EPILOG: The Computational System for Episodic Logic

NEW USERS TUTORIAL

Stephanie Schaeffer

Chung Hee Hwang

John de Haan

Lenhart K. Schubert

August 1993

Revised September 2000

Prepared for Boeing Computer Services, Seattle, Washington

Under Purchase Contract W-278258

Contents

Chapter 1

Introduction to EPILOG

EPILOG is an inference engine designed to handle the representation and low level reasoning necessary for natural language understanding. Although it was designed with this purpose in mind, the representation and inference it does can be used for other domains with similar requirements. It uses an extended first order logic called **episodic logic** (**EL**) (Hwang and Schubert) which allows propositional attitudes, unreliable generalizations, and other non-standard constructs, including ones involving events, actions, facts, kinds and donkey sentences. Episodic sentences are represented using episodic variables, which can be used to capture temporal and causal relationships. Axiom schemas help to control the number of rules required, as well as represent narrative and meaning postulate information. The rules of inference include probabilistic versions of deduction rules resembling the forward and backward chaining rules in expert systems. Special inference methods for certain specific domains assist the general inference mechanism to increase overall efficiency.

EPILOG maintains the knowledge given to it (including type and part hierarchies, general world knowledge, specific story knowledge, meaning postulates, narrative rules, etc), and then makes inferences based on this information (logical consequences of the input). It can also answer yes/no and some wh questions. Optionally, the system will try to "say" the input sentences, inferences, and question answers in English (although currently the English to episodic logic translation has not been completely automated and so input in English is not possible yet).

This section contains a brief overview of **EPILOG** , describing some of its capabilities in the context of a simple "hands-on" session. This session and description of **EPILOG** start at very basic operations and gradually move to some more complex ones. This should be enough to get you started playing with the system. More details are available in later parts of the manual.

There are examples of EPILOG commands and input here and throughout the manual, and in addition, there are some example files included with the system - these files are called *test1, test2* , and so on. The input in those files is acceptable to the system, and the results can be seen by running the system and loading those files, or by looking at the *eg1.lis, eg2.lis* , etc output files, also included with the system. Many of the examples in this manual come from the story of *Little Red Riding Hood* .

The following sections contain examples of the basic commands needed to operate the system. There are many ways to modify the system's performance, including tweakable parameters for most aspects of the system, adding translation and lexical information for the response generator, setting trace values so that the system prints information during its operation, etc. These are described later in the manual.

Don't be alarmed by the use of the unfamiliar syntax of the logic - this is explained in more detail in

the User's Guide. Only simple examples are used in this introduction.

1.1 Getting Started

The root directory of the **EPILOG** distribution is called *EPILOG*. The system itself is in a subdirectory of *EPILOG* called *epi*. In that subdirectory is the system startup file *epi.lisp*.

Throughout the example, user input will be in bold, system output in regular print.

lisp
> **(load "EPILOG/epi/epi")** ;; use the proper path to EPILOG on your system.
****
* Welcome to EPILOG: the computational system for Episodic Logic *
****
... messages about files being loaded and specialists being activated ...

Enter the *EPILOG* package, where the system resides:

> **(in-package epilog)**
<The EPILOG package>

Next, load the file *test-files/test-common.lisp*. The part of this file which is relevant for our example is the loading of translation information, which helps the system present English paraphrasals of Episodic Logic formulas.

> **(load "EPILOG/test-files/test-common.lisp")**
...
T

1.2 Assertion of Formulas

The **EPILOG** system loads linguistic and world knowledge by appropriately indexing it, and processes story sentences doing input-driven inferencing (i.e. figures out the "consequences" of the input story facts). Optionally, responses to questions and input to the system may be "said" in English.

EPILOG works with logical formulas (or *wffs* - well formed formulas) - axiom schemas, rules, and story facts. Simple formulas can be input straight into the system with no other information needed - the system can use some default information for things it doesn't know about.

Story sentences are typically non-conditionals, and indexed by (non-variable) individuals that appear in the formula. Two classification methods are used, and formulas are available under both. The main indexing method uses the actual predicates in a formula, along with the arguments, and the roles they play. This classification method gives optimum performance for both input-driven and goal-driven inference. In addition, formulas are classified topically, using the topics indicated by the predicates and arguments. This classification method is more useful for displaying known facts about a particular entity, and perhaps in the future for answering questions like *What does Little Red Riding Hood look like?* . So, *Fido loves Morris* will be classified under *(Fido love subject)* and *(Morris love object)* for the main classification,

and *(Fido tp.emotional-giver)* and *(Morris tp.emotional-object)* for the topical classification. Story facts are entered into the system as follows:

```
> (story '(wolf1 wolf))
 Loading story sentence ...
    WFF1: (WOLF1 WOLF)
    Stored in Input Array [0]
    Probability: 1
 WOLF1 is a wolf.
(WFF1)
```

Note that prefix order is used, and that the system responded with an English translation of the input. To see where that formula was actually stored, we can display known information about *wolf1* :

```
> (display 'wolf1)
 WOLF1
   TP.TYPE
     WFF1: (WOLF1 WOLF)     Probability: 1
```

Knowledge is expected to take the form of a conditional: either a universal implication or a generic conditional. The conditional knowledge is classified and stored, and is available for both input-driven and goal-directed reasoning. For example, *Most dogs hate cats* will be indexed by *(dog hate subject)* and *(cat hate object)* for the main classification, and (*dog tp.emotional-giver)* and and *(cat tp.emotional-object)* for the topical classification. The following is a universal implication:

```
> (kn '(a x (x girl) (x pretty)))
Loading knowledge ...
    WFF4: (A X (X GIRL) (X PRETTY))
    Probability: 1
 Girls are pretty.
(WFF4)
```

And we can look at everything in the knowledge base to see where it was put:

```
> (display 'wffs)
Accessible environments: Main Environment,
Contents of current environment Main Environment:
(WOLF1 TYPE-SPECIALIST)
   WFF1: (WOLF1 WOLF)     Probability: 1
(GIRL TYPE-SPECIALIST)
   WFF4: (A X (X GIRL) (X PRETTY))     Probability: 1
(GIRL PRETTY SUBJECT)
   WFF4: (A X (X GIRL) (X PRETTY))     Probability: 1
(WOLF)
   WFF1: (WOLF1 WOLF)     Probability: 1
```

And to see the topical classification arrangement:

```
> (display 'wffs-by-topic)
Accessible environments: Main Environment,
Contents of current environment Main Environment:
(WOLF1 TP.TYPE)
   WFF1: (WOLF1 WOLF)     Probability: 1
(GIRL TP.APPEARANCE)
   WFF4: (A X (X GIRL) (X PRETTY))     Probability: 1
(WOLF TP.SPEC)
   WFF1: (WOLF1 WOLF)     Probability: 1
(GIRL TP.TYPE)
   WFF4: (A X (X GIRL) (X PRETTY))     Probability: 1
```

Note the classifications used for the conditional just entered.

If an input sentence is non-conditional, **EPILOG** attempts input-driven inference based on the conditionals that have been loaded earlier using rule instantiation. For example, given *John admires every girl he knows* and *Sue is a girl; John knows her,* **EPILOG** will infer that *John admires Sue .*

Note the inference using rules in the knowledge base when the story fact below is asserted:

```
> (story '(lrrh girl))
Loading story sentence ...
    WFF5: (LRRH GIRL)
    Stored in Input Array [1]
    Probability:  1
 ** INFERRED FORMULA **
     WFF6: (LRRH PRETTY)
     Rank:  1
     Parents:  (WFF4 WFF5)
     Probability:  1
 Little Red Riding Hood is a girl.
 She is pretty.
(WFF5 WFF6)
```

Notice in the above example that *lrrh* is said as *Little Red Riding Hood* . This is because of information loaded from the file *test-files/test-common.lisp.* Input of this nature will be discussed later.

Input driven inference is attempted on conditionals (rules) as well. The following is a generic conditional knowledge wff. When entered, input-driven inferencing will be done on it using story facts in the knowledge base.

```
> (kn '((e x (x wolf)) 0.8 (x grey)))
Loading knowledge ...
    WFF10: ((E X (X WOLF)) 0.8 (X GREY))
    Probability:  1
```

```
 ** INFERRED FORMULA **
     WFF11: (WOLF1 GREY)
     Rank:  1
     Parents:  (WFF10 WFF1)
     Probability:  0.8
 A wolf is usually grey.
 WOLF1 is a grey wolf.
(WFF10 WFF11)
```

Note also that the inferred wff has a probability less than 1, because of the rule used to generate it.

Episodic facts are entered with a **, *, or @ predicate and an episode, such as:

```
> (story '((wolf1 meet lrrh) ** ep1_ep))
Loading story sentence ...
    WFF13: ((WOLF1 MEET LRRH) ** EP1)
    Stored in Input Array [2]
    Probability:  1
WARNING: Response: Missing translation information for MEET - guessing
 The wolf met Little Red Riding Hood.
(WFF13)
```

The *_ep* on *ep1* indicates that *ep1* is an episode. Some specialists require sorts; these are described in Chapter 2. In some cases (including this one), the sort may be left off and the system will assume it. The warning from the response generator just means it had to guess how to say *meet* - in this case its default mechanism says it properly.

Conditional story sentences are indexed similar to conditional knowledge, but if a non-variable individual is in the sentence, e.g., *John admires every girl he knows,* the formula is indexed under that individual as well.

Equivalences are implications that work in both directions. For example, note that this equivalence fires in the "reverse" direction. The equivalence says that "meeting" and "greeting" are equivalent. Note that the probability is 1 in both directions.

```
> (kn '(A x (x creature) (A y (y creature) (A z_ep (((x greet y) * z) <=> ((x meet y) * z))))))
Loading knowledge ...
    WFF23: (A X (X CREATURE) (A Y (Y CREATURE) (A EP-Z (((X GREET Y) * EP-Z)
          <=> ((X MEET Y) * EP-Z)))))
    Probability:  1
 ** INFERRED FORMULA **
     WFF25: ((WOLF1 GREET LRRH) * EP1)
     Rank:  1
     Parents:  (WFF23 WFF5 WFF1 WFF13)
     Probability:  1
WARNING: Response: Missing translation information for GREET - guessing
 Creatures that greet other creatures meet the other creatures.
```

The wolf greeted Little Red Riding Hood.
(WFF23 WFF25)

Rules can be used in both the forward and backward direction. When an input-driven inference is made with a rule in the backward direction, it is a contrapositive inference. For example:

> **(kn '(a x ((x wolf) and (x smart)) (x friendly)))**
Loading knowledge ...
WFF29: (A X ((X WOLF) AND (X SMART)) (X FRIENDLY))
Probability: 1
WARNING: Response: Missing translation information for SMART - guessing
WARNING: Response: Missing translation information for FRIENDLY - guessing
Smart wolves are friendly.
(WFF29)

In this case, the contrapositive of this rule says that *Only friendly wolves are smart* . If a non-friendly wolf is smart, the rule would fire to say that it is friendly (a contradiction), so given a wolf that is not friendly, we can infer that it cannot be smart.

> **(story '(not wolf1 friendly))**
Loading story sentence ...
WFF30: (NOT WOLF1 FRIENDLY)
Stored in Input Array [3]
Probability: 1
** INFERRED FORMULA **
WFF32: (NOT WOLF1 SMART)
Rank: 1
Parents: (WFF29 WFF1 WFF30)
Probability: 1
WOLF1 isn't a friendly wolf.
It isn't smart.
(WFF30 WFF32)

Meaning postulates (in the form of axiom schemas) may also be entered into the system, and will be used in input-driven inferencing. They are discussed in Chapter 3.

1.3 Questions

Besides making input-driven inferences, the system can answer Yes/No questions, and also some simple wh-questions. Inference used in answering questions includes evaluation (using specialists to assist), and a natural-deduction-like breakdown of subgoals. This is described in detail later in the manual.

The following question asks: *Did a wolf meet Little Red Riding Hood?* , or more specifically *Is there an episode of a wolf meeting Little Red Riding Hood?* .

> **(q '(e x (x wolf) (e y_ep ((x meet lrrh) ** y))))**

```
Questioning (E U (U WOLF) (E EP-V ((U MEET LRRH) ** EP-V)))
Answer: YES with probability 1
 Yes, the wolf met Little Red Riding Hood.
   Used 2 iterations  Searched to depth 1
((YES (WFF13 WFF1) (1 WFF13 WFF1)))
```

The answer to the question is the last line. The three part answer includes the answer itself (YES), the formulas used to determine that answer, and the support set used to determine the probability of the answer.

WH-questions can also be answered. The quantifier *WH* differentiates these from regular questions. For example, the following question asks: *Who met whom?* .

```
> (q '(WH x (WH y (E z ((x meet y) @ z)))))
Questioning (WH WH-U (WH WH-V (E W ((WH-U MEET WH-V) @ W))))
Depth much lower than previous answers
Answer(s):
        (WOLF1 LRRH)  with probability 1
 The wolf met Little Red Riding Hood.
   Used 19 iterations  Searched to depth 1
(((WOLF1 LRRH) (WFF13) (1 WFF13)))
```

Note that the answer to this question (first part of last line) was not a YES or NO, but a list of two names - these "answer" the question by filling in for the variables quantified by WH.

Optionally, results of queries may be automatically saved by the system. For existentially quantified questions, or wh-questions, the entities which matched the variables are substituted before saving the result.

