TINA: A Natural Language System for Spoken Language Applications

Stephanie Seneff*†
Massachusetts Institute of Technology

A new natural language system, TINA, *has been developed for applications involving spoken language tasks.* TINA *integrates key ideas from context free grammars, Augmented Transition Networks (ATN's), and the unification concept.* TINA *provides a seamless interface between syntactic and semantic analysis, and also produces a highly constraining probabilistic language model to improve recognition performance. An initial set of context-free rewrite rules provided by hand is first converted to a network structure. Probability assignments on all arcs in the network are obtained automatically from a set of example sentences. The parser uses a stack decoding search strategy, with a top-down control flow, and includes a feature-passing mechanism to deal with long-distance movement, agreement, and semantic constraints.* TINA *provides an automatic sentence generation capability that has been effective for identifying overgeneralization problems as well as in producing a word-pair language model for a recognizer. The parser is currently integrated with MIT's* SUMMIT *recognizer for use in two application domains, with the parser screening recognizer outputs either at the sentential level or to filter partial theories during the active search process.*

1. Introduction and Overview

Over the past few years, there has been a gradual paradigm shift in speech recognition research both in the U.S. and in Europe. In addition to continued research on the transcription problem, i.e., the conversion of the speech signal to text, many researchers have begun to address as well the problem of speech understanding.[1] This shift is at least partly brought on by the realization that many of the applications involving human/machine interface using speech require an "understanding" of the intended message. In fact, to be truly effective, many potential applications demand that the system carry on a dialog with the user, using its knowledge base and information gleaned from previous sentences to achieve proper response generation. Current advances in research and development of spoken language systems[2] can be found, for example, in the proceedings of the DARPA speech and natural language workshops, as well as in publications from participants of the ESPRIT SUNDIAL project. Representative systems are described in Boisen et al. (1989), De Mattia and Giachin (1989), Niedermair (1989), Niemann (1990), and Young (1989).

* Spoken Language Systems Group, Laboratory for Computer Science, MIT, Cambridge MA 02139
† This research was supported by DARPA under Contract N00014-89-J-1332, monitored through the Office of Naval Research.
1 Speech understanding research flourished in the U.S. in the 1970s under DARPA sponsorship. While "understanding" was one of the original goals, none of the systems really placed any emphasis on this aspect of the problem.
2 We will use the term "speech understanding systems" and "spoken language systems" interchangeably.

A spoken language system relies on its natural language component to provide the meaning representation of a given sentence. Ideally, this component should also be useful for providing powerful constraints to the recognizer component in terms of permissible syntactic and semantic structures, given the limited domain. If it is to be useful for constraint, however, it must concern itself not only with coverage but also, and perhaps more importantly, with overgeneralization. In many existing systems, the ability to parse as many sentences as possible is often achieved at the expense of accepting inappropriate word strings as legitimate sentences. This had not been viewed as a major concern in the past, since systems were typically presented only with well-formed text strings, as opposed to errorful recognizer outputs.

The constraints can be much more effective if they are embedded in a probabilistic framework. The use of probabilities in a language model can lead to a substantially reduced perplexity[3] for the recognizer. If the natural language component's computational and memory requirements are not excessive, and if it is organized in such a way that it can easily predict a set of next-word candidates, then it can be incorporated into the active search process of the recognizer, dynamically predicting possible words to follow a hypothesized word sequence, and pruning away hypotheses that cannot be completed in any way. The natural language component should be able to offer significant additional constraint to the recognizer, beyond what would be available from a local word-pair or bigram[4] language model, because it is able to make use of long-distance constraints in requiring well-formed whole sentences.

This paper describes a natural language system, TINA, which attempts to address some of these issues. The mechanisms were designed to support a graceful, seamless interface between syntax and semantics, leading to an efficient mechanism for constraining semantics. Grammar rules are written such that they describe syntactic structures at the high levels of a parse tree and semantic structures at the low levels. All of the meaning-carrying content of the sentence is completely encoded in the names of the categories of the parse tree, thus obviating the need for separate semantic rules. By encoding meaning in the structural entities of the parse tree, it becomes feasible to realize probabilistic semantic restrictions in an efficient manner. This also makes it straightforward to extract a semantic frame representation directly from an unannotated parse tree.

