Contents

Chapter 1

Introduction

1.1 Introduction

"The time has come," the Walrus said, "To talk of many things"
Lewis Carroll in *Through the Looking Glass*

The goal of this research is to develop methods for representing and accessing knowledge to support multiple tasks. The knowledge representation research attacks three problems of frame-based representation languages: representing quantified assertions, representing both definitional and nondefinitional assertions, and representing information contextually. The knowledge access research addresses three problems as well: how to provide a content addressable knowledge base, how to provide a virtual knowledge base, and how to access *viewpoints* of concepts.

1.2 Motivation

Consider the following activities that a human expert, such as a professor of botany, might perform on a routine day:

- answering students' questions in class,
- writing an explanation of photosynthesis for a textbook,
- diagnosing and treating a wilting plant in the greenhouse, and
- learning about new discoveries from a journal article.

Human beings have great flexibility in applying their knowledge of a domain to a variety of tasks. For each task they perform, only a small portion of their knowledge is relevant, yet they are able to identify that portion and distinguish it from the irrelevant knowledge. This activity is referred to here as *knowledge access*. Designing computational methods for knowledge access is crucial to developing an artificially intelligent agent that uses a corpus of domain knowledge to perform multiple tasks.

Knowledge access is rich with interesting problems. For example, consider the problems a botanist faces when writing the following description of cell nuclei:

> The nucleus is typically the largest structure in the cytoplasm of a eukaryotic cell. The nucleus performs two essential functions. First, it controls the ongoing activities of the cell. It does this by determining which protein molecules are produced by the cell and when. Second, it stores the cell's genetic information, passing it on to the daughter cells in the course of cell division.[1]

This passage illustrates several problems of knowledge access. First, the passage contains only a small fraction of everything the author knows about cell nuclei. Otherwise, the text would overwhelm the reader. Yet the author was able to determine which facts to include and which to exclude so that the material would be coherent, rather than a random sampling of information.

Second, the passage contains knowledge that is *content addressable*. For example, the concept "the cytoplasm of a eukaryotic cell," like many concepts in botany (or any other domain), has no "official" technical name. The author was able to access the concept purely via its contents; partial knowledge of the concept (*e.g.*, that it is a kind of cytoplasm and that it is part of a eukaryotic cell) is activated, resulting in the rest of the knowledge about it becoming accessible.

Third, the passage contains knowledge of potentially *ad hoc* concepts, such as "the cytoplasm of a eukaryotic cell." Even if that concept were not stable in the author's mind, the author still accessed the concept by creating it *ad hoc* using implicit knowledge of the concept distributed among other stable concepts, such as "cytoplasm" and "eukaryotic cell."

Finally, there arises the question of how the author's knowledge is encoded so as to support solutions to the above problems.

This research addresses the above problems of knowledge access in the context of computer systems performing knowledge-based tasks. The specific goals are threefold. The first goal is to make it possible to represent in a formal language the kind of domain knowledge that supports a variety of tasks (called *multifunctional knowledge*) in such a way that general (task independent) methods can access it. The second goal of the research is to develop methods to enable computer systems to access concepts by their contents, regardless of whether the concepts exist explicitly or implicitly in the knowledge base. The third goal of the research is to develop methods for accessing coherent portions of knowledge about a given concept from a large knowledge base. The next three sections discuss these goals and the approach taken to each.

[1] Adapted from *Biology of Plants* [68], page 36.

1.3 Multifunctional Knowledge Representation

Most present-day knowledge-based systems rely on representations of domain knowledge that were designed to support the single task the system performs. Such task specific knowledge bases suffer from two limitations. First, they include only the knowledge needed to perform the task for which they were engineered. For example, the IF-THEN production rule bases that expert systems use typically lack the support knowledge that underlies the rules. As a result, expert systems that rely on them cannot adapt to unanticipated circumstances because they lack knowledge of "first principles" from which to reason. In addition, the rules are usually unsuitable for explanation and teaching, as the Guidon project demonstrated [18].

The second limitation of task specific knowledge bases is that the form of their knowledge is tailored for a particular application. For example, many advisory and tutoring systems rely on knowledge in the form of "canned text," multisentence text passages hand-crafted for specific contexts. Although a special-purpose form enables more efficient performance of one task, it usually precludes the knowledge from being applied to other tasks within the same domain. For example, while knowledge of plant diseases in the form of canned text can be used to provide instruction, it cannot be used for automated diagnosis.

More flexibility is achieved by building a comprehensive, fine grained representation of domain knowledge, *i.e.*, one that provides broad coverage of the domain, represented as atomic facts that can be combined and used in multiple ways. For example, consider a knowledge base containing the following facts, represented so that an application program can examine and manipulate them.

1. Photosynthesis requires light, carbon dioxide, and water.

2. Photosynthesis produces glucose and oxygen.

3. Respiration requires glucose and oxygen.

4. Respiration produces carbon dioxide and water.

A system could use facts (1) and (4) to reason about how carbon dioxide is involved in plant physiological processes. Similarly, reasoning about the role of oxygen would involve facts (2) and (3). The first three facts would be relevant to reasoning about how light deprivation affects respiration. Finally, a system would need all four facts to reason about how photosynthesis and respiration are complementary processes.

Such comprehensive, fine grained representations of knowledge are called *multifunctional knowledge bases.* Because multifunctional knowledge bases contain knowledge to support a variety of tasks, they are large and costly to build. A hypothesis underlying this work is that it is more cost-efficient in the long run to construct a single multifunctional knowledge base for a domain than to construct a separate knowledge base dedicated to each task to be performed in that domain.

In the last few years, interest in multifunctional knowledge bases has grown rapidly within the artificial intelligence community. In 1988, the American Association for Artificial Intelligence sponsored a workshop on the topic. The CYC project [40] has dedicated the last seven years to developing the largest multifunctional knowledge base in existence. Brachman lists large knowledge bases as one of the most important areas of future knowledge representation research [10].

Multifunctional knowledge bases can be constructed using a variety of representational languages, including predicate logic, Prolog, and frame-based languages. This work assumes a frame-based language, although many of the ideas presented here apply to other representational paradigms as well. There are four reasons that a frame-based language was chosen over other representational paradigms. First, several common types of inference are very efficient in frame-based languages. These include inheritance of features from one class to another, retrieving the value of an attribute for an entity, and retrieving all known facts about an entity. Second, using a frame-based language yields a modular knowledge base, one that is convenient for browsing and editing. Modularity is especially important for a large, multifunctional knowledge base. Third, it is easy to describe the properties of relations in a frame-based language. Fourth, it is possible to describe *intensions* of concepts in a frame-based language (discussed in Chapter 2).

An important aspect of this work is its broad applicability. Any application program using a frame-based knowledge base can use the knowledge access methods presented here, regardless of the task being performed or the domain.

Although domain independent, this work is undertaken in the context of the Botany Knowledge Base project [67]. The Botany Knowledge Base is a multifunctional representation of botany, with emphasis on plant anatomy and physiology. It currently contains over 2,600 concepts and over 28,000 facts. The domain of botany was chosen because it is a non-formal domain, yet it is relatively self-contained. Like many domains, botany is concerned primarily with physical objects (anatomy) and physical processes (physiology).

While developing access methods for large multifunctional knowledge bases such as the Botany Knowledge Base, several representational problems were encountered. The rest of this section discusses these problems.

1.3.1 Representing Necessity, Sufficiency, and Likelihood

In most frame-based languages, frames represent categories, and slots and their values represent features of members of those categories. In some languages, such as KL-ONE and other terminological languages, features represent necessary and sufficient criteria for category membership, as with the features $color = Black$ and $isa = Telephone$ for the category *Black-Telephone* [86]. In other languages, features represent not defining properties, but defaults, as with the feature $color = Gray$ for *Elephant* [9]. Nonetheless, these languages make no provision for representing the likelihood of the default (*e.g.*, how likely it is that a particular elephant is gray). Still other languages seem to allow users to impose their own semantics for features [50].

The problem with these approaches is not the particular interpretation they assume, but that they all assume a single interpretation for every $\langle frame\ slot\ value \rangle$ triple in the knowledge base. A comprehensive corpus of domain knowledge consists of features of all kinds. A more flexible representation language is needed that allows the knowledge engineer to

- distinguish features that are definitional
  from those that merely happen to occur,
- distinguish between necessity and sufficiency,
- distinguish between defeasible and absolute features, and
- specify different degrees of defeasibility (*e.g.*, likelihood).

A goal of this research is to develop constructs for representing this kind of information about features. These constructs must allow information about a feature (its necessity, sufficiency, likelihood, etc.) to be represented in the same context as that feature, as opposed to requiring the reification of features as frames. In addition, these constructs must be convenient and efficient to use.

1.3.2 Representing Quantified Assertions

Just as domain knowledge includes features with different necessity, sufficiency, and likelihood, domain knowledge also includes assertions with different *quantificational patterns*:

- *Every* plant has as a part *some* stem.
- Plant01 has as a part *some* stem.
- *Every* plant has color green.
- Plant01 has color green.

Most existing frame-based representation languages do not allow quantified assertions to be *conveniently* represented (*i.e.*, represented with a single $\langle$*frame slot value*$\rangle$ triple). Ground propositions are represented with a single triple, but quantified assertions require either additional representational constructs (such as a rule or constraint language) or multiple triples. This requirement makes knowledge representation much more tedious.

A goal of this work is to allow assertions having commonly occurring patterns of quantification to be represented with a minimum of notation, while maintaining rigorous and explicit semantics. In other words, the goal is to represent quantified assertions in the same way as unquantified assertions, as single $\langle$*frame slot value*$\rangle$ triples. For example, the above four assertions would all be represented in the same syntactic form, even though each has a different quantificational pattern:

- $\langle$*Plant has-part Stem*$\rangle$
- $\langle$*Plant01 has-part Stem*$\rangle$
- $\langle$*Plant color Green*$\rangle$
- $\langle$*Plant01 color Green*$\rangle$

The correct interpretation could be determined automatically as needed.

1.3.3 Representing Contextual Information

Consider representing the statement "The cells of the root system contain no chlorophyll." Traditional frame-based representation languages provide two unsatisfactory alternatives. The first technique is to associate an "if-needed" rule with the *amount-chlorophyll* slot on the *Cell* frame that states, in effect, "If a cell is part of a root system, then it has no chlorophyll." This approach requires that the representation language include a formalism for representing and reasoning about such rules. It also has the disadvantage that it provides no direct access path from the frame *Root-System* to the knowledge about cells that are part of root systems unless a special mechanism installs pointers from frames to the rules in whose antecedents they appear.

The second technique is to create a frame for the concept "Cell of a root system," then fill in the appropriate slot values on that frame, as in

Cell-of-A-Root-System

---

generalizations: Cell
part-of: Root-System
amount-chlorophyll: Zero

If little or no additional knowledge differentiates the concept *Cell-of-A-Root-System* from the concept *Cell*, then this approach requires an inordinate amount of effort.

Another disadvantage of this approach is that it results in a proliferation of frames corresponding to concepts that are important only in very limited contexts (such as "water contained in a guard-cell that is collapsing" and "water pore in the membrane of the epidermis of a root"). Ideally, every frame in the knowledge base would correspond to a stable concept in the mind of the domain expert, that is, a concept one would expect to find in the index of a comprehensive text on the domain. Such a knowledge base is easier to navigate, for both knowledge engineers and application programs.

An alternative technique for representing "The cells of the root system contain no chlorophyll" that does not necessitate creating a frame for "Cell of a root system" is to represent the information *contextually.* That is, the triple ⟨Cell amount-chlorophyll Zero⟩ is represented in the context of the triple ⟨Root-System has-part Cell⟩, signifying that cells *that are part of a root system* have no chlorophyll. One goal of this work is to allow convenient representation of contextual information within the ⟨*frame slot value*⟩ paradigm (*i.e.*, without resorting to a rule or constraint language).

1.3.4 The Approach

The development of the knowledge representation language used in this research (called KM), took advantage of existing technology as much as possible. The starting point for KM was the Theo system developed by Tom Mitchell at Carnegie-Mellon University [50]. To allow the representation of both contextual information and necessity, sufficiency, and likelihood information, KM was extended to include *annotations*: each value of a frame-slot can be annotated with additional filled slots that either

- are relevant to the value being annotated only within the current context, as in

  Root-System
  ———————
  has-part: Cell
  amount-chlorophyll: Zero

- or provide information about the entire ⟨*frame slot value*⟩ triple, as in

  Plant
  ———
  color: Green
  likelihood: High

A second feature that distinguishes KM from Theo is a precise semantics. The semantics is expressed by a *semantic mapping* from ⟨*frame slot value*⟩ triples in the knowledge base to formulae in probabilistic logic. This mapping covers annotations as well. To allow relations carrying different quantificational patterns to be represented by simple ⟨*frame slot value*⟩ triples, the semantic mapping is not uniform for every triple in the knowledge base, but varies according to the slot appearing in the triple. Furthermore, slots are overloaded in the sense that the quantificational pattern a slot implies varies with the frame on which it appears and the value(s) it has. The knowledge enterer need not explicitly indicate the quantificational pattern of every triple in the knowledge base because this can be determined automatically.

The design of KM involved, to some degree, all members of the Botany Knowledge Base project [67]. Erik Eilerts was largely responsible for the implementation of KM. The contribution of this research is, first, providing a theoretical grounding for our design decisions, and second, specifying the semantics of KM's representational constructs. Chapter 2 gives the specifics of the KM representation language and its semantics.

1.4 Knowledge-Base Access

The term *knowledge-base access* is used here to mean identifying a portion of a knowledge base that is relevant to a particular task. For a frame-based knowledge base, a "portion" is a group of one or more $\langle$*frame slot value*$\rangle$ triples. Typical examples of access methods for frame-based representations include

- single frame-slot access, in which the user, either a human or an application program, specifies a frame and a slot occurring on that frame, and the access method returns the value(s) of the slot,
- entire frame access, in which the user specifies a frame, and the access method returns the values of all the slots occurring on that frame, and
- task specific access methods, such as the content-determination operators of an explanation-generation or question-answering system.

This research addresses two issues regarding knowledge-base access: (1) allowing users to access frames by their contents (as well as by name), regardless of whether they exist explicitly or implicitly in the knowledge base, and (2) accessing coherent portions of knowledge about a particular concept from a knowledge base.

1.4.1 Providing a Content Addressable, Virtual Knowledge Base

Building a knowledge base involves making numerous decisions, including

- what name to give to each knowledge-base frame (*e.g.*, "Plant-Stem" vs. "Stem-of-Plant"), and
- what concepts and relations will be represented explicitly in the knowledge base (*i.e.*, will have an associated frame).

Domain theory and principles of knowledge representation guide some of these decisions. Many of these decisions, however, are arbitrary. This is especially true for multifunctional knowledge because there is not a specific task determining what the knowledge base should contain or how it should be represented. Furthermore, the knowledge engineer is often unaware of these arbitrary decisions.

Although many of the decisions involved in knowledge engineering are arbitrary, they nonetheless impact users of the knowledge base. For example, if a user's only access to frames in the knowledge base is through the frames' names, then the user's ability to find a frame depends on whether he knows (or can guess) what the knowledge engineer named it. If a user only has access to concepts that are explicitly represented in the knowledge base (*i.e.*, concepts that have a corresponding knowledge-base frame), then the knowledge engineer's decision not to reify a concept that happens to be important for a particular task severely limits the user's ability to perform that task. One goal of this research is to insulate users from the effects of the (sometimes arbitrary) choices made during knowledge representation.

An access method can insulate knowledge-base users from the effects of arbitrary choices of frame names by providing *content addressability.* A content addressable knowledge base allows users to access frames not only by specifying the frame name, but also by specifying a partial description of the frame's contents. For example, to access the frame for the concept "eukaryotic cell cytoplasm," the user could describe the concept (in a formal language) as "a kind of cytoplasm that is part of a eukaryotic cell." (More complex types of descriptions are also possible, such as those involving multiple features and nested descriptions.) When given this description in place of a frame name, the access method searches the knowledge base for the frame matching the description and uses the name of that frame in servicing the access request. The advantage of content addressability is that users can access the knowledge base without extensive *a priori* knowledge of its contents.

An access method can insulate knowledge-base users from the effects of arbitrary choices about what is made explicit in the knowledge base by providing a *virtual knowledge base*. In the actual knowledge base, the only concepts and relations that are accessible are those that have been explicitly represented in the knowledge base. In a virtual knowledge base, by contrast, concepts and relations that are implicit in the knowledge base are also accessible. Much research in artificial intelligence has been devoted to developing access methods for *relations* in the virtual knowledge base (*e.g.*, inheritance and rule chaining). The work described here focuses on methods for accessing *concepts* in the virtual knowledge base.

A concept is implicit in the knowledge base (*i.e.*, it is in the virtual knowledge base) if it can be completely defined in terms of other concepts and relations in the knowledge base. For example, if the concepts *Eukaryotic-Cell* and *Cytoplasm* and the relation *part-of* are each represented by a frame in the knowledge base, then the concept "cytoplasm of a eukaryotic cell" is in the virtual knowledge base.

To access a concept in the virtual knowledge base, the user supplies (in a formal language) a definition of the concept in terms of other knowledge-base concepts, such as "the cytoplasm that is part of a eukaryotic cell." The access method creates a new frame for the concept and reorganizes the knowledge base to accommodate it. This reorganization involves finding

- the maximally specific set of concepts in the knowledge base that are more general than the new concept, and

- the maximally general set of concepts that are more specific than the new concept,

and installing the appropriate generalization and specialization links between these frames and the new frame. It also involves recording on the new frame all the information given in the user's specification of the concept (in this example, *part-of* = *Eukaryotic-Cell*). The access method then uses the name of the new frame to service the access request.

When an access method provides both content addressability and access to the virtual knowledge base, users do not need to know whether concepts are explicit in the knowledge base or implicit. Users simply supply a description of the concept to be accessed, embedded in the access request. If the concept already has a frame associated with it, then that frame is found and used; otherwise, a new frame is created and used.

The dynamic creation of new concepts is not a new idea. A major contribution of the KL-ONE knowledge representation system [12] was the introduction of a mechanism for automatically assimilating new concepts into an existing taxonomy based on their descriptions. Most of the terminological languages that descended from KL-ONE also have this facility, called *automatic classification.* Automatic classification as done in KL-ONE and its descendants has several limitations, which this research addresses. Chapter 3 discusses these limitations in more detail.

Providing a content addressable, virtual knowledge base is essentially a problem of automatically extending an index (the taxonomy of frames) and circumventing that index when it is insufficient. Hence, this task does not arise for representation languages that do not provide an index, such as predicate logic. This work is an effort to combine the access flexibility of nonindexed languages with the advantages of an indexed, frame-based language (efficiency of inference, modularity, the ability to describe relations, and the ability to represent intensions).

Chapter 3 details methods for providing a content addressable, virtual knowledge base. Chapter 3 also presents a three-part evaluation of these methods:

- An empirical evaluation of the hypothesis "concepts that are candidates for content addressability and concepts that might exist in the virtual knowledge base but not in the actual knowledge base are prevalent in human-generated text,"

- An analytic evaluation of the strengths and limitations of the approach, and

- An empirical evaluation of the expected cost of accessing concepts in the virtual knowledge base. The results of this study are compared with the theoretical analysis given by Woods [85].

1.4.2 Accessing Coherent Portions of Knowledge

Traditional access methods for frame-based knowledge bases retrieve either the value(s) of a single frame-slot or the values of all slots of a given frame. These methods are often ill-suited to the needs of application programs. For example, consider an advisory or tutoring system that generates explanations of domain knowledge from a knowledge base, such as [43]. With conventional access methods, the system has two unsatisfactory options for selecting the relevant ⟨*frame slot value*⟩ triples from the knowledge base. One, the system can request the value(s) of individual frame-slots one at a time using the "single frame-slot" access method. This requires that the system know in advance which frame-slots are relevant to the explanation it is constructing. Two, the system can request the values of all slots of a frame, then examine each slot value to determine its relevance. This approach is very inefficient, because an explanation of some concept usually requires only a small fraction of all the information found on the frame for that concept. Furthermore, information that is not stored directly on that frame (*i.e.*, information stored on neighboring frames) may be relevant to the explanation of that concept. With this method, the system will miss that information unless it also examines all the slot values on all the neighboring frames.

Methods are needed for accessing, from all the information available on some concept, a coherent subset that is appropriate for a particular task. As a solution, this work includes methods for accessing *viewpoints* of concepts. A viewpoint is a coherent collection of facts that describes a concept from a particular perspective. For example, a structural viewpoint of the concept *Seed-Coat* describes the substances and parts that make up a seed coat and how they are connected. The viewpoint of *Seed-Coat* as a kind of *Container* includes information about what parts of the seed are contained by the seed coat, whether the seed coat has openings, etc. The viewpoint of *Seed-Coat* as having no chlorophyll includes the fact that seed coats are not photosynthetic.

Viewpoints are essential for a variety of tasks. Explanation-generation, advisory, and tutoring systems depend on viewpoints to ensure the coherence of the explanations they generate [43, 46, 47, 49, 78, 64, 65, 79, 51, 52]. Learning systems also use viewpoints. For example, KI uses *views* to constrain the search for consequences of adding new information to a knowledge base [54, 57], and Shrager uses *views* to guide incremental changes to a learner's theory of how a device works so that only coherent theories are learned [72]. Other systems use viewpoints to constrain automated reasoning. For example, Falkenhainer and Forbus use *perspectives* in compositional modeling to ensure consistent modeling assumptions and to increase efficiency [23]. ISAAC [62] and APEX [38] use viewpoints in solving physics problems. BLAH [82] and Algernon [19] use *partitions* and *views* to constrain problem solving and default reasoning. Finally, systems use viewpoints for natural language processing. For example, Grosz uses *focus spaces* to guide disambiguation in discourse understanding [28], and KING uses *views* to guide linguistic and conceptual choices in natural language generation [37]. Although viewpoints are crucial for a variety of tasks, existing methods for dynamically generating viewpoints from a knowledge base are limited. This research provides general (domain and task independent) methods for generating viewpoints.

The approach taken here to the problem of generating viewpoints from knowledge bases is to identify *viewpoint types* by analyzing human-generated texts, then to develop methods for constructing each type of viewpoint from the knowledge base. To access a viewpoint, the user specifies the type of viewpoint wanted and the concept of which the viewpoint will be taken (the *concept of interest*). The access mechanism then determines which relations in the knowledge base are relevant to the requested viewpoint and accesses those relations using frame-slot access methods.

The types of viewpoints identified here are

- *as-kind-of* viewpoints, which describe the concept of interest by relating it to a more general concept. For example, the viewpoint of *Seed-Coat* as a kind of *Container* is an *as-kind-of* viewpoint.

- viewpoints constructed along *basic dimensions*, which describe particular kinds of features of the concept of interest (structural features, functional features, etc.). An example is a structural viewpoint of *Seed-Coat*.

- *as-having* viewpoints, which include features about the concept of interest that are relevant to a user-specified feature of the concept. For example, the viewpoint of *Seed-Coat* as having no chlorophyll is an *as-having* viewpoint.

Chapter 4 describes these viewpoint types in more detail along with methods for generating viewpoints from knowledge bases, either singly or in combinations. Chapter 4 also presents two evaluations of the methods developed for generating viewpoints. The first is an analysis to assess the completeness of the current set of viewpoint types and to guide further refinements and extensions. The second is an objective evaluation to assess the quality of automatically generated viewpoints, as compared to the quality of viewpoints found in human-generated text.

1.4.3 System Architecture

The previous sections have introduced the notions of multifunctional knowledge, content addressability, virtual knowledge, and viewpoints. This section describes how these ideas are integrated into a computer system. Figure 1.1 shows the architecture for a knowledge-base access tool that accesses viewpoints from a content addressable, virtual knowledge base of multifunctional knowledge. In the absence of such a tool, users access the knowledge base solely through modules for frame-slot access or frame access. Using the tool, users can additionally access *viewpoints* from the knowledge base through a module called the *View Retriever* (a term proposed by Suthers [75]). As they perform knowledge-base access, the View Retriever and the modules for frame-slot access and frame access locate frames in the knowledge base using the *Finder*, a module that provides content addressability. The Finder is given either a frame name or a concept description. When given a concept description, the Finder determines the name of the frame matching that description. If the Finder fails to find a frame in the actual knowledge base matching the given description, it passes the description to the *Creator*, a module that creates new frames for concepts in the virtual knowledge base.

The architecture shown in Figure 1.1 has been implemented in a system called *KASTL*, for **K**nowledge **A**ccess **T**ool.

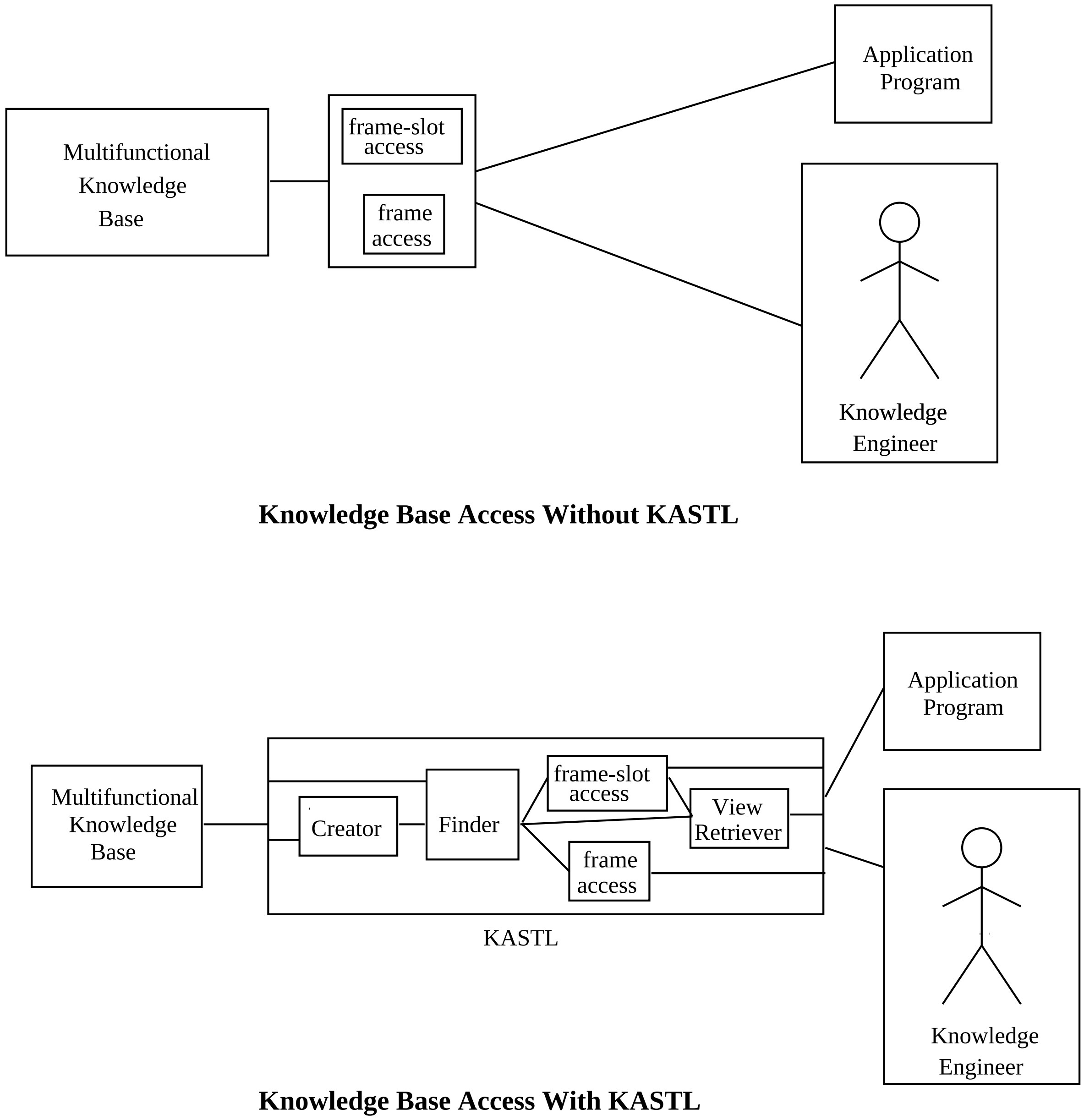

Figure 1.1: Architecture for KASTL Knowledge-Base Access Tool.

1.5 Summary

Human beings have great flexibility in applying their knowledge of a domain to a variety of tasks. An important component of this ability is *knowledge access*, identifying the portion of their knowledge that is relevant to a particular task. Designing computational methods for knowledge access is crucial to developing an artificially intelligent agent that performs multiple knowledge-based tasks.

This research addresses three problems of knowledge access. The first is representing domain knowledge in a formal language in such a way that it can be accessed in support of a variety of tasks. A frame-based language has been developed for constructing comprehensive, fine grained representations of knowledge. This language, KM, allows

- convenient representation of quantified assertions,
- representation of several kinds of statements, both definitional and assertional, and
- convenient representation of contextual information.

A contribution of this research is a specification of the semantics of KM.

The second problem of knowledge access this research addresses is providing users with a content addressable, virtual knowledge base. This allows users to access concepts by their contents, regardless of whether the concepts exist explicitly or implicitly in the knowledge base. The advantages of a content addressable, virtual knowledge base are

- knowledge-base users are insulated from the effects of representational choices made by the knowledge engineer, and
- users can access the knowledge base without extensive *a priori* knowledge of its contents (what frames exist in the knowledge base and what their names are).

The third problem of knowledge access this research addresses is accessing coherent portions of knowledge about a given concept from a large knowledge base. Traditional access methods for frame-based knowledge bases allow users to retrieve either the values of a single frame-slot or the values of all slots on a particular frame. KASTL presents methods for accessing *viewpoints*, coherent collections of facts that describe a concept from a particular perspective. This work identifies several viewpoint types and presents general methods for accessing viewpoints of each type, either singly or in combinations.

The next three chapters discuss this research on each of these three problems of knowledge access.

Chapter 2

Multifunctional Knowledge Representation

> These are the kinds of questions that philosophers have been asking ever since they realized that being a philosopher did not involve any heavy lifting.
> *Dave Barry*

This chapter describes KM, a frame-based language for representing multifunctional knowledge. The first section discusses the motivation for developing a new representation language, and the following sections describe the representational constructs that distinguish KM from traditional frame-based languages.

2.1 Introduction

The motivation for developing a new representation language was the need for more expressive power than existing frame-based languages provide. Although predicate logic provides a great deal of expressive power, a logical representation lacks the advantages of a structured (*i.e.*, frame-based) representation: modularity, the ability to describe relations, the ability to represent concept intensions, and efficiency of inference. Although increasing the expressiveness of a frame-based language reduces the efficiency of inference in general, certain common types of inference (*e.g.*, inheritance, retrieving the value an entity has for a particular attribute, and retrieving all the attributes of entity) remain much more efficient than with predicate logic.

Designing a language for increased expressiveness, even at the cost of intractable or undecidable inference, runs counter to much of the work in knowledge representation. For example, KRYPTON and many of its successors limit expressiveness in an effort to achieve tractability of certain inference algorithms [11]. Experience has shown, however, that this approach results in representation languages so severely limited that they are no longer generally useful, because complete and tractable inference algorithms are impossible for all but the least expressive languages [86, 22]. The tradeoff between expressiveness and efficiency of inference is not an equitable exchange because, although increased expressiveness can substitute for limited inference, inferential power cannot substitute for limited expressiveness; knowledge can only be inferred if it can be represented.

A second motivation for developing a new representation language was the need for constructs that enable convenient representation of common kinds of assertions. This goal reflects the point of view that it is preferable for a representation language to be more convenient for people to use, even if it means that it is less convenient for computer systems to use.

As with other frame-based languages, the basic representational units of KM are *frames* (also called concepts or units), which are collections of *slots* (relations, roles) and their *values* (fillers). The value of a slot on a frame is either a frame name or a constant, such as a string or number. When a frame is intended to describe a set of entities, the frame is said to represent a *category*. Frames that are intended to describe a single individual are said to represent *instances* (of some category). Often a frame represents both a category and an instance. For example, the category *Elephant* is an instance of the category Animal-Species. The term *noncategory instance* refers here to an instance that does not itself have instances, such as *Clyde-the-Elephant*.

Frames in KM also represent slots. For example, the frame representing the slot *part-of* includes information about the slot's inverse (*has-part*), its domain (the frames on which the slot may appear), its range (what kinds of values the slot takes), how many values the slot allows, what slots are more general or more specific, etc. Frames also represent *properties* (such as *oblong* or *patchy*). This chapter concentrates on the representation and semantics of frames that represent categories and instances rather than slots or properties.

The next three subsections discuss the representational constructs of KM that collectively distinguish it from traditional frame-based representation languages. These are constructs for

- representing quantified assertions,
- representing both definitional and assertional statements, and
- representing information contextually.

2.2 Representing Quantified Assertions

Frame-based languages provide a straightforward representation for ground propositions. For example, the triple $\langle$*Clyde color Gray*$\rangle$ represents the logical proposition $color(Clyde, Gray)$. However, many (even most) of the relationships constituting fundamental, general domain knowledge are not between individuals (such as *Clyde* and *Gray*). Rather, they relate categories (such as *Elephant* and *Trunk*). These relationships usually involve universal or existential quantifiers, as in

1. *Every* person has *some* parent that is a person.
2. Joe has *some* parent that is a person.
3. *Every* elephant has color gray.

Although quantified assertions constitute a large portion of fundamental, general domain knowledge, most existing languages, unlike KM, do not allow quantified assertions to be *conveniently* represented (*i.e.*, represented with simple ⟨*frame slot value*⟩ notation). Ground propositions are represented with a single ⟨*frame slot value*⟩ triple, but quantified assertions are more cumbersome to represent, requiring additional representational constructs (such as a rule or constraint language) or multiple triples. For example, in KL-ONE, representing embedded existential quantifiers [as in (1) and (2) above] requires attaching a *value restriction* to a slot (role) of a frame [12]. For example, the fact that Joe has a parent would be represented by a value restriction of *Person* attached to the *parent* slot on the *Joe* frame, rather than by the triple ⟨Joe parent Person⟩. Similarly, the CycL language uses *entryIsA* constraints to encode embedded existential quantifiers: the knowledge enterer must create a new frame, called a *constraint unit*, for the ⟨Joe parent⟩ pair and store *Person* on the *entryIsA* slot on that frame [40]. Representing the same statement in Theo requires a *range facet* associated with the ⟨*Joe parent*⟩ pair [50]. To represent embedded existential quantifiers in Algernon, the knowledge enterer must resort to a rule language [19].

Universal quantification [as in (1) and (3) above] is also cumbersome to represent in many languages. To represent the assertion "All elephants are gray" in CycL, the knowledge enterer states that the value *Gray* for the slot *color* is to be inherited to all frames filling the *allInstances* slot of the *Elephants* frame. To represent the same assertion in Algernon, the knowledge enterer must create a *representative* frame to be associated with the *Elephant* frame and store *Gray* on the *color* slot of the representative frame.

The claim underlying this work is that it is possible to represent quantified assertions in the same way as ground assertions, as single ⟨*frame slot value*⟩ triples, while maintaining a rigorous and explicit notational semantics. Such a representation is made possible by overloading the semantics of slots, in the same sense that operators of a programming language are sometimes overloaded. For example, KM allows "Every person has some parent" to be represented simply as ⟨*Person parent Person*⟩, "Joe has a parent" as ⟨*Joe parent Person*⟩, and "Elephants are gray" as ⟨*Elephant color Gray*⟩, yet it is still possible to determine (from the context) the appropriate quantifiers when reasoning with the information.

Woods outlines the different quantificational patterns that may be implicit in a ⟨*frame slot value*⟩ triple [85]. The quantificational pattern implicit in a triple describes the quantifiers that appear in the logical assertion(s) represented by that triple. The basic nine varieties are based on whether the *frame* and the *value* are to be interpreted as the scope of a universal quantifier, an existential quantifier, or no quantifier. For example, the triple ⟨*Person live-in Place*⟩ may represent "every person lives in some place," called the AE pattern (using A for All and E for Exists). It may alternatively represent "Some person lives in every place," the EA pattern, as "Some person lives in some place," the EE pattern, or “Every person lives in every place,” the AA pattern. Similarly, $\langle$*Person likes John*$\rangle$ suggests the AI and EI patterns (using I for Instance, meaning “unquantified”), which are “Everyone likes John” and “Someone likes John.” For $\langle$*John likes Person*$\rangle$ the IA and IE patterns are “John likes everyone” and “John likes someone.” Finally, the II pattern involves no quantifiers, as with $\langle$*John likes Mary*$\rangle$.

Woods suggests a technique for allowing single $\langle$*frame slot value*$\rangle$ triples to represent assertions having different quantificational patterns. He introduces a set of *relation-forming operators* that construct relations whose semantics entail quantification. For example, the AE operator applies to a relation r to produce a new relation, AE[r], that relates two categories. The new relation asserts that for each instance x of the first, there exists some instance y of the second such that r(x,y). Thus, the relation $\langle$*Person AE[parent] Person*$\rangle$ asserts that every person has as a parent some person. (This is one of several possible interpretations. The next section discusses other kinds of assertions that a particular triple can represent, each sensitive to the intended quantificational pattern.) Woods defines similar relation-forming operators for the other quantificational patterns.

Woods states that the quantificational pattern of the relation of a triple (the operator used to construct it) should be explicitly distinguishable from the underlying relation so that inference methods can reason with it. He proposes representing the operator as an explicit quantificational tag associated with the triple. The disadvantage of this approach is that the knowledge enterer must tag every triple in the knowledge base to indicate its quantificational pattern. This dramatically increases the cost of building and storing the knowledge base.

The KM approach recognizes that quantificational tags are generally unnecessary because the intended quantificational pattern usually can be determined automatically. This disambiguation is possible because most slots are only sensibly combined with a small subset of all the quantificational patterns, and this subset is such that, for a particular occurrence of the slot in a $\langle$*frame slot value*$\rangle$ triple, the characteristics of the frame and value involved determine which quantificational pattern is appropriate.

Experience with the Botany Knowledge Base points to four *semantic types* of slots. A semantic type is an equivalence class of slots based on the quantificational patterns that are implicit in triples involving those slots. In other words, slots of the same semantic type are overloaded in the same way. By attending to the semantic type of slots, a system can automatically determine the quantificational pattern implicit in a particular triple. Type 1 slots are those that typically relate a universally quantified category with an existentially quantified category. An example of a Type 1 slot is *has-part*. When a Type 1 slot relates two categories, as in $\langle$*Plant has-part Root*$\rangle$, the implied quantificational pattern is AE, as in “Every plant has some root.” In addition to AE, Type 1 slots also allow the AI, IE, and II quantificational patterns. When a Type 1 slot relates a noncategory instance to a category, as in $\langle$*Plant-01 has-part Root*$\rangle$, the implied quantificational pattern is IE, as in “Plant-01 has some root.” When the slot relates a category to a noncategory instance, the pattern is AI, and when the slot relates two noncategory instances, the pattern is II.

Semantic Type	*Example Slot*	*Quantificational Patterns Allowed*	*Example Occurrences*
⟨*1*⟩	*has-part*	AE	⟨*Plant has-part Root*⟩
		IE	⟨*Plant-01 has-part Root*⟩
		AI	⟨*Plant has-part Root-01*⟩
		II	⟨*Plant-01 has-part Root-01*⟩
⟨*2*⟩	*color*	AI	⟨*Plant color Green*⟩
		II	⟨*Plant-01 color Green*⟩
⟨*3*⟩	*leaf-shape-of*	IA	⟨*Needle-like leaf-shape-of Pine-Leaf*⟩
		II	⟨*Needle-like leaf-shape-of Pine-Leaf-01*⟩
⟨*4*⟩	*specializations*	II	⟨*Plant specializations Tree*⟩

Table 2.1: Slots are grouped into four semantic types based on the quantificational patterns they allow.

Type 2 slots typically relate a universally quantified category to an instance. An example of this type of slot is *leaf-shape*. When a Type 2 slot occurs on a category frame, as in ⟨*Pine-Leaf leaf-shape Needle-Like*⟩, the implied quantificational pattern is AI, as in "Every pine tree leaf has shape needle-like." Type 2 slots also allow the II quantificational pattern, as in ⟨*Pine-Leaf-01 leaf-shape Needle-Like*⟩.

Type 3 slots typically relate an instance to a universally quantified category. For these slots, the possible interpretations are IA or II. An example of a Type 3 slot is leaf-shape-of, as in ⟨*Needle-Like leaf-shape-of Pine-Leaf*⟩ or ⟨*Needle-Like leaf-shape-of Pine-Leaf-01*⟩.

Type 4 slots relate two instances. For these slots, the only interpretation is II. For example, *specializations* is a Type 4 slot. Although this slot relates a category to a more specialized category, as in ⟨*Plant specializations Tree*⟩, the categories are treated as instances in this context (*i.e.*, they are not given a quantifier) because Type 4 slots allow only the II (unquantified) pattern. Table 2.1 summarizes the four semantic types of slots and the quantificational patterns each allows.

