Learning Word Meanings by Instruction

Kevin Knight
Information Sciences Institute and Computer Science Department
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
knight@isi.edu

Abstract

We develop techniques for learning the meanings of unknown words in context. Working within a compositional semantics framework, we write down equations in which a sentence's meaning is some combination function of the meaning of its words. When one of the words is unknown, we ask for a paraphrase of the sentence. We then compute the meaning of the unknown word by inverting parts of the semantic combination function. This technique can be used to learn word-concept mappings, decomposed meanings, and mappings between syntactic and semantic roles. It works for all parts of speech.

Introduction

The acquisition of semantic knowledge is a major bottleneck for building high-performance natural language processing systems. This knowledge includes complex mappings between words and concepts as well as mappings between syntactic and semantic roles. Usually, these things must be specified manually by system designers, an approach akin to teaching by performing brain surgery—slow and expensive. We would like to extend our systems with greater ease.

Inductive Learning (IL) provides one possible paradigm for lexical semantic acquisition: a program synthesizes its own concepts, roles, and mappings by finding regularities in on-line text. Currently, there is no large corpora annotated with target sentential semantics, although experiments have been conducted with small data sets (Thompson 1995; Haas & Jayaraman 1993). And while some semantic features can be learned from raw text (Pereira, Tishby, & Lee 1993; Hatzivassiloglou & McKeown 1993; Jacobs & Zernik 1988; Zernik 1987), IL seems to be a far-off prospect for general acquisition of word meanings, and its data requirements bear no resemblance to the kind of one-shot learning that we observe in people.

Another approach is Explanation-Based Learning (EBL), which might also be called Expectation-Based Learning. If a system has very strong expectations (scriptal or semantic) about what will be uttered to it, then it can figure out the meanings of unknown words. One example of this approach is (Carbonell 1979). This system parses a sentence like *Russia sent massive arms shipments to the MPLA in Angola* with unknown word *MPLA*. Using its script-based knowledge of international relations, the system concludes that the MPLA is an Angolan communist political faction—or else a location inside Angola. Carbonell considers using IL to gradually converge on a single meaning, but opts instead for a guess-and-recover strategy. (Granger 1977) is another piece of work in this area. He considers sentences like *I woke up yesterday, turned off my alarm clock, took a shower, and cooked myself two grimps for breakfast.* Again, EBL can help us find a rough meaning for the unknown word *grimp*.

EBL is a powerful approach, but it depends on a great deal of world knowledge, something present systems lack. While IL does not suffer from this problem, shares another drawback with EBL: even people have trouble figuring out word meanings from context alone.

Fortunately, we can turn to more interactive paradigms, such as Learning by Instruction. Here, a person teaches a computer about new words and their meanings. The challenge is to make this instruction setting as natural as possible. Within the Learning by Instruction paradigm, two major approaches show promise:

- reduce semantic acquisition to a multiple choice test
- reduce semantic acquisition to a natural language dialogue

The first approach seeks to cover all possible cases, eliminating the need to "color outside the lines," and thereby eliminating the need for the expert system designer. This has worked well for syntax, where the cases and features are few, and the linguistic diagnostics fairly sharp. In semantics, a scenario might run as follows:

```
H:  "John is hungry."
C:  I don't know the meaning of "hungry."
    Is "very hungry" a reasonable phrase?
H:  Yes.
C:  Is "hungry" a visually detectable
    property?
```

```
H:  No.
C:  Does "hungry" refer to a capability
    (or lack of one)?
H:  No.
C:  Does "hungry" refer to a desire?
H:  Yes.
C:  Is it a desire to (a) do something,
    (b) own something, (c) be something,
    (d) other?
H:  (a).
C:  A desire to do what?
H:  To eat.
```

The system directs the acquisition with easily answerable questions, and builds semantic representations based on the user's responses. In the scenario above, multiple choice finally gives way to a freer-form input, but with careful design, this type of interaction can be limited. Note that the above scenario must be continued for the acquisition of syntax/semantics mapping information (e.g., the noun syntactically modified by *hungry* plays the role of agent in both the wanting and the eating events).

A second approach, pursued in this paper, is to allow free-form natural language interaction. This is a bootstrapping approach, in which the system uses its current knowledge of language to gain more. Semantics gets a constant workout, as we try to mimic the way people learn new words and mappings. Here is such a scenario:

```
H:  "John is hungry."
C:  I don't know the meaning of "hungry."
H:  I mean: "John wants to eat."
C:  (I know the meanings of "John",
     "wants", "to", "eat", and "is"...
     let me figure out what "hungry"
     means... okay, I've got it.)
```

The user has provided an interpretable paraphrase of the uninterpretable sentence. The system computes the meaning of the paraphrase, then deduces the meaning of the unknown word.

