIGHT, RIGHT, and FAR-RIGHT). (28)

Low-level behavior: Behavior of and between components of a device that are not observed as the function of the device. (77)

M

Machine plans: Plans which are used to achieve mechanical instrumental goals, and which are associated with particular machine functions. (216)

Machine Primitive: A primitive machine function, which is represented as a schemata associated with a specific sequence of behavioral processes. (3)

Material division: Any DEFORM process which results in a new region and releases internal energy, such as a cut or tear. (105)

Material stiffness: An object property associated with the ability to transmit mechanical force. (22)

McDypar: The micro version of Dyer's demon parser (see Dyer, 1983). (271)

Mechanical advantage: Mechanical force magnification brought about as a tradeoff between applied force and the distance through which it acts. (95)

Mechanical instrumental goal: Goals which relate a desired function to the devices which can instantiate it. (212)

Mechanical understanding: The ability to recognize and comprehend the relationship between the use of an object and the manner in which the object behaves. (4)

MP Diagram: A diagram of device function composed of MP blocks and links representing interobject connectivity. (156)

N

Naive Mechanics: A description of mechanical devices which is based on knowledge shared by members of a culture through experience with objects in their everyday lives. (3)

Naive reasoner: Someone whose knowledge is based on experience and whose reasoning is based on commonsense or culturally shared principles. (3)

Nodal points: Discrete points which define the boundaries of a finite element. All behavior and states of the finite element are defined at nodal points. (330)

O

Object behavior: How and object works. The internal interactions between object components which is ignored by object function. Behavior is described most simply as a state history for any particular object property. (3)

Object deformation: In mechanics this is called strain, or the ratio between the change in length to the original length. The theory of elasticity is based on small deformations, those which do not exceed the 0.2% strain limit. (98)

Object function: What an object does when perturbed, represented with a schemata which has initiating and terminating states, and I/O regions associated with the function. (3)

Object motion: The time history which describes an object's change of position and how it is brought about. Object motion is represented with BPP-MOTION. (83)

Object primitive: An object which is commonly recognized and associated with a particular function but is not decomposed to more primitive functional objects. (43)

Object regions: General appearance of an object, described by shape, size, and location. (22)

Object restraint: An object behavioral property associated with contact between objects and motion constraint. (22)

Off-axis forcing: A force which is applied to an object which is not directed through the object CG. (85)

Offset angle: The angle between two surface regions. (59)

Offset ratio: The ratio of vertical travel to horizontal travel in inclined planes. Used as a measure of planar motion advantage. (184)

On-axis forcing: A force applied through the object's center of gravity. (85)

Orientation: An object relational characteristic that defines dimensional alignment between two objects. (58)

P

Parse Diagram: A graphical representation showing how textual entries are associated with each other with demons. (274)

Physical state: A knowledge structure which represents object material properties and behavior. (22)

Pivot: A means of modifying how force is applied to object is contact. Three pivot types are defined in FONM, a fulcrum pivot, a fixed pivot, and an offset pivot. (95)

Plastic deformation: Changes in object size and shape which remain when the applied force is removed. (98)

Predictive consistency: A model which, when given the same object and initial conditions, makes the same predictions of behavior as would applied mechanics. (79)

Process: A knowledge structure which forms the backbone for behavioral process primitives. (79)

Property history: A sequence of object states for a particular behavioral property. (78)

Protuberance: A region representing a concave volume and associated with interference. (27)

Q

Qualitative Process: A dynamic structure (Forbus) that control the change of object state and individual views. A qualitative process describes object time histories. (311)

Quant: A knowledge structure for representing property values (quantities). The quant KS has three slots - state, unit, and value. (25)

Quantity space: A one-dimensional array of values for a particular unit type. The array describes a continuum from the minimum possible value to the maximum possible value, but is represented only with the minimum and maximum known values. (25)

R

Radius ratio: The ratio of wheel, pulley, and gear radii used to determine relative motion. (166)

Reacted force: When force is applied to an object at one location, and the object is in contact with another object at a second location, then the applied force is reacted at the second location. (93)

Relational characteristic: A spatial relationship between two primitive objects, typically orientation and placement. (58)

Representational granularity: The degree of specificity of a set of representational constructs, and hence the degree of inference associated with them. (79)

