relations between sentences as [Hobbs 77] (p110) " an operation which matches successive sentences against a small number of common patterns and builds up a tree-like structure representing the text's patterns of coherence." There are a number of important ways in which NEXUS differs from Hobbs' theory of coherence. First, Hobbs presents a general theory of rhetorical coherence, while NEXUS represents a theory of event/state conceptual coherence. Secondly, Hobbs' theory, when applied to text, produces a single tree-like structure; NEXUS produces any number of concept trees. But the most important distinction has to do with the difference between the two sets of coherence relations, the methods used to compute coherence, and the resulting representations they produce.

Hobbs' relations are really a set of meta-relations defined over his dictionary His dictionary is composed of a set of inference rules attached to each word in the dictionary. The patterns he uses to construct the coherence relations are coded as a list of relationships that have to be established between the predicates or arguments which compose the representation of the two sentences he is trying to connect. NEXUS does not try to establish meta-relations between concepts. Instead it copies relationships between concepts in the dictionary. To see how these two theories differ in practice, we will look at two of Hobbs' examples. From Hobbs' 77 IJCAI paper [Hobbs 77] (p113)

Republicans were encouraged about their prospects.
The party chairmen believed that Dewey would be elected.

Hobbs connects these two sentences with his example coherence relation, which is defined (p113):

> . The predicate and arguments of the assertion of the current sentence stand in a subset or element-of relations to those of the previous sentence.

To connect these two sentences NEXUS would trace through the subnet of its dictionary depicted in figure 2-5.

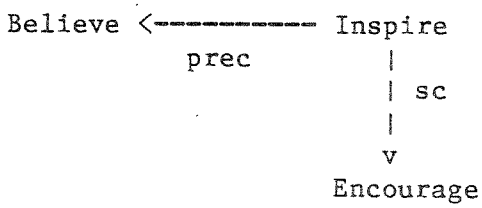

Figure 2-5: Believe and Encourage

To represent the connection between the sentences, Hobbs' system establishes a rhetorical relationship, while NEXUS selects the mesh of event/state concepts which are implicitly or explicitly used by the text; Hobbs' system characterizes the text by the style techniques employed by the author, while NEXUS' representation captures the event/state concepts implicitly in the text. From a rhetorical point of view, the second sentence is an example of the first. From a concept coherence point of view, concepts like 'encourage' and 'believe' invoke a network of concepts including 'inspire'.

Another example, from Hobbs paper on coherence and coreference [Hobbs 79] (p78):

John can open Bill's safe.
He knows the combination.

Here Hobbs uses his elaboration coherence relations to connect the two sentences.

> S1 is an Elaboration of S0 if a proposition P follows-from the assertions of both S0 and S1 (but S1 contains a property of one of the elements of P that is not in S0).

To connect these two sentences NEXUS would use the subset of its dictionary depicted in figure 2-6; to 'open' a safe one must 'know' the combination.

know(combination ...) <-------- open(safe ...)
prec

Figure 2-6: Opening a safe

Again, the connection Hobbs establishes is rhetorical, while NEXUS' is conceptual. From a style point of view, the second sentence elaborates on the first. From a concept coherence point of view, 'opening a safe' probably entails 'knowing the combination'.

2.4.7 Perspective Summarized

In this section we have investigated the breadth and nature of NEXUS' representation scheme by comparing it to other schemes discussed in the literature.

From an analysis of scripts and plans we saw that NEXUS' seven coherence relations will cover the same territory. Of course the goals of the these systems are different. Scripts represent

highly stylized routines, plans more general routines, and NEXUS the conceptual coherence of event/state descriptions in text. .

There are correspondences between NEXUS and each of these systems. If a a piece of text is covered by a single script then NEXUS will construct an instance of a single concept. If a goal and implementation of a plan are covered by a D-GOAL and a planbox or script, NEXUS collects together both the goal and the implementation of the plan into a structured chunk of information. Furthermore scripts and plans, in NEXUS, can be differentiated by the fact that script relationships have tighter restrictions on how the case arguments match.

From "The Margie Story" we learned that NEXUS could represent text which neither scripts nor plans account for; it could represent the connection between the descriptions 'the wind carried the balloon' and 'the balloon burst'. We also saw that to represent this story NEXUS produced a number of concept trees, and discussed the difference between a story tree representation, which represents the overall structure of a story, and NEXUS' representation, which captures the implicit relationships between the event/state concepts used in the text.

The section on "A Black and Yellow V-2 Rocket" discussed Simmons' schema/narrative trees, which combine story structure

information with event schemas. Each event in the tree was composed of a setting and a sequence of events. We learned that NEXUS could represent the sequence portion of the trees, but not the setting portion. The other importance of this example was that it provided evidence that coherence packets could be organized into story trees.

The "Passing the Salt" example demonstrated the difference between speech-act and concept oriented representations. For a speech-act representation to account for this example it needs to include characterizations of belief and knowledge spaces, plans, and goals. NEXUS represents speech-act text by collecting together instances of concepts without committing to an interpretation of the speaker's and hearer's belief and knowledge spaces.

Finally, the discussion of Hobbs' work explicated the difference between representations of the rhetorical coherence of text and NEXUS' representation of the conceptual coherence. A rhetorical coherence representation establishes meta-relations between sentences. NEXUS' representation is a map of the implicit relationships between event/state concepts used in the text.

Chapter 3

The Construction and Use of the Dictionary

3.1 The Dictionary

3.1.1 The Dictionary as a Semantic Network

The dictionary can be thought of as a semantic network. What is a semantic network? It is a graph - a collection of nodes and interconnecting arcs, semantic because the nodes represent concepts and the arcs define associations between concepts. In this study the nodes represent events or states and the arcs one of the seven relations that we described in the previous chapter[3] .

In chapter two we saw numerous examples of semantic net representations of text. Each representation discussed was, in fact, a semantic net. Figure 3-1 depicts a subnet of event/state concepts as might appear in the dictionary. The node labelled 'have' is a state that can occur as a consequent of either a 'giving' or an

---

[3]The discussion of semantic nets will be at what Brachman [Brachman 79] call the concept level.