This sort of reasoning is similar to solving arithmetic equations. We might fancifully suppose that word meanings were numbers, and that the meaning of a sentence were an arithmetic combination of the meanings of its words. For example, *John is hungry* might mean:

$$\text{John} + \text{is} \cdot \text{hungry}^2 \tag{1}$$

and *John wants to eat* might mean:

$$\text{John} + \frac{5 \cdot \text{wants}}{\text{to}^{\text{eat}}} \tag{2}$$

If the meaning of *hungry* is unknown, and if we know that (1) and (2) are paraphrases of each other, we can solve for the unknown meaning by cancelling John and inverting the arithmetic functions:

$$\text{hungry} = \sqrt{\frac{5 \cdot \text{wants}}{\text{is} \cdot \text{to}^{\text{eat}}}} \tag{3}$$

Evaluating the right hand side of (3) yields a meaning for *hungry*.

Of course, meanings aren't numbers and we don't take their square roots. The rest of this paper is concerned with algorithms for learning word meanings within the framework of real computational semantics, in which meanings are things like lambda-calculus expressions or graphs, combining via function application or graph unification.

The technology we will develop has a number of applications. Learning meanings of new words is one. Another is lexical decomposition. We might first interpret *hungry* as a primitive one-place predicate. If we break its meaning down, we reduce the number of primitives, making it easier to specify semantic constraints and inference rules. Breaking down meanings is also useful for capturing intensional constructs like *alleged murderer* (paraphrase: *person who is said to be a murderer,* not *murderer* with the property *alleged*). Decomposition is critical for machine translation, where one language's word is another's lexical gap. For example, the Japanese verb *seimai-suru* means *to clean rice.* We can build this meaning automatically if we are told that *Yoko ga seimai-suru* means *Yoko cleans rice.* So paraphrases need not be in the same language. Finally, as a longer term goal, we aim toward processing dictionary and glossary definitions for massive lexicon construction.

Semantic Framework

This section lays out the framework for semantic interpretation that we will assume. The framework is well known, simple, and concrete, allowing us to specify new algorithms in detail. It is also one that is used in practical natural language systems like (Dowding *et al.* 1993) and (Knight *et al.* 1995).

In the tradition of Montague (Dowty, Wall, & Peters 1981), we adopt a compositional semantics, in which the meaning of a sentence is a function of the meanings of its parts. Those parts are phrases and, ultimately, words. So we need a database of word meanings and a set of functions to combine word and phrase meanings. These functions can be specified declaratively and applied with a handful of meta-operators. The lambda calculus is commonly used for representing semantic functions and data (Chierchia & McConnell-Ginet 1990). A flexible implementation popular in computational circles is via graphs and graph unification. (Moore 1989) gives descriptions of these notational systems. We will assume that word meanings and combination functions are all represented as labeled, directed acyclic graphs. We also assume that a syntactic analysis precedes semantic processing. Syntactic rules strongly restrict the ways in which we might decide to combine meanings, and our semantic

combination functions are keyed to them. Such a function might say, *if a constituent of type ADJ and a constituent of type N combine to form a new constituent of type NP, then the meaning of the NP is such-and-such a function of the meanings of the N and the ADJ.*[1] We also assume that the semantic combination rules themselves are fixed while we learn individual word meanings. This assumption is reasonable, given the no-longer-recent trends toward simplifying rules and pushing complexity into the lexicon (Karttunen 1986; Bresnan & Kaplan 1982).

Figure 1 shows formalized word meanings and combination functions for a sample sentence *John wants to eat.*[2] Graphs encode attributes and values, and values may be atoms or (recursively) graphs themselves. There are two important top-level attributes. The attribute `sem` encodes the core meaning—for example, *wants* refers to a `Wanting` with an agent (`agt`) and a wanted-thing (`goal`). The `map` attribute carries syntax-semantics role mapping information. A path through a graph corresponds to a role chain in knowledge representation languages. When two paths in a graph converge, then the two role chains must have the same value. In the case of *wants*, `sem agt` (semantic agent) is identified with `map subj sem` (semantics of the yet-to-be determined subject of the verb).

Semantic interpretation proceeds as a bottom-up traversal of the parse tree that assigns a meaning to each node. Leaf node meanings are retrieved from a dictionary of known words (from Figure 1). Assigning meanings to internal nodes requires three operators: *unification* (`u`), *extraction* (`e`), and *subordination* (`s`):

- `(u x y)` — unify graphs `x` and `y` by combining information from both (Robinson 1965; Shieber 1986).
- `(e f x)` — extract the subgraph of `x` beneath the top-level arc labeled `f`.
- `(s f x)` — create a new graph with a single arc labeled `f`, pointing to graph `x`.

