Artificial Intelligence Review **14:** 377–401, 2000.
Stephen J. Hegner et al. (eds.), Intelligent help systems for UNIX.

Representing UNIX Domain Metaphors

JAMES H. MARTIN
Department of Computer Science, University of Colorado at Boulder, ECOT 7-7, Campus Box 430, Boulder, CO 80309-0430, USA; E-mail: martin@cs.colorado.edu

Abstract. The language used to describe technical domains like UNIX is filled with metaphor. An approach to metaphor, based on the explicit representation of knowledge about metaphors, has been developed. MIDAS (Metaphor Interpretation, Denotation, and Acquisition System) is a computer program that that has been developed based upon this approach. MIDAS can be used to represent knowledge about conventional metaphors, interpret metaphoric language by applying this knowledge, and dynamically learn new metaphors as they are encountered during normal processing.

Keywords: knowledge representation, metaphor, natural language processing

1. Introduction

The language used to describe technical domains like UNIX is filled with metaphor. A consultant system that is going to accept natural language input from users, and provide appropriate natural language advice, must be prepared to handle such metaphorical language. Consider the following UNIX examples.

(1) How can I *kill* a process?
(2) I am *in* emacs.
(3) How do I *get out of* emacs?
(4) You can *enter* emacs by typing emacs to the shell.
(5) I want to *give* someone *permission* to read a file.

The italicized words in each of these examples are being used to metaphorically refer to concepts that are quite distinct from those that might be considered the normal meanings of the words. Consider the use of *enter* in Example 4. *Enter* is being used, in this example, to refer to the actions on a computer system that result in the activation of a program. This use is clearly different from what might be called the ordinary or basic meaning of the word that has to do with the actions that result in an agent entering an enclosure.

While the word *enter* is used metaphorically in Example 4, this metaphor is neither novel nor poetic. Instead, the metaphorical use of *enter* results from

a conventional, systematic, conceptual metaphor that allows computer processes to be viewed as enclosures. The various actions and states that have to do with containment are used to refer to actions and states that have to do with the activation, deactivation, and use of these computer processes. This conceptual metaphor, structuring processes as enclosures, underlies the normal conventional way of speaking about these processes. Therefore, the uses of the words *enter* in Example 4, *in* in 2 and *get out of* in 3 are ordinary conventional ways of expressing these concepts that nevertheless involve this enclosure metaphor.

The approach to conventional metaphor described in this chapter is a knowledge-intensive one. This approach was inspired by the work of Lakoff and Johnson (1980), and builds on the computational approaches begun in (Jacobs 1985; Martin 1986, 1987; Norvig 1987) and (Wilensky 1986). This approach asserts that the interpretation of conventional metaphoric language proceeds through the direct application of specific knowledge about the metaphors in the language. The interpretation of novel metaphors is accomplished through the systematic extension, elaboration, and combination of already well-understood metaphors. The proper way to approach the topic of metaphor, therefore, is to study the details of both individual metaphors and the system of metaphors in the language.

This approach has been embodied in MIDAS (Metaphor Interpretation, Denotation, and Acquisition System) (Martin 1988). MIDAS is a set of computer programs that can be used to perform the following tasks: explicitly represent knowledge about conventional metaphors, use this knowledge to interpret metaphoric language, and learn new metaphors as they are encountered.

In order to make the problem of understanding metaphors more concrete, consider the following session with the UNIX Consultant (Wilensky et al. 1988), involving an enclosure metaphor.

```
> (do-sentence)
Interpreting sentence:
 How can I get into lisp?

  Applying conventional metaphor Enter-Process.

UC: You can get into lisp by typing lisp to the shell.
```

In this example, the user has employed the conventional metaphor, described above, that entails that programs can be viewed as enclosures or environments. The action of entering such an enclosure, underlying the phrase *get into*, corresponds to the action that begins the use of the program.

In order to appropriately handle this example, UC must be able to access and apply specific knowledge about this conventional metaphor.

UC handles this kind of metaphoric language by calling upon MIDAS. In this example, UC calls upon MIDAS to find a coherent interpretation for this use of *get into*. MIDAS finds and applies the conventional metaphor that allows the invocation of a program to be viewed as an entering.

Section 3 focuses on the structure of individual metaphors and the systematicities evident among some of the important core metaphors in UNIX. Section 4 describes how these systematic metaphors can be represented using the KODIAK representation language (Wilensky 1986). Section 5 describes how MIDAS can use this metaphoric knowledge to interpret known conventional metaphors. Finally, Section 6 shows how MIDAS can use the systematic structure of known metaphors to dynamically learn new metaphors as they are encountered.

2. Previous Computational Approaches to Metaphor

The metaphoric knowledge approach, described here, is an attempt to fuse the notion of a systematic language convention with the notion of analogical systematicity that has been at the core of the most previous computational work on metaphor (Carbonell 1981; DeJong and Waltz 1983; Fass 1988; Gentner et al. 1988; Indurkhya 1987). These approaches assert that metaphors like those cited above arise solely from an underlying conceptual similarity or analogy between the concepts representing the literal meaning of the words and the concepts underlying the ultimate meaning of the utterance. The task of interpreting metaphoric language is usually accomplished through the use of an analogical transfer process that directly matches the concepts from the two domains. Most importantly, these processes make no use of explicit knowledge about metaphors that are a conventional part of the language. The goal of MIDAS has been to account for the analogical systematicity of these metaphors while at the same time representing their conventionality in a way that permits efficient processing.

3. Representing UNIX Metaphors

A detailed analysis of what needs to be represented necessarily precedes the task of constructing a knowledge base. This analysis should reveal the phenomena that need to be captured and suggest certain requirements for how these phenomena should be captured. The analysis of some conventional UNIX metaphors, given here, will reveal some of the salient characteristics

that need to be captured. This analysis will only touch upon some of most important characteristics. For a more in-depth discussion of metaphorical systematicities see (Lakoff and Johnson 1980), and for a complete discussion of how these systematicities can be represented see (Martin 1988). Section 4 on representation will present some of the details of how these characteristics are captured using KODIAK.

Consider Example 2 again. The metaphorical use of the word *in* reflects a systematic metaphorical structuring of processes as enclosures. Metaphors like this may be said to consist of two sets of component concepts, a *source* component and a *target* component. The target consists of the concepts to which the words are actually referring. The source refers to the concepts in terms of which the intended target concepts are being viewed. In this example, the target concepts are those representing the state of currently using a computer process, in this case EMACS. The target concepts are those that involve the state of being contained within an enclosure.