The context-free rules are automatically converted to a shared network structure, and probability assignments are derived automatically from a set of parsed sentences. The probability assignment mechanism was deliberately designed to support an ability to predict a set of next-word candidates with associated *word* probabilities. Constraint mechanisms exist and are carried out through feature passing among nodes. A unique aspect of the grammar is that unification constraints are expressed one-dimensionally, being associated directly with categories rather than with rules. Syntactic and semantic fields are passed from node to node by default, thus making available by default the second argument to unification operations. This leads to a very efficient implementation of the constraint mechanism. Unifications introduce additional syntactic and semantic constraints such as person and number agreement and subject/verb semantic restrictions.

This paper is organized as follows. Section 2 contains a detailed description of the grammar and the control strategy, including syntactic and semantic constraint mech-

3 A technical term used in speech recognition to denote the geometric mean of the number of alternative word hypotheses that may follow each word.
4 Each word is associated with a list of the probabilites for all the words that could possibly follow it anywhere in a sentence.

anisms. Section 3 describes a number of domain-dependent versions of the system that have been implemented, and addresses, within the context of particular domains, several evaluation measures, including perplexity, coverage, and portability. Section 4 discusses briefly two application domains involving database access in which the parser provides the link between a speech recognizer and the database queries. The last section provides a summary and a discussion of our future plans. There is also an appendix, which walks through an example grammar for three-digit numbers, showing how to train the probabilities, parse a sentence, and compute perplexity on a test sentence.

2. Detailed Description

This section describes several aspects of the system in more detail, including how the grammar is generated and trained, how the control strategy operates, how constraints (both syntactic and semantic) are enforced, and practical issues having to do with efficiency and ease of debugging.

2.1 Overview

TINA is based on a context-free grammar augmented with a set of features used to enforce syntactic and semantic constraints. The grammar is converted to a network structure by merging common elements on the right-hand side (RHS) of all rules sharing the same left-hand side (LHS) category. Each LHS category becomes associated with a parent *node* whose children are the collection of unique categories appearing in the RHSs of all the rules in the common set. Each parent *node* establishes a two-dimensional array of permissible links among its children, based on the rules. Each child can link forward to all of the children that appear adjacent to that child in *any* of the shared rule set. Probabilities are determined for pairs of siblings through frequency counts on rules generated by parsing a set of training sentences. The parsing process achieves efficiency through structure-sharing among rules, resembling in this respect a top-down chart processor.

The grammar nodes are contained in a static structure describing a hierarchy of permissible sibling pairs given each parent, and a node-dependent set of constraint filters. Each grammar node contains a name specifying its category, a two-dimensional probability array of permissible links among the next lower level in the hierarchy and a list of filter specifications to be applied either in the top-down or the bottom-up cycle. When a sentence is parsed, a dynamic structure is created, a set of *parse nodes* that are linked together in a hierarchical structure to form explicit paths through the grammar. During the active parse process, the parse nodes are entered into a queue prioritized by their path scores. Each node (except terminals) in a given parse tree enters the queue exactly twice: once during the top-down cycle, during which it enters into the queue all of its possible first children, and once again during the bottom-up cycle, during which it enters all of its possible right siblings, given its parent. The control strategy repeatedly pops the queue, advancing the active hypothesis by exactly one step, and applying the appropriate node-level unifications.

Each feature specification for each grammar node contains a feature name, a value or set of values for that feature, a logic function, and a specification as to whether the unification should take place during the top-down or during the bottom-up cycle. All features are associated with nodes (categories) rather than with rules, and each node performs exactly the same unifications without regard to whatever rule it might be a part of. In fact, during the active parse process, a rule is not an explicit entity while it is being formed. Each instantiation of a rule takes place only at the time that the next sibling is the distinguished [end] node, a special node that signifies a return to the level of the parent. The rule can be acquired by tracing back through the left siblings, until the distinguished [start] node is encountered, although this is not done in practice until the entire parse is completed.

The parse nodes contain a set of features whose values will be modified through the unification process. All modifications to features are made nondestructively by copying a parse node each time a hypothesis is updated. Thus each independent hypothesis is associated with a particular parse node that contains all of the relevant feature information for that hypothesis. As a consequence, all hypotheses can be pursued in parallel, and no explicit backtracking is ever done. Control is repeatedly passed to the currently most probable hypothesis, until a complete sentence is found and all of the input stream is accounted for. Additional parses can be found by simply continuing the process.

2.2 Training the Probabilities

The grammar is built from a set of training sentences, using a bootstrapping procedure. Initially, each sentence is translated by hand into a list of the rules invoked to parse it. After the grammar has built up a substantial knowledge of the language, many new sentences can be parsed automatically, or with minimal intervention to add a few new rules incrementally. The arc probabilities can be incrementally updated after the successful parse of each new sentence.