[1] Note that we assume a syntactic parse is computable even if the sentence contains a semantically unknown word; learning both lexical syntax and lexical semantics simultaneously is a topic for future investigation.

[2] The linguistic structures in Figure 1 implement one simple account of the syntax/semantics of the sample sentences. There are other analyses. In particular, our method of discharging verb-argument features is not the best; see (Pollard & Sag 1994) for another treatment. Also note that we handle the subject-control facts of the verb *wants* during semantic interpretation; however, our role-filling takes the key syntactic processes into account. For example, our lexical entry for *wants* might have directly identified the agent of Wanting with the agent of the goal of Wanting (i.e., `sem agt = sem goal agt`), eschewing complicated subject role mapping (i.e., `map inf map subj = map subj`). But this would not generalize to sentences like *John wants to be seen,* where it is apparent that syntactic subjects should be unified, not semantic agents. This point will come up again when we consider the options that face a learning program.

A meaning is assigned to each internal node as follows: (1) retrieve a semantic combination function (a graph) based on the syntactic categories of the node and its parse-tree children, (2) create a new graph for each nth parse-tree child by subordinating its previously computed meaning graph to feature `xn`, (3) unify all of the graphs from these two steps, and (4) extract the `x0` subgraph of the result. This is repeated for each node in the tree.

Figure 2 traces the semantic interpretation of the sample sentence. The meaning of *John wants to eat* is:

```
(e x0 (u R4
        (u (s x1 JOHN)
           (s x2 (e x0 (u R2
                         (u (s x1 WANTS)                    (4)
                            (s x2 (e x0 (u R1
                                          (u (s x1 TO)
                                             (s x2 EAT)]
```

which evaluates to the graph shown at the top left corner of the figure.

Solving for Unknowns

Interpretation goes smoothly when everything is known. But our premise is that one word is semantically unknown, in the sentence *John is hungry.* Even though *hungry* is not yet in the dictionary, we know by the rules of semantic combination that the meaning of *John is hungry* is computed by:

```
(e x0 (u R4
        (u (s x1 JOHN)                                      (5)
           (s x2 (e x0 (u R3
                         (u (s x1 IS)
                            (s x2 HUNGRY)]
```

We want to solve for the unknown word meaning, given that the expressions (4) and (5) are equal. That is, we want to write an equation like:

HUNGRY = (... (... IS ... EAT ...) ... WANTS ... TO)

with the ellipses filled in.

Our basic approach is to invert the operators that combine semantic data (i.e., rules and word meanings), just as we would invert arithmetic operators when solving arithmetic equations. There are only three operators to worry about: `u`, `e`, and `s`. The subordination operator `s` inverts easily. `(s-inv f x)` is just `(e f x)`. Unfortunately, `(e-inv f x)` is multi-valued. It is the set of all `z` such that `(e f z) = x`, and each `z` may contain different information under arcs other than `f`. Likewise for `(u x y)`.[3]

Rather than invert each operator, we will invert an idiom repeatedly used for semantic combination. We define a new operator `(n f x y)`:

`(n f x y) = {z | (e x0 (u x (s f z))) = y}`

Here, `x` is a semantic rule unified with all known constituents, `f` is the feature under which the unknown constituent goes, and `y` is the desired result. With this inversion operator `n`, we can write:

[3] Our arithmetic analogy in Section 1 also had this problem, as square roots may be either positive or negative.