1.4 Other Information Input

The system uses information about particular predicates, operators and functions to normalize, classify, and say formulas. Although it can sometimes operate without this information (for predicates, but not operators or functions), better results are obtained with it. Note what happens in the following example when an unknown predicate *snarl-at* is used.

```
> (story '((wolf1 snarl-at lrrh) ** ep2_ep))
WARNING: Normalization: SNARL-AT is a new predicate
WARNING: Classification: SNARL-AT has no topic indicators - assuming (TP.UNKNOWN)
Loading story sentence ...
    WFF34: ((WOLF1 SNARL-AT LRRH) ** EP2)
    Stored in Input Array [4]
    Probability: 1
WARNING: Response: Missing translation information for SNARL-AT - guessing
 The wolf snarl-ated Little Red Riding Hood.
(WFF34)
```

Note that the system warned about the new predicate, and then guessed how to classify it and say it. It's guess at how to say it wasn't particularly good in this case, however.

```
> (display 'wolf1)
WOLF1
   TP.TYPE
    WFF1: (WOLF1 WOLF)     Probability: 1
   TP.UNKNOWN
    WFF34: ((WOLF1 SNARL-AT LRRH) ** EP2)     Probability: 1
   TP.PHYSICAL-QUALITY
      TP.EXTERNAL-QUALITY
         TP.APPEARANCE
            TP.COLORING
              WFF11: (WOLF1 GREY)     Probability: 0.8
   TP.ABSTRACT-QUALITY
      TP.MENTAL-QUALITY
         TP.EMOTIONAL-DISPOSITION
           WFF30: (NOT WOLF1 FRIENDLY)     Probability: 1
         TP.INTELLECTUAL-DISPOSITION
           WFF32: (NOT WOLF1 SMART)     Probability: 1
   TP.BEHAVIOR
      TP.SOCIAL
       WFF13: ((WOLF1 MEET LRRH) ** EP1)     Probability: 1
         TP.COMMUNICATION
           WFF25: ((WOLF1 GREET LRRH) * EP1)     Probability: 1
```

1.4.1 Types and Topics

As noted earlier, the system classifies things in two ways - the main classification using predicates, and the topical classification. For both classifications, the system needs to know if something is a type predicate - these are predicates which are located on a type hierarchy. In addition, the topical classification requires that topics be associated with each predicate (or specialist). The default is *tp.unknown* . The system comes with this information set up for a number of type and other predicates in the files *eg.hier* and *eg.indicate* . You may easily add to them.

To make a predicate into a type predicate, it just has to exist on a type hierarchy. To do this:

```
> (add-hier 'dog 'dog 'sporting-dog 'working-dog 'toy-dog)
((DOG SPORTING-DOG WORKING-DOG TOY-DOG))
> (add-hier 'dog 'sporting-dog 'cocker-spaniel 'pointer 'retriever)
 ((DOG COCKER-SPANIEL POINTER RETRIEVER))
> (display 'hier 'dog '-full)
DOG   Root: DOG   Hierarchy type: EXCLUSION
  Connections to other hierarchies:
```

```
    Hier: THING     Predicates: (DOG)
DOG [1,7]
  SPORTING-DOG [2,5]
    COCKER-SPANIEL [3,3]
    POINTER [4,4]
    RETRIEVER [5,5]
  WORKING-DOG [6,6]
  TOY-DOG [7,7]
NIL
```

Now *dog* and all of its "children" are available for use as predicates in formulas. Note the hierarchy type "exclusion". This means that sibling nodes are incompatible (e.g. a cocker spaniel cannot also be a retriever). Overlap hierarchies are also available. More information on the hierachies can be found in the User's Guide.

To add topics for a non-type predicate, there are two methods - add-topic, to add several predicates to a topic, or add-indicate, which adds topics to a single predicate (both have the identical result, the use of one over the other is purely a matter of convenience). If there isn't a topic which seems appropriate, add one to the topic hierarchy using the add-hier command (look at the topic hierarchy in *eg.hier* to decide where to add the new topic).

```
> (add-hier 'tp.topics 'tp.constructive-activity 'tp.gardening-activity)
((TP.TOPICS TP.GARDENING-ACTIVITY))
> (add-topic 'tp.gardening-activity 'rake 'mow)
 NIL
> (add-indicate 'weed 'tp.gardening-activity)
 NIL
```

Now the predicates *rake* , *mow* and *weed* are ready to use in formulas.

1.4.2 Response Information

Response information can also be given to the system to tell it how to say formulas in a more natural way. Many predicates already have translation information - this information are in the files *eg.trans* and *eg.lex* . Note what happens if we add some information about *snarl-at* :

```
> (add-word 'snarl-at '(S (NP 1) (VP (verb snarl) (objects (PP (prep at) (NP 2))))))
(S (NP 1) (VP (VERB SNARL) (OBJECTS (PP (PREP AT) (NP 2)))))
> (say-it '(wff34))
  The wolf snarled at Little Red Riding Hood.
```

Details on adding this information can be found in the User's Guide.

Chapter 2

Introduction to Specialists

The general inference method used by **EPILOG** is as efficient as we could make it, but, like all general purpose reasoners, it is quite inefficient in some specific, well understood domains, like temporal reasoning. To compensate for this, specialists assist the general reasoner by making very fast evaluations and comparisons using their own special-purpose mechanisms.

Specialists are special purpose inference mechanisms that can short-circuit long proof attempts into a single inference. They are particularly useful where lengthy chains of reasoning are required - such as with types, event orderings, set membership, etc. The specialists may maintain their own internal representation of the story facts - this is how they achieve their speed. A number of specialists have been included with **EPILOG** . Many of these are automatically started up with the system, including:

type-specialist - handles type predicates using type hierarchies

hier-specialist - handles non-type predicates arranged hierarchically

part-specialist - handles part-of relationships

episode-specialist - handles the relationship between **, *, and @

equality-specialist - handles equality among constants

other-specialist - handles predicates and functions externally defined

meta-specialist - handles predicates and functions at the meta level

There are some which you may optionally start up yourself, as well. These are:

time-specialist - handles event orderings and durations, and absolute times

set-specialist - handles set membership and related relations

number-specialist - handles number ordering and arithmetic functions

color-specialist - handles relations between colors

This chapter first describes in general how to use a specialist (mainly how to start one up), and then gives examples for each specialist. The specialists are ordered so that the ones most likely to be of interest to a user are described first. The meta-specialist discussion is saved for the next chapter. The last sections in this chapter describe specialist communication and some more complicated specialist activities, like literal comparison and the use of subnets for modally embedded formulas. They are left to the end so that you will have some familiarity with the specialists first.

2.1 Using Specialists

The specialists which are automatically started up with the system will operate without your even being aware of them. However, you can trace the specialists' actions, and even control them to some extent.

To start up another specialist, say the time-specialist, you may use its full name, or a recognized nickname (found in the User's Guide). Note that the specialist should really be started up immediately after

EPILOG is - that way we can guarantee that anything in its domain is actually also stored in it. In this case we haven't yet entered anything in the time-specialist's domain, so no inconsistencies will result.

> **(use-spec 'time)**
Specialist TIME-SPECIALIST is now active
(TIME-SPECIALIST)

In order to watch what the specialist does, you can trace its actions (or a subset of them). The trace values are also available in the User's Guide. For example:

> **(trace 'time-entry 'time-eval)**
NIL

Some specialists require that the arguments in the literals they deal with be of specific sorts. Where this is the case in the following sections, it will be pointed out.

2.2 Type Specialist

Currently the type-specialist is used only to compare predicates, using the type hierarchies described earlier. To see where this happens, we can trace the action of the specialist while we ask the question *Is Little Red Riding Hood a human?* .

> **(trace 'type-test)**
NIL
> **(q '(lrrh human))**
Questioning (LRRH HUMAN)
Type specialist: compared HUMAN with GIRL - result SUBSUMES
Answer: YES with probability 1
Yes, Little Red Riding Hood is a girl.
Used 0 iterations Searched to depth 0
((YES (WFF5) 1))

Here you can see that the type-specialist was used to compare

human , which is what we were asking, with *girl* , which was known about *lrrh* . This comparison is used for more than just simple evaluations, however, and can get quite messy if you trace it for long periods of time, so we'll turn it off now.

```
> (untrace 'type-test)
NIL
```

2.3 Time Specialist

Having activated the time-specialist earlier, and started tracing for it, we can now enter something in the time-specialist's domain and watch what it does with it. Note that **before-0** means that the first event is before the second, with its endpoint the same as the second event's start point (so they *meet*).

```
> (story '(ep1_ep before-0 ep2_ep))
   Time Specialist: Evaluating EP1 BEFORE-0 EP2
Loading story sentence ...
    WFF35: (EP1 BEFORE-0 EP2)
    Stored in Input Array [5]
    Probability:  1
   Time Specialist: Entering EP1 BEFORE-0 EP2
 The wolf greeted Little Red Riding Hood before it snarled at her.
(WFF35)
```

The *_ep* attached to the arguments set them up to be of sort episode, so that the time-specialist will recognize them as being in its domain. The time specialist requires that arguments be of sort *episode (ep)* , or *time* .

We can display information in the time specialist about a specific event. The chain and psuedo information shows how the points were placed in the time-specialist' internal time graph. The "Absolute" information contains date and time information, if available.

```
> (display 'event-info 'ep1 '-f)
EP1
 Start  EP1START
EP1START        Node    EP1START
        Chain   0
        Psuedo  -1000
        Min-psuedo      -infinity
        Max-psuedo      +infinity
        Absolute-min    UNKNOWN
        Absolute-max    UNKNOWN
 End    EP1END
EP1END  Node    EP2START
        Chain   0
        Psuedo  1
        Min-psuedo      -infinity
        Max-psuedo      +infinity
        Absolute-min    UNKNOWN
        Absolute-max    UNKNOWN
```

```
NIL
```

And we can ask a question which the time specialist can help answer -
Is the event we referred to as ep2 after the event we referred to as ep1? .

```
> (q '(ep2 after ep1))
Questioning (EP2 AFTER EP1)
   Time Specialist: Evaluating EP2 AFTER EP1
   Time Specialist: Evaluated EP2 AFTER EP1 to YES
Answer: YES with probability 1
 Yes, the wolf snarled at Little Red Riding Hood after it greeted her.
   Used 0 iterations  Searched to depth 0
((YES (T-WFF8) 1))
```

We can get absolute time information using similar constructions and the **date** function. We can trace it to watch the absolute times propagate.

```
> (trace 'abs-time-entry)
NIL
> (story '(ep1 after (date 1991 08 05 00 00 00)))
   Time Specialist: Evaluating EP1 AFTER ($ 'TIME 1991 8 5 0 0 0)
Loading story sentence ...
    WFF39: (EP1 AFTER ($ 'TIME 1991 8 5 0 0 0))
    Stored in Input Array [6]
    Rank:  0
    Parents:  (WFF37)
    Probability:  1
   Time Specialist: Entering EP1 AFTER ($ 'TIME 1991 8 5 0 0 0)
   Time Specialist: Adding absolute time minimum (1991 8 5 0 0 0) to EP1START
   Time Specialist: Propagating absolute time minimum (1991 8 5 0 0 0) to EP2START
   Time Specialist: Propagating absolute time minimum (1991 8 5 0 0 0) to EP2END
 The wolf greeted Little Red Riding Hood after August 5, 1991.
(WFF39)
```

Note that the function was evaluated to a record, which then replaces the function.

Now we can display the events again to see what changes this made:

```
> (display 'event-info '-f)
Context /
Event Table
EP1
 Start  EP1START
EP1START Node    EP1START
       Chain   0
```

```
        Psuedo  -1000
        Min-psuedo      -infinity
        Max-psuedo      +infinity
        Absolute-min    (1991 8 5 0 0 0)
        Absolute-max    UNKNOWN
 End    EP1END
EP1END  Node    EP2START
        Chain   0
        Psuedo  1
        Min-psuedo      -infinity
        Max-psuedo      +infinity
        Absolute-min    (1991 8 5 0 0 0)
        Absolute-max    UNKNOWN
EP2
 Start  EP2START
EP2START Node   EP2START
        Chain   0
        Psuedo  1
        Min-psuedo      -infinity
        Max-psuedo      +infinity
        Absolute-min    (1991 8 5 0 0 0)
        Absolute-max    UNKNOWN
 End    EP2END
EP2END  Node    EP2END
        Chain   0
        Psuedo  1000
        Min-psuedo      -infinity
        Max-psuedo      +infinity
        Absolute-min    (1991 8 5 0 0 0)
        Absolute-max    UNKNOWN
```

All the lower bounds on the absolute times (dates) of this chain of time points were set by the one assertion.

Durations are numbers which represent the number of seconds elapsed. They can be added for an event (the duration between the start and end points), or between two arbitrary time points or events. In this next example, we add time point *t1* before event *ep1* , and the duration between is at most 1 hour. We can also add another point before that one by exactly 2 hours.

```
> (trace 'time-duration-entry)
NIL
> (story '(t1_time at-most-before e1 3600))
   Time Specialist: Evaluating T1 AT-MOST-BEFORE EP1 3600
Loading story sentence ...
    WFF40: (T1 AT-MOST-BEFORE EP1 3600)
```

Stored in Input Array [7]
Probability: 1
Time Specialist: Entering T1 AT-MOST-BEFORE EP1 3600
Time Specialist: Adding duration maximum 3600 between T1 and EP1START
T1 is before the wolf greeted Little Red Riding Hood by at most 3600 seconds.
(WFF40)
> **(story '(t1 exactly-after t2_time 7200))**
Time Specialist: Evaluating T1 EXACTLY-AFTER T2 7200
Loading story sentence ...
WFF41: (T1 EXACTLY-AFTER T2 7200)
Stored in Input Array [8]
Probability: 1
Time Specialist: Entering T1 EXACTLY-AFTER T2 7200
Time Specialist: Adding duration minimum 7200 between T2 and T1
Time Specialist: Adding duration maximum 7200 between T2 and T1
T1 is after T2 by 7200 seconds.
(WFF41)

Durations are cumulative, so we should be able to ask questions about the duration between *t2* and *ep1* .