Rigid: The object has no mobility but can transmit all forces. (245)

Role-play link: A link between two knowledge structures which associates the role of one with the role of the other. Whichever filler is defined for one will also be defined for the other. (357)

S

Semantic and Situation Memory: An inviolate or template memory of problem-solving situations and concepts. (236, 270)

Shared dimensions: When two object regions lie on the same dimension but have different locations they share the dimension. Two ends of a stick share the longitudinal axis. (160)

Simple device: A device where all the components are connected together and move as a unit. (62)

Situation interpretation: Recognizing and understanding goals within a problem-solving context. (4)

Spawning demons: Placing demons onto an agenda. The term spawn is associated with a demon's activation because of its context specificity and competitive nature. (271)

Static device description: The device characteristics which describe a device's construction and appearance. (21)

Static Device Diagram (SDD): An idealization of the connectivity of device components as FONM primitive objects and arcs representing the associated RESTRAIN relationship between the components. (62, 64)

Static equilibrium: When all forces in a specified dimension balance. (81)

Store: The process primitive which represents elastic energy storage in objects. (98)

Subgoal resolution: A subgoal specialization of a plan which identifies a set of goal types which satisfy the plan but is not instantiated with a particular goal. (214, 218)

Surface region: Represents an area or volume. There are three surface region types - an indentation, a face, and a protuberance, depending on whether the object is concave, flat, or convex over the volume. (27)

T

Taxonomy: In knowledge representation, a taxonomy refers to a finite set of knowledge structures which span a knowledge class. (8)

Temporally sequential: Actions, scripts, or plans which occur in order. (219)

Temporally simultaneous: Temporally Parallel. Actions, scripts, or plans which occur at the same time. (219)

Transient behavior: Behavior which occurs over short periods of time immediately following a perturbation. FONM is a steady state approach to representing device dynamics. (79)

Type-1 lever: A lever which has the pivot or fulcrum between the location where force is applied and force is reacted. (161)

Bibliography

Allen, J. (1984). Toward a general theory of action and time. *Artificial Intelligence*, 23(2):123–154.

Allen, J. F. (1985). Maintaining knowledge about temporal intervals. In *Readings in Knowledge Representation*, pages 502–522. Morgan Kaufman.

Alonso, M. and Finn, E. J. (1970). *Physics*. Addison-Wesley, Reading, MA.

Alvarado, S. J. (1989). *Understanding Editorial Text: A Computer Model of Argument Comprehension*. PhD thesis, University of California at Los Angeles.

Archer, R. R., Cook, N. H., Crandall, S. H., Dahl, C. N., Lardner, T. J., McClintock, Frank A. Rabinowicz, E., and Reichenbach, G. S. (1972). *An Introduction to the Mechanics of Solids*. McGraw-Hill, New York, 2 edition.

Arens, Y. (1986). *Cluster: An Approach to Contextual Language Understanding*. Report ucb/csd 86/293, University of California at Berkekley, Computer Science Division.

Arkes, H. and Harkness, A. (1980). Effect of making a diagnosis on subsequent recogniton of symptoms. *Journal of Experimental Psychology: Human Learning and Memory*, 6:568–575.

Bateman, J. A. and Paris, C. L. (1989). Phrasing a text in terms the user can understand. In *proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Vol. II, pages 1511-1517, Detroit, MI.

Bramwell, M. and Mostyn, D. (1984). *How Things Work*. Usborne Publishing.

Brown, D. C. and Breau, R. (1986). Types of constraints in routine design problem-solving. In Sriram, D. and Adey, R., editors, *Applications of Artificial Intelligence in Engineering Problems*, volume 1, pages 383–390, Southampton, United Kingdom. 1st International Conference, Springer Verlag.

Bullock, M., Gelman, R., and Baillargeon, R. (1982). The development of causal reasoning. In Friedman, W. J., editor, *The Developmental Psychology of Time*, pages 209–254. Academic Press.

Cammarata, S. J. and Melkanoff, M. A. (1988). An information dictionary for managing cad/cam databases. *Database Programming and Design*, 1(3):26–35.

Caney, S. (1984). *Invention Book*. Workman Publishing.

Carbonell, J. (1982). Metaphors and physical envisionment. In Lehnert, W. and Ringle, M., editors, *Strategies for Natural Language Processing*, chapter 15, pages 415–434. Lawrence Erlbaum Associates.