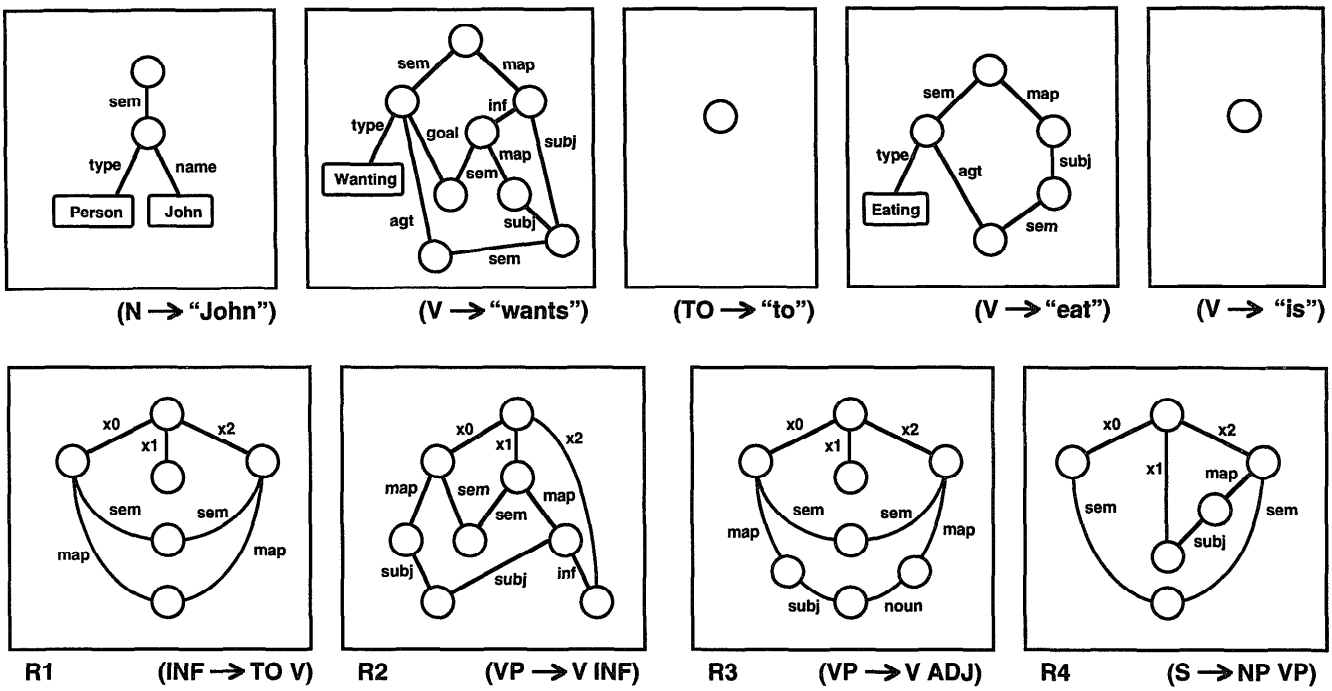

Figure 1: Word meanings and semantic combination rules.

```
HUNGRY =
    (n x2 (u R3 (s x1 IS))
       (n x2 (u R4 (s x1 JOHN))
          (e x0 (u R4
                   (u (s x1 JOHN)
                      (s x2 (e x0 (u R2
                                     (u (s x1 WANTS)
                                        (s x2 (e x0 (u R1
                                                       (u (s x1 TO)
                                                          (s x2 EAT)]
```

The depth of the unknown word in the parse tree determines the number of embedded calls to `n`. We now give an algorithm for computing `(n f x y)`:

Algorithm

1. Compute `(e f (u x (s x0 y)))`. Call this the *raw graph.* It forms the basis of the information to be contributed by the as-yet-unknown constituent, but it is not necessarily a subsumer or subsumee of the final product.
2. Repeatedly merge the pairs of nodes in the raw graph satisfying the following conditions:
   - the nodes have the same label (for leaf nodes).
   - the nodes have identical arc sets (i.e., the same labels and destinations).
   - merging the nodes yields a graph `h` that maintains `(e x0 (u x (s f h))) = y`.

   This merging proceeds bottom-up until no more merges are possible. Call the result the *merged graph. (Note: merging is a specialization operation.)*
3. Prune the merged graph. The merged graph generally contains too much information. Many prunings are possible, so this step returns multiple values, called *pruned graphs.* To prune, consider all subsets of arcs in the merged graph, including the empty set.[4] For each subset, form a new graph `h` by deleting the arcs in that set from the merged graph. If `h` satisfies `(e x0 (u x (s f h))) = y`, then keep it; otherwise throw it out. *(Note: pruning is a generalization operation.)*
4. Return the set of pruned graphs, removing duplicates.

Because calls to `(n f x y)` are nested, the value of `y` may be a set. In that case, we repeat the above algorithm for each element of `y`, combining all the results.

Example

We have implemented the above algorithm for ultimate use in acquiring semantic information for a large-scale machine translation system. This section shows results for the example we have been following in this paper.

Solving for the meaning of *hungry* yields 5 possible solutions, shown in Figure 3. All of these graphs "work" in the sense of achieving equivalent meanings for *John wants to eat* and *John is hungry,* and the first

[4] For efficient processing, it is only necessary to consider sets of individually pruneable arcs in the merged graph.