The process of converting the rules to a network form is straightforward. All rules with the same LHS are combined to form a structure describing possible interconnections among children of a parent node associated with the left-hand category. A probability matrix connecting each possible child with each other child is constructed by counting the number of times a particular sequence of two siblings occurred in the RHSs of the common rule set, and normalizing by counting all pairs from the particular left-sibling to *any* right sibling.[5] Two distinguished nodes, a [start] node and an [end] node, are included among the children of every grammar node. A subset of the grammar nodes are terminal nodes whose children are a list of vocabulary words.

This process can be illustrated with the use of a simple example.[6] Suppose there exists a grammar for noun phrases that can be expressed through the single compact rule form:

Rule 1
[NP] ⇒ [article] ([adjective]) ([adjective]) [noun]

where the parentheses signify optional nodes. This grammar would be converted to a network as shown in Figure 1, which would be stored as a single grammar node with the name [NP]. The resulting grammar could be used to parse the set of phrases shown on the left, each of which would generate the corresponding rule shown on the right.

"the boy"	[NP] ⇒ [article] [noun]
"a beautiful town"	[NP] ⇒ [article] [adjective] [noun]
"a cute little baby"	[NP] ⇒ [article] [adjective] [adjective] [noun]
"the wonderful pudding"	[NP] ⇒ [article] [adjective] [noun]

5 In general, a particular rule will occur repeatedly in the training data, and each instantiation of the rule will add to the counts on its arcs.

6 A more complete example is given in the appendix.

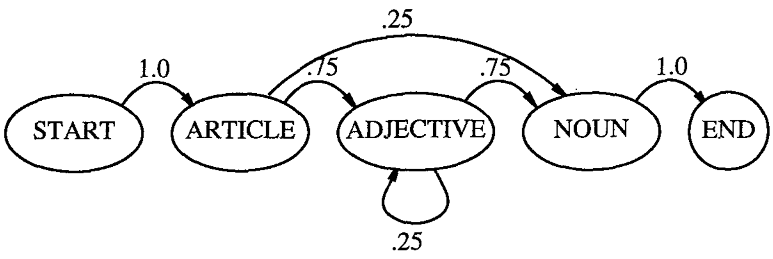

Figure 1
Illustration of probabilistic network obtained from four rules with the same LHS (NP), as given in the text. A parent node, named [NP], would contain these five nodes as its children, with a probability matrix specifying the network connections.

To train the probabilities, a record is kept of the relative counts of each subseqent sibling, with respect to each permissible child of the parent node, in our case, [NP], as they occurred in an entire set of parsed training sentences. In the example, [adjective] is followed three times by [noun] and once by [adjective], so the network shows a probability of 1/4 for the self loop and 3/4 for the advance to [noun]. Notice that the system has now generalized to include any number of adjectives in a row. Each rule in general would occur multiple times in a given training set, but in addition there is a significant amount of sharing of individual sibling pairs among different rules, the so-called cross-pollination effect.

This method of determining probabilities effectively amounts to a bigram language model[7] embedded in a hierarchical structure, where a separate set of bigram statistics is collected on category pairs for each unique LHS category name. The method is to be distinguished from the more common method of applying probabilities to entire rule productions, rather than to sibling pairs among a shared rule set. An advantage to organizing probabilities at the sibling-pair level is that it conveniently provides an explicit probability estimate for a single next word, given a particular word sequence. This probability can be used to represent the language model score for the next word, which, when used in conjunction with the acoustic score, provides the overall score for the word.

We make a further simplifying assumption that each sentence has only a single parse associated with it. This is probably justified only in conjunction with a grammar that contains semantic categories. We have found that, within the restricted domains of specific applications, the first parse is essentially always a correct parse, and often, in fact, the only parse. With only a single parse from each sentence, and with the grammar trained at the sibling-pair level, training probabilities becomes a trivial exercise of counting and normalizing sibling-pair frequencies within the pooled context-free rule sets. Training is localized such that, conditional on the parent, there is an advance from one sibling to some next sibling with probability 1.0. Normalization requires only this locally applied constraint, making it extremely fast to train on a set of parsed sentences. Furthermore, the method could incorporate syntactic and semantic constraints, by simply renormalizing the probabilities at run time, after paths that fail due to constraints have been eliminated.