The approach taken here is to explicitly represent conventional metaphors as sets of associations between source and target concepts. The metaphor specifies how the source concepts correspond to various target concepts. In this case, the metaphor consists of associations that specify that the state of being enclosed represents the idea of currently using the editor, the user plays the role of the enclosed thing, and the Emacs process plays the role of the enclosure. Therefore, the first requirement for the representation language is to be able to capture these sets of metaphorical associations between concepts from diverse conceptual domains. This first requirement for the representation is summarized as follows:

> **Fundamental Representational Requirement:** Conventional metaphors must be explicitly represented as coherent sets of associations between source and target concepts.

It is, however, clearly not sufficient to merely represent the metaphor underlying Example 2 independently of the metaphors underlying Examples 3 and 4. The metaphor that a process can be viewed as an enclosure lies at the core of each of these uses. This *Enclosure* metaphor is extended in Examples 3 and 4 in ways that are predictable from the semantics of the source and target domains.

The common shared component metaphor among a group of related metaphors will be referred to as a *core metaphor*. Correspondingly an *extended metaphor* is a metaphor that includes all the associations of a core metaphor and adds new associations that coherently extend the core metaphor. Table 1 gives some examples of common UNIX core metaphors and various extensions to them.

Table 1. Core and Extended Metaphors.

Core	
Process-As-Living-Thing	
Extensions	
Creator-As-Parent	*Fork returns a pid to the parent process.*
Created-As-Child	*The child gets a copy of its parent's descriptors.*
Terminating-As-Killing	*How can I kill a process?*
Termination-As-Death	*My emacs just died.*
Core	
Process-As-Enclosure	
Extensions	
Using-As-Enclosed	*You can edit files when you are in the editor.*
Invoking-As-Entering	*You can get into lisp by typing lisp to the shell.*
Uninvoking-As-Exiting	*How can I get out of lisp?*

For example, consider the core metaphor Process-As-Living-Thing. This core metaphor is the basis for the extended metaphors Creator-As-Parent, Created-As-Child, Terminating-As-Killing and Termination-As-Death. In general, the Process-As-Living-Thing metaphor is used to structure the actions that processes perform and the actions that are performed upon them. The Termination-As-Death metaphor structures the target concept of termination in terms of the death of a living thing. The corresponding Terminating-As-Killing metaphor allows the actions that cause the termination to be viewed as a killing (an action that causes a death).

The Process-As-Enclosure metaphor is a distinct core metaphor that also structures some kinds of processes. It primarily structures the target concepts of actively using a process, starting to use a process, and finishing the use, in terms of entering, being enclosed within, and exiting. In each of these cases, the extended-metaphor contains the core-metaphor, and extends it with the addition of further associations. The representation of the metaphorical associations that comprise the core metaphors must be capable of capturing the fact that these associations may be shared by several other extended metaphors. This second representational requirement can be stated as follows:

> **Extended Metaphor Requirement:** The representation must facilitate the sharing of component metaphorical associations from core metaphors to related extended metaphors.

Now consider the following metaphor examples from outside the UNIX domain.

(6) John can *kill* a conversation by walking into a room.

(7) Tom couldn't wait to *get into* the game.

(8) The manager *took* Tom *out of* the game after one inning.

Each of these examples contains a metaphor whose overall structure strongly resembles the structure of the UNIX metaphors given in Examples 1 through 5. In Example 6, a conversation is being viewed as a living thing, motivating the metaphor that an action that causes its termination can be viewed as a killing. This is similar to the structure of Example 1 where the termination of an abstract concept (a process) is also viewed as a killing.

Examples 7 and 8 are further illustrations of this phenomenon. Enclosure or Container metaphors occur across many diverse domains. The commonality among these metaphors is that an ongoing process or activity is often viewed as an enclosure where participation in the process is structured as a containment. Correspondingly the concepts of *entering* and *exiting* denote the concepts of starting and finishing participation.

These examples illustrate that, far from being unique to UNIX, the metaphors used in Examples 1 through 5 are simply specializations of abstract metaphors that are already a widespread conventional part of English. The representation chosen to capture individual core metaphors and their extensions should also capture this overall similarity of structure among metaphors from diverse domains. This final requirement for the representation can be stated as follows:

> **Similarity Requirement:** Representations of conventional metaphors must capture the similarity relationships among metaphors from diverse conceptual domains.

4. Representing Conventional Metaphors

The following sections show how the Fundamental Representational Requirement, Extended Metaphor Requirement, and Similarity Requirement can all be fulfilled through the straightforward use of KODIAK'S structured association and inheritance facilities.

4.1. *Individual conventional metaphors*

The first requirement for the representation is to represent metaphors as concepts consisting of sets of associations between source and target concepts. Consider Example 9.

(9) How can I *kill* a process?

This example, from the UNIX domain, involves the conventional metaphor that to kill an ongoing process means to terminate it. The target concepts involve computer processes and the actions that terminate them. The source concept is that of the action of causing a living thing to die. The metaphor consists of the source, target, and the set of associations linking them.

Figure 1 shows the KODIAK representation of the source domain from Example 9. It states that a `killing` is a kind of `action` with a `result` that is a `death-event` which is in turn an `event`. The `kill-victim` of the `killing` is an inherited role from `action` indicating that the `kill-victim` is effected by the action. The `kill-victim` is constrained to be a `living-thing` and the `killer` must be an `animate-agent`. Finally the equate links require that the `kill-victim` must be the same as the `dier`, the role in the `death-event` representing the deceased.

Figure 2 shows the corresponding concepts from the target domain. It states that a `terminate-process-action` is a `terminate-action` which is a kind of `action`. The `terminated-process` role is an inherited role specifying the patient of the action. The result of the action is a `terminate-process-effect` which is a kind of `terminate-event`. Finally, the `terminated-process` is equated to the `terminated-process-event` of the `terminate-process-effect`. This is analogous to the relationship between the `kill-victim` and the `dier` shown in Figure 1.

What is needed is a way of associating the appropriate source and target concepts. Such an association is realized in KODIAK by using a relation called a *metaphor-map*. A metaphor-map is simply a relation whose roles specify the needed source and target concepts. Metaphor maps are needed to link all the core source concepts in Figure 1 to their counterparts in the target domain. In particular, the `killing` maps to the `terminate-action`, the `kill-victim` maps to the `terminated-process`, the `killer` maps to the `actor` of the `terminate-action`, and the `result` of the `killing` maps to the `result` of the`terminating`. Figure 3 shows the complete set of maps underlying Example 9. It is the co-occurrence of all these maps that constitutes the conventional metaphor that terminating something can be viewed as a killing.