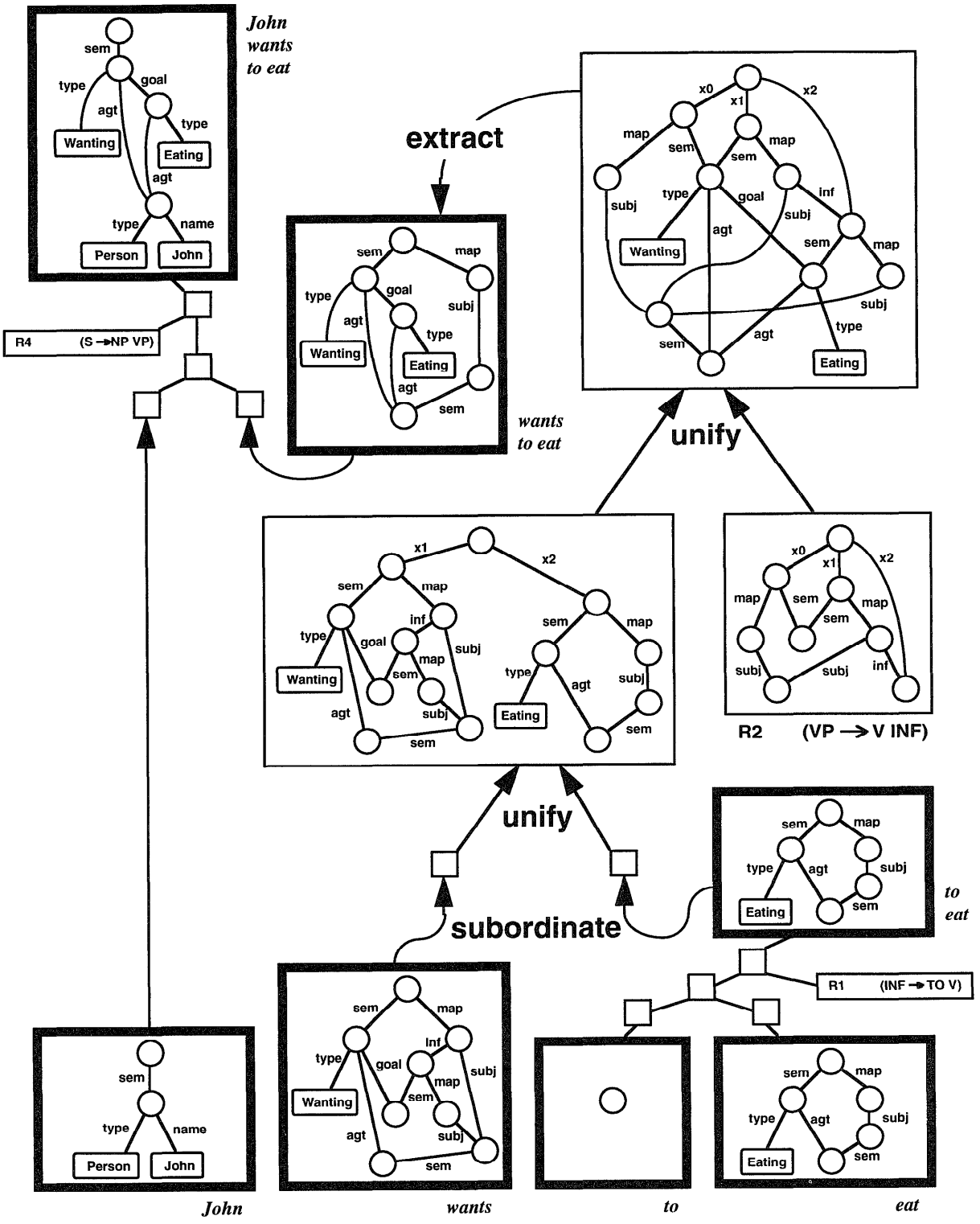

Figure 2: Word meanings combine to form a sentence meaning in a compositional framework.

graph is exactly what a human linguistic expert would produce. Of the other four meanings generated by the algorithm, one requires that the noun modified by *hungry* be a person—this would be reasonable if we used different adjectives to describe hunger in different types of entities. Less plausible is another meaning that restricts *hungry* to modify only things named John. Yet another meaning for *hungry* dispenses with the modified noun altogether: *X is hungry* always means *John wants to eat.*

To test the algorithm further, we can turn things around and assume that *hungry* is known, but *wants* is unknown. The case is interesting because the raw graph is significantly different from our target meaning (Figure 4). We write:

```
WANTS =
   (n x1 (u R2 (s x2 (e x0 (u R1 (u (s x1 TO) (s x2 EAT))))))
      (n x2 (u R4 (s x1 JOHN)
         (e x0 (u R4
                  (u (s x1 JOHN)
                     (s x2 (e x0 (u R3
                                    (u (s x1 IS)
                                       (s x2 HUNGRY)]
```

The algorithm generates 26 semantic candidates for *wants.* Many of these candidates include the direct unification of `(sem agt)` with `(sem goal agt)`, which works for *John wants to eat,* but will fail for sentences like *John wants to be seen,* where `(sem agt)` should be unified with `(sem goal patient)`, via the identification of the two syntactic subjects.