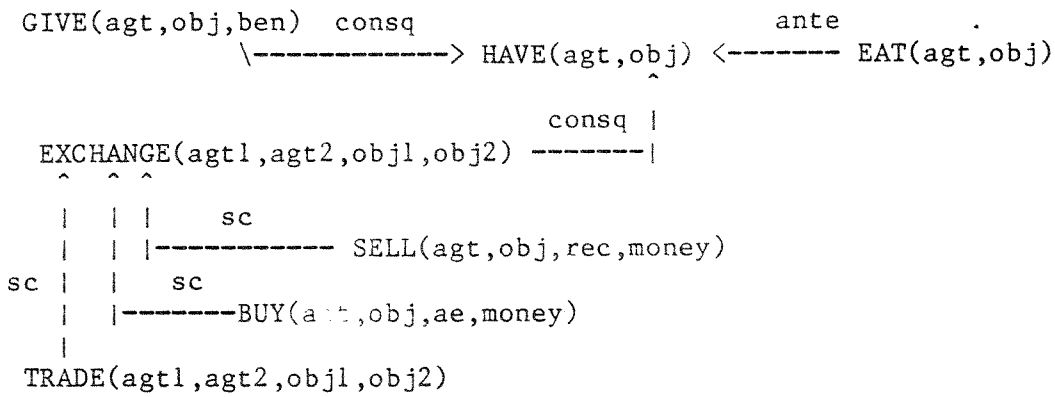

Figure 3-1: A section of a semantic network.

'exchange'. The arc from 'eat' to 'have' represents an association between 'eating' and 'having'; to 'eat' one must 'have' the thing that is being eaten. In a similar vein, the arcs from 'trade','buy', and 'sell' denote their taxonomic relationship to the concept 'exchange'.

Given a pair of sentences like,

(1) John bought some food.
He ate it.

a net interpreter could use a path-based inference scheme to find an association between the 'buying' and 'eating'. Starting at the 'buy' node the interpreter would traverse the subclass arc to the 'exchange', traverse the consequent arc to the 'have' node, and then move from 'have' to 'eat' via the antecedent* arc. Its reasoning at each stage would be something like: " A kind of exchange is buying, a consequent of an exchange is that the participants end up having

(possessing) something, and a condition for eating is that the eater has (possess) that which is eaten."

But, unfortunately, an interpreter could use the same network to relate mistakenly sentences like:

(2) John bought a kite.
He ate.

A consequent of 'buying' a kite is 'having' a kite, and an antecedent of 'eating' a kite is 'having' it? To avoid this kind of faulty reasoning, constraints are attached to arcs; So, to traverse the arc from 'have' to 'eat', the object of 'have' must match the default value, 'food', for the object of eating, and the agent of the two relations must match, thus insuring coherency and in the process resolving ellipsis and some references.

There are several reasons for choosing to work with semantic networks. First, and most importantly, the proximity of concepts in the network reflects the strength of associations between those concepts. For example, the close association between 'having' and 'eating' concepts is captured by their close connection in the dictionary's semantic net. Schubert et. al. [Schubert 79] describe the 'proximity principle' as follows:

> The fundamental assumption in the use of semantic nets is that the knowledge required to perform an intellectual task generally lies in the semantic vicinity of the concepts involved in the task.

Secondly, as a natural by-product of storing the dictionary's knowledge in a semantic network, guide-posts for knowledge retrieval are incorporated into the net; to move between nodes in the network there must be an arc, or set of arcs, connecting the two nodes. Thus the search for an association between two concepts is controlled because the arcs serve as indices.

Furthermore, constraints can be attached to arc selections. If a node n_1 has m arcs associated with it, a_1 .... a_m, and the search process arrived at node n_1 via arc a_i then a <u>proper</u> subset of the arcs a_1 .. a_{i-1} and a_{i+1} .. a_m can be used to depart from node n_1. For example, suppose the search process was trying to connect the events described in (1). Suppose the search process began at the node associated with buy and moved to the node associated with have. Two of the arcs accessible to it at this point are a consequent arc, which goes from give to have, and an antecedent* arc, which goes from have to eat. It should not try the first of these arcs because a path containing a consecutive pair of consequent and consequent* arcs makes no sense; it would be equivalent to stating that a 'having' event was an immediate consequent of two non-associated events.

Another advantage of semantic networks is the relative independence of the network and the processes that interpret it. So the content of the dictionary can be modified, and consequently TRACE's output, without necessitating changes to TRACE. Vica versa

is also true; TRACE can be changed without modifying entries into the dictionary. This flexibility greatly aids the development of the dictionary.

Given that we choose to represent the dictionary's knowledge with a semantic network there still remains the issue of how to select arc types. In part we've said that we've selected them because they were produced by an analysis of text, but that doesn't answer questions like: What are the advantages of the seven arcs that were chosen, why not a different seven arcs? Why not 50 arc types or only 1? There are several criteria that should be applied to choosing arc types. One has to do with their associative power; how much of the data can be covered by the relation types allowed in the network. The previous chapter established the adequacy of the relations for representing a large domain of coherence in text. Because of the reciprocal relationship between the dictionary and the coherence representations of text (i.e. The representations are copies of the relevant portion of the dictionary.), this effectively demonstrates that the relations can cover a wide variety of the relationships between concepts that are to be included in the dictionary.

A second criterion considers the computational advantages of the arc types. For example, what happens as the number of arc types is varied? Consider the extreme cases. Suppose the net allows only

one arc type, named 'intrinsic'. What was previously covered with seven arc types can now be covered with one. Where before the relationship between throw and move was described as consequent, now it would be described 'intrinsic', as would the relationships between carry and hold (previously coordinates) and the relationships between clean and wash (previously subsequence), etc. The disadvantage of having only one arc type is that the structure of the net would provide less useful processing information because it no longer would make any distinctions between types of relationships. So TRACE could no longer use the names associated with the arcs to constrain the path finding algorithm. Also the representations (explanations) TRACE produces would be less informative, and consequently the range of questions that the structure helps answer would be considerably shrunk, and the capacity to produce useful summaries reduced.

Consider the other extreme case, where the net has too many arc types. In the extreme case there are arc types for every possible relationship between event/state concepts, and consequently no generality in processing. TRACE would need special heuristics for each arc type in order to control the path finding algorithm. Also, since the representations (semantic nets) that TRACE produces are copies of the relationships between concepts in the dictionary, interpretations of the TRACE-produced representations would be directly affected by the number and types of relations. Consequently, QUEST and SUM's heuristics would also become more

specialized. The point is that some of the finer relationships can be collapsed into a single relationship, because the various interpreters do not process them differently. With an expanded set of relations the net would fail to capture these process-oriented generalities.

Consequently our goal lies somewhere in between. We want a small enough set of arc types so that the structure captures process-oriented generalities, which will lead to simple (elegant) algorithms for interpreting the net. But we also want the set of arc types to be large enough to allow us to represent a variety of relationships between concepts. Thus we can use the structure, instead of additional inference based on content, to process knowledge.

A third criterion is: Do the arc types of the dictionary support concept generalities? This translates into a requirement that some of the arc types be taxonomic, and therefore exploitable for property inheritance. For the example depicted in Figure 1, because 'trade', 'buy', and 'sell' are taxonomically related to 'exchange' they inherit its consequent relationship with 'have'; thus it is not necessary to include specific consequent arcs from each of the subclasses of 'exchange' to 'have'.