> **(q '(t2 at-most-before e1 11000))**
Questioning (T2 AT-MOST-BEFORE EP1 11000)
Time Specialist: Evaluating T2 AT-MOST-BEFORE EP1 11000
Time Specialist: Evaluated T2 AT-MOST-BEFORE EP1 11000 to YES
Answer: YES with probability 1
Yes, T2 is before the wolf greeted Little Red Riding Hood by at most 11000 seconds.
Used 0 iterations Searched to depth 0
((YES (T-WFF8) 1))

2.4 Equality Specialist

The equality specialist not only detects and saves equality information, it also transmits it back to EPILOG where it is used in low level system operations (like unification and lookup) as well. Even though we can trace the operations actually done by the specialist, we cannot trace the aftereffects inside EPILOG. This specialist automatically starts up when EPILOG does so no special steps are needed to activate it. Note in this next example that we are making use of the fact that EPILOG can "remember" a variable it has skolemized and we can use the variable to represent the skolemized constant.

> **(trace 'equality-all)**
NIL
> **(story '(e x (x wolf) (x fierce)))**
Loading story sentence ...
WFF44: (C42 WOLF)
Stored in Input Array [9]

```
    Probability:  1
  ** INFERRED FORMULA **
     WFF45: (C42 GREY)
     Rank:  1
     Parents:  (WFF10 WFF44)
     Probability:  0.8
Loading story sentence ...
    WFF46: (C42 FIERCE)
    Stored in Input Array [9]
    Probability:  1
 There is a fierce wolf.
 It is grey.
(WFF44 WFF45 WFF46)
> (story '(e y (y wolf)))
Loading story sentence ...
    WFF48: (C47 WOLF)
    Stored in Input Array [10]
    Probability:  1
  ** INFERRED FORMULA **
     WFF49: (C47 GREY)
     Rank:  1
     Parents:  (WFF10 WFF48)
     Probability:  0.8
 There is a wolf.
 It is grey.
(WFF48 WFF49)
```

Now we will equate the two existentially quantified individuals. Anything true about one is automatically true about the other. Notice that the "x" and "y" are not quantified here, so the system will "remember" the skolem constants they were skolemized to and replace the variables with those.

```
> (story '(x equal y))
   Equality Specialist: Evaluating C42 EQUAL C47
Loading story sentence ...
    WFF50: (C42 EQUAL C47)
    Stored in Input Array [11]
    Probability:  1
   Equality Specialist: Entering C42 EQUAL C47
 A wolf is a wolf.
(WFF50)
```

We can now ask if the second wolf we entered is fierce (and it should be, since the first one is fierce, and the two wolves are equal).

```
> (q '(y fierce))
```

Questioning (C47 FIERCE)
Answer: YES with probability 1
Yes, there is a fierce wolf.
Used 0 iterations Searched to depth 0
((YES (WFF46) (1 PROB43)))

In this next question, we use the assumption that user named individuals are unique.

> **(q '(lrrh equal sue))**
Questioning (LRRH EQUAL SUE)
Equality Specialist: Evaluating LRRH EQUAL SUE
Equality Specialist: Evaluated LRRH EQUAL SUE to NO
Answer: NO with probability 1
No, Little Red Riding Hood isn't SUE.
Used 0 iterations Searched to depth 0
((NO (T-WFF11) 1))

However, if we turn off the assumption that unique user given names denote unique individuals, and ask the same question, the question cannot be answered. (***question*** holds the last question asked, unless an assertion was made in between).

> **(tweak '*unique-names-assumption* nil)**
NIL
> **(q *question*)**
Questioning (LRRH EQUAL SUE)
Equality Specialist: Evaluating LRRH EQUAL SUE
No more actions on agenda
Answer: UNKNOWN
Used 12 iterations
NIL

Now we'll tweak that flag back to its default value (t).

> **(tweak '*unique-names-assumption*)**
T
> **(untrace 'equality-all)**
NIL

2.5 Color Specialist

The color specialist compares predicates about color. It can also take into consideration some operators on the color predicates which might change the comparison slightly - in particular hedging operators (like *almost* , or

sort-of). This is an optional specialist so you must activate it before it can operate.

```
> (use-spec 'color)
(COLOR-SPECIALIST)
> (trace 'color-test)
 NIL
> (story '(cape1 cape) '(cape1 crimson))
Loading story sentence ...
    WFF51: (CAPE1 CAPE)
    Stored in Input Array [12]
    Probability:  1
 CAPE1 is a cape.
Loading story sentence ...
    WFF52: (CAPE1 CRIMSON)
    Stored in Input Array [13]
    Probability:  1
WARNING: Response: Missing translation information for CRIMSON - guessing
 CAPE1 is a crimson cape.
(WFF51 WFF52)
```

Now we can ask questions about *cape1* 's color - *Is cape1 blue?* .

```
> (q '(cape1 blue))
Questioning (CAPE1 BLUE)
   Color specialist: Found relation DISJOINT between BLUE and CRIMSON
   Color specialist: Found relation DISJOINT between BLUE and CRIMSON
Answer: NO with probability 1
 No, CAPE1 is a crimson cape.
   Used 0 iterations  Searched to depth 0
((NO (WFF52) 1))
```

And *Is cape1 red?*

```
> (q '(cape1 red))
Questioning (CAPE1 RED)
   Color specialist: Found relation SUBSUMES between RED and CRIMSON
Answer: YES with probability 1
 Yes, CAPE1 is a crimson cape.
   Used 0 iterations  Searched to depth 0
((YES (WFF52) 1))
```

In this next example, note how the hedging operator expands the color predicate to include colors which it normally would not include.

```
> (story '(sky1 aqua))
Loading story sentence ...
    WFF53: (SKY1 AQUA)
```

```
    Stored in Input Array [14]
    Probability:  1
WARNING: Response: Missing translation information for AQUA - guessing
 SKY1 is aqua.
(WFF53)
```

Since *aqua* is close to *blue* , but not *blue* , if we ask

Is sky1 blue? , we should not get an affirmative response, but if we hedge it, *Is sky1 sort of blue?* , then we should get a YES answer.

```
> (q '(sky1 blue))
Questioning (SKY1 BLUE)
No more actions on agenda
Answer: UNKNOWN
   Used 6 iterations
NIL
```

```
> (q '(sky1 (sort-of blue)))
Questioning (SKY1 (SORT-OF BLUE))
   Color specialist: Found relation SUBSUMES between (SORT-OF BLUE) and AQUA
Answer: YES with probability 1
 Yes, SKY1 is aqua.
   Used 0 iterations   Searched to depth 0
((YES (WFF53) 1))
```

If you are not sure what colors are recognized by the color specialist, the display command can help.

```
> (display 'colors)
Known colors:
RUST AQUA CRIMSON TAN BEIGE SALMON LEAD BLUEGREEN
MAGENTA CHARTREUSE GREY PINK BROWN ORANGE PURPLE GREEN
YELLOW RED BLUE BLACK WHITE
NIL
```

2.6 Number Specialist

The number specialist handles the orderings between numbers, as well as some arithmetic functions. It too is an optional specialist, and so must be activated before use. This time we'll trace the "minimum" about the number specialist to see what it does.

```
> (use-spec 'number)
Specialist NUMBER-SPECIALIST is now active
(NUMBER-SPECIALIST)
```

> **(trace 'number-min)**
NIL

The number specialist recognizes several sorts of entities - *integers* ,

reals , and *number* (same as *real*). Arguments in literals to the number specialist must be of one of these sorts.

> **(story '(n1_integer < n2_real))**
Loading story sentence ...
WFF54: (N1 LESS-THAN N2)
Stored in Input Array [15]
Probability: 1
Number specialist: Entering N1 LESS-THAN N2
N1 is less than N2.
(WFF54)

Note in this examle that < was replaced by the system-preferred *less-than* .

In this next example, the number-specialist also simplifies a formula by evaluating a functional term. If for some reason the function could not be evaluated (one argument was a variable, or a constant whose value was not known), the term would remain as a functional term.

> **(story '(n1 gt= (add 9 8 6)))**
Number specialist: ADD (9 8 6) evaluated to 23
Loading story sentence ...
WFF57: (N1 GT= 23)
Stored in Input Array [16]
Rank: 0
Parents: (WFF56)
Probability: 1
Number specialist: Entering N1 GT= 23
Number specialist: Setting minimum of N1 to 23
N1 is at least 23.
(WFF57)

When an actual value or bound for a number is entered, it is propagated to other numbers related to that number (similar to the absolute time propagation done by the time specialist). In this next question we can see that *n2* has had its lower bound updated as well.

> **(q '(n2 gt 15))**
Questioning (N2 GT 15)
Number Specialist: Evaluated N2 GT 15 to YES
Answer: YES with probability 1
Yes, N2 is more than 15.
Used 0 iterations Searched to depth 0
((YES (T-WFF10) 1))

2.7 String Specialist

The string specialist can evaluate literals involving string predicates, including many Lisp string predicates, and also some string functions (including Lisp ones). It is an optional specialist and so must be activated before use.

> **(use-spec 'string)**
(STRING-SPECIALIST)
> **(trace 'string-all)**
NIL
> **(q '((string-concat "abc" "def" "ghi") string-contain (string-field "abc-ABC" 1)))**
Questioning ((STRING-CONCAT abc def ghi) STRING-CONTAIN (STRING-FIELD abc-ABC 1))
String Specialist: STRING-CONCAT (abc def ghi) evaluated to abcdefghi
String Specialist: STRING-FIELD (abc-ABC 1) evaluated to abc
String Specialist: Evaluated abcdefghi STRING-CONTAIN abc to YES
Answer: YES with probability 1
Yes, the concatenated string of abc, def, and ghi contains the part of abc-ABC.
Used 0 iterations Searched to depth 0
((YES (T-WFF12) 1))

> **(q '((string-upcase (string-trim " " " hello")) string= "HELLO"))**
Questioning ((STRING-UPCASE (STRING-TRIM hello)) STRING= HELLO)
String Specialist: Evaluated HELLO STRING= HELLO to YES
Answer: YES with probability 1
Yes, the capitalized version of the trimmed version of hello equals HELLO.
Used 0 iterations Searched to depth 0
((YES (T-WFF12) 1))

Specialists can also cooperate to answer question. Here the string and number specialists must both be involved.

> **(q '((string-number "123") lt 500))**
Questioning ((STRING-NUMBER 123) LT 500)
String Specialist: STRING-NUMBER (123) evaluated to 123
Number Specialist: Evaluated 123 LT 500 to YES
Answer: YES with probability 1
Yes, the number for 123 is less than 500.
Used 0 iterations Searched to depth 0
((YES (T-WFF11) 1))

2.8 Set Specialist

This specialist deals with set membership, union, intersection, and cardinality. This is an optional specialist and so must be activated before using. This section is quite long, as there are a number of things handled by the set specialist, and we have tried to illustrate most of them here.

```
> (use-spec 'set)
(SET-SPECIALIST)
> (trace 'set-min)
NIL
```

Not only does this specialist maintain story facts about sets in its own domain, it also asserts inferences about them back to EPILOG. These inferences are usually about set member type, or equalities detected through set membership and cardinality. Note the inference made by the set specialist in the next examples.

First let's set up two sets with member types, and give them a member each.

```
> (story '(s1_set (coll human)))
   Set Specialist: Evaluating S1 (COLL HUMAN)
Loading story sentence ...
    WFF59: (S1 (COLL HUMAN))
    Stored in Input Array [17]
    Probability: 1
   Set Specialist: Entering S1 (COLL HUMAN)
   Set Specialist: Setting member type of set S1 to HUMAN
   Set Specialist: Asserting: (A X (X MEMBER-OF S1) (X HUMAN))
 ** INFERRED FORMULA **
      WFF62: (A X (X MEMBER-OF S1) (X HUMAN))
      Rank: 1
      Parents: (WFF59)
      Probability: 1
 S1 is a group of humans.
 Members of the group of humans are humans.
(WFF59 WFF62)
```

Now if we assert a particular member of the set, the regular EPILOG core will apply the new rule to the member to give its type, but the set specialist also maintains the members in its own domain.

```
> (story '(john member-of s1))
   Set Specialist: Evaluating JOHN MEMBER-OF S1
   Set Specialist: Testing to see if JOHN is a member of set S1
Loading story sentence ...
    WFF63: (JOHN MEMBER-OF S1)
    Stored in Input Array [18]
    Probability: 1
```

```
   Set Specialist: Entering JOHN MEMBER-OF S1
   Set Specialist: Adding member JOHN to set S1
   Set Specialist:  Setting contents of set (S1) to (JOHN +)
   Set Specialist: Changing cardinality lower bound of set (S1) to 1
 ** INFERRED FORMULA **
      WFF64: (JOHN HUMAN)
      Rank:  1
      Parents:  (WFF62 WFF63)
      Probability:  1
 John is a member of a group of humans.
 He is someone.
(WFF63 WFF64)


> (display 'set-info 's1 '-f)
S1      Set             (S1)     partially known
        Member Type :   HUMAN
        Cardinality :   (1 +INFINITY)
        Contents        (JOHN)
NIL


> (story '(s2_set (coll girl)))
 Set Specialist: Evaluating S2 (COLL GIRL)
Loading story sentence ...
     WFF66: (S2 (COLL GIRL))
     Stored in Input Array [19]
     Probability:  1
   Set Specialist: Entering S2 (COLL GIRL)
   Set Specialist: Setting member type of set S2 to GIRL
   Set Specialist: Asserting: (A X (X MEMBER-OF S2) (X GIRL))
 ** INFERRED FORMULA **
      WFF68: (A X (X MEMBER-OF S2) (X GIRL))
      Rank:  1
      Parents:  (WFF66)
      Probability:  1
 S2 is a group of girls.
 Members of the group of girls are girls.
(WFF66 WFF68)


> (story '(mary member-of s2))
```

```
   Set Specialist: Evaluating MARY MEMBER-OF S2
   Set Specialist: Testing to see if MARY is a member of set S2
Loading story sentence ...
    WFF69: (MARY MEMBER-OF S2)
    Stored in Input Array [20]
    Probability:  1
   Set Specialist: Entering MARY MEMBER-OF S2
   Set Specialist: Adding member MARY to set S2
   Set Specialist:  Setting contents of set (S2) to (MARY +)
   Set Specialist: Changing cardinality lower bound of set (S2) to 1



 ** INFERRED FORMULA **
     WFF70: (MARY GIRL)
     Rank:  1
     Parents:  (WFF68 WFF69)
     Probability:  1
  ** INFERRED FORMULA **
      WFF71: (MARY PRETTY)
      Rank:  2
      Parents:  (WFF4 WFF70)
      Probability:  1
 Mary is a member of a group of girls.
 She is a girl.
 She is pretty.
(WFF69 WFF70 WFF71)
> (untrace 'equality-all)
NIL
```