Carbonell, J. (1983). Learning by analogy: Formulating and generalizing plans from past experience. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning: An Artificial Intelligence Approach*. Tioga, Palo Alto, CA.

Carbonell, J. (1986). Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning: An Artificial Intelligence Approach*, volume II. Morgan Kauffman, San Mateo, CA.

Chittaro, L. and Tasso, C. and Toppano, E. (1993). Putting Functional Knowledge on Firmer Ground. In *Proceedings of the 1993 AAAI workshop on Reasoning about Function*. July, 1993.

Clough, R.W. (1960): The Finite Element Method in Plane Stress Analysis. In *Proceedings of the Second Conference on Electronic Computation,* ASCE, Pittsburgh, Pa, Sept 8-9 1960.

Coyne, R. D., Rosenman, M. A., Radford, A. D., and Gero, J. S. (1987). A logic model of creativity in knowledge-based cad. Unpublished research.

Cullingford, R. E. (1978). *Script Application: Computer Understanding of Newspaper Stories*. PhD thesis, Yale University, New Haven, Connecticut. Department of Computer Science, Research Report 116.

DeBono, E. (1980). *Children Solve Problems*. Penguin.

Dejong, G. and Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145–176.

DeKleer, J. (1984). How circuits work. *Artificial Intelligence*, 24:205–280.

DeKleer, J. and Seeley Brown, J. (1983). A qualitative physics based on confluences. *Artificial Intelligence*, 24:7–84.

Desai, C. S. and Abel, J. F. (1972). *Introduction to the Finite Element Method: A Numerical Method for Engineering Analysis*. Van Nostrand Reinhold, New York.

Dietterich, T. (1986). Learning at the knowledge level. *Machine Learning*, 1(3):287–315.

diSessa, A. A. (1983). Phenomenology and the evolution of intuition. In Gentner, D. and Stevens, A., editors, *Mental Models*, chapter 2, pages 15–34. Lawrence Erlbaum Associates, Hillsdale, N.J.

Doyle, R. (1989a). *Hypothesizing Device Mechanisms: Opening Up The Black Box*. PhD thesis, Massachussetts Institute of Technology. Report TR-1047.

Doyle, R. (1989b). Reasoning about Hidden Mechanisms. In *Proceedings of the Eleventh International Joint Coference on Artificial Intelligence*, Vol. 2, pages 1343-1349. Detroit, MI.

Duncker, K. (1945). On problem solving. *Psychological Monographs*, 58(5).

Dyer, M. G. (1983). *In-Depth Understanding*. MIT Press, Cambridge, Massachusetts.

Dyer, M. G. and Flowers, M. (1984). Automating design invention. In *Proceedings of Autofact 6*, Anaheim, CA.

Dyer, M. G., Flowers, M., and Hodges, J. (1985). Mechanical invention in edison. *IEEE Proceedings*.

Dyer, M. G., Flowers, M., and Hodges, J. (1986). Edison: an engineering design invention system operating naively. In D. Sriram and R. Adey (eds), *Applications of Artificial Intelligence in Engineering Problems*, Springer-Verlag, pp 327–361.

Dyer, M. G., Flowers, M., and Hodges, J. (1987a). Naive mechanics comprehension and invention in edison. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, volume 2, pages 696–699. Morgan-Kaufman.

Dyer, M. G., Flowers, M., and Hodges, J. (1987b). Computer comprehension of mechanical device descriptions. Technical report UCLA-AI-87-7, University of California at Los Angeles, Artificial Intelligence Laboratory. 1987.

Falkenhainer, B., Forbus, K., and Gentner, D. (1986). The structure-mapping engine. *Science*, pages 272–277.

Faltings, B. (1987). Qualitative kinematics in mechanisms. In *Tenth International Joint Conference on Artificial Intelligence*, pages 436–442. American Association for Artificial Intelligence (AAAI), Morgan Kaufman.

Forbus, K. (1984). Qualitative process theory. *Artificial Intelligence*, 24:85–168.

Forbus, K. D. (1985). Qualitative process theory. In Hobbs, J. R. and Moore, R. C., editors, *Formal Theories of the Commonsense World*, chapter 5, pages 185–226. Ablex Publishing Corporation.

