A Logic for Semantic Interpretation[1]

Eugene Charniak and **Robert Goldman**
Department of Computer Science
Brown University, Box 1910
Providence RI 02912

Abstract

We propose that logic (enhanced to encode probability information) is a good way of characterizing semantic interpretation. In support of this we give a fragment of an axiomatization for word-sense disambiguation, noun-phrase (and verb) reference, and case disambiguation. We describe an inference engine (Frail3) which actually takes this axiomatization and uses it to drive the semantic interpretation process. We claim three benefits from this scheme. First, the interface between semantic interpretation and pragmatics has always been problematic, since all of the above tasks in general require pragmatic inference. Now the interface is trivial, since both semantic interpretation and pragmatics use the same vocabulary and inference engine. The second benefit, related to the first, is that semantic guidance of syntax is a side effect of the interpretation. The third benefit is the elegance of the semantic interpretation theory. A few simple rules capture a remarkable diversity of semantic phenomena.

I. Introduction

The use of logic to codify natural language syntax is well known, and many current systems can parse directly off their axiomatizations (e.g.,)[1]. Many of these systems simultaneously construct an intermediate "logical form" using the same machinery. At the other end of language processing, logic is a well-known tool for expressing the pragmatic information needed for plan recognition and speech act recognition [2–4]. In between these extremes logic appears much less. There has been some movement in the direction of placing semantic interpretation on a more logical footing [5,6], but it is nothing like what has happened at the extremes of the language understanding process.

To some degree this is understandable. These "middle" parts, such as word-sense disambiguation, noun phrase reference, case disambiguation, etc. are notoriously difficult, and poorly understood, at least compared to things like syntax, and the construction of intermediate logical form. Much of the reason these areas are so dark is that they are intimately bound up with pragmatic reasoning. The correct sense of a word depends on context, as does pronoun resolution, etc.

Here we rectify this situation by presenting an axiomatization of fragment of semantic interpretation, notably including many aspects previously excluded: word-sense disambiguation, noun-phrase reference determination, case determination, and syntactic disambiguation. Furthermore we describe an inference engine, Frail3, which can use the logical formulation to carry out semantic interpretation. The description of Frail3 is brief, since the present paper is primarily concerned with semantic interpretation. For a more detailed description, see [7]. The work closest to what we present is that by Hobbs [5]; however, he handles only noun-phrase reference from the above list, and he does not consider intersentential influences at all.

Our system, Wimp2 (which uses Frail3), is quite pretty in two respects. First, it integrates semantic and pragmatic processing into a uniform whole, all done in the logic. Secondly, it provides an elegant and concise way to specify exactly what has to be done by a semantic interpreter. As we shall see, a system that is roughly comparable to other state-of-the-art semantic interpretation systems [6,8] can be written down in a page or so of logical rules.

Wimp2 has been implemented and works on all of the examples in this paper.

II. Vocabularies

Let us start by giving an informal semantics for the special predicates and terms used by the system. Since we are doing semantic interpretation, we are translating between a syntactic tree on one hand and the logical, or internal, representation on the other. Thus we distinguish three vocabularies: one for trees, one for the internal representation, and one to aid in the translation between the two.

The vocabulary for syntactic trees assumes that each word in the sentence is represented as a word instance which is represented as a word with a numerical postfix (e.g., boy22). A word instance is associated with the actual lexical entry by the predicate word-inst:

[1] This work has been supported in part by the National Science Foundation under grants IST 8416034 and IST 8515005 and Office of Naval Research under grant N00014-79-C-0529.

(word-inst *word-instance part-of-speech lexical-item*).

For example, (word-inst case26 noun case). (We use "part of speech" to denote those syntactic categories that are directly above the terminal symbols in the grammars, that is, directly above words.)

The relations between word instances are encoded with two predicates: syn-pos, and syn-pp. Syn-pos

(syn-pos *relation head sub-constituent*),

indicates that the ***sub-constituent*** is the ***relation*** of the ***head***. We distinguish between positional relations and those indicated by prepositional phrases, which use the predicate syn-pp, but otherwise look the same. The propositions denoting syntactic relations are generated during the parse. The parser follows all possible parses in a breadth-first search and outputs propositions on a word-by-word basis. If there is more than one parse and they disagree on the propositional output, a disjunction of the outputs is asserted into the database. The correspondence between trees and formulas is as follows:

Trees	Formulas
s — np (vp ... head-v ...)	(syn-pos subject ***head-v np***) head-v symbol is s symbol
vp → ... head-v np ...	(syn-pos object ***head-v np***)
vp → ... head-v np1 np2 ...	(syn-pos indirect-object ***head-v np1***) (syn-pos object ***head-v np2***)
vp → ... head-v ... (pp prep ...)	(syn-pp head-prep ***head-v prep***)
pp — prep np	(syn-pp prep-np ***prep np***)
np → ... head-n ...	head-n symbol is np symbol
np → pronoun	pronoun symbol is np symbol
np — propernoun	propernoun symbol is np symbol
np → ... adj head-n ...	(syn-pos adj ***adj head-n***)
np → ... head-n ... (pp prep ...)	(syn-pp head-prep ***head-n prep***)
np — that s	s symbol is np symbol
s — np (vp ... copula (pp prep ...))	(syn-pp head-prep ***np prep***)
s — np (vp ... copula adj)	(syn-pos adj ***adj np***)

This is enough to express a wide variety of simple declarative sentences. Furthermore, since our current parser implements a transformational account of imperatives, questions (both yes-no and wh), complement constructions, and subordinate clauses, these are automatically handled by the above as well. For example, given an account of "Jack wants to borrow the book." as derived from "Jack wants (np that (s Jack borrow the book))." or something similar, then the above rules would produce the following for both (we also indicate after what word the formula is produced):

Words	Formulas
Jack	(word-inst jack1 propernoun jack)
wants	(word-inst want1 verb want) (syn-pos subject want1 jack1)
to	
borrow	(word-inst borrow1 verb borrow) (syn-pos object want1 borrow1) (syn-pos subject borrow1 jack1)
the	
book	(word-inst book1 noun book) (syn-pos object borrow1 book1)

This is, of course, a fragment, and most things are not handled by this analysis: negation, noun-noun combinations, particles, auxiliary verbs, etc.

Now let us consider the internal representation used for inference about the world. Here we use a simple predicate-calculus version of frames and slots. We assume only two predicates for this: == and inst. Inst,

(inst *instance frame*),

is a two-place predicate on an instance of a frame and the frame itself, where a "frame" is a set of objects, all of which are of the same natural kind. Thus (inst boy1 boy-) asserts that boy1 is a member of the set of boys, denoted by boy-. (Frames are symbols containing hyphens, e.g., supermarket-shoping. Where a single English word is sufficiently descriptive, the hyphen is put at the end.)

The other predicate used to describe the world is the "better name" relation ==:

(== *worse-name better-name*).

This is a restricted use of equality. The second argument is a "better name" for the first, and thus may be freely substituted for it (but not the reverse). Since slots are represented as functions, == is used to fill slots in frames. To fill the agent slot of a particular action, say borrow1, with a particular person, say jack1, we say

(== (agent borrow1) jack1).

At an implementation level, == causes everything known about its first argument (the worse name) to be asserted about the second (the better name). This has the effect of concentrating all knowledge about all of an object's names as facts about the best name.

Frail will take as input a simple frame representation and translate it into predicate-calculus form. Figure 1 shows a frame for shopping along with the predicate-calculus translation.

Naturally, a realistic world model requires more than these two predicates plus slot functions, but the relative success of fairly simple frame models of reasoning indicates that they are a good starting set. The last set of predicates (word-sense, case, and role-inst) are used in the translation itself. They will be defined later.

```
(defframe shop-
   isa     action
           ;(inst ?s.shop- action)
   slots   (agent (person-))
           ;(inst (agent ?s.shop-) person-)
           (store-of (store-))
           ;(inst (store-of ?s.shop-) store-)
   acts    (go-step
           (go- (agent (agent ?shop-))
                (destination (store-of ?shop-))))
           ;(== (agent (go-step ?shop-)) (agent ?shop-))
           ;(== (destination (go-step ?s.shop-))
           ;     (store-of ?s.shop-))
```

Figure 1: A frame for shopping

III. Word-Sense Disambiguation

We can now write down some semantic interpretation rules. Let us assume that all words in English have one or more word senses as their meaning, that these word senses correspond to frames, and that any particular word instance has as its meaning exactly one of these senses. We can express this fact for the instances of any particular lexical entry as follows:

(word-inst *inst part-of-speech word*) ⇒
(inst *inst sense*$_1$) ∨ ... ∨ (inst *inst sense*$_n$)

where *sense*$_1$ through *sense*$_n$ are senses of *word* when it is used as a *part-of-speech* (i.e., as a noun, verb, etc.)