To summarize, we have already shown that the arc types cover a large domain of knowledge. It remains for us to show that the arc

types can represent structurally useful information and thereby can be used to construct simple, elegant, and computationally efficient net interpreters. Also we need show that NEXUS supports event/state concept generalities. In this chapter we will tackle these issues with regard to the dictionary and TRACE, and in chapter four the derived effects of these decisions with regard to SUM and QUEST.

3.1.2 Notation & Terminology

Each node in the Dictionary's network represents an event or state. A node has a name and a set of case arcs associated with it. For example, the concept 'give' has the name 'give' associated with it and the case relations agent, object, recipient, location, time, etc. Attached to each arc in the network is a set of constraints. For example, one of the arcs from 'give' to 'have' has the constraints that both the agents of 'give' and 'have' and the objects of 'give' and 'have' must match. Numbers and # marks are appended to words to differentiate between various sense meanings; thus agent and agentless uses of move would be designated move2# and move1#, respectively. Attached to each node in the network is a template that lists default values for matching. So for a drinking event the default value for the object being ingested is a liquid.

All the knowledge in the network is stored in relational form. Relations between nodes in the network are stored in a 4-tuple.

[relation, event/state1, event/state2, (constraints)]

So the antecedent relation between eating and having would be represented:

[ante eat have1# ((MATCH agt agt) (MATCH obj obj))]

Roughly the relation states: "There exists an antecedent relationship between 'eating' and 'having". To establish this relationship the agents and objects of 'eating' and 'having' must match." MATCH is one of a set of functions that are used to match the values of case arguments of two different instantiated event/state concepts. For MATCH to succeed, its two arguments must be identical, or one must be a pronoun form of the other, or they must have a taxonomic relationship, or one of the arguments is empty (i.e. (MATCH JOHN JOHN) = True, (MATCH JOHN HE) = True, (MATCH PERSON JOHN) = True, (MATCH JOHN NIL) = True).

Another example is the relationship between the concepts 'lose' and 'find'.

```
[seq find lose ((MATCH agt agt) (MATCH obj obj)
                (MATCH-C-NOTRANSFER tv not)) ]
```

Here the relationship should be read: " A sequel of 'find' is 'lose' if the agents and objects of 'find' and 'lose' match, and if the truth-value argument associated with find is 'not'". MATCH-C-NOTRANSFER succeeds only if the instantiated instance of 'find' has a truth-value argument whose value is "not", so (MATCH-C-NOTRANSFER NOT NOT) = True and (MATCH-C-NOTRANSFER NIL NOT) = False.

The default values for each node are stored in a 3-tuple.

(template event/state (list of default values)).

For example, the default object of an eating event is food.

(template eat ((obj food) ...))

The template for eat could be used to prevent an interpreter from wrongly associating the concepts 'buying' and 'eating' described in example (2). Suppose the interpreter has moved through the network from the 'buy' node to the 'have' node. The instantiated value of 'have' would be: (have (agt John) (obj kite)). To traverse the arc from 'have' to 'eat' the objects of 'eating' and 'having' must MATCH. Since the sentence "John ate" doesn't have an object argument, the interpreter would use the default object value of eat to traverse the arc from 'have' to 'eat'. (MATCH KITE FOOD) fails, so the interpreter correctly fails to find an association.

3.1.3 Converting to 4-tuples & Inheritance

Coherence diagrams were converted by hand to 4-tuples via a three step process. Each relation was converted to an arc, and each event/state concept to a node. Constraints were identified where pairs of arguments needed to match to insure coherency. Given the representation,

```
(drop agt peasant obj axe
     ante (hold agt peasant obj axe
                coord* (chop agt peasant ae tree)))
```

the coordinate relationship would be converted to

(arriving) must be on water, and the its instrument a boat. The subsequence relationship between travel and arrive is refined to be, in the case of travel3#, a subsequence relationship between travel3# and dock.

So a pair of sentence like:

John travelled to San Francisco.
He docked late in the evening.

could be connected either by descending the subclass arc from travel to travel3# and the subsequence arc from travel3# to dock, or by descending the subsequence arc from travel3# to arrive and from arrive to dock. In either case, because of the templates, the instrument and the location of the travelling will be marked as 'boat' and 'water' respectively.

3.2 Overview of TRACE

TRACE uses the knowledge base to construct coherence representations of segments of text. The relational knowledge base represents intrinsic relationships between event/state concepts. A segment of text is coherent if it is an instantiation of some portion of the knowledge base. The coherence representations that TRACE constructs are a map of a successful trace of an inference path between two concepts in the knowledge base; each concept lying on an inference path between the two instantiated concepts is added to the

representation. We can think of the procedure that TRACE follows as a process which reveals, or uncovers, the relevant portions of the underlying conceptual knowledge of event/state concepts.

TRACE considers each new event or state in sequence. Its strategy is to take the new concept and see if it can relate it to a previously instantiated portion of the knowledge base. Figure 3-2 shows the top-level flow of control. TRACE begins (step 1) by getting the first event/state description. A stack of schemas, SCHEMAS, is kept which holds in reverse textual order previously instantiated portions of the knowledge base (i.e. schemas). Since SCHEMAS is initially empty TRACE takes the failure branch of step 2, adds the new event/state description (step 2a) to SCHEMAS, and then goes back to (1) and gets the next event/state description. Again it proceeds to step 2 getting the schema at the top of SCHEMAS. Next, TRACE converts the schema to a goal-event-list format (step 2b) by deleting from the schema events which were marked as being 'completed'. The goal-event-list is a list of events that TRACE tries to connect to the new event description. 'Completed events' are events that have been concluded and will not continue to be described in the text. For a trio of sentences like:

The pig trotted to the stream.
She washed the laundry.
She dried the laundry.

when TRACE tries to relate the 'drying' to the 'washing' schema on

the stack of schemas, 'trotting' is deleted from the goal-event-list because the 'washing' event concluded it.

At step 3 TRACE tries to find an inference path from the new event/state concept to one of the instantiated concepts in the schema. If TRACE fails to find an inference path it goes back to step (2). If TRACE succeeds, it adds the new inference path to the schema (step 4), and then gets the next event/state concept (step 1).

1. Get the next event/state description.
2. Get the next instantiated portion of knowledge base (schema).
    a. If it does not exist add the new event/state description to the stack of schemas. Go to step 1.
    b. If it does exist convert it to goal-event-list format by deleting all 'completed' events.
3. Trace an inference path between the new event description and one of the concepts included in the schema.
    a. If it fails go to step 2.
4. Add a new inference path to the schema.
5. Add the schema to the stack of schemas.