Table 2.2 illustrates how different combinations of frame, slot, and value yield the different quantificational patterns AE, AI, IE, IA, and II. The Botany Knowledge Base does not use the AA pattern because one rarely encounters knowledge of the form "Every X is related to every Y," such as "Every person requires every type of vitamin." The EA, EE, and EI patterns (those that involve an initial existential quantifier) are not used because fundamental domain knowledge deals in generalities (features true of most or all members of a category). When an unusual feature occurs in some subset of a category, that subset usually has other distinguishing characteristics, which leads to its reification as a separate category. The next section discusses another technique for representing assertions such as "Some seeds have an aril" without using an implicit existential quantifier: attaching a likelihood measure L to the assertion "All seeds have an aril," yielding the assertion "All seeds have likelihood (probability) L of having an aril."

Semantic Type of Slot	*Status of Frame*	*Status of Value*	*Quantificational Pattern*
(1)	category	category	AE
(1) (2)	category category	noncategory either	 AI
(1)	noncategory	category	IE
(1) (2) (3) (4)	noncategory noncategory either either	noncategory either noncategory either	 II
(3)	either	category	IA

Table 2.2: The quantificational pattern implicit in triple ⟨*Frame Slot Value*⟩ is determined by the semantic type of *Slot* and the category status of *Frame* and *Value*.

Although slots of the four semantic types described above are overloaded, triples involving them are unambiguous. This is because the choice of interpretation is never between a universal quantifier and an existential quantifier. The choice is always between a universal quantifier and no quantifier at all, or between an existential quantifier and no quantifier at all. Therefore, the decision of which quantificational pattern is appropriate for a particular ⟨*frame slot value*⟩ triple is made by determining whether the frame and value involved are categories: a category takes a quantifier, while a noncategory instance does not.

To summarize, KM represents both quantified and unquantified assertions in the same manner, as simple ⟨*frame slot value*⟩ triples, with no explicit tags, annotations, or constraints needed on triples to signify their quantificational pattern. This is accomplished by overloading the semantics of each slot and assigning each slot to a semantic type according to the kind of overloading. The system can determine automatically the semantics of a particular triple by attending to the semantic type of the slot involved and the category status of the frame and value involved. The knowledge enterer specifies the semantic type of each slot as the slot is created. The type can be recorded on the slot's frame so that it can be referred to by inference methods that must be sensitive to the quantificational pattern of knowledge-base triples.

One important inference method that must be sensitive to the semantic type of slots is *inheritance*. Inheritance is the propagation of slot values from one frame to another, typically to the specializations or instances of a category. The idea behind inheritance is that features common to all members of a set will also occur for any particular member of that set and for any member of any subset of that set. Hence, if a frame represents a category, and a slot value on that frame represents a feature common to all instances of that category, then automatic inheritance can soundly propagate the slot value to any frame representing a *specialization* of the category, and the slot will have the same quantificational pattern on the second frame as on the original frame. Inheritance can also soundly propagate the slot value to a frame representing an *instance* of the category. In this case, however, the slot has a different quantificational pattern on the second frame: the universal quantifier is dropped.

Inheritance is sound only for certain semantic types of slots. Type 1 and Type 2 slots allow an AE or AI interpretation when they appear on frames representing categories (that is, they allow an implicit universal quantifier). Thus, inheritance is sound for triples involving slots of Type 1 or 2, depending on the kind of assertion represented by that triple (as discussed in the next section). Inheritance is not sound for assertions involving slots of type 3 or 4, however, because these slots do not allow the AE or AI interpretations. For example, it would be incorrect to propagate *basic-unit-of*[1] = *Plant* from *Botanical-Cell* to *Root-Xylem-Cell.*

A second important inference method that must be sensitive to the semantic type of slots is *inverse maintenance.* Inverse maintenance is the automatic installation of back pointers in the knowledge base. For example, if the inverse of slot s is defined to be slot s', then given triple $\langle X\ s\ Y \rangle$, automatic inverse maintenance asserts $\langle Y\ s'\ X \rangle$. Automatic inverse maintenance is sound only when the two triples involved have exactly the same semantics (*i.e.*, they represent equivalent logical formulae). That is, representing $\langle X\ s\ Y \rangle$ should trigger the automatic installation of $\langle Y\ s'\ X \rangle$ only when $\langle X\ s\ Y \rangle$ and $\langle Y\ s'\ X \rangle$ mean exactly the same thing. This requirement implies that the inverse of a triple with an AI quantificational pattern is a triple with an IA pattern, and vice versa. It also implies that the inverse of a triple with an II pattern also has an II pattern. Triples with AE or IE patterns, however, do not have inverses that correspond to any of the nine basic quantification patterns. For example, ⟨*Elephant has-part Trunk*⟩ with the AE pattern means "Every elephant has some trunk," but ⟨*Trunk part-of Elephant*⟩ with the EA pattern means something different, that "Some (single) trunk is part of every elephant." Thus triples having AE patterns do not have representable inverses. Similarly, neither do triples having IE patterns.

Because slots of semantic type 2, 3, or 4 do not allow the AE or IE patterns, inverse maintenance for these slots is sound assuming that

- the inverse slot of a Type 4 slot (which admits only the II pattern) is also of Type 4, and
- the inverse slot of a Type 2 slot (which admits only the AI and II patterns) is a slot of Type 3 (which admits only the IA and II patterns), and vice versa.

Because Type 1 slots allow AE and IE quantificational patterns, automatic inverse maintenance for triples involving these slots is not sound. For these triples, the system cannot determine autonomously whether the inverse holds. In this situation the system consults the knowledge enterer to approve or reject the proposed inverse. For example, after encoding the triple ⟨*Photosynthesis raw-materials Water*⟩ to represent the assertion "All photosynthesis events consume some portion of water,"[2] the knowledge engineer would reject the installation of ⟨*Water raw-material-for Photosynthesis*⟩ because it is incorrect that all portions of

[1] a Type 3 slot

[2] The next section discusses how the "all" can be weakened to "most" using likelihood annotations.

water are consumed by some photosynthesis event. (This would be the implied quantificational pattern because *raw-materials* is a Type 1 slot, which assumes the AE pattern when it relates two categories.) On the other hand, after encoding $\langle$*Plant has-part Root*$\rangle$ to represent "All plants have (some) root," the knowledge engineer would accept the installation of $\langle$*Root part-of Plant*$\rangle$ to represent "All roots are part of some plant." These decisions require domain knowledge and cannot be automated.

This section has presented a technique whereby assertions having common patterns of quantification are conveniently represented with single $\langle$*frame slot value*$\rangle$ triples while maintaining a rigorous and explicit semantics. The next section discusses the different kinds of statements to which these quantificational patterns are applied and also gives the *semantic mappings* that translate $\langle$*frame slot value*$\rangle$ triples represented in KM into logical formulae.

2.3 Representing Definitions and Assertions

A *frame* in KM represents a *description* that includes both definitional and assertional components. The definitional component of a description is its essence or meaning, what the description is *intended* to describe. Definition, as the term is used here, corresponds to the term *intension* Woods uses [85, 84]. A definitional feature is more than just a feature that is universal for the category or instance being described. For example, although all calico cats are female, that feature is not part of the definition (intension, meaning, essence) of the category "calico cat." Only the species and color pattern appear in the definition.

The assertional component of a description is a set of statements describing the properties of its *extension*, the things in the world that are characterized by the description. The real world entities that are characterized by a description (*i.e.*, the entities in its extension) are those that people judge to have a sufficient degree of match with the definitional component of the description or those that the description was formed to characterize.

The definitional and assertional components of a description may be partial or empty. Many categories (especially "natural kinds") cannot be completely defined. Thus some descriptions have an empty or partial definitional component. Similarly, some descriptions have no extension and thus no assertional component (*e.g.*, "colorless green ideas" or "unicorns"). This allows the representation of concepts that have various kinds of existence (or nonexistence) [34].

A description, as the term is used here, differs from a mathematical set or a logical predicate. First, two sets (or two predicates) are equivalent if they are coextensional. By contrast, two descriptions that have different definitional components are distinguishable, even if they are coextensional (*e.g.,* the Morning Star and the Evening Star). Second, while the definition of a set in set theory (or a predicate in logic) fully describes what the set's extension is, the definitional component of a description only describes what the extension is *intended* to be. There may be a discrepancy between the definitional component of a description and the description's extension. For example, part of the definition of "mammal" is "gives birth to live young," yet its extension includes the egg-laying echidna and platypus.

Because there may be a discrepancy between the definitional component of a description and its extension, definitions cannot be interpreted as making assertions about the extension. Such assertions constitute the assertional component of a description. The degree of match between statements in the definitional component and statements in the assertional component reflects the degree to which a description's definition matches its extension.

Because of the differences between descriptions and mathematical sets, and because first-order logic is based on set theory, first-order logic cannot completely capture the semantics of KM's frames. The definitional component of a description D is characterized with statements of the form

> D is intended to characterize the entity/entities x for which
> $\{$logical formula involving $x\}$ holds.

The assertional component of a description, however, is expressed by logical formulae.

Unlike KM, most frame-based languages do not allow both definitions and assertions to be represented within the $\langle$*frame slot value*$\rangle$ framework. KL-ONE, NIKL, and other terminological languages deal almost exclusively with definitions rather than assertions [86]. KRYPTON and other hybrid languages represent assertions as well as definitions, but most represent them in the "ABox" using a logic-like language or rules rather than the frame language of the "TBox" [11]. (One exception is CLASSIC, which uses the same language for both [86].) CycL and Algernon represent assertions, but not definitions (as the term is used here) [40, 19]. KM provides an expressive language for representing both the definitional and assertional components of a description within the $\langle$*frame slot value*$\rangle$ framework.

2.3.1 Representing Definitions

Although definitions cannot be represented in or manipulated by first-order logic, they are nonetheless useful for explanation and as metaknowledge for reasoning. For example, features that are part of the definition of a concept should be the last features to be relaxed in the face of a counterexample [85].

To represent the definitional component of a description on the same frame with the assertional component, KM must include representational constructs that distinguish definitional features from nondefinitional assertions. In addition, KM must distinguish between necessity and sufficiency. Most terminological languages (*e.g.*, KL-ONE) represent definitions in terms of features that are *both* necessary *and* sufficient [86]. Many natural concepts, however, have only partial definitions. That is, they have features that are necessary but not sufficient, or vice versa. To represent the definitions of such concepts, KM must include representational constructs that allow separate statements of necessary features versus sufficient features.

As a solution, KM allows *annotations* on triples in the knowledge base: each triple can be modified with any number of filled slots. Some of these annotations represent domain knowledge. For example, annotations may describe the conditions under which a relationship holds or give an explanation of a relationship. Other annotations signify the semantics of the triple they modify. I call the latter *semantic annotations*. The semantic annotations of a triple, together with its implicit quantificational pattern, determine its semantics. Two of the semantic annotations distinguish definitional features. They are

- *definitionally-necessary?* = T, and
- *definitionally-sufficient?* = T

These annotations can occur either singly or together. (The next section discusses the semantic annotations that express nondefinitional assertions.)

The intuitive semantics of definitional necessity and sufficiency is as follows. A feature (*i.e.*, slot value) marked as definitionally necessary for membership in a category indicates that the category is intended to include *only* (but not necessarily all) things having that feature. A set of features marked as definitionally sufficient for membership in a category indicates that the category is intended to include *all* (but not necessarily only) things that have *all*[3] those features.

Features can be annotated as definitionally necessary and definitionally sufficient for individuals as well as categories. A definitionally necessary feature on a description of an individual indicates that the description is intended to describe *some* entity having that feature. A definitionally sufficient feature indicates that the description is intended to describe *the* entity having that feature and any other features marked as definitionally sufficient. Thus, for individuals, sufficiency implies necessity.

The default value for *definitionally-necessary?* and *definitionally-sufficient?* is *nil*, so that only triples representing definitional features require these annotations. Experience with the Botany Knowledge Base suggests that most features are purely assertional rather than definitional.

Table 2.3 characterizes the meaning of features that are annotated as definitionally necessary or definitionally sufficient. This characterization is given by a *semantic mapping* from triples to definitions. This mapping is sensitive to both the semantic annotations that occur (*i.e., definitionally-necessary?* and *definitionally-sufficient?*) and the quantificational pattern (as determined by the semantic type of the slot and the category status of the frame and value).

2.3.2 Representing Assertions

The assertional component of a description is a set of assertions describing the properties of the things in the world that are characterized by the description. KM accommodates a variety of assertions types, including defeasible assertions (with varying degrees of defeasibility) and nondefeasible assertions.

Most representation languages describe extensions solely with sets of defeasible assertions (usually default features with no degree of defeasibility) [9] or solely with sets of universal (nondefeasible) features, as in the variations of predicate logic or the Prolog-like rules used

[3] Definitionally sufficient features are *jointly* sufficient for category membership.

Semantic Type of Slot S	*Status of Frame* F	*Status of Value* V	*Semantics when* definitionally-necessary? = T	*Semantics when* definitionally-sufficient? = T
(1)	category	category	F is intended to include only entities x for which $\exists y \in V.S(x,y)$.	F is intended to include all entities x for which $\exists y \in V.S(x,y)$ and x is otherwise sufficient.
(1) (2)	category category	noncategory either	F is intended to include only entities x for which $S(x,V)$.	F is intended to include all entities x for which $S(x,V)$ and x is otherwise sufficient.
(1)	noncategory	category	F is intended to be some entity x for which $\exists y \in V.S(x,y)$	F is intended to be the entity x for which $\exists y \in V.S(x,y)$ and x is otherwise sufficient.
(1) (2) (3) (4)	noncategory noncategory either either	noncategory either noncategory either	F is intended to be some entity x for which $S(x,V)$.	F is intended to be the entity x for which $S(x,V)$ and x is otherwise sufficient.
(3)	either	category	F is intended to be some entity x such that $\forall y \in V.S(x,y)$	F is intended to be the entity x such that $\forall y \in V.S(x,y)$ and x is otherwise sufficient.

Table 2.3: The semantic mapping of triple $\langle F\ S\ V \rangle$ into a definition, as determined by the triple's semantic annotations, the semantic type of S, and the category status of F and V. (In the table, "otherwise sufficient" means "satisfies other criteria marked as definitionally sufficient.")

by hybrid languages (*e.g.*, KRYPTON [11]). The advantage of the first approach is that it allows us to represent the great proportion of knowledge that is defeasible. The advantage of the second approach is that nondefeasible assertions allow sound and more efficient inference methods. Both advantages are important enough to warrant a knowledge representation language that allows *both* kinds of assertions to be represented.

KM accommodates both defeasible and absolute assertions, as well as different degrees of defeasibility. To represent both kinds of knowledge using the same ⟨*frame slot value*⟩ format, KM (again) uses semantic annotations. The knowledge enterer attaches semantic annotations to triples to indicate:

- the degree of belief, as a qualitative or quantitative probability, and
- the assertion that is modified by the degree of belief,
  out of the potentially several assertions that the same triple can represent.

First, semantic annotations convey the knowledge enterer's *degree of belief* in some assertion. Degrees of belief are expressed as probabilities. Although probabilities are commonly thought to correspond to frequency ratios, Cheeseman argues that, to the contrary, probabilities always correspond to a measure of belief [16]. (He notes, however, that for large sets, measure of belief approximates frequency.) Probability theory is emerging as a powerful paradigm for representing and reasoning about uncertainty. Neufeld posits that probability theory underlies every current AI formalism for reasoning about uncertainty [59]. Cheeseman argues that, under the measure of belief interpretation, probability theory also subsumes fuzzy logic [17]. Furthermore, Rich shows that when default reasoning is treated as likelihood reasoning, natural solutions emerge for several problems encountered by traditional methods [69].

Probabilities are specified in KM either as qualitative ranges or as specific numbers. Although numeric probabilities facilitate reasoning, they are difficult to acquire. Domain experts find it difficult to assign a precise number to their degree of belief in an assertion. Furthermore, the precision that numeric probabilities provide is often unnecessary. A study of the certainty factors used in MYCIN demonstrated that, although the system allows 1000 distinct numeric values, reducing the set to five qualitative ranges provided comparable system performance [13]. To simplify the representation of probabilities, the Botany Knowledge Base uses a set of eight qualitative values, each of which represents a range of probabilities. This chapter uses a simplified set consisting of *Low, Medium*, and *High* (in addition to the endpoints zero and one). Although KM does not currently include methods for reasoning with qualitative probabilities, Grosof's methods for reasoning with bounded conditional probabilities apply [27].

We turn now to a discussion of the second kind of information that semantic annotations modifying assertions convey. Each triple in the knowledge base can simultaneously represent several assertions about the extension of a description. (These multiple interpretations are orthogonal to the choice of quantificational pattern, discussed in the previous section. These two issues will be integrated shortly.) A semantic annotation on a triple indicates which of the assertions the triple represents has the specified degree of belief.

Consider the different assertions that a triple in the knowledge base can represent about the extension of a description (in addition to any definitional statements that the triple represents). Just as definitions comprise two different kinds of statements (statements of definitional necessity and definitional sufficiency), assertions come in two analogous varieties. For example, assume that for triple $\langle frame\ slot\ val\rangle$, *frame* represents category C and $slot = val$ represents feature F. The strongest assertions regarding how satisfaction of F relates to membership in C are

1. all instances of C have feature F: $\forall x.[x \in C \Rightarrow slot(x, val)]$
2. all entities having feature F are instances of C: $\forall x.[slot(x, val) \Rightarrow x \in C]$

Although this is the same distinction made in the previous section between statements of definitional necessity and definitional sufficiency, and although logicians often refer to these as assertions of necessity and sufficiency, this differs from definitional necessity/sufficiency. Definitional necessity is part of the meaning or essence of a concept, but assertion (1) above expresses necessity in the sense of "happens to be true in all cases." Hence, I call the latter *extensional necessity* to distinguish it from definitional necessity. Similarly, I call the variety of sufficiency expressed by assertion (2) *extensional sufficiency.* The degree of match between the *extensional* necessity/sufficiency of features in a description and the *definitional* necessity/sufficiency of those same features reflects the degree to which the description's extension matches its definitional component.

We see then that assertions (1) and (2) above are the two kinds of nondefinitional assertions that a particular triple can represent. Consider how KM might associate degrees of belief with each of these assertions. One approach is to attach a probability to the entire assertion. Nilsson's probabilistic logic allows probabilities to be associated with logical sentences [61]. In probabilistic logic, the interpretation of a logical sentence is a probability distribution rather than a truth value. Thus, probabilities become generalized truth values. The probability of a sentence S being true is the probability (degree of belief) that the actual world corresponds to some possible world in which S is true.

The problem with attaching probabilities directly to assertions (1) and (2) is that what one usually wants to express is not, for example, the degree of belief in the assertion "all birds fly," but rather the degree of belief in the assertion that a randomly selected bird B flies (possibly based on an estimate of the percentage of birds that fly). Rather than associate a probability with assertion (1) or (2) as a whole, one usually wants to express the probability of the *consequent* of the implication given that the antecedent is satisfied for some entity within the scope of quantification. (Grosof extends Nilsson's probabilistic logic to allow such conditional probabilities to be attached to assertions [27].)

Before attaching probabilities to the consequents of the implications in assertions (1) and (2), we must recognize that each assertion has two possible forms: the forms shown above, and their contrapositives. That is, the set of assertions above is equivalent to the following set:

(1A)	$\forall x.[x \in C \Rightarrow slot(x, val)]$	
(1B)	$\forall x.[\neg slot(x, val) \Rightarrow x \notin C]$	(the contrapositive of (1A))
(2A)	$\forall x.[slot(x, val) \Rightarrow x \in C]$	
(2B)	$\forall x.[x \notin C \Rightarrow \neg slot(x, val)]$	(the contrapositive of (2A))

Although (1A) and (1B) are logically equivalent (as are (2A) and (2B)), they differ in their consequents. Hence, for the purpose of associating probabilities, they are distinct assertions. A particular ⟨*frame slot value*⟩ triple in the knowledge base can simultaneously represent all four of these assertions, each with a different probability (degree of belief) associated with its consequent.

When we associate a probability with the consequent of (1A), we express the *likelihood* that a particular instance of C has feature F (an estimate of the frequency with which F occurs among instances of C). When we associate a probability with the consequent of (1B), we express the degree of *extensional necessity* of feature F for category C, the probability with which an entity is not an instance of C based on the absence of F. When we associate a probability with the consequent of (2A), we express the degree of *extensional sufficiency* of feature F for category C, the probability with which an entity is an instance of C based on the presence of F. This corresponds to the term *cue validity* used in the psychological literature. When we associate a probability with the consequent of (2B), we express the rarity of feature F outside category C. I call this the *uniqueness* of F for C. Likelihood and uniqueness information is useful for prediction, while necessity and sufficiency information is useful for classification.

To provide concise and convenient representations, KM allows assertions of the likelihood, extensional necessity, extensional sufficiency, and uniqueness of a feature for a category all to be represented with the same ⟨*frame slot value*⟩ triple. (The same triple can also represent definitional statements, as described in the previous subsection.) This is accomplished by using semantic annotations called *likelihood, necessity, cue-validity*, and *uniqueness*, each taking a qualitative or quantitative probability value. For example, the following representation

```
Human-Male
——————————
has-disease: Hemophilia
                    likelihood: Low
                    necessity: Low
                    cue-validity: High
                    uniqueness: High
```

makes the following assertions:

1. Hemophilia is rare among men.
2. Absence of hemophilia is very weak evidence that an entity is *not* a man.
3. Having hemophilia is very good evidence that an entity is a man.
4. Hemophilia is rare for things that aren't men.

Although the value of cue-validity (extensionally sufficiency) is *High* (and even if the value were 1), the absence of semantic annotation *definitionally-sufficient?* = T signifies that the concept *Human-Male* is not *intended* to include all hemophiliacs, even though most hemophiliacs are men.

KM combines the technique of representing a variety of assertions by the same knowledge-base triple with the technique introduced in the previous section for representing different quantificational patterns. This allows the representation of assertions about the extensions of categories *and* individuals, and the representation of assertions involving either quantifier-free features or features having existential quantifiers. For a particular ⟨*frame slot value*⟩ triple, the semantic type of *slot* and the category status of *frame* and *value* determine the quantificational pattern of all the assertions that triple represents, and the semantic annotations of the triple determine the degree of belief assigned to each assertion. Table 2.4 gives the semantic mapping from triples to logical formulae.

As these mappings illustrate, values given for likelihood and uniqueness specify the *absolute* probability that some entity has a feature, given that it does or does not belong to some category (or, for an instance, given that it is or is not some individual). For example, an annotation of *likelihood* = 0 on a feature of a category indicates the belief that no instances of the category have that feature. An annotation of *likelihood* = 1 indicates the belief that all instances of the category have that feature.

Values given for necessity and cue-validity specify a *change* in belief (rather than an absolute degree of belief) that an entity is an instance of some category (or is some individual), given the absence or presence of some feature. For example, an annotation of *cue-validity* = 0 on feature F of category C indicates that the presence of F does not change the strength of *a priori* belief that an entity is an instance of C. An annotation of *cue-validity* = 1 indicates the belief that the presence of F guarantees that an entity is an instance of C.

Necessity and cue-validity are defined in the same way as the measures of belief/disbelief that make up MYCIN's certainty factors [13]. Experience with MYCIN indicates that, for classification tasks (one purpose of necessity and cue-validity information), experts are more willing to give changes in belief than absolute probabilities. On the other hand, likelihood and uniqueness seem to be more naturally expressed as absolute probabilities.

Semantic Type of Slot S	*Status of Frame* F	*Status of Value* V	*Semantics of* $\langle F\ S\ V \rangle$ *assuming likelihood* $= L$, *cue-validity* $= CV$, *necessity* $= N$, and *uniqueness* $= U$
(1)	category	category	$\forall x.[P(\exists v \in V.S(x,v)/x \in F) = L]$ $\forall x.[\frac{P(x \in F/\exists v \in V.S(x,v)) - P(x \in F)}{1 - P(x \in F)} = CV]$ $\forall x.[\frac{P(x \in F) - P(x \in F/(\nexists v \in V.S(x,v)))}{P(x \in F)} = N]$ $\forall x.[P(\nexists v \in V.S(x,v)/x \notin F) = U]$
(1) (2)	category category	noncategory either	$\forall x.[P(S(x,V)/x \in F) = L]$ $\forall x.[\frac{P(x \in F/S(x,V)) - P(x \in F)}{1 - P(x \in F)} = CV]$ $\forall x.[\frac{P(x \in F) - P(x \in F/\neg S(x,V))}{P(x \in F)} = N]$ $\forall x.[P(\neg S(x,V)/x \notin F) = U]$
(1)	noncategory	category	$P(\exists v \in V.S(F,v)) = L$ $\forall x.[\frac{P(x=F/\exists v \in V.S(x,v)) - P(x=F)}{1 - P(x=F)} = CV]$ $\forall x.[\frac{P(x=F) - P(x=F/(\nexists v \in V.S(x,v)))}{P(x=F)} = N]$ $\forall x.[P(\nexists v \in V.S(x,v)/x \neq F) = U]$
(1) (2) (3) (4)	noncategory noncategory either either	noncategory either noncategory either	$P(S(F,V)) = L$ $\forall x.[\frac{P(x=F/S(x,V)) - P(x=F)}{1 - P(x=F)} = CV]$ $\forall x.[\frac{P(x=F) - P(x=F/\neg S(x,V))}{P(x=F)} = N]$ $\forall x.[P(\neg S(x,V)/x \neq F) = U]$
(3)	either	category	$\forall v.P(S(F,v)/v \in V) = L$ $\forall x.[\frac{P(x=F/\forall v \in V.S(x,v)) - P(x=F)}{1 - P(x=F)} = CV]$ $\forall x.[\frac{P(x=F) - P(x=F/(\exists v \in V.\neg S(x,v)))}{P(x=F)} = N]$ $\forall x.[P(\exists v \in V.\neg S(x,v)/x \neq F) = U]$

Table 2.4: Semantic mapping of a triple $\langle F\ S\ V \rangle$ to formulae in probabilistic logic, as determined by the semantic type of slot S, the category status of frame F and value V, and the semantic annotations of the triple. $P(A)$ indicates the probability of logical sentence A, and $P(A1/A2)$ indicates the conditional probability of A1 given that $P(A2) = 1$.

A possible explanation for this difference is that classification tasks typically involve

- combining *multiple* pieces of evidence, and
- ranking *multiple* competing hypotheses, possibly from an unspecified set.

For example, a disease diagnosis problem usually has the form "Given symptoms s_1,s_2,...,s_n, which of diseases d_1, d_2,...,d_m (or even "which of the diseases I know of") is most likely?" It is more important to determine which diagnosis (classification) is most likely than to determine the exact likelihood of any particular diagnosis. Necessity and cue validity, defined as measures of increases/decreases in belief, are more convenient for computations of this sort than are absolute probabilities.

Prediction tasks, by contrast, typically involve

- determining a feature based on membership in a *single* category, and
- determining the likelihood of a *single*, prespecified feature,

as in "I believe/know that this thing is a C; how likely is it that it has feature F?" An example is "That animal appears to be a dog. How likely is it that it will bite me?" For this kind of task, absolute probabilities are more appropriate than measures of increased/decreased belief.

Likelihood, necessity, cue-validity, and uniqueness annotations can be given qualitative ranges, such as *Low, Medium*, and *High*, in addition to numeric values. For likelihood and uniqueness, which give a conditional probability that an entity will have some feature, it is convenient to use a range of qualitative values centered on the *a priori* probability of that feature. For example, *Low* could be defined to include most numbers less than the *a priori* probability, *Medium* the numbers close to it on either side, and *High* most numbers greater than it. Necessity and cue-validity, by contrast, represent a change in belief rather than an absolute belief (and thus are relative to the *a priori* probability by construction). Hence, it is more convenient for these annotations to use qualitative values that constitute a uniform partitioning of the range [0..1].

Inheritance of features and automatic inverse maintenance must be sensitive to likelihood, necessity, cue-validity, and uniqueness annotations. The only assertions for which inheritance is sound are those for which $likelihood = 0$, those for which $likelihood = 1$, those for which $necessity = 1$, and those for which $cue\text{-}validity = 0$. The only semantic annotation for which automatic inverse maintenance is sound is the likelihood annotation, but this is limited to triples involving slots of semantic type 2, 3, or 4, or slots of semantic type 1 under the II quantificational pattern.

To summarize, the semantic annotations *likelihood, necessity, cue-validity*, and *uniqueness* allow each triple in the knowledge base to simultaneously represent a variety of assertions about the extension of a description. By assigning probabilities representing degrees of belief (or changes in belief) to the semantic annotations, KM accommodates both defeasible and nondefeasible assertions. Attaching semantic annotations directly to triples makes knowledge representation more convenient than the approach CycL uses, in which necessity, cue validity, etc., must be stored on constraint units, separate frames associated with the triples being described [40].

The Botany Knowledge Base uses the semantic annotations described here with the exception of *uniqueness*. Uniqueness seems to have limited utility for two reasons. First, uniqueness represents assertions about *nonmembers* of a category C on the frame representing C (*e.g.*, "Things that aren't cats usually don't have tails" stored on the *Cat* frame). It seems unlikely that a reasoning system would access the frame for C to predict features of individuals that are not instances of C. Second, uniqueness is rarely informative, because almost all features in a nontrivial domain have high uniqueness. That is, in a domain for which most categories constitute a small portion of the universe and for which most features occur in a small percentage of entities, most features will likewise occur in a small percentage of entities *not* in any given category, meaning they have high uniqueness for that category. For exceptional situations where uniqueness is low (or zero) for some feature on some category, it seems more natural to create a complement category and annotate the feature with $likelihood = High$ (or $likelihood = 1$) on the complement category. When $uniqueness = 1$, that information need not be represented at all, because it turns out that $uniqueness = 1$ if and only if $cue\text{-}validity = 1$. Because uniqueness annotations appear to be neither useful nor informative, the Botany Knowledge Base does not include them.

2.4 Representing Information Contextually

This section describes the third major extension of KM, *value annotations* for representing information contextually.

Consider representing the assertion "Cells of plants have cell walls." Traditional representation languages provide two unsatisfactory alternatives. The first technique is to associate an "if-needed" rule with the *has-part* slot on the *Cell* frame that states, in effect, "If the cell is part of a plant, then one of its parts is a cell wall." This approach requires that the representation language include a formalism for representing and reasoning about such rules. It also has the disadvantage that it provides no direct access path from the frame *Plant* to the knowledge about cells that are part of plants unless a special mechanism installs pointers from frames to the rules in whose antecedents they appear.

The second technique is to create a frame for the concept "Cell that is part of some plant," then fill in the appropriate slot values on that frame, as in

Cell-of-A-Plant
————————-
generalizations: Cell
part-of: Plant
has-part: Cell-Wall

(Semantic annotations would also be installed if they differ from the defaults.)

If little or no additional knowledge differentiates the concept *Cell-of-A-Plant* from the concept *Cell*, then this approach requires an inordinate amount of effort. With this approach, representing a single assertion involves

- creating and naming a new frame,
- reorganizing the taxonomy to accommodate the new frame,
- installing the defining properties of the new concept (*e.g.*, part-of = Plant), and
- representing the original assertion (*e.g.*, has-part = Cell-Wall).

Thus the overhead for representing a single assertion of this kind is increased threefold or more above that required for other assertions.

Another disadvantage of this approach is that it results in a proliferation of frames corresponding to concepts that are important only in very limited contexts (such as "water contained in a guard-cell that is collapsing" and "water pore in the membrane of the epidermis of a root"). Ideally, each frame in the knowledge base would correspond to a stable concept in the mind of the domain expert, that is, a concept one would expect to find in the index of a comprehensive text on the domain. Such a knowledge base is easier to navigate, for both knowledge engineers and application programs (such as a program that performs spreading activation searches). Lenat and Guha give a similar argument for limiting frame proliferation [40].

An alternative technique for representing the statement "Cells of plants contain cell walls," one that necessitates neither a special rule language nor the creation of a frame for "cell that is part of some plant," is to represent the information *contextually*. That is, the triple ⟨Cell has-part Cell-Wall⟩ occurs in the context of the triple ⟨Plant has-part Cell⟩, signifying that cells *that are part of some plant* have a cell wall. The advantages of this approach over the two previous techniques are

- assertions like the one above are more convenient to represent, and
- the resulting knowledge base is easier to inspect, edit, and reason with, because related information is kept together and because only the major concepts of the domain are reified as frames.

To allow contextual representations, KM allows *value annotations*, whereby values of slots can be further described by any number of filled slots. Hence, KM has a recursive notation: frames have slots with values, which themselves have slots with values, which have slots with values, etc. Value annotations differ from the semantic annotations described in the previous section (and from the slot entry details of the CycL language [40]) in that semantic annotations describe entire triples and the assertions they represent, while value annotations describe categories, just as filled slots on frames describe categories. For example, a semantic annotation, such as *likelihood* = 1, on triple ⟨*Plant has-part Cell*⟩ gives information about the relationship between Plants and Cells, while a value annotation on ⟨*Plant has-part Cell*⟩ gives information about cells (in particular, about cells that are part of some plant).

The CycL language provides a construct similar to value annotations [40]. In CycL, associated with each frame-slot is a set of features (filled slots) to be inherited to all values filling that frame-slot. For example, the statement "Regions with hilly topography tend to have rocky soil" can be represented by associating the feature *soil-rockiness* = *High* with the frame-slot ⟨*Hilly topography-of*⟩ as a feature to be inherited to all frames acting as values of that frame-slot (*i.e.*, all frames that have the slot *topography* filled with value *Hilly*). This construct differs from KM's value annotations in that, in CycL, the set of to-be-inherited features must apply to every value filling the frame-slot, while KM allows the knowledge enterer to describe each value of the frame-slot independently of other values. For example, with value annotations one could represent "Regions with hilly topography that are temperate regions have rocky soil" and "Regions with hilly topography that are desert regions tend to have sandy soil."

Value annotations do not extend the expressive power of KM. Hence, the semantics of value annotations can be specified by describing a procedure for transforming a knowledge base containing value annotations into a knowledge base containing only ⟨*frame slot value*⟩ triples and then relying on the semantic mappings for triples given previously. Below is a description of the semantics of the general form of a value annotation, illustrated with an example:[4]

```
Frame1
-------
slot1: Frame2
          slot2: Frame3

Plant
-----
has-part: Cell
            (likelihood: 1)
            has-part: Cell-Wall
                        (likelihood: High)
```

Intuitively, the meaning of the example is as follows:

> All plants have as a part one or more cells, and most of *those* cells have as a part one or more cell walls. In other words, most cells that are part of some plant have as a part one or more cell walls.

[4]For clarity, semantic annotations are shown in parentheses to distinguish them from value annotations.

KM requires that the triple $\langle$*Frame1 slot1 Frame2*$\rangle$ represents assertions in which $Frame2$ is an existentially quantified category. That is, $Frame2$ represents a category and *slot*1 is of semantic type 1 taking quantificational pattern AE or IE in this context. The intuitive meaning is that the category $Frame2$ is being further constrained (specialized) by this context (the relationship to $Frame1$), and the value annotation *slot*2 $=$ $Frame3$ provides additional information about this specialization of $Frame2$. For example, $Cell$ is being (implicitly) specialized in the context of $\langle$*Plant has-part Cell*$\rangle$, and the value annotation *has-part* $=$ *Cell-Wall* describes this new kind of cell (*i.e.*, a plant cell).

The reason KM requires that $Frame2$ represent a category is that if $Frame2$ represents an individual in this context, then value annotations are unnecessary. If $Frame2$ represents an individual then $Frame2$ is not being constrained (specialized) by this context, and information about $Frame2$ as an individual can be represented on the frame called $Frame2$.

The semantics of the above constructs are obtained by conjoining

- the formulae represented by the triple $\langle$*Frame1 slot1 Frame2*$\rangle$, as determined by the semantic mappings given in the previous section (for example, $\langle$*Plant has-part Cell*$\rangle$ above represents "All plants have as a part some cells."), and
- the formulae represented by the triples introduced by the following procedure, which translates the value annotations into standard $\langle$*frame slot value*$\rangle$ triples. (KM does not actually perform this procedure; it is given here solely to explain the semantics of value annotations.)

  1. Create an explicit specialization of $Frame2$, $Frame2'$. For example, create a specialization of $Cell$, $Cell'$ (which corresponds to the category "Cell of a plant").
  2. Represent the definition of the new category by installing $\langle Frame2'\ slot1'\ Frame1\rangle$ (where $slot1'$ is defined as the inverse of *slot*), with the following semantic annotations:
     - *definitionally-necessary?* $= T$,
     - *definitionally-sufficient?* $= T$,
     - $necessity = 1$,
     - $likelihood = 1$, and
     - *cue-validity* $= 1$.

     (The last three annotations assert that, because this new category is not a "natural kind," the category's extension matches its definition perfectly.) For example, assert as the defining criterion of category $Cell'$ the feature *part-of* $=$ *Plant.*
  3. Assert $\langle Frame2'\ slot2\ Frame3\rangle$ with the same semantic annotations as given for the value annotation $slot2 = Frame3$. For example, assert $\langle$*Cell' has-part Cell-Wall*$\rangle$ with semantic annotation *likelihood* $=$ *High.*

The result of steps (1) through (3) for the value annotation on $\langle$*Plant has-part Cell*$\rangle$ is shown below.

Cell'
——-
generalizations: Cell
part-of: Plant
    (definitionally-necessary?: T)
    (definitionally-sufficient?: T)
    (necessity: 1)
    (likelihood: 1)
    (cue-validity: 1)
has-part: Cell-Wall
    (likelihood: High)

In this example, the triple whose value was annotated ($\langle$*Plant has-part Cell*$\rangle$) had semantic annotation *likelihood* = 1, and the triple serving as the value annotation ($\langle$*Cell has-part Cell-Wall*$\rangle$) had semantic annotation *likelihood* = *High*. Thus the resulting interpretation was

> All plants have as a part one or more cells, and most of *those* cells have as a part one or more cell walls. In other words, most cells that are part of some plant have as a part one or more cell walls.

Below are three different interpretations that are achieved by using different combinations of likelihood values *High* and 1 (the most common values). Although in these examples both triples use the AE quantificational pattern, other patterns are possible, depending on the semantic type of the slots involved and the category status of the frames involved.

Plant
———
has-part: Cell
    (likelihood: High)
    has-part: Cell-Wall
        (likelihood: High)

> Most plants have cells, and most of those cells have cell walls.

Plant
———
has-part: Cell
    (likelihood: High)
    has-part: Cell-Wall
        (likelihood: 1)

Most plants have cells, and all those cells have cells walls. In other words, most plants have cells with cell walls, and all cells of plants have cell walls.

Plant
———
has-part: Cell
    (likelihood: 1)
    has-part: Cell-Wall
        (likelihood: 1)

All plants have cells, and all those cells have cell walls. In other words, all plants have cells with walls, and all cells of plants have cell walls.

Adding value annotations to KM necessitated modifying KM's frame-slot access method. In frame-based languages, slot values are accessed by specifying the *address* of the required value. For conventional languages, an address is simply a (frame-name slot-name) pair. To provide access to value annotations nested to any depth, KM accepts addresses of the form

$$\text{(frame-name slot-name \{frame-name slot-name\}}*\text{)}$$

where $*$ indicates zero or more repetitions (Kleene star). For instance, to access the value annotation in the previous examples, the address required is (Plant has-part Cell has-part), which retrieves the part(s) of plant cells. Given this address, the access function returns the value *Cell-Wall* (along with any other valid values). Incidentally, values for semantic annotations (*likelihood, necessity*, etc.) have addresses of the same form. For example, the address (Plant has-part Cell likelihood) refers to the probability that a particular plant has a cell part.

As suggested above, a value annotation can be thought of as information about an *implicit specialization* of a category. For instance, the value annotation on ⟨*Plant has-part Cell*⟩ in the above example can be thought of as describing the implicit specialization of Cell, "Cell that is part of some plant." In KM this notion has been incorporated into the representation language. The set of value annotations associated with a particular slot value V is said to constitute an *embedded unit* representing an implicit specialization of V. KM treats embedded units as equals with explicit frames: they are automatically linked into the concept taxonomy, they participate in inverse maintenance and inheritance, and slot values are stored on and retrieved from embedded units just as they are for explicit frames. Thus, information stored on value annotations is just as accessible as information stored on explicit frame slots, yet the disadvantages of creating frames to house them are avoided.

Although KM treats embedded units equivalently with explicit frames, embedded units are still distinguishable from explicit frames. Thus, users of the knowledge base can ignore embedded units when it is convenient to do so. In this way KM retains the advantages of having frames represent only the most important concepts in the domain.