Not all words in English have meanings in this sense. "The" is an obvious example. Rather than complicate the above rules, we assign such words a "null" meaning, which we represent by the term **garbage***. Nothing is known about **garbage*** so this has no consequences. A better axiomatization would also include words which seem to correspond to functions (e.g., **age**), but we ignore such complications.

A minor problem with the above rule is that it requires us to be able to say at the outset (i.e., when we load the program) what all the word senses are, and new senses cannot be added in a modular fashion. To fix this we introduce a new predicate, word-sense:

(word-sense *lex-item part-of-speech frame*)
(word-sense straw noun drink-straw)
(word-sense straw noun animal-straw).

This states that *lex-item* when used as a *part-of-speech* can mean *frame*.

We also introduce a pragmatically different form of disjunction, →OR:

(→OR *formula*$_1$ *formula*$_2$).

In terms of implementation, think of this as inferring *formula*$_1$ in all possible ways and then asserting the disjunction of the *formula*$_2$s with each set of bindings. So if there are two sets of bindings, the result will be to assert

(OR *formula*$_2$/*bindings*$_1$ *formula*$_2$/*bindings*$_2$).

Logically, the meaning of →OR is that if x_1 ... x_n are unbound variables in *formula*$_1$, then there must exist x_1 ... x_n that make *formula*$_1$ and *formula*$_2$ true.

We can now express our rule of word-sense ambiguity as:

```
(word-inst ?instance ?part-of-speech ?lex-item) ⇒
   (→OR (word-sense ?lex-item ?part-of-speech ?frame)
         (inst ?instance ?frame))
```

IV. The Inference Engine

While it seems clear that the above rule expresses a rather simple-minded idea of how words relate to their meanings, its computational import may not be so clear. Thus we now discuss Wimp2, our language comprehension program, and its inference engine, Frail3.

Like most rule-based systems, Frail distinguishes forward and backward-chaining use of modus-ponens. All of our semantic interpretation rules are forward-chaining rules:

```
(→ (word-inst ?instance ?part-of-speech ?lex-item)
   (→OR (word-sense ?lex-item ?part-of-speech ?frame)
         (inst ?instance ?frame)))
```

Thus, whenever a new word instance is asserted, we forward-chain to a statement that the word denotes an instance of one of a set of frames.

Next, Frail uses an ATMS [9,10] to keep track of disjunctions. That is, when we assert (OR *formula*$_1$ ...*formula*$_n$) we create *n* assumptions (following DeKleer, these are simply integers) and assert each formula into the data-base, each with a *label* indicating that the formula is not true but only true given some assumptions. Here is an example of how some simple disjunctions come out.

```
A   (→ A (OR B C))
    (→ B (OR D E))
```

Formulas	Assumptions	Labels
A		(())
B	1	((1))
C	2	((2))
D	3	((1 3))
E	4	((1 4))

Figure 2 represents this pictorially. Here D, for example, has the label ((13)), which means that it is true if we grant assumptions 1 and 3. If an assumption (or more generally, a set of assumptions) leads to a contradiction, the assumption is declared a "nogood" and formulas which depend on it are no longer believed. Thus if we learn (not D) then (1 3) is a nogood. This also has the consequence that E now has the label (1). It is as if different sets of assumptions correspond to different worlds. Semantic interpretation then is finding the "best" of the worlds defined by the linguistic possibilities.

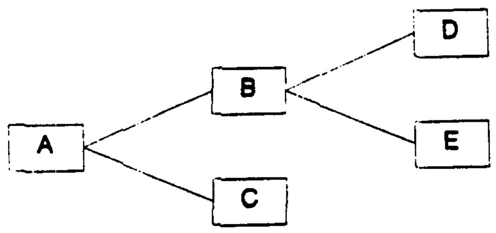

Figure 2: Pictorial representation of disjunctions

We said "best" in the last sentence deliberately. When alternatives can be ruled out on logical grounds the corresponding assumptions become nogoods, and conclusions from them go away. But it is rare that *all* of the candidate interpretations (of words, of referents, etc.) reduce to only one that is logically possible. Rather, there are usually several which are logically consistent, but some are more "probable" than others. For this reason, Frail associates probabilities with sets of assumptions ("alternative worlds") and Wimp eventually "garbage collects" statements which remain low-probability alternatives because their assumptions are unlikely. Probabilities also guide which interpretation to explore. Exactly how this works is described in [7]. Here we will simply note that the probabilities are designed to capture the following intuitions:

1. Uncommon vs. common word-senses (marked vs. unmarked) are indicated by probabilities input by the system designer and stored in the lexicon.
2. Wimp prefers to find referents for entities (rather than not finding referents).
3. Possible reasons for actions and entities are preferred the more specific they are to the action or entity. (E.g., "shopping" is given a higher probability than "meeting someone" as an explanation for going to the supermarket.)
4. Formulas derived in two differents ways are more probable than they would have been if derived in either way alone.
5. Disjunctions which lead to already considered "worlds" are preferred over those which do not hook up in this way. (We will illustrate this later.)

V. Case Disambiguation

Cases are indicated by positional relations (e.g., **subject**) and prepositional phrases. We make the simplifying assumption that prepositional phrases only indicate case relations. As we did for word-sense disambiguation, we introduce a new predicate that allows us to incrementally specify how a particular head (a noun or verb) relates to its syntactic roles. The new predicate,

(case *head syntactic-relation slot*),

states that ***head*** can have its *slot* filled by things which stand in ***syntactic-relation*** to it. For example

(inst ?g go-) ⇒ (case ?g subject agent).

This can also be expressed in Frail using the typed variables

(case ?g.go- subject agent).

This says that any instance of a **go-** can use the subject position to indicate the agent of the **go-** event. These facts can be inherited in the typical way via the isa hierarchy, so this fact would more generally be expressed as

(case ?a.action- subject agent).

Using **case** and the previously introduced **→OR** connective, we can express the rule of case relations. Formally, it says that for all syntactic positional relations and all meanings of the head, there must exist a case relation which is the significance of that syntactic position:

```
(syn-pos ?rel ?head ?val) ∧ (inst ?head ?frame) ⇒
        (→OR (case ?head ?rel ?slot)
                (== (?slot ?head) ?val))).
```

So, we might have

```
(syn-pos subject go1 jack1) ∧ (inst go1 go-)
        ∧ (case go1 subject agent)
        ⇒ (== (agent go1) jack1).
```

A similar rule holds for case relations indicated by prepositional phrases.

```
(syn-pp head-prep ?head ?pinst)
    ∧ (syn-pp prep-np ?pinst ?np)
    ∧ (word-inst ?pinst prep ?prep) ∧ (inst ?head ?frame)
        ⇒ (→OR (case ?head ?prep ?slot)
                (== (?slot ?head) ?np))
```

For example, "Jack went to the supermarket." would give us

```
(syn-pp head-prep go1 to1) ∧ (case go1 to destination)
    ∧ (syn-pp prep-np to1 supermarket1)
    ∧ (word-inst to1 prep to) ∧ (inst go1 go-)
        ⇒ (== (destination go1) supermarket1).
```

We now have enough machinery to describe two ways in which word senses and case relations can help disambiguate each other. First consider the sentence

Jack went to the supermarket.

Wimp currently knows two meanings of "go," to travel and to die. After "Jack went" Wimp prefers travel (based upon probability rule 1 and the probabilities assigned to these two readings in the lexicon) but both are possible. After "Jack went to" the die reading goes away. This is because the only formulas satisfying

(case go1 to ?slot)

all require **go1** to be a travel rather than a die. Thus "die" cannot be a reading since it makes

```
(→OR (case ?head ?prep ?slot)
        (== (?slot ?head) ?val))
```

false (a disjunction of zero disjuncts is *false*).

We also have enough machinery to see how "selectional restrictions" work in Wimp2. Consider the sentence

Jack fell at the store.

and suppose that Wimp knows two case relations for "at," **loc** and **time**. This will initially lead to the following disjunction:

```
                                   ((1))
                                  (== (loc fell1) store1)
(syn-pp head-prep fell1 at1)<
                                  ((2))
                                   (== (time fell1) store1).
```

However, Wimp will know that

(inst (time ?a.action) time-).

As we mentioned earlier, == statements cause everything known about the first argument to be asserted about the second. Thus Wimp will try to believe that **store1** is a time, so (2) becomes a nogood and (1) becomes just *true*.

It is important to note that both of these disambiguation methods fall out from the basics of the system. Nothing had to be added.