Heuristics

We now have as many as 26 solutions to an equation that is supposed to give us one lexical entry. Fortunately, there are a number of strategies for trimming the solution set.

One idea is to preemptively narrow the set of candidates using heuristics—i.e., go beyond the data and make an inductive leap. For example, for *hungry,* we can throw out candidates that mention John or Person, under the assumption that this information was contributed by the word *John.* This is heuristic—the same piece of information may actually come from several sources, as is commonly the case with agreement checking, e.g., `(x1 syn agr num) = (x2 syn agr num)`. A second heuristic is to discard specific pruned graphs in favor of more general ones, on the principle that we want the most succinct graph that does the job.[5] These heuristics narrow the solutions for *hungry* from 5 to 1 and for *wants* from 26 to 3.

Figure 5 shows solution sets narrowed with these heuristics, including an exact duplicate of the human expert's entry for *wants* (cf. Figure 1), as well as some incorrect (though rather ingenious) structures proposed by the algorithm.

[5]Note that these two heuristics have different effects. For example, the "more specific" heuristic only rules out three of the five learned graphs for *hungry.*

A completely different strategy is to use induction—i.e., maintain multiple entries for a word in the lexicon, and as the system encounters new sentences, jettison entries that stop working.

Discussion and Future Research

We have built an algorithm for learning the meanings of unknown words in context. The algorithm is driven by an interpretable paraphrase of the context, supplied by the user. The algorithm is general with respect to part of speech and can acquire word-concept mappings, lexical decompositions, and syntactic-semantic role mappings.

Previous work in this vein includes (Reeker 1976; Anderson 1977; Langley 1982). We have attempted to strip away the assumptions and heuristics used by these systems, and to provide a simple algorithm suited to modern computational linguistic methods, both practical and theoretical. More recent work by (Thompson 1995) is also aimed at learning lexical semantics; it works by induction rather than instruction, and it does not yet deal with case-role mappings. Combining inductive and instructive approaches to language learning is an open and potentially very interesting area of research. Other related work includes that on unification-based generation (Shieber *et al.* 1989) and on higher-order unification. Higher-order unification has been used to encode equations that provide solutions to the interpretation of ellipsis (Dalrymple, Shieber, & Periera 1991), and it would be useful to explore how this relates to our inversion algorithm.

Formal analysis of complexity and completeness would also be valuable. While much is known about first-order unification of feature structures (Kasper 1987; Ait-Kaci 1984; Carpenter 1992), comparatively little is known about its inverse. For example, one simple problem is: Given ϕ and ψ, compute all τ satisfying $\phi \sqcup \tau = \psi$. For atomic unification (variable symbol $\top$, failure symbol $\bot$, and atoms a_1, a_2, ...), the answer is:

ϕ	ψ	$\{\tau \mid \phi \sqcup \tau = \psi\}$
$\top$	$\top$	$\{\top\}$
$\top$	$\bot$	$\{\bot\}$
$\top$	a_i	$\{a_i\}$
$\bot$	$\top$	$\{\}$
$\bot$	$\bot$	$\{\top, \bot, a_1, a_2, \ldots\}$
$\bot$	a_i	$\{\}$
a_i	$\top$	$\{\}$
a_i	$\bot$	$\{\bot, a_1, \ldots, a_{i-1}, a_{i+1}, \ldots\}$
a_i	a_i	$\{\top, a_i\}$
a_i	a_j	$\{\}$

Tree unification, graph unification, and unification under equational theories are natural extensions to consider.