Figure 3-2: Flow of Control

TRACE uses two major procedures to construct coherence diagrams (step 3). FIND-PATH attempts to find a path between the new event and one of the non-completed instantiated event/state concepts in the schema. TRY-PATH checks the semantic constraints attached to

each relationship along the inference path, simultaneously performing reference resolution, marking 'completed' events, and adding the new event to the schema. If TRY-PATH fails FIND-PATH continues its search for an inference path.

3.2.1 Find-Path

FIND-PATH's basic algorithm consists of spreading across the space of general concepts seeking a path of relations from the new instantiated event/state concept to one of the concepts attached to the schema. It begins by asserting the new event as the source and the list of non-completed instantiated concepts currently in the schema as the goal. The search proceeds in a breadth-first fashion, expanding the smallest frontier at each step. Relations are exploited to control the size of the search space; for any arbitrary concept a proper subset of its relationships can be expanded and used to continue the path-finding process. When FIND-PATH succeeds it passes the inference path back to TRACE. If TRY-PATH subsequently rejects the path, FIND-PATH picks up where it left off.

Let's begin by seeing how the search process would work if FIND-PATH did not limit the arc selection process at each step. Figure 3-3 shows a subgraph of the knowledge space. A kind of WORK is CHOP. A sequel to chopping down a tree is that the tree FALLs. A coordinate of CHOP is HOLD. A coordinate of CARRY is HOLD. An

Figure 3-3: A subgraph of the knowledge space.

antecedent of DROP is HOLD. A consequent of DROP is FALL. Constraints are shown in curly brackets.

Suppose FIND-PATH is trying to find a relationship between the schema produced by previous text, " The peasant worked all day. He was chopping wood.", and a new sentence, "He dropped his axe."

```
New instantiated event concept:
                (drop agt peasant obj axe)
Schema:         ((sc (work agt peasant per (all day))
                     (chop agt peasant ae wood)))
```

The source list would contain the concept 'drop'. The goal list would contain the concepts 'work' and 'chop'. Since the source list contains a smaller number of concepts, FIND-PATH would begin by expanding the 'drop' concept. In this case 'drop' participates in two relationships; the path from 'drop' to 'hold' and the one from 'drop' to 'fall' are added to the source list. Next, FIND-PATH tries to expand the goal list. Three paths are added to the goal list: work-chop, chop-fall, and chop-hold. At this point FIND-PATH has found two paths from 'chop' to 'drop' (i.e. chop-fall-drop and chop-hold-drop). Suppose FIND-PATH returns the chop-fall-drop path first. Because the object of the 'fall' sequel of 'chop' is constrained to be a tree TRY-PATH would reject this path. Next FIND-PATH returns the chop-hold-drop path, which TRY-PATH will accept.

Arc selection can be restricted by limiting the combinations of consecutive arc types; structurally we know that the cross product

of certain relations will lead to incorrect interpretations. During the selection process FIND-PATH avoids illegal combinations of relations by using the arc most recently added to the path as a filter. The cross product restriction can be characterized as one of two types, identity or bridge.

The identity of an instantiated event/state concept is the collection of event/state concepts which are its taxonomic offspring. The identity of chopl# is specified by its subsequences, coordinates, and subclasses. Here the restriction is motivated by the observation that an instantiated event/state concept cannot simultaneously be part of the identity of two different event/state concepts. For example, the concept 'hold' is a coordinate of both 'chopping' and 'carrying'; but an instantiated 'hold' event, as in 'hold the axe', can not simultaneously be a coordinate of a 'chopping' and a 'carrying'. Either a 'chopping' event or a 'carrying' event is occurring, but not both; thus as FIND-PATH expands a path it can not traverse a coord arc and then subsequently traverse a coord* arc. Similar restrictions can be placed on the arc pairs sc/sc* and subseq/subseq*.

The temporal relations act as bridges between event/state identities. This type of restriction is motivated by observations like: an instantiated event/state concept cannot simultaneously be the consequent of two different concepts. For example, suppose FIND-

PATH has traversed a seq arc from chop1# to fall, it cannot subsequently traverse a consq* arc from 'fall' to 'drop'; to do so would be tantamount to inferring that the 'falling' could have occurred as an outcome of two disjoint events, 'chopping' and 'dropping'. Either a tree fell because it was chopped down, or it fell because it was dropped by a machine - but not because both events occurred.[4] Other restrictions of the bridge type include restrictions on the arc pairs consq/consq*, ante/ante*, and consq/seq*.

3.2.2 TRY-PATH

TRY-PATH traverses the path produced by FIND-PATH, testing the semantic constraints, performing reference resolution, and filling arguments. The input to TRY-PATH is a list of 4-tuples, which represent the path and a list of instantiated concepts. As previously described, the 4-tuples are of the form:

[relation event1 event2 ((matchfn1 arg1 arg2) ...)]

For each 4-tuple in the path TRY-PATH cycles through a two step

---

[4]If TRACE traverses a conseq arc from 'buy' to 'have', it cannot subsequently (immediately) traverse an consq* arc from 'have' to 'give'; to do so would be tantamount to inferring that the ''having' could have occurred as a consequent of two disjoint events, 'buying' and 'giving'. Either someone has a book because it was given to him, or he has a book because he purchased it - but not because both these events occurred.

process. For each constraint it checks that the indicated arguments match. If an argument is missing it uses the default value for that argument as a basis for matching. If all the constraints are met it replaces the old arguments attached to an event by the new argument values which were derived from the matching process. If one of the constraints fails TRY-PATH fails. To succeed TRY-PATH must traverse the path twice, once in each direction. This is necessary because TRY-PATH is propagating argument values across the path, so in the worst case arguments must be propagated from either end across the entire length of the path.

Each of the match functions has four parameters, two input and two output. The input parameters represent current values, the output parameters replacement values. Given the constraint (MATCH agt agt), if MATCH is called with the input parameters 'John' and 'he', MATCH will return the values 'John' and 'John'. Thus the new agent values for both events will be 'John'.

Suppose TRY-PATH is testing the coordinate relationship between instantiated 'chopping' and 'holding' events.

(coord chop1 hold1 ((Match agt agt) (Match instr obj)))

Furthermore, suppose that the list of instantiated concepts has the following values for chop1 and hold1:

(. . . (chop1 (agt peasant)) (hold1 (agt he)) . . .)