Gentner, D. (1983a). Mental models. In Gentner, D. and Stevens, A., editors, *Mental Models*. Lawrence Erlbaum Associates, Hillsdale, N.J.

Gentner, D. (1983b). Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, 7(2).

Goel, A. K. (1989). *Integration of Case-Based and Model-Based Reasoning for Adaptive Design Problem Solving*. PhD thesis, Ohio State University.

Granacki, J. J. and Parker, A. C. and Arens, Y. (1989). Understanding System Specifications Written in Natural Language. *Eleventh International Joint Conference on Artificial Intelligence*, pages 688–691. American Association for Artificial Intelligence (AAAI), Morgan Kaufman.

Granacki, J. J. and Parker, A. C. (1986). A natural language interface for specifying digital systems. In Sriram, D. and Adey, R., editors, *Applications of Artificial Intelligence in Engineering Problems*, volume 1, pages 215–226, Southampton, United Kingdom. 1st International Conference, Springer Verlag.

Granada Publishing, L. (1972). *How Things Work 1*. Granada Publishing Limited, London, UK.

Hammond, K. J. (1989). *Case-Based Planning: Viewing Planning as a Memory Task*. Perspectives in Artificial Intelligence. Academic Press, New York, NY.

Hayes, P. J. (1979). The naive physics manifesto. In *Expert Systems in the Microelectronic Age*, pages 242–270. Edinbugh University Press, Edinburgh, UK.

Hayes, P. J. (1985). The second naive physics manifesto. In *Readings in Knowledge Representation*, pages 467–486. Morgan Kaufmann.

Herskovits, A. (1986). *Language and Spatial Cognition*. Cambridge University Press, Great Brittain.

Hodges, J. (1989). Device representation for modeling improvisation in mechanical use situations. In *Proceedings of the 11th Annual Meeting of the Cognitive Science Society*, pages 643–650, Ann Arbor, Michigan.

Hodges, J. (1992). Naive mechanics: A computational model of device use and function in design improvisation. *IEEE Expert*, 7(1):14–27. cover article.

Hodges, J., Flowers, M., and Dyer, M. G. (1987). Knowledge representation for design creativity. In Liu, C. R., Requicha, A., and Chandrasekar, S., editors, *ASME Winter Annual Meeting*, volume PED-Vol. 25, pages 81–93. American Society of Mechanical Engineers, Boston, MA.

Hodges, J., Flowers, M., and Dyer, M. G. (1992). Knowledge representation for design improvisation. In Sriram, D. and Tong, C., editors, *Artificial Intelligent in Engineering Design*, volume I, chapter 6, pages 193–217. Academic Press.

Holyoak, K. and Glass, A. (86). *Cognition*. Random House, New York.

Johnson-Laird, P. (1983). *Mental Models*. Cambridge Unversity Press, New York.

Kahneman, D., Slovic, P., and Tversky, A. (1983). *Judgement Under Uncertainty- Heuristics and Biases*. Cambridge University Press, New York, NY.

Kannapan, S. and Marschek, K. M. (1989). An algebraic and predicate logic approach to representation and reasoning in machine design. Mechanical Systems and Design TR 212, University of Texas, Austin, Austin, Texas 78712-1063.

Kannapan, S. M. (1993). Functional Evaluation of Patentability. In *Proceedings of the 1993 AAAI workshop on Reasoning about Function*. July, 1993.

Kempton, W. (1986). Two theories of home heat control. *Cognitive Science*, 10(1):75–90.

Keuneke, A. M. (1989). *Machine Understanding of Devices Causal Explanation of Diagnostic Conclusions*. PhD thesis, Ohio State University.

Kintsch, W. (1986). Learning from text. *Cognition and Instruction*, 3:87–108.

Kintsch, W. (1987). Knowledge organization and text organization. *Cognition and Instruction*, 4(2):91–115.

Kintsch, W. (1988). The role of knowledge in discourse comprehension: A construction-integration model. *Psychological Review*, 95(2):163–182.

Kintsch, W., Kozminsky, E., Streby, W., McKoon, G., and Keenen, J. (1975). Comprehension and recall as a function of content variables. *Journal of Verbal Learning and Verbal Behavior*, 14:196–214.

Kintsch, W. and Van Dijk, T. (1978). Toward a model of text comprehension and production. *Psychological Review*, 85(5):363–394.