Now we can create a new set which is the intersection of those two sets. Its member type will be the more specific of the two member types, and its members will be the intersection of two sets of members (in this case, none). If the same new member is added to both original sets, that member is automatically added to the intersection set as well. Note that the set specialist makes up its own name for the intersection - this way no matter how that intersection set is referred to, the appropriate set and all its information will be available.

```
> (story '(s3_set equal (intersect s1 s2)))
   Set Specialist: Asserting: (A X (X MEMBER-OF INTERSECTS1S2) (X GIRL))
   Set Specialist: INTERSECT (S1 S2) evaluated to INTERSECTS1S2
   Set Specialist: Evaluating S3 EQUAL INTERSECTS1S2
   Set Specialist: Testing to see if set S3 is equal to set INTERSECTS1S2
Loading story sentence ...
    WFF75: (S3 EQUAL INTERSECTS1S2)
    Stored in Input Array [21]
```

```
    Rank:  0
    Parents:  (WFF73)
    Probability:  1
  Set Specialist: Entering S3 EQUAL INTERSECTS1S2
  Set Specialist: Setting S3 equal to INTERSECTS1S2
  Equality Specialist:  Setting contents of set (S3-EQUAL) to (+
                                                    S3
                                                    INTERSECTS1S2)
  Equality Specialist: Changing cardinality lower bound of set (S3-EQUAL) to 2



 ** INFERRED FORMULA **
     WFF77: (A X (X MEMBER-OF INTERSECTS1S2) (X GIRL))
     Rank:  1
     Parents:  (WFF75)
     Probability:  1
 S3 is INTERSECTS1S2.
 Members of INTERSECTS1S2 are girls.
 Members of INTERSECTS1S2 are girls.
(WFF75 WFF77)



> (display 'set-info 's3 '-f)
S3     Set          (S3 INTERSECTS1S2)     partially known
       Member Type :   GIRL
       Cardinality :   unknown
       Contents        unknown
       Connections:
            INTERSECT (S1 S2)
NIL
```

In this next example, the same two sets are combined with *union* - the member-type is the less specific of the member types, and the members themselves are the union of the members from both sets. Any new member added to either set will also be added to the union set.

```
> (story '(s4_set equal (union-of s1 s2)))
  Set Specialist: Asserting: (A X (X MEMBER-OF UNION-OFS1S2) (X HUMAN))
  Set Specialist: UNION-OF (S1 S2) evaluated to UNION-OFS1S2
  Set Specialist: Evaluating S4 EQUAL UNION-OFS1S2
  Set Specialist: Testing to see if set S4 is equal to set UNION-OFS1S2
Loading story sentence ...
    WFF80: (S4 EQUAL UNION-OFS1S2)
    Stored in Input Array [22]
```

```
    Rank:  0
    Parents:  (WFF79)
    Probability:  1
  Set Specialist: Entering S4 EQUAL UNION-OFS1S2
  Set Specialist: Setting S4 equal to UNION-OFS1S2
  Equality Specialist:  Setting contents of set (S4-EQUAL) to (+
                                                    S4
                                                    UNION-OFS1S2)
  Equality Specialist: Changing cardinality lower bound of set (S4-EQUAL) to 2



** INFERRED FORMULA **
    WFF82: (A X (X MEMBER-OF UNION-OFS1S2) (X HUMAN))
    Rank:  1
    Parents:  (WFF80)
    Probability:  1
S4 is UNION-OFS1S2.
Members of UNION-OFS1S2 are humans.
Members of UNION-OFS1S2 are humans.
(WFF80 WFF82)
```

Now we can ask if *Mary* is a member of this new set. She should be, as she is a member of one of the sets in the union that makes up this set.

```
> (q '(mary member-of s4))
Questioning (MARY MEMBER-OF S4)
   Set Specialist: Evaluating MARY MEMBER-OF S4
   Set Specialist: Testing to see if MARY is a member of set S4
   Set Specialist: Evaluated MARY MEMBER-OF S4 to YES
Answer: YES with probability 1
 Yes, Mary is a member of S4.
   Used 0 iterations  Searched to depth 0
((YES (T-WFF13) 1))
```

Now if we add a new member to one of the sets making up the union, this should also be reflected in *s4* .

```
> (story '(lrrh member-of s1))
   Set Specialist: Evaluating LRRH MEMBER-OF S1
   Set Specialist: Testing to see if LRRH is a member of set S1
Loading story sentence ...
    WFF83: (LRRH MEMBER-OF S1)
    Stored in Input Array [23]
    Probability:  1
```

Set Specialist: Entering LRRH MEMBER-OF S1
Set Specialist: Adding member LRRH to set S1
Set Specialist: Setting contents of set (S1) to (LRRH JOHN +)
Set Specialist: Changing cardinality lower bound of set (S1) to 2
Set Specialist: Adding member LRRH to set UNION-OFS1S2
Set Specialist: Setting contents of set (S4 UNION-OFS1S2) to (LRRH MARY JOHN +)
Set Specialist: Changing cardinality lower bound of set (S4
UNION-OFS1S2) to 3
Little Red Riding Hood is a member of a group of humans.
(WFF83)

Here we did not need to make the inference *(lrrh human)* because it was already true.

> **(display 'set-info 's4 '-f)**
S4 Set (S4 UNION-OFS1S2) partially known
Member Type : HUMAN
Cardinality : (3 +INFINITY)
Contents (LRRH MARY JOHN)
Connections:
UNION-OF (S1 S2)
NIL

Just to show that intersections are also updated, we can add the same new member to the other set making up the intersection for *s3* .

> **(story '(lrrh member-of s2))**
Set Specialist: Evaluating LRRH MEMBER-OF S2
Set Specialist: Testing to see if LRRH is a member of set S2
Loading story sentence ...
WFF85: (LRRH MEMBER-OF S2)
Stored in Input Array [24]
Probability: 1
Set Specialist: Entering LRRH MEMBER-OF S2
Set Specialist: Adding member LRRH to set S2
Set Specialist: Setting contents of set (S2) to (LRRH MARY +)
Set Specialist: Changing cardinality lower bound of set (S2) to 2
Set Specialist: Adding member LRRH to set INTERSECTS1S2
Set Specialist: Setting contents of set (S3 INTERSECTS1S2) to (LRRH +)
Set Specialist: Changing cardinality lower bound of set (S3 INTERSECTS1S2) to 1
Little Red Riding Hood is a member of a group of girls.
(WFF85)

Here we did not need to make the inference *(lrrh girl)* because it was already true.

The set specialist also deals with set cardinality.

```
> (story '(s5_set has-cardinality 5))
   Set Specialist: Evaluating S5 HAS-CARDINALITY 5
Loading story sentence ...
    WFF87: (S5 HAS-CARDINALITY 5)
    Stored in Input Array [25]
    Probability:  1
   Set Specialist: Entering S5 HAS-CARDINALITY 5
   Set Specialist: Setting cardinality of set S5 to 5
 S5 has 5 members' cardinality.
(WFF87)
```

```
> (display 'set-info 's5 '-f)
S5      Set             (S5)       partially known
        Cardinality :   5
        Contents        unknown
NIL
```

And ask questions about it:

```
> (q '((cardinality-of s5) gt 3))
Questioning ((CARDINALITY-OF S5) GT 3)
   Set Specialist: CARDINALITY-OF (S5) evaluated to 5
   Number Specialist: Evaluated 5 GT 3 to YES
Answer: YES with probability 1
 Yes, S5's cardinality is more than 3.
   Used 0 iterations   Searched to depth 0
((YES (T-WFF16) 1))
```

The operator *coll-of* adds both member type and cardinality at once. This next example builds a set of 3 humans, whose exact members are not known.

```
> (story '(s6_set ((coll-of 3) human)))
   Set Specialist: Evaluating S6 ((COLL-OF 3) HUMAN)
Loading story sentence ...
    WFF90: (S6 ((COLL-OF 3) HUMAN))
    Stored in Input Array [26]
    Probability:  1
   Set Specialist: Entering S6 ((COLL-OF 3) HUMAN)
   Set Specialist: Setting member type of set S6 to HUMAN
   Set Specialist: Asserting: (A X (X MEMBER-OF S6) (X HUMAN))
   Set Specialist: Setting cardinality of set S6 to 3
  ** INFERRED FORMULA **
```

```
    WFF92: (A X (X MEMBER-OF S6) (X HUMAN))
    Rank:  1
    Parents:  (WFF90)
    Probability:  1
 S6 is a group of 3 humans.
 Members of the group of 3 humans are humans.
(WFF90 WFF92)


> (display 'set-info 's6 '-f)
S6     Set          (S6)      partially known
       Member Type :   HUMAN
       Cardinality :   3
       Contents        unknown
NIL
```

2.9 Part Specialist

The part specialist deals with part-of relationships and part types. It is started up automatically when EPILOG starts up, so does not need to be activated. It uses a part hierarchy, similar to the type hierarchy, as well as maintaining the explicit relationship between objects and parts from assertions. The part hierarchy keeps track of subparts as well as the number of such parts allowed. The part specialist knows what parts a thing CAN have, what parts it CANNOT have, but never assumes that the thing actually HAS a part. This must be done with explicit assertions.

Part hierarchies are assumed exhaustive (unless otherwise specified), and may contain leaf nodes such as *arm-other* or *foot-rest* which cover those parts of the hierarchy which we do not know exhaustively.

Let's try entering some parts and part-of relationships.

```
> (trace 'part-min)
NIL
> (story '(a1 arm) '(a1 part-of lrrh) '(f1 finger) '(f1 part-of a1))
   Part Specialist: Evaluating A1 ARM
Loading story sentence ...
    WFF93: (A1 ARM)
    Stored in Input Array [27]
    Probability:  1
   Part Specialist: Entering A1 ARM
   Part specialist: Adding part type ARM to A1
 A1 is an arm.


   Part Specialist: Evaluating A1 PART-OF LRRH
```

```
Loading story sentence ...
     WFF94: (A1 PART-OF LRRH)
     Stored in Input Array [28]
     Probability:  1
    Part Specialist: Entering A1 PART-OF LRRH
    Part specialist: Adding part A1 to LRRH
    Part Specialist: Asserting (A1 GIRL-ARM)
  Normalization: Split GIRL-ARM into GIRL and ARM
    - assuming GIRL-ARM is similar to ARM
      Part Specialist: Evaluating A1 GIRL-ARM
  ** INFERRED FORMULA **
       WFF95: (A1 GIRL-ARM)
       Rank:  1
       Parents:  (WFF94)
       Probability:  1
      Part Specialist: Entering A1 GIRL-ARM
      Part specialist: Adding part type GIRL-ARM to A1
WARNING: Response: Missing translation information for GIRL-ARM - guessing
 Little Red Riding Hood has the girl arm.
 It is a girl arm.



    Part Specialist: Evaluating F1 FINGER
Loading story sentence ...
     WFF96: (F1 FINGER)
     Stored in Input Array [29]
     Probability:  1
    Part Specialist: Entering F1 FINGER
    Part specialist: Adding part type FINGER to F1
 F1 is a finger.
    Part Specialist: Evaluating F1 PART-OF A1



Loading story sentence ...
     WFF97: (F1 PART-OF A1)
     Stored in Input Array [30]
     Probability:  1
    Part Specialist: Entering F1 PART-OF A1
    Part specialist: Adding part F1 to A1
    Part Specialist: Asserting (F1 GIRL-FINGER)
  Normalization: Split GIRL-FINGER into GIRL and FINGER
    - assuming GIRL-FINGER is similar to FINGER
      Part Specialist: Evaluating F1 GIRL-FINGER
```

** INFERRED FORMULA **
WFF98: (F1 GIRL-FINGER)
Rank: 1
Parents: (WFF97)
Probability: 1
Part Specialist: Entering F1 GIRL-FINGER
Part specialist: Adding part type GIRL-FINGER to F1
WARNING: Response: Missing translation information for GIRL-FINGER - guessing
The girl arm has the girl finger.
The finger is a girl finger.
(WFF93 WFF94 WFF95 WFF96 WFF97 WFF98)

Now we can ask a question which requires the transitive property of *part-of* .

> **(q '(f1 part-of lrrh))**
Questioning (F1 PART-OF LRRH)
Part Specialist: Evaluating F1 PART-OF LRRH
Part Specialist: Evaluated F1 PART-OF LRRH to YES
Answer: YES with probability 1
Yes, Little Red Riding Hood has the girl finger.
Used 0 iterations Searched to depth 0
((YES (T-WFF15) 1))

The part specialist can also be used to determine that a particular entity cannot have a particular part (if the part is a "familiar" part). This makes use of exhaustiveness of the part hierarchies, as well as a "normal" level where such a part would occur in a hierarchy if it were there. In this next question we ask *Does Little Red Riding Hood have a tail?* .

> **(q '(e x (x tail) (x part-of lrrh)))**
Questioning (E U (U TAIL) (U PART-OF LRRH))
Part Specialist: Evaluating C99 TAIL
Part Specialist: Entering C99 TAIL
Part specialist: Adding part type TAIL to C99
Part Specialist: Evaluating C99 PART-OF LRRH
Part Specialist: Evaluated C99 PART-OF LRRH to NO
Answer: NO with probability 1
No, Little Red Riding Hood doesn't have a tail.
Used 0 iterations Searched to depth 0
((NO (T-WFF25) 1))

Little Red Riding Hood is a *girl* and therefore a *human* and since there is no special hierarchy for *girl* she inherits the *human* one.