The previous example serves to illustrate KM's treatment of embedded units:

```
Plant
-----
has-part: Cell
               has-part: Cell-Wall
```

When the knowledge enterer installs the value annotation *has-part* = *Cell-Wall*, KM performs the following activities automatically:

1. Represent the definition of the embedded unit: add *part-of* = *Plant* as a definitionally necessary and sufficient feature of the new concept by installing it as a (second) value annotation of ⟨*Plant has-part Cell*⟩. This annotation also serves as an inverse for ⟨*Plant has-part Cell*⟩. It is safe to assume that the alternative inverse ⟨*Cell part-of Plant*⟩ was rejected by the knowledge engineer (that it isn't true that most cells are part of some plant), because otherwise the knowledge engineer would have no reason to annotate ⟨*Plant has-part Cell*⟩: he could represent information about cells that are part of some plant directly on the *Cell* frame instead.

2. Connect the embedded unit into the knowledge-base taxonomy: install an *implicit-specialization* link from the *Cell* frame to the embedded unit representing "Cell that is part of some plant"; also install the inverse generalization link. Embedded units are referenced (named, pointed to) in KM by specifying the address of the value whose annotations constitute the embedded unit. For example, the embedded unit shown above (consisting of the single value annotation *has-part* = *Cell-Wall*) is referenced by (Plant has-part Cell). To connect this embedded unit into the knowledge-base taxonomy, KM asserts ⟨*Cell implicit-specializations (Plant has-part Cell)*⟩ and ⟨*(Plant has-part Cell) generalizations Cell*⟩.

The result of these two activities, performed automatically by KM, is shown below:

```
Plant
-----
has-part: Cell
          has-part: Cell-Wall
          generalizations: Cell
          part-of: Plant
                   (definitionally-necessary?: T)
                   (definitionally-sufficient?: T)
```

As stated above, embedded units are referenced in KM by giving the address of the value whose annotations constitute the embedded unit. This implementation has the advantage that the defining feature of an embedded unit is implicit in its "name." For example, the name of the embedded unit in the above example, (Plant has-part Cell), can be parsed to yield the definition of the concept it represents, "cell that is part of some plant."

To summarize, KM provides *value annotations* for representing information contextually. Although value annotations do not extend the expressiveness of KM, they have several advantages over conventional frame-based languages. First, they allow information to be represented contextually within the ⟨*frame slot value*⟩ format, as opposed to a rule or constraint language. Second, they provide a more convenient way to represent knowledge in many situations. Rather than creating explicit frames to hold each assertion, some assertions are stored on embedded units as value annotations. In addition, taxonomic and defining information about embedded units is installed automatically by KM rather than by the knowledge enterer. Third, representing knowledge contextually results in a knowledge base that is easier to use, for both people and machines, because related information is bundled together and because important domain concepts are automatically distinguished from those important only in very limited contexts.

2.5 Summary and Limitations

This chapter presents KM, a frame-based knowledge representation language designed to provide increased expressiveness and a more convenient representation of knowledge than existing frame-based languages provide. KM includes three major extensions that collectively distinguish it from traditional languages. The first extension (the use of different semantic types of slots to represent quantified assertions) is purely semantic (a change in the way that triples represented in the language are interpreted). The other two extensions (semantic annotations and value annotations) are changes in both the form and the semantics of the language.

The first extension to KM allows quantified assertions to be represented with the same ease as ground assertions, as simple ⟨*frame slot value*⟩ triples. This is accomplished by overloading slots (in the same sense that operators of a programming language are sometimes overloaded) with different semantics, depending on the frames and values that the slot relates. Different combinations of categories and noncategory instances give rise to different quantificational patterns. Slots that are overloaded in the same way (and that share the same semantic mapping) are grouped into equivalence classes called *semantic types.* When the semantic type of each slot is explicitly represented, a system can automatically determine the semantics of a particular triple. Slot overloading makes knowledge representation more convenient than with conventional frame-based languages, and it allows the representation of several different forms of quantified assertions.

For some situations this technique is not suitable. These are situations in which KM cannot determine automatically the correct semantics of a triple. For example, consider assertions involving the relation *cardinality* between a category and the number of its instances, as in

1. cardinality(Kitchen-chair-at-my-house,4)
   ("There are four kitchen chairs at my house..")

2. cardinality(Kitchen-chair-set-at-Al's-furniture-warehouse,1000)
   ("There are 1,000 sets of kitchen chairs at Al's furniture warehouse.")

3. $\forall x \in$ Kitchen-chair-set-at-Al's-furniture-warehouse, cardinality(x,4)
   ("There are four chairs in each kitchen chair set at Al's.")

Assume the knowledge engineer creates slot *card* to represent the *cardinality* relation and assigns it semantic type 2 so that it can represent assertions having the II quantificational pattern [*e.g.*, (1) and (2)] as well as the AI pattern [*e.g.*, (3)]. Under the approach described here, the above three assertions would be represented as

1. $\langle$*Kitchen-chair-at-my-house card 4*$\rangle$

2. $\langle$*Kitchen-chair-set-at-Al's card 1000*$\rangle$

3. $\langle$*Kitchen-chair-set-at-Al's card 4*$\rangle$

This presents two problems. First, triple (1) would be interpreted not as "There are four kitchen chairs at my house," but as "Each kitchen chair at my house has four instances." This interpretation is incorrect because a particular chair is not a category and thus cannot have instances. KM makes the incorrect interpretation because *Kitchen-chair-at-my-house* is a category and slot *card* is of semantic type 2, meaning that when *card* appears on a category, the category is interpreted as the scope of a universal quantifier. To avoid the error, *card* must be declared semantic type 4, which admits only the II (quantifier-free) interpretation. Declaring *card* to be of semantic type 4, however, makes it unsuitable for representing assertion (3) above.

The second problem is that triples (2) and (3) are contradictory. This occurs because one of the triples is interpreted incorrectly. Declaring *card* to be of semantic type 2 leads to an incorrect interpretation of triple (2), while declaring it as semantic type 4 leads to an incorrect interpretation of triple (3).

This problem occurs whenever the knowledge enterer wants to represent categories whose instances are themselves categories (*e.g., Kitchen-chair-set-at-Al's*) and he wants to represent the same kind of information about both the encompassing categories and their instances (*e.g.,* cardinality). For these situations the knowledge enterer must create separate slots for each of the possible interpretations, such as one slot, *card1*, of semantic type 2 and another slot, *card2*, of semantic type 4. Fortunately, such problematic situations are rare.

A second disadvantage of overloading the semantics of slots is that any inference method that reasons with triples in the knowledge base must be sensitive to the semantic type of slots and the context in which they appear. The reasoner cannot assume that every triple is mapped to logical formulae in the same way. This chapter discusses the modifications required for inheritance and inverse maintenance; other inference methods require similar modifications. Although this requirement makes reasoning more complex, it is more important for a representation language to be easy for people to use, even if this means it is harder for machines to use. The requirement also makes designing inference methods more difficult, but designing a particular method is a one-time cost, while the costs of knowledge representation are incurred every time the language is used. To minimize representation costs, KM allows semantic overloading.

The second extension of KM, *semantic annotations*, provides greater expressiveness by providing constructs for representing both the definitional and assertional components of a description. Definitions are represented using semantic annotations that distinguish between definitionally necessary features and definitionally sufficient features. This distinction allows concepts having partial definitions to be represented. Nondefinitional assertions are also represented using semantic annotations (likelihood, necessity, cue-validity, and uniqueness). By attaching probabilities to these semantic annotations to represent degrees of belief, KM accommodates both defeasible and nondefeasible assertions as well as assertions of graded defeasibility.

This chapter provides a semantics for nondefinitional assertions by specifying a semantic mapping from knowledge-base triples to formulae in probabilistic logic. This approach has two disadvantages. First, mapping each triple to logic *independently* and conjoining the resulting formulae is not suitable for

- slots for which one wants the order of values to be important, and
- slots for which one wants values to be considered collectively. For example, one might want the values of a slot to be interpreted disjunctively rather than conjunctively.

Second, assertions in probabilistic logic are more difficult to reason with than assertions in traditional predicate logic. Abadi and Halpern show that probabilistic logics that include binary predicates are undecidable [1]. As stated before, however, KM was designed for greater expressive power, even at the expense of intractable or undecidable inference methods, because inferential power cannot make up for limited expressiveness.

The third extension of KM is *value annotations* for representing information contextually. Although value annotations do not add expressive power to the language, they have several advantages:

- they make knowledge representation much more convenient,
- they do not require the use of a rule or constraint language,
- information represented with value annotations is just as accessible as the rest of the information in the knowledge base, and
- the resulting knowledge base is easier to inspect and use, because only the most important domain concepts are reified as frames.

The major limitation of value annotations and the implicit specializations they define is that value annotations as described here are useful primarily for defining implicit specializations having a single necessary and sufficient feature. (In the example of the previous section, the single defining feature of the implicit specialization of *Cell* corresponding to "cell of some plant" was "is a part of some plant.")

Additional necessary and sufficient features could be specified *within* an embedded unit, as in the following embedded unit representing the concept "substance that is transported by mineral transport and that contains minerals":

```
Mineral-Transport
-----------------
transportee: Substance
                 contains: Mineral
                           (definitionally-necessary? T)
                           (definitionally-sufficient? T)
```

However, this has two disadvantages. First, the semantic mapping becomes more complex because determining the semantics of a triple requires examining all the value annotations modifying it. For example, consider an extended version of the above example that captures the additional information that the definition of *Mineral-Transport* includes the feature "transports a substance that contains minerals":

```
Mineral-Transport
————————————
transportee: Substance
                 (definitionally-necessary? T)
                 (definitionally-sufficient? T)
                 contains: Mineral
                              (definitionally-sufficient? T)
                              (definitionally-necessary? T)
```

Because the implicit specialization of *Substance* has a defining feature (*contains* = *Mineral*) within the embedded unit, it is no longer possible to determine the semantics of the triple ⟨*Mineral-Transport transportee Substance*⟩ by attending only to that triple and its *semantic* annotations (*definitionally-necessary?* and *definitionally-sufficient?*). All of its *value* annotations must be examined also. If the *contains* = *Mineral* value annotation is ignored, an incorrect definition of *Mineral-Transport* results: "a process that transports some substance (of any kind)." This disadvantage is even more severe considering that value annotations can be nested to any depth.

The second disadvantage of allowing implicit specializations to have more than one defining feature is that such implicit specializations can be represented in multiple ways (as embedded units on different frames). For example, the concept "substance containing minerals that is transported by mineral transport" could be represented as shown above or as an embedded unit on the *Mineral* frame. As a result, implicit specializations could become distributed (multiply defined) in the knowledge base, with partial knowledge of the concept in one embedded unit and partial knowledge in others. In the worst case, an implicit specialization with n defining features could have n different locations. This would lead to inconsistencies in the knowledge base, access problems, and redundant representations.

Turner proposes a more expressive kind of value annotation than those KM uses [81]. Although Turner's approach does not suffer from the first disadvantage above, the problem of distributed representations is much worse. With Turner's value annotations, the representation of an implicit specialization imposes an ordering on the features composing its definition, thus in the worst case an implicit specialization with n defining features could have $n!$ different locations, one for each different ordering of the features.

Despite its limitations, KM has proven very useful for representing fundamental domain knowledge. The Botany Knowledge Base, represented in KM, currently contains over 28,000 facts from college-level botany. The rest of this dissertation describes methods for accessing knowledge represented in a frame-based language such as KM.

Chapter 3

A Content Addressable, Virtual Knowledge Base

To destroy is always the first step in any creation.
e e cummings

This chapter describes methods for making users of a knowledge base less dependent on the particulars of how knowledge is represented. This is done by providing a *content addressable* knowledge base and by providing access to concepts in the *virtual knowledge base.*

3.1 Introduction

While representing knowledge, a knowledge engineer makes numerous decisions, many of them arbitrarily. For example, the choice of what name to give each frame is often arbitrary (*e.g.*, "Plant-Stem" vs. "Stem-of-Plant"). Similarly, the choice of which domain concepts to reify (create a frame for) in the knowledge base depends on the knowledge engineer's subjective judgment of the relative importance of concepts. For instance, the knowledge engineer might create a frame for *Condensation* and a frame for *Water*, but not a frame for *Water-Condensation.* Because relative importance varies from one task to another, decisions the knowledge engineer makes regarding which concepts to reify in a multifunctional knowledge base will not be appropriate for all tasks in all situations. A goal of this research is to insulate users of the knowledge base from the effects of the (sometimes arbitrary) choices made during knowledge representation.

A knowledge-base access method can insulate knowledge-base users from the effects of arbitrary frame-name choices by providing *content addressability.* A content addressable knowledge base allows users to access frames using a partial description of the frame's contents. For example, to access the frame for "plant cell," the user could describe the concept as "cell that is part of a plant." This description could be given in a formal language as (Cell (part-of Plant)). When given this description in place of a frame name, the access method searches the knowledge base for the frame that matches the description. It then uses the name of that frame in servicing the access request. For example, consider a frame-slot access method that provides content addressability. Given the knowledge-base fragment shown in Figure 3.1 and the following frame-slot query (which requests the parts of a cell that is part of a plant):

((Cell (part-of Plant)) has-parts)

the system would

1. Recognize the first item of the address, (Cell (part-of Plant)), as a frame description rather than a frame name.

2. Find the name of the frame matching that description, *Botanical-Cell*.

3. Substitute the frame name for the description to yield the modified frame-slot address, (Botanical-Cell has-parts).

4. Locate (or compute) and return the values of slot *has-parts* on frame *Botanical-Cell*: *Cell-Wall, Protoplast*, etc.

Content addressability can also be used when *storing* slot values. For example, the knowledge engineer can use the address ((Cell (part-of Plant)) has-parts) when asserting what the parts of a plant cell are. Although content addressability supports both querying and updating the knowledge base, all the examples in this chapter are query accesses.

The advantage of content addressability is that users (either people or application programs) can access the knowledge base without extensive prior knowledge of how it has been represented. In particular, users can access concepts without knowing the names of all the frames in the knowledge base. They need only to know the names of the most general frames and slots (the top level of the taxonomy), and they can access other concepts by describing them in terms of more general frames and slots. Thus, users have more flexibility in how they request information from the knowledge base. Application programs that use the knowledge base can pass this flexibility on to their users. For example, a question-answering system that accesses a content addressable knowledge base can accept questions whose topics are descriptions of concepts, rather than frame names. This flexibility is crucial for systems whose users are unfamiliar with the knowledge base, such as students using a tutoring system.

The first way, then, that an access method can insulate users from the effects of knowledge representation decisions is to provide content addressability. The second way is to provide a *virtual knowledge base*. In the actual knowledge base, only concepts and facts that are explicitly represented are accessible. In other words, the only concepts that are accessible are those that are reified as frames, and the only facts that are accessible are those represented by an explicit ⟨*frame slot value*⟩ triple. In a virtual knowledge base, by contrast, concepts and facts that are implicit in the knowledge base are also accessible. That is, the virtual knowledge base consists of all concepts that can be defined in terms of other concepts and

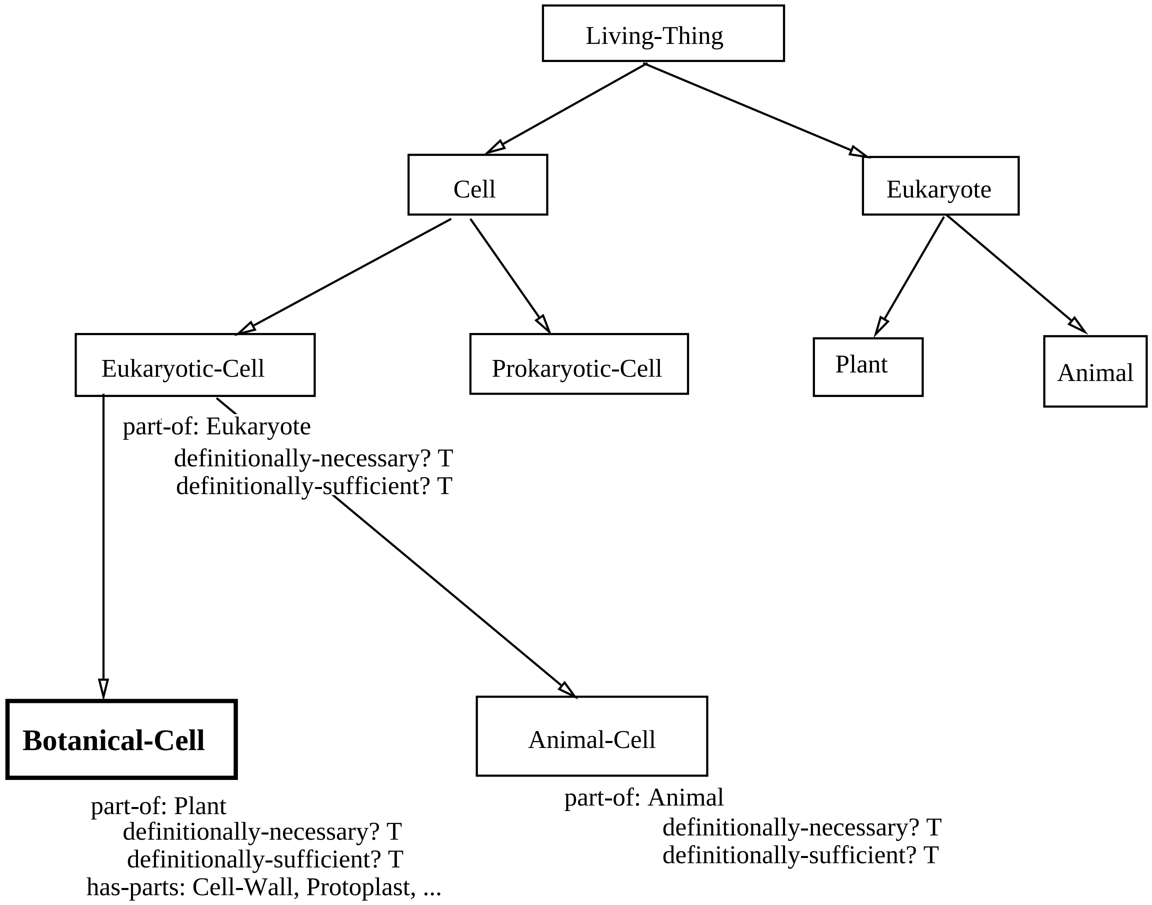

Figure 3.1: Knowledge-base fragment used to illustrate content addressability, shown as a graph. Nodes represent frames, and arcs represent specialization relations. Only the relevant features of each concept are shown. Given the concept description (Cell (part-of Plant)), KASTL matches the description with the concept *Botanical-Cell*.

slots in the knowledge base, and all facts implied by other facts in the knowledge base. Several methods exist for accessing *facts* in the virtual knowledge base (inheritance, rule chaining, etc.). This chapter describes methods for accessing *concepts* in the virtual knowledge base.

A general description of the method for providing access to concepts in the virtual knowledge base is as follows. First, the user supplies a description of the concept, in the same way that content addressable concepts are described. The access method creates a new frame matching the given description and reorganizes the knowledge base to accommodate it. This task involves finding, in the taxonomy, the immediate generalizations and specializations of the new concept and installing links between these frames and the new frame. (These links allow the new frame to participate in inheritance.) It also involves recording on the new frame any known information about the new concept, including the information given in the input description. Finally, the access method uses the name of the new frame in servicing the access request.

As an example, consider a frame-slot access method that provides access to concepts in the virtual knowledge base. Given the hypothetical knowledge-base fragment shown in the top portion of Figure 3.2 and the following frame-slot query (which requests the site of origin of oxygen that is the end-product of photosynthesis):

((Oxygen (end-product-of Photosynthesis)) site-of-origin)

the system would

1. Recognize the first item, (Oxygen (end-product-of Photosynthesis)), as a frame description rather than a frame name.

2. Create a new frame to represent the specified concept. Unless a frame name is specified by the user, the system gives the frame a machine-generated name (*e.g.*, $Oxygen'$).

3. Find the most specific concepts that are more general than the user-defined concept: *Product-of-Photosynthesis* and *Biologically-Produced-Oxygen*. These can occur in the knowledge base either as frames or as embedded units.

4. Find the most general concepts that are more specific than the user-defined concept: *Oxygen-of-Leaf-Photosynthesis*. Again, these can occur in the knowledge base either as frames or as embedded units.

5. Install generalization links from $Oxygen'$ to *Product-of-Photosynthesis* and *Biologically-Produced-Oxygen*, and install specialization links from $Oxygen'$ to *Oxygen-of-Leaf-Photosynthesis*. Remove redundant links. The result of this knowledge base reorganization is shown at the bottom of Figure 3.2.

6. Record on $Oxygen'$ information about the new concept given by the input specification. Install *end-product-of* = *Photosynthesis* with semantic annotations *definitionally-necessary?* = T and *definitionally-sufficient?* = T.

7. Locate (or compute) and return the value(s) of slot *site-of-origin* on the new frame *Oxygen$'$*: *Photosynthetic-Cell*, inherited from *Product-of-Photosynthesis.*

In this way, the access method would return *Photosynthetic-Cell* as the response to the above query.

If users have access only to concepts that are explicitly represented in the knowledge base, then the knowledge engineer's decision not to reify a concept that is important for a particular task limits the user's ability to perform that task. For example, if a question-answering system has access only to the actual knowledge base, then that system can generate answers only to questions about concepts that have been explicitly represented. By providing access to concepts in the virtual knowledge base, an access method makes users less vulnerable to the particulars of how knowledge is represented. For example, a question-answering system could describe how a decrease in the amount of water in the soil surrounding a plant affects plant growth, even if the knowledge base contains no frame corresponding to "water in the soil surrounding a plant." In addition to the practical advantages, providing a virtual knowledge base also has psychological validity. Barsalou's experiments indicate that people readily construct *ad hoc categories* (categories not well established in memory) for use in specialized contexts [5].

The crucial step of providing content addressability and of providing access to concepts in the virtual knowledge base is determining, for each concept in the knowledge base, whether the input description matches it exactly, is more general than it, is more specific than it, or is neither more general nor more specific. This step is called *subsumption* [86, 85]. Performing subsumption requires that the input description provide a complete definition of the concept to be accessed, one containing definitionally necessary and definitionally sufficient criteria that completely delineate the concept's intension. (It is not possible to determine whether two *partial* definitions describe exactly the same concept.) This implies that, although any concept in the knowledge base can be identified as a *potential* match for a given description, only completely defined concepts can be *uniquely* identified by description (*i.e.*, identified as the only possible match for the description). Fortunately, concepts that can be only partially defined usually have standard names, such as "water" or "photosynthesis." Section 3.3.3 describes how KASTL allows users to access partially defined concepts by description by identifying potential matches for a given description.

The fact that it is not possible, in general, to determine the subsumption relationship between two partial definitions also implies that a concept is in the virtual knowledge base only if it can be completely defined in terms of other concepts and relations in the knowledge base (either the actual knowledge base or the virtual knowledge base). For example, if the concepts *Eukaryotic-Cell* and *Cytoplasm* and the relation *part-of* are reified in the knowledge base, then the concept "Cytoplasm of a Eukaryotic Cell" is in the virtual knowledge base. Although only concepts that can be completely defined in terms of other concepts are in the virtual knowledge base, such concepts appear to occur frequently enough to make developing methods for accessing them worthwhile. In an analysis of a chapter from a biology textbook, of the 899 concepts referenced in 55 paragraphs, approximately 29% of them referred to

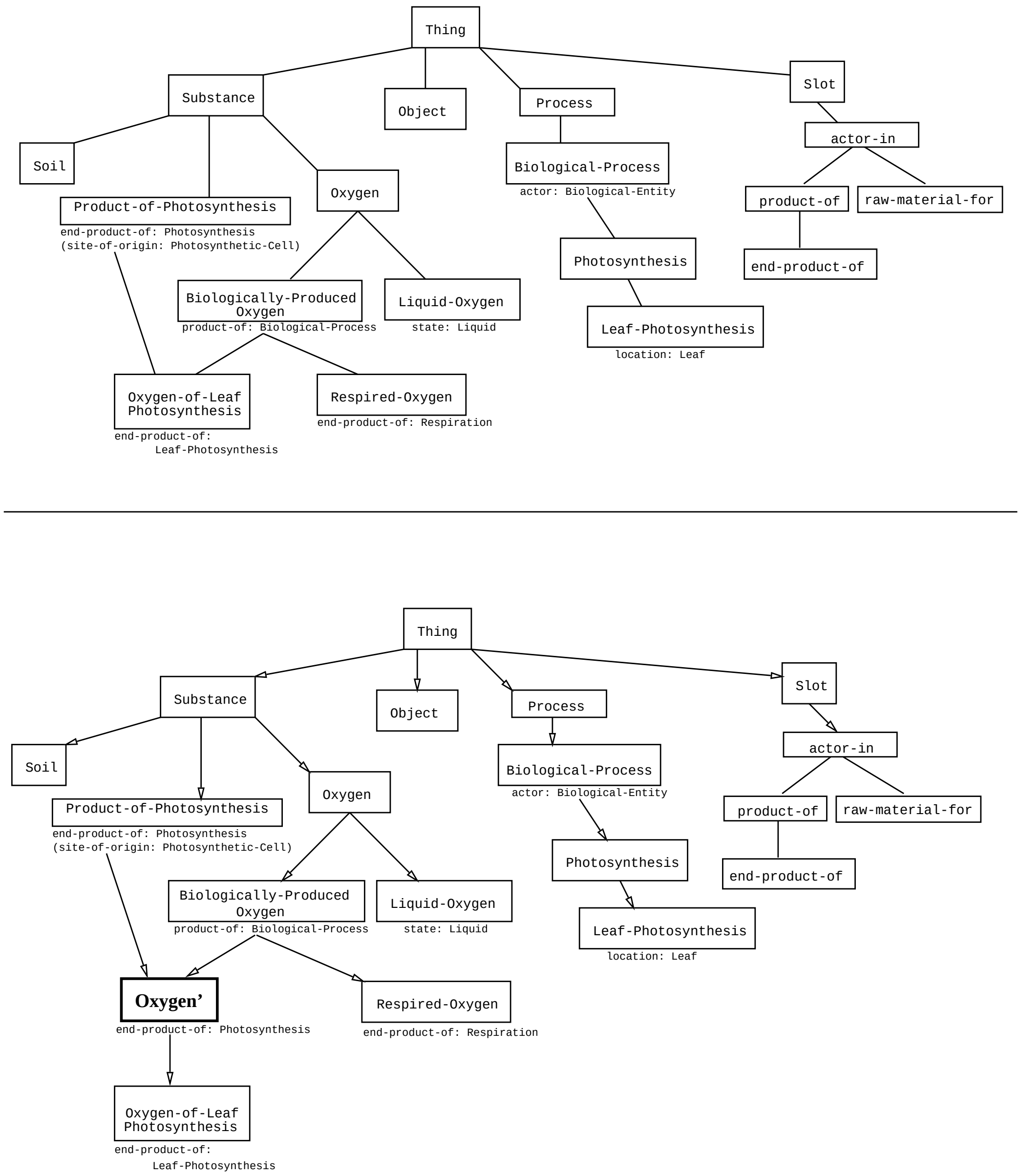

Figure 3.2: Knowledge-base fragments before and after reorganization to accommodate the new concept *Oxygen*′, described by (Oxygen (end-product-of Photosynthesis)). Only the relevant portions are shown. All features shown are definitionally necessary and sufficient, with the exception of those shown in parentheses.

concepts that could be completely defined.

When an access method provides both content addressability and access to concepts in the virtual knowledge base, users need not know whether concepts are explicit in the knowledge base. Users simply supply a description of the concept, embedded in an access request. If the concept has a frame associated with it, then the system will find and use that frame. Otherwise, the system will create and use a new frame. From the user's point of view, there is no distinction between accessing concepts by description and accessing concepts in the virtual knowledge base. In terms of the system architecture given in Chapter 1, the Finder and the Creator share a single user interface. For this reason, and because the computations of the Finder and the Creator overlap significantly, this chapter combines the discussions of content addressability and providing a virtual knowledge base. The next section discusses the related work on these topics, and the following section describes the approach taken here. The chapter concludes with some examples taken from the Botany Knowledge Base and with an empirical cost analysis.

3.2 Related Work

Providing access to concepts in the virtual knowledge base is essentially an *automatic classification* task. Automatic classification involves inserting a new concept into a taxonomy so that it is directly linked to the most specific concepts that subsume it and to the most general concepts that it subsumes [86]. Automatic classification originated with KL-ONE [12], and most of KL-ONE's successors, including KRYPTON [11], include classifiers.

As implemented in KL-ONE, KRYPTON, and their descendants, automatic classification has several limitations. First, many of these languages, including KRYPTON, KANDOR, and CLASSIC, limit expressive power in an effort to achieve tractable subsumption algorithms [86]. (Recall that subsumption is the step of classification that compares two concept descriptions to determine their relationship. There is a tradeoff between the expressiveness of a representation language and the complexity of computing subsumption for descriptions represented in that language [44, 86, 71, 15, 66, 58].) The philosophy of sacrificing expressiveness for tractable subsumption is still widely embraced, as evidenced by the statement in [71]: "a (representation) formalism with an undecidable subsumption is unsatisfactory." As noted in Chapter 2, however, this approach results in languages so limited that they are no longer generally useful. KRYPTON, one of the few languages (if not the only language) to achieve a tractable subsumption algorithm, never found its way into applications, partially because of its limited expressiveness [86].

A second limitation of traditional classifiers is that they use ill-characterized subsumption algorithms. These systems use the following criterion for subsumption [12]:

> A concept X subsumes a concept Y if and only if, in all possible interpretations, the extension of X is a superset of the extension of Y.

Woods calls this definition of subsumption *extensional subsumption* [85]. This criterion for subsumption has been found to lead to intractability for most representation languages, even for most of the languages that limit expressiveness [86, 58]. As a result, most classifiers have retreated to tractable but incomplete subsumption algorithms [66]. These algorithms are sound with respect to the above subsumption criterion, but they lack a precise specification of what subsumption relationships they detect (their degree of completeness). This is surprising given the strong KL-ONE tradition of grounding the representational system in formal semantics. (A notable exception is Patel-Schneider's approximate account of the subsumptions that the NIKL classifier detects [66].) An alternative approach is to define a new criterion for subsumption, one that has a more tractable computation. A classification algorithm based on such a criterion would have a precise account of what the algorithm computes, without sacrificing expressiveness of the language.

The third limitation of traditional classifiers is that they are based on the extensional subsumption criterion, but they are restricted to using only terminological (*i.e.*, definitional) knowledge, knowledge that carries no assertional import [12, 11]. Extensional subsumption cannot always be computed solely from terminological or definitional knowledge (such as the information in KRYPTON's TBox [11]). For example, if "Triangle" is defined as "Polygon with three angles," determining that "Polygon with three or more sides" subsumes "Triangle" requires the fact that "every angle of a polygon has a corresponding side," knowledge that is strictly assertional rather than definitional (and hence would appear in KRYPTON's ABox rather than its TBox).

If subsumption is to be computed using only definitional knowledge, a new criterion for subsumption must be used, one that is based on concept intensions rather than extensions. Woods introduces such a criterion, called *intensional subsumption* [85]. Intensional subsumption means that the definition (intension) of the subsuming concept is more general than the definition of the subsumed concept. (Definition $D1$ is more general than definition $D2$ when every definitionally sufficient feature of $D1$ is definitionally necessary for $D2$ or generalizes some feature that is definitionally necessary for $D2$.) For example, under intensional subsumption, "Person whose children are professionals" subsumes "Woman whose children are doctors" (assuming that "Woman" is defined as a kind of "Person" and "Doctor" is defined as a kind of "Professional"), but "Polygon with three or more sides" does not subsume "Polygon with three angles."

To overcome the limitations of traditional classifiers, KASTL's classification algorithm is based on the criterion of intensional subsumption rather than extensional subsumption. Although Woods proposes intensional subsumption as an alternative to extensional subsumption, he retains extensional subsumption as the criterion of completeness. That is, Woods says that intensional subsumption should entail extensional subsumption, and that, all else being equal, it is desirable to be as complete as possible with respect to extensional subsumption. This work, by contrast, rejects extensional subsumption in favor of intensional subsumption because extensional subsumption has the undesirable property that two concepts that have empty extensions in all possible worlds, such as "Round square" and "Colorless green idea," are considered to subsume one another (*i.e.*, to be equivalent). Under intensional subsumption, concepts are equivalent only when they have identical intensions.

The final limitation of traditional classifiers that this work addresses is that most classifiers were designed to accompany representation languages less expressive than KM. In particular, KM allows the knowledge enterer to represent necessary features separately from sufficient features. It also allows definitional and nondefinitional assertions to be represented with the same constructs. In the KL-ONE family of languages, by contrast, definitional features are usually interpreted as both necessary and sufficient [85], and in most of these languages nondefinitional assertions are either not representable or are represented separately from definitions, as in KRYPTON's ABox [11]. KASTL's classification and subsumption algorithms were designed to accommodate the increased expressiveness KM offers.

3.3 The Approach

This section describes the methods KASTL uses to provide a content addressable, virtual knowledge base. The first two subsections describe separately the tasks of accessing concepts by description and accessing concepts in the virtual knowledge base. The third subsection describes how these tasks are integrated in a single module of KASTL. The section concludes with examples of system performance from the Botany Knowledge Base.

3.3.1 Accessing Concepts by Description (Content Addressability)

The task of accessing concepts by description can be described informally as "given a description of a concept, find the knowledge-base frame that describes the concept." A more precise formulation of the task as performed by KASTL is given below:

Given: A concept description, in a formal language, consisting of

- a base concept B (more precisely, the name of the frame representing B), and
- a set F of features (*i.e.*, slot-value pairs),

Return: The name of the frame representing the concept C that matches the description. C matches the description if and only if

- C is a specialization of B (although not necessarily an immediate specialization),
- All features in F are definitionally necessary for membership in C, and
- The features in F are (jointly) definitionally sufficient for membership in C for all members of B.

For example, in the concept description (Cell (part-of Plant)), the base concept B is *Cell* and the only feature in F is *part-of* = *Plant*. The task to be performed is to find a frame representing a specialization of *Cell* for which the feature *part-of* = *Plant* is definitionally necessary and sufficient. In other words, the task is to find the frame representing the concept whose definition is "cell that is part of a plant."

Although this example consists of a simple concept description, KASTL also accommodates more complex descriptions. Figure 3.3 shows the grammar for concept descriptions. This *concept specification language* has the same syntax as frames in KM so that concepts can be described in the same way that they are represented.

Concept descriptions may have multiple features, as in

(Cell (part-of Plant) (producer-in Photosynthesis)).

Multiple features are interpreted conjunctively; *all* of the specified features must be annotated as definitionally necessary and sufficient on the matching concept. Thus, the above example describes "photosynthetic plant cell."

The concept specification language is recursive. In other words, the language allows nested concept descriptions. For example, the description

(Water (transportee-in (Diffusion (source Soil-Region) (destination Plant))

describes "water transported by diffusion from the soil into a plant." KASTL matches nested descriptions from the inside out. For example, KASTL first searches for a frame matching (Diffusion (source Soil-Region) (destination Plant)), such as *Plant-Water-Uptake*. It then substitutes that frame name for the nested description to yield

(Water (transportee-in Plant-Water-Uptake)).

Finally, KASTL searches for a frame matching the modified description.

In addition to concepts, the concept specification language in Figure 3.3 also allows users to access *slots* by description. A list of slots in place of a slot name refers to the disjunction of those slots. For example, (*husband wife*) would refer to the slot *spouse*. Users can also refer to the transitive closure (Kleene star) of a slot. For example, (transitive-closure-of *parent*) would refer to the slot *ancestor*.

Figure 3.4 gives the procedure KASTL uses to provide content addressability. The first step is to ensure that the given concept description is meaningful. This involves checking that each frame name and slot name in the description exists and that each feature is valid. A feature *slot* = *value* is valid when *value* is in the *range* of *slot*. For example, the description (Glucose (product-of Photosynthetic-Cell)) is not valid if slot *product-of* has range *Process*, because *Photosynthetic-Cell* is not a *Process*.

The next step is an efficiency measure. To reduce the amount of search, KASTL converts the base concept within the given concept description to a more specific concept if the conversion does not change the meaning of the description. For example, given the concept description (Object (parent-in Sexual-Reproduction), "an object that reproduces sexually,"

```
<Concept-description> ::= ( <Concept> <Features> )    |    ( <Concepts> )

<Concept>  ::= <Frame-name>  |     <Concept-description>

<Concepts> ::= <Concept>     |     <Concept> <Concepts>

<Features> ::= <Feature>     |     <Feature>  <Features>

<Feature>  ::= ( <Slot>  <Facet-list>  <Value> )

<Slot> ::=  <Slot-name>          |
    ( <Slot-list> )           |          ; disjunction of slots
    ( transitive-closure-of   <Slot> )   ; Kleene star

<Slot-list> ::= <Slot-name>   |  <Slot-name>   <Slot-list>

<Facet>     ::= <Facet-name>  |  ( <Facet-list> )

<Facet-list> ::= <Facet> <Facet-list>   |    epsilon

<Value>      ::= <Concept>              |   <kb-constant>
```

Figure 3.3: The concept specification language for describing concepts to KASTL. The same language is used both to access concepts by description and to access concepts in the virtual knowledge base. (The *facets* shown above are a representational construct of KM used to annotate frame-slots independently of their values. Although KASTL supports facets, they are rarely used in the Botany Knowledge Base and are not discussed here.)

1. Insure that the given concept description is meaningful. Each frame name and slot name must exist, and each feature must be valid for the specified base concept.

2. Convert the base concept B to a more specific concept, if possible, by examining the features given in the description and the constraints the knowledge base contains.

3. If the base concept B matches the input description, return B.

4. Otherwise, examine each immediate specialization S of B.

   - If S matches the input description, return S.
   - If S is more general than the input description, then repeat step 4 for all specializations of S.

5. If no match is found for the given concept description, then reify the concept from the virtual knowledge base and reorganize the taxonomy to include it.

Figure 3.4: Procedure KASTL uses to provide content addressability.

KASTL modifies the description to (Living-Thing (parent-in Sexual-Reproduction)). Changing the base concept from *Object* to *Living-Thing* does not change the meaning of the description because KASTL determines from examining the knowledge base that only a living thing can reproduce (*i.e.*, the *domain* of slot *parent-in* is *Living-Thing*). This modification greatly simplifies the search for a match; specializations of *Object* that are not specializations of *Living-Thing* need not be examined.

The third step of the procedure is to determine whether the (possibly modified) base concept matches the given description. For example, KASTL determines whether the *Living-Thing* frame matches the description (Living-Thing (parent-in Sexual-Reproduction)). A frame matches a description if every feature annotated as definitionally necessary or definitionally sufficient on that frame is present in the description and every feature in the description is present on the frame and is annotated as both definitionally necessary and sufficient. For example, for the *Living-Thing* frame to match the above description *parent-in =Sexual-Reproduction* must be represented as definitionally necessary and sufficient for *Living-Thing* and *Living-Thing* must have no other definitional features. In this example, the match fails, so KASTL proceeds to the next step.

Step 4 of the procedure is to examine each immediate specialization of the base concept for a match. (KASTL examines both explicit specializations and implicit specializations represented by embedded units.) If a match is found, it is returned. Otherwise, KASTL repeats step 4 for each specialization whose definition is more general than the given concept description. A specialization S is more general than the concept description if S has at least one definitionally sufficient feature (*i.e.*, it has a definition), and each such feature appears in the concept description or generalizes some feature in the concept description. If no specialization of the base concept is more general than the given description, KASTL fails to find an exact match.

The knowledge-base fragment shown in Figure 3.5 can be used to illustrate step 4 for the concept description (Living-Thing (parent-in Sexual-Reproduction)). The search for a match is restricted to the portion of the taxonomy rooted at *Living-Thing*. KASTL first attempts to match each specialization of *Living-Thing* with the given description. Neither *Reproducing-Structure* nor *Nonreproducing-Structure* is an exact match, but *Reproducing-Structure* is more general than the concept description because the feature *parent-in* = *Reproduction* on *Reproducing-Structure* subsumes the feature *parent-in* = *Sexual-Reproduction* in the input description. Therefore KASTL repeats the matching process for the specializations of *Reproducing-Structure*. On this iteration a match is found, *Sexually-Reproducing-Organism*.

If KASTL fails to find an exact match for a given concept description, the described concept is not explicitly represented in the knowledge base, either as a frame or as an embedded unit. Although the concept does not exist explicitly in the knowledge base, it does exist in the virtual knowledge base, because it can be completely defined in terms of other concepts and relations (as evidenced by the given description). Thus, KASTL can still access the concept by reifying it using the method described in the next subsection.

3.3.2 Accessing Concepts in the Virtual Knowledge Base

The task of accessing concepts in the virtual knowledge base can be described informally as "given a description of a concept, modify the knowledge base to include that concept." A more precise formulation of the task as performed by KASTL is shown below:

Given: A concept description consisting of

- A base concept, and
- A set of features (*i.e.*, slot-value pairs),

Return:

- The name of the frame created to represent the described concept, and
- A new knowledge base reorganized to accommodate the new frame.