This co-occurrence of a set of more primitive inheritance relations is the definition of a structured association. Therefore, a kind of structured

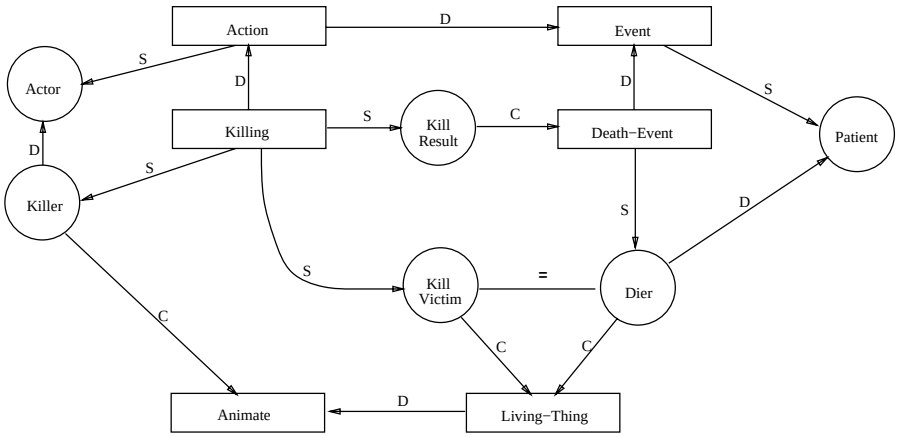

Figure 1. A Kodiak definition for the concept Killing.

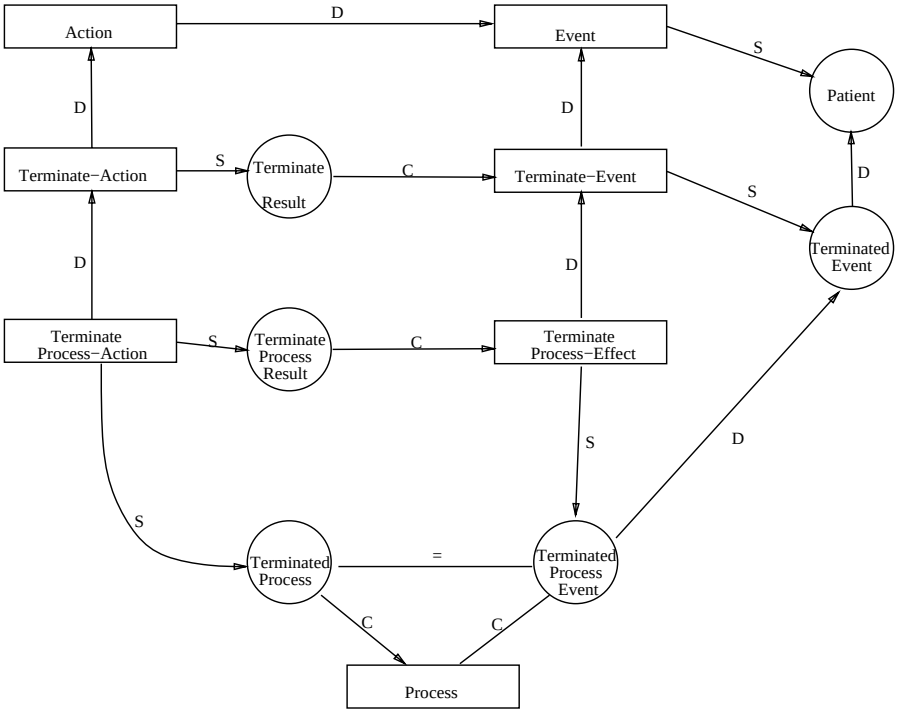

Figure 2. A Kodiak definition for the concept Terminating.

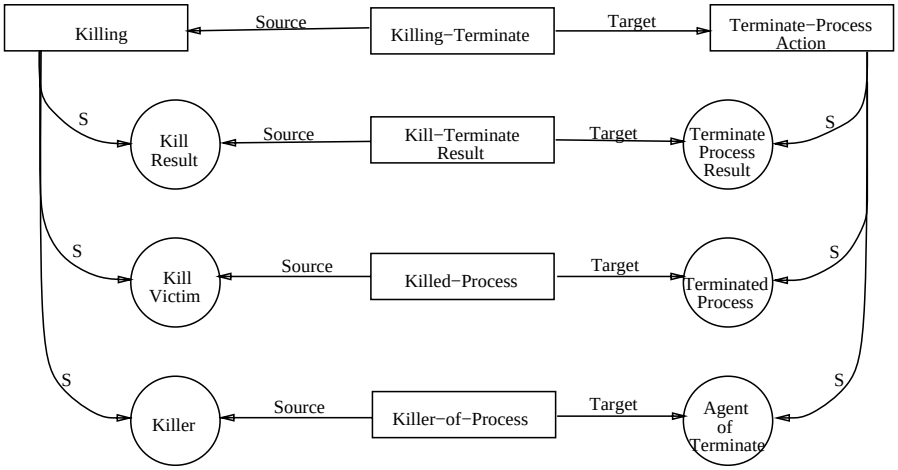

Figure 3. Kill-terminate-maps.

association called a metaphor-sense is introduced to capture this meaningful co-occurrence of metaphor-maps. These maps tie together concepts that are highly inter-connected in the source and target domains. In Example 9, the metaphor-maps tie a single concept and its attendant roles on the source side to an equivalent set on the target side. A metaphor-sense is, therefore, a structured association that ties together sets of component metaphor-maps that together constitute a meaningful conventional metaphor. A metaphor-sense represents a meaningful unit in the same way that the concept `killing` and its relations taken together form a meaningful unit. Figure 4 shows the abbreviated notation for illustrating metaphor-senses. The sense itself is represented as the box enclosing the individual maps.

To a significant extent, metaphor-senses are the minimal meaning-bearing unit of conventional metaphors. Metaphor-maps represent the building blocks out of which meaningful metaphor-senses are constructed. The metaphor-sense represents the level at which one would say that there is a conventional metaphor that to terminate something is to kill it. This level of representation will frequently correspond to a single metaphorical word sense.