VI. Reference and Explanation

Definite noun phrases (np's) typically refer to something already mentioned. Occasionally they do not, however, and some, like proper names may or may not refer to an already mentioned entity. Let us simplify by saying that all np's may or may not refer to something already mentioned. (We will return to indefinite np's later.) We represent np's by always creating a new instance which represents the entity denoted by the np. Should there be a referent we assert equality between the newly minted object and the previously mentioned one. Thus, in "Jack went to the supermarket. He found some milk on the shelf.", the recognition that "He" refers to Jack would be indicated by

(== **he24 jack3**).

(Remember that == is a best name relation, so this says that **jack3** is a better name for the new instance we created to represent the "he," **he24**.)

As for representing the basic rule of reference, the idea is to see the call for a referent as a statement that something exists. Thus we might try to say

(inst ?x ?frame) ⇒ (Exists (y \ ?frame) (== ?x ?y)).

This is intended to say, if we are told of an object of type ?frame then there must exist an earlier one y of this same type to which the new one can be set equal.

The trouble with this formula is that it does not say "earlier one." Exists simply says that there has to be one, whether or not it was mentioned. Furthermore, since we intend to represent an np like "the taxi" by (inst taxi27 taxi-) and then look for an earlier taxi, the Exists would be trivially satisfied by taxi27 itself.

Our solution is to introduce a new quantifier called "previously exists" or PExists. (In [5] a similar end is achieved by putting weights on formula and looking for a minimum-weight proof.) Using this new quantifier, we have

(inst ?x ?frame) ⇒ (PExists (y \ ?frame) (== ?x ?y)).

If there is more than one a disjunction of equality statements is created. For example, consider the story

Jack went to the supermarket. He found the milk on the shelf. He paid for it.

The "it" in the last sentence could refer to any of the three inanimate objects mentioned, so initially the following disjunction is created:

```
                          (== it8 shelf6)
(inst it8 inanimate-) <   (== it8 milk5)
                          (== it8 supermarket2).
```

This still does not allow for the case when there is no referent for the np. To understand our solution to this problem it is necessary to note that we originally set out to create a plan-recognition system. That is to say, we wanted a program which given a sentence like "Jack got a rope. He wanted to kill himself." would recognize that Jack plans to hang himself. We discuss this aspect of Wimp2 in greater detail in [7]. Here we simply note that plans in Wimp2 are represented as frames (as shown in Figure 1.) and that sub tasks of plans are actions which fill certain slots of the frame. So the **shop-** plan has a **go-step** in Figure 1. and recognizing the import of "Jack went to the supermarket." would be to infer that (== (**go-step shop-74**) **go61**) where **go61** represented the verb in "Jack went to the supermarket." We generalize this slightly and say that all inputs must be "explained"; by this we mean that we must find (or postulate) a frame in which the input fills a slot. Thus the **go-step** statement explains **go61**. The presence of a supermarket in the story would be explained by (== (**store-of shop-74**) **supermarket64**). The rule that everything mentioned must be explained looks like this:

```
(inst?x ?frame) ⇒
    (→OR (role-inst ?x ?slot ?superfrm)
            (Exists (y \ ?superfrm) (== (?slot ?y) ?x))).
```

(Some things cannot be explained, so this rule is not strict.) Here the **role-inst** predicate says that ?x can fill the ?slot role of the frame ?superfrm. E.g., (**role-inst ?r.store- store-of shop-**) says that stores can fill the **store-of** slot in the **shop-** frame. Here we use Exists, not PExists since, as in the rope example, we explained the existence of the rope by postulating a *new* hanging event. The semantics of Exists is therefore quite standard, simply saying that one must exist, and making no commitment to whether it was mentioned earlier or not. As a matter of implementation, we note that it works simply by *always* creating a new instance. The impact of this will be seen in a moment.

We said that all inputs must be explained, and that we explain by seeing that the entity fills a slot in a postulated frame. There is one exception to this. If a newly mentioned entity refers to an already extant one, then there is no need to explain it, since it was presumably explained the first time it was seen. Thus we combine our rule of reference with our rule of explanation. Or, to put it slightly differently, we handle the exceptions to the rule of reference (some things do not refer to entities already present) by saying that those which do not so refer must be explained instead. This gives the following rule:

```
(inst ?x ?frame) ∧ (not (= ?frame garbage*)) ⇒
      (OR (PExists (y \ ?frame) (== ?x ?y)) .9
          (—OR (role-inst ?x ?superfrm ?slot)
                (Exists (s \ ?superfrm)
                        (== (?slot ?s) ?x))) .1)
```