For example, given the concept description (Oxygen (end-product-of Photosynthesis)), the task is to create a frame whose definition is "Portion of oxygen produced by photosynthesis" and to modify the taxonomy to include that frame.

The same concept specification language used to provide content addressability, shown in Figure 3.3, is also used to provide access to concepts in the virtual knowledge base. The

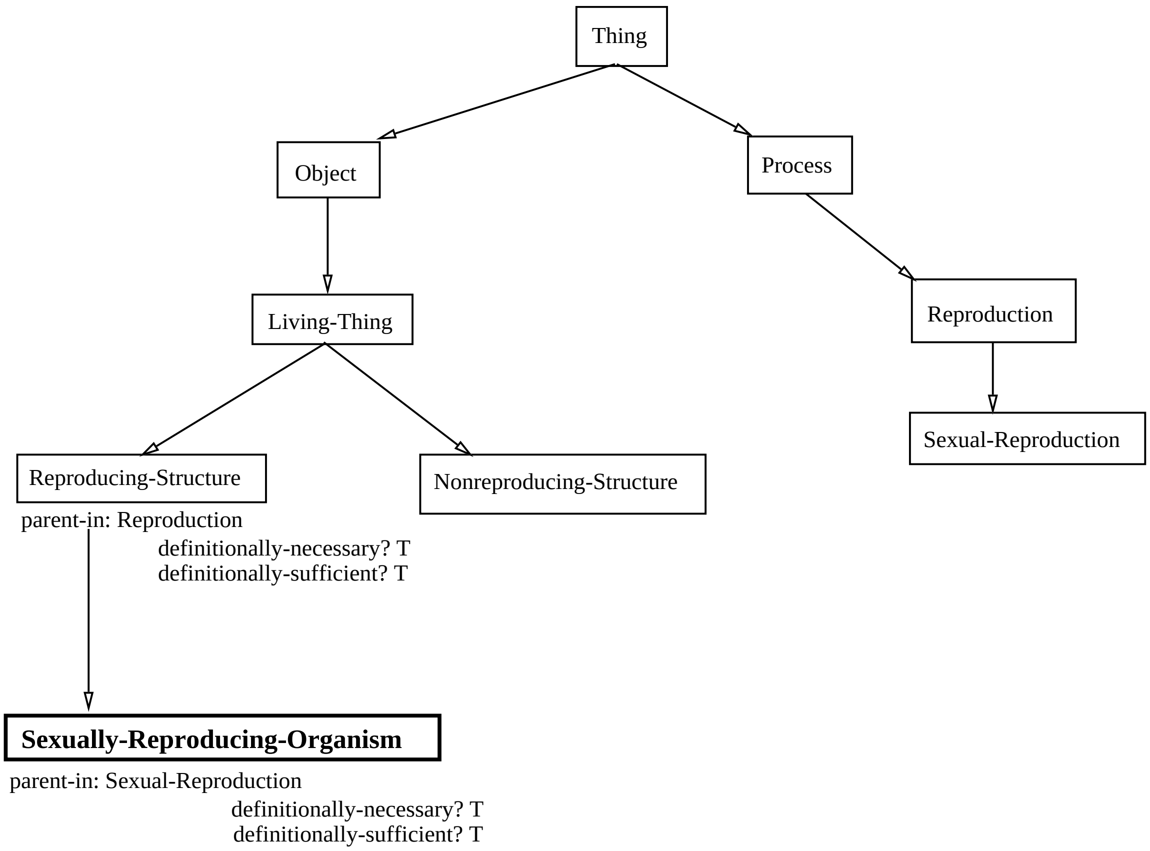

Figure 3.5: Knowledge base fragment used to illustrate content addressability for *Sexually-Reproducing-Organism*, described by (Living-Thing (parent-in Sexual-Reproduction)). Only the relevant features of each concept are shown.

1. Find immediate generalizations of the new concept. Recall the failure points of the search for a match. (Figure 3.4 gives the algorithm for this search.) These failure points are the most specific concepts that are more specific than the base concept but more general than the new concept.
2. Find additional immediate generalizations of the new concept. Find the most specific concepts that are both
   - neither more general nor more specific than the base concept, and
   - more general than the given description.
3. Find the immediate specializations of the new concept. Choose one of the generalizations G found in step 1 or 2, and find the most general concepts that are specializations of G and that are more specific than the given description.
4. Create a new frame.
   (a) Install generalization relations to concepts found in steps 1 and 2.
   (b) Install specialization relations to concepts found in step 3.
   (c) Assert on the new frame each feature given in the input description.
   (d) Remove redundant taxonomic relations.
5. Infer new features for the new concept (optional).

Figure 3.6: Procedure KASTL uses to access concepts in the virtual knowledge base.

semantics of the language is the same for both uses; the base concept is a generalization of the described concept, features modifying the base concept are jointly necessary and sufficient, and nested descriptions are matched or created from the inside out. Using the same concept specification language for both content addressability and accessing concepts in the virtual knowledge base allows a single user interface. In this way, users do not need to know whether they are accessing existing concepts or virtual concepts. (The next subsection discusses this in more detail.)

Figure 3.6 gives the procedure KASTL uses to access concepts in the virtual knowledge base. The first two steps of the procedure find the immediate generalizations of the concept to be created. This involves finding the most specific concepts in the knowledge base that subsume the given concept description. (Recall that a concept in the knowledge base subsumes the given description if it has at least one definitionally sufficient feature, and each such feature appears in the description or generalizes some feature in the description.) Step 1 takes advantage of the fact that KASTL creates a new concept only after having failed to find the concept in the knowledge base. If KASTL fails to find an exact match for the given concept description, it terminates its search after encountering the most specific specializations of the base concept that are more general than the given concept description. These concepts will be immediate generalizations of the concept to be created. Thus, KASTL avoids repeating the search for these generalizations by recording the failure points of the search for a match.

For example, consider the concept description (Oxygen (end-product-of Photosynthesis)) and the knowledge-base fragment shown in the top portion of Figure 3.2. When KASTL searches the knowledge base for a concept matching the description, it begins at the base concept, *Oxygen*. One specialization of *Oxygen*, *Liquid-Oxygen*, neither matches nor is more general than the given description, so it is not pursued. The other specialization, *Biologically-Produced-Oxygen*, is more general than the given description (because *product-of* is more general than *end-product-of* and *Biological-Process* is more general than *Photosynthesis*), so KASTL repeats the search from that concept. At this point, none of the specializations of *Biologically-Produced-Oxygen* can be pursued. (None of them matches or subsumes the given concept description.) Thus, the only failure point is *Biologically-Produced-Oxygen*. After failing to find a match for the specified concept, KASTL proceeds to create it. KASTL recalls the single failure point, *Biologically-Produced-Oxygen*, to be installed as an immediate generalization of the new concept. *Biologically-Produced-Oxygen* is guaranteed to be the only immediate generalization in the portion of the taxonomy rooted at the base concept, *Oxygen*.

Step 2 of the procedure finds the generalizations of the new concept that are *outside* the portion of the taxonomy rooted at the base concept. Although the search for an exact match for a concept description can be limited to the portion of the taxonomy rooted at the base concept, the search for generalizations of the described concept must originate at the root of the taxonomy. Because KASTL is searching for maximally *specific* generalizations only, it need not examine ancestors of the base concept. (The base concept subsumes the described concept by construction, and it is more specific than any of its generalizations.) Specializations of ancestors of the base concept, however, must be examined. For example, although the ancestors of *Oxygen* (*Substance* and *Thing*) will not be maximally specific generalizations of the concept described by (Oxygen (end-product-of Photosynthesis)), the immediate specializations of *Substance* and *Thing* (*Soil, Product-of-Photosynthesis, Object, Process*, and *Slot*) must be examined. If any of these concepts subsumes the given description, then KASTL must also examine its specializations. (KASTL examines both explicit specializations and implicit specializations represented by embedded units.)

KASTL continues searching the taxonomy in this way until it finds no more concepts that subsume the input description. The most specific concepts found will be immediate generalizations of the newly created concept. For example, although none of *Soil, Object, Process*, or *Slot* subsumes the concept described by (Oxygen (end-product-of Photosynthesis)), *Product-of-Photosynthesis* does subsume it. Because *Product-of-Photosynthesis* has no specializations, it is a maximally specific generalization of the given description, and KASTL will install it as an immediate generalization of the new concept.

Step 3 of the procedure is to find the immediate specializations of the concept to be created. This involves finding the most general concepts in the knowledge base that the given concept description subsumes. (The given description subsumes a concept in the knowledge base if every feature in the description is definitionally necessary for the concept or generalizes some feature that is definitionally necessary.) The search for specializations can originate with any of the generalizations found in steps 1 and 2, preferably the one with the fewest specializations. KASTL chooses a starting concept and examines its immediate specializations. If a concept is subsumed by the given concept description, then KASTL retains it as a maximally general specialization. (Its specializations need not be examined because KASTL is searching for the maximally *general* specializations.) If a concept is *not* subsumed by the given description, then KASTL must also examine all of its specializations. In the worst case, the search continues to the most specific concepts in that portion of the taxonomy. The most general specializations found will be immediate specializations of the new concept.

Recall that for the hypothetical knowledge-base fragment shown in the bottom portion of Figure 3.2, the most specific generalizations of the concept description
(Oxygen (end-product-of Photosynthesis)) are *Product-of-Photosynthesis* and *Biologically-Produced-Oxygen.* Assume that KASTL chooses *Biologically-Produced-Oxygen* as the starting point of the search for specializations. The first specialization of *Biologically-Produced-Oxygen*, *Oxygen-of-Leaf-Photosynthesis*, is subsumed by the given description because *Leaf-Photosynthesis* is a specialization of *Photosynthesis*. Thus, *Oxygen-of-Leaf-Photosynthesis* is a maximally general specialization and its specializations (if it had any) would not be examined. The second specialization of *Biologically-Produced-Oxygen*, *Respired-Oxygen*, is not subsumed by the given description, thus KASTL must also examine its specializations. In this example, *Respired-Oxygen* has no specializations, so the search terminates. The only specialization found is *Oxygen-of-Leaf-Photosynthesis*, which will be installed as an immediate specialization of the new concept.

Step 4 of the procedure is to create a frame to represent the new concept (*e.g.*, $Oxygen'$) and reorganize the knowledge base to accommodate it. Reorganization involves installing generalization relations from the new frame to the frames found in steps 1 and 2, installing specialization relations from the new frame to the frames found in step 3, and installing each feature given in the concept description on the new frame as a definitionally necessary and sufficient feature. It also involves removing taxonomic relations that become redundant after installing the new taxonomic relations. The bottom portion of Figure 3.2 shows the result of this reorganization for the knowledge-base fragment shown in the top portion of Figure 3.2, following the creation of a new frame for the concept described by (Oxygen (end-product-of Photosynthesis)). The specialization relation from *Biologically-Produced-Oxygen* to *Oxygen-of-Leaf-Photosynthesis* is removed because it is redundant given the new specialization relations from *Biologically-Produced-Oxygen* to $Oxygen'$ and from $Oxygen'$ to *Oxygen-of-Leaf-Photosynthesis*. Similarly, the redundant specialization relation from *Product-of-Photosynthesis* to *Oxygen-of-Leaf-Photosynthesis* is removed.

The final step in reifying concepts in the virtual knowledge base is to infer new information about the concept and install it on the new frame. This can be done using standard inference methods such as induction from specializations or instances, deduction from rules, or inheritance. This step can be done as the system installs the new frame in the knowledge base, or it can be done on demand as users request slot values. KASTL takes the latter approach. With KASTL, users can request slot values directly, or they can request them through requests for *viewpoints* of the new concept. The next chapter describes the methods KASTL uses for accessing viewpoints of concepts.

3.3.3 Combining Content Addressability with a Virtual Knowledge Base

This section has described two tasks, accessing concepts by description and accessing concepts in the virtual knowledge base. These tasks are performed by a single module of KASTL with a single user interface. Combining the tasks has two advantages. First, the procedure that reifies concepts in the virtual knowledge base can use information gathered while attempting to find a concept in the actual knowledge base (step 1 of Figure 3.6). When both procedures are executed, combining them makes the system more efficient. The second advantage is that users of KASTL do not need to know whether the concept they want to access exists in the actual knowledge base. They simply provide a description of the concept. If KASTL fails to find a frame representing that concept, it automatically creates one. Users do not need to know or specify whether they are accessing existing concepts by description or accessing concepts in the virtual knowledge base.

If users want to access concepts by description without accessing concepts in the virtual knowledge base, they can operate KASTL in the *recognition-only* mode. In the recognition-only mode, KASTL attempts only to find the described concept in the actual knowledge base and does not create a new frame. If KASTL finds no match, then rather than create a new frame, KASTL returns the list of concepts in the knowledge base that are more specific than the described concept. The user can then select from this list of partial matches the concept that is most appropriate for the task at hand. For example, for the knowledge-base fragment shown in the top portion of Figure 3.2 and the concept description

(Oxygen (end-product-of Photosynthesis)),

KASTL (operating in recognition-only mode), upon failing to find an exact match, would return the list (*Oxygen-of-Leaf-Photosynthesis*). This facility allows users to locate a concept through a description more general than the one that matches the concept exactly. More general descriptions are more convenient for users to provide because they require less prior knowledge of the frames and slots in the knowledge base. This facility also allows users to access by description concepts that cannot be completely described (because they lack complete definitions).

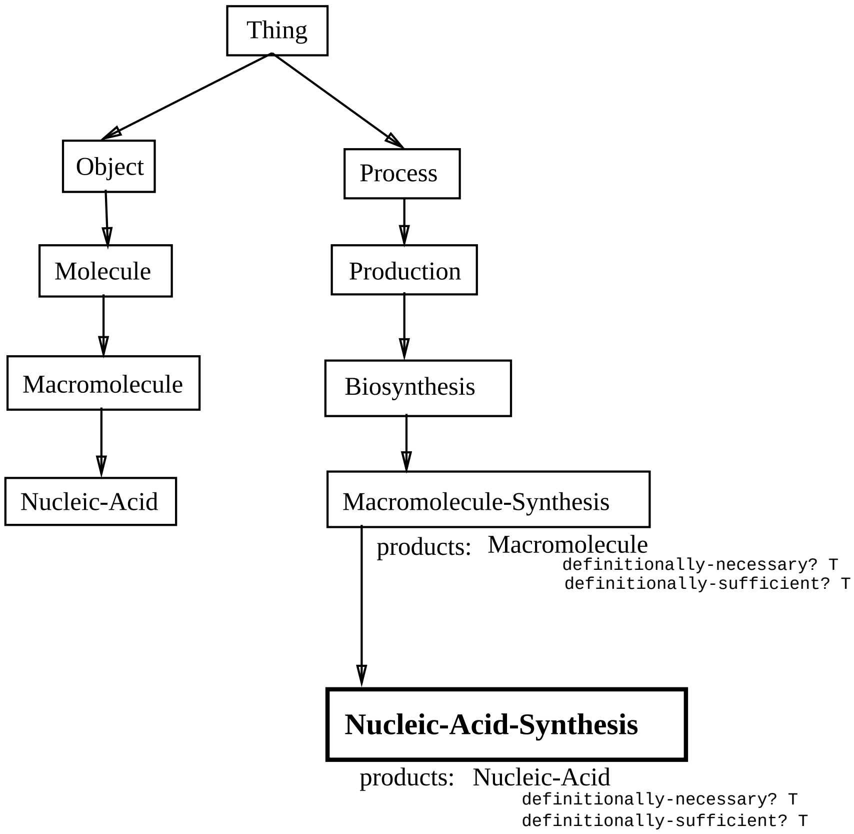

Figure 3.7: Knowledge-base fragment used to illustrate content addressability for the concept description (Biosynthesis (products Nucleic-Acid)). KASTL returns *Nucleic-Acid-Synthesis* as a match.

3.3.4 Further Examples

The example used to illustrate accessing concepts in the virtual knowledge base was hypothetical, specially constructed to illustrate all aspects of the algorithm. Following is a sample of actual results KASTL produced from the Botany Knowledge Base.

Content Addressability Examples

- Given (Biosynthesis (products Nucleic-Acid)), which describes "Biosynthesis process that produces a nucleic acid," and the knowledge-base fragment shown in Figure 3.7, KASTL returns *Nucleic-Acid-Synthesis.*

- Given (Process (raw-materials Water)), which describes "Process that consumes water," and the knowledge-base fragment shown in Figure 3.8, KASTL, in *recognition-only*

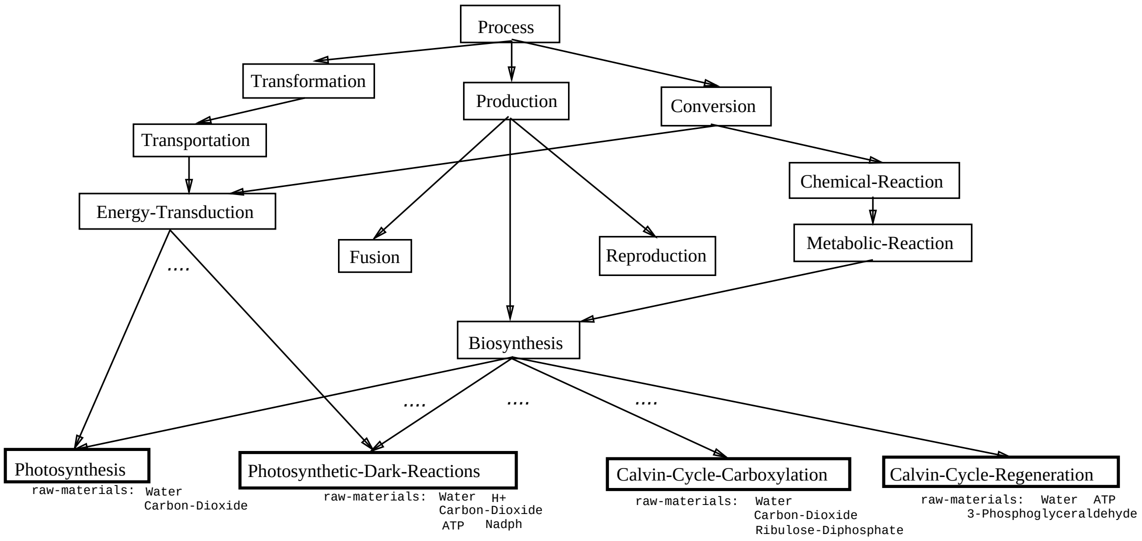

Figure 3.8: Knowledge-base fragment used to illustrate content addressability for the description (Process (raw-materials Water)). KASTL, in *recognition-only* mode, returns the list (Photosynthesis Photosynthetic-Dark-Reactions Calvin-Cycle-Carboxylation Calvin-Cycle-Regeneration).

mode, returns the list

(Photosynthesis Photosynthetic-Dark-Reactions Calvin-Cycle-Carboxylation Calvin-Cycle-Regeneration).

- Given (Energy-Transduction (input-energy-form (Energy (energy-holder ATP)))), which describes "Energy transduction process whose input energy is held by ATP," and the knowledge-base fragment shown in Figure 3.9, KASTL, in *recognition-only* mode, returns the list (Photosynthetic-Dark-Reactions). This example illustrates an embedded concept description.

Virtual Knowledge Base Access Examples

- Given (Plant (producer-in Plant-Photosynthesis)), which describes "Plant that is photosynthetic," and the knowledge-base fragment shown in the top portion of Figure 3.10, KASTL modifies the knowledge base as shown in the bottom portion of Figure 3.10. (In this example, the user has specified the name to be given to the new frame, *Photosynthetic-Plant*.) This example illustrates KASTL's treatment of embedded units [*e.g.*, the embedded unit referred to by the address

(Seedling-Emergence developee Seedling after-state Seedling)].

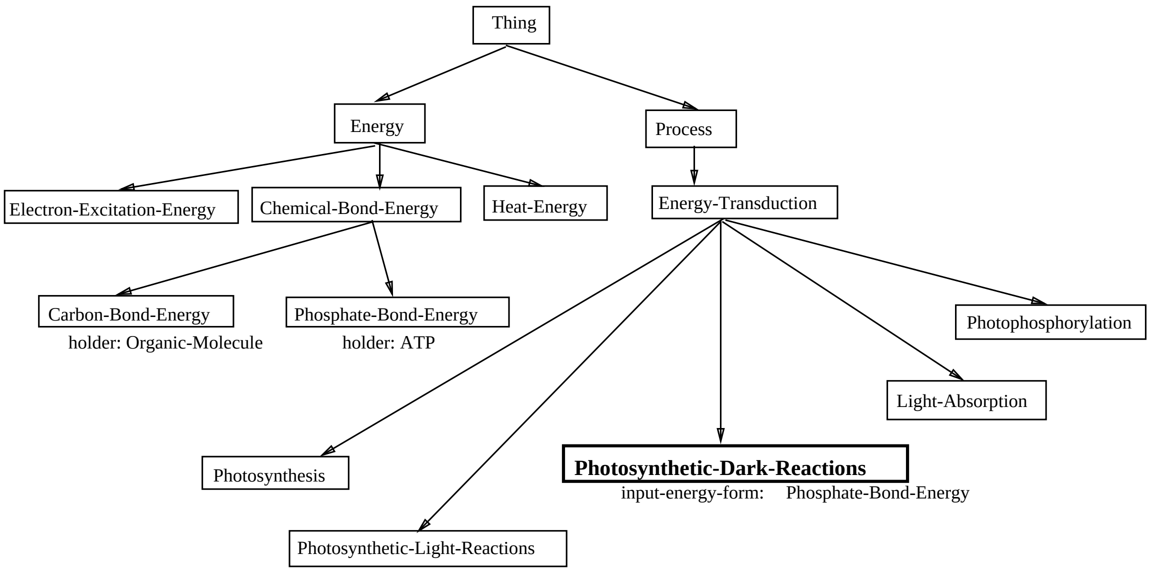

Figure 3.9: Knowledge-base fragment used to illustrate content addressability for the description (Energy-Transduction (input-energy-form (Energy (energy-holder ATP)))). KASTL, in *recognition-only* mode, returns the list (Photosynthetic-Dark-Reactions).

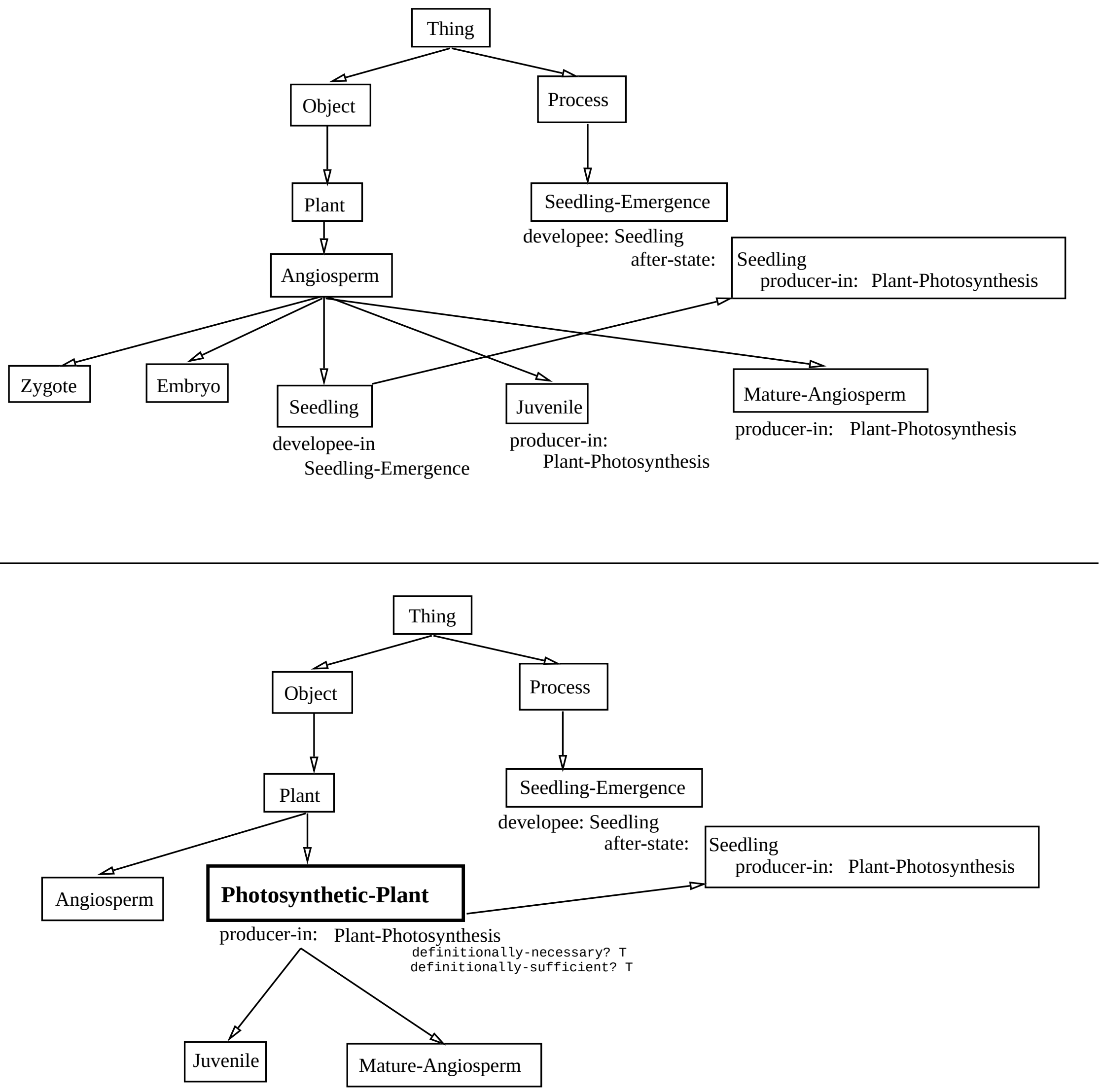

Figure 3.10: Snapshots of the knowledge base before and after reification of *Photosynthetic-Plant*, described by (Plant (producer-in Plant-Photosynthesis)).

- Given (Water (transportee-in (Diffusion (source Soil-Region) (destination Plant)))), which describes "Water moved by diffusion from a soil region to a plant," and the knowledge-base fragment shown in the top portion of Figure 3.11, KASTL modifies the knowledge base as shown in the bottom portion of Figure 3.11. In addition to creating a frame to represent the new specialization of *Water*, KASTL also creates a frame to represent the new specialization of *Diffusion*, "diffusion of water from a soil region to a plant," referenced by the nested description.

- Given (Evaporation (transportee Water) (source Soil-Region)), which describes "Evaporation of water from the soil," and the knowledge-base fragment shown in the top portion of Figure 3.12, KASTL modifies the knowledge base as shown in the bottom portion of Figure 3.12.

- Given (Substance (raw-material-for Photosynthesis)), which describes "Substance that is consumed by some photosynthesis event," and the knowledge-base fragment shown in the top portion of Figure 3.13, KASTL modifies the knowledge base as shown in the bottom portion of Figure 3.13. In this example, the user has specified the name to be given to the new frame, *Raw-Materials-for-Photosynthesis.*

3.4 Dynamic Partitioning

The previous section discussed the task of reifying single concepts that are in the virtual knowledge base. This section discusses a related task, one that involves reifying several concepts st once. This is the task of dynamically creating new partitionings in the knowledge base.

A partitioning is a portion of a knowledge base in which a concept is partitioned (broken down) in some way. There are at least four types of partitionings. First, a concept can be partitioned into specializations. For example, *Cell* can be partitioned into specializations *Animal-Cell* and *Botanical-Cell.* Second, objects can be partitioned into their physical parts or composing substances. For example, a seed can be partitioned into the seed coat, the embryo, and the endosperm. Third, objects can be partitioned into temporal parts (called *states* or *stages*). For example, a plant can be partitioned into five stages: zygote, embryo, seedling, juvenile, and mature plant. Fourth, events can be partitioned into steps (called *subevents*). For example, photosynthesis can be partitioned into the light reactions and the dark reactions. A slot that represents a partitioning (*e.g., specializations, has-parts, composed-of, stages, subevents*) is called a *partitioning slot.*

A concept can be partitioned in multiple ways on any partitioning slot, depending on the *partitioning criterion.* For example, the partitioning of *Human* into specializations *Male* and *Female* is based on a gender criterion. Other criteria yield other partitionings. For example, a hair color criterion gives rise to specializations *Blonde*, *Brunette*, etc. Each element of the partitioning (*e.g.*, each specialization of *Human*) has a more specialized value for the partitioning criterion than the partitioned concept has [*e.g.*, a more specialized value for

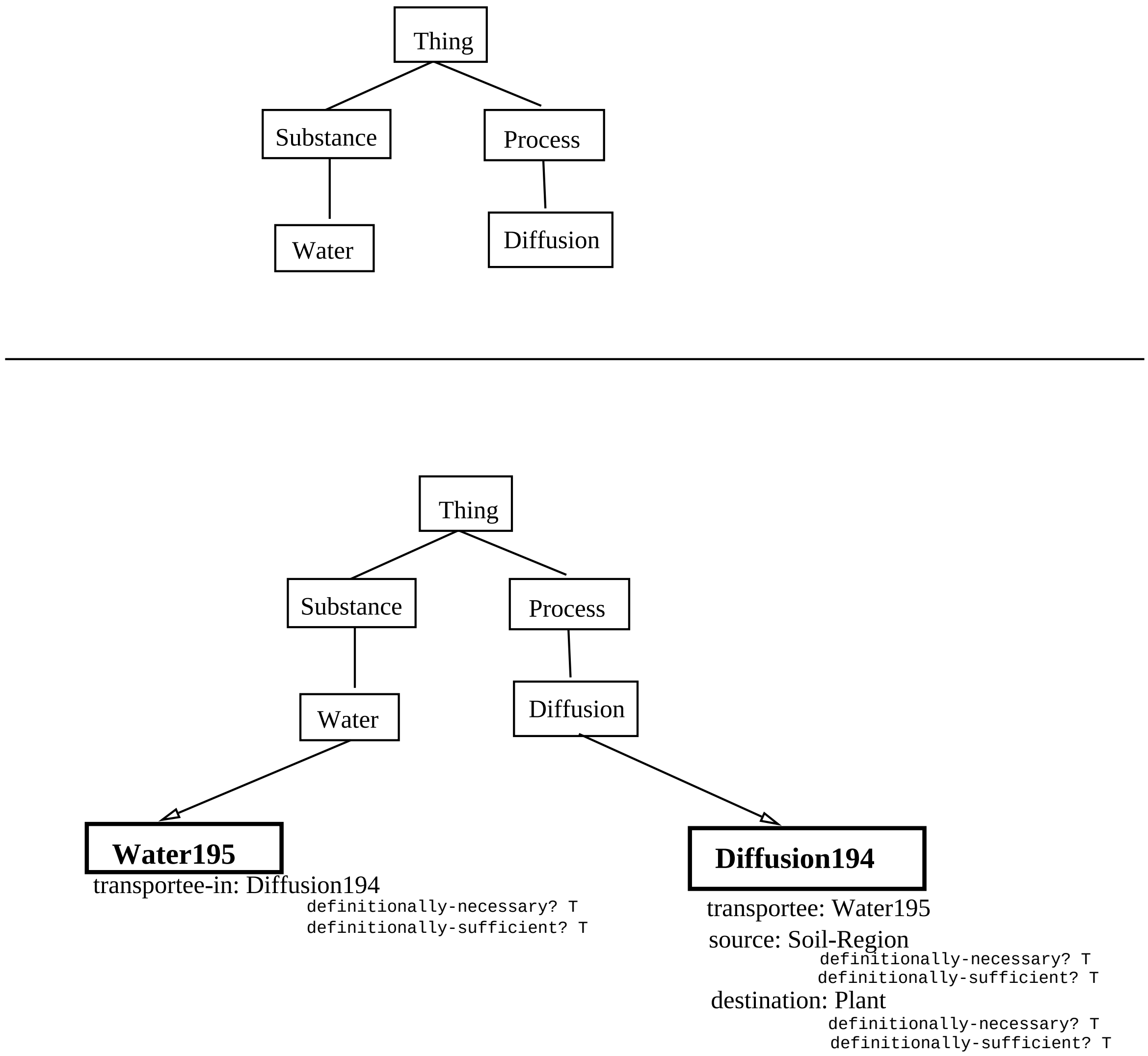

Figure 3.11: Snapshots of the knowledge base before and after reification of *Water195*, described by (Water (transportee-in (Diffusion (source Soil-Region) (destination Plant)))).

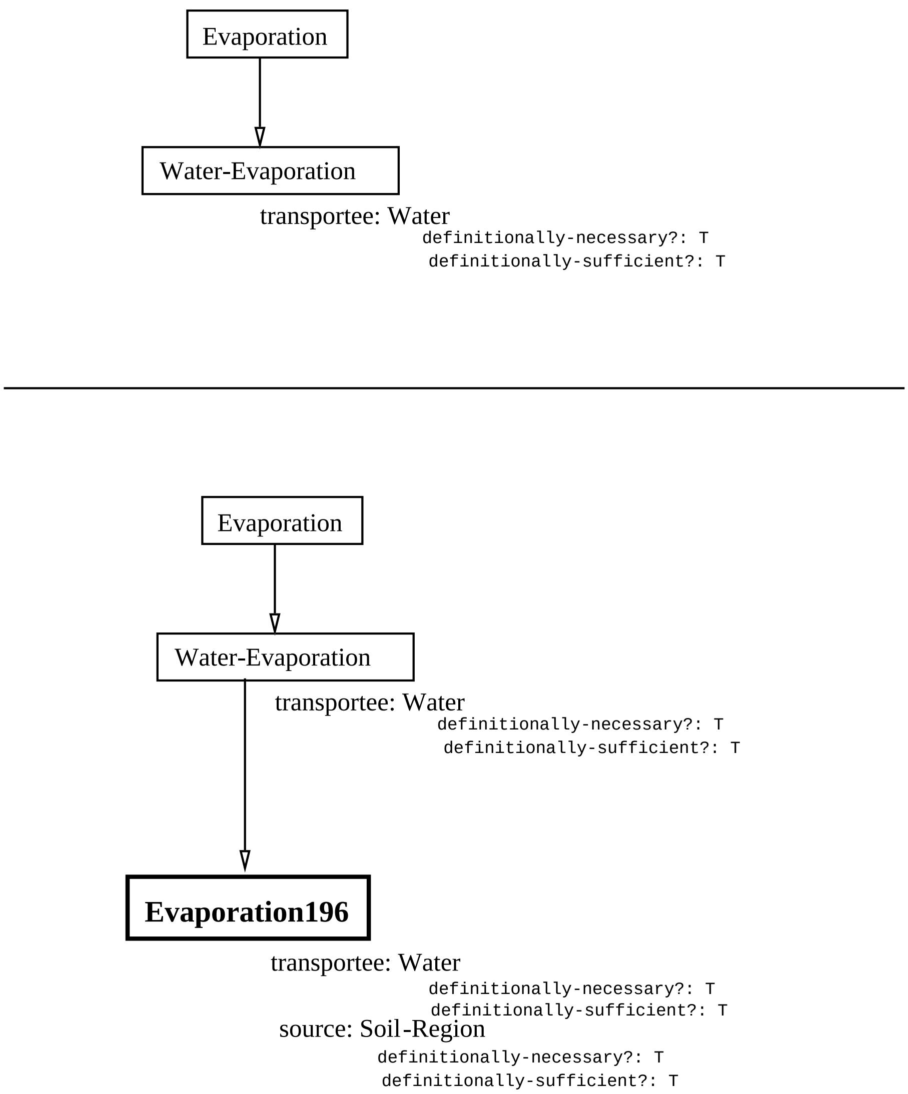

Figure 3.12: Snapshots of the knowledge base before and after reification of *Evaporation196*, described by (Evaporation (transportee Water) (source Soil-Region)).

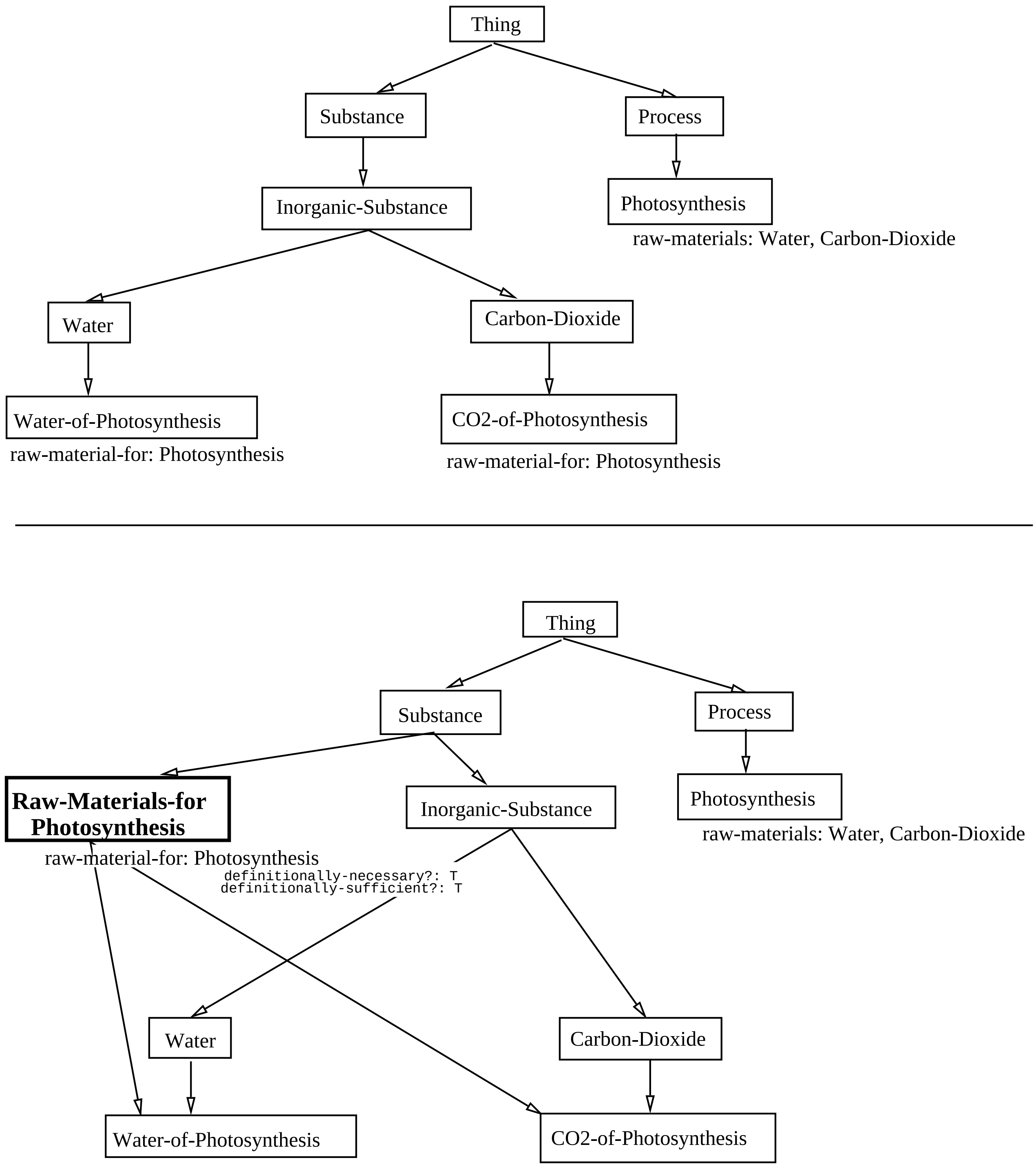

Figure 3.13: Snapshots of the knowledge base before and after reification of *Raw-Materials-for-Photosynthesis*, described by (Substance (raw-material-for Photosynthesis)).

hair-color than the value of (*Human hair-color*)]. KM allows any partitioning slot to have multiple, orthogonal partitionings and provides a mechanism for representing the criterion for each.

Although multiple partitionings are representable for each partitioning slot, the knowledge engineer typically represents only a few of the possible partitionings. Application programs may need some of the unrepresented partitionings to support some task. For example, a tutoring system generating a description of the different ways that a leaf acquires glucose throughout its lifetime would need a partitioning of *Leaf* into stages according to glucose acquisition method. To service this need, KASTL provides dynamic creation of new partitionings. In other words, KASTL reifies partitionings that are in the virtual knowledge base but not in the actual knowledge base. For example, given the concept *Energy-Transduction*, partitioning slot *specializations*, and criterion slot *input-energy-form*, KASTL automatically creates a new partitioning of *Energy-Transduction* into specializations $ET1$ (which has *input-energy-form* =*Light-Energy*) and $ET2$ (which has *input-energy-form* = *Electron-Excitation-Energy*), as shown in Figure 3.14.