TRY-PATH begins by attempting to match the agents of chop1 and hold1. The agent value for chop1 is 'peasant', the agent value for hold1 is 'he'. Procedure MATCH successfully matches 'peasant' and 'he', returning the replacement value 'peasant' for both output parameters. Next the instrument of chop1 is matched against the object of hold1. Chop1 has no instrument value, but the default instrument value for chop2# is 'axe'. Hold1 has neither an object argument nor a default value for its object argument, so TRY-PATH uses the value nil. Again MATCH succeeds producing the value 'axe' for both output arguments. After replacing old values by new values, the instantiated concept list now looks like:

```
(. . . (chop1 (agt peasant)(instr axe))
       (hold1 (agt peasant)(obj axe)) . . .)
```

It is because the match function returns replacement values that TRY-PATH can fill missing arguments and perform reference resolution. In the above example, as TRY-PATH traverses the arc from chop1 to hold1 it resolves the 'he' reference. So FIND-PATH, in effect, orders the the possible pairings of referent and reference, and TRY-PATH determines if the resolution is correct.

Consider a path which TRY-PATH has rejected. Suppose TRACE is trying to connect the event concepts described in the following pair of sentences.

The pig cleaned the laundry at the stream.
She trotted home.

1. An antecedent of 'cleaning' is that the laundry must be 'moved' to the place where it is 'cleaned'.
2. A subclass of 'moving' is 'carrying'.
3. A coordinate of 'carrying' is 'travelling'.
4. A subsequence of 'travelling' is 'moving1#'.
5. A subclass of 'moving' is 'trotting'.

Figure 3-4: A Path That FIND-PATH Suggests

1. [ante clean move2# ((match loc dest) ...)]
2. [sc move2# carry ((match dest dest) ...)]
3. [coord carry travel ((match dest dest) ...)]
4. [coord travel move1# ((match dest dest) ...)]
5. [sc move1# trot ((match dest dest) ...)]

Figure 3-5: The Constraints Attached to Each Arc in the Path

Figure 3-4 shows a path that FIND-PATH suggests to TRY-PATH. TRY-PATH will eventually reject this path because the destination of the 'trotting' does not match the location of the 'cleaning'. To see how this works we need to look at the constraints that are attached to each arc along the path (see figure 3-5). Figure 3-5 only shows the relevant constraints. As TRY-PATH tests the path the location of the 'cleaning' will be propagated thru the concepts 'move2#', 'carry', 'travel', and 'move1#' until it is matched against the destination of the 'trotting'. The destination value of 'move1#' (stream), derived from the 'cleaning', will fail to MATCH the destination of the 'trotting' (home), consequently TRY-PATH rejects the path.

3.2.3 Some Rejected Heuristics

3.2.3.1. Match-Foci

MATCH-FOCI was intended to act as a coarse filter to the computation process. It was to be the first approximation to the likelihood that the new event/state description could be added to a schema. If the new event and the events in the schema share no argument values then MATCH-FOCI would reject the pairing of event and schema from further consideration.

The focus of an event description was defined to be the values associated with the case arguments attached to the instantiated concept. The focus of a schema was defined to be the

set of values associated with the case arguments attached to each instantiated concept in the schema. MATCH-FOCI intersected the two sets of values, using the match function MATCH to determine if the instantiated event/state concept shared a common focus with the schema.

Suppose MATCH-FOCI was given the following event and schema descriptions:

```
New instantiated event concept: (eat agt John obj apple)
Schema:   ((subseq (clean agt Mary obj clothes)
                   (wash agt Mary obj clothes))
           (subseq (clean agt Mary obj clothes)
                   (dry agt Mary obj clothes)))
```

The focus of the instantiated event concept includes the values: John and apple. The focus of the schema includes the values: Mary and clothes. The event and schema contain no common focus, so MATCH-FOCI would correctly inform TRACE that it should not try to find a path between the 'eating' event and the schema.

Suppose MATCH-FOCI was instead given the event and schema:

```
New instantiated event concept:
          (carry agt Mary obj clothes dest home)
Schema:   ((subseq (clean agt Mary obj clothes)
                   (wash agt Mary obj clothes))
           (subseq (clean agt Mary obj clothes)
                   (dry agt Mary obj clothes)))
```

The focus of the instantiated event concept and the schema share a focus, both Mary and clothes, so MATCH-FOCI would allow TRACE to try to find a path between the 'carrying' event and the schema.

One problem with MATCH-FOCI was that it turned out not to be too useful. MATCH-FOCI blocked only one potential pairing of schema and event. When MATCH-FOCI was removed TRACE was able to find a novel (good) interpretation for their connection (See the subsection on "The TALE of the Pig" for further details.). The other problem with MATCH-FOCI is that the matching it needs to perform can become as complicated as the TRACE's path finding/testing process. An example is the pair of sentences:

John went to the restaurant.
The waiter came to the table.

For MATCH-FOCI to determine that the waiter's 'coming to the table' could be related to John's 'trip to a restaurant' it must find a path either from waiter to restaurant or from table to restaurant. The space of concepts that MATCH-FOCI would need to use in order to find these connections would be at least as complicated as NEXUS' event/state concept space.

3.2.3.2. Temporal Orderings

Ideally FIND-PATH would not have to consider every incoming and outgoing arc during its expansion phase. Perhaps arc selection could be restricted on the basis of the temporal orderings of events in the story. FIND-PATH could expand forward from previous events and backwards from a new event. Paths on the source list would be expanded via prec, ante, consq*, and seq* relations, and paths on the goal list expanded via prec*, ante*, consq, and seq relations. Both

types of paths could be expanded via the taxonomic arcs. For the previously discussed pair of sentences:

> The pig cleaned the laundry at the stream.
> She trotted home.

FIND-PATH would never find the erroneous path from 'trotting' to 'cleaning' depicted in figure 3-4. For FIND-PATH to find this path it must either move temporally backwards from the 'cleaning' described in the first sentence, or forwards from the 'trotting' described in the second.

Unfortunately, such a restriction on arc selection also would cause TRY-PATH to miss some good paths. Take a simplified version of an example which occurred:

> The pig cleaned the clothes.
> She gathered the laundry.

In this case the connection between 'cleaning' and 'gathering' is:

1. After 'cleaning' the laundry the pig 'moves' it home.
2. To 'move' the laundry the pig must 'have' it.
3. A consequent of 'gathering' is 'having'.

But to find this connection FIND-PATH needs to move forward from 'gather' to 'move'; 'gathering' can only be related to 'cleaning' if its purpose, 'moving', which occurs after it, is found by TRY-PATH.

Other examples of this phenomenon occurred during the testing phase, forcing the eventual rejection of this heuristic.

3.3 Limitations

Woods [Woods 75] was the first to raise issue on the logical adequacy of semantic nets. He argued that there was not a theory of semantic nets (circa 1975), that even though the notion of semantic nets was an attractive one it was still an unproven assumption that they were adequate for representing knowledge in general.

Clearly NEXUS does not represent a theory of semantic nets at the epistemic level. Our interest has been to use semantic nets as a conceptual tool (abstraction), not to investigate their foundations. Our method has been to separate semantic nets, the abstract data type, from semantic nets, the concrete representation.