The problem addressed in this paper is a more complex variant: Given feature structures ϕ and ψ, and

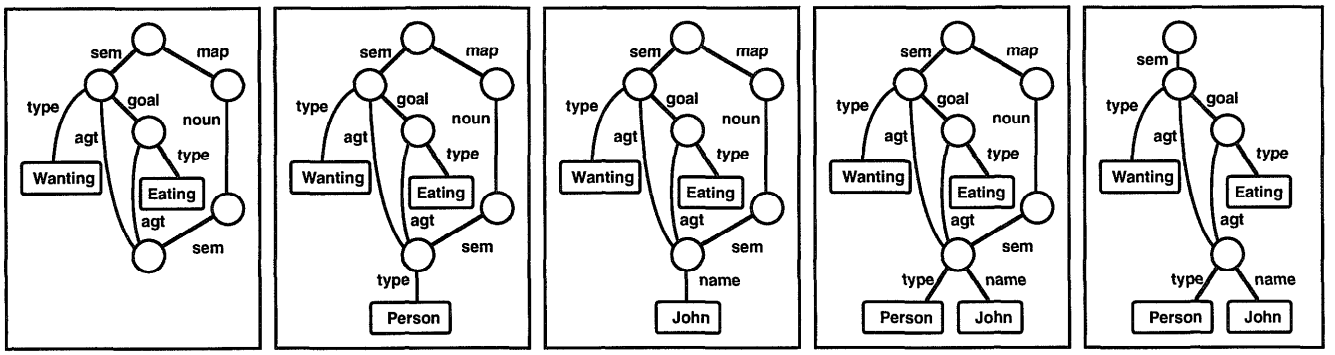

Figure 3: Sample solutions for *hungry* produced automatically.

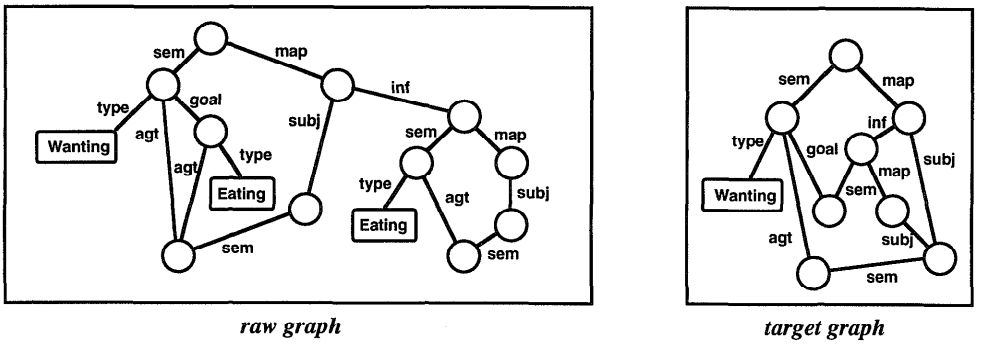

Figure 4: Raw graph and target graph for the meaning of *wants*.

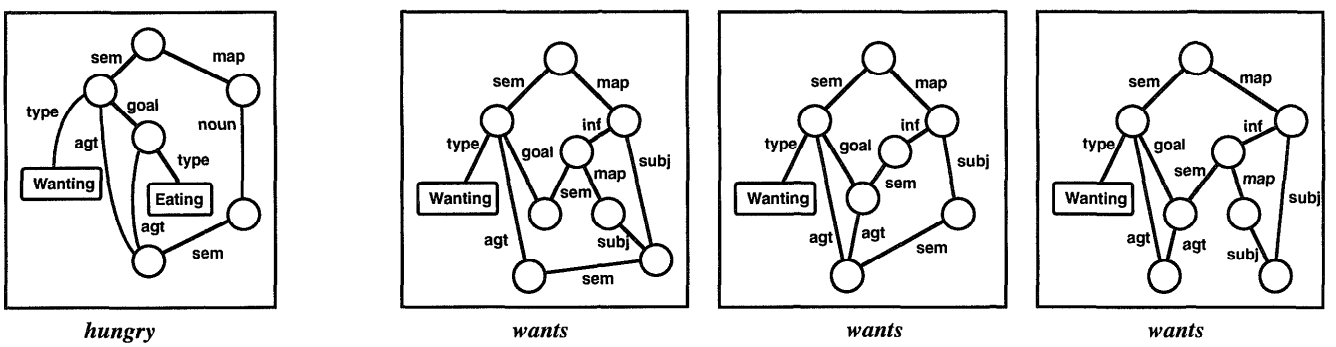

Figure 5: Sample solutions for *hungry* and *wants* narrowed heuristically.

given feature label x_i, compute all τ satisfying:

$$\phi \sqcup [\, x_i : \tau \,] = \begin{bmatrix} x_0 : \psi \\ x_1 : \text{don't care} \\ \vdots \quad\quad \vdots \\ x_n : \text{don't care} \end{bmatrix}$$

There are also several directions in which to extend this work for practical use. One is to learn syntactic and semantic structures at the same time. Another is to recognize when a known word should have its meaning extended. For example, *hungry for bananas* can be paraphrased as *wants to eat bananas.* Given this new *hungry* sentence, we want to add new role mapping information (for-PP maps to patient of the Eating, not the Wanting) to the existing lexical entry for *hungry.* Finally, we have assumed that the user-provided paraphrase is semantically unambiguous, but this will not always be the case. The learning algorithm may be able to reject some, but not all, of the paraphrase interpretations. The system will then need to interact with the user to disambiguate the paraphrase. If this interaction is to be natural, some ability to render different meanings unambiguously will be required.