7 A bigram language model is commonly used in speech recognition systems, where bigram statistics (frequency counts on adjacent word pairs) are collected from words or word categories in sample sentences.

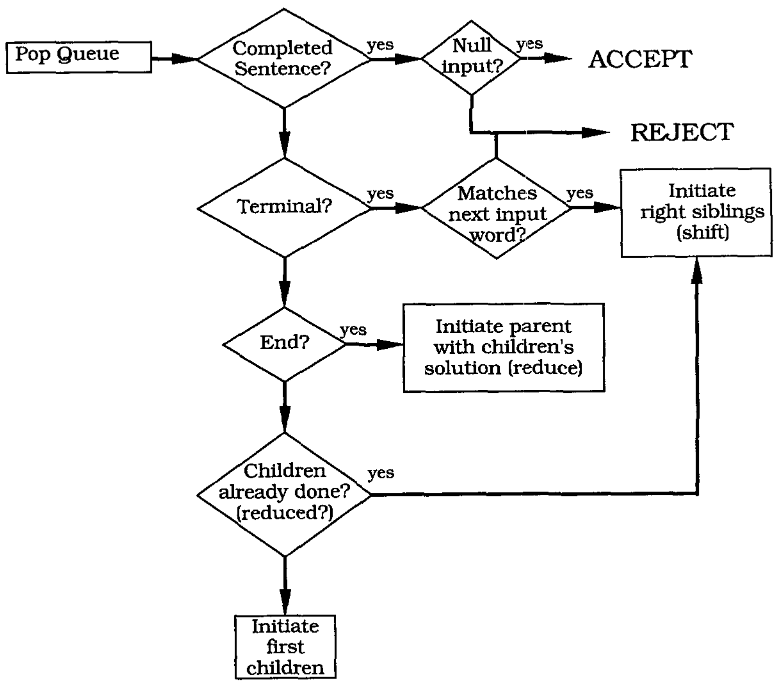

Figure 2
Functional block diagram of control strategy. (Note: "Initiate" means "enter into the queue ranked by probability.")

2.3 Control Strategy

A functional block diagram of the control strategy is given in Figure 2. At any given time, a set of active *parse nodes* are arranged on a priority queue. Each parse node contains a pointer to a corresponding grammar node, and has access to all the information needed to pursue its partial theory. The top node is popped from the queue, and it then creates a number of new nodes (either first children[8] or right siblings[9] depending on its state), and inserts them into the queue according to their probabilities. If the node is an [end] node, it returns control to the parent node, giving that node a completed subparse. As each new node is considered, unifications of syntactic and semantic constraints are performed, and may lead to failure. The process can terminate on the first successful completion of a sentence, or the Nth successful completion if more than one hypothesis is desired.

A parse in TINA begins with a single parse node linked to the grammar node [sentence], which is entered on the queue with probability 1.0. This node creates new parse nodes that might have categories such as [statement], [question], and [request], and places them on the queue, prioritized. If [statement] is the most likely child, it gets popped from the queue, and returns nodes indicating [subject], [it], etc., to the queue. When [subject] reaches the top of the queue, it activates units such as [noun phrase], [gerund], and [noun clause]. Each node, after instantiating first-children, becomes inactive, pending the return of a successful subparse from a sequence of children. Eventually, the cascade of first-children reaches a terminal node such as [article],

8 All of the categories that can initiate the RHS of rules containing its category on the LHS.
9 All of the categories that can follow its own category anywhere on the RHS in the common rule set sharing its parent's category on the LHS.

which proposes a set of words to be compared with the input stream. If a match with an appropriate word is found, then the terminal node fills its subparse slot with an entry such as ([article] "the"), and activates all of its possible right-siblings.

Whenever a terminal node has successfully matched an input word, the path probability is reset to 1.0.[10] Thus the probabilities that are used to prioritize the queue represent not the *total* path probability but rather the probability *given* the partial word sequence. Each path climbs up from a terminal node and back down to a next terminal node, with each new node adjusting the path probability by multiplying by a new conditional probability. The resulting conditional path probability for a next word represents the probability of that word in its linguistic role given all preceding words in *their* linguistic roles. With this strategy, a partial sentence does not become increasingly improbable as more and more words are added.