The theory presented in this dissertation is a theory of event/state coherence. It matters little to its concerns whether the net, as we think of it, is in fact represented in terms of epistemologically sounder networks such as NETL [Fahlman 77], or KLONE [Brachman 78, Brachman 79], or the partitioned networks of Hendrix [Hendrix 75, Hendrix 79]; we are perfectly happy to work with semantic nets at what Brachman [Brachman 79] calls the conceptual level. It is to our advantage that many of the complex issues dealt with in these other nets is hidden by an abstraction [Parnas 72].

The remainder of this section attempts to point out some of NEXUS' limitations when compared to systems whose major concern is the semantic net itself and not the usages of it.

First, current efforts in semantic nets are concerned with working on larger pieces of knowledge than single nodes. . Nodes are grouped together into subnets which are dealt with collectively, and are frequently referred to as structured nodes. Minsky's frame theory paper [Minsky 75] and Schank & Abelson's book on script and plan theory [Schank & Abelson 77] were the first to conceptualize this effort. Frames were a more general theory; the idea was to gather together and structure "'chunks' of reasoning, language, memory, and 'perception' (p211)." Scripts were only concerned with stereotyped situations. Neither theory was committed to semantic nets, but for those who were it meant that single nodes (concepts) had to be gathered together and dealt with collectively.

For example, Hendrix [Hendrix 79] uses three methods for structuring, partitioning, his network: spaces, vistas, and supernodes. Arbitrary groups of nodes and arcs can be bundled together into spaces, which are the fundamental units, along with individual nodes and arcs, of partitioned networks. Vistas represent pictures of the semantic net from various vantage points; they are composed of an arbitrary bundle of spaces, and can be used to reduce the size of the network for various interpreters. Supernodes are used to name a space, and therefore make the proposition denoted by the space available to other concepts in the network.

Nexus does not provide any general structuring techniques,

but there are two ways in which its net is structured. First, each node in the network represents a structured concept, it bundles together an event with its case related constituents (In a partitioned network each node in NEXUS would be a space.). Concepts in the network are also structured by the coherence relations; the 'chunk' travelling is composed of all the event/state concepts which are related to it by one of the seven coherence relations.

Another important issue in the literature on semantic networks is structured inheritance. One of the favorite targets has been the isa link. Brachman [Brachman 82] claims that it has been used to mean any number of things, sometimes by the same system. Typically isa links have been used to represent both inheritance between concepts, and instances of concepts. So, for example, an 'elephant isa animal' and a 'Clyde isa elephant'.

NEXUS uses different methods to represent inheritance between concepts and instances of concepts. The subclass relation is used for inheritance between concepts. The relationship between a concept and its instance is represented typographically; a concept is designated by its lexical name, and an instance of a concept by appending a number to the name of the concept it is an instance of. So 'carry' is the name of a concept, and 'carry1' is the name of an instance of a concept. To traverse an 'arc' from 'carry' to 'carry1' NEXUS strips the number off of 'carry1'. But again some issues have

been finessed. For example, NEXUS fails to differentiate between subclasses and conceptual instances; a subclass of 'travelling' is 'crossing', and a conceptual instance of 'travelling' is 'travelling by boat'. NEXUS uses a subclass arc for both types of relationships.

Another problem with NEXUS is that it uses the same set of arcs for representing relationships between concepts and relationships between instances of concepts. For example, to say that a coordinate of the concept 'carry' is the concept 'hold' is not the same thing as saying that a coordinate of an 'instance of carry' is an 'instance of hold'. Technically the usage of coherence relation coordinate between the instances of concepts means: these are instances of concepts which are related by the coherence relation coordinate. NEXUS leaves it to the interpreter to distinguish between the two meanings of the coherence relations.

Finally, TRACE's interpretation of the net is not much different from Quillian's [Quillian 68] original conception of a 'spreading activation' of concepts across a semantic network. It differs in two regards: 1) constraints are attached to each arc in the network, and 2) the cross-product of certain pairs of arcs are not permitted during TRACE's search phase. TRACE makes no attempt to control visibility of the net during its search phase. Context mechanisms for controlling the visibility of the net have been suggested by M.K. Smith [Smith 82] and Grosz [Grosz 77].

Chapter 4

Results

In this chapter we continue the examination, which began in chapter two, of NEXUS's representation scheme. Here we will show some sample protocols from TRACE in action. It is important to remember that for each of the examples described in this chapter TRACE uses the exact same dictionary of 150 (or so) event/state concepts.

Table 4-1 shows the distribution of coherence relations in the final output of each of the eight examples. The majority of the relations that TRACE used in its final output were taxonomic (subclass, subseq, coord). A lot of this has to do with property inheritance. The reader should recall that as the net was compiled the subclass relations were exploited to reduce the amount of redundant information, therefore it is not surprising that TRACE makes frequent use of taxonomic arcs. For example, the network contains the information that a precedent of 'cleaning' laundry is 'moving' the laundry to the place where it will be 'cleaned'. So if the text says, "The pig trotted to the stream. She cleaned her laundry.", TRACE must use a combination of taxonomic arcs in order to

get at the precedent relationship (i.e. 'trotting' is a subclass of 'moving', which is a 'subsequence' of 'travelling', which is a 'coordinate' of 'carrying', which is a subclass of 'moving2#', which is a precedent of 'cleaning').

The temporal relations are fairly evenly split between before and after categories. Sixteen of the relationships were either antecedent or precedent. Fourteen of them were either consequent or sequel.

Table 4-2 shows some statistics gathered during TRACE's processing of the samples. Columns two and three show the size of the search space for each example. So for "The Margie Story", during its search phase, TRACE encountered 48 different concepts, 63 if concepts which lay on more than one path are included in the count. Column four shows the number of paths that TRY-PATH rejected during its processing of each example, the next column shows the number of correctly found paths, and the last, the average length of the paths. The average path length is computed by dividing the number of relations in the representation produced by TRACE by the number of paths (i.e. #relationships/(#input-concepts - 1)).