4.2. *Extended metaphors*

Consider the problem of capturing the relationships among extended and core metaphors. As described in Section 3 the fundamental phenomenon is that the metaphorical associations that constitute a core metaphor play a central role in metaphors that are the extensions to the core metaphor. This is captured simply in KODIAK since the metaphor-maps that constitute the core metaphor exist explicitly in the system as independent concepts. They are shared as the

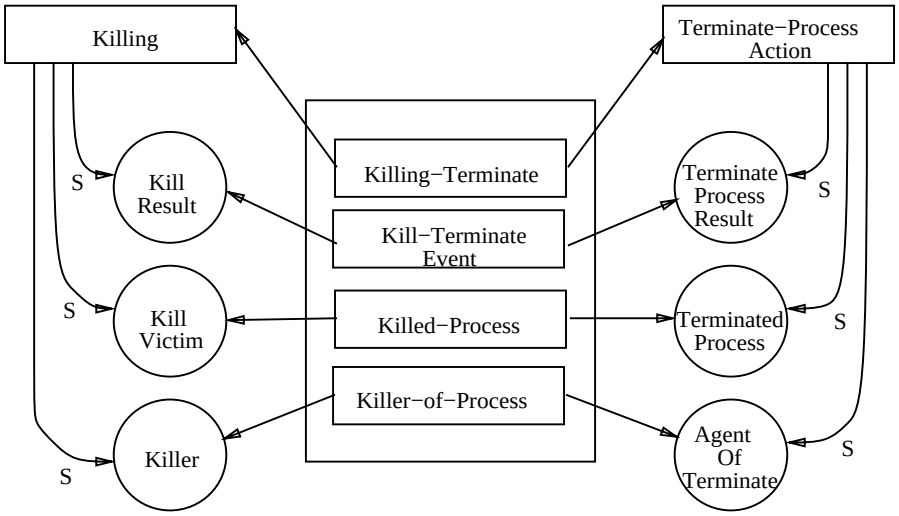

Figure 4. Kill-Terminate-Sense Abbreviated.

fundamental building blocks in all the metaphors that extend from the core metaphor. An extended metaphor uses the maps from the core and adds more of its own to constitute an extension.

Consider again the following *Enclosure* metaphors.

(10) I am in emacs.

(11) How can I *get out of* emacs?

(12) You can *enter* emacs by typing emacs to the shell.

The metaphor-sense underlying the metaphor in Example 10 consists of three metaphor-maps: `in-emacs-map`, `enclosed-user` and `enclosing-used`. These three metaphor-maps link the concepts of the state of being enclosed, the enclosed thing and the enclosure, to the to the concepts of the state of using an EMACS process, a user of the process and the EMACS process being used.

The metaphors underlying Examples 11 and 12 are clearly extensions to the `In-Emacs` metaphor-sense. They both refer to changes of state with respect to the same source and target concepts that make up the `In-Emacs` metaphor. They, therefore, both share the metaphor-maps that make up the core and add new maps to elaborate it. For example, the Enter-Emacs metaphor-sense underlying Example 12 adds two new maps to the core metaphor. The first map associates the source concept of entering with the target concept of invoking an emacs process. The second adds an association linking the source and target initial states of such an action. The maps that make up the core metaphor specify the final state of the action.

4.3. *Capturing similarity*

Section 3 noted that many UNIX metaphors bear a strong resemblance to conventional metaphors that are already a normal part of English. This systematic relationship among metaphors is captured in KODIAK through the use of its inheritance mechanisms. Metaphor-maps and metaphor-senses are both full-fledged KODIAK concepts, and can therefore be arranged in abstraction hierarchies. Hierarchies are the primary mechanism used to account for the similarities and differences among conventional metaphors.

Consider again the metaphor underlying Example 1. As discussed above, the metaphor-sense underlying this example contains a metaphor-map linking the source concept of a living-thing with the target concept computer-process. This is an instantiation of a more general metaphor that allows us to view non-living things in terms of living things for the purpose of explaining or understanding their behavior in terms of living things. Examples 14 through 17, from (Lakoff and Johnson 1980), all contain specialized instances of this general metaphor.

(13) How can I *kill* a process?
(14) Inflation is *eating up* our savings.
(15) Those ideas *died* a long time ago.
(16) He is the *father* of modern quantum mechanics.
(17) Those ideas won't yield any *fruit.*

Example 14 is motivated by the metaphor that the reduction in savings, caused by inflation, can be viewed as inflation consuming the savings. Inflation is viewed as an animal that can consume things. Example 15 contains a metaphor dealing with the duration of ideas. When an idea is no longer held or believed it has died. Example 16 contains the notion that the creation of an idea is a birth-event, and that the originator of the idea plays the role of the father in the birth event with the created idea playing role of the child. Once again, in this metaphor, there is a metaphor-map from a concept that is not a living thing (the created idea) to a role that must be one (the child being born). This metaphor-map, however, is more specific since the constraint is not only to be a living thing but to be human. Finally, Example 17 contains the notion that an idea can produce new ideas. This is metaphorically structured as a plant producing new fruit. In this case, an idea is viewed as a specific kind of living thing, a plant.

What these examples all have in common is the idea that an abstract concept like a process or idea can be viewed as a living thing to explain some aspect of its nature. They differ in the particular kind of living-thing

that is used and in the role that it plays. These similarities and differences result in specific metaphor-maps in each particular case. What is needed is a mechanism that can capture the commonalities and differences among these various metaphor-maps.

This mechanism is provided by the general inheritance mechanisms provided by KODIAK. Metaphor-maps that share properties are dominated by more abstract parent maps that capture the commonalities among the children. The source and target roles of the parent map are constrained by concepts that are more abstract than, and dominate, the constrainers on the children's source and target roles. Figure 5 illustrates this situation with the maps from Examples 13 through 17.

The top half of Figure 5 shows the hierarchical relationships among the maps underlying the above examples. They all converge on the abstract metaphor-map representing the idea of viewing a non-living-thing as a living-thing. The two metaphor-maps in the dotted box are expanded in the bottom half of the diagram to show the exact details of the inheritance links. In this expansion, we can see that the `idea-as-living-thing` metaphor-map dominates the `forgotten-idea-as-dier` map. In the parent map, the `idea-as-target` role is constrained to be an `idea`. The `living-thing-as-source` role is constrained to be a `living-thing`. In the `forgotten-idea-as-dier` map, we see that the inherited source role is specialized by being constrained by the `dier` role of the `death-event`. The inherited target role is further specialized by being constrained by the `forgotten-idea` role.