Here we added the restriction that the frame in question cannot be the **garbage*** frame, which has no properties by definition. We have also added probabilities to the disjunctions that are intended to capture the preference for previously existing objects (probability rule 2). The rule of reference has several nice properties. First, it might seem odd that our rule for explaining things is expressed in terms of the Exists quantifier, which we said always creates a new instance. What about a case like "Jack went to the supermarket. He found the milk on the shelf." where we want to explain the second line in terms of the shopping plan created in the first? As we have things set up, it simply creates a new shopping plan. But note what then occurs. First the system asserts (inst **new-shopping5** **shopping-**). This activates the above rule, which must either find a referent for it, or try to explain it in terms of a frame for which it fills a role. In this case there is a referent, namely the shopping created in the course of the first line. Thus we get (== **new-shopping5 shopping4**) and we have the desired outcome. This example also shows that the reference rule works on event reference, not just np reference.

This rule handles reference to "related objects" rather well. Consider "Jack wanted to play the stereo. He pushed the on-off button." Here "the on-off button" is to be understood as the button "related" to the stereo mentioned in the first line. In Wimp this falls out from the rules already described. Upon seeing "the on-off button" Wimp creates a new entity which must then either have a referent or an explanation. It does not have the first, but one good explanation for the presence of an on-off button is that it fills the **on-off-switch** slot for some **power-machine**. Thus Wimp creates a machine and the machine then has to be explained. In this case a referent is found, the stereo from the first sentence.

VII. Pragmatic Influence

We finish with three examples illustrating how our semantic interpretation process easily integrates pragmatic influences: one example of pronoun reference, one of word-sense disambiguation, and one of syntactic ambiguity. First pronoun reference:

> Jack went to the supermarket. He found the milk on the shelf. He paid for it.

In this example the "milk" of sentence two is seen as the **purchased** of **shop-1** and the "pay" of sentence three is postulated to be the **pay-step** of a shopping event, and then further postulated to be the same shopping event as that created earlier. (In each case other possibilities will be considered, but their probabilities will be much lower.) Thus when "it" is seen Wimp is in the situation shown in Figure 3. The important thing here is that the statement (== it7 milk5) can be derived in two different ways, and thus its probability is much higher than the other possible referents for "it" (probability rule 4). (One derivation has it that since one pays for what one is shopping for, and Jack is shopping for milk, he must be paying for the milk. The other derivation is that "it" must refer to something, and the milk is one alternative.)

The second example is one of word-sense disambiguation:

> Jack ordered a soda. He picked up the straw.

Here sentence one is seens as the **order-step** of a newly postulated **eat-out1**. The soda suggests a drinking event, which in turn can be explained as the **eat-step** of **eat-out1**. The straw in line two can be one of two kinds of straw, but the **drink-straw** interpretation suggests (via a **role-inst** statement) a **straw-drinking** event. This is postulated, and Wimp looks for a previous such event (using the normal reference rule) and finds the one suggested by the soda. Wimp prefers to assume that the drinking event suggested by "soda" and that from "straw" are the same event (probability rule 2) and this preference is passed back to become a preference for the **drink-straw** meaning of "straw" (by probability rule 5). The result is shown in Figure 4.

Our third and last example shows how semantic guidance of syntax works:

> Janet wanted to kill the boy with some poison.

Starting with the "with" there are two parses which disagree on the attachment of the prepositional phrase (pp). There are also two case relations the "with" can indicate if it modifies "kill," **instrument** and **accompaniment**. When Wimp sees "poison" it looks for an explanation of its presence, postulates a poisoning and which is found to be potentially coreferential with the "kill." The result looks like Figure 5. In this interpretation the poison can be inferred to be the instrument of the poisoning, so this option has higher probability (probability rule 4). This