"The Wishing Ring" is somewhat of an anomaly. Why was it necessary to reject so many paths? Most of the paths that TRY-PATH rejected centered around the concept 'having'. Part of the input for

story	subclass	subseq	coord	ante	conseq	prec	seq
peasant		2	1	1			1
wtell	2	3	3				1
margie	2		1	3	1		1
rob	5	3		1	1	2	1
rest	5	5		1		1	2
wish	2	2		2	2	1	
waterman	4	3	4	1	1		1
pig	7	4	3	2	1	1	1
total	27	22	12	11	6	5	8

Table 4-1: Distribution of Relations

peasant = "The Clever Peasant and the Czar's General"
wtell = "The Archer, William Tell"
margie = "The Margie Story"
rob = "Robbing a Liquor Store"
rest = "The Restaurant Story"
wish = "The Wishing Ring"
waterman= "The Waterman and the Peasant"
pig = "The Tale of the Pig"

story	# of concepts	# of concepts/dups	rejected	accepted	average path length
peasant	17	21	0	3	6/3 - 2
wtell	39	49	3	5	9/5 - 1.8
margie	48	63	2	5	11/5 - 2.2
rob	48	86	4	6	17/6 - 2.83
rest	46	109	8	7	15/7 - 2.14
wish	46	112	34	5	10/5 - 2
waterman	115	196	12	6	15/6 - 2.5
pig	88	219	6	9	23/9 - 2.35

Table 4-2: Search Space Size

this example includes a 'having'. As TRACE tried to add it to the schema it was constructing, multiple short 'having' paths were rejected; 'having' is related to a large number of concepts, and TRACE needed to sift through these in an attempt to find the correct path.

Table 4-3 compares longest, second longest, and average path lengths. The average path length does not vary much from 2. The longest path TRACE makes is a path of length 6 which occurs in "The Peasant and the Waterman".

story	longest-path	second-longest	average path length
peasant	4	1	6/3 - 2
wtell	3	2	9/5 -1.8
margie	3	2	11/5 - 2.2
rob	4	3	17/6 - 2.83
rest	4	3	15/7 - 2.14
wish	3	2	10/5 - 2
waterman	6	2	15/6 - 2.5
pig	4	4	23/9 - 2.35

Table 4-3: Path Lengths

Table 4-4 gives a list of the concepts which appeared in TRACE's output. This does not represent a complete list of concepts, some of the concepts that are in the network, which were derived during the analysis stage, were not needed for any of the examples that TRACE was tested on. But just same they were partially tested because they were included in the search space, and rejected, in part because of their constraints, by TRY-PATH if they appeared in an erroneous path.

word	pig	waterman	wish	rest	rob	Margie	archer	peasant
appear	1							
break						1		
burst						1		
carry	2					1		
catch						1		
chop		1						
clatter							1	
come	1							
cry						1		
depart				1				
descend		1						
dig								2
dive		1						
drop		1						
dry	1							
eat			1	1				
escape					1			
fall		1						
farm			1					
feed			1					
find		1						
follow		1					1	
gallop							1	
gather	1							
get					1			
get2#					1			
getaway					1			
give				2	1			
go				1				
grow			1					
hang	1							
have			3					
have2#	1		2		1	1		
hear							1	
hit						1		1
hold		1				1		
labor			1					
leave				1	1			
leave2#				1				
move1#	2			1	1		1	1
move2#	2					1		1
order				1				
perceive		1						

Table 4-4: Number of Times Concepts Appeared in TRACE's Output

word	pig	waterman	wish	rest	rob	margie	archer	peasant
plow			1					
push								1
rein							1	
rob					1			
ride							1	
say					1			
scour	1							
seat				1				
see	1	1						
seek		1						
sell			1					
serve				1				
soak	1							
sound							1	
spy	1							
stop							1	
swim		1						
take						1		
tell					1			
tip				1				
travel	2	1		1	1		1	
travel2#		1						
trot	2							
uncover								1
unhappy						1		
walk					1			
want					2			
wash	1							
watch	2							
total	24	24	12	16	15	12	10	7

Figure 4-4, continued

4.1 The Tale of the PIG

This is a more complicated version of the paragraph of text from The Tale of the Pig than the one we previously discussed. In this version of the paragraph many of the descriptions of the pig's activities are embedded in sentences describing the perceptions of a prince.

> One day while the pig trotted toward the stream, carrying a neat little bundle of clothes, she was spied by a prince. He watched while the animal expertly soaked and scoured the laundry. He watch the pig cleverly hang the clothes in the sun. (We will skip some text where the pig steps in the bubbling water, instantly changes into a beautiful young women, and recites a jingle that say she will be pig to a man weds her.) The pig gathered up the laundry and trotted home.

Figure 4-1 shows TRACE's processing of the paragraph. Beginning at line 1 the input to TRACE is shown - a list of case representations of the sentences of the text. The case representations have been simplified. In general the input representations do not include information like tense, mood, determiners, etc. TRACE can handle this information, but it really does not use it to aid processing. By paring the text hopefully increased comprehensibility has been achieved.

Note that events embedded in other events are extricated in their case representations. So the text says "The prince watched the pig cleverly hang the clothes in the sun.", and the case representations (lines 7 and 8) breaks the sentence into two units

```
((TRACE
1 ((TROT1 (AGT PIG) (TWD STREAM))     ***** INPUT *****
2  (CARRY1 (AGT PIG) (OBJ CLOTHES))
3  (SPY1 (PERCEPTOF PRINCE) (THM PIG))
4  (WATCH1 (PERCEPTOF HE) (THM (AND SOAK1 SCOUR1)))
5  (SOAK1 (AGT PIG) (OBJ LAUNDRY))
6  (SCOUR1 (AGT PIG) (OBJ LAUNDRY))
7  (WATCH2 (PERCEPTOF HE) (THM HANG1))
8  (HANG1 (OBJ CLOTHES) (IN SUN))
9  (GATHER1 (AGT PIG) (OBJ LAUNDRY))
10  (TROT2 (AGT PIG) (DEST HOME)))
11((((GATHER1 (AGT PIG)              ***** OUTPUT *****
              (OBJ LAUNDRY)
12            (CONSQ (HAVE2#96 (AGT PIG) (OBJ LAUNDRY))))
13   (APPEAR26 (PERCEPTOF PRINCE)
              (OBJ PIG)
              (LOC WATER)
14     (COORD (SEE23 (THM PIG)
                     (PERCEPTOF PRINCE)
                     (LOC WATER)
15                   (SC (SPY1 (THM PIG) (PERCEPTOF PRINCE)
                               (LOC WATER)))
16                   (CONTINUATION (WATCH1 (THM (AND SOAK1 SCOUR1))
                                           (PERCEPTOF PRINCE)
                                           (LOC WATER)))
17                   (CONTINUATION (WATCH2 (THM HANG1)
                                           (PERCEPTOF PRINCE)
                                           (LOC WATER)))))
18      (PREC (COME40 (AGT PIG) (DEST WATER))))
19   (CLEAN83
      (AGT PIG)
      (LOC WATER)
      (OBJ CLOTHES)
20    (ANTE (MOVE2#88
             (AGT PIG)
             (OBJ LAUNDRY)
             (DEST WATER)
21      (SC (CARRY1 (AGT PIG)
                    (OBJ LAUNDRY)
                    (DEST WATER)
```