Tail is a familiar part, and occurs on at least one hierarchy (the *animal* hierarchy), so the part specialist knows at what level is should occur on the

human hierarchy, if it is there. Since the hierarchy is exhaustive up to that level, and *tail* has not

been found, the specialist can infer that *tail* cannot be on the

human part hierarchy, even hidden in one of the remainder nodes (the nodes such as *arm-rest* that handle non-exhaustive parts of the hierarchy).

Sets of parts can also be added. Note in the following example that the set specialist also contributes to the maintenance of these and asserts inferences about sets of parts.

```
> (story '(ls_set ((coll-of 2) leg)) '(ls part-of lrrh))
   Set Specialist: Evaluating LS ((COLL-OF 2) LEG)
   Part Specialist: Evaluating LS ((COLL-OF 2) LEG)
Loading story sentence ...
    WFF101: (LS ((COLL-OF 2) LEG))
    Stored in Input Array [31]
    Probability:  1
   Set Specialist: Entering LS ((COLL-OF 2) LEG)
   Set Specialist: Setting member type of set LS to LEG
   Set Specialist: Asserting: (A X (X MEMBER-OF LS) (X LEG))
   Set Specialist: Setting cardinality of set LS to 2
   Part Specialist: Entering LS ((COLL-OF 2) LEG)
   Set Specialist: MEMBER-TYPE-OF (LS) evaluated to LEG
   Part specialist: Adding member part type LEG to LS


 ** INFERRED FORMULA **
     WFF104: (A X (X MEMBER-OF LS) (X LEG))
     Rank:  1
     Parents:  (WFF101)
     Probability:  1
LS is a group of 2 legs.
Members of the group of 2 legs are legs.


   Part Specialist: Evaluating LS PART-OF LRRH
   Set Specialist: MEMBER-TYPE-OF (LS) evaluated to LEG
   Set Specialist: CARDINALITY-OF (LS) evaluated to 2
Loading story sentence ...
    WFF105: (LS PART-OF LRRH)
    Stored in Input Array [32]
    Probability:  1
   Part Specialist: Entering LS PART-OF LRRH
   Part specialist: Adding part LS to LRRH
   Set Specialist: CARDINALITY-OF (LS) evaluated to 2
   Part Specialist: Asserting (A X (((X LEG) AND (X PART-OF LRRH)) <=> (X MEMBER-OF LS)))
```

```
   Part Specialist: Asserting (A X (X MEMBER-OF LS) (X GIRL-LEG))


 ** INFERRED FORMULA **
     WFF111: (A X (((X LEG) AND (X PART-OF LRRH)) <=> (X MEMBER-OF LS)))
     Rank:  1
     Parents:  (WFF105)
     Probability:  1
      Set Specialist: Evaluating LS MEMBER-OF LS
      Set Specialist: Testing to see if LS is a member of set LS
      Part Specialist: Evaluating LS LEG
      Part Specialist: Evaluated LS LEG  to NO
      Set Specialist: Evaluating LS MEMBER-OF LS
      Set Specialist: Testing to see if LS is a member of set LS
   ** INFERRED FORMULA **
       WFF113: (NOT LS MEMBER-OF LS)
       Rank:  1
       Parents:  (WFF111 WFF105)
       Probability:  1
      Set Specialist: Evaluating A1 MEMBER-OF LS
      Set Specialist: Testing to see if A1 is a member of set LS
      Set Specialist: Evaluated A1 MEMBER-OF LS to NO
      Part Specialist: Evaluating LS LEG
      Part Specialist: Evaluated LS LEG  to NO


  Normalization: Split GIRL-LEG into GIRL and LEG
    - assuming GIRL-LEG is similar to LEG
  ** INFERRED FORMULA **
      WFF115: (A X (X MEMBER-OF LS) (X GIRL-LEG))
      Rank:  1
      Parents:  (WFF105)
      Probability:  1
     Part Specialist: Evaluating LS LEG
     Part Specialist: Evaluated LS LEG  to NO
 Little Red Riding Hood has 2 legs.
 She has legs that are only members of the group of 2 legs.
 They aren't a member.
WARNING: Response: Missing translation information for GIRL-LEG - guessing
 Legs that are members of the group of 2 legs are girl.
(WFF101 WFF104 WFF105 WFF111 WFF113 WFF115)
```

Checking part totals against such set cardinalities can also be used to tell when a set of parts cannot

belong to an individual. For example, in this next example, Little Red Riding Hood cannot have a set of 4 arms!

```
> (q '(E x_set (x ((coll-of 4) arm)) (x part-of lrrh)))
Questioning (E SET-U (SET-U ((COLL-OF 4) ARM)) (SET-U PART-OF LRRH))
   Set Specialist: Evaluating SET-C116 ((COLL-OF 4) ARM)
   Part Specialist: Evaluating SET-C116 ((COLL-OF 4) ARM)
   Set Specialist: Entering SET-C116 ((COLL-OF 4) ARM)
   Set Specialist: Setting member type of set SET-C116 to ARM
   Set Specialist: Asserting: (A X (X MEMBER-OF SET-C116) (X ARM))
   Set Specialist: Setting cardinality of set SET-C116 to 4
   Part Specialist: Entering SET-C116 ((COLL-OF 4) ARM)
   Set Specialist: MEMBER-TYPE-OF (SET-C116) evaluated to ARM
   Part specialist: Adding member part type ARM to SET-C116
   Part Specialist: Evaluating SET-C116 PART-OF LRRH
   Set Specialist: MEMBER-TYPE-OF (SET-C116) evaluated to ARM
   Set Specialist: CARDINALITY-OF (SET-C116) evaluated to 4
   Part Specialist: Evaluated SET-C116 PART-OF LRRH to NO
Answer: NO with probability 1
 No, Little Red Riding Hood doesn't have 4 arms.
   Used 0 iterations  Searched to depth 0
((NO (T-WFF34) 1))
```

2.10 Predicate Hierarchy Specialist

This specialist deals with predicates which are arranged hierarchically, but are not type predicates. These hiearchies are added in exactly the same way as type hierarchies, but must have used *set-hier-type* to indicate they are non-type hierarchies before adding the hierarchy itself. The predicate hierarchy specialist is automatially activated when EPILOG starts up.

To add a predicate hierarchy:

```
> (set-hier-type 'repair 'pred-hier)
(NON-EXHUASTIVE)
> (add-hier 'repair 'repair 'weld 'glue 'sew)
 ((REPAIR WELD GLUE SEW))
```

Now we can add formulas using these predicates, and allow the predicate hierarchy specialist to recognize that *sew* ing is a kind of *repair* ing.

```
> (story '(lrrh sew cape1))
WARNING: Classification: SEW has no topic indicators - assuming (TP.UNKNOWN)
Loading story sentence ...
    WFF116: (LRRH SEW CAPE1)
    Stored in Input Array [33]
```

```
    Probability:  1
WARNING: Response: Missing translation information for SEW - guessing
 Little Red Riding Hood sews the cape.
(WFF116)
```

Note that we could get rid of the classification warning by adding topic indicators to *sew* , or to *repair* .

This next question asks *Did a human repair something?* . We are turning tracing on before the question to see the actual predicate comparison.

```
> (trace 'hier-test)
NIL
> (q '(e x (x human) (e y (y thing) (x repair y))))
Questioning (E U (U HUMAN) (E V (V THING) (U REPAIR V)))
    Hierarchy predicate specialist: compared REPAIR with SEW - result SUBSUMES
Answer: YES with probability 1
 Yes, the girl sews the cape.
    Used 2 iterations  Searched to depth 1
((YES (WFF51 WFF5 WFF116) (1 WFF51 WFF5 WFF116)))
> (untrace 'hier-test)
NIL
```

2.11 Episode Specialist

This specialist is always active, and is the very simplest of specialists, simply handling the subsumption relations among **, *, and @.

```
> (trace 'episode-test)
NIL
> (story '((lrrh happy) ** ep3_ep))
Loading story sentence ...
    WFF118: ((LRRH HAPPY) ** EP3)
    Stored in Input Array [34]
    Probability:  1
WARNING: Response: Missing translation information for HAPPY - guessing
 Little Red Riding Hood was happy.
(WFF118)
```

Now we can see the episode specialist in action by directly asking if the * relation holds when ** does, and asking if there is anything that is happy under a * relation.

```
> (q '((lrrh happy) * ep3))
Questioning ((LRRH HAPPY) * EP3)
    Episode-specialist: Compared * with ** - result SUBSUMES
Answer: YES with probability 1
```

```
 Yes, Little Red Riding Hood was happy.
   Used 0 iterations  Searched to depth 0
((YES (WFF118) 1))
> (untrace 'episode-test)
NIL



> (q '(e x (e y_ep ((x happy) * y))))
Questioning (E U (E EP-V ((U HAPPY) * EP-V)))
   Episode-specialist: Compared * with ** - result SUBSUMES
Answer: YES with probability 1
 Yes, Little Red Riding Hood was happy.
   Used 2 iterations  Searched to depth 1
((YES (WFF118) (1 WFF118)))
```

2.12 Other Specialist

This specialist is automatically activated when EPILOG starts up. It is used to connect with external (to EPILOG) routines, so first we must define some to use. This example sets up a new predicate *next-to* , which is true if two objects are directly next to each other, but false if there is anything in between, and a new function *nearest-neighbors* , which returns a set of the neighbors which are "next to" the argument given.

Note, this next example is more complicated than previous ones have been since we need to add some external routines to connect with. This section can be safely ignored unless you have plans to add externally evaluated functions and predicates.

```
> (provide 'space)
("SPACE" "SET-SPECIALIST" "STRING-SPECIALIST" "NUMBER-SPECIALIST"
 "COLOR-SPECIALIST" "TIME-SPECIALIST" "test-common" "SET-EQUAL-STUFF"
 "EQUALITY-SPECIALIST" "OTHER-SPECIALIST" ...)
> (defpackage space (:use lisp user lib spec))
#<The SPACE package>
> (in-package 'space :use '(lisp user lib spec))
#<The SPACE package>
>(defun next-to (&rest args &aux
                   (a1 (car args))
                   (a2 (cadr args)))
  (if (or (member a2 (get a1 'next) :test 'equal)
          (member a2 (get a1 'prev) :test 'equal))
      'yes
      (if (or (path-from a1 a2) (path-from a2 a1))
          'no
          'unknown))
```

```
  )
NEXT-TO



>(defun path-from (a b &optional already &aux res)
  (dolist (n (get a 'next))
        (unless (member n already :test 'equal)
              (if (equal n b)
                 (setq res t)
                 (setq res (path-from n b (push n already)))
                 )
              )
        )
  res)
PATH-FROM



>(defun nearest-neighbors (&rest args &aux
                           (a1 (car args))
                           res)
  (setq res (append (get a1 'next) (get a1 'prev)))
  (when res
       (setq res (normalize-term (new-record res 'set) '+))
  (if (traced-p 'epilog::function-eval)
     (print-line "    Space: NEAREST-NEIGHBORS " a1 " evaluated to "
              (print-info res))))
  res)
NEAREST-NEIGHBORS



>(defun set-next (&rest args &aux
                  (a1 (car args))
                  (a2 (cadr args)))
  (pushnew a2 (get a1 'next))
  (pushnew a1 (get a2 'prev))
  )
SET-NEXT
```

Now to tell EPILOG what and where they are, and to get pretty output from the response generator:

```
>  (in-package 'epilog)
#<The EPILOG package>
>  (add-predicate 'next-to :package 'space :entry-rtn 'space::set-next)
```

```
NIL
> (add-function 'nearest-neighbors :package 'space)
NIL
> (add-word 'next-to '(PP (name 1) (pre next) (prep to) (NP 2)))
(PP (NAME 1) (PRE NEXT) (PREP TO) (NP 2))
> (add-word 'nearest-neighbors '(NP (determiner the) (AP (adj nearest))
    (noun neighbor) (number plural) (PP (prep of) (NP 1))))
(NP (DETERMINER THE) (AP (ADJ NEAREST)) (NOUN NEIGHBOR) (NUMBER PLURAL)
   (PP (PREP OF) (NP 1)))
```

Now we are ready to use the new predicate and function.

```
> (trace 'other-eval 'other-entry)
NIL
> (story '(p1 next-to p2))
Loading story sentence ...
     WFF119: (P1 NEXT-TO P2)
     Stored in Input Array [35]
     Probability:  1
   Other Specialist: Entered  P1 NEXT-TO (P2)
 P1 is next to P2.
(WFF119)
```

```
> (story '(p2 next-to p3))
Loading story sentence ...
     WFF120: (P2 NEXT-TO P3)
     Stored in Input Array [36]
     Probability:  1
   Other Specialist: Entered  P2 NEXT-TO (P3)
 P2 is next to P3.
(WFF120)
```

Now we can ask questions about the *next-to* relation. *P1* is not next to *p3* because *p2* is in-between them.

```
> (q '(p1 next-to p3))
Questioning (P1 NEXT-TO P3)
   Other Specialist: Evaluated  P1 NEXT-TO (P3) to NO
Answer: NO with probability 1
 No, P1 isn't next to P3.
   Used 0 iterations  Searched to depth 0
((NO (T-WFF23) 1))
```

And *p2* IS next to *p1* because of symmetry (handled internally by the external *next-to* routine).

```
> (q '(p2 next-to p1))
Questioning (P2 NEXT-TO P1)
   Other Specialist: Evaluated  P2 NEXT-TO (P1) to YES
Answer: YES with probability 1
 Yes, P2 is next to P1.
   Used 0 iterations   Searched to depth 0
((YES (T-WFF22) 1))
```

To make proper use of the external function *nearest-neighbors* , since it returns a set, the set specialist should be active. However, we already activated it earlier, so it is available.

```
> (q '(p1 member-of (nearest-neighbors p2)))
Questioning (P1 MEMBER-OF (NEAREST-NEIGHBORS P2))
   Set Specialist: Evaluating P1 MEMBER-OF ($ 'SET P3 P1)
   Set Specialist: Testing to see if P1 is a member of set ($ 'SET P3 P1)
   Set Specialist: Evaluated P1 MEMBER-OF ($ 'SET P3 P1) to YES
Answer: YES with probability 1
 Yes, P1 is a member of P2's nearest neighbors.
   Used 0 iterations   Searched to depth 0
((YES (T-WFF23) 1))
```

2.13 Belief Specialist

The belief specialist allows the system to reason about the beliefs of other agents by simulation: the system can temporarily assume someone else's beliefs (those it knows about) as its own, and perform some inference to discover what else that person must believe as a consequence of those beliefs.