As mentioned above, metaphor-senses may also be included in these hierarchies. When it appears that an entire set of metaphor-maps is being used repeatedly in specialized domains then a more abstract metaphor-sense can be created from a set of abstract metaphor-senses. This is more fully discussed in (Martin 1988).

This use of an abstraction hierarchy provides the link from specialized metaphors in the UNIX domain to the existing conventional metaphors in English. Section 6 will show how these hierarchies can be exploited to dynamically induce new metaphors in the UNIX domain by analogy to known metaphors from other domains.

5. Conventional Metaphor Interpretation

The main thrust of the MIDAS approach to metaphor interpretation is that normal processing of metaphoric language proceeds through the direct application of specific knowledge about the metaphors in the language. The interpretation of sentences containing metaphoric language is a two-step

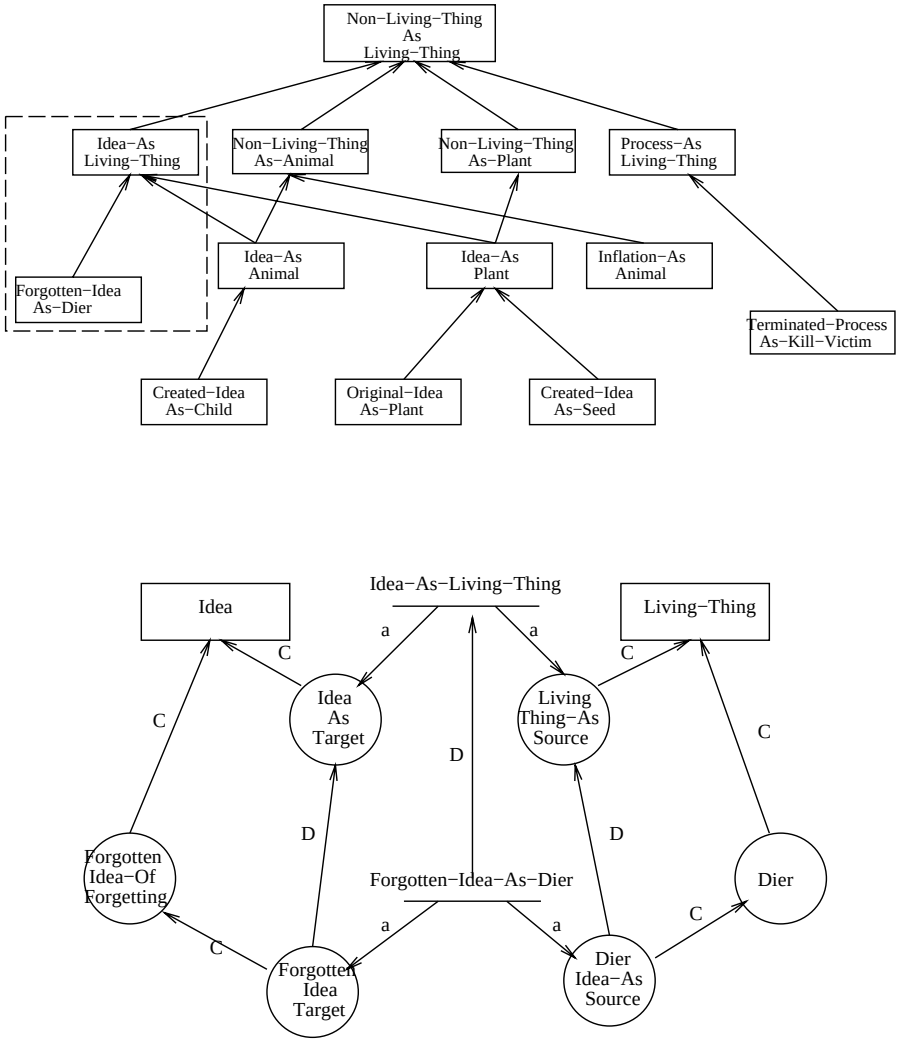

Figure 5. Metaphor-Map Hierarchy.

process. The first step in the interpretation of an input sentence is the production of a syntactic parse and a preliminary semantic representation. In the second step, this preliminary representation is replaced by the most specific interpretation that can coherently account for the input. This interpretation may be a literal one or one of a number of conventional metaphorical interpretations.

This general interpretation process has been implemented in the Metaphor Interpretation System (MIS) component of MIDAS. The MIS examines the initial primal representation in an attempt to detect and resolve uses

of conventional UNIX metaphors. In the following UC example, a user has posed a question involving the conventional metaphor structuring processes as enclosures. The MIS component finds and resolves this metaphor. The resolution produces an instantiation of a coherent target concept representing the correct conventional meaning of the utterance.

In the following examples, trace output is interspersed with a running commentary shown in normal Roman font.

```
> (do-sentence)
Interpreting sentence:
 How can I get into lisp?

Interpreting primal input.

(A Entering50 (ISA Entering)
   (agent597 (ISA agent) (A I203 (ISA I)))
   (patient562 (ISA patient) (A Lisp58 (ISA Lisp))))
```

The input phrase *get into* is treated as a phrasal unit with a conventional meaning corresponding to `Entering`. The preliminary semantic representation produced in this step is called the *primal representation* (Wilensky 1987). The primal representation produced by the parser represents concepts derivable from knowledge of the grammar and lexicon available to the parser. In particular, the primary task accomplished in this phase is the appropriate assignment of filled case roles to the concept underlying the head of a phrase. This primal representation represents a level of interpretation that is explicitly in need of further semantic processing.

```
Concreting input relations.
  Concreting  patient to entered.
  Concreting  agent to enterer.
```

The `patient` and `agent` roles, with their respective filler concepts `I203` and `Lisp58`, were derived solely from the verb class that *enter* belongs to, and the syntax of the sentence. In this next step of processing, these generic roles are replaced by the more specific semantic roles that are actually attached to the `Entering` concept.

```
Interpreting concreted input.

(A Entering50 (ISA Entering)
   (enterer50 (ISA enterer) (A I203 (ISA I)))
   (entered50 (ISA entered) (A Lisp58 (ISA Lisp))))
```

```
Failed interpretation: Entering50 as Entering.
Failed interpretation: Entering50 as Enter-Association.
```

The literal interpretation and one of the other known Entering metaphors are rejected before the correct metaphor is found and applied. These interpretations are rejected because the input concepts filling the roles of `enterer` and `entered` do not match the requirements for these roles in these interpretations. In particular, the interpretation as an actual `Entering` requires that the `entered` concept must be a kind of enclosure. The filler of the `entered` role in the input, `Lisp58`, fails this requirement, therefore this interpretation is rejected. Similarly the `Enter-Association` metaphor specifies that the `entered` concept must be a kind of `Association`. Again, `Lisp58` fails to satisfy this constraint and causes the rejection of the metaphoric interpretation posing this constraint.