Because of the sharing of common elements on the right-hand side of rules, TINA can automatically generate new rules that were not explicitly provided. For instance, having seen the rule X $\Rightarrow$ A B C and the rule X $\Rightarrow$ B C D, the system would automatically generate two new rules, X $\Rightarrow$ B C and X $\Rightarrow$ A B C D. Although this property can potentially lead to certain problems with overgeneralization, there are a number of reasons why it should be viewed as a feature. First of all, it permits the system to generalize more quickly to unseen structures. For example, having seen the rule [question] $\Rightarrow$ [aux] [subject] [predicate] (as in "May I go?") and the rule [question] $\Rightarrow$ [have] [subject] [link] [pred-adjective] (as in "Has he been good?"), the system would also understand the forms [question] $\Rightarrow$ [have] [subject] [predicate] (as in "Has he left?") and [question] $\Rightarrow$ [aux] [subject] [link] [pred-adjective] (as in "Should I be careful?").[11] Secondly, it greatly simplifies the implementation, because rules do not have to be explicitly monitored during the parse. Given a particular parent and a particular child, the system can generate the allowable right siblings without having to note who the left siblings (beyond the immediate one) were. Finally, and perhaps most importantly, probabilities are established on arcs connecting sibling pairs regardless of which rule is under construction. In this sense the arc probabilities behave like the familiar word-level bigrams of simple recognition language models (Jelinek 1976), except that they apply to siblings at multiple levels of the hierarchy. This makes the probabilities meaningful as a product of conditional probabilities as the parse advances to deeper levels of the parse tree and also as it returns to higher levels of the parse tree. This approach implies an independence assumption that claims that what can follow depends only on the left sibling and the parent.

One negative aspect of the cross-pollination is that the system can potentially generalize to include forms that are agrammatical. For instance, the forms "Pick the box up" and "Pick up the box," if defined by the same LHS name, would allow the system to include rules producing forms such as "Pick up the box up" and "Pick up the box up the box!" This problem can be overcome either by giving the two structures different LHS names or by grouping "up the box" and "the box up" into distinct parent nodes, adding another layer to the hierarchy on the RHS. Another solution is to use a trace mechanism to link the two positions for the object, thus preventing it from occurring in both places. A final alternative is to include a PARTICLE bit among

10 Some modification of this scheme is necessary when the input stream is not deterministic. For the A* algorithm (Hart et al. 1968) as applied to speech recognition, the actual path score is typically augmented with an estimated score for the unseen portion. Unless some kind of normalization is done, the short theories have an unfair advantage, simply because fewer probability scores have been multiplied. With a deterministic word sequence it seems reasonable to assume probability 1.0 for what has been found.

11 The auxiliary verb sets the mode of the main verb to be root or past participle as appropriate.

the features which, once set, cannot be reset. In fact, there were only a few situations where such problems arose, and reasonable solutions could always be found.

2.4 Design Issues

TINA's design includes a number of features that lead to rapid development of the grammar and/or porting of the grammar to a new domain, as well as efficient implementation capabilities, in terms of both speed and memory. Among its features are semi-automatic training from a set of example sentences, a sentence generation capability, and a design framework that easily accomodates parallel implementations.

It is a two-step procedure to acquire a grammar from a specific set of sentences. The rule set is first built up gradually, by parsing the sentences one-by-one, adding rules and/or constraints as needed. Once a full set of sentences has been parsed in this fashion, the parse trees from the sentences are automatically converted to the sequence of rules used to parse each sentence. The training of both the rule set and the probability assignments is then established directly in a second pass from the provided set of parsed sentences; i.e., the parsed sentences *are* the grammar.

Generation mode uses the same routines as those used by the parser, but chooses a small subset of the permissible paths based on the outcome of a random-number generator, rather than exploring all paths and relying on an input word stream to resolve the correct one. Since all of the arcs have assigned probabilities, the parse tree is traversed by generating a random number at each node and deciding which arcs to select based on the outcome. The arc probabilities can be used to weigh the alternatives. Occasionally, the generator chooses a path that leads to a dead end, because of unanticipated constraints. Hence we in general need to keep more than one partial theory alive at any given time, to avoid having to backtrack upon a failure condition. We could in fact always choose to sprout two branches at any decision point, although this generally leads to a much larger queue than is really necessary. We found instead that it was advantageous to monitor the size of the queue, and arbitrarily increase the number of branches kept alive from one to two whenever the queue becomes dangerously short, shrinking it back to one upon recovery. We have used generation mode to detect overgeneralizations in the grammar, to build a word-pair language model for use as a simple constraint mechanism in our recognizer, and to generate random sentences for testing our interface with the back-end.