The knowledge in the system is stored in structures called *environments*. The system's own beliefs are stored in one environment, and the beliefs of each other believer the system knows about are stored in another environment. This means the system can reason about people whose beliefs differ from its own.

First we'll start up the specialist, since it is not automatically loaded with the system.

```
> (use-spec 'belief)
Specialist BELIEF-SPECIALIST is now active
(BELIEF-SPECIALIST)
> (trace 'belief-enter 'belief-evaluate)
NIL
```

Earlier we told the system that all girls are pretty. Now we'll tell it that John believes the opposite, that girls are not pretty. The transcript shows that the system runs a "simulation" of John learning that girls are not pretty.

```
> (story '(john believe (that (a x (x girl) (not x pretty)))))
```

```
Belief specialist: beginning simulation of JOHN with query (A X (X GIRL) (NOT X PRETTY))
Questioning (A X (X GIRL) (NOT X PRETTY))
No more actions on agenda
Answer: UNKNOWN
   Used 4 iterations
Belief specialist: simulation answered NIL
Loading story sentence ...
    WFF124: (JOHN BELIEVE (THAT (A X (X GIRL) (NOT X PRETTY))))
    Stored in Input Array [37]
    Probability:  1
Belief specialist: beginning simulation of JOHN adding belief (A X (X GIRL) (NOT X PRETTY))
Loading story sentence ...
    WFF127: (A X (X GIRL) (NOT X PRETTY))
    Stored in Input Array [0]
    Probability:  1
 No girl is pretty.
Belief specialist: simulation returned (WFF127)
WARNING: Response: Missing translation information for BELIEVE - guessing
 John believes that no girl is pretty.
(WFF124)
```

Now we tell the system that John believes that LRRH is a girl. It concludes, by simulation, that John believes that LRRH is not pretty.

```
> (story '(john believe (that (lrrh girl))))
Belief specialist: beginning simulation of JOHN with query (LRRH GIRL)
Questioning (LRRH GIRL)
Answer: UNKNOWN
   Used 6 iterations
Belief specialist: simulation answered NIL
Loading story sentence ...
    WFF129: (JOHN BELIEVE (THAT (LRRH GIRL)))
    Stored in Input Array [38]
    Probability:  1
Belief specialist: beginning simulation of JOHN adding belief (LRRH GIRL)
Loading story sentence ...
    WFF130: (LRRH GIRL)
    Stored in Input Array [1]
    Probability:  1
  ** INFERRED FORMULA **
      WFF131: (NOT LRRH PRETTY)
      Rank:  1
      Parents:  (WFF127 WFF130)
      Probability:  1
```

```
 Little Red Riding Hood is a girl.
 She isn't pretty.
Belief specialist: simulation returned (WFF130 WFF131)
  Belief specialist: beginning simulation of JOHN with query (NOT LRRH PRETTY)
Questioning (NOT LRRH PRETTY)
Answer: YES with probability 1
 Yes, Little Red Riding Hood isn't pretty.
    Used 0.0 seconds CPU time   0.053 seconds real time
    Used 0 iterations   Searched to depth 0
Belief specialist: simulation answered YES
** INFERRED FORMULA **
     WFF134: (JOHN BELIEVE (THAT (NOT LRRH PRETTY)))
     Rank:  1
     Parents:  (WFF129)
     Probability:  1
Belief specialist: beginning simulation of JOHN adding belief (NOT LRRH PRETTY)
    NOT LOADed!
      Existing formula: WFF131: (NOT LRRH PRETTY)
       Inferred from:
          WFF127: (A X (X GIRL) (NOT X PRETTY))
          WFF130: (LRRH GIRL)
Belief specialist: simulation returned NIL
 John believes that Little Red Riding Hood is a girl.
 He believes that she isn't pretty.
(WFF129 WFF134)
```

The system can also answer questions about agents' beliefs. Here we ask whether John believes LRRH is pretty.

```
>  (q '(john believe (that (lrrh pretty))))
Questioning (JOHN BELIEVE (THAT (LRRH PRETTY)))
Belief specialist: beginning simulation of JOHN with query (LRRH PRETTY)
Questioning (LRRH PRETTY)
Answer: NO with probability 1
 No, Little Red Riding Hood isn't pretty.
    Used 0.0 seconds CPU time   0.011 seconds real time
    Used 0 iterations   Searched to depth 0
Belief specialist: simulation answered NO
Answer: NO with probability 1
 No, John doesn't believe that Little Red Riding Hood is pretty.
    Used 0 iterations   Searched to depth 0
((NO (T-WFF25) 1))
```

In a typical application, there will be a vast amount of world knowledge which the system believes,

and which it also believes that others believe. Rather than requiring all of this knowledge to be duplicated in the system's environment and in all of the simulation environments, the belief specialist provides a special environment called **shared-kb**, the contents of which are accessible in all other environments. To enter information into the shared environment, temporarily bind the global variable **environment**, which indicates the currently active environment, to **shared-kb**. Here we tell the system that everyone believes that wolves are hairy.

```
> (let ((*environment* belief:*shared-kb*))
    (kn '(a x (x wolf) (x hairy))))
Loading knowledge ...
   WFF137: (A X (X WOLF) (X HAIRY))
   Probability:  1
WARNING: Response: Missing translation information for HAIRY - guessing
 Wolves are hairy.
(WFF137)
```

Next, we show that the system knows that wolf1 is hairy.

```
> (q '(wolf1 hairy))
Questioning (WOLF1 HAIRY)
Answer: YES with probability 1
 Yes, WOLF1 is a wolf.
 Wolves are hairy.
   Used 2 iterations  Searched to depth 1
((YES (WFF137 WFF1) (1 WFF137 WFF1)))
```

If we tell the system that John believes wolf1 is a wolf, then it infers that John believes the wolf is hairy.

```
> (kn '(john believe (that (wolf1 wolf))))
Belief specialist: beginning simulation of JOHN with query (WOLF1 WOLF)
Questioning (WOLF1 WOLF)
Answer: UNKNOWN
   Used 6 iterations
Belief specialist: simulation answered NIL
Loading knowledge ...
   WFF139: (JOHN BELIEVE (THAT (WOLF1 WOLF)))
   Probability:  1
Belief specialist: beginning simulation of JOHN adding belief (WOLF1 WOLF)
Loading story sentence ...
   WFF140: (WOLF1 WOLF)
   Stored in Input Array [3]
   Probability:  1
 ** INFERRED FORMULA **
```

```
      WFF141: (WOLF1 HAIRY)
      Rank:  1
      Parents:  (WFF137 WFF140)
      Probability:  1
 WOLF1 is a wolf.
 It is hairy.
Belief specialist: simulation returned (WFF140 WFF141)
  ** INFERRED FORMULA **
      WFF144: (JOHN BELIEVE (THAT (WOLF1 HAIRY)))
      Rank:  1
      Parents:  (WFF139)
      Probability:  1
  Belief specialist: beginning simulation of JOHN adding belief (WOLF1 HAIRY)
    NOT LOADed!
     Existing formula: WFF141: (WOLF1 HAIRY)
      Inferred from:
          WFF137: (A X (X WOLF) (X HAIRY))
          WFF140: (WOLF1 WOLF)
Belief specialist: simulation returned NIL
 John believes that WOLF1 is a wolf.
 He believes that it is hairy.
(WFF139 WFF144)
```

2.14 Specialist Communication

Sometimes a specialist may need extra information to do its task - information which could be supplied by another specialist. The specialist interface has been set up so that specialists can request that someone evaluate a functional term or literal for them, and they don't have to know which specialist can do it, or even if there is one that can.

In addition, sometimes this information is not available when needed on entry, but later becomes available or changes. In either case, the specialist would like to be notified.

2.14.1 Immediate Evaluation

First lets set up tracing of the specialist interface so we can see what happens. Then we'll set up some information in the number specialist.

```
> (trace 'spec-int)
NIL
> (story '(d1_integer = 3600))
  Specialists: Evaluating (D1 EQUAL 3600)
Loading story sentence ...
     WFF145: (D1 EQUAL 3600)
```

```
    Stored in Input Array [39]
    Probability:  1
  Specialists: Entering (D1 EQUAL 3600)
    Number specialist: Entering D1 EQUAL 3600
    Number specialist: Setting value of D1 to 3600
 D1 is 3600.
(WFF145)
```

In this example, the time specialist needs the information in the number specialist to update the duration bounds, so it asks for functional term evaluations for *(max-of d1)* and *(min-of d1)* . The specialist interface does all the work of deciding who to send the request to.

```
> (story '(ep1 has-duration d1))
  Specialists: Evaluating (EP1 HAS-DURATION D1)
Loading story sentence ...
    WFF146: (EP1 HAS-DURATION D1)
    Stored in Input Array [40]
    Probability:  1
  Specialists: Entering (EP1 HAS-DURATION D1)
    Time Specialist: Entering EP1 HAS-DURATION D1
  Specialist interface: Adding to interested party list of D1
          TIME-SPECIALIST / WFF146: (EP1 HAS-DURATION D1)
  Specialists: Evaluating function MIN-OF (D1)
    Number specialist: MIN-OF (D1) evaluated to 3600
    Time Specialist: Adding duration minimum 3600 between EP1START and EP2START
    Time Specialist: Adding absolute time minimum (1991 8 5 1 0 0) to EP2START
    Time Specialist: Propagating absolute time minimum (1991 8 5 1 0 0) to EP2END
  Specialists: Evaluating function MAX-OF (D1)
    Number specialist: MAX-OF (D1) evaluated to 3600
    Time Specialist: Adding duration maximum 3600 between EP1START and EP2START
 The wolf greeting Little Red Riding Hood has d1 seconds' duration.
(WFF146)
```

The statement about adding to the interested party list is used for delayed communication - i.e. if the bounds on *d1* became more strict (here they can't), the time specialist would also want to update duration bounds for *ep1* .

A specialist can also ask that a literal be evaluated (to YES, NO, or UNKNOWN). Here the part specialist will ask if someone can determine for it whether

John and *Lrrh* are equal - if not, they cannot share the same part.

```
> (story '(a2 arm) '(a2 part-of john))
  Specialists: Evaluating (A2 ARM)
    Part Specialist: Evaluating A2 ARM
Loading story sentence ...
    WFF147: (A2 ARM)
```

```
    Stored in Input Array [41]
    Probability:  1
  Specialists: Entering (A2 ARM)
    Part Specialist: Entering A2 ARM
    Part specialist: Adding part type ARM to A2
 A2 is an arm.
  Specialists: Evaluating (A2 PART-OF JOHN)
    Part Specialist: Evaluating A2 PART-OF JOHN


Loading story sentence ...
    WFF148: (A2 PART-OF JOHN)
    Stored in Input Array [42]
    Probability:  1
  Specialists: Entering (A2 PART-OF JOHN)
    Part Specialist: Entering A2 PART-OF JOHN
    Part specialist: Adding part A2 to JOHN
    Part Specialist: Asserting (A2 HUMAN-ARM)
  Asserting (A2 HUMAN-ARM) from specialists
    Specialists: Evaluating (A2 HUMAN-ARM)
      Part Specialist: Evaluating A2 HUMAN-ARM


  ** INFERRED FORMULA **
      WFF149: (A2 HUMAN-ARM)
      Rank:  1
      Parents:  (WFF148)
      Probability:  1
    Specialists: Entering (A2 HUMAN-ARM)
      Part Specialist: Entering A2 HUMAN-ARM
      Part specialist: Adding part type HUMAN-ARM to A2
WARNING: Response: Missing translation information for HUMAN-ARM - guessing
 John has the human arm.
 It is a human arm.
(WFF147 WFF148 WFF149)


> (trace 'equality-all)
NIL
> (q '(a2 part-of lrrh))
Questioning (A2 PART-OF LRRH)
  Specialists: Evaluating (A2 PART-OF LRRH)
```

```
    Part Specialist: Evaluating A2 PART-OF LRRH
  Specialists: Evaluating (JOHN EQUAL LRRH)
    Equality Specialist: Evaluating JOHN EQUAL LRRH
    Equality Specialist: Evaluated JOHN EQUAL LRRH to NO
    Part Specialist: Evaluated A2 PART-OF LRRH to NO
Answer: NO with probability 1
 No, Little Red Riding Hood doesn't have the human arm.
    Used 0 iterations  Searched to depth 0
((NO (T-WFF28) 1))
> (untrace 'equality-all)
NIL
```

2.14.2 Delayed Communication

Delayed communication is used when specialists want to be notified of any new assertions or inferences about a particular entity. This is accomplished using an "interested party list" attached to each entity - this list consists of a specialist name, the literal which was entered into the specialist that made it interested in the entity, and the embedding subnet (next section).