Klahr, D. and Wallace, J. (1976). *Cognitive Development: An Information Processing View*. Lawrence Erlbaum, Hillsdale, NJ.

Kolodner, J. A. (1984). *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model*. Lawrence Erlbaum Associates, Hillsdale, NJ.

Kuipers, B. (1986). Qualitative simulation. *Artificial Intelligence*, 29:289–388.

Lakoff, G. and Johnson, M. (1980). *Metaphors We Live By*. University of Chicago Press, Chicago, IL.

Lange, T., Hodges, J., Fuenmayor, M., and Belyaev, L. (1989). Descartes: Development environment for simulating connectionist architectures. In *Proceedings of the Eleventh Annual Meeting of the Cognitive Science Society*, pages 698–705.

Lebowitz, M. (1980). *Generalization and Memory in an Integrated Understanding System*. PhD thesis, Yale University. 186.

Lebowitz, M. (1985). RESEARCHER: An experimental inelligent information system. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 858-862. Los Angeles.

Lehnert, W. G. (1978). *The Process of Question Answering*. Lawrence Erlbaum Associates.

Lehnert, W. G. (1982). *Strategies of Natural Language Processing*, chapter 14, pages 375–414. Lawrence Erlbaum Associates.

Lenat, D. B. (1976). *AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search*. PhD thesis, Stanford University. STAN-CS-76-570.

Lenat, D. B. (1983). Eurisko: A program that learns new heuristics and domain concepts. *Artificial Intelligence*, 21(1,2):61–98.

Lenat, D. B. and Seeley Brown, J. (1984). Why am and eurisko appear to work. *Artificial Intelligence*, 23(3).

Lind, M. (1990). Representing Goals and Functions of Complex Systems: An Introduction to Multilevel Flow Modelling. Technical note 90-D-381, Institute of Automatic Control Systems, Technical University of Denmark. November.

Lind, M. and Larsen, M. and Osman, A. (1992). Applications of Multiple Flow Modeling. In *Proceedings of the ANP-92 International Conference on Design and Safety of Advanced Nuclear Power Plants.* Tokyo, Japan, Oct. 25-29.

Macaulay, D. (1988). *The Way Things Work.* Houghton Mifflin Company.

McCloskey, M. (1983). *Naive Theories of Motion*, chapter 13, pages 299–323. Lawrence Erlbaum, Hillsdale, NJ.

Metz, K. E. (1985). The development of children's problem solving in a gears task: A problem space perspective. *Cognitive Science*, 9(4):431–472.

Minsky, M. (1975). A framework for representing knowledge. In Winston, P., editor, *The Psychology of Computer Vision*, pages 211–277. McGraw-Hill, New York.

Minsky, M. (1981). A framework for representing knowledge. In Haugeland, J., editor, *Mind Design*, pages 95–128. MIT Press, Cambridge, MA.

Mostow, J. (1985). Toward better models of the design process. *AI Magazine*, 6(1):44–57.

Murphy, G. and Medin, D. (1985). The role of theories in conceptual coherence. *Psychological Review*, 92(3):289–314.

Narasimhan, S. and Navinchandra, D. and Sycara K. (1993). An Explicit Qualitative Representation of Function and Behavior for Conceptual Mechanical Design. In *Proceedings of the 1993 AAAI workshop on Reasoning about Function*. July, 1993.

Navinchandra, D. (1989). *Exploration and Innovation in Design: Towards a Computational Model*. Springer-Verlag, New York.

Navinchandra, D. and Marks, D. H. (1986). Intelligent use of constraints for activity scheduling. In Sriram, D. and Adey, R., editors, *Applications of Artificial Intelligence in Engineering Problems*, volume 1, pages 369–379, Southampton, United Kingdom. 1st International Conference, Springer Verlag.

Navinchandra, D., Sriram, D., and Kedar-Cabelli, S. (1987). The role of analogy in engineering problem solving. In *Second International Conference on Applications of Artificial Intelligence in Engineering*.

Navinchandra, D. and Sycara, Katia, P. (1989). A process model of experience-based design. In *The 11th Annual Conference of the Cognitive Science Society*, volume 1, pages 283–290, Hillsdale, NJ. Cognitive Science Society, Lawrence Erlbaum.