Note that the fact that the system considers the literal interpretation first is an artifact of the search procedure. It does not indicate any reliance on attempting the literal meaning first as was the case in previous approaches. All the conventional metaphorical uses have equal status with the known literal concept, `Entering`.

```
Valid known metaphorical interpretation.
Applying conventional metaphor Enter-Lisp.

(A Enter-Lisp  (ISA Container-Metaphor Metaphor-Schema)
   (enter-lisp-res enter-res  lisp-invoke-result)
   (lisp-enterer enterer  lisp-invoker)
   (entered-lisp entered  lisp-invoked)
   (enter-lisp-map Entering  Invoke-Lisp))

Mapping input concept Entering50 to concept Invoke-Lisp30
  Mapping input role enterer50 with filler I203 to
               target role lisp-invoker30
  Mapping input role entered50 with filler Lisp58 to
               target role lisp-invoked30

Yielding interpretation:
 (A Invoke-Lisp30 (ISA Invoke-Lisp)
   (lisp-invoked30 (ISA lisp-invoked)
    (A Lisp58 (ISA Lisp)))
   (lisp-invoker30 (ISA lisp-invoker)
    (A I203 (ISA I))))
```

The Enter-Lisp metaphor has been found and applied to the given input concepts. The main source concept is interpreted as an instance of the Invoke-Lisp concept according to the enter-lisp-map. The input roles enterer and entered are interpreted as the target concepts lisp-invoker and lisp-invoked respectively.

This interpretation of the Entering concept is then used to fill the role of the topic role of the How-Question that constitutes the representation of the rest of the sentence.

```
Final interpretation of input:
 (A How-Q207 (ISA How-Q)
   (topic206 (ISA topic)
    (A Invoke-Lisp30 (ISA Invoke-Lisp)
       (lisp-invoked30 (ISA lisp-invoked)
        (A Lisp58 (ISA Lisp)))
       (lisp-invoker30 (ISA lisp-invoker)
        (A I203 (ISA I))))))
```

This how-question, with the reinterpreted topic concept, is then passed along to the next stage of UC processing. UC then prints the answer as follows.

```
Calling UC on input:

(A How-Q207 (ISA How-Q)
   (topic206 (ISA topic)
    (A Invoke-Lisp30 (ISA Invoke-Lisp)
       (lisp-invoked30 (ISA lisp-invoked)
        (A Lisp58 (ISA Lisp)))
       (lisp-invoker30 (ISA lisp-invoker)
        (A I203 (ISA I))))))

UC: You can get into lisp by typing lisp to the shell.
```

Note that when a conventional metaphor has been employed by the user in asking a question, UC's natural language generator uses the same metaphor in producing the answer. In this example, the system uses the same enclosure metaphor employed by the user to express the plan.

6. Interpreting New Metaphors

MIDAS will inevitably face the situation where a metaphor is encountered for which none of the known metaphors provides an adequate explanation. This situation may result from the existence of a gap in the system's knowledge-base of conventional metaphors, or from an encounter with a novel metaphor. In either case, the system must be prepared to handle the situation.

The approach taken by MIDAS to understanding new or unknown metaphors is called the Metaphor Extension Approach. This approach asserts that a new metaphor can best be understood by extending an existing metaphor in a systematic fashion. The basis for this approach is the belief that the known set of conventional metaphors constitutes the best source of information to use in understanding new metaphors.

The basic strategy is to first find a known metaphor that is systematically related to the new example. This candidate metaphor is then applied to the new example in an attempt to produce an appropriate target meaning. The process of applying the candidate metaphor to the new example is dependent upon the kind of semantic connection between the candidate and the new example. Three kinds of connections are recognized, yielding three kinds of extension inferences: *similarity extension, core-extension* and *combined-extension.*

Once the intended target meaning of the new example has been determined, a new metaphor is created and stored away for future use. When metaphors of this type are encountered again the system can interpret them directly.

This strategy is realized in the Metaphor Extension System (MES) component of MIDAS. When no coherent explanation can be found for a given primal input it is passed along to the MES. The basic steps of MES algorithm are given in the following here. These steps will then be made more concrete in terms of a detailed trace from the MES.

Step 1: Characterize the new input. Partial source and target components of a new metaphor are extracted from the primal representation accepted as input. The terms *current source* and *current target* will be used to refer to the source and target concepts derived from the input example.

Step 2: Search for related metaphors. This step searches for any known metaphors that are potentially related to this new use. The search consists of an attempt to find a path or paths through the network from the current source to the current target concepts that contains a known metaphor. A metaphor contained in such a path is judged to be relevant.

Step 3: Evaluate the set of candidate metaphors found in Step 2. The purpose of this step is to select a metaphor from the set found in Step 2 for

further processing. This choice is based on a set of criteria to determine the metaphor that is closest conceptually to the current example.

Step 4: Apply this previously understood metaphor to the current example. The candidate metaphor is applied to the current target based on the relationship between the candidate mapping and the current example. Depending on this relationship, either a similarity, core, or combined extension inference is performed.

Step 5: Store the new metaphor. Create and store a new metaphor consisting of the source and target concepts identified in the above steps along with appropriate associations between them. This new metaphor will be used directly when future instances of this metaphor are encountered.

Consider the processing of the following example. In this example, UC encounters a metaphor that it has not seen before and has no direct knowledge of. MIDAS makes use of its knowledge of a related metaphor to determine the likely meaning of this new use and creates a new metaphor to be used directly in the future.

```
> (do-sentence)
Interpreting sentence:
 How can I kill a process?

Interpreting primal input.

(A Killing16 (ISA Killing)
   (agent87 (ISA agent) (A I46 (ISA I)))
   (patient76 (ISA patient)
    (A Computer-Process10 (ISA Computer-Process))))
```

The parser accepts the input sentence, as specified by the user, and produces a primal representation of the input in the form of KODIAK concepts.

```
Concreting input relations.
  Concreting  patient to kill-victim.
  Concreting  agent to killer.

Interpreting concreted input.

(A Killing16 (ISA Killing)
   (killer16 (ISA killer) (A I46 (ISA I)))
   (kill-victim16 (ISA kill-victim)
    (A Computer-Process10 (ISA Computer-Process))))
```

```
Failed interpretation: Killing16 as Killing.
Failed interpretation: Killing16 as Kill-Delete-Line.
Failed interpretation: Killing16 as Kill-Sports-Defeat.
Failed interpretation: Killing16 as Kill-Conversation.

No valid interpretations.
Attempting to extend existing metaphor.
```

Once the concreted representation has been created the system attempts to determine if the given input is consistent with any of the known conventional interpretations, literal or metaphorical. In this case, the input is not consistent with either the literal `Killing` concept or any of the three known metaphorical uses of *kill*.