In this next example, an upper bound is set for *d2* , which is the duration for *ep2* . In case this bound gets more strict, the time specialist wants to "watch" *d2* , so it adds to its interested party list.

```
> (story '(d2_integer lt 4000))
  Specialists: Evaluating (D2 LT 4000)
Loading story sentence ...
     WFF150: (D2 LT 4000)
     Stored in Input Array [43]
     Probability:  1
  Specialists: Entering (D2 LT 4000)
    Number specialist: Entering D2 LT 4000
    Number specialist: Setting maximum of D2 to 3999
 D2 is less than 4000.
(WFF150)



> (story '(ep2 has-duration d2))
  Specialists: Evaluating (EP2 HAS-DURATION D2)
Loading story sentence ...
     WFF151: (EP2 HAS-DURATION D2)
     Stored in Input Array [44]
     Probability:  1
  Specialists: Entering (EP2 HAS-DURATION D2)
    Time Specialist: Entering EP2 HAS-DURATION D2
  Specialist interface: Adding to interested party list of D2
```

```
        TIME-SPECIALIST / WFF151: (EP2 HAS-DURATION D2)
  Specialists: Evaluating function MIN-OF (D2)
    Number specialist: MIN-OF (D2) evaluated to NIL
  Specialists: Evaluating function MAX-OF (D2)
    Number specialist: MAX-OF (D2) evaluated to 3999
    Time Specialist: Adding duration maximum 3999 between EP2START and EP2END
 The wolf snarling at Little Red Riding Hood has d2 seconds' duration.
(WFF151)
```

Now if we update the bound on *d2* , we can watch the original literal reasserted to the time specialist. When it does its immediate evaluation for the *(max-of d2)* , this time it will get the new bound, and update

ep2 's duration accordingly.

```
> (story '(d2 less-than 3600))
  Specialists: Evaluating (D2 LESS-THAN 3600)
Loading story sentence ...
     WFF152: (D2 LESS-THAN 3600)
     Stored in Input Array [45]
     Probability:  1
  Specialists: Entering (D2 LESS-THAN 3600)
    Number specialist: Entering D2 LESS-THAN 3600
    Number specialist: Setting maximum of D2 to 3599
  Specialist interface: Reasserting from interested party list of D2
        TIME-SPECIALIST / WFF151: (EP2 HAS-DURATION D2)
    Time Specialist: Entering EP2 HAS-DURATION D2
  Specialists: Evaluating function MIN-OF (D2)
    Number specialist: MIN-OF (D2) evaluated to NIL
  Specialists: Evaluating function MAX-OF (D2)
    Number specialist: MAX-OF (D2) evaluated to 3599
    Time Specialist: Adding duration maximum 3599 between EP2START and EP2END
 D2 is less than 3600.
(WFF152)
```

Now lets "untrace" some of this for the next examples.

```
> (untrace 'interested-party 'spec-eval 'function-eval)
NIL
```

2.15 Flattening of Asserted Formulas

Because functions are used mainly for evaluation, entry of information using the functions doesn't always update specialist information properly. Specialists will only accept assertions with simple constants or records as arguments - not un-evaluated functional terms. However, sometimes it is easier to express

information using the functions, and it is more consistent to be able to do so because one can evaluate using them.

To get around this problem, the specialist interface "flattens" such formulas when they are asserted. The specialist which is interested in the function supplies a relational predicate to use to help with this process, and the formula is split into several, simpler formulas. New constants are invented to correspond to the functional terms in these formulas.

This is best seen in an example. In this next example, we want to say that the cardinality of the set *set1* is less than 5. We could say this as

(set1 has-cardinality c1) and (c1 lt 5) (in which case the above problem does not occur), or we could say it as *((cardinality-of set1) lt 5)* . In the latter case, the system "flattens" the formula by transforming it into the former case, automatically - the function *cardinality-of* is paired with the predicate *has-duration* .

```
> (story '((cardinality-of set1_set) less-than 5))
   Set Specialist: CARDINALITY-OF (SET1) evaluated to NIL
Loading story sentence ...
    WFF154: ((CARDINALITY-OF SET1) LESS-THAN 5)
    Stored in Input Array [46]
    Probability:  1
  Specialists: Entering (CARDINALITY-OFSET1 LESS-THAN 5)
   Number specialist: Entering CARDINALITY-OFSET1 LESS-THAN 5
   Number specialist: Setting maximum of CARDINALITY-OFSET1 to 4
  Specialists: Entering (SET1 HAS-CARDINALITY CARDINALITY-OFSET1)
   Set Specialist: Entering SET1 HAS-CARDINALITY CARDINALITY-OFSET1
   Number specialist: MIN-OF (CARDINALITY-OFSET1) evaluated to NIL
   Number specialist: MAX-OF (CARDINALITY-OFSET1) evaluated to 4
   Set Specialist: Setting cardinality lower bound of set SET1 to CARDINALITY-OFSET1
   Set Specialist: Setting cardinality upper bound of set SET1 to 4
 SET1's cardinality is less than 5.
(WFF154)
```

You can see that the formula was split into two, and these were entered into the appropriate specialists. Note that they were NOT entered into EPILOG in this split form though - just into the specialists.

The time-specialist has two such functions - *elapsed* , which is paired with predicate *has-duration* , and *duration-of* , which is paired with *exactly-before* . Note in this next example that all such functional terms are flattened, so the original is split into THREE formulas this time.

```
> (story '((duration-of ep1_ep) > (duration-of ep3_ep)))
Loading story sentence ...
    WFF157: ((DURATION-OF EP1) GREATER-THAN (DURATION-OF EP3))
    Stored in Input Array [47]
    Probability:  1
  Specialists: Entering (DURATION-OFEP1 GREATER-THAN DURATION-OFEP3)
   Number specialist: Entering DURATION-OFEP1 GREATER-THAN DURATION-OFEP3
```

Specialists: Entering (EP1 HAS-DURATION DURATION-OFEP1)
Time Specialist: Entering EP1 HAS-DURATION DURATION-OFEP1
Number specialist: MIN-OF (DURATION-OFEP1) evaluated to NIL
Number specialist: MAX-OF (DURATION-OFEP1) evaluated to NIL
Specialists: Entering (EP3 HAS-DURATION DURATION-OFEP3)
Time Specialist: Entering EP3 HAS-DURATION DURATION-OFEP3
Number specialist: MIN-OF (DURATION-OFEP3) evaluated to NIL
Number specialist: MAX-OF (DURATION-OFEP3) evaluated to NIL
The duration of the wolf greeting Little Red Riding Hood is more than the duration of she being happy.
(WFF157)

And now to see the same thing with *elapsed* :

> **(story '((elapsed ep1 ep3) equal 7200))**
Loading story sentence ...
WFF159: ((ELAPSED EP1 EP3) EQUAL 7200)
Stored in Input Array [48]
Probability: 1
Specialists: Entering (ELAPSEDEP1EP3 EQUAL 7200)
Number specialist: Entering ELAPSEDEP1EP3 EQUAL 7200
Number specialist: Setting value of ELAPSEDEP1EP3 to 7200
Specialists: Entering (EP1 EXACTLY-BEFORE EP3 ELAPSEDEP1EP3)
Time Specialist: Entering EP1 EXACTLY-BEFORE EP3 ELAPSEDEP1EP3
Time Specialist: Adding absolute time minimum (1991 8 5 1 0 0) to EP3START
Time Specialist: Propagating absolute time minimum (1991 8 5 1 0 0) to EP3END
Number specialist: MIN-OF (ELAPSEDEP1EP3) evaluated to 7200
Time Specialist: Adding duration minimum 7200 between EP2START and EP3START
Time Specialist: Adding absolute time minimum (1991 8 5 3 0 0) to EP3START
Time Specialist: Propagating absolute time minimum (1991 8 5 3 0 0) to EP3END
Number specialist: MAX-OF (ELAPSEDEP1EP3) evaluated to 7200
Time Specialist: Adding duration maximum 7200 between EP2START and EP3START
The elapsed time between the wolf greeting Little Red Riding Hood and she being happy is 7200.
(WFF159)

2.16 Specialist Subnets for Modal Embedding

It is possible for different people to say different things about the same objects. It is also possible for the same person to have different mental attitudes about the same objects - he may tell someone one thing while hoping for another, for example. To help maintain consistency within the specialists, facts which are modally embedded are stored separately in different "subnets". The subnet identifier is made up of the arguments and predicate(s) in the modal embedding context. If there is no embedding context, the subnet identifier is /. If the formula is *(john say ...)* , the context is */john-say* , for *(john tell mary ...)* it is */johnmary-tell* , and for

(john say (that (mary say ...))) it is */john-say/mary-say* .

Specialists enter information for different subnets into different places in their domains, and when asked to evaluate a literal for a particular subnet will use ONLY that information stored in that subnet area - no inheritance is done. So if we ask whether John says that Mary thinks something, we will not check any of John's beliefs other than those concerning what Mary thinks.

Earlier we told the system that event *ep1* is *before-0* event *ep2* , with no embedding context. Now note what happens when we assert that John says something to the contrary.

> **(story '(john say (that (ep1 after-1 ep2))))**
Time Specialist: Evaluating EP1 AFTER-1 EP2 in subnet /JOHN-SAY
Loading story sentence ...
WFF162: (JOHN SAY (THAT (EP1 AFTER-1 EP2)))
Stored in Input Array [49]
Probability: 1
Specialists: Entering (EP1 AFTER-1 EP2) in subnet(s) (/JOHN-SAY)
Time Specialist: Entering EP1 AFTER-1 EP2 into subnet /JOHN-SAY
WARNING: Response: Missing translation information for SAY - guessing
John says that the wolf greeted Little Red Riding Hood after it snarled at her.
(WFF162)

Notice that when the time specialist said it was entering the information, it also indicated that it was going into subnet */john-say* . Now we can ask questions for both the top level and John's subnets:

Did event ep1 occur before event ep2? .

> **(q '(ep1 before ep2))**
Questioning (EP1 BEFORE EP2)
Time Specialist: Evaluating EP1 BEFORE EP2
Time Specialist: Evaluated EP1 BEFORE EP2 to YES
Answer: YES with probability 1
Yes, the wolf greeted Little Red Riding Hood before it snarled at her.
Used 0 iterations Searched to depth 0
((YES (T-WFF45) 1))

And now we can ask a modally embedded question:

Does John say that event ep1 occured before event ep2? .

> **(q '(john say (that (ep1 before ep2))))**
Questioning (JOHN SAY (THAT (EP1 BEFORE EP2)))
Time Specialist: Evaluating EP1 BEFORE EP2 in subnet /JOHN-SAY
Time Specialist: Evaluated EP1 BEFORE EP2 to NO
Answer: NO with probability 1
No, John doesn't say that the wolf greeted Little Red Riding Hood before it snarled at her.
Used 0 iterations Searched to depth 0
((NO (T-WFF48) 1))

Note that the answers are different, as they should be.

2.17 Literal Comparison

When literal comparison done by the specialists, a successful comparison results in the specialist adding a subgoal action to the agenda, consisting of the comparison information, which includes substitutions. In this next example, we ask the system if every event was properly during event *ep1* . This is false, of course, since we know that *ep2* occurred immediately after *ep1* . The time specialist here compares the set-of-support formula in the disproof with the known fact about *ep1* and *ep2* .

```
> (trace 'qa-int 'time-test)
NIL
> (q '(a x_ep (x during-1-1 ep1)))
Questioning (A EP-X (EP-X DURING-1-1 EP1))
Adding PROOF subgoal to prove: (A EP-U (EP-U DURING-1-1 EP1))
Assuming (E EP-U NIL) PROOF
   Time Specialist: Evaluating EP-C163 DURING-1-1 EP1
Adding PROOF subgoal to prove: (EP-C163 DURING-1-1 EP1)
Adding DISPROOF subgoal to prove: (E EP-X (NOT EP-X DURING-1-1 EP1))
  Specialists: Trying to determine compatibility of (NOT EP-X DURING-1-1 EP1)
          and ((ELAPSED EP1 EP3) EQUAL 7200)
  Specialists: Trying to determine compatibility of (NOT EP-X DURING-1-1 EP1)
          and (EP1 HAS-DURATION D1)
  Specialists: Trying to determine compatibility of (NOT EP-X DURING-1-1 EP1)
          and (EP1 AFTER ($ 'TIME 1991 8 5 0 0 0))
    Time Specialist: Trying to determine compatibility of  (NOT EP-X DURING-1-1 EP1)
            and (EP1 AFTER ($ 'TIME 1991 8 5 0 0 0))
      Time Specialist: Comparing (NOT EP1 DURING-1-1 EP1) with (EP1 AFTER ($ 'TIME 1991 8 5 0 0 0))
       Time Specialist: Comparing (NOT ($ 'TIME 1991 8 5 0 0 0) DURING-1-1 EP1) with (EP1 AFTER ($
'TIME 1991 8 5 0 0 0))
    Time Specialist: Evaluating ($ 'TIME 1991 8 5 0 0 0) DURING-1-1 EP1
    Time Specialist: Evaluated ($ 'TIME 1991 8 5 0 0 0) DURING-1-1 EP1 to NO
    Time Specialist: Evaluated (EP1 AFTER ($ 'TIME 1991 8 5 0 0 0)) to YES
    Checking compatibility attempts with evaluations: YES and YES
    Time Specialist:Found (NOT EP-X DURING-1-1 EP1) compatible with
            (EP1 AFTER ($ 'TIME 1991 8 5 0 0 0))
      with substitutions: ((EP-X by ($ 'TIME 1991 8 5 0 0 0) 2))
  Subgoal using key (NOT EP-X DURING-1-1 EP1) in the disproof subgoal (E EP-X (NOT EP-X DURING-1-1
EP1))
     and (EP1 AFTER ($ 'TIME 1991 8 5 0 0 0)) in WFF39: (EP1 AFTER ($ 'TIME 1991 8 5 0 0 0))
       with substitutions: TERM38/EP-X  in (NOT EP-X DURING-1-1 EP1)
                  and  nothing  in (EP1 AFTER ($ 'TIME 1991 8 5 0 0 0))
    yielding ... YES       depth 1
  Answer YES for (E EP-X (NOT EP-X DURING-1-1 EP1))
```

```
** Answer: NO with probability 1
Answer: NO with probability 1
 No, the wolf greeted Little Red Riding Hood after August 5, 1991.
    Used 3 iterations  Searched to depth 1
((NO (WFF39) (1 WFF39)))
> (untrace 'qa-int 'spec-int)
NIL
```

Chapter 3

Introduction to Axiom Schemas and Meta Information

3.1 Meaning Postulate Axiom Schemas

These axiom schemas can operate over all or a number of meta-level objects such as predicates, terms, operators, formulas, etc. These allow more complex expressions about formulas, or combinations of them, as well as reducing the number of knowledge rules needed. For example, if we want to say that the operator *very* does not change the truth value of the predicate on the operator, we could either enter a separate rule for every predicate that could have *very* operating on it (e.g *(kn '(a x (x (very happy)) (x happy)))* , etc), or we could add a single meaning postulate axiom schema. These schemas are entered using the command **mp** or **meaning-postulate** . Only quantification over meta-level objects is allowed here. Any other quantification must be within a quasi-quoted expression (to be discussed later).