The task of dynamically creating partitionings can be described as follows:

Given:

- A concept C to partition (*e.g., Energy-Transduction*),
- A partitioning slot, S (*e.g., specializations*), and
- A partitioning criterion (*e.g., input-energy-form*),

Return:

- A set of newly created concepts that are related to C by slot S and that have more specific values for the partitioning criterion than C has (*e.g.*, specializations of *Energy-Transduction* that have different specific forms of input energy), and
- A knowledge base reorganized to accommodate the newly created concepts.

The partitioning criterion may be a simple slot name or a slot path. For example, given the concept *Angiosperm* (flowering plant), partitioning slot *specializations*, and (as the partitioning criterion) the slot path *(has-part Flower color)*, KASTL would partition *Angiosperm* into specializations according to the different colored flowers they have. The criterion may also include a value filter, which signifies that when creating the partitioning, KASTL should ignore values of the partitioning criterion that are not specializations of the concept given as the filter. For example, given the concept *Tree* to partition, partitioning slot *specializations*, and partitioning criterion *has-parts* with value filter *Leaf*, KASTL would partition *Tree* into specializations according to the different types of leaves they have.

To dynamically create a new partitioning of concept C along partitioning slot S according to a given partitioning criterion, KASTL first explores an existing partitioning of

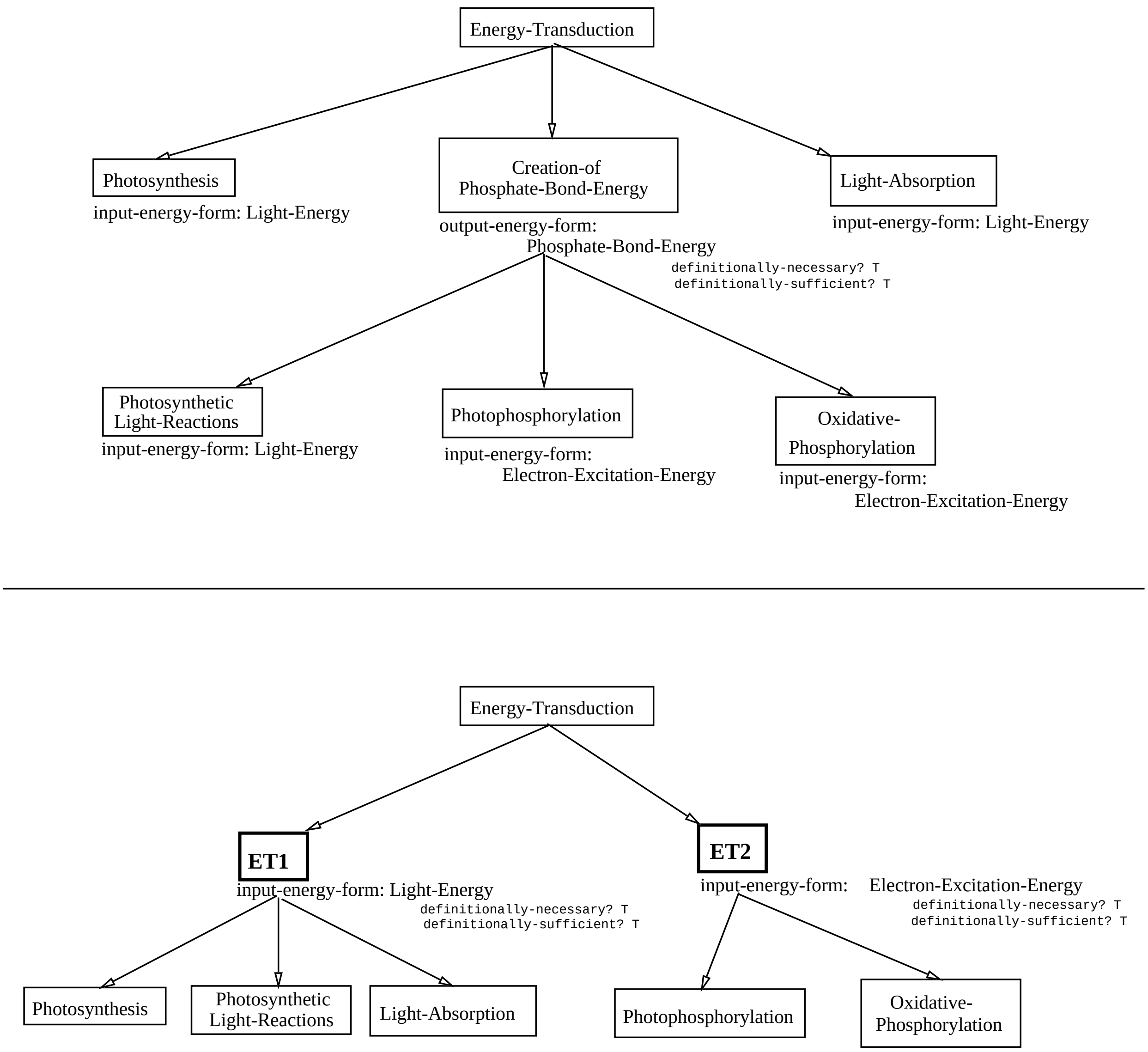

Figure 3.14: Knowledge-base fragment used to illustrate dynamic partitioning of *Energy-Transduction* along partitioning slot *specializations* according to criterion *input-energy-form*. Although not shown, the existing partitioning remains intact.

C along partitioning slot S to find the top-most concepts that are different from (or more specialized than) C with respect to the partitioning criterion. For example, to create a partitioning of *Energy-Transduction* into specializations according to *input-energy-form*, KASTL first finds the most general specializations of *Energy-Transduction* that have a different (or more specific) value for *input-energy-form* than *Energy-Transduction* does. Assuming the knowledge-base fragment shown in the top portion of Figure 3.14, this set is *Photosynthesis, Photosynthetic-Light-Reactions, Photophosphorylation, Oxidative-Phosphorylation*, and *Light-Absorption*.

The next step is to collect the values that these concepts have for the partitioning criterion (the *criterial values*). For example, the criterial values of the concepts *Photosynthesis, Photosynthetic-Light-Reactions, Photophosphorylation, Oxidative-Phosphorylation*, and *Light-Absorption* on slot *input-energy-form* are *Light-Energy* and *Electron-Excitation-Energy*. The new partitioning will have one "branch" for each distinct criterial value. For each criterial value, KASTL creates a new concept (using the procedure given in Figure 3.6) to represent the category of entities having that criterial value. The new concept will encompass all the concepts encountered in the previous search that have that criterial value. For example, the new partitioning of *Energy-Transduction* will include the newly created specialization of *Energy-Transduction* $ET1$, for which *input-energy-form* = *Light-Energy*. This specialization will have specializations *Photosynthesis, Photosynthetic-Light-Reactions*, and *Light-Absorption*. Similarly, the new specialization $ET2$ for which *input-energy-form* = *Electron-Excitation-Energy* will have specializations *Photophosphorylation* and *Oxidative-Phosphorylation*. The bottom portion of Figure 3.14 shows the knowledge base of the top portion of Figure 3.14 modified to accommodate the new partitioning of *Energy-Transduction*. (Existing partitionings are left intact; Figure 3.14 shows only the new partitioning.)

KASTL creates dynamic partitionings to one level only. If users require additional levels of partitioning, they can create them easily through recursive calls to the procedure described above.

Although KASTL can automatically create a new partitioning for any partitioning slot and any partitioning criterion, not all combinations of partitioning slots and partitioning criteria make sense. For example, consider automatically partitioning *Flower* into parts according to weight, using the algorithm described above. The standard partitioning of *Flower* into parts is *Calyx, Corolla, Androecium*, and *Gynoecium*. Suppose these parts have *weight* values of six grams, nine grams, one gram, and one gram. To create a new partitioning based on weight, KASTL would create a "six gram part" (the calyx), a "nine gram part" (the corolla), and a "one gram part," which has as parts both the androecium and the gynoecium. KASTL would assert that the "one gram part" weighs one gram and that it has two parts, each of which also weighs one gram. This inconsistency arises because it does not make sense to speak of "*the* part of the flower that weighs one gram." It does make sense, however, to speak of "the category of flower parts that weigh one gram." Thus, a more meaningful request is for a partitioning of *Flower-Part* into specializations according to weight, rather than partitioning the typical *Flower* into parts according to weight.

The class of partitionings that will not raise this problem (*i.e.*, those that are guaranteed not to introduce inconsistencies) can be circumscribed as follows. A partitioning along partitioning slot S according to partitioning criterion slot P is meaningful if the following implication holds:

$$\forall C, X.[\forall V, X'.inKB(C, S, V) \Rightarrow inKB(V, P, X') \wedge X' \subseteq X)] \Longrightarrow inKB(C, P, X)$$

where $inKB(frame, slot, value)$ is true when triple ⟨*frame slot value*⟩ is in (or implied by) the knowledge base. That is, if all values for slot S on the frame for concept C have values for slot P that are X (or a specialization of X), then C also has value X for slot P. The consequent of the above implication characterizes the triples that KASTL adds to the knowledge base as it creates a partitioning based on P and S. Thus, if this implication holds, then the partitioning will not introduce inconsistencies. For example, consider the combination of partitioning slot *has-part* and partitioning criterion *color*. If all of the parts of an object C are the same color X (or are some shade (specialization) of X), then the color of C is also X. Thus the above implication holds, so a partitioning of an object into parts based on color will not introduce inconsistencies. Reconsider the earlier example that combined partitioning slot *has-part* with partitioning criterion *weight*. If all of the parts of some object C have the same weight X, it is *not* true that C also has that weight. Thus, the above implication does not hold, so this combination of partitioning slot and partitioning criterion is not meaningful (*i.e.*, it may introduce inconsistencies).

The broadest class of partitionings that satisfy the constraint given above is the set of partitionings that combine *specializations* as the partitioning slot and a partitioning criterion slot that is of semantic type 1 or 2 (such as *has-part, color, weight*, or *actor-in*).

3.5 Cost Analysis

This section presents a cost analysis of automatic classification, adding new concepts to an existing taxonomy. Automatic classification is the primary and most costly activity in accessing concepts in the virtual knowledge base.

Previous complexity analyses have four limitations [85]. First, most past research has focused on the cost of subsumption rather than the cost of classification. Recall that subsumption, determining whether one concept is more general than another, is but one step in classification; more important is the complexity of the overall classification task. Second, past research has analyzed the cost of determining *extensional subsumption*. As described in Section 3.2, KASTL uses a classification algorithm based on *intensional subsumption* rather than extensional subsumption. Intensional subsumption allows the definition of a precisely characterized classification algorithm without restricting expressive power. The third limitation of past results is that most of them give worst-case complexity analyses. Users of the knowledge base are often more interested in average-case predictions. The fourth limitation is that most complexity analyses are given in terms of the size of the concept description being classified. The size of the taxonomy has a much greater impact on the overall cost of classification, because description size is typically small, but taxonomy size may be quite large, especially for a multifunctional knowledge base.

To address the limitations of past research, Woods has analyzed the complexity of classification based on intensional subsumption [85]. Woods shows that the complexity of classification, as a function of how many of the N frames in the taxonomy are compared to the input concept description, is logarithmic or better for typical inputs and linear in the worst case.

Woods uses three parameters to capture how the characteristics of the knowledge base affect the cost of classification. The first is the downward branching ratio, r, the average number of immediate specializations for concepts that are not leaves of the taxonomy. The second is a parameter B, which reflects how "out of balance" the taxonomy is. The third is a parameter W, which gives the width of the ancestor chain above a typical concept due to multiple generalizations. Woods estimates that B and W are in the range of one to three for most knowledge bases.

The cost-dominant steps of KASTL's classification algorithm (shown in Figure 3.6) are steps 1, 2, and 3. For steps 1 and 2, which find the most specific generalizations of the new concept, Woods estimates a typical cost of $rBWlogN$ frames examined. For step 3, which finds the most general specializations of the new concept, Woods estimates that, for typical inputs, $r(r+1)$ frames must be compared to the input description. Thus the total number of comparisons is $rBWlogN + r(r+1)$ frames. The value of r (the downward branching ratio) for the Botany Knowledge Base is 4. Using this value of r and an estimate of 2 for parameters B and W, as suggested by Woods, yields an estimated typical-case cost of $16logN + 20$ comparisons. For the Botany Knowledge Base, this would mean comparing a typical input description with up to 5.6% of the 2665 frames currently in the knowledge base.

To empirically evaluate the above cost estimate based on Woods's analysis, 53 botanical concepts that are not explicitly represented in the Botany Knowledge Base, such as "nucleus of an epidermal cell" and "tree that grows in a swamp," were selected. These concepts were chosen at random from a biology textbook, so presumably the sample is representative of the domain. The evaluation consisted of measuring, for each concept, the maximum number of frames that KASTL would examine to classify it in the Botany Knowledge Base taxonomy. The average cost was 117 of the 2665 total frames, about 4.3%, not significantly different from Woods's estimate. The minimum was 32 frames (1%), and the maximum was 804 frames (30%). (Recall that in the worst case, KASTL would have to examine 100% of the frames in the taxonomy.) The cost of classifying a particular concept depends largely on the position of the concept in the taxonomy. The more general the concept, (*i.e.*, the closer the concept is to the root of the taxonomy), the more costly it is to classify.

The above analysis determined the number of frames to be examined when classifying a new concept. For each frame examined, KASTL must determine the subsumption relationship of the concept that frame represents and the given concept description. Woods estimates the typical-case cost of determining intensional subsumption as $(2m^2 + p^2)$ frame-slot access operations, where m is the number of features in the input concept description and p is the number of base concepts in the input description. (The concept specification language allows multiple base concepts, interpreted conjunctively.) This estimate assumes that concept descriptions do not contain binding constraints among slot values.

To empirically evaluate Woods's estimate of the cost of computing intensional subsumption, actual values for m and p were derived from concept descriptions found in a biology textbook. Of 155 concept descriptions, all but four had a single base concept (*i.e.*, $p \approx 1$ on average), and the average number of features modifying the base concept (m) was 1.3. Thus, the average cost of computing intensional subsumption for these concepts is 4.4 frame-slot access operations, assuming the descriptions do not include binding constraints. If binding constraints do appear, then the cost of computing intensional subsumption is probably exponential in m. Nevertheless, it seems that the cost will not be prohibitive given that most concept descriptions include very few features.

The cost of classifying a new concept, then, is the product of

- the number of frames to be compared with the input description, and
- the number of frame-slot accesses required to compare a particular frame with the input description (and compute the subsumption relationship between them).

The empirical evidence indicates that, in terms of the actual coefficients, the dominant factor by far is usually the number of frames to be examined, rather than the number of frame-slot accesses required to compute subsumption. The complexity KASTL's classification algorithm, as a function of the number of frames to be examined, is tractable: logarithmic for typical inputs, and linear in the worst case.

3.6 Summary and Limitations

This chapter presents techniques for insulating users of the knowledge base from the particulars of how knowledge is represented. To insulate users from the effects of choices regarding the names of frames, KASTL provides content addressability. The advantage of content addressability is that users can access the knowledge base without extensive prior knowledge of how it has been represented. In particular, users can access concepts without knowing the names of all the frames in the knowledge base. They need only to know the names of the most general frames and slots (the top level of the taxonomy). They can access other concepts by describing them in terms of more general frames and slots. Although partially defined concepts, such as "natural kinds," cannot always be uniquely identified by description alone, these concepts usually have standard names (*e.g.*, Photosynthesis, Flower, and Soil).

To insulate users from the effects of representational choices regarding which concepts are reified in the knowledge base, KASTL provides access to concepts in the virtual knowledge base. Accessing concepts in the virtual knowledge base requires performing automatic classification of the given concept within the knowledge-base taxonomy. Many existing systems perform automatic classification, including most languages in the KL-ONE family [86]. These systems, however, have several limitations. First, many of them limit expressiveness in an effort to achieve tractable algorithms for the subsumption step of classification. This results in languages so limited that they are no longer generally useful.

The second limitation of traditional classifiers is that they use ill-characterized subsumption algorithms. Past systems have used the extensional definition of subsumption, which has been found to be intractable for most languages. As a result, systems that are based on extensional subsumption have retreated to tractable but incomplete algorithms. These algorithms lack a precise specification of what subsumption relationships they detect.

The third limitation of traditional classifiers is that they are based on extensional subsumption, but they are restricted to using only definitional (terminological) knowledge, knowledge that has no assertional import. Extensional subsumption cannot always be computed solely from definitional knowledge.

To address the limitations of existing classifiers, KASTL is based on the *intensional subsumption* criterion Woods gives [85]. Intensional subsumption means that X subsumes Y when the definition (intension) of X is more general than the definition of Y. Intensional subsumption makes possible a well-characterized classification algorithm without limiting the expressiveness of the representation language.

The disadvantage of using intensional subsumption rather than extensional subsumption is that some relationships of extensional subsumption that would be useful to a reasoning system will not be discovered. For example, KASTL might not discover that "Cell with a chloroplast" subsumes (extensionally) "Cell that is photosynthetic," even if the knowledge base contains the information that all photosynthetic cells have chloroplasts, because the subsumption cannot be determined solely by examining the two concept definitions.

KASTL performs both the tasks of accessing concepts by description and accessing concepts in the virtual knowledge base within a single module with a single user interface. This has two advantages. First, the procedure that reifies a concept in the virtual knowledge base can use information gathered while attempting to find that concept in the actual knowledge base, making the system more efficient. Second, users of KASTL do not need to know whether concepts exist in the actual knowledge base; they simply provide a description of the concept needed. If KASTL fails to find a frame representing that concept, it automatically creates one. Users do not need to know or specify whether they are accessing existing concepts by description or accessing concepts in the virtual knowledge base.

Users of KASTL describe concepts using the *concept specification language*, a formal language that has the same syntax as KM, the representation language. The advantage of using the same syntax is that users can describe concepts in the same way they represent them. A limitation of KASTL's concept specification language is that there are constructs available in KM that are not included in the specification language. Thus, there are concepts representable in KM that cannot be described in the specification language, and hence cannot be accessed by description or automatically reified from the virtual knowledge base.

One such construct is *role value maps*, which express binding constraints. With role value maps, one can say that two frame-slots are filled by the same value. For example, with role value maps one could describe the concept "Person whose spouse is his/her best friend." Without role value maps, one can only describe "Person whose spouse is a person and whose best friend is a person." As discussed in the previous section, extending the concept specification language to include role value maps will probably result in a subsumption algorithm whose cost is exponential in the length of the input description, but description length is typically so small that the cost will nevertheless be reasonable.

A second limitation of KASTL's concept specification language is that, unlike KM, each feature is interpreted as both necessary and sufficient. Separate statements of necessary features and sufficient features are not allowed. One area for future work is to extend KASTL's concept specification language to include role value maps and to distinguish necessary features from sufficient features.

In addition to dynamically reifying concepts that are in the virtual knowledge base, KASTL also performs dynamic partitioning. For a particular partitioning slot (*specializations, has-part, etc.*), the knowledge engineer typically represents only a few of the possible partitionings. KASTL allows users to access partitionings that are not explicit in the knowledge base by dynamically creating new partitionings. Although KASTL can automatically create a new partitioning for any combination of partitioning slot and partitioning criterion, not all possible partitionings are meaningful. Section 3.4 gives a formal characterization of the class of partitionings guaranteed not to introduce inconsistencies in the knowledge base.

To summarize, KASTL makes users of a knowledge base less vulnerable to the particulars of how knowledge is represented by

- providing content addressability,
- providing access to concepts in the virtual knowledge base, and
- providing dynamic partitioning.

These facilities make it easier for users to locate the frames that are relevant to a particular task. Once the relevant frames have been isolated, users must then determine which slot values of those frames are relevant. The next chapter discusses KASTL's techniques for assisting users in selecting slot values. These are techniques for accessing *viewpoints* of concepts.

Chapter 4

Accessing Viewpoints of Concepts

To be truly absurd, you need a coherent point of view!!
Zippy the Pinhead by Bill Griffith

4.1 Introduction

This chapter presents general methods for accessing *viewpoints* of concepts, coherent collections of facts that describe a concept from a particular perspective. Viewpoints are essential for a variety of tasks, including explanation generation, compositional modeling, problem solving, and learning. This research identifies several types of viewpoints and develops computational methods for dynamically generating viewpoints of each type.

4.1.1 Motivation

Consider a question-answering system generating a description of photosynthesis using the knowledge-base fragment shown in Figure 4.1. In most contexts, a coherent description requires only a small fraction of all of the available information, because a description containing all the information would overwhelm the user. The system cannot, however, select an arbitrary subset of the information and still produce a coherent description. Even selecting relations that are closest to (or directly on) the *Photosynthesis* frame does not ensure coherence, as evidenced by the following description:

> Photosynthesis is a kind of production and also a kind of energy transduction. It occurs in a chloroplast. In photosynthesis, a photosynthetic cell produces carbon bond energy. The input energy form is light energy, provided by a photon. The raw materials of photosynthesis are carbon dioxide and water. The energy source for photosynthesis is ATP. The products of photosynthesis are glucose and oxygen. Photosynthesis has some temporal duration. One step of photosynthesis is the light reactions, and another step is the dark reactions.

A shorter response is not necessarily more coherent. Omitting information in an unprincipled way introduces incoherence because of incompleteness:

> Photosynthesis is a kind of production that occurs in a chloroplast. The energy source for photosynthesis is ATP. Photosynthesis uses carbon dioxide and produces oxygen. One step of photosynthesis is the dark reactions.

Selecting relations based on an *a priori* importance ranking is also not sufficient to ensure a coherent response, as the SCHOLAR project demonstrated [14].

To generate a coherent description, the system needs to select from all the available information a subset that can be understood as a whole, a collection of facts that are relevant to one another, as in the following description:

> Photosynthesis is a kind of production in which a photosynthetic cell uses water and carbon dioxide to produce glucose and oxygen.

One way to select coherent portions of knowledge is to access *viewpoints* of concepts.

4.1.2 Viewpoints

A viewpoint is a coherent collection of facts (*i.e.*, ⟨*frame slot value*⟩ triples) that describes a concept from a particular perspective. For example, a structural viewpoint of the concept *Seed-Coat* describes the substances and parts that make up a seed coat and how they are connected. The viewpoint of *Seed-Coat* as a kind of *Container* includes information about what parts of the seed the seed coat contains, whether the seed coat has openings, etc. The viewpoint of *Seed-Coat* as having no chlorophyll includes the fact that seed coats are not photosynthetic. Each concept has multiple viewpoints, which give different aspects or perspectives of the concept. The union of all the viewpoints provides complete knowledge of the concept.

Barsalou gives evidence of the psychological validity of viewpoints [6]. Although he uses the term *concept* rather than *viewpoint*, he describes viewpoints in this way (with terminology substitution added):

> ...the knowledge for a particular category contains many, many [viewpoints]. On a given occasion, the [viewpoint] that is constructed to represent a category only traces out a small subset of all the knowledge available in long-term memory for representing the category. ...the [viewpoint] used to represent a category on a particular occasion contains information that provides relevant expectations about the category in *that* context.

Although viewpoints constitute a subset of the total knowledge of a concept, viewpoints are not limited to the information found on the frame representing that concept. The terms *frame* and *concept* are often used interchangeably, but a single frame does not provide complete knowledge of a concept. The total meaning of a concept is given by a network of relations centered on the frame representing that concept. The frame includes only the relations

Figure 4.1: The portion of the Botany Knowledge Base in the region of *Photosynthesis*.

that explicitly reference the concept. (For example, the total knowledge of photosynthesis includes the information that it produces the glucose required for respiration, represented by the triples ⟨*Photosynthesis product Glucose*⟩ and ⟨*Glucose required-for Respiration*⟩, but only the first of these triples appears on the *Photosynthesis* frame. The second appears on the *Glucose* frame.) A viewpoint of a concept includes any relation that is relevant, regardless of whether it appears on the frame representing the concept.

4.1.3 Viewpoint Coherence and Other Sources of Coherence

Viewpoints were described above as *coherent* collections of facts. A viewpoint is coherent to the extent that all the facts it comprises are relevant to one another. For example, all the facts in a coherent viewpoint of "photosynthesis as a carbon dioxide utilization process" must be relevant to the fact that photosynthesis utilizes carbon dioxide. Because relevance is both subjective and graded, viewpoint coherence cannot be precisely defined. One can better understand what viewpoint coherence is, however, by understanding what it is *not*.

Hobbs points out that coherence is not simply sharing a common referent [35]. That is, a set of facts that describe the same concept is not necessarily coherent. Consider the following example from [35]:

> Ronald Reagan used to act in cowboy movies. He appointed Caspar Weinberger as Secretary of Defense.

Although the statements are *cohesive*, because they both refer to Ronald Reagan [32, 26, 21], they are not coherent. Coherence requires a stronger relationship between facts.

Viewpoint coherence should also be distinguished from other sources of coherence, such as *discourse coherence*. Discourse coherence reflects the degree to which a discourse accomplishes general communication goals, such as motivating or persuading the reader, explicitly linking new information to what the reader already knows, and guiding the reader's inferences about what is said. Hobbs [35, 36] and Rhetorical Structure Theory [45] characterize discourse coherence with sets of *coherence relations* and *rhetorical relations* that hold between two segments of a discourse. These relations describe valid conversational "moves" [35].

In general, coherence relations and rhetorical relations characterize discourse coherence rather than viewpoint coherence. They reflect the rhetorical intentions of the speaker rather than the semantic or conceptual connectivity of information. Two of Hobbs's coherence relations, however, are the kind of relation that holds between facts in a coherent viewpoint. These are *cause* and *enablement*, generalized by *occasion*. According to Hobbs, these relations reflect coherence in the world or the structure of memory and the ways we are conventionally reminded of things, rather than discourse goals. This is the kind of relationship that holds among facts in a coherent viewpoint. Alterman gives a larger set of coherence relations of this kind that characterize taxonomic, partonomic, and temporal relations between events or states [2, 3].

Another source of coherence is *global coherence* [35]. The global coherence of a discourse is the degree to which each of its utterances is relevant to the speaker's specific communication plan [29]. (This is the criteria that Moore's explanation generator uses when constructing its *perspectives* [51, 52].) Viewpoint coherence, by contrast, is a more general notion that applies even in the absence of any problem-solving goal or discourse context. Although some viewpoints are coherent because their facts share a common relevance to some problem-solving goal, this research focuses on task independent viewpoints.

A source of coherence that is very similar to viewpoint coherence is *explanatory coherence* [80, 60]. Explanatory coherence is a specialized form of viewpoint coherence that measures how well a hypothesis explains a set of observations. Explanatory coherence is a property of a logical proof, but viewpoint coherence applies to more general sets of assertions. The underlying notion, however, is the same. The greater the "connectedness" within a set of assertions (or within a proof), the greater its coherence.

Another source of coherence is *textual coherence*, the way that a collection of facts making up a text is organized and presented. Viewpoint coherence, unlike textual coherence, is influenced solely by content (the facts the viewpoint comprises).

Finally, viewpoint coherence is not the same as *conceptual coherence*. Conceptual coherence is a term used in the psychology literature to describe the degree to which the members of a category form a comprehensible, informative, or useful class [53]. Conceptual coherence describes groupings of entities into categories; viewpoint coherence describes groupings of assertions about categories (or instances) into descriptions. Although conceptual coherence and viewpoint coherence differ, they are related. Murphy and Medin propose that one component of conceptual coherence is the degree to which a category's description includes features that are connected via some relationship [53]. Thus, only coherent concepts can be described by coherent viewpoints.

To summarize, viewpoint coherence is a property of a set of facts that reflects the degree to which the facts are interrelated. The coherence of a viewpoint is determined entirely by its contents, and not by the way it is organized or presented. Furthermore, the coherence of many viewpoints is apparent in the absence of any problem-solving task or dialogue context. These are the viewpoints of interest in this work. After much analysis by researchers in several disciplines, coherence remains an ill-defined notion. The goal of this work, however, is not to provide a precise definition of coherence, but to understand how an access method can generate viewpoints that will be judged coherent by human subjects.

4.1.4 Applications of Viewpoints

As Section 4.1.1 suggested, an important application of viewpoints is to ensure the coherence of the explanations that a question-answering or explanation-generation system produces. (Other sources of coherence, such as discourse coherence and textual coherence, also contribute to the overall coherence of an explanation.) Viewpoints lend coherence to both the content and the organization of an explanation. In selecting the content of an explanation, it is important for a system to be sensitive to the current context, because the coherence of the explanation will be judged relative to that context. Different viewpoints provide different explanations of domain knowledge, each appropriate for different levels of expertise [76, 64, 65, 79], different user needs [49], different system goals [51, 52], and different dialogue contexts [46, 47, 51]. In organizing an explanation, it is important for a system to group together facts that are related to one another [48]. One way to accomplish this is to group together facts that belong to the same viewpoint [42]. Suthers points out an additional benefit of constructing explanations from viewpoints: accessing the knowledge base at the right level of abstraction (*i.e.*, at the viewpoint level) allows an explanation generator to concentrate on issues of discourse management, and it facilitates portability across domains and representational formalisms [75].

In addition to their utility for explanation generation, viewpoints are also important for a variety of other applications, such as natural language processing. In discourse understanding, Grosz uses viewpoint-like knowledge structures called *focus spaces* to focus the system's attention on the knowledge that is most salient at a given point in the dialogue [28]. For example, even if "the mayor of San Diego" and "my neighbor" refer to the same person, the information in focus differs depending on whether the system views the person as a mayor or as a neighbor, which in turn depends on which reference the system encounters. This focusing enables the system to access more important information first as it disambiguates a sentence. For example, focus spaces constrain the search for possible referents of pronouns and definite noun phrases.

The KING system uses *views* to guide linguistic and conceptual choices that arise in natural language generation [37]. For example, whether the system generates the sentence "Mary was sold the book by John" or the sentence "Mary bought the book from John" depends in part on whether the *actor* of the event is Mary or John, which in turn depends on whether the system views the event as a buying event or a selling event.

Other systems use knowledge similar to viewpoints to constrain automated reasoning. For example, Falkenhainer and Forbus use *perspectives* to construct models of physical devices [23]. Their system has a knowledge base of model fragments, each pertinent to a particular perspective. The system constructs a model by composing model fragments that are relevant to the chosen perspective. Perspectives ensure that the model makes consistent simplifying assumptions. For example, a model should not view a feed tank simultaneously as an infinite capacity liquid source and as a container that may be emptied. Although incompatible, each of these different views is appropriate for different reasoning tasks. Perspectives also ensure the relevance of the model to the reasoning task for which it is constructed. For example, the system should not model a steam plant as an abstract heat engine if the user asks about the mass flows that occur in it. Finally, perspectives yield simplified models. Simplified models are crucial because reasoning about every aspect of a mechanism is computationally prohibitive, even for simple objects. In addition, simplified models are more generally applicable (because they require less data), and they yield more coherent explanations of problem-solving behavior.

The ability to take multiple viewpoints of an object is also important for solving physics problems. For example, the ISAAC and APEX problem solvers construct a formal representation of the given problem by viewing each object in the problem as a *canonical object*, an idealized or abstract object such as a point mass or a lever [62, 38]. Viewing actual objects as canonical objects is important because, while problems are stated in terms of complex, real-world objects, the principles and laws of physics are stated in terms of canonical objects. Competence in solving physics problems derives mainly from the ability to formulate the problem in terms of canonical objects so that domain principles and laws can be applied [38]. In addition, viewing a real-world object as a canonical object restricts the information about the actual object that may be used to solve the problem, which greatly reduces the size of the problem space [62, 63].

Algernon uses *views* for default reasoning [19]. Many default reasoning schemes are intractable, because they require the system to ensure that a default assumption is consistent with everything that is known before making that assumption. Restricting inference to the information found within a chosen view makes default reasoning more efficient. BLAH uses *partitions* in a similar way to constrain its search for rules relevant to a given problem-solving task [82].

Finally, learning systems use viewpoints. Murray's KI system uses *views* for knowledge integration, the task of incorporating new information into an existing knowledge base [54, 57]. Knowledge integration involves identifying conflicts between the new information and existing knowledge and identifying how new information can explain existing beliefs. This is difficult because finding all of the subtle interactions between new and existing knowledge requires computing the deductive closure of the knowledge base, an intractable operation. KI uses views to limit its search for the consequences of new information. Views determine which concepts and propositions in the knowledge base are most relevant to the new information, then KI applies inference methods only to the knowledge within the selected view(s).

The principal component of Shrager's model of learning by experimentation is *view application* [72]. In the model, views guide incremental changes to the learner's theory of how a complex device works. Using views ensures that the space of theories is searched rapidly and that only coherent theories are explored.

Although viewpoints are important for a variety of applications, existing methods for dynamically generating viewpoints from a knowledge base are limited. The goal of this research is to provide general methods for generating viewpoints (*i.e.*, methods that are applicable to a variety of tasks and domains).

4.1.5 Generating Viewpoints

The task of generating viewpoints can be described as follows:

Given:

- a knowledge base containing declarative knowledge of domain concepts, and
- a viewpoint specification, which indicates the type of viewpoint required and the concept of which the viewpoint will be taken (the *concept of interest*),

Return: A collection of facts (*i.e.*, ⟨*frame slot value*⟩ triples) from the knowledge base that constitutes the specified viewpoint. This collection includes both facts that are explicit in the knowledge base as well as facts that must be computed (*i.e.*, facts that are in the virtual knowledge base).

The methodology used in this research on generating viewpoints is as follows. First, identify recurrent *viewpoint types* by analyzing human-generated texts. Second, develop methods for generating viewpoints of each type from a knowledge base. Third, implement these methods in a system, called the *View Retriever*. Fourth, evaluate the quality of the viewpoints the View Retriever generates. Fifth, identify weaknesses of the current set of viewpoint types through further text analysis, and use these weaknesses to guide future research.

A major contribution of this work is a framework of viewpoint types that are independent of any particular domain and task. The current framework of viewpoint types consists of

- *as-kind-of* viewpoints, which describe the concept of interest by relating it to a more general concept. For example, the viewpoint of *Seed-Coat* as a kind of *Container* is an *as-kind-of* viewpoint.
- viewpoints constructed along *basic dimensions*, which describe particular kinds of features of the concept of interest (structural features, functional features, etc.). An example is a structural viewpoint of *Seed-Coat*.
- *as-having* viewpoints, which include features about the concept of interest that are relevant to a user-specified feature of the concept. For example, the viewpoint of *Seed-Coat* as having no chlorophyll is an *as-having* viewpoint.

The next three sections discuss these types of viewpoints in more detail.

A user or application program specifies a viewpoint by indicating the type of viewpoint required and the concept of interest. The concept of interest can be a concept represented either by a frame or by an embedded unit. Furthermore, the concept of interest can be specified by description or by name, and it can be a concept in the actual or virtual knowledge base. The Finder and Creator modules of KASTL replace concept descriptions in viewpoint specifications with frame names as a preprocessing step, using the methods described in Chapter 3.

A second major contribution of this work is a collection of methods for dynamically generating viewpoints. The next three sections describe the methods the View Retriever uses to generate viewpoints of the three types given above. Section 4.2 discusses *as-kind-of* viewpoints, Section 4.3 discusses viewpoints constructed along basic dimensions, and Section 4.4 discusses *as-having* viewpoints. In addition to generating viewpoints of a single type, the View Retriever also constructs composite viewpoints, as Section 4.5 describes. Section 4.6 discusses related work on dynamically generating multiple types of viewpoints and representing viewpoints in a knowledge base.

Section 4.7 presents two evaluations of this work. The first is a subjective analysis of the coverage of the current set of viewpoint types. The second is an objective evaluation that compares the coherence of automatically generated viewpoints to the coherence of human-generated viewpoints. The chapter concludes with a discussion of the limitations of the View Retriever.

4.2 *As-kind-of* Viewpoints

An *as-kind-of* viewpoint describes a concept in terms of a more general concept. For example, the viewpoint "photosynthesis *as-kind-of* production" consists of those facts that explain how photosynthesis is a special kind of production, such as what its raw materials and products are. Figure 4.2 shows a portion of this viewpoint as produced by the View Retriever.

As-kind-of viewpoints correspond to what Lakoff and Johnson call *categorization* [39]. In *Metaphors We Live By*, the authors show that an important way that people structure concepts is by relating them to other concepts. For example, one can think of a conversation as a journey. Categorization is a special case of this in which one relates a concept to a more general concept, a concept of which it is a kind.

The specification of an *as-kind-of* viewpoint requires two parameters:

- the *concept of interest*, the concept that is the focus of the viewpoint, and
- the *reference concept*, a generalization of the concept of interest (although not necessarily an immediate generalization).

For example, the viewpoint shown in Figure 4.2 has the following specification:

(Photosynthesis *as-kind-of* Production).

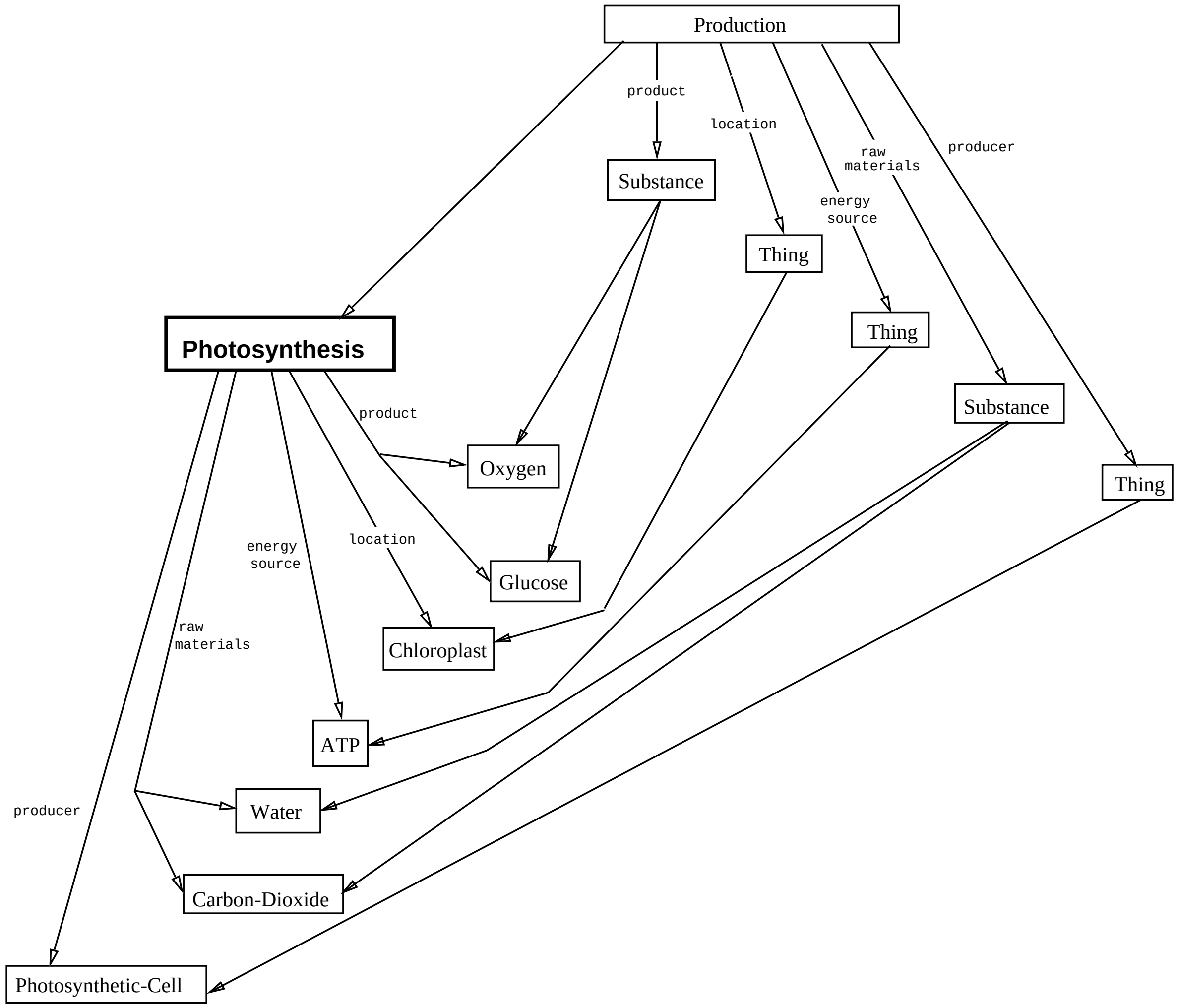

Figure 4.2: The viewpoint of "photosynthesis *as-kind-of* production," as generated from the Botany Knowledge Base by the View Retriever.

The View Retriever constructs *as-kind-of* viewpoints by first selecting relevant facts about the concept of interest (*i.e.*, triples of the form $\langle$*concept-of-interest slot*[1] *value*$\rangle$). A triple is considered relevant if some more general triple appears on the frame for the reference concept. The triple $\langle$*reference-concept slot′ value′*$\rangle$ is more general than $\langle$*concept-of-interest slot value*$\rangle$ if any of the following conditions holds:

1. $slot = slot'$ and $value$ is a specialization of $value'$.