At this point, all the possible conventional interpretations of the primal input have been eliminated as potential readings. The input is now passed to the Metaphor Extension System in an attempt to extend an existing metaphor to cover this new use and determine the intended meaning.

```
Entering Metaphor Extension System
Searching for related known metaphors.

Metaphors found:
 Kill-Conversation Kill-Delete-Line Kill-Sports-Defeat
```

The first step in the extension step is to collect all the relevant known metaphors that might be related to this new use. This initial search scans through all the metaphors directly attached to the input concept, and also at all the metaphors attached to concepts that are core-related to the input concept. In this case, the system has knowledge of three metaphors that share the same source concept with the current use.

```
Selecting metaphor Kill-Conversation to extend from.

(A Kill-Conversation (ISA Kill-Metaphor Metaphor-Schema)
   (kill-c-res kill-result  conv-t-result)
   (killed-conv kill-victim  conv-termed)
   (killer-terminator killer  conv-termer)
   (kill-term Killing  Terminate-Conversation))
```

The candidate metaphors are ranked according to a “conceptual distance” metric. This is a measure of how close the candidate metaphors are to the new example. The primary factor contributing to this metric is a measure

of similarity between the target concepts of the candidate metaphor and the input filler concepts. The candidate metaphor that is judged to be closest to the input example according to this metric is chosen as the candidate metaphor for further processing.

The selected metaphor is classified for further processing according to its relationship to the input example. In this case, the candidate metaphor is in a similarity relationship to the input metaphor.

```
Attempting a similarity extension inference.

Extending similar metaphor Kill-Conversation with
target concept Terminate-Conversation.

Abstracting Terminate-Conversation to ancestor concept
Terminating producing abstract target meaning:

(A Terminating3 (ISA Terminating)
   (terminated3 (ISA terminated)
    (A Computer-Process10 (ISA Computer-Process)))
   (terminator3 (ISA terminator) (A I46 (ISA I))))
```

The first step in the processing of a similarity extension inference is to identify the concepts specified in the input example with their corresponding target concepts. In this example, the concept `Computer-Process10` is identified with the target role of `terminated-conversation`, and the role of `I46` is identified with the target role of `conversation-terminator`. The constrainers of these concepts, however, are too specific to accept these input concepts. In this example, there is a mismatch between the input concept `Computer-Process` and the candidate target concept `Conversation`.

The next step, therefore, is to abstract the target concept of the candidate to the first concept that can accept the concepts specified in the input. In this case, the concept `Terminate-Conversation` is abstracted to its ancestor concept `Terminating`. The ancestor of the `terminated-conversation` role has as a constrainer the abstract concept `Process`, which can constrain the more specific concept `Computer-Process`.

```
Concreting target concept Terminating to
Terminate-Computer-Process producing concreted
meaning:
 (A Terminate-Computer-Process10
   (ISA Terminate-Computer-Process)
   (c-proc-termer10 (ISA c-proc-termer)
    (A I46 (ISA I)))
```

```
(c-proc-termed10 (ISA c-proc-termed)
 (A Computer-Process10 (ISA Computer-Process))))
```

The next step in the similarity extension inference is to look down the hierarchy from `Terminating` to see if there are any more specific concepts beneath this one that can adequately accommodate the input concepts. The specific existing concept `Terminate-Computer-Process10` is found. The concept `c-proc-termed10` is a more specific concept than `terminated` and can still accept the input concept `Computer-Process10` as a filler, since the constraining concept on the concept `c-proc-termed` is a `Computer-Process`.

```
Creating new metaphor:
 Mapping main source concept Killing
  to main target concept Terminate-Computer-Process.
 Mapping source role killer
  to target role c-proc-termer.
 Mapping source role kill-victim
  to target role c-proc-termed.

(A Killing-Terminate-Computer-Process (ISA Kill-Metaphor)
   (killing-terminate-computer-process-map
                   Killing Terminate-Computer-Process))
   (killer-c-proc-termer-map
                   killer c-proc-termer)
   (kill-victim-c-proc-termed-map
                   kill-victim c-proc-termed)
```

The next stage of processing creates a new metaphor that represents this newly learned use. The role correspondences from the input example to the target concepts of the candidate metaphor form the basis for a new set of metaphoric associations that make up the new metaphor-sense. In this case, the main source concept, `Killing`, is mapped to the intended target concept `Terminate-Computer-Process`. The source roles `killer` and `kill-victim` are mapped to the concepts `c-proc-termer` and `c-proc-termed`, respectively. In each case, a new metaphor-map is created to connect the source and target concept in the knowledge base. The map is then classified properly in the hierarchy of existing maps and connected to the newly created metaphor-sense representing this new metaphor.

In the case of a similarity extension inference, the newly created metaphor-maps are made siblings (children of the same parent) of the corresponding metaphor-maps from the candidate metaphor used. The

newly created metaphor-sense is also made a sibling of candidate metaphor. In the current example, the newly created metaphor-sense, `Kill-Terminate-Computer-Process`, is made a sibling of the candidate metaphor `Kill-Conversation`.

```
Final interpretation of input:

(A How-Q46 (ISA How-Q)
   (topic46 (ISA topic)
    (A Terminate-Computer-Process10
       (ISA Terminate-Computer-Process)
       (c-proc-termer10 (ISA c-proc-termer)
        (A I46 (ISA I)))
       (c-proc-termed10 (ISA c-proc-termed)
        (A Computer-Process10
           (ISA Computer-Process))))))
```

The final representation of the input sentence now contains the intended target concept, `Terminate-Computer-Process`, as the topic of the user's original how-question.

```
Calling UC on input:

(A How-Q46 (ISA How-Q)
   (topic46 (ISA topic)
    (A Terminate-Computer-Process10
       (ISA Terminate-Computer-Process)
       (c-proc-termer10 (ISA c-proc-termer)
        (A I46 (ISA I)))
       (c-proc-termed10 (ISA c-proc-termed)
        (A Computer-Process10
           (ISA Computer-Process))))))

UC: You can kill a computer process by typing
    ^C to the shell.
```