> **(mp '(a x_pred (a y_term ((qq (y (very x))) true) ((qq (y x)) true))))**

Loading meaning postulate ...

WFF171: (A PRED-X (A TERM-Y ((QQUOTE (TERM-Y (VERY PRED-X))) TRUE)
((QQUOTE (TERM-Y PRED-X)) TRUE)))

Probability: 1

(WFF171)

We can even qualify the objects the schema acts on using meta-level predicates. For example, we could say that a non-montonic operator, like *almost* does change the truth value of the predicate acting on the argument.

> **(mp '(a x_op (x non-monotonic) (a y_pred (a z_term ((qq (z (x y))) true) ((qq (not z y)) true)))))**

Loading meaning postulate ...

WFF182: (A OP-X (OP-X NON-MONOTONIC) (A PRED-Y
(A TERM-Z ((QQUOTE (TERM-Z (OP-X PRED-Y))) TRUE)
((QQUOTE (NOT TERM-Z PRED-Y)) TRUE))))

Probability: 1

(WFF182)

Now we can add story facts which use different predicates with the *very* operator acting on them. Note that the first mp will operate, but not the second (*very* is not non-monotonic).

```
> (story '(lrrh (very happy)))
Loading story sentence ...
    WFF184: (LRRH (VERY HAPPY))
    Stored in Input Array [50]
    Probability: 1
 ** INFERRED FORMULA **
     WFF117: (LRRH HAPPY)
     Rank: 1
     Parents: (WFF171 WFF184)
     Probability: 1
 Little Red Riding Hood is very happy.
 She is happy.
(WFF184 WFF117)
> (story '(wolf1 (very nasty)))
Loading story sentence ...
    WFF186: (WOLF1 (VERY NASTY))
    Stored in Input Array [51]
    Probability: 1
 ** INFERRED FORMULA **
     WFF187: (WOLF1 NASTY)
     Rank: 1
     Parents: (WFF171 WFF186)
     Probability: 1
WARNING: Response: Missing translation information for NASTY - guessing
 WOLF1 is a very nasty wolf.
 It is nasty.
(WFF186 WFF187)
```

More complicated schemas which have quantification over non-meta-level variables, or deal with formulas as variables need to use quasi-quote syntax. Within a quasi-quoted item, meta-level variables may be substituted for, and meta-level functions may be evaluated, but no other substitutions or evaluations take place. No meta-level quantification is allowed in quasi-quoted expressions. Quasi-quoted expresssions representing formulas are usually coupled with the meta-predicate *true* in axiom schemas. *((qq (a b c)) true)* is equivalent to *(a b c)* and is transformed into this automatically by the system if entered at the top level. The following mp describes the inference made by the set specialist when *coll* or *coll-of* are used.

```
> (mp '(A p_pred (A x_term ((qq (x (coll p))) true) ((qq (A y (y member-of x) (y p))) true))))
Loading meaning postulate ...
    WFF198: (A PRED-X (A TERM-Y ((QQUOTE (TERM-Y (COLL PRED-X))) TRUE)
          ((QQUOTE (A Z (Z MEMBER-OF TERM-Y) (Z PRED-X))) TRUE)))
    Probability: 1
```

(WFF198)

3.2 Simplification Schemas

Simplification schemas look much like mp's, except that they MUST be equivalences (i.e. use the predicate <=>), and when they operate, they REPLACE the original form by the inference made. They act as transformation rules. Note that even though they are eqivalences, they operate only in one direction - as if they were simply implications. Their purpose is to transform a less desirable input form into a more desirable one - we don't want the reverse happening if the more desirable form is entered!

For example, if we have an operator *nearly* , which operators on quantifiers, we might use one of these to transform it into a simpler syntactic form.

> **(trace 'simplification-schema)**

NIL

> **(simplification-schema '(a x_wff (a y_wff (((qq ((nearly a) z x y)) true) <=> ((qq ((E z x) 0.9 y)) true)))))**

Loading simplification schema ...

WFF209: (A WFF-X (A WFF-Y (((QQUOTE ((NEARLY A) U WFF-X WFF-Y)) TRUE)
<=> ((QQUOTE ((E U WFF-X) 0.9 WFF-Y)) TRUE))))

Probability: 1

(WFF209)

And now we can see how it replaces input forms:

> **(add-word 'fly '(S (NP 1) (VP (verb fly))))**

(S (NP 1) (VP (VERB FLY)))

> **(kn '((nearly a) x (x bird) (x fly)))**

Replacing ((NEARLY A) X (X BIRD) (X FLY)) by

(YES <=> ((QQUOTE ((E U (U BIRD)) 0.9 (U FLY))) TRUE))

Loading knowledge ...

WFF216: ((E X (X BIRD)) 0.9 (X FLY))

Rank: 0

Parents: (WFF209 WFF212)

Probability: 1

A bird usually flies.

(WFF216)

Also, we might input a negated, quantified expression which EPILOG cannot normalize properly. A simplification schema can help here, as EPILOG expects that quantified formulas will not be negated when they are stored.

> **(simplification-schema '(a x_wff (a y_wff (((qq (not (nearly a) z x y)) true)**
<=> ((qq (YES 0.11 (E z x (not y)))) true)))))

Loading simplification schema ...

WFF226: (A WFF-X (A WFF-Y (((QQUOTE (NOT (NEARLY A) U WFF-X WFF-Y)) TRUE)

```
            <=> ((QQUOTE (E U WFF-X (NOT WFF-Y))) TRUE))))
    Probability:  0.11
(WFF226)
> (kn '(not (nearly a) x (x bird) (x mean)))
Replacing (NOT (NEARLY A) X (X BIRD) (X MEAN)) by
    (YES <=> ((QQUOTE (E U (U BIRD) (NOT U MEAN))) TRUE))
Loading knowledge ...
    WFF234: (C232 BIRD)
    Rank:  0
    Parents:  (WFF226 WFF229)
    Probability:  1
 ** INFERRED FORMULA **
     WFF235: (C232 FLY)
     Rank:  1
     Parents:  (WFF216 WFF234)
     Probability:  0.9
Loading knowledge ...
    WFF236: (NOT C232 MEAN)
    Rank:  0
    Parents:  (WFF226 WFF229)
    Probability:  0.11
WARNING: Response: Missing translation information for MEAN - guessing
 There isn't a mean bird.
 It flies.
(WFF234 WFF235 WFF236)
```

3.3 Meta-Level Facts

To use mp's like the one about non-monotonic operators, we need to be able to assert facts about meta-level objects, like predicates and operators. The command **meta** is used for this. If you try to use **story** , it will complain that you are making predications about meta-level objects.

```
> (meta '(almost non-monotonic))
Loading meta information ...
    WFF237: (ALMOST NON-MONOTONIC)
    Probability:  1
(WFF237)
```

And now we can assert a story fact using *almost* , and it will trigger the meaning postulate entered earlier about non-monotonic predicates.

```
> (story '(wolf1 (almost happy)))
Loading story sentence ...
    WFF239: (WOLF1 (ALMOST HAPPY))
```

```
    Stored in Input Array [52]
    Probability:  1
 ** INFERRED FORMULA **
     WFF240: (NOT WOLF1 HAPPY)
     Rank:  1
     Parents:  (WFF182 WFF237 WFF239)
     Probability:  1
 WOLF1 is an almost happy wolf.
 It isn't happy.
(WFF239 WFF240)
```

Now let's see what happens with slightly more complicated predicates.

```
> (story '(wolf1 (almost eat) john))
Loading story sentence ...
    WFF242: (WOLF1 (ALMOST EAT) JOHN)
    Stored in Input Array [53]
    Probability:  1
 The wolf almost eats John.
(WFF242)
```

Although it looks like the mp should have applied to this, it shouldn't. If we convert this formula to its equivalent lambda form, we get *(wolf1 (L x (x (almost eat) john)))* , which does not match *(term (operator pred))* . The next example shows the proper way to enter such things. The *almost* must be over both the predicate and any other arguments.

```
> (story '(wolf1 (almost (eat john))))
Loading story sentence ...
    WFF246: (WOLF1 (ALMOST (L L-U (L-U EAT JOHN))))
    Stored in Input Array [54]
    Probability:  1
 ** INFERRED FORMULA **
     WFF247: (NOT WOLF1 EAT JOHN)
     Rank:  1
     Parents:  (WFF182 WFF237 WFF246)
     Probability:  1
 The wolf almost eats John.
 It doesn't eat him.
(WFF246 WFF247)
```

3.4 Meta Specialist

Some functions and predicates which operate on meta-level objects are predefined and handled by the meta-specialist. The predicates include information about whether something is a type-predicate, action predicate or action formula, whether one expression contains another, etc. These are used mainly for

mp's and simplification schemas. Those which begin with % must only be used in these constructs.

```
> (trace 'meta-all)
NIL
> (q '(human %type-predicate))
Questioning (HUMAN %TYPE-PREDICATE)
   Meta-specialist: Evaluating (HUMAN %TYPE-PREDICATE)
   Meta-specialist: Evaluated (HUMAN %TYPE-PREDICATE) to YES
((YES (T-WFF57) 1))
```

In the next example, the *%contains* predicate and *%subst* function are used in a simplification schema to replace any occurrence of *(very happy)* with *delerious* . Note that unlike the previous mp's and simplification schemas shown, this one will apply at ANY level of embedding. Previous ones would apply only at the top level, and if embedded in a modal or episodic context would not apply.

```
> (simplification-schema '(a x_wff (x true)
                ((x %contains (qq (extremely happy))) <=>
                 ((%subst (qq delerious) (qq (extremely happy)) x) true))))
Loading simplification schema ...
    WFF255: (A WFF-X (WFF-X TRUE) ((WFF-X %CONTAINS (QQUOTE (EXTREMELY HAPPY)))
        <=> ((%SUBST (QQUOTE DELERIOUS) (QQUOTE (EXTREMELY HAPPY)) WFF-X) TRUE)))
    Probability: 1
(WFF255)
> (story '(john (extremely happy)))
WARNING: Normalization: DELERIOUS is a new predicate
   Meta Specialist: Evaluated %SUBST ((QQUOTE DELERIOUS) (QQUOTE (EXTREMELY HAPPY))
        (JOHN (EXTREMELY HAPPY))) to (JOHN DELERIOUS)
    Meta-specialist: Evaluating ((JOHN (EXTREMELY HAPPY)) %CONTAINS (QQUOTE (EXTREMELY
HAPPY)))
    Meta-specialist: Evaluated ((JOHN (EXTREMELY HAPPY)) %CONTAINS (QQUOTE (EXTREMELY
HAPPY)))
        to YES
Replacing (JOHN (EXTREMELY HAPPY)) by
    (YES IMPLIES (YES <=>
        ((%SUBST (QQUOTE DELERIOUS) (QQUOTE (EXTREMELY HAPPY))
            (JOHN (EXTREMELY HAPPY))) TRUE)))
   Meta Specialist: Evaluated %SUBST ((QQUOTE DELERIOUS) (QQUOTE (EXTREMELY HAPPY))
        (JOHN (EXTREMELY HAPPY))) to (JOHN DELERIOUS)
WARNING: Classification: DELERIOUS has no topic indicators - assuming (TP.UNKNOWN)
Loading story sentence ...
    WFF260: (JOHN DELERIOUS)
    Stored in Input Array [55]
    Rank: 0
    Parents: (WFF259 WFF255 WFF257)
    Probability: 1
```

WARNING: Response: Missing translation information for DELERIOUS - guessing
John is delerious.
(WFF260)
> **(story '(lrrh say (that (john (extremely happy)))))**
Meta Specialist: Evaluated %SUBST ((QQUOTE DELERIOUS) (QQUOTE (EXTREMELY HAPPY))
(LRRH SAY (THAT (JOHN (EXTREMELY HAPPY)))))
to (LRRH SAY (THAT (JOHN DELERIOUS)))
Meta-specialist: Evaluating ((LRRH SAY (THAT (JOHN (EXTREMELY HAPPY)))) %CONTAINS
(QQUOTE (EXTREMELY HAPPY)))
Meta-specialist: Evaluated ((LRRH SAY (THAT (JOHN (EXTREMELY HAPPY)))) %CONTAINS
(QQUOTE (EXTREMELY HAPPY))) to YES
Replacing (LRRH SAY (THAT (JOHN (EXTREMELY HAPPY)))) by
(YES IMPLIES (YES <=>
((%SUBST (QQUOTE DELERIOUS) (QQUOTE (EXTREMELY HAPPY))
(LRRH SAY (THAT (JOHN (EXTREMELY HAPPY))))) TRUE)))
Meta Specialist: Evaluated %SUBST ((QQUOTE DELERIOUS) (QQUOTE (EXTREMELY HAPPY))
(LRRH SAY (THAT (JOHN (EXTREMELY HAPPY))))) to (LRRH SAY (THAT (JOHN DELERIOUS)))
Loading story sentence ...
WFF266: (LRRH SAY (THAT (JOHN DELERIOUS)))
Stored in Input Array [56]
Rank: 0
Parents: (WFF264 WFF255 WFF262)
Probability: 1
Little Red Riding Hood says that John is delerious.
(WFF266)

One must be VERY careful in using the *%contains* predicate and *%subst* function to ensure that by the time they are applied, all meta-variables within will have been matched. Also note that we had to add the *(x true)* key so that there would be something for the system to match on.