2. $value = value'$, $slot$ is a specialization of $slot'$, and $slot$ emerges either at the concept of interest or on the path between the reference concept and the concept of interest (*i.e.*, the *domain* of $slot$ is some concept that is a specialization of the reference concept and a generalization of the concept of interest).

3. $value$ is a specialization of $value'$, $slot$ is a specialization of $slot'$, and $slot$ emerges on the path between the reference concept and the concept of interest.

For example, the viewpoint shown in Figure 4.2 contains the fact that photosynthesis produces glucose, because it is known that production processes typically produce some substance and glucose is a special kind of substance. More specifically, the View Retriever includes $\langle$*Photosynthesis product Glucose*$\rangle$ in the viewpoint "photosynthesis *as-kind-of* production" because the knowledge base contains $\langle$*Production product Substance*$\rangle$, which is more general than $\langle$*Photosynthesis product Glucose*$\rangle$ because *Substance* is a generalization of *Glucose* [condition (1) above]. The View Retriever also includes in the viewpoint any annotations found on selected triples.

After the View Retriever selects relevant facts involving the concept of interest, it adds to the viewpoint the connections between these facts and the more general facts involving the reference concept. For example, the viewpoint in Figure 4.2 includes not only the fact that photosynthesis produces glucose, but also the facts that photosynthesis is a kind of production, that production processes produce some substance(s), and that glucose is a kind of substance. These connections provide the justification for the viewpoint's contents and are sometimes useful for application programs. For example, a tutoring system can use them to relate new information in an explanation to the student's background knowledge. Figure 4.3 gives the procedure the View Retriever uses to construct as-kind-of viewpoints.

By comparing the knowledge-base fragment shown in Figure 4.1 with the viewpoint shown in Figure 4.2, one sees that constructing a viewpoint involves "filtering out" many irrelevant facts. *As-kind-of* viewpoints provide two kinds of filtering. The first filter removes redundant features, those that the concept of interest and the reference concept have in common. For example, the View Retriever excludes from the viewpoint "photosynthesis *as-kind-of* production" the fact that photosynthesis has a temporal duration because this is true of any production event, and the viewpoint contains the fact that photosynthesis is a kind of production. (If, on the other hand, *Photosynthesis* had a more specific value

[1]Only slots of semantic types 1 and 2 are appropriate for *as-kind-of* viewpoints, because these are the slots that represent inheritable features.

```
Given reference concept R and concept of interest C:

 Ensure that R is a generalization of C.
 For each slot S in (get-slots C R),
     For each value V of (C S),
       For each slot-value pair (RS RV) on R that subsumes (S V),
          Unless RS=S and RV=V and the annotations on (R RS RV)
            do not differ from the annotations on (C S V),
            include in the viewpoint:

              1. (C S V)
              2. (R RS RV)
              3. (V generalizations RV)
              4. all annotations stored on (C S V)
              5. annotations inferred for (C S V) that are
                 subsumed by some annotation on (R RS RV)

(get-slots C R) returns all slots that have stored values on R
  and all slots that are specializations of such slots and that
  originate on the path linking R and C.  (A slot S
  originates on a frame F iff the domain of S is F.)
  A slot S is omitted from the result if S is single-entry
  (it takes at most one value) and S has a specialization S'
  such that the value of (C S) subsumes the value of (C S').
```

Figure 4.3: Procedure the View Retriever uses to construct *as-kind-of* viewpoints.

for *duration* than *Production* has, then the View Retriever would include *duration* in the viewpoint.) The second filter removes irrelevant features, features of the concept of interest that do not fit within the conceptual structure of the reference concept. For example, although the knowledge base contains the information that photosynthesis converts light energy into carbon bond energy, the View Retriever excludes this information because it does not fit within the conceptual structure of production, as represented in the Botany Knowledge Base. (That is, *Production* is not within the domain of slots *input-energy-form* and *output-energy-form*.) The View Retriever *does*, however, include such information in the viewpoint "photosynthesis *as-kind-of* energy transduction," shown in Figure 4.4.

The central notion behind *as-kind-of* viewpoints is that one can emphasize different aspects of a concept by differentiating it with respect to different generalizations. This, of course, is not a novel idea; it has appeared in a variety of disciplines since the time of Aristotle [4]. In the field of artificial intelligence, it appeared as early as 1977 as the basis of KRL, one of the first frame-based representation languages [8]. More recently, McKeown and Suthers have designed systems that automatically generate concept descriptions of this sort [48, 49, 78].

The problem with *as-kind-of* viewpoints that the View Retriever generates (and with the descriptions McKeown's and Suthers's systems generate, when applied to a large, multifunctional knowledge base) is that, even though they focus on a single aspect of a concept, they are nonetheless too unconstrained. They often mix several different kinds of information. For example, the complete viewpoint "pine tree *as-kind-of* tree" includes facts about how a pine tree looks different from a prototypical tree, how its internal structure is different, how its development differs, how its physiology differs, etc.

As a solution, this research includes an additional, orthogonal method of structuring concepts, one that the View Retriever can combine with the method for generating *as-kind-of* viewpoints to further constrain their contents. The next section discusses this method of generating viewpoints and how the View Retriever uses it to enhance the coherence of *as-kind-of* viewpoints.

4.3 Viewpoints Constructed Along Basic Dimensions

Lakoff and Johnson say that people structure their concepts in two ways, by relating them to other concepts (as with *as-kind-of* viewpoints) and according to *basic dimensions* of experience [39]. Basic dimensions are general types of facts, such as facts about the structure, function, or appearance of an object, or facts about the actors or steps of a process. Facts within the same basic dimension convey similar kinds of information. The View Retriever constructs viewpoints along the following basic dimensions for objects and processes:

Basic dimensions for objects:

- **Structural**, which includes the parts or substances that make up the object and their relative sizes and numbers. It also includes the connections and spatial relations among them, as *interconnection relations*. The structural dimension also includes

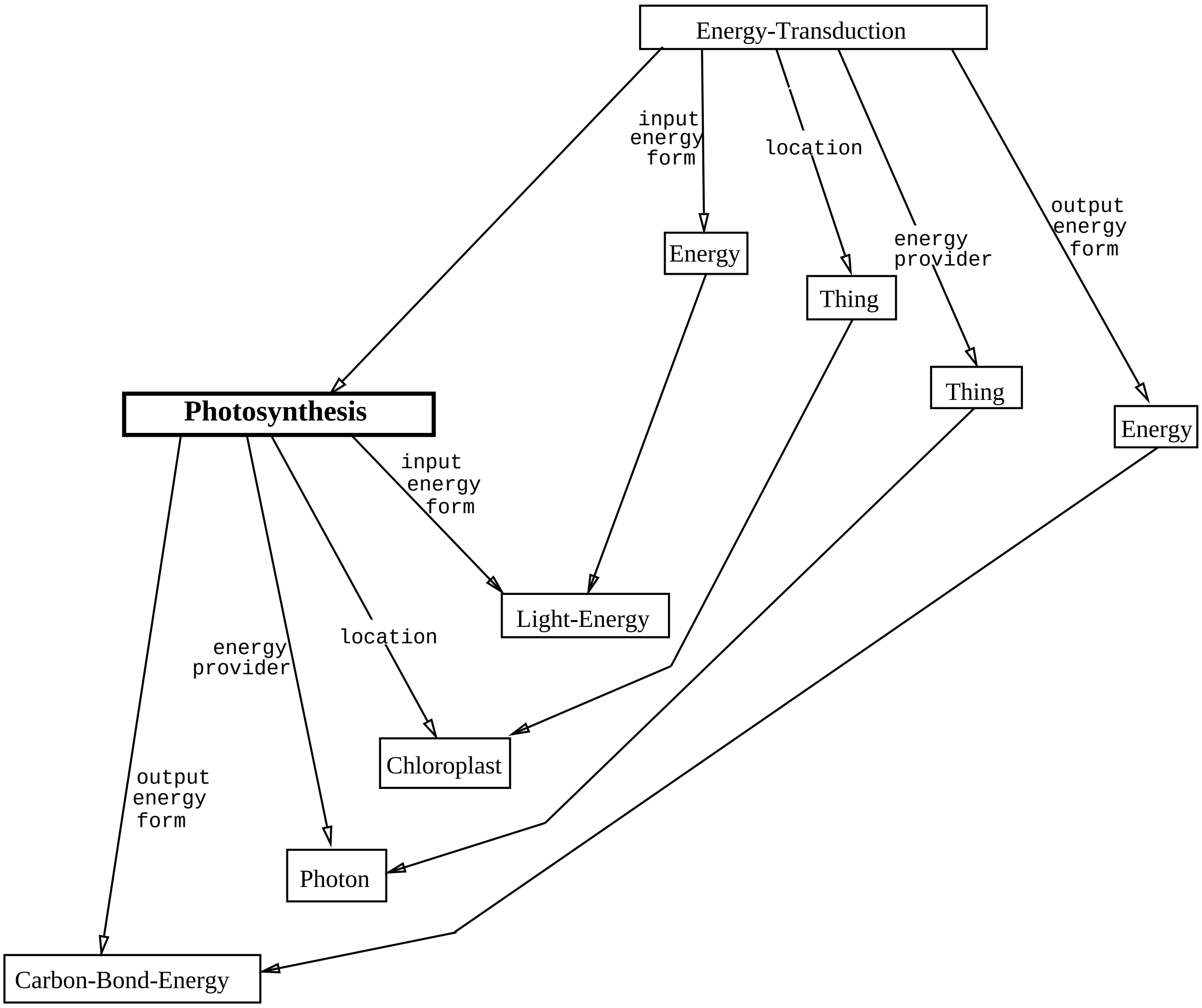

Figure 4.4: The viewpoint of "photosynthesis *as-kind-of* energy transduction," as generated from the Botany Knowledge Base by the View Retriever.

properties of the object that suggest an unspecified part, such as *moisture-content, total-chromosome-number*, and *ovary-position*.

- **Spatial-Superstructural**, which includes the object(s) of which the object is a part, the other sibling parts, and the connections and spatial relations among them.
- **Perceptual**, which includes information regarding how people perceive (see, hear, etc.) the object. This dimension includes the shape, symmetry, size, color, and temperature of the object.
- **Functional**, which includes what the object "does" (the processes in which it is an actor). The functional dimension also includes properties of an object that are suggestive of some unspecified process in which the object is involved, such as *life-span* and *metabolic-rate*. The **Active-Functional** dimension is a subtype of the functional dimension that includes only information about processes in which the object is an active, rather than a passive, actor. (For example, the producer in a production process is an active actor, but the location of the process is a passive actor.)
- **Temporal**, which includes the temporal parts of an object (its stages or states). It also includes, as interconnection relations, the temporal ordering constraints among the stages or states.
- **Temporal-Superstructural**, which includes the objects of which this object is a stage or state, the other sibling stages/states, and their temporal ordering constraints.

Basic dimensions for processes:

- **Behavioral**, which includes the types and roles of the actors in the process and the changes that the process effects upon them. The behavioral dimension also includes initial and final conditions of the process, the relative amounts/sizes of the actors, and the forms of the actors.
- **Procedural**, which includes the steps (subevents) of the process and (as interconnection relations) the temporal ordering constraints among the steps.
- **Event-Superstructural**, which includes the process(es) of which the process is a step, the other sibling steps, and the temporal ordering constraints among them.

Basic dimensions for both objects and processes:

- **Taxonomic**, which includes the subcategories (specializations) of a category, the relative sizes of the subcategories, the criteria for the breakdown, and, as interconnection relations, information about which subcategories are disjoint.
- **Taxonomic-Superstructural**, which includes the generalization(s) of the given category, other categories that share the same generalization(s), their relative sizes, and disjointness relations among them.

- **Modulatory**, which includes information about how one object or process affects other objects or processes. This dimension includes causal relationships (*e.g.* causes, enables, prevents, facilitates), which constitute the subtypes **Causal-Agent** and **Causal-Recipient**, and influences between quantities [24] which constitute the subtypes **Influence-Agent** and **Influence-Recipient**. The latter subtypes include information about the relative strengths of influences, the conditions under which they hold, and their saturation points.

This set of basic dimensions is designed for domains concerned with physical objects and processes. It is a compilation of knowledge types drawn from several sources, including Lakoff and Johnson [39], instructional text analyses [74], coherence relations [2, 35] and major knowledge-engineering efforts [40, 67]. The list must be extended to reflect the types of knowledge found in other kinds of domains and to reflect types of knowledge specific to a particular domain. For example, Lakoff and Johnson suggest two basic dimensions for human artifacts in addition to those given above for objects: *purposive*, which includes information about the human goals that the object was designed to satisfy, and *motor-activity*, which includes information about how people use or interact with the object. For human activities, they suggest the *purpose* dimension in addition to those given above for processes.

To construct viewpoints along basic dimensions, the View Retriever requires knowledge of which slots in the knowledge base are within each dimension. Experience with the Botany Knowledge Base indicates that this knowledge is easily represented directly in the knowledge base, first because the distinctions the basic dimensions make also occur in the slot hierarchy, and second, because it is usually appropriate for a slot to inherit the basic dimension of its generalizations. In the Botany Knowledge Base, each frame that represents a slot has a *slot-dimension* slot to indicate which basic dimension(s) the slot belongs to. If no value is specified for *slot-dimension*, then the slot inherits the *slot-dimension* of more general slots. Most slots belong to exactly one basic dimension, but this is not required by the View Retriever.

Representing knowledge of basic dimensions directly in the knowledge base has two advantages. First, the set of basic dimensions is easily refined and extended. Second, the View Retriever's algorithm for constructing viewpoints is independent of the particular set of basic dimensions used.

4.3.1 Requesting Viewpoints Along Basic Dimensions

The specification for a viewpoint constructed along a basic dimension has two required parameters, the concept of interest and the name of one or more basic dimensions. For example, a *structural* viewpoint of *Seed* has the following specification:

(Seed *dimension* structural)

As with *as-kind-of* viewpoints, the concept of interest can be given by name or by description, and it can be a reified frame, an embedded unit, or a concept reified from the virtual knowledge base by the Creator module of KASTL.

Some basic dimensions have as optional parameters *relation restrictions* and *value restrictions*. When the specification includes a relation restriction, the View Retriever constructs a viewpoint that includes only triples involving slots that are specializations of (or equal to) the given relation. For example,

(Chloroplast *dimension* functional (*relation-restriction* producer))

specifies a viewpoint that includes only information about processes in which a chloroplast is the producer. When the specification includes a value restriction, the View Retriever constructs a viewpoint that includes only triples whose values are specializations of (or equal to) the given concept. For example,

(Xylem *dimension* functional (*value-restriction* Transportation))

specifies a viewpoint that includes only information about transportation processes involving the xylem (the conduit for water flow in a plant).

Another optional parameter is the *partition-by criterion*. Some concepts have multiple viewpoints under a particular basic dimension. This occurs with basic dimensions that include partitionings (*i.e.*, the *structural, temporal, procedural*, and *taxonomic* dimensions). For example, *Plant* has two predominant *structural* viewpoints: one divides plants into roots, stems, leaves, etc., and the other divides plants into the symplast (the living tissues) and the apoplast (the nonliving tissues). To indicate which viewpoint is needed, the user specifies the criterion of the partitioning, in the same way that a user specifies the criterion of a dynamic partitioning (see Chapter 3). For example, the following specification describes the latter viewpoint:

(Plant *dimension* structural (*partition-by* health-state))

If a partitioning having that criterion is not explicitly represented in the knowledge base, then the View Retriever calls on the Creator module of KASTL to create a new partitioning. The View Retriever then returns a viewpoint consisting of the new partitioning. If a concept has multiple partitionings under a requested dimension and the specification does not include a partition-by criterion, then the View Retriever returns a viewpoint containing all the partitionings.

4.3.2 Generating Viewpoints Along Basic Dimensions

The View Retriever constructs a viewpoint along a basic dimension first by retrieving facts about the concept of interest that belong to the basic dimension. For example, to construct a *structural* viewpoint of a plant seed, specified by (Seed *dimension* Structural), the View Retriever first retrieves from the *Seed* frame the values of all slots that belong to the *structural* dimension. A *structural* slot that appears on the *Seed* frame is *has-parts*, so the View Retriever retrieves the following triples, using traditional frame-slot access methods:

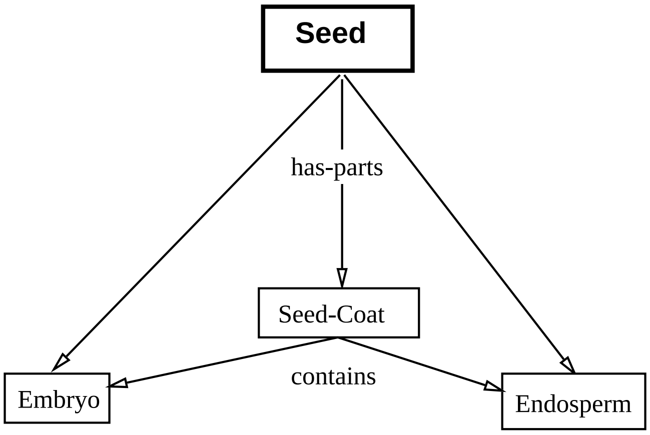

Figure 4.5: A structural viewpoint of *Seed* as generated from the Botany Knowledge Base by the View Retriever.

- $\langle$*Seed has-parts Seed-Coat*$\rangle$,
- $\langle$*Seed has-parts Embryo*$\rangle$, and
- $\langle$*Seed has-parts Endosperm*$\rangle$.

The View Retriever also includes in the viewpoint the annotations that appear on selected triples.

Next, the View Retriever selects interconnection relations for the specified basic dimension. For the *structural* dimension, interconnection relations are *connected-to, contains, surrounds*, etc. The View Retriever looks for these sorts of relationships between the selected parts of the seed: the seed coat, embryo, and endosperm. (The View Retriever finds these relationships even if they are not represented directly on the *Seed* frame.) The View Retriever finds the following triples:

- $\langle$*Seed-Coat contains Embryo*$\rangle$, and
- $\langle$*Seed-Coat contains Endosperm*$\rangle$.

The resulting viewpoint, shown in Figure 4.5, contains the information that the seed is made up of a seed coat containing an embryo and an endosperm.

As another example, consider the viewpoint specified by the following:

((Water (composes Plant)) *dimension* Influence-Recipient)

This specification requests information regarding influences on the water that composes a plant (a *modulatory* viewpoint). The concept of interest, "water that composes a plant,"

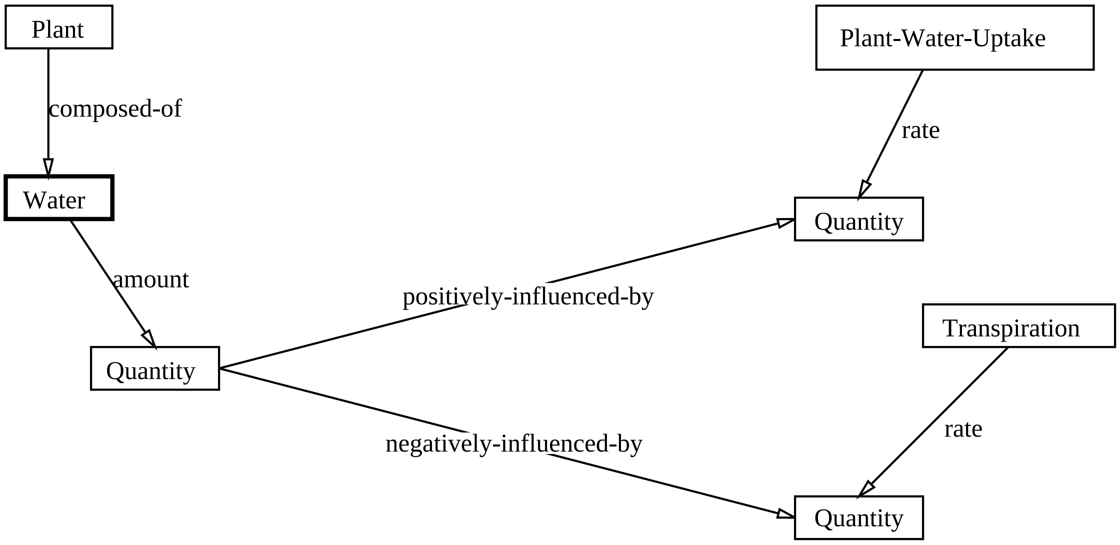

Figure 4.6: A modulatory viewpoint of *(Water (composes Plant))*, as generated from the Botany Knowledge Base by the View Retriever.

is given by description rather than by frame name, so the View Retriever first calls on the Finder/Creator module to locate or reify the concept. In this example the concept happens to be represented in the Botany Knowledge Base as an embedded unit located at the address (Plant composed-of Water). (That is, information about "water that composes a plant" is represented by value annotations on the value *Water* of slot *composed-of* on the *Plant* frame.) The Finder returns the address of the embedded unit to the View Retriever and the View Retriever proceeds to construct the viewpoint.

To construct the viewpoint, the View Retriever selects the features of "water that composes a plant" that are within the *influence-recipient* basic dimension. The only slots within the *influence-recipient* dimension that appear on the embedded unit for "water that composes a plant" are *positively-influenced-by* and *negatively-influenced-by*. These slots do not appear directly on the embedded unit itself. Rather, they appear as annotations on the *amount* slot (see Figure 4.6). Although *amount* is not intrinsically a *modulatory* slot, the View Retriever includes the triple ⟨*(Plant composed-of Water) amount Quantity*⟩ in the viewpoint so that it can also include its annotations that carry modulatory information. (This search for annotations within the specified basic dimension is restricted to the first layer of annotations.) The resulting viewpoint, shown in Figure 4.6, contains the facts that (roughly speaking) the amount of water in a plant is directly proportional to the plant's rate of water uptake and inversely proportional to the plant's rate of transpiration. (This example has no interconnection relations.)

Figure 4.7 gives the procedure the View Retriever uses to construct viewpoints along basic dimensions.

Combining Basic Dimensions with *As-kind-of* Viewpoints

As the previous section mentioned, basic dimensions can be combined with *as-kind-of* viewpoints to enhance their coherence. To do so, the user requests an *as-kind-of* viewpoint as before, except that instead of specifying a reference concept, he specifies a *viewpoint* of the reference concept along some basic dimension(s). For example, to request a viewpoint of photosynthesis as a kind of production, but one that includes only information about the actors in photosynthesis (*i.e.*, information within the *behavioral* basic dimension), the user gives the following viewpoint specification:

(Photosynthesis *as-kind-of* (Production *dimension* Behavioral)

This is the viewpoint actually shown in Figure 4.2. The complete viewpoint of "photosynthesis *as-kind-of* production" additionally includes information from the *procedural* and *modulatory* dimensions, such as the subevents of photosynthesis and what other processes it affects. The viewpoint (Photosynthesis *as-kind-of* Energy-Transduction) shown in Figure 4.4 is likewise restricted to the *behavioral* dimension.

Related Work

Viewpoints created by the View Retriever along basic dimensions are similar to the *perspectives* Suthers suggests for explanation generation [77, 78]. Suthers describes a perspective as "an abstract characterization of the kind of knowledge provided by a class of models." Suthers's set of perspectives appears to be a subset of the basic dimensions given here, although he gives only their names: structural, functional, causal, constraint, and process. Suthers uses perspectives to restrict the information selected for an explanation so as to minimize the number of unfamiliar concepts. A concept is considered familiar if its relation to a known concept lies within a chosen perspective.

Viewpoints constructed along basic dimensions are also similar to the *domain perspectives* McCoy's ROMPER uses to generate corrective responses to users' misconceptions about domain concepts [46, 47]. A domain perspective is a list of slots and their associated *salience values*. A perspective acts as a filter on the attributes of a domain concept; only the values of slots having the highest salience values, as prescribed by that perspective, are included in the response.

ROMPER's perspectives are unlike the basic dimensions given here in that perspectives are specific to the domain of financial securities, but the basic dimensions are generally applicable. The advantage of McCoy's approach is that the slots within a perspective have different degrees of relevance, rather than being simply relevant or irrelevant.

ROMPER does not include a type of perspective analogous to *as-kind-of* viewpoints. Because viewpoints that are tied to the generalization hierarchy (*e.g.*, *as-kind-of* viewpoints) are

```
Given concept of interest C and basic dimension D:

 Initialize the list of values encountered, E, to nil.

 For each slot S legal for C and in dimension D and not having
 an overriding specialization:
   For each value V of (C S):
     Include (C S V) in the viewpoint.
     Include in the viewpoint all annotations stored on (C S V).
     Add V to E (the list of values encountered).

 For each slot S legal for C and not in dimension D:
   For each value V of (C S):
      For each tuple annotation slot T legal for (C S V) and
      within dimension D:
        For each value AV of (C S V T), include in the viewpoint:
           1. (C S V)
           2. (C S V T AV)
           3. all annotations stored on (C S V T AV)

 Include interconnections between encountered values:

 If D is a superstructural dimension
 (spatial-superstructural, temporal-superstructural,
 event-superstructural,  or taxonomic-superstructural),
   For each value V encountered (for each value V in E):
    For each slot S on which V was encountered,
      For each value V2 of (V S-inverse), include in the viewpoint:
          1. (V S-inverse V2)
          2. all annotations stored on (V S-inverse V2)

 If D is a substructural dimension
 (spatial, temporal, procedural, or taxonomic),
   For each value V encountered (for each value V in E):
    For each slot S that is legal for V  and that is defined
    to be an interconnection slot for dimension D,
      For each value V2 of (V S) that is in E (the list of
      values encountered), include in the viewpoint
           1. (V S V2)
           2. all annotations stored on (V S V2)
```

Figure 4.7: Procedure the View Retriever uses to construct viewpoints along basic dimensions. Although this description assumes a single basic dimension in the viewpoint specification, multiple dimensions can be specified. Optional relation restrictions, value restrictions, and partition-by restrictions are omitted here.

insufficient to characterize the breadth of viewpoints that people use, and because McCoy's method for generating domain perspectives (and the method given here for basic dimensions) is at least as powerful as a method for generating *as-kind-of* viewpoints, McCoy rejects the latter in favor of the former. The advantage of including a mechanism for generating *as-kind-of* viewpoints is that they impose fewer demands on the knowledge engineer. The knowledge engineer must explicitly represent the group of slots making up each perspective (and each basic dimension), but the system generates *as-kind-of* viewpoints using only pre-existing domain knowledge. For this reason, the View Retriever includes methods for generating *as-kind-of* viewpoints. It also includes methods for generating *as-having* viewpoints, as the next section describes.

4.4 *As-having* Viewpoints

An *as-having* viewpoint contains information about a concept that is relevant to some specified feature of the concept. For example, the viewpoint "seed coat as having no chlorophyll" contains facts like "a seed coat is usually not green" and "a seed coat is not photosynthetic."

The specification of an *as-having* viewpoint has the following form:

$$(\langle\text{concept of interest}\rangle\ \textit{as-having}\ \langle\text{slot}\rangle\ \langle\text{value}\rangle)$$

As with other types of viewpoints, the concept of interest can be given by name or by description, and it can be a reified frame, an embedded unit, or a concept reified from the virtual knowledge base by the Creator module of KASTL. The *slot* and *value* in the specification indicate the *feature of interest*, the feature to which facts in the viewpoint must be relevant. For example, the viewpoint "seed coat as having no chlorophyll" is specified by

(Seed-Coat *as-having* percent-chlorophyll Zero)

If the feature of interest is not a typical characteristic of the concept of interest, then the View Retriever calls on the Finder/Creator module to locate or reify a new specialization of the concept of interest for which the feature is typical. For example, assume that a user submits the following viewpoint specification:

(Botanical-Cell *as-having* producer-in Photosynthesis)

This specification requests information about botanical cells relevant to the fact that they (sometimes) undergo photosynthesis. Because the feature of interest, *producer-in* = *Photosynthesis*, is not true of most botanical cells, the View Retriever modifies the concept of interest to be *Photosynthetic-Cell*, a specialization of *Botanical-Cell* for which the feature is typical. This modification facility provides users more flexibility in requesting *as-having* viewpoints. That is, users can specify the concept of interest abstractly if it is clear from the rest of the viewpoint specification what concept is intended.

The ideal method for constructing *as-having* viewpoints is to use a theory of relevance to determine what facts about the concept of interest are most relevant to the feature of interest. The View Retriever would compare each fact known about the concept of interest to the feature of interest using a relevance measure, and it would include in the *as-having* viewpoint only the facts judged most relevant. Unfortunately, a general, prescriptive measure of relevance is not yet available. (Hobbs's coherence relations [35, 36] and Mann and Thompson's rhetorical predicates [45] characterize some of the ways in which one fact is relevant to another, but most of these characterize discourse coherence rather than viewpoint coherence. In addition, they are too ill-defined to be directly applied by the View Retriever. Alterman's coherence relations are restricted to relations between events or states [2, 3].) Therefore, to select the facts that constitute an *as-having* viewpoint, the View Retriever depends on stored knowledge of relevance. This knowledge takes the form of viewpoints stored in the knowledge base. These viewpoints are either handcoded by the knowledge engineer or computed and cached by the View Retriever.

To construct an *as-having* viewpoint, the View Retriever first looks for a stored *as-having* viewpoint whose feature of interest is the same as (or more general than) the specified feature of interest, but with a different concept of interest. For example, to construct the following viewpoint:

(Squirrel *as-having* agent-in Seed-Dispersal)

the View Retriever first searches the knowledge base for a similar stored viewpoint, such as one of the following:

- (Mammal *as-having* agent-in Seed-Dispersal),
- (Bird *as-having* agent-in Seed-Dispersal), or
- (Animal *as-having* agent-in Transportation).

Note that although the stored viewpoint's feature of interest must be either the same as or more general than the specified feature of interest, the stored viewpoint's concept of interest need not be related to the specified concept of interest (as with *Squirrel* and *Bird*).

If a similar viewpoint is found in the knowledge base, the View Retriever uses it to determine which facts to include in the new viewpoint. For each feature in the stored viewpoint, the View Retriever looks for a corresponding feature of the specified concept of interest. The way this is done depends on the taxonomic relationship between the concepts of interest of the two viewpoints.

If the stored viewpoint's concept of interest is a generalization of the specified concept of interest, then finding corresponding features involves finding features of the specified concept of interest that specialize features in the stored viewpoint. For example, consider constructing the viewpoint

(Squirrel *as-having* agent-in Seed-Dispersal)

using the stored viewpoint

(Animal *as-having* agent-in Transportation).

If the stored viewpoint contains the fact that animals are usually larger than the things they transport, then the View Retriever would include in the new viewpoint the fact that squirrels are usually larger than the seeds they disperse.

If the concepts of interest of the two viewpoints are *siblings*, as with *Squirrel* and *Bird*, then finding corresponding features is more difficult. It involves finding pairs of features that share a common abstraction, such as ⟨*Bird mode-of-travel Flight*⟩ and ⟨*Squirrel mode-of-travel Walking*⟩. This matching task is similar to the task of constructing an analogy, given the target and base concepts and the relevant features of the base concept [25].

If the View Retriever does not find a similar viewpoint in the knowledge base from which to construct the requested viewpoint, then the View Retriever determines what other type of viewpoint includes the feature of interest and returns that viewpoint. This involves determining which basic dimension includes the feature of interest and constructing a viewpoint of the concept of interest along that basic dimension. For example, to construct

(Squirrel *as-having* agent-in Seed-Dispersal)

the View Retriever recognizes that the slot *agent-in* belongs to the *functional* basic dimension (by retrieving the value of (*agent-in slot-dimension*), so it constructs instead the viewpoint

(Squirrel *dimension* Functional),

which includes information about other activities in which squirrels engage. This method of constructing *as-having* viewpoints takes advantage of the inter-relevance of features within the same basic dimension.

Figure 4.8 gives the procedure the View Retriever uses to construct *as-having* viewpoints.

4.5 Composite Viewpoints

In addition to constructing individual viewpoints as described above, the View Retriever also constructs composite viewpoints. This involves more than simply concatenating the contents of two individual viewpoints. Rather, it involves putting them into correspondence and removing the portions that do not correspond.

The View Retriever constructs three types of composite viewpoints. The first two are *compare* and *contrast*, wherein the View Retriever highlights the similarities or differences between two concepts under a particular viewpoint. For example, the View Retriever can compare a structural viewpoint of a root to a structural viewpoint of a stem, as shown in Figure 4.9. To compare two viewpoints, the View Retriever retains only those features that are common to both viewpoints or that share a common abstraction. For example, one part of *Root* is *Root-Intercellular-Space*, and one part of *Stem* is *Stem-Intercellular-Space*, so the View Retriever includes in the comparison the feature *has-parts* = *Intercellular-Space*.

```
Given a concept of interest C and feature of interest Slot=Val:

If (C Slot Val) is not in the knowledge base,
  find/create a specialization C' of C for which Slot=Val and
  use C' as the concept of interest instead of C.

Look for a similar as-having viewpoint in the knowledge base.

If found, construct a new as-having viewpoint based on the
existing viewpoint.

Otherwise, construct an as-kind-of viewpoint (using C as the
concept of interest and the domain of Slot as the reference
concept) restricted to the basic dimensions under which Slot
is classified.

To find a similar as-having viewpoint in the knowledge base:
Find one having any concept of interest and having feature
of interest Slot'=Val', where either

  1.  Slot' = Slot  and  Val' = Val,
  2.  Slot' = Slot  and  Val' subsumes Val,
  3.  Val' = Val    and  Slot' subsumes Slot, or
  4.  Slot' subsumes Slot  and  Val' subsumes Val.

To construct a new as-having viewpoint based on an existing viewpoint:

 For each slot-value pair (S V) in the existing viewpoint,
   For each common generalization G of the concepts of interest of
   the new and existing viewpoints, if S is legal for G or
   if S originates on some specialization of G,
      For each maximally specific generalization S' of S that is legal for G,
         For each maximally specific value V' of (G S') that subsumes V,
           For each specialization S'' of slot S' that is legal for G
           (or that originates on the path linking G and C)
           and that does not have an overriding specialization,
              For each value V'' of (C S'') that V' subsumes,
                 Include in the new viewpoint (C S'' V'').
```

Figure 4.8: Procedure the View Retriever uses to construct *as-having* viewpoints.

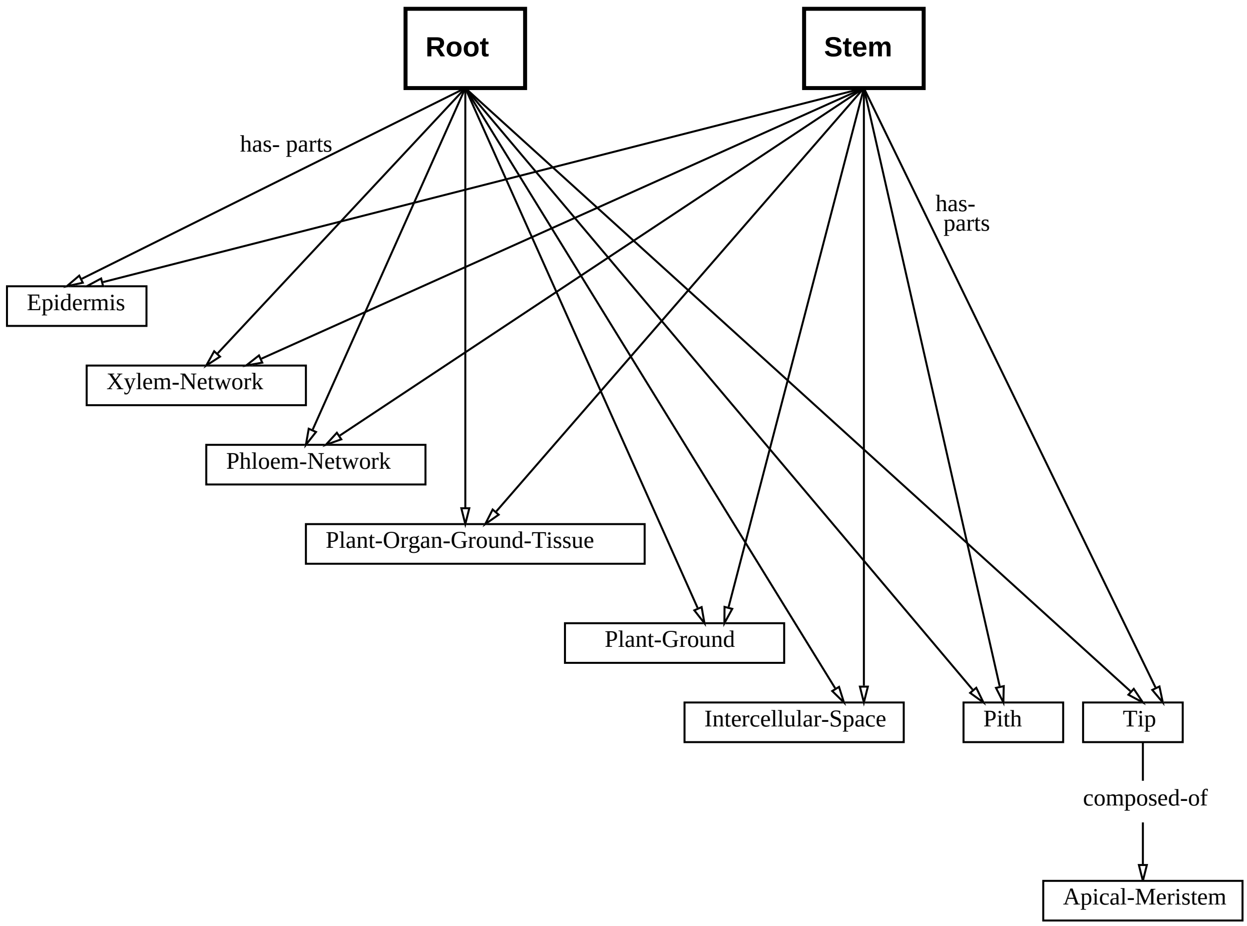

Figure 4.9: Composite viewpoint that compares the structure of a root to the structure of a stem, as generated from the Botany Knowledge Base by the View Retriever.

One part of *Root* that does not occur in *Stem* is *Root-Hair*, so the View Retriever excludes that part from the comparison. Figure 4.10 gives the procedure the View Retriever uses to construct comparison viewpoints. To contrast two viewpoints, the View Retriever retains only the facts that appear in one viewpoint but not the other. Figure 4.11 gives the procedure the View Retriever uses to construct contrast viewpoints. McCoy stresses the importance of adhering to a single type of viewpoint when comparing or contrasting two concepts [47].

Constructing composite viewpoints involves finding the elements of two primitive viewpoints that correspond. For comparisons, equality or similarity determines the correspondence. For contrasts, inequality determines the correspondence. For the third type of composite viewpoint, a relation (*part-of, actor-in*, etc.) determines the correspondence. In this type of composite viewpoint, a group of entities from the first viewpoint all have the same

```
Given two viewpoints, VP1 and VP2,
where VP1 has concept of interest C1:

For each slot-value pair (S V) on viewpoint VP1,

  ;;; Look for identical matches between the viewpoints:
  If V is a value of (VP2 S),
    Then include in the viewpoint:
      1.  (C1 S V)
      2.  a comparison of the annotations on (VP1 S V) and (VP2 S V)

    Else  ;; look for a different value on the same slot and generalize:
      For each value V2 of (VP2 S) not on (VP1 S),
         For each common generalization G of V and V2,
           If G is a valid entry for S, include in the viewpoint:
             1.  (C1 S G)
             2.  a comparison of the annotations on (VP1 S V) and (VP2 S V2)

      ;;; look for the same (or a similar) value on a different slot:
      For each slot S2 on VP2,
        For each common generalization slot SG of S and S2,
          If SG is a true slot, rather than a slot class,
            If V is a value of (VP2 S2), include in the viewpoint:
               1.  (C1 SG V)
               2.  a comparison of the annotations on (VP1 S V) and (VP2 S2 V)

            Else for each value V2 of (VP2 S2),
              For each common generalization G of V and V2,
                 If G is a valid entry for SG, include in the viewpoint:
                        1.  (C1 SG G)
                        2.  a comparison of the annotations on
                            (VP1 S V) and (VP2 S2 V2).
```

Figure 4.10: Procedure the View Retriever uses to construct comparison viewpoints.

```
Given two viewpoints, VP1 and VP2:

For each slot S occurring on VP1,

  For all values V of (VP1 S),

    If V is a value of (VP2 S)

      Then include in the viewpoint any annotations
      on (VP1 S V) and (VP2 S V) that differ.

      Else include in the viewpoint:
         1.  (C1 S V), where C1 is the concept of interest of VP1
         2.  any annotations stored on (VP1 S V)
```

Figure 4.11: Procedure the View Retriever uses to construct contrast viewpoints.

kind of relationship to entities from the second viewpoint. For example, the View Retriever can put a *structural* viewpoint of an object (which describes the object's parts) into correspondence with a *procedural* viewpoint of an event (which describes the event's steps or subevents) along *actor-in* relations. This correspondence links each of the object's parts to the subevent(s) in which it is an actor of some kind. The resulting viewpoint describes the roles that the object's parts play in the subevents of the event. (Paris's "process strategy" generates similar descriptions of how the components of a device enable it to perform some function [64, 65]. The View Retriever generates this and other kinds of composite viewpoints, as described shortly.)