The following session demonstrates the altered processing by the system now that the `Killing-Terminate-Computer-Process` metaphor has been acquired. The same question is again posed to the system.

```
> (do-sentence)
Interpreting sentence:
 How can I kill a process?
```

```
Interpreting primal input.
 (A Killing17 (ISA Killing)
   (agent88 (ISA agent) (A I47 (ISA I)))
   (patient77 (ISA patient)
    (A Computer-Process11 (ISA Computer-Process))))

Concreting input relations.
  Concreting  patient to kill-victim.
  Concreting  agent to killer.

Interpreting concreted input.
 (A Killing17 (ISA Killing)
   (killer17 (ISA killer) (A I47 (ISA I)))
   (kill-victim17 (ISA kill-victim)
    (A Computer-Process11 (ISA Computer-Process))))

Failed interpretation: Killing17 as Killing.
Valid known metaphorical interpretation.
Applying conventional metaphor
  Killing-Terminate-Computer-Process.

(A Killing-Terminate-Computer-Process (ISA Kill-Metaphor)
  (kill-victim-c-proc-termed-map
                   kill-victim c-proc-termed)
  (killer-c-proc-termer-map
                   killer c-proc-termer)
  (killing-terminate-computer-process-map
                   Killing Terminate-Computer-Process))
```

The application of this known metaphor immediately yields the intended interpretation.

```
Yielding interpretation:

(A Terminate-Computer-Process11
   (ISA Terminate-Computer-Process)
   (c-proc-termed11 (ISA c-proc-termed)
    (A Computer-Process11 (ISA Computer-Process)))
   (c-proc-termer11 (ISA c-proc-termer)
    (A I47 (ISA I))))
```

As in the previous example, all the conventional meanings are attempted before an interpretation is settled upon.

```
Failed interpretation: Killing17 as Kill-Delete-Line.
Failed interpretation: Killing17 as Kill-Sports-Defeat.
Failed interpretation: Killing17 as Kill-Conversation.

Final interpretation:

(A How-Q47 (ISA How-Q)
   (topic47 (ISA topic)
    (A Terminate-Computer-Process11
       (ISA Terminate-Computer-Process)
       (c-proc-termed11 (ISA c-proc-termed)
        (A Computer-Process11
           (ISA Computer-Process)))
       (c-proc-termer11 (ISA c-proc-termer)
        (A I47 (ISA I))))))

UC: You can kill a computer process by typing
    ^C to the shell.
```

7. Summary

Consultant systems in technical domains must be capable of dealing with the conventional metaphors that structure those domains. The MIDAS system provides this capability for the UNIX Consultant. In particular, MIDAS has been used to systematically represent knowledge about the conventional metaphors in UNIX, interpret metaphorical language based on these metaphors, and finally to learn new UNIX domain metaphors as they are encountered by UC during normal processing.

8. Recent Developments

While the MIDAS project (Martin 1990, 1992) demonstrated some significant results, it nevertheless had a major shortcoming. The effectiveness of the approach for language interpretation, generation and acquisition was obviously dependent on the size and the correctness of the knowledge-base of non-literal conventions. Unfortunately, the knowledge-base used by MIDAS did not have any kind of real coverage, nor did it have an empirically verifiable basis. Our more recent empirical work (Martin 1994) has been an attempt to address some of these problems. Specifically, we have been

developing corpus-based techniques to identify and analyze the conventional metaphors that tend to occur within large text collections genres. The results of these empirically-based analyses then forms the basis for the creation of a knowledge-base.

References

Carbonell, J. (1981). Invariance Hierarchies in Metaphor Interpretation. In *Proceedings of the Third Meeting of the Cognitive Science Society*, 292–295. Berkeley, CA: Cognitive Science Society.

DeJong, G. F. & Waltz, D. L. (1983). Understanding Novel Language. *Computers and Mathematics with Applications* **9**(1): 131–147.

Fass, D. (1988). Collative Semantics: A Semantics for Natural Language. Ph.D. diss., New Mexico State University, Las Cruces, New Mexico. CRL Report No. MCCS-88-118.

Gentner, D., Falkenhainer B. & Skorstad J. (1988). Viewing Metaphor as Analogy. In Helman, D. (ed.) *Analogical Reasoning*. Dordrecht: Kluwer Academic Publishers.

Indurkhya, B. (1987). Approximate Semantic Transference: A Computational Theory of Metaphors and Analogy. *Cognitive Science* **11**(4): 445–480.

Jacobs, P. S. (1985). A Knowledge-Based Approach to Language Production. Ph.D. diss., University of California, Berkeley, Computer Science Department, Berkeley, CA. Report No. UCB/CSD 86/254.

Lakoff, G. & Johnson, M. (1980). *Metaphors We Live By*. Chicago, Illinois: University of Chicago Press.

Martin, J. H. (1986). The Acquisition of Polysemy. In *The Proceedings of the Fourth International Conference on Machine Learning*, 198–204. Irvine, CA,

Martin, J. H. (1987). Understanding New Metaphors. In *The Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, 137–139. Milan, Italy.

Martin, J. H. (1988). A Computational Theory of Metaphor. Ph.D. diss., University of California, Berkeley, Computer Science Department, Berkeley, CA. Report No. UCB/CSD 88–465.

Martin, J. H. (1990). *A Compututational Model of Metaphor Interpretation*. Cambridge, MA: Academic Press.

Martin, J. H. (1992). Computer Understanding of Conventional Metaphoric Language. *Cognitive Science* **16**(2): 233–270.

Martin, J. H. (1994). Metabank: A Knowledge-base of Metaphoric Language Conventions. *Computational Intelligence* **10**(2): 134–149.

Norvig, P. (1987). A Unified Theory of Inference for Text Understanding. Ph.D. diss., University of California, Berkeley, Computer Science Department, Berkeley, CA. Report No. UCB/CSD 87–339.

Wilensky, R. (1986). Some Problems and Proposals for Knowledge Representation. Technical Report UCB/CSD 86/294, University of California, Berkeley, Computer Science Division.

Wilensky, R. (1987). Primal Content and Actual Content: An Antidote to Literal Meaning. Technical Report UCB/CSD 87/365, University of California, Berkeley, Computer Science Division.

Wilensky, R., Chin D., Luria M., Martin J., Mayfield J., & Wu D. (1988). The Berkeley UNIX Consultant Project. *Computational Linguistics* **14**(4): 35–84.