Acknowledgments

Thanks to Ed Hovy and Yolanda Gil for comments on a draft of this paper. This research was carried out with funding from the Department of Defense.

References

Ait-Kaci, H. 1984. *A Lattice Theoretic Approach to Computation Based on a Calculus of Partially Ordered Type Structures.* Ph.D. Thesis, University of Pennsylvania.

Anderson, J. R. 1977. Induction of augmented transition networks. *Cognitive Science* 1.

Bresnan, J., and Kaplan, R. 1982. Lexical-Functional Grammar: a formal system for grammatical representation. In Bresnan, J., ed., *The Mental Representation of Grammatical Relations.* MIT Press (Cambridge, MA).

Carbonell, J. 1979. Towards a self-extending parser. In *Proc. ACL.*

Carpenter, B. 1992. *The Logic of Typed Feature Structures,* volume 32 of *Tracts in Theoretical Computer Science.* Cambridge: Cambridge University Press.

Chierchia, G., and McConnell-Ginet, S. 1990. *Meaning and Grammar: An Introduction to Semantics.* The MIT Press.

Dalrymple, M.; Shieber, S.; and Periera, F. 1991. Ellipsis and higher-order unification. *Linguistics and Philosophy* 14(4).

Dowding, J.; Gawron, J.; Appelt, D.; Bear, J.; Cherny, L.; Moore, R.; and Moran, D. 1993. Gemini: A natural language system for spoken-language understanding. In *Human Language Technology: Proc. ARPA Human Language Technology Workshop.* Plainsboro, NJ: ARPA.

Dowty, D. R.; Wall, R.; and Peters, S. 1981. *Introduction to Montague Semantics.* Dordrecht: Reidel.

Granger, R. 1977. Foulup: A program that figures out meanings of words from context. In *Proc. IJCAI.*

Haas, J., and Jayaraman, B. 1993. From context-free to definite-clause grammars: A type-theoretic approach. *Journal of Logic Programming* 12(1).

Hatzivassiloglou, V., and McKeown, K. 1993. Towards the automatic identification of adjectival scales: Clustering adjectives according to meaning. In *Proc. ACL.*

Jacobs, P., and Zernik, U. 1988. Acquiring lexical knowledge from text: A case study. In *Proc. AAAI.*

Karttunen, L. 1986. Radical lexicalism. Tech. Report CSLI-86-68, Center for the Study of Language and Information.

Kasper. 1987. *Feature Structures: A Logical Theory with Application to Language Analysis.* Ph.D. Thesis, University of Michigan.

Knight, K.; Chander, I.; Haines, M.; Hatzivassiloglou, V.; Hovy, E.; Iida, M.; Luk, S. K.; Whitney, R.; and Yamada, K. 1995. Filling knowledge gaps in a broad-coverage MT system. In *Proc. IJCAI.*

Langley, P. 1982. Language acquisition through error recovery. *Cognition and Brain Theory* 5(3).

Moore, R. 1989. Unification-based semantic interpretation. In *Proc. ACL.*

Pereira, F. C. N.; Tishby, N. Z.; and Lee, L. 1993. Distributional clustering of English words. In *Proc. ACL.* Harriman, New York: Morgan Kaufman.

Pollard, C., and Sag, I. 1994. *Head Driven Phrase Structure Grammar.* University of Chicago Press.

Reeker, L. H. 1976. The computational study of language acquisition. *Advances in Computers* 15.

Robinson, J. A. 1965. A machine-oriented logic based on the resolution principle. *Journal of the ACM* 12(1).

Shieber, S. M.; van Noord, G.; Moore, R. C.; and Pereira, F. C. N. 1989. A semantic-head-driven generation algorithm for unification based formalisms. In *Proc. ACL.*

Shieber, S. 1986. *An Introduction to Unification-Based Approaches to Grammar.* University of Chicago. Also, CSLI Lecture Notes Series.

Thompson, C. 1995. Acquisition of a lexicon from semantic representations of sentences. In *Proc. ACL, Student Session.*

Zernik, U. 1987. Language acquisition: Learning a hierarchy of phrases. In *Proc. IJCAI.*