Figure 4-1: The Tale of the Pig - 24 event/state concepts

```
22              (COORD (TRAVEL17 (TWD STREAM)
                                 (AGT PIG)
                                 (DEST WATER)
23                        (SUBSEQ (MOVE1#20
                                   (DEST WATER)
                                   (OBJ PIG)
                                   (TWD STREAM)
24                                 (SC (TROT1 (DEST WATER)
                                              (AGT PIG)
                                              (TWD STREAM)))))
25                       (SC (COME40 (AGT PIG)
                                     (DEST WATER)))))))))
26    (SEQ (MOVE2#103
            (AGT PIG)
            (OBJ LAUNDRY)
            (DEST HOME)
            (SOURCE WATER)
27          (SC (CARRY166 (AGT PIG)
                  (OBJ LAUNDRY)
                  (DEST HOME)
                  (SOURCE WATER)
28                (COORD (TRAVEL140 (AGT PIG)
                           (SOURCE WATER)
                           (DEST HOME)
29                         (SUBSEQ
                            (MOVE1#137 (OBJ PIG)
30                                     (SOURCE WATER)
                                       (DEST HOME)
                                       (SC (TROT2 (DEST HOME)
31                                                (SOURCE WATER)
                                                  (AGT PIG)))))
                                  ))))
32          (ANTE (HAVE2#96 (AGT PIG) (OBJ LAUNDRY)))))
33    (SUBSEQ (WASH80 (AGT PIG)
                      (OBJ LAUNDRY)
                      (LOC WATER)
34                    (SUBSEQ (SOAK1 (AGT PIG)
                                 (OBJ LAUNDRY) (LOC WATER)))
35                    (SUBSEQ (SCOUR1 (AGT PIG)
                                 (OBJ LAUNDRY) (LOC WATER)))))
```

Figure 4-1 continued

```
36    (SUBSEQ (DRY94 (AGT PIG)
                     (LOC WATER)
                     (OBJ CLOTHES)
37                   (SC (HANG1 (IN SUN)
                            (LOC WATER) (OBJ CLOTHES)))))))))))
```

Figure 4-1 concluded

connected by hangl, which is the theme of the 'watching' and the the name of the 'hanging' event that occurs.

There are several noteworthy aspects of TRACE's output. First, this example introduces the concept of continuation. It is not unusual to find instances of event/state concepts repeated in text. Usually the repetition of an event description both adds new information and conveys a sense of duration. TRACE handles repetitions by marking the relationship between the two descriptions as 'continuation'. In this example the concept of 'seeing' is repeated. The initial 'seeing' (see line 14), motivated by the connection between 'spying' (line 15) and appearing (line 13), is connected to the second 'seeing', a 'watching', at line 16 via a continuation arc. So the prince's 'spying' of the pig (line 15) continues as he 'watches' (line 16) the pig 'soak' and 'scour' the laundry. And again the 'seeing' is continued as the prince 'watches' (line 17) the pig 'hang' the clothes in the sun to 'dry'.

Some simple constraints are used by TRACE to control the construction of continuation bridges between concepts. So TRACE would not connect sentences like

The pig trotted to the stream.
The pig trotted home.

via continuation arcs since the destinations of the two events do not match.

FIND-PATH checks to see if a concept is a continuation of previous one before beginning its breadth first search through the dictionary network. Two descriptions are potentially in a continuation relationship if both concepts are in a subclass relationship to the same concept, or one concept is a subclass of the other, or they name the same event concept. If FIND-PATH fails to establish a continuation relationship it calls BREADTH to continue the search for connections between the new concept and previous concepts.

Another noteworthy aspect of this example is the connection that TRACE finds between the prince's 'spying' and the pig's 'trotting' and 'carrying'. Roughly the representation says:

1) The pig 'appeared' in the perception of the prince. (line 13)

2) For the pig to 'appear' she had to 'come' to the water. (line 18)

3) 'Coming' is a subclass of 'travelling'. (line 25)

4) 'Travelling' is a part of the 'carrying' event described in the text. (line 21)

5) For the pig to 'appear' someone had to 'see' her. (line 14)

6) A subclass of 'seeing' is 'spying'. (line 15)

TRACE's finding this connection was not expected. It was not included in the analysis of this paragraph. Originally the MATCH-FOCI mechanism prevented FIND-PATH from attempting to connect these concepts. The concept of 'appearance' was derived from some text in

another folktale. Originally, a focusing mechanism had prevented TRACE from trying to find coherence paths between the 'carrying' and 'spying'. When the focusing mechanism was removed TRACE produced the representation shown above.

Also of interested is the TRACE's handling of 'completed' events. When the 'soaking' (line 5) is added to the schema the 'trotting' (line 1) and 'carrying' (line 2) are marked as 'completed'. Event/state descriptions which follow the 'soaking' no longer include 'trot1' and 'carry1' and the event/states that connect them in the search space. Similarly when the 'gathering' (line 9) is added to the schema 'soaking' (line 5), 'scouring' (line 6), and 'hanging' (line 8), as well as all the other instantiated parts of the 'cleaning' (lines 19 and 33 thru 37) are marked as 'completed'. Before TRACE prints out its representation of the text it strips the 'completed' markings from the events which were concluded as the text progressed.

Finally, of interest is the fact that instances of the conceptual path from 'moving' to 'trotting' occur twice in the output: once when the pig moves the laundry to the stream (lines 20 thru 24), and a second time when she moves the laundry home (lines 26 thru 31).

4.2 The Peasant and the Waterman

This example comes from "The Peasant and the Waterman". In this chapter we have already discussed parts of it.

> A peasant was chopping a tree in the woods by the lake. He dropped his axe and it fell with a splash into the water. Quickly he dove into the lake, hoping to find his precious axe, his only axe. But no matter how many times he swam to the bottom of the icy lake, no axe did he see.

Figure 4-2 shows TRACE's input (lines 1 thru 9) and output (lines 8 thru 25). There are a number of interesting aspects to TRACE's processing of this text. The first is that it handles the dropping without any difficulty.

1) When the peasant 'chopped' wood he 'held' the axe. (line 9)

2) In order for the peasant to 'drop' the axe he must first 'hold' it. (line 11)

In terms of scripts, Schank and Abelson [Schank & Abelson 77] refer to this kind of problem as an obstacle; an enabling condition for 'chopping' (i.e. 'holding the axe') is missing when the peasant 'drops' the axe. TRACE handles obstacles in the same manner that it handles any other case. Because the text describes a 'chopping', 'holding' is implicitly in the text, which has a relationship with 'dropping'; there is an implicit relationship between the concepts 'chopping' and 'dropping' and when they appear in the text TRACE finds the connection. To TRACE there is no difference between the methods it uses for connecting 'chopping' and 'dropping' and the ones it uses to connect, for example, 'washing' and 'drying'. For TRACE