NAVY, U.S. (1971). *Basic Machines and How They Work*. Dover Publications, Inc.

Nielsen, P. (1989). The role of abstraction in place vocabularies. In *Proceedings of the Eleventh Annual Meeting of the Cognitive Science Society*, pages 267–274, Ann Arbor, Michigan.

Norman, D. A. (1983). Some observations on mental models. In Gentner, D. and Stevens, A., editors, *Mental Models*, chapter 1, pages 7–14. Lawrence Erlbaum Associates, Hillsdale, N.J.

Oswalt, W. H. (1976). *An Anthropological Analysis of Food-Getting Technology*. John Wiley & Sons.

Paris, C. (1987). Combining discourse strategies to generate descriptions to users along a naive/expert spectrum. In *proceedings of the 1987 international joint conference on artificial intelligence*, pages 626-632, Milan, Italy.

Pazzani, M. (1988). *Learning Causal Relationships: An Integration of Empirical and Explanation-Based Learning Methods*. PhD thesis, University of California at Los Angeles. UCLA-AI-88-10.

Pittges, J. and Eiselt, K and Goel, A. and Gomez, A. and Mahesh, K. and Peterson, J. (1993). Representation and Use of Function in Natural Language Understanding. In *Proceedings of the 1993 AAAI workshop on Reasoning about Function*. July, 1993.

Pylyshyn, Z. (1985). *Computation and Cognition*. MIT Press, second edition.

Qian, L. and Gero, J. (1993). Design Part Classification by Goal Achievement. In *Proceedings of the 1993 AAAI workshop on Reasoning about Function*. July, 1993.

Rajamooney, S., DeJong, G., and Faltings, B. (1985). Towards a model of conceptual knowledge acquisition through directed experimentation. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*. AAAI, Morgan Kauffman.

Rieger, C. (1976). An organization of knowledge for problem solving and language comprehension. *Artificial Intelligence*, 7(2):89–127.

Rieger, C. and Grinberg, M. (1977). The declarative representation and procedural simulation of causality in physics mechanisms. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 250–255. AAAI.

Riesbeck, C. and Schank, R. (1989). *Inside Case-Based Reasoning*. Lawrence Erlbaum, Hillsdale, NJ.

Rivlin, E. and Rosenfeld, A. and Perlis, D. (1993). Recognition of Object Functionality in Goal-Directed Robotics. In *Proceedings of the 1993 AAAI workshop on Reasoning about Function*. July, 1993.

Rosch, E. (1978). *Principles of Categorization*. Lawrence Erlbaum Associates, Hillsdale, NJ.

Rubinstein, M. F. (1986). *Tools for Thinking and Problem Solving*. Prentice-Hall.

Sacerdoti, E. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135.

Schank, R. (1973). Identification of the conceptualizations underlying natural language. In Schank, R. and Colby, K., editors, *Computer Models of Thought and Language*. W.H. Freeman, San Francisco, CA.

Schank, R. (1975). *Conceptual Information Processing*. American Elsevier, New York.

Schank, R. (1982). *Dynamic Memory: A Theory of Learning in Computers and People*. Cambridge University Press, New York.

Schank, R. (1986). *Explanation Patterns: Understanding Mechanically and Creatively*. Lawrence Erlbaum, Hillsdale, NJ.

Schank, R. and Abelson, R. (1977). *Scripts, Plans, Goals, and Understanding*. The Artificial Intelligence Series. Lawrence Erlbaum, Hillsdale, NJ.

Schank, R. C. and Birnbaum, L. (1980). Memory, meaning, and syntax. Research Report 189, Department of Computer Science, Yale University.

Schmalhofer, F. and Glavanov, D. (1986). Three components of human understanding a programmer's manual: Verbatim, propositional and situational representations. *Journal of Memory and Language*, 25:279–294.

Sembugamoorthy, V. and Chandrasekaran, B. (1986). Functional representation of devices and compilation of diagnostic problem-solving systems. In Kolodner, J. and Riesbeck, C., editors, *Experience, Memory, and Reasoning*, chapter 4, pages 47–73. Lawrence Erlbaum Associates.

Skorstad, G. and Forbus, K. (1989). Qualitative and quantitative reasoning about thermodynamics. In *Proceedings of the Eleventh Annual Meeting of the Cognitive Science Society*, pages 892–899, Ann Arbor, Michigan.