The specification for a composite viewpoint has one of the following forms:

- (*composite* ⟨viewpoint1⟩ ⟨viewpoint2⟩ *compare*),
- (*composite* ⟨viewpoint1⟩ ⟨viewpoint2⟩ *contrast*), or
- (*composite* ⟨viewpoint1⟩ ⟨viewpoint2⟩ ⟨slot-name⟩).

where *viewpoint1* and *viewpoint2* are individual viewpoints (or specifications for them). For the third type of composite viewpoint, *slot-name* is a relation that holds between the concepts of interest of *viewpoint1* and *viewpoint2*. It specifies the correspondence to be established between the viewpoints. The remainder of this section focuses on the third type of composite viewpoint.

Consider again the composite viewpoint that puts a *structural* viewpoint of an object into correspondence with a *procedural* viewpoint of an event along *actor-in* relations. This

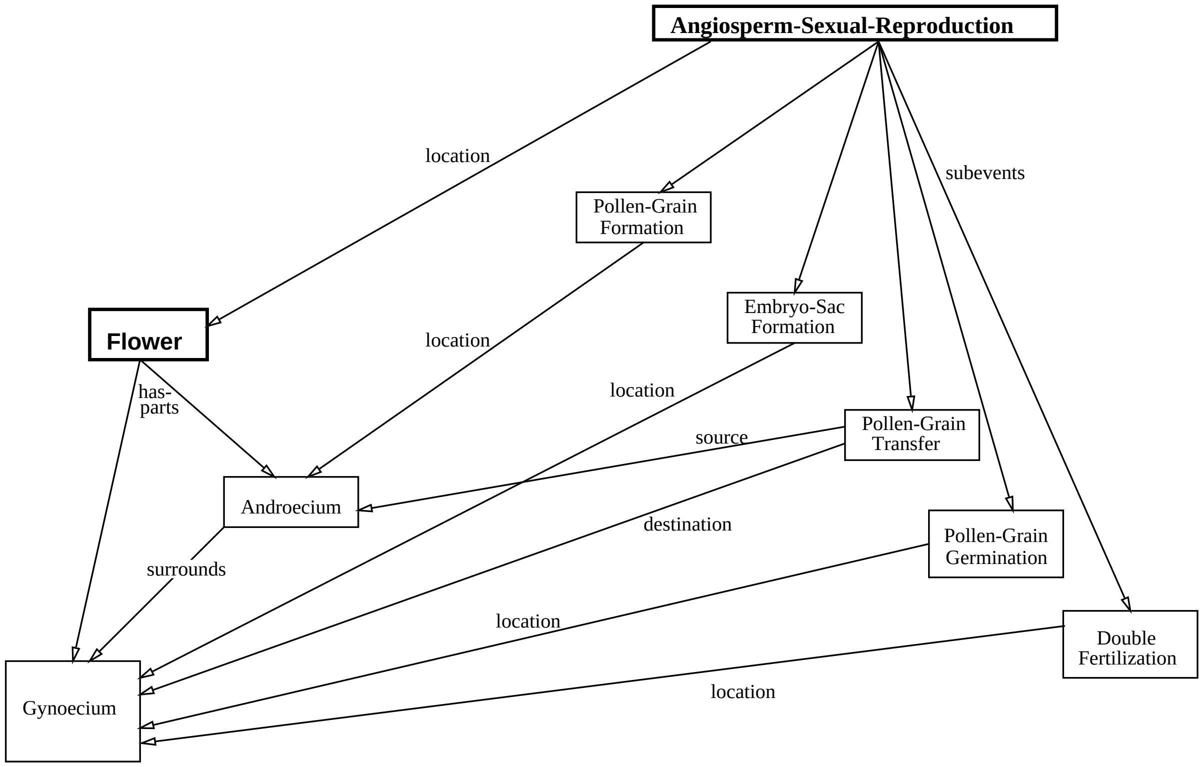

Figure 4.12: The composite viewpoint that describes the roles of the flower's parts in the subevents of angiosperm sexual reproduction. The View Retriever constructed this viewpoint from the Botany Knowledge Base.

composite viewpoint, which describes the function of an object (and its parts) in an event (and its subevents), is requested by the following specification:

(*composite* (⟨object⟩ *dimension* structural) (⟨event⟩ *dimension* procedural) actor-in)

For example, the viewpoint that describes the roles of a flower's parts in the steps of angiosperm (flowering plant) sexual reproduction is specified as follows:

(*composite* (Flower *dimension* structural)
(Angiosperm-Sexual-Reproduction *dimension* procedural)
actor-in)

Figure 4.12 shows the contents of this viewpoint, as generated from the Botany Knowledge Base by the View Retriever.

The View Retriever constructs this composite viewpoint by the following procedure. First, it constructs the two individual viewpoints (the *structural* viewpoint of *Flower* and the *procedural* viewpoint of *Angiosperm-Sexual-Reproduction*). Then, it determines which of the Flower parts in the *structural* viewpoint are related to *Angiosperm-Sexual-Reproduction* or one of its subevents (as given in the *procedural* viewpoint) by an *actor-in* relation (or some more specific relation, such as *location-of*). The View Retriever omits from the composite viewpoint those parts that are not actors in any subevent of *Angiosperm-Sexual-Reproduction*. For example, *Corolla* (the flower's petals) appears in the *structural* viewpoint of *Flower*, but the View Retriever excludes it from the composite viewpoint because the corolla does not participate in reproduction. Similarly, the View Retriever omits those subevents of *Angiosperm-Sexual-Reproduction* that do not involve any of the parts in the *structural* viewpoint of *Flower*, such as *Fruit-Ripening*.

Another example of the third type of composite viewpoint is one that describes the parts of a plant ovary as related to the parts of the fruit of which the ovary is a developmental stage. This viewpoint has the following specification:

(*composite* (Fruit *dimension* structural) (Ovary *dimension* structural) stages)

The composite viewpoint, shown in Figure 4.13 includes the parts of the fruit (the pericarp and the seed), the parts of the ovary (the ovule and the ovarian wall), and the *stage* relations between them, such as the facts that the ovule is a developmental stage of the seed and the ovarian wall is a developmental stage of the pericarp.

Other examples involve two *taxonomic* viewpoints put into correspondence. For example, living things participate in reproduction, so the View Retriever can put different specializations of *Reproducing-Structure* into correspondence with the different specializations of *Reproduction* they engage in. Similarly, angiosperms have flowers, so the View Retriever can put different specializations of *Angiosperm* into correspondence with different specializations of *Flower*. Figure 4.14 gives the procedure the View Retriever uses to construct composite viewpoints.

When constructing two individual viewpoints to be combined into a composite viewpoint, the View Retriever can construct them so as to maximize the correspondence between them. For example, to create the composite viewpoint shown in Figure 4.12 so as to maximize the correspondence between the flower's parts and the subevents of angiosperm sexual reproduction, the View Retriever can, upon request, create a *structural* viewpoint of Flower that includes parts that are not explicitly represented in the knowledge base. Unlike the usual viewpoint, the new *structural* viewpoint divides *Flower* into parts based on the criterion of reproductive function, as in "all the parts of the flower involved in pollen grain development" or "all the parts of the flower involved in double fertilization." The View Retriever constructs this new *structural* viewpoint by calling on the dynamic partitioning facility of the Creator module of KASTL, described in Chapter 3.

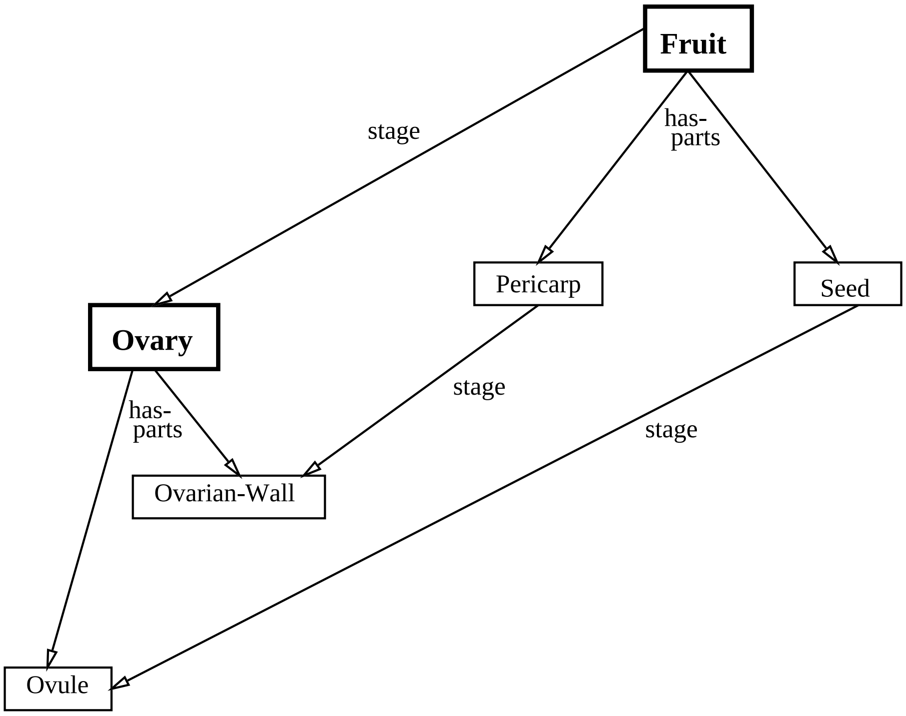

Figure 4.13: The composite viewpoint that describes the *stage* relations between the parts of the fruit and the parts of the ovary. The View Retriever constructed this viewpoint from the Botany Knowledge Base.

```
Given
  1.  viewpoints VP1 and VP2, which have concepts of interest
      C1 and C2, respectively, and which constitute partitionings
      along slots S1 and S2, respectively, and
  2.  correspondence relation R:

Check that C1 is related to C2 by R (or some specialization of R, R').

Include in the viewpoint (C1 R C2)  [or (C1 R' C2)].

For each value V of (VP1 S1),
    If V is related to some value V2 of (VP2 S2) by R or
    some specialization of R,
       Include in the viewpoint (C1 S1 V)
       For each V2 in (VP2 S2) such that (V R' V2),
         where R' is a specialization of R (or is equal to R),
         include in the viewpoint:
            1.  (V  R' V2)
            2.  (C2 S2 V2)

Record interconnection relations among retained values:

For each value V of (VP1 S1),
   If (C1 S1 V) has been included in the composite viewpoint,
      For each annotation (AS AV) stored on (VP1 S1 V),
          If (C2 S2 AV) has been included in the viewpoint,
             Then include in the viewpoint (V AS AV).
```

Figure 4.14: Procedure the View Retriever uses to construct composite viewpoints.

4.6 Related Work

This section discusses related work in two areas: dynamically generating multiple types of viewpoints, and representing viewpoints in a knowledge base.

4.6.1 Generating Viewpoints

This chapter's introduction mentioned several systems that use viewpoints to support a particular application. Previous sections have already discussed those that generate viewpoints of one of the three major types. This section discusses two systems, TEXT and KI, that are not limited to viewpoints of a single type.

The TEXT System

McKeown's TEXT is a system that answers questions about the structure of a database [48]. TEXT treats the database schema as a source of domain knowledge and uses it to generate definitions, descriptions, and comparisons of domain concepts.

To determine the content of a response to a question, TEXT first constructs a *relevant knowledge pool*, all the information that is potentially relevant. TEXT then selects propositions from the relevant knowledge pool one at a time by following a *schema* of common text patterns.

McKeown recognizes that to convey information about several properties of an entity, the system should not generate an arbitrary list of properties. To ensure coherence, the system should group together "properties that are in some way related to each other." Stated in the terminology used here, coherence requires organizing information according to viewpoints.

In an effort to structure knowledge according to viewpoints, TEXT uses the following heuristic (called a *focus constraint*) for selecting the next proposition from the relevant knowledge pool: to decide what to say next about some entity, choose the proposition with the greatest number of links to previously selected propositions. Links between propositions can be explicit (as when two propositions mention the same entity) or implicit (as when an entity mentioned by one proposition is a specialization or part of an entity mentioned by another proposition). The result is a sequence of propositions in which each proposition is related to some preceding proposition.

Although the above heuristic increases the coherence of the responses TEXT generates, selecting propositions one at a time is not an optimal strategy for achieving viewpoint coherence. Because viewpoints often overlap, TEXT may unknowingly progress from one viewpoint to another related viewpoint without completing the first viewpoint. This results in fragmented or incomplete viewpoints, which degrade the coherence of the response. Using complete viewpoints, rather than atomic propositions, as the building blocks of a response yields greater coherence.

The KI System

Murray's KI is a tool for assisting knowledge enterers in making extensions to a large knowledge base [54, 57]. Given a proposed knowledge-base extension, KI identifies how the new information violates or conforms to expectations arising from the existing knowledge.

KI uses *views* to constrain the search for consequences of new information by applying inference methods to only the knowledge within the selected view(s). Murray characterizes views as sets of propositions that interact in some significant way and should therefore be considered together. Some of KI's views are analogous to *as-kind-of* viewpoints (*e.g., Qua Container* and *Qua Tangible Object*). Other are analogous to *as-having* viewpoints (*e.g., Qua Food Source, Qua Product of Reproduction, Qua Component*, and *Qua Developing Object*).

Abstract views (*view types*) are defined by semantic network templates, each represented as a set of paths emanating from a root node. For example, Figure 4.15A shows the view type *Qua Container*. This view type identifies the portion of the knowledge base surrounding a concept that is relevant to its function as a container [56]. KI generates an instantiated view by applying a view type to a domain concept. This involves binding the domain concept to the root node of the view type and instantiating each path of the view type with a portion of the knowledge base surrounding the domain concept. For example, Figure 4.15B shows the result of applying the *Qua Container* view type to the concept *Leaf-Epidermis*. This view identifies the portion of the knowledge base that represents the leaf epidermis in its role as a container (of leaf mesophyll). For example, it includes the fact that leaf transpiration transports water vapor from the leaf mesophyll, contained in the leaf epidermis, to the atmosphere outside the leaf.

This research extends KI's ability to dynamically generate viewpoints. Although KI's mechanism for generating views is general-purpose, the set of view types Murray provides is quite limited. They are not intended to provide broad coverage, and many of them are specific to the domain of botany. An emphasis of this work has been to provide a fairly complete set of viewpoint types useful in all physical domains.

4.6.2 Representing Viewpoints

Although the knowledge base from which a viewpoint is generated contains all of the facts the viewpoint comprises, representing the boundaries of the viewpoint itself in the knowledge base is also useful. This allows caching of commonly used viewpoints, and it provides a means for hand-coding viewpoints that are specific to a particular task or domain. Furthermore, storing a viewpoint in the knowledge base allows assertions about the viewpoint to be represented, such as the tasks for which it is likely to be useful or whether a user is expected to be familiar with its contents.

Several researchers have proposed suitable approaches for representing viewpoints within a knowledge base. One of the earliest formalisms for representing viewpoints appeared in the KRL representation language [8]. In KRL, the knowledge engineer describes concepts by comparing them to other, more general concepts. Thus, the KRL formalism is limited to

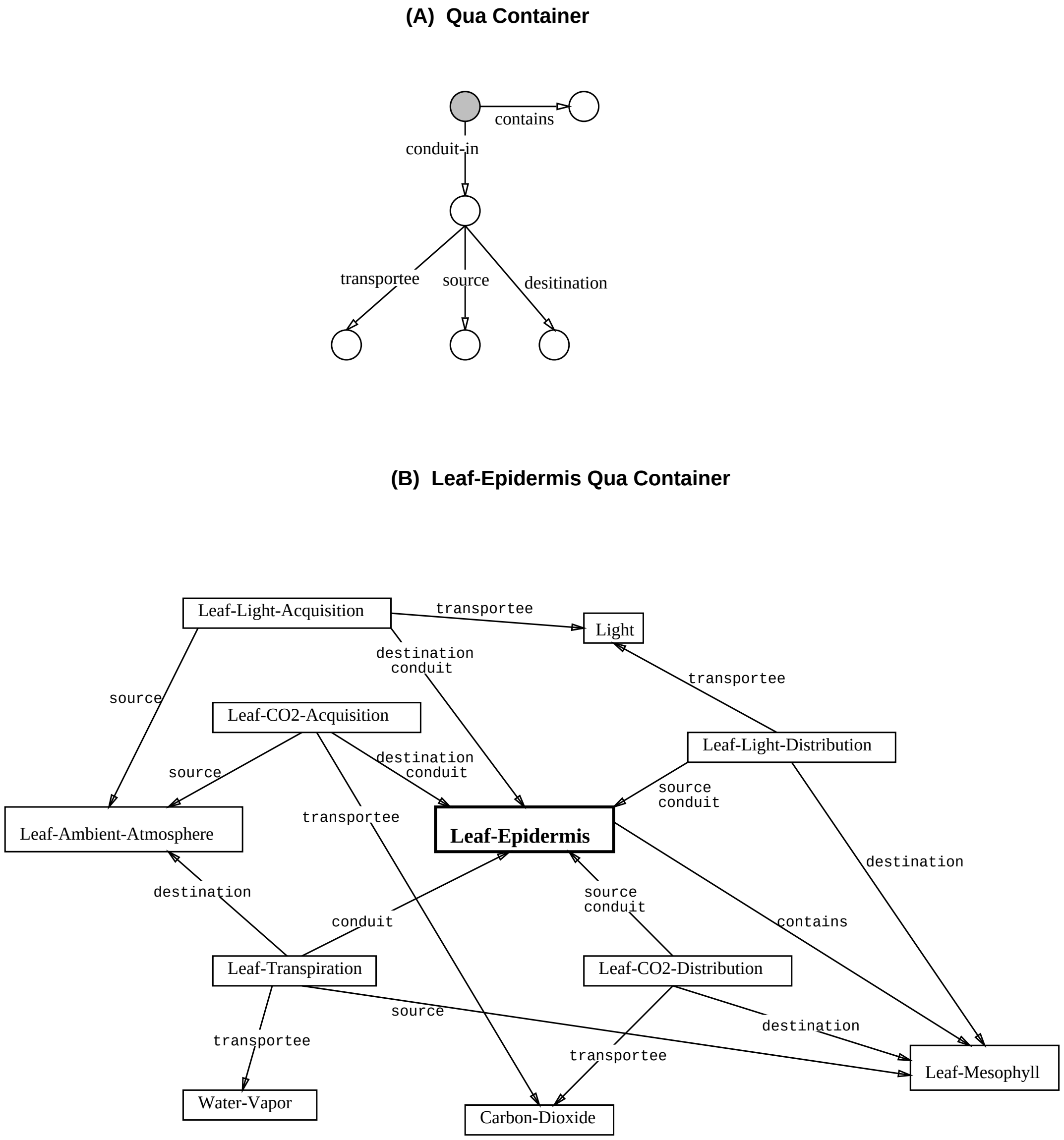

Figure 4.15: (A) View type *Qua Container* identifies the portion of the knowledge base relevant when considering a concept as a container. The shaded node designates the root node. (B) The view *Leaf-Epidermis Qua Container* is the result of applying *Qua Container* to *Leaf-Epidermis*. It consists of the portion of the knowledge base that represents the leaf epidermis in its role as a container of leaf mesophyll.

representing *as-kind-of* viewpoints. KODIAK provides a similar construct [83].

Hendrix later proposed a formalism for partitioning semantic networks into *net spaces* [33]. Hendrix used net spaces to delimit the scope of quantification, to encode alternate worlds and hypothetical situations, and to delimit levels of detail. Grosz extended Hendrix's technique to allow a semantic network to be partitioned in more than one way and to allow net spaces to overlap. This extension makes net spaces suitable for representing "alternate views" of concepts [28], although it is not clear whether the representation allows assertions to be made about each alternate view.

The CycL representation language allows different frames to represent the *multiple models* of a concept, where each model represents a different aspect (viewpoint) of a single object [40]. The CycL approach of reifying viewpoints as first-class objects allows the knowledge engineer to represent multiple models and to represent assertions about them. Sowa [73] and Crawford [19] provide similar facilities for representing viewpoints (which they call *perspectives* and *views*).

Guha presents the most general approach [30, 7, 31]. In his formalism, any group of related assertions can be represented explicitly as a unit, called a *context* (or *microtheory*). Each context has an associated frame that contains the list of assertions within that context as well as assertions about the context (*e.g.*, what assumptions the knowledge engineer made when creating it). Guha presents several uses for contexts, including representing hypothetical situations, representing the same phenomenon using different primitives, different simplifying assumptions, or different levels of detail, and representing the focus of a natural language utterance or problem-solving task. He also suggests using contexts to represent different *perspectives*. Although Guha's notion of a perspective (a description of an event slanted toward one of its actors) is more restrictive than our notion of a viewpoint, his formalism is nonetheless suitable for representing viewpoints. An important contribution of Guha's work is his *lifting axioms* for pulling together information from contexts that use different primitives or different simplifying assumptions.

The representation of viewpoints used in this project is similar to CycL's treatment of multiple models. In addition to frames that represent concepts and slots, the knowledge base also contains *viewpoint frames*. Assertions that constitute a viewpoint are stored on a viewpoint frame in the same notation used to represent assertions on concept frames. Concept frames are linked to viewpoint frames via the slot *has-viewpoints* and its inverse *viewpoint-of*. The type of viewpoint that a viewpoint frame represents is indicated by slots *as-kind-of, as-having, basic-dimensions*, and (for composite viewpoints) *correspondence-type* and *correspondence-with*. Figure 4.16 shows the frame that represents the structural viewpoint of a seed. (The same viewpoint is shown graphically in Figure 4.5.)

The main difference between the representation of viewpoints used in this project and CycL's representation of multiple models is the use of value annotations. In the representation used here, triples in a viewpoint that do not directly involve the concept of interest of the viewpoint are stored as value annotations on triples that do involve the concept of interest. For example, in Figure 4.16, the *contains* relationships between the parts of the

Viewpoint41

instance-of: Viewpoint

viewpoint-of: Seed

basic-dimensions: Structural

has- parts: Seed-Coat Embryo Endosperm

Seed-Coat
contains: Embryo Endosperm

Endosperm
contained-in : Seed-Coat

Embryo
contained-in : Seed-Coat

slot-dimension: Structural

Figure 4.16: The frame that represents the structural viewpoint of a seed. The same viewpoint is shown graphically in Figure 4.5.

seed are represented by value annotations on the triples ⟨*Viewpoint41 has-parts Seed-Coat*⟩, ⟨*Viewpoint41 has-parts Embryo*⟩, and ⟨*Viewpoint41 has-parts Endosperm*⟩. Using value annotations allows all the assertions that constitute the viewpoint to be localized on a single frame.

A disadvantage of this method of representing viewpoints is that it introduces redundancy into the knowledge base. In addition to appearing on concept frames, assertions also appear on (potentially several) viewpoint frames. Redundancy is problematic because it wastes space and because it may lead to inconsistencies in the knowledge base. To avoid inconsistencies, cached viewpoints must be discarded or recomputed following any modification of the concept frames from which those viewpoints were constructed.

Another disadvantage of this method is that, although determining which assertions are in a given viewpoint is straightforward, determining which viewpoints contain a given assertion requires examining every viewpoint frame that might contain it. One solution to this problem is to annotate each assertion in the knowledge base to indicate the viewpoints containing it, if any. Such annotations would also make avoiding inconsistencies more efficient. When an assertion is modified, only the viewpoints containing it must be discarded or recomputed.

4.7 Evaluation

This section presents two evaluations of the work on generating viewpoints. The first is a subjective analysis of the completeness of the set of viewpoint types. The second is an experiment to assess the quality of the viewpoints the View Retriever generates.

4.7.1 Coverage of Viewpoint Types

The purpose of the first evaluation was to assess the degree to which the viewpoint types developed in this research cover the space of viewpoints that people use and to guide further refinements and extensions of the framework of viewpoint types. Recall that the current framework consists of three basic types of viewpoints (*as-kind-of, as-having*, and *basic dimensions*) and three composite types (*compare, contrast*, and correspondence along a relation). The *basic dimensions* comprise several subtypes, given in Section 4.3. Figure 4.17 summarizes the current framework with the grammar of the viewpoint specification language.

The first evaluation consists of a subjective analysis of an entire chapter on plant physiology from a college-level biology textbook [20]. The content of each paragraph was characterized, as much as possible, according to the types of viewpoints it contained. The analysis considered only portions of the text containing fundamental domain knowledge. In particular, purely rhetorical text (figure references, reminders, organizational aids, etc.) was omitted, as were illustrative examples.

Of the text that was considered, roughly 85% was characterized as composing a type of viewpoint that the current framework includes. The remaining 15% of the text was not characterized for a variety of reasons. Some of the uncharacterized text includes

```
view-specification   ::=  viewpoint-name                                    |
                          (concept DIMENSION dimensions )                   |
                          ( concept AS-KIND-OF view-specification )         |
                          ( concept AS-HAVING slot-name value )             |
                          ( COMPOSITE view-specification view-specification
                                      correspondence-type )


dimensions     ::=  view-dimension      |
                    view-dimension  dimensions

view-dimension ::=  PERCEPTUAL                                                  |
                    FUNCTIONAL {( VALUE-RESTRICTION process )}
                               {( RELATION-RESTRICTION slot-name )}             |
                    ACTIVE-FUNCTIONAL {( VALUE-RESTRICTION process )}
                                      {( RELATION-RESTRICTION slot-name )}      |
                    TAXONOMIC   {( PARTITION-BY criterion )}                    |
                    STRUCTURAL  {( PARTITION-BY criterion )}                    |
                    TEMPORAL    {( PARTITION-BY criterion )}                    |
                    PROCEDURAL  {( PARTITION-BY criterion )}                    |
                    TAXONOMIC-SUPERSTRUCTURAL                                   |
                    SPATIAL-SUPERSTRUCTURAL                                     |
                    TEMPORAL-SUPERSTRUCTURAL                                    |
                    EVENT-SUPERSTRUCTURAL                                       |
                    BEHAVIORAL  {( VALUE-RESTRICTION object )}
                                {( RELATION-RESTRICTION slot-name )}            |
                    modulatory

modulatory     ::=  CAUSAL-AGENT            |
                    INFLUENCE-AGENT         |
                    CAUSAL-RECIPIENT        |
                    INFLUENCE-RECIPIENT

criterion      ::=  slot-name   |  slot-path
slot-path      ::=  ( slots value )
slots          ::=  slot-name slots  |
                    slot-name

correspondence-type ::=  COMPARE  |  CONTRAST  |  slot-name
value               ::=  concept  |  constant
object              ::=  concept
process             ::=  concept
concept             ::=  frame-name  |  embedded-unit-address
```

Figure 4.17: The grammar for the viewpoint specification language, the input language for the View Retriever. The grammar shown here assumes that the Finder and Creator modules of KASTL have already replaced concept descriptions with frame names.

information about scientific method (experiments, hypotheses, theories, etc.) rather than botanical objects and processes. The basic dimensions do not currently characterize this kind of information, but they can be extended to do so. Other passages seem to be included as rhetorical devices (*e.g.*, as evidence of a preceding statement, to relate the discussion to upcoming material, or to correct misconceptions the reader might have). Other passages discuss how a property of some object prevents or enables some process, as in "Because the diameter of the vessels is very small, gas bubbles do not form." The modulatory basic dimension can be easily extended to include this kind of information. One of the uncharacterized passages defines a property ("adventitious") rather than an object or a process. Two passages reflect a more sophisticated contrast technique than the View Retriever possesses: one assesses the degree of difference between two objects, rather than simply listing the differences, and the other gives an explanation of the differences ["An animal requires less water (than a plant does) because ..."]. Other passages were not characterized for reasons that are still not fully understood.

The analysis suggests that most explanations consist of several viewpoints used in concert. (The average was 5.4 viewpoints per paragraph.) For example, the following paragraph

> Stems hold the photosynthetic structures — the leaves — up to the light, conduct water to the photosynthetic cells, and transport sugars from them. The outer surface of a green stem is made up of epidermal cells. The bulk of the stem is ground tissue, which may be divided into an outer cylinder (the cortex) and an inner core (the pith). The ground tissue is largely composed of parenchyma cells but also may contain fibers and sclereids.[2]

consists of a functional viewpoint of *Stem*, followed by a structural viewpoint of *Stem* and a structural viewpoint of *Ground-Tissue*. An important area for future work is identifying prevalent combinations of viewpoints.

Although the results of this analysis are encouraging, they cannot be taken as conclusive evidence of the coverage of the current set of viewpoint types. One reason for this is that the analysis is subjective. A more objective analysis requires an external judge not familiar with the characterization of viewpoints used here. Another reason is that the analysis is necessarily speculative. Identifying viewpoints in the text requires speculating as to the form of the author's knowledge and the plan used to generate the text. This is a difficult task given only the end product and no knowledge of what the author had in mind. Furthermore, authors often omit information or convey it implicitly, so the viewpoints that a text comprises are often incomplete. Partial viewpoints are more difficult to recognize than complete viewpoints.

Despite its limitations, however, the analysis indicates that the framework of viewpoint types developed here makes appropriate distinctions, and the uncharacterized passages suggest important areas for extension and refinement of the current set of viewpoint types.

[2] *Invitation to Biology*, H. Curtis and N. Barnes, Worth Publishers, NY, 1981, p. 309.

4.7.2 Coherence of Viewpoints

The purpose of the second evaluation of this work was to measure the quality of viewpoints the View Retriever generates, as compared to the quality of viewpoints found in human-generated text.

For each of 12 preselected concepts in botany (selected because they are well represented in the Botany Knowledge Base), sets of facts were drawn from 3 sources:

- the View Retriever applied to the Botany Knowledge Base, as provided by the domain expert, and
- a college-level botany textbook [68],
- facts selected randomly from a particular frame in the Botany Knowledge Base, using a random number generator.

The viewpoints ranged in size from 3 to 11 facts. For each concept, textbook passages and random sets of facts were chosen to be roughly the same size as the View Retriever's viewpoint of that concept. The number of viewpoints constructed was 13, the number of textbook passages was 17, and the number of random collections of facts was 23.

Each group of facts (including the textbook passages) was translated manually into "simple English" to normalize presentation style. For example, one of the viewpoints selected was the viewpoint shown in Figure 4.12. This viewpoint was rendered in English as follows:

> The flower is the location of angiosperm sexual reproduction. The two main parts of the flower that are involved in reproduction are the androecium and the gynoecium. (The androecium surrounds the gynoecium.) The gynoecium is the location of embryo sac formation, and the androecium is the location of pollen grain formation. The androecium is the source of the pollen grain transfer, and the gynoecium is the destination. At the gynoecium, pollen grain germination and double fertilization occur.

The textbook passage selected for the same concept was the following:

> Most flowers contain two sets of sterile appendages, the sepals and petals, which are attached to the receptacle below the fertile parts of the flower, the stamens and carpels. The sepals occur below the petals, and the stamens below the carpels. Collectively, the sepals form the calyx and the petals the corolla. Together, the calyx and corolla constitute the perianth ("around the flower").

This passage was simplified as follows:

> Most flowers contain two sets of sterile appendages, the sepals and petals. The sepals and petals are attached to the receptacle below the stamens and carpels. The stamens and carpels are the fertile parts of the flower. The sepals are below

> the petals, and the stamens below the carpels. Collectively, the sepals form the calyx and the petals the corolla. Together, the calyx and corolla constitute the perianth.

The random collection of facts selected from the *Flower* frame in the Botany Knowledge Base consisted of the following triples:

```
((FLOWER FLORAL-SYMMETRY SYMMETRY-VALUE)
 (FLOWER ACQUIRER-IN ACQUISITION)
 (FLOWER PARTS PLANT-GROUND)
 (FLOWER LOCATION-OF METABOLIC-REACTION)
 (FLOWER PARTS PERIANTH)
 (FLOWER DEGENERATED-BY DEGENERATION)
 (FLOWER DEVELOPEE-IN FLOWER-DEVELOPMENT)
 (FLOWER METABOLIC-RATE QUANTITY)
 (FLOWER LOCATION-OF ANGIOSPERM-SEXUAL-REPRODUCTION)
 (FLOWER METABOLIZER-IN METABOLIC-REACTION))
```

This collection of facts was rendered in English as follows:

> Flowers are the site of angiosperm sexual reproduction and metabolic reactions. They acquire materials, develop, metabolize, and degenerate. Flowers require nutrients. Two parts of the flower are the ground tissue and the perianth. Flowers tend to have symmetry.

Ten subjects (senior undergraduates and graduate students from the Botany and Biology Departments of the University of Texas at Austin) judged the coherence of passages from each source. (Each subject received 6 textbook passages, 6 random collections of facts, and 12 viewpoints generated by the View Retriever.) Subjects were given the following instructions:

> Each of the following pages contains a brief passage of text along with its subject. Please judge the coherence of the passage on a scale of 1 to 5. A passage should be scored "1" if it seems no more coherent than a randomly selected group of facts on the subject. A passage should be scored "5" if it is as coherent as a passage of comparable length on the subject from a good textbook.
>
> Limit your consideration to the contents of each passage, and ignore issues of organization and rhetoric (such as writing style, wording, and diction). If you feel that the presentation of the material is poor, give the passage the same score that you would give a passage containing the same information but organized and presented in a better fashion.

Table 4.1 summarizes the subjects' responses. Statistical analysis (using a T-test with 0.95 level of confidence) yields the following results:

Source	*Coherence* *Mean*	*Standard Deviation*
(1) Textbook Viewpoints	4.23	0.56
(2) View Retriever's Viewpoints	3.76	0.74
(3) Degraded Viewpoints	2.86	0.94
(4) Random Collections of Facts	2.62	0.86

Table 4.1: Ten judges rated the coherence of sets of facts from four sources (1=incoherent; 5=coherent). A statistical analysis using the T-test with 0.95 level of confidence shows no significant difference in coherence between sources (1) and (2) or between sources (3) and (4). There is a significant difference between all other pairs.

- The mean coherence of viewpoints from textbooks, averaged across subjects, did not differ significantly from the mean coherence of viewpoints the View Retriever generated.
- The mean coherence of generated viewpoints *did* differ significantly from the mean coherence of random collections of facts drawn from the same frame.

A further study gives additional evidence that the View Retriever generates coherent viewpoints. Along with passages from the three sources described above, the subjects were given passages from a fourth source: viewpoints constructed by the View Retriever and then degraded by replacing some of their facts with randomly selected facts on the same topic. Twenty-eight such degraded viewpoints were constructed, each with between one and seven facts replaced. Of the twenty-eight, each subject received six. Table 4.1 shows the mean coherence score of the degraded viewpoints. Statistical analysis shows a significant difference in the mean coherence of pure viewpoints and degraded viewpoints.

4.8 Summary and Limitations

To generate coherent explanations of domain knowledge, question-answering and advisory systems must select, from all of the available knowledge, collections of facts that are relevant to one another. One way to select coherent portions of knowledge is to access *viewpoints* of concepts, collections of facts that describe a concept from a particular perspective. Different viewpoints provide different presentations of domain knowledge, each appropriate for different users, different system goals, and different dialogue contexts. Accessing the knowledge base at the level of viewpoints allows an explanation generator to concentrate on issues of discourse management, and it facilitates portability [75].

In addition to their utility for explanation generation, viewpoints are also important for a variety of other applications, including natural language processing, compositional modeling, problem solving, default reasoning, and learning. Although viewpoints are crucial for a variety of tasks, existing methods for dynamically generating viewpoints from a knowledge base are limited. This research provides general methods for generating viewpoints. In particular, this work provides

- a framework of viewpoint types that are independent of any domain and task, and
- methods for generating viewpoints of each type, either singly or in combinations.

The current framework of viewpoint types consists of *as-kind-of* viewpoints, viewpoints constructed along basic dimensions, and *as-having* viewpoints. *As-kind-of* viewpoints describe a concept by relating it to a more general concept. The View Retriever constructs *as-kind-of* viewpoints by first selecting relevant features of the concept of interest (features subsumed by some feature of the more general concept), then adding the connections between these features and the more general features. These connections provide the justification for the viewpoint's contents.

As-kind-of viewpoints provide two kinds of filtering. The first filter removes redundant features, those that the concept of interest and the more general concept have in common. The second filter removes irrelevant features, features of the concept of interest that do not fit within the conceptual structure of the more general concept.

The second type of viewpoint is viewpoints constructed along basic dimensions. Basic dimensions are general types of facts, such as facts about an object's structure, function, or appearance, or facts about a process's actors or steps. Facts within the same basic dimension convey similar kinds of information. Basic dimensions are especially useful for providing added focus to *as-kind-of* viewpoints.

The set of basic dimensions given in Section 4.3 provides broad coverage for domains concerned with physical objects and processes. The list must be extended to reflect the kinds of knowledge found in other domains. Because knowledge of basic dimensions is represented declaratively in the knowledge base, however, the View Retriever easily accommodates any set of basic dimensions.

As-having viewpoints include features about the concept of interest that are relevant to a user-specified feature of the concept (the feature of interest). Ideally, the View Retriever would construct *as-having* viewpoints by using a theory of relevance. Unfortunately, a general, prescriptive measure of relevance is not yet available. Therefore, the View Retriever depends on knowledge of relevance stored in the knowledge base in the form of cached viewpoints. The View Retriever looks for a cached *as-having* viewpoint similar to the requested viewpoint and uses this viewpoint to determine which facts to include in the new viewpoint.

If the View Retriever does not find a similar viewpoint, then the View Retriever determines what other type of viewpoint includes the feature of interest and returns that viewpoint. This method takes advantage of the inter-relevance of features within the same basic dimension.

In addition to constructing individual viewpoints, the View Retriever also constructs composite viewpoints. This involves more than simply concatenating the contents of two individual viewpoints. Rather, it involves putting them into correspondence and removing the portions that do not correspond. The View Retriever constructs three types of composite viewpoints. The first two are *compare* and *contrast*, wherein the View Retriever highlights the similarities or differences between two concepts under a particular viewpoint. In the third type of composite viewpoint, correspondence is determined by a knowledge-base relation, such as *part-of* or *actor-in*.

A user or application program specifies a viewpoint by indicating the type of viewpoint required and the concept of interest. Figure 4.17 shows the grammar for the viewpoint specification language. This specification language allows the concept of interest to be a concept represented either by a frame or by an embedded unit. Furthermore, the concept of interest can be specified by description or by name, and it can be a concept in the actual or virtual knowledge base. For example, in the following viewpoint specification

((Water (composes Plant)) *dimension* Influence-Recipient)

the concept of interest ("the water that composes a plant") is given by description rather than by name, and that concept is represented in the knowledge base by an embedded unit rather than by an explicit frame. (Figure 4.6 shows the viewpoint that the View Retriever constructs given this specification.) The Finder and Creator modules of KASTL replace concept descriptions in viewpoint specifications with frame names (or embedded unit addresses) as a preprocessing step, using the methods described in Chapter 3.

This work includes two evaluations of the methods developed for generating viewpoints. The first is an analysis to assess the completeness of the current set of viewpoint types and to guide further refinements and extensions. Although the analysis is subjective and speculative, its results suggest that the framework of viewpoint types developed here provides broad coverage for physical domains such as botany. Limited coverage is the major limitation of past work on generating viewpoints [46, 47, 54, 57, 77, 78, 48, 49]. The analysis also suggests important directions for future work. For example, the textbook analysis reveals that most explanations consist of several viewpoints used in concert. Although the View Retriever can construct composite viewpoints, an important area for future work is identifying which combinations are most useful.

The second evaluation of this work assesses the quality of the viewpoints the View Retriever generates. Ten independent judges rated the content coherence of both machine-generated viewpoints and viewpoints taken from a textbook. The results of this evaluation indicate that viewpoints the View Retriever generates are comparable in coherence to human-generated viewpoints.

One limitation of the View Retriever is that it has no knowledge of which viewpoint(s) of a given concept are the most significant or of which facts within a particular viewpoint are most important. A more sophisticated View Retriever would assist users in requesting salient viewpoints and provide "viewpoint highlighting" to call attention to facts that are especially relevant.

A second limitation of this work is that, although the View Retriever provides general methods for accessing viewpoints, it does not prescribe how to select the viewpoint most appropriate for a particular task. Although this work provides a task independent language for describing viewpoints and task independent methods for generating viewpoints, viewpoint selection requires task specific heuristics. Many of the systems described in Section 4.1.4 include such heuristics. The next chapter discusses early work on heuristics for selecting the most appropriate viewpoint for explanation generation.