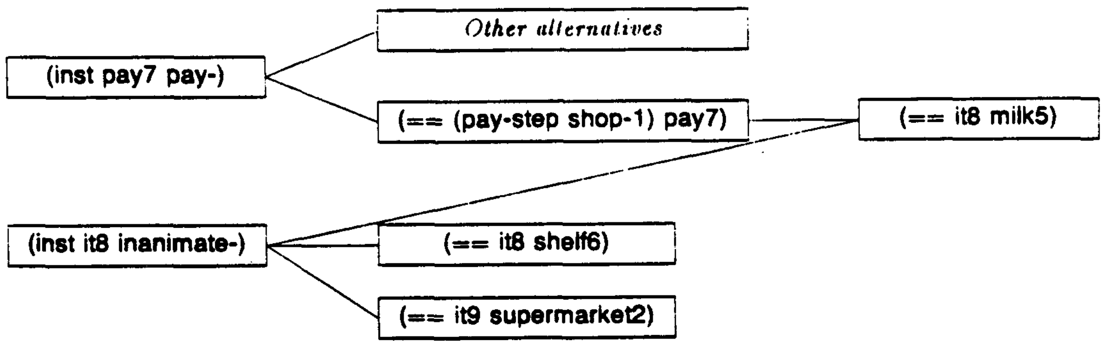

Figure 3: A pronoun example

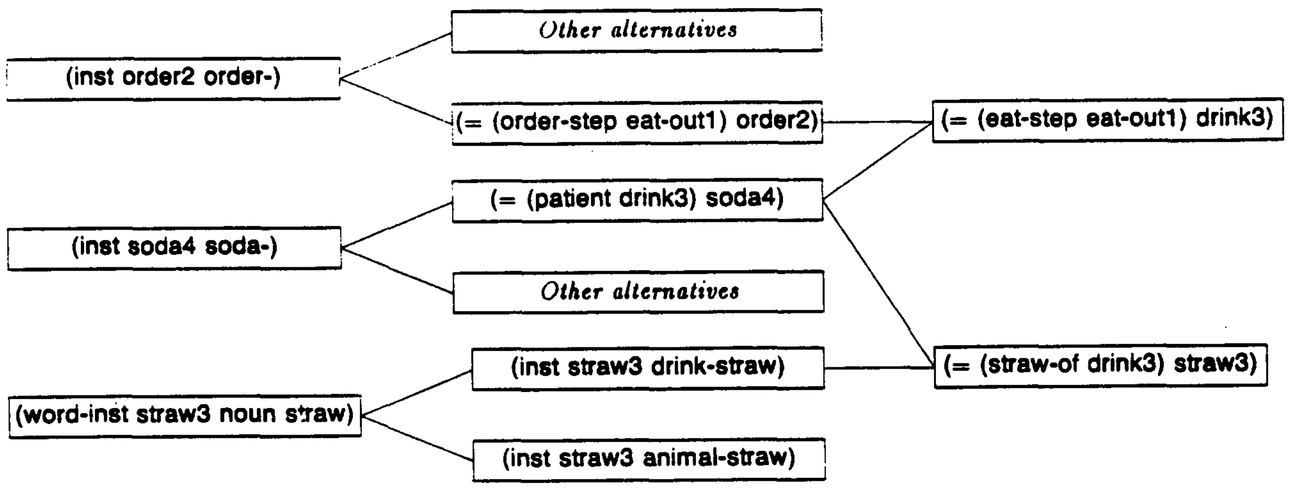
(inst order2 order-)
Other alternatives
(= (order-step eat-out1) order2)
(= (eat-step eat-out1) drink3)
(= (patient drink3) soda4)
(inst soda4 soda-)
Other alternatives
(inst straw3 drink-straw)
(= (straw-of drink3) straw3)
(word-inst straw3 noun straw)
(inst straw3 animal-straw)

Figure 4: A word-sense example

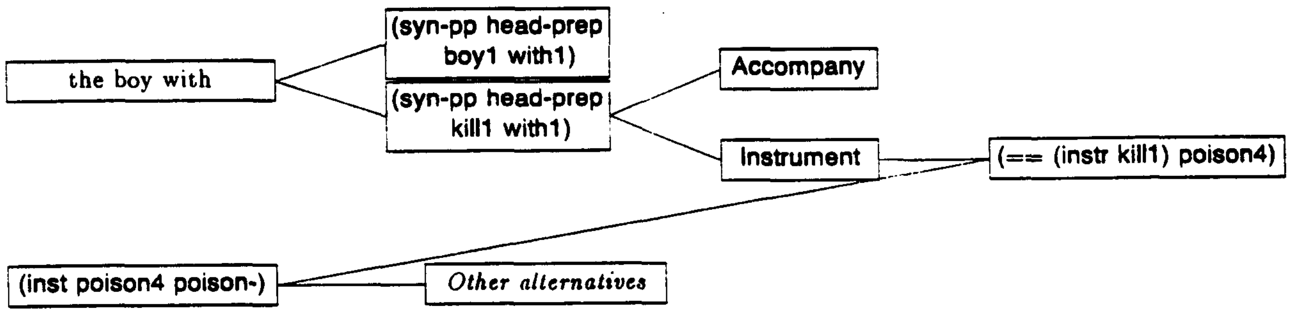
the boy with
(syn-pp head-prep boy1 with1)
(syn-pp head-prep kill1 with1)
Accompany
Instrument
(== (instr kill1) poison4)
(inst poison4 poison-)
Other alternatives

Figure 5: A syntactic disambiguation example

higher probability is passed back to the disjuncts representing a) the choice of instrument over accompanyment, and b) the choice of attaching to "kill" over "boy" (probability rule 5). This last has the effect of telling the parser where to attach the pp.