Sriram, D. (1987). *Knowledge-Based Approaches for Structural Design*. Computational Mechanics, Southampton, UK.

Sriram, D. and Maher, M. (1986). The representation and use of constraints in structural design. In Sriram, D. and Adey, R., editors, *Applications of Artificial Intelligence in Engineering Problems*, volume 1, pages 355–368, Southampton, United Kingdom. 1st International Conference, Springer Verlag.

Stahovich, T. and Davis, R. and Shrobe, H. (1993). An Ontology of Mechanical Devices. In *Proceedings of the 1990 AAAI workshop on Qualitative Vision*. July, 1993.

Stark, L. and Bowyer, K. (1990). Achieving generalized object recognition through reasoning about association of function to structure. In *Proceedings of the 1993 AAAI workshop on Reasoning about Function*, pp 137-141, 1990.

Sticklen, J. (1987). *MDX2: An Integrated Medical Diagnostic System*. PhD thesis, Ohio State University, Columbus, OH. Department of Computer and Information Science.

Swinney, D. (1979). Lexical access during sentence comprehension: (re)consideration of context effects. *Journal of Verbal Learning and Verbal Behavior*, 18:645–660.

Swinney, D. (1981). The process of language comprehension; an approach to examining issues in cognition and language. *Cognition*, 10:307–312.

Sycara, K. (1987). *Resolving Adversial Conflicts: An Approach to Integrating Case-Based and Analytic Methods*. PhD thesis, Georgia Institute of Technology, School of Information andComputer Science.

Thagard, P. and Holyoak, K. (1989). Why indexing is the wrong way to think about analog retrieval? In *Proceedings of the Second DARPA Case-Based Reasoning Workshop*, pages 31–35.

Thomson, D. and Tulving, E. (1970). Associative encoding and retrieval: Weak and strong cues. *Journal of Experimental Psychology*, 86:255–262.

Thorndyke, P. (1977). Cognitive structures in comprehension and memory of narrative discourse. *Journal of Cognitive Psychology*, 9:77–110.

Tulving, E. (1972). Episodic and semantic memory. In Tulving, E. and Donaldson, W., editors, *Organization and Memory*. Academic Press, New York.

Tulving, E. (1983). *Elements of Episodic Memory*. Oxford Psychology Series. Oxford University Press, Oxford, UK.

Turner, S. and Reeves, J. (1987). The rhapsody manual. Technical report, UCLA Artificial Intelligence Laboratory, Los Angeles, CA. UCLA-AI-87-3.

Tversky, A. and Kahneman, D. (1983). *Causal Schemas in Judgements Under Uncertainty*, pages 117–128. Cambridge University Press, New York, NY.

Ulrich, K. (1986). The use of objects in problem-solving. Personal Communication.

Ulrich, K. and Seering, W. (1987). Conceptual design: Synthesis of systems of components. In *Intelligent and Integrated Manufacturing Analysis and Synthesis*, volume PED Vol. 25, pages 57–66. American Society of Mechanical Engineers (ASME), ASME.

Van Dijk, T. and Kintsch, W. (1983). *Strategies of Discourse Comprehension*. Academic Press, New York.

Wasserman, K. and Lebowitz, M. (1983). Representing complex physical objects. *Cognition and Brain Theory*, 6(3):259–285.

Waterman, D. and Hayes-Roth, F. (1983). An investigation of tools for building expert systems. In Hayes-Roth, Waterman, and Lenat, editors, *Building Expert Systems*. Addison-Wesley.

Weiss, H. (1983). *Machines and How They Work*. Thomas Y. Crowell.

Wickelgren, W. A. (1974). *How to Solve Problems: Elements of a Theory f Problems and Problem Solving*. W.H. Freeman and Company.

Wilensky, R. (1983). *Planning and Understanding*. Addison-Wesley, Reading, MA.

Wilensky, R. (1987). Some problems and proposals for knowledge representation. Technical Report UCB/CSD 87/351, Computer Science Division, University of California, Berkeley, CA.

Wilks, Y. (1975). An intelligent analyzer and understander in english. *Communications of the Association of Computing Machinery*, 18(5):264–274.

Winograd, T. (1972). *Understanding Natural Language*. Academic Press, New York.