A third limitation of this work is the lack of evidence as to the utility of the View Retriever for application programs. Although the utility of viewpoints for applications is apparent, whether a system can benefit from the services of the View Retriever depends on whether the View Retriever generates the types of viewpoints needed. The results of the text analysis described in Section 4.7.1 suggests that the View Retriever is based on a framework of viewpoint types that makes appropriate distinctions, but conclusive evidence of the utility of the View Retriever requires designing and developing application programs that use the View Retriever to access viewpoints. The next chapter describes initial efforts and plans for future work in this direction.

Chapter 5

Summary and Future Work

The plain fact is that there are no conclusions. *Sir James Jeans*

5.1 Summary

The goal of this research is to develop methods for representing and accessing knowledge to support multiple tasks. The specific goals of the research are threefold:

- to provide an expressive and convenient language for representing multifunctional knowledge,
- to insulate users of the knowledge base from the effects of (sometimes arbitrary) decisions of knowledge representation, and
- to provide access to coherent portions of knowledge about a given concept (*viewpoints*).

5.1.1 Multifunctional Knowledge Representation

The motivations for developing a new representation language were, first, the need for more expressive power than existing frame-based languages provide and, second, the need for constructs that enable convenient representation of common kinds of assertions. These goals reflect the point of view that it is preferable for a representation language to be more convenient for people to use, even if it means that it is less convenient for computer systems to use.

Chapter 2 presents KM, an expressive, frame-based language for representing multifunctional knowledge. KM includes three major extensions that collectively distinguish it from traditional languages. These are constructs for

- representing quantified assertions,
- representing both definitional and nondefinitional statements, and

- representing information contextually.

The first extension to KM allows quantified assertions to be represented with the same ease as ground assertions, as simple ⟨*frame slot value*⟩ triples. This is accomplished by overloading slots (in the same sense that operators of a programming language are sometimes overloaded) with different semantics, depending on the frames and values that a slot relates. Different combinations of categories and noncategory instances give rise to different quantificational patterns. Slots that are overloaded in the same way (and that share the same semantic mapping) are grouped into equivalence classes called *semantic types*. When the semantic type of each slot is explicitly represented, a system can automatically determine the semantics of a particular triple. Slot overloading makes knowledge representation more convenient than with conventional languages, and it allows the representation of several different forms of quantified assertions.

The second extension of KM, *semantic annotations*, provides greater expressiveness by providing constructs for representing both the definitional and assertional components of a description. Definitions are represented using semantic annotations that distinguish between definitionally necessary features and definitionally sufficient features. This distinction allows concepts having partial definitions to be represented. Nondefinitional assertions are also represented using semantic annotations (likelihood, necessity, cue-validity, and uniqueness). By attaching probabilities to these semantic annotations to represent degrees of belief, KM accommodates both defeasible and nondefeasible assertions as well as assertions of graded defeasibility.

The third extension of KM is *value annotations* for representing information contextually. Although value annotations do not add expressive power to the language, they have several advantages:

- they make knowledge representation much more convenient,
- they do not require the use of a rule or constraint language,
- information represented with value annotations is just as accessible as the rest of the information in the knowledge base, and
- the resulting knowledge base is easier to inspect and use, because only the most important domain concepts are reified as frames.

5.1.2 A Content Addressable, Virtual Knowledge Base

The second goal of this research is to insulate users of the knowledge base from the effects of arbitrary decisions made during knowledge representation. One kind of arbitrary decision is the choice of frame names (*e.g.*, *Plant-Stem* vs. *Stem-of-Plant*). KASTL insulates knowledge-base users from the effects of arbitrary frame-name choices by providing *content addressability*.

With content addressability, users can access frames using either the frame name or a description of the concept that the frame represents. When given an access request containing a concept description, KASTL searches the relevant portion of the knowledge base for a frame with a definitional component that matches that description. KASTL then substitutes the name of the matching frame for the description in the access request and passes the request to the appropriate access function.

The advantage of content addressability is that users can access the knowledge base without extensive prior knowledge of how it has been represented. In particular, users can access concepts without knowing the names of all the frames in the knowledge base. They need only know the names of the most general frames and slots (the top level of the taxonomy), and they can access other concepts by describing them in terms of more general frames and slots. Thus, users have more flexibility in requesting information from the knowledge base. Application programs can pass this flexibility on to their users. For example, a question-answering system that accesses a content addressable knowledge base can accept questions whose topics are descriptions of concepts, rather than frame names. This flexibility is crucial for systems whose users are unfamiliar with the knowledge base, such as students using a tutoring system.

Another kind of knowledge representation decision is the choice of which concepts to reify in the knowledge base. This choice depends on the knowledge engineer's subjective judgment of the relative importance of concepts. Because the importance of concepts varies from one task to another, decisions the knowledge engineer makes regarding which concepts to reify in a multifunctional knowledge base are not appropriate for all tasks in all situations.

To insulate users from the effects of this kind of representational decision, KASTL provides access to concepts in the virtual knowledge base. With a virtual knowledge base, concepts that are implicit in the knowledge base are just as accessible as concepts that are explicit (*i.e.*, that have an associated frame). When KASTL receives an access request containing a concept description for which it cannot find a match in the knowledge base, KASTL creates a new frame to represent the described concept and modifies the knowledge base to accommodate it.

The advantage of providing a virtual knowledge base is that users are less vulnerable to the particulars of how knowledge is represented. If users have access only to concepts that are explicitly represented in the knowledge base, then the knowledge engineer's decision not to reify a concept that is important for a particular task limits the user's ability to perform that task. For example, if a question-answering system has access only to the actual knowledge base, then that system can generate answers only to questions about concepts that have been explicitly represented. By providing access to concepts in the virtual knowledge base, an access method allows users to retrieve information about any concept that can be described in terms of other knowledge base concepts, regardless of whether it has been explicitly represented.

Accessing concepts in the virtual knowledge base requires performing automatic classification of the given concept within the knowledge-base taxonomy. Many existing systems perform automatic classification, including most languages in the KL-ONE family [86]. These systems, however, have several limitations. First, many of them limit expressiveness to achieve tractable algorithms for the subsumption step of classification [86]. This results in languages so limited that they are no longer generally useful.

The second limitation of traditional classifiers is that they use ill-characterized subsumption algorithms. Past systems use the extensional definition of subsumption, in which X subsumes Y when the extension of X must be a superset of the extension of Y. Extensional subsumption has been found to be intractable for most languages [86, 58]. As a result, systems that are based on extensional subsumption have retreated to tractable but incomplete algorithms [66]. These algorithms lack a precise specification of what subsumption relationships they detect.

The third limitation of traditional classifiers is that they are based on extensional subsumption, but they are restricted to using only definitional (terminological) knowledge, knowledge that has no assertional import [12, 11]. Extensional subsumption cannot always be computed solely from definitional knowledge.

To address the limitations of existing classifiers, KASTL is based on the *intensional subsumption* criterion Woods gives [85]. Intensional subsumption means that X subsumes Y when the definition (intension) of X is more general than the definition of Y. Intensional subsumption makes possible a well-characterized classification algorithm without limiting the expressiveness of the representation language.

KASTL performs both the tasks of accessing concepts by description and accessing concepts in the virtual knowledge base by using a single module with a single user interface. This has two advantages. First, the procedure that reifies a concept in the virtual knowledge base can use information gathered while attempting to find that concept in the actual knowledge base, making the system more efficient. Second, users of KASTL do not need to know whether concepts exist in the actual knowledge base; they simply provide a description of the concept. If KASTL fails to find a frame representing that concept, it automatically creates one. Users do not need to know or specify whether they are accessing existing concepts by description or accessing concepts in the virtual knowledge base.

In addition to dynamically reifying concepts that are in the virtual knowledge base, KASTL also performs dynamic partitioning. For a particular partitioning slot (*specializations, has-parts*, etc.), the knowledge engineer typically represents only a few of the possible partitionings. Application programs may need some of the unrepresented partitionings to support some task. For example, a tutoring system generating a description of the different ways that a leaf acquires glucose throughout its lifetime would need a partitioning of *Leaf* into stages according to glucose acquisition method. KASTL allows users to access partitionings that are not explicit in the knowledge base by dynamically creating new partitionings.

To summarize, KASTL makes users of a knowledge base less vulnerable to the particulars of how knowledge is represented by

- providing content addressability,
- providing access to concepts in the virtual knowledge base, and

- providing dynamic partitioning.

These facilities make it easier for users to access the concepts that are relevant to a particular task. Once the relevant concepts have been found, users must then determine what information about those concepts is relevant. KASTL assists users in selecting relevant information by providing access to *viewpoints* of concepts.

5.1.3 Accessing Viewpoints of Concepts

The third goal of this research is to provide access to *viewpoints* of concepts. A viewpoint is a coherent collection of facts that describes a concept from a particular perspective (*e.g.*, a structural viewpoint of *Seed-Coat*, a viewpoint of *Seed-Coat* as a kind of container, a viewpoint of *Seed-Coat* as having no chlorophyll).

Viewpoints are essential for a variety of tasks. Explanation-generation, advisory, and tutoring systems depend on viewpoints to ensure the coherence of the explanations they generate [43, 46, 47, 49, 78, 64, 65, 79, 51, 52]. Learning systems also use viewpoints. For example, KI uses *views* to constrain the search for consequences of adding new information to a knowledge base [54, 57], and Shrager uses *views* to guide incremental changes to a learner's theory of how a device works so that only coherent theories are learned [72]. Other systems use viewpoints to constrain automated reasoning. For example, Falkenhainer and Forbus use *perspectives* in compositional modeling to ensure consistent modeling assumptions and to increase efficiency [23]. ISAAC [62] and APEX [38] use viewpoints in solving physics problems. BLAH [82] and Algernon [19] use *partitions* and *views* to constrain problem solving and default reasoning. Finally, systems use viewpoints for natural language processing. For example, Grosz uses *focus spaces* to guide disambiguation in discourse understanding [28], and KING uses *views* to guide linguistic and conceptual choices in natural language generation [37].

Although viewpoints are crucial for a variety of tasks, existing methods for dynamically generating viewpoints from a knowledge base are limited. This research provides general methods for generating viewpoints. In particular, this work provides, first, a framework of viewpoint types that are independent of any particular domain and task, and second, methods for generating viewpoints of each type, either singly or in combinations.

The current framework of viewpoint types consists of

- *as-kind-of* viewpoints, which describe a concept by relating it to a more general concept. For example, the viewpoint of *Seed-Coat* as a kind of *Container* is an *as-kind-of* viewpoint.

- viewpoints constructed along *basic dimensions*, which describe particular kinds of features of a concept (structural features, functional features, etc.). An example is a structural viewpoint of *Seed-Coat*. Section 4.3 gives a set of basic dimensions that provides broad coverage for physical domains. The View Retriever easily accommodates basic dimensions for other kinds of domains as well.

- *as-having* viewpoints, which include features about a concept that are relevant to a user-specified feature of the concept. For example, the viewpoint of *Seed-Coat* as having no chlorophyll is an *as-having* viewpoint.

In addition to constructing individual viewpoints of these three types, the View Retriever also constructs composite viewpoints. This involves more than simply concatenating the contents of two individual viewpoints. Rather, it involves putting them into correspondence and removing the portions that do not correspond. The View Retriever constructs three types of composite viewpoints. The first two are *compare* and *contrast*, wherein the View Retriever highlights the similarities or differences between two concepts under a particular viewpoint. In the third type of composite viewpoint, correspondence is determined by a knowledge-base relation, such as *part-of* or *actor-in*.

This work includes two evaluations of the methods developed for generating viewpoints. The first is an analysis to assess the completeness of the current set of viewpoint types and to guide further refinements and extensions. Although the analysis is subjective and speculative, its results suggest that the framework of viewpoint types developed here provides broad coverage for physical domains such as botany. Limited coverage is the major limitation of past work on generating viewpoints [46, 47, 54, 57, 77, 78, 48, 49]. The analysis also suggests important directions for future work.

The second evaluation of this work assesses the quality of the viewpoints the View Retriever generates, as compared to the quality of viewpoints found in human-generated text. In this experiment, biologists and botanists judged the coherence of text passages consisting of viewpoints the View Retriever generated and viewpoints taken from textbooks. The results of this experiment show no statistically significant difference in the mean coherence of viewpoints the View Retriever generates and the mean coherence of human-generated viewpoints.

5.2 Future Work

This section discusses two predominant areas for future work. The first is extending the set of access methods developed here to include new paradigms for knowledge access. The second is designing application programs that take advantage of these access methods to perform knowledge intensive tasks.

5.2.1 New Paradigms for Knowledge Access

The two traditional access methods for frame-based knowledge bases are frame-slot access, wherein the user specifies a (*frame slot*) pair and the system returns the value(s) of that slot on that frame, and the second is frame access, wherein the user specifies a frame name and the system returns all the ⟨*frame slot value*⟩ triples stored on that frame. This research extends traditional methods to include methods for

- accessing concepts by description,
- accessing concepts in the virtual knowledge base,
- accessing partitionings in the virtual knowledge base, and
- accessing viewpoints of concepts.

One area for future research is to further expand this set with new paradigms for knowledge access.

Model Composition

A relatively new paradigm of knowledge access that researchers are investigating is model composition. A *model* of an object or process differs from a viewpoint in that a model can be executed (*e.g.*, via numerical or qualitative simulation) to yield predictions. Thus, models typically require fewer kinds of knowledge than viewpoints do (usually only structural knowledge of objects, modulatory relationships between processes, and functional and differential relationships between quantities). *Model composition* is the task of automatically selecting from a knowledge base the information that constitutes the minimal model adequate for a particular task.

As with viewpoint construction, the goal of model composition is to identify all and only the domain knowledge that is pertinent to a particular task. If a model includes too much information, then execution of the model (*i.e.*, simulation) will be inefficient and costly. In addition, an overly complex model does not yield a coherent explanation. If a model includes too little information, then it may not be adequate to make the required predictions, or its predictions may be unsound.

Falkenhainer and Forbus have made a significant contribution to model composition [23]. They propose constructing models from fine grained *model fragments*. Each model fragment is conditioned on the set of assumptions that it requires. These assumptions prescribe which domain objects to include in the model, what viewpoints to impose on them, and other simplifying assumptions. Falkenhainer and Forbus's system selects a minimal set of model fragments that constitutes a model of the quantities to be predicted (the *quantities of interest*). By attending to the assumptions accompanying each model fragment, the system ensures that the model it constructs is consistent.

Rickel points out that a major limitation of Falkenhainer and Forbus's method is that it does not always construct an adequate model [70]. For example, consider the prediction question, "How would a decrease in the amount of water in the soil affect the growth rate of a plant?" Because it is possible to reason about growth rate independently of the soil, there is a minimal model of growth rate that excludes the soil (and hence excludes the amount of water in the soil as well). Falkenhainer and Forbus's method will construct such a model, even though satisfactorily answering the given prediction question requires a model that includes the interaction between soil water amount and plant growth rate.

Addressing this limitation of the "minimal model" approach is nontrivial because the system cannot assume that every quantity mentioned in the question should be included in the model. The user may provide a complex scenario, much of which is irrelevant. Users cannot be expected to know what information is relevant to their questions, and automatically determining what is relevant is difficult.

Rickel has proposed research to address this problem [70]. Rickel's approach is to find and exploit *interaction paths*, sets of functional and differential relationships that connect given quantities (*e.g.*, the amount of water in the soil) with quantities of interest (*e.g.*, plant growth rate), describing how they affect each other. Interaction paths guide the selection of objects to include in the model and the selection of an appropriate level of detail at which to model each object and each relationship. By exploiting interaction paths, Rickel's method ensures that the resulting model is adequate for the given prediction task.

Accessing Levels of Detail

Another new paradigm for knowledge access is accessing deeper levels of detail for a given fact. Often a fact is an abstraction of a collection of facts at a finer level of detail. The detailed facts provide an explanation of the more general fact. For example, one can describe a causal relationship between two events at a finer level of detail as a sequence of causal relationships involving intermediate events. Similarly, one can explain a logical implication by a series of logical implications. One can detail functional relationships between quantities either by replacing qualitative relationships with more precise quantitative relationships or by specifying more detailed dynamics at a faster time scale [70].

Although fine grained knowledge bases often contain multiple levels of detail, the boundaries of these levels are usually not explicitly represented. That is, accessing deeper levels of detail usually requires searching the knowledge base. Because a variety of applications require multiple levels of detail, including compositional modeling and explanation generation, an access method capable of retrieving the information that constitutes a deeper level of detail for a given fact would simplify the design of these application programs. In addition, it would enhance their modularity and portability across domains and across representational formalisms.

An access method for levels of detail would perform the following task:
Given:

- a knowledge base,
- a triple, $\langle F\ S\ V \rangle$, and
- the type of detail needed for the given triple,

Return: A collection of one or more triples that provides more detail (of the given type) for the given triple. This collection of triples forms a path through the knowledge base starting at F and ending at V. For example, given a request for causal detail for the triple $\langle$*Plant-Dehydration causes Stoma-Closing*$\rangle$, an access method could return

- ⟨*Plant-Dehydration causes Concentration-of-Guard-Cell-ABA*⟩,
- ⟨*Concentration-of-Guard-Cell-ABA causes Guard-Cell-Potassium-Loss*⟩,
- ⟨*Guard-Cell-Potassium-Loss causes Guard-Cell-Osmosis*⟩,
- ⟨*Guard-Cell-Osmosis causes Guard-Cell-Water-Loss*⟩,
- ⟨*Guard-Cell-Water-Loss causes Guard-Cell-Collapse*⟩, and
- ⟨*Guard-Cell-Collapse causes Stoma-Closing*⟩,

assuming each of these triples was represented in the knowledge base.

The challenge in developing an access method that performs this task is identifying important types of detail (ways that a fact can be refined by other facts). The types of detail mentioned above are causal refinement, logical refinement, qualitative-to-quantitative refinement, and time-scale refinement. The type of detail needed constrains the search through the knowledge base; for each type, only certain classes of slots need to be explored. For instance, to access causal detail (as in the above example), an access method need only traverse relations such as *causes, enables*, and *prevents*. As with basic dimensions for viewpoints, it is likely that the knowledge of which slots are pertinent to each type of detail can be represented directly in the knowledge base, on the frames that represent slots. In this way, the access tool can operate on any knowledge base.

Like viewpoints, levels of detail that users access frequently could be cached in the knowledge base. Mallory is developing a formalism for representing the collection of facts that detail a given fact. (The formalism is currently limited to facts about how an increase or decrease in one quantity causes an increase or decrease in another quantity.) In this formalism, a ⟨*frame slot value*⟩ triple in the knowledge base can have an associated *story* that comprises the triples that explain it. Mallory is also developing methods for presenting these stories in a human-readable form.

5.2.2 Applications

The second area of future work is to design application programs that use the access methods developed here to perform knowledge intensive tasks. Aside from the merits of the application programs themselves, designing them will provide an empirical evaluation of the utility of the access methods.

Knowledge Acquisition

One application of the access methods developed here is knowledge acquisition. Knowledge acquisition is a machine-learning task in which a computer system assists a knowledge engineer in constructing or augmenting a knowledge base.

Extending a knowledge base includes

- adding new concepts to the taxonomy, and
- extending existing concepts by adding new slots and values.

A knowledge-acquisition tool incorporating the access methods developed here can assist the knowledge engineer in these activities by

- suggesting new concepts and new slot values to add to the knowledge base, using knowledge of viewpoint types,
- making concept creation easier by reifying concepts from the virtual knowledge base, and
- making concept extension easier by providing content addressability.

First, a knowledge-acquisition tool can suggest new information to add to the knowledge base, using knowledge of viewpoint types. A problem that knowledge engineers face when constructing a multifunctional knowledge base is deciding what information to add next. Because the knowledge is not tailored to a specific problem-solving task, the knowledge engineer needs another source of guidance to identify gaps in the knowledge base. Viewpoints provide this guidance.

A knowledge-acquisition tool can use viewpoints to guide knowledge entry in the following way. Given a concept to be extended, the system first determines which viewpoint types apply to that concept. For each of these types, the system attempts to construct a viewpoint. If it cannot find some of the relations that the viewpoint type requires, the system suggests that the knowledge engineer add them to the knowledge base. By soliciting information within one viewpoint at a time, the system provides a sense of focus to knowledge entry.

Murray's KI system performs a similar function as it does knowledge integration [55]. When KI attempts to apply a view type to a domain concept, and KI cannot complete the view, then it suggests that the knowledge engineer enter values for the necessary slots. In addition, KI sometimes makes suggestions for appropriate entries of those slots. A knowledge-acquisition tool can generalize this method by augmenting KI's limited set of view types with the more general set of viewpoint types described in Chapter 4.

The second way that a knowledge-acquisition tool can assist in knowledge entry is by making it easier for the knowledge engineer to create new concepts by reifying concepts from the virtual knowledge base. The usual technique for adding a new concept to the knowledge base is for the knowledge engineer to

1. create a new frame to represent the concept,
2. name the new frame,
3. select the generalizations of the new concept,
4. select the specializations of the new concept,

5. install generalization and specialization relations between the new frame and its generalizations and specializations, and

6. install defining properties of the new concept that distinguish it from its generalizations, if applicable.

A knowledge-acquisition tool can simplify this process by performing these steps automatically. Given a description of the concept to be created (in the specification language given in Chapter 3), the system uses the method described in Chapter 3 to reify the concept from the virtual knowledge base and reorganize the existing knowledge base to accommodate it. The knowledge engineer can then proceed to add additional slot values to the new frame.

Third, a knowledge-acquisition tool can make it easier for the knowledge engineer to locate the frame to which new knowledge is to be added, by providing content addressability. To add new knowledge about a particular concept, the knowledge engineer normally specifies the name of the frame representing that concept, and the system presents that frame for editing. If the knowledge engineer does not know or cannot remember the name of the frame, then he must search the knowledge base for it. With content addressability, the knowledge engineer can find the frame easily by describing it (in the specification language given in Chapter 3). Similarly, to specify a frame as the value of a slot on some other frame, the knowledge engineer can give a description, and the system will replace it with the appropriate frame name, using the method described in Chapter 3.

Question Answering

A second application of the access methods developed here is question answering. The question-answering task can be described as follows:
Given:

- a question, and
- a knowledge base,

Do:

- *query interpretation:* translate the question into an unambiguous internal representation grounded in knowledge-base frames and slots,
- *content determination:* select the portion of the knowledge base that constitutes a correct and coherent response,
- *organization:* arrange the information in a linear sequence of facts for presentation, and
- *realization:* translate the information into a form for presentation (*e.g.*, natural language).

Question answering is an important component of intelligent tutoring systems, expert systems, and advisory systems.

Query interpretation involves translating user-supplied descriptions of concepts into names of frames in the knowledge base. For example, given the question "How does the amount of water in the soil around a plant affect the plant's growth rate?" the system must determine that "the soil around a plant" refers to the frame *Plant-Ambient-Soil*. The methods given in Chapter 3 for accessing concepts by description simplify this translation greatly. Furthermore, if a concept described in a user's question is not explicitly represented in the knowledge base, then the system can reify it from the virtual knowledge base. This facility gives the system more flexibility in answering unanticipated questions.

Content determination involves selecting coherent portions of knowledge from the knowledge base. As Chapter 4 describes, accessing viewpoints of concepts is one technique for selecting coherent portions of knowledge. Accessing knowledge at the viewpoint level also increases the modularity and portability of the system. To use viewpoints as the building blocks of a response, a question answerer must solve the following problems:

- determine which sequences and combinations of viewpoints are most useful and most coherent, and
- determine which types of viewpoints are most appropriate for the given question.

One way to determine useful sequences and combinations of viewpoints is to analyze human-generated texts. For example, in the text analysis described in Chapter 4, some patterns of viewpoints are apparent. The most common is *structural-functional-behavioral.* In this sequence of viewpoints, the text first gives a *structural* viewpoint of an object (which describes its parts), followed by a *functional* viewpoint of each part (which describes the processes in which it is an actor), followed by a *behavioral* viewpoint of each process (which describes the other actors in the process). Another common pattern is *behavioral-modulatory.* In this sequence, the text first gives a *behavioral* viewpoint of a process (which describes its actors) followed by a *modulatory* viewpoint of the same process (which describes what processes it causes or enables. The text then describes those processes by another *behavioral-modulatory* viewpoint combination.

Lester has developed a representation for this kind of discourse knowledge, called *Abstract Discourse Plans*, or *ADPs* [41]. An ADP for a given topic includes a list of potential subtopics to be discussed, conditions that govern when to include each subtopic, and instructions on how to create viewpoint specifications to pass to the View Retriever. ADPs also specify how to organize the information that the View Retriever returns. An important area for future work is building a library of domain independent ADPs that capture common patterns of viewpoints found in human-generated text. This effort will serve to further evaluate the coverage of the set of viewpoint types developed here and the extent to which domain independent viewpoint types are sufficient.

Another problem in using viewpoints for question answering is determining which viewpoints are most appropriate for the given question. Presumably, different types of questions require different types of viewpoints. The selection of viewpoints should also be sensitive to the system's model of the user (responses should contain viewpoints that relate new information to what the user already knows) and the preceding dialogue (the viewpoints contained in a response should continue the theme of the preceding discourse).

When knowledge of common patterns of viewpoints is available in the form of ADPs, the problem of selecting the viewpoints appropriate for a given question is reduced to the problem of selecting an appropriate ADP. Lester is developing heuristics for selecting the ADP most appropriate for a given question. These heuristics use the type and the main topic(s) of the given question to index into a library of ADPs stored in the knowledge base. The heuristics are also sensitive to the system's model of the user and the dialogue history.

Once a question-answering system has selected the appropriate viewpoints to constitute the content of the response, it must then organize the material. The knowledge selected during content determination constitutes a network of ⟨*frame slot value*⟩ triples. Natural language, however, has a strictly linear physical representation. Thus, the system must organize the selected knowledge linearly before it translates it into natural language.

Using viewpoints as the building blocks of an explanation simplifies organization. Because the information within a viewpoint forms a coherent whole, the task of organizing a response is reduced to organizing the material within each viewpoint and then imposing a linear ordering on the viewpoints. Lester shows how viewpoints also guide restructuring of an explanation when the initial organization is suboptimal [42].

Finally, the question answerer must translate the response it has constructed into natural language. As demonstrated by the KING natural language generator, the knowledge of which viewpoint the system imposes on concepts in the explanation can guide linguistic and conceptual choices [37].

5.3 Conclusions

A primary goal of artificial intelligence is to develop an artificially intelligent agent capable of performing a variety of knowledge-based tasks. For an agent to have such capabilities, it must first possess a body of knowledge that supports multiple tasks. In other words, it must have multifunctional knowledge. This research provides an expressive and convenient language for representing multifunctional knowledge.

For an agent to perform a variety of knowledge-base tasks, it must also be able to access from its knowledge base the domain concepts that it requires for the task at hand. This research provides methods for accessing the concepts that are needed through content addressability. The agent must also be able to access the knowledge of those concepts that is relevant in the current problem-solving context. This research provides methods for accessing relevant knowledge in coherent groups, called viewpoints.

An important aspect of this research is its broad applicability. Although this work is presented in the context of a frame-based representation language, many of the ideas (particularly those regarding viewpoint access) apply to other representational paradigms as well. Furthermore, any application program can use the knowledge access methods developed in this research, regardless of its task or domain.

Much of the past research in artificial intelligence has focused on problem solving tasks, assuming that the relevant knowledge to support those tasks will be available. This research provides methods for making the relevant knowledge available.

Bibliography

[1] Martin Abadi and Joseph Y. Halpern. Decidability and expressiveness for first-order logics of probability. Technical Report 73, Digital Systems Research Center, June 1991.

[2] Richard Alterman. A dictionary based on concept coherence. *Artificial Intelligence*, 25:153–186, 1985.

[3] Richard Alterman. Event concept coherence. In David Waltz, editor, *Semantic Structures*, pages 57–87. Lawrence Erlbaum, New York, 1989.

[4] Hippocrates G. Apostle. *Aristotle's Metaphysics*. The Peripatetic Press, Grinnell, Iowa, 1979.

[5] Lawrence W. Barsalou. Ad hoc categories. *Memory and Cognition*, 11(3):211–227, 1983.

[6] Lawrence W. Barsalou. The instability of graded structure: Implications for the nature of concepts. In U. Neisser, editor, *Concepts and Conceptual Development: Ecological and Intellectual Factors in Categorization*, pages 101–140. Cambridge University Press, New York, 1987.

[7] Paul Blair, R. V. Guha, and Wanda Pratt. Microtheories: An ontological engineer's guide. Technical Report CYC-050-92, Microelectronics and Computer Technology Corporation, March 1992.

[8] Daniel G. Bobrow and Terry Winograd. An overview of KRL, a knowledge representation language. *Cognitive Science*, 1:3–46, 1977.

[9] Ronald J. Brachman. I lied about the trees. *AI Magazine*, 6(3):80–93, 1985.

[10] Ronald J. Brachman. The future of knowledge representation. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 1082–1092, July 1990.

[11] Ronald J. Brachman, Richard E. Fikes, and Hector J. Levesque. KRYPTON, a functional approach to knowledge representation. In Hector J. Levesque and Ronald J. Brachman, editors, *Readings in Knowledge Representation*, pages 412–429. Morgan Kaufman, Los Altos, CA, 1985.

[12] Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9:171–216, 1985.

[13] Bruce G. Buchanan and Edward H. Shortliffe. *Rule-Based Expert Systems.* Addison-Wesley, 1985.

[14] Jaime R. Carbonell. AI in CAI: An artificial intelligence approach to computer-assisted instruction. *IEEE Transactions on Man-Machine Systems*, 11(4):190–202, 1970.

[15] Jacqueline Castaing. A new formalisation of subsumption in frame-based representation systems. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, pages 22–25. Morgan Kaufmann, April 1991.

[16] Peter Cheeseman. In defense of probability. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 1002–1009, San Mateo, CA, 1985. Morgan Kaufmann.

[17] Peter Cheeseman. Probabilistic versus fuzzy reasoning. In L.N. Kanal and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, pages 85–102. Elsevier Science Publishers, Amsterdam, 1986.

[18] William J. Clancey. The epistemology of a rule-based expert system – a framework for explanation. *Artificial Intelligence*, 20:215–251, 1983.

[19] James Crawford. Access-limited logic—a language for knowledge representation. Technical Report AI90-141, University of Texas at Austin AI Laboratory, September 1990.

[20] Helene Curtis and N. Sue Barnes. *Invitation to Biology.* Worth Publishers, New York, 1981.

[21] Robert deBeaugrande. *Text Discourse and Process.* Ablex, Norwood, NJ, 1980.

[22] Jon Doyle and Ramesh S. Patil. Two dogmas of knowledge representation. Technical Report MIT/LCS/TM-387.b, MIT Laboratory for Computer Science, September 1989.

[23] Brian Falkenhainer and Kenneth D. Forbus. Compositional modeling: Finding the right model for the job. *Artificial Intelligence*, 51:95–143, 1991.

[24] Kenneth D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24:85–168, 1984.

[25] Dedre Gentner. Structure mapping: A theoretical framework for analogy. *Cognitive Science*, 7(2):155–170, 1983.

[26] Joseph E. Grimes. *The Thread of Discourse.* Mouton, The Hague, Paris, 1975.

[27] Benjamin N. Grosof. An inequality paradigm for probabilistic knowledge. In L.N. Kanal and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, pages 259–275. Elsevier Science Publishers, Amsterdam, 1986.

[28] Barbara J. Grosz. The representation and use of focus in a system for understanding dialogs. In B. Grosz, K. Jones, and B. Webber, editors, *Readings in Natural Language Processing*, pages 353–362. Morgan Kaufman, 1986.

[29] Barbara J. Grosz and Candace L. Sidner. Attention, intentions, and the structure of discourse. *Computational Linguistics*, 12(3):175–204, July-September 1986.

[30] R. V. Guha. Micro-theories and contexts in Cyc. Technical Report ACT-CYC-129-90, Microelectronics and Computer Technology Corporation, June 1990.

[31] R. V. Guha. Contexts: A formalization and some applications. Technical Report ACT-CYC-423-91, Microelectronics and Computer Technology Corporation, November 1991.

[32] Michael A. K. Halliday and Ruqaiya Hasan. *Cohesion in English.* Longman, London, 1976.

[33] Gary G. Hendrix. Expanding the utility of semantic networks through partitioning. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pages 115–121, 1975.

[34] Graeme Hirst. Ontological assumptions in knowledge representation. In R.J. Brachman, H.J. Levesque, and R. Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 157–169, San Mateo, CA, 1989. Morgan Kaufmann.

[35] Jerry R. Hobbs. Towards an understanding of coherence in discourse. In Wendy Lehnert and Mark Ringle, editors, *Strategies for Natural Language Processing*, pages 223–243. Lawrence Erlbaum, Hillsdale, NJ, 1982.

[36] Jerry R. Hobbs. On the coherence and structure of discourse. Technical Report CSLI-85-37, Computer Science Department, Stanford University, 1985.

[37] Paul S. Jacobs. KING: A knowledge intensive natural language generator. In Gerard Kempen, editor, *Natural Language Generation*, pages 219–225. Martinus Nijhoff, Dordrecht, The Netherlands, 1987.

[38] Hyung J. Kook. A model-based representational framework for expert physics problem solving. Technical Report AI89-103, Artificial Intelligence Laboratory, The University of Texas at Austin, May 1989.

[39] George Lakoff and Mark Johnson. *Metaphors We Live By.* University of Chicago Press, 1980.

[40] Douglas B. Lenat and R. V. Guha. *Building Large Knowledge Based Systems.* Reading, MA:Addison-Wesley, 1990.

[41] James C. Lester. Abstract discourse plans: A knowledge-based approach to representing discourse knowledge for explanation generation. Working paper, 1991.

[42] James C. Lester and Bruce W. Porter. A revision-based model of instructional multi-paragraph discourse production. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, pages 796–800, August 1991.

[43] James C. Lester and Bruce W. Porter. A student-sensitive discourse generator for intelligent tutoring systems. In *Proceedings of the International Conference on the Learning Sciences*, pages 298–304, 1991.

[44] Hector J. Levesque and Ronald J. Brachman. A fundamental tradeoff in knowledge representation and reasoning. In Hector J. Levesque and Ronald J. Brachman, editors, *Readings in Knowledge Representation*, pages 42–70. Morgan Kaufman, Los Altos, CA, 1985.

[45] William C. Mann and Sandra A. Thompson. Rhetorical structure theory: A theory of text organizations. Technical Report ISI/RS-87-190, Information Sciences Institute, University of Southern California, 1987.

[46] Kathleen McCoy. The role of perspective in responding to property misconceptions. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 791–793, 1985.

[47] Kathleen McCoy. Generating context-sensitive responses to object-related misconceptions. *Artificial Intelligence*, 41:157–195, 1989.

[48] Kathleen R. McKeown. *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text.* Cambridge University Press, New York, 1985.

[49] Kathleen R. McKeown. Generating goal-oriented explanations. *International Journal of Expert Systems*, 1(4):377–395, 1988.

[50] T. Mitchell, J. Allen, P. Chalasani, J. Cheng, O. Etzioni, M. Ringuette, and J. Schlimmer. Theo: A framework for self-improving systems. In Kurt VanLehn, editor, *Architectures for Intelligence.* Lawrence Erlbaum Associates, 1988.

[51] Johanna D. Moore and William R. Swartout. A reactive approach to explanation. In *Proceedings of the Fourth International Workshop on Natural Language Generation*, pages 17–21, July 1988.

[52] Johanna D. Moore and William R. Swartout. A reactive approach to explanation. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1504–1510, 1989.

[53] Gregory L. Murphy and Douglas L. Medin. The role of theories in conceptual coherence. *Psychological Review*, 92(3):289–316, July 1985.

[54] Kenneth S. Murray. KI: An experiment in automating knowledge integration. Technical Report AI88-90, Artificial Intelligence Laboratory, The University of Texas at Austin, October 1988.

[55] Kenneth S. Murray, May 1992. Personal communication.

[56] Kenneth S. Murray. Learning as knowledge integration: A case study. 1992. Presented at the AAAI Spring Symposium on Knowledge Assimilation, Stanford University, C. Elkan, T. Dietterich, O. Etzioni, and B. Selman, organizers.

[57] Kenneth S. Murray and Bruce W. Porter. Controlling search for the consequences of new information during knowledge integration. In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 290–295, San Mateo, CA, 1989. Morgan Kaufmann.

[58] Bernhard Nebel. Computational complexity of terminological reasoning in BACK. *Artificial Intelligence*, 34(3):371–383, 1988.

[59] Eric Neufeld. Defaults and probabilities; extensions and coherence. In R.J. Brachman, H.J. Levesque, and R. Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 312–323, San Mateo, CA, 1989. Morgan Kaufmann.

[60] Hwee Tou Ng and Raymond J. Mooney. On the role of coherence in abductive explanation. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 337–342, Boston, MA, 1990.

[61] Nils Nilsson. Probabilistic logic. *Artificial Intelligence*, 28:71–87, 1986.

[62] Gordon S. Novak Jr. Representations of knowledge in a program for solving physics problems. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 286–291, August 1977.

[63] Gordon S. Novak Jr. and Agustin A. Araya. Physics problem solving using multiple views. Technical Report TR-173, The University of Texas at Austin, March 1981.

[64] Cecile L. Paris. Description strategies for naive and expert users. In *Proceedings of the Twenty-third Annual meeting of the Association for Computational Linguistics*, pages 238–245, 1985.

[65] Cecile L. Paris and Kathleen R. McKeown. Discourse strategies for describing complex physical objects. In Gerard Kempen, editor, *Natural Language Generation*, pages 97–115. Martinus Nijhoff, Dordrecht, The Netherlands, 1987.

[66] Peter F. Patel-Schneider. Undecidability of subsumption in NIKL. *Artificial Intelligence*, 39(2):263–272, 1989.

[67] B. Porter, J. Lester, K. Murray, K. Pittman, A. Souther, L. Acker, and T. Jones. AI research in the context of a multifunctional knowledge base: The Botany Knowledge Base project. Technical Report AI88-88, University of Texas at Austin, 1988.

[68] P. Raven, R. Evert, and H. Curtis. *Biology of Plants.* Worth Publishers, New York, 1976.

[69] Elaine Rich. Default reasoning as likelihood reasoning. In *Proceedings of the Third National Conference on Artificial Intelligence*, pages 348–351, San Mateo, CA, 1983. Morgan Kaufmann.

[70] Jeff W. Rickel. Automated modeling for answering prediction questions: Exploiting interaction paths. Proposal for Dissertation Research, 1992.

[71] Manfred Schmidt-Schaub. Subsumption in KL-ONE is undecidable. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 421–431. Morgan Kaufmann, May 1989.

[72] Jeff Shrager. Theory change via view application in instructionless learning. *Machine Learning*, 2:247–276, 1987.

[73] John F. Sowa. *Conceptual Structures.* Addison-Wesley, Reading, MA, 1984.

[74] Albert Stevens and Cindy Steinberg. A typology of explanations and its applications to intelligent CAI. Technical Report TR 4626, Bolt Beranek and Newman, Inc., 1981.

[75] Daniel D. Suthers. Providing multiple views of reasoning for explanation. In *Proceedings of the International Conference on Intelligent Tutoring Systems*, pages 435–442, 1988.

[76] Daniel D. Suthers. Perspectives in explanation. Technical Report COINS-89-24, Department of Computer Science, University of Massachusetts at Amherst, 1989.

[77] Daniel D. Suthers. The epistemological structure of explanations. In *Proceedings of the AAAI-90 Workshop on Explanation*, pages 178–187, July 1990.

[78] Daniel D. Suthers. Task-appropriate hybrid architectures for explanation. In *Proceedings of the AAAI-91 Workshop on Comparative Analysis of Explanation Planning Architectures*, pages 80–94, July 1991.

[79] William Swartout. XPLAIN: A system for creating and explaining expert consulting programs. *Artificial Intelligence*, 21(3):285–325, 1983.

[80] Paul Thagard. Explanatory coherence. Technical Report 16, Cognitive Science Laboratory, Princeton University, March 1988.

[81] Hudson Turner. Two notions to improve a generic frame language. Knowledge representation class project, 1990.

[82] J. L. Weiner. BLAH, a system which explains its reasoning. *Artificial Intelligence*, 15:19–48, 1980.

[83] Robert Wilensky. KODIAK—a knowledge representation language. In *Proceedings of the Sixth Annual Conference of the Cognitive Science Society*, pages 344–352, June 1984.

[84] William A. Woods. What's in a link? Foundations for semantic networks. In Hector J. Levesque and Ronald J. Brachman, editors, *Readings in Knowledge Representation*, pages 218–241. Morgan Kaufman, Los Altos, CA, 1985.

[85] William A. Woods. Understanding subsumption and taxonomy: A framework for progress. In John F. Sowa, editor, *Principles of semantic networks: Explorations in the representation of knowledge*, pages 45–94. Morgan Kaufmann, San Mateo, CA, 1991.

[86] William A. Woods and James G. Schmolze. The KL-ONE family. Technical Report TR-20-90, Harvard University Center for Reserach in Computing Technology, 1990.