VIII. Future Research

This work can be extended in many ways: increased syntactic coverage, more realistic semantic rules, improved search techniques for possible explanations, etc. Here we will simply look at some fairly straightforward extensions to the model.

Our rule preferring finding a referent to not finding a referent is not reasonable for indefinite np's. Thus Wimp currently misinterprets

Jack bought a gun. Mary bought a gun.

since it wants to interpret the second gun as coreferential with the first. A simple change would be to have two rules of reference/explanation. The rule for indefinite np's would look like this:

```
(inst ?x ?frame) ∧ (not (= ?frame garbage*))
   ∧ (syn-pos indef-det ?x ?det)
        ⇒ (OR (PExists (y \ ?frame) (== ?x ?y)) .1
              (→OR (role-inst ?x ?superfrm ?slot)
                   (Exists (s \ ?superfrm)
                           (== (?slot ?s) ?x))) .9)
```

This looks just like our earlier rule, except a check for an indefinite determiner is added, and the probabilities are reversed so as to prefer a new object over an already existing one. The earlier reference rule would then be modified to make sure that the object did not have an indefinite determiner.

Another aspect of language which fits rather nicely into this framework is metonymy. We have already noted that the work closest to ours is [5], and in fact we can adopt the analysis presented there without a wrinkle. This analysis assumes that every np corresponds to two objects in the story, the one mentioned and the one intended. For example:

I read Proust over summer vacation.

The two objects are the entity literally described by the np (here the person "Proust") and that intended by the speaker (here a set of books by Proust). The syntactic analysis would be modified to produce the two objects, here proust1 and read-obj1 respectively.

```
(syn-pos direct-object read1 read-obj1)
(word-inst proust1 propernoun proust)
(syn-pos metonymy read-obj1 proust1)
```

It is then assumed that there are a finite number of relations that may hold between these two entities, most notably equality, but others as well. The rule relating the two entities would look like this:

```
(→ (syn-pos metonymy ?intended ?given)
    (OR (== ?intended ?given) .9
        (== (creator-of ?intended) ?given) .02)
        ...)).
```

This rule would prefer assuming that the two individuals are the same, but would allow other possibilities.

IX. Conclusion

We have presented logical rules for a fragment of the semantic interpretation (and plan recognition) process. The four simple rules we gave already capture a wide variety of semantic and pragmatic phenomena. We are currently working on diverse aspects of semantics, such as definite vs. indefinite np's, noun-noun combinations, adjectives, non-case uses of prepositions, metonymy and relative clauses.

References

[1] F. Pereira & D. Warren, "Definite clause grammar for language analysis – a survey of the formalism and a comparison with augmented transition networks," *Artificial Intelligence* 13 (1980), 231–278.

[2] Philip R. Cohen & C. Raymond Perrault, "Elements of a plan-based theory of speech acts," *Cognitive Science* 3 (1979), 177–212.

[3] Eugene Charniak, "A neat theory of marker passing," *AAAI-86* (1986).

[4] Henry Kautz & James Allen, "Generalized plan recognition," *AAAI-86* (1986).

[5] Jerry R. Hobbs & Paul Martin, "Local pragmatics," *Ijcai-87* (1987).

[6] Graeme Hirst, *Semantic Interpretation and the Resolution of Ambiguity*, Cambridge University Press, Cambridge, 1987.

[7] Robert Goldman & Eugene Charniak, "A probabilistic ATMS for plan recognition," forthcomming.

[8] Barbara J. Grosz, Douglas E. Appelt, Paul A. Martin & Fernando C.N. Pereira, "Team: an experiment in the design of transportable natural-language interfaces," *Artificial Intelligence* 32 (1987), 173–243.

[9] Drew V. McDermott, "Contexts and data dependencies: a synthesis," *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-5 (1983).

[10] Johan deKleer, "An assumption-based TMS," *Artificial Intelligence* 28 (1986), 127–162.