A final practical feature of TINA is that, as in unification grammars, all unifications are nondestructive, and as a consequence, explicit backtracking is never necessary. Every hypothesis on the queue is independent of every other one, in the sense that activities performed by pursuing one lead do not disturb the other active nodes. This feature makes TINA an excellent candidate for parallel implementation. The control strategy would simply deliver the most probable node to an available processor.

TINA has been implemented in Commonlisp and runs on both a Sun workstation and a Symbolics LISP machine. A deterministic word sequence can be parsed in a small fraction of real-time on either machine. Of course, once the input is a speech waveform rather than a word sequence, the uncertainty inherent in the proposed words will greatly increase the search space. Until we have a better handle on control strategies in the best-first search algorithm, it is impossible to predict the computational load for a spoken-input mode.

2.5 Constraints and Gaps

This section describes how TINA handles several issues that are often considered to be part of the task of a parser. These include agreement constraints, semantic restrictions, subject-tagging for verbs, and long distance movement (often referred to as *gaps*, or the *trace*, as in "(which article)$_i$ do you think I should read (t_i)?") (Chomsky 1977). The gap mechanism resembles the *Hold* register idea of ATNs (Woods 1970) and the treatment of bounded domination metavariables in lexical functional grammars (LFGs) (Bresnan 1982, p. 235 ff.), but it is different from these in that the process of filling the Hold register equivalent involves two steps separately initiated by two independent nodes.

Our approach to the design of a constraint mechanism is to establish a framework general enough to handle syntactic, semantic, and, ultimately, phonological constraints using identical functional procedures applied at the node level. The intent was to design a grammar for which the rules would be kept completely free of any constraints. To achieve this goal, we decided to break the constraint equations usually associated with rules down into their component parts, and then to attach constraints to *nodes* (i.e., categories) as equations in a single variable. The missing variable that must be unified with the new information would be made available by default. In effect, the constraint mechanism is thus reduced from a two-dimensional to a one-dimensional domain. Thus, for example, the developer would not be permitted to write an f-structure (Bresnan 1982) equation of the form $[\text{subj}]_{\text{Inf}} = [\text{np}]$ associated with the rule [vp] $\rightarrow$ [verb] [np] [inf], to cover, "I told John to go." Instead, the [np] node (regardless of its parent) would generate a CURRENT-FOCUS (defined later) from its subparse, which would be passed along passively to the verb "go." The verb would then simply consult the CURRENT-FOCUS (regardless of its source) to establish the identity of its subject.

The procedure works as follows. In the absence of any explicit instructions from its grammar node, a parse node simply passes along all features to the immediate relative (first child in the top-down cycle, and right sibling during the bottom-up cycle[12]). Any constraints specified by the grammar node result in a modification of certain feature values. The modifications are specified through a four-tuple of (feature-name new-value logic-function cycle). The possible features include person and number, case, determiner (DEFINITE, INDEFINITE, PROPER, etc.), mode (ROOT, FINITE, etc.), and a semantic category bit map. The new value, entered as a bit pattern, could be a single value, such as SINGULAR, or could be multiple valued as in the number for the noun "fish." Furthermore, during the bottom-up cycle, the new value can be the special variable *top-down-setting*, i.e., the value for that feature that currently occupies the slot in the parse node in question. This has the effect of disconnecting the node from its children, with respect to the feature in question. The logic function is one of AND, OR, or SET, and the cycle is either top-down or bottom-up.

A parse node has jurisdiction over its own slots only during the bottom-up cycle. During the top-down cycle, its feature value modifications are manifested only in its descendants. The node retains the values for the features that its parent delivered, and may use these for unifications prior to passing information on to its right siblings. This additional complexity was felt necessary to handle number agreement in questions of the form "Do John and Mary eat out a lot?" Here, the auxiliary verb "do" sets the number to plural, but the two individual nouns are singular. The SUBJECT node *blocks* transfer of number information to its children (by setting the value to all 1s), but unifies the value for number returned during the bottom-up cycle with the value previously delivered to it by its left sibling, the auxiliary verb. There is a node, [and-noun-phrase], that deals specifically with compound nouns. This node blocks transfer

12 If the right sibling happens to be the distinguished [end] node, then the features get passed up to the parent.

of number information to its children and *sets* number to plural during the bottom-up cycle.

It has been found expedient to define a meta-level operator named "detach" that invokes a block operation during both the top-down and bottom-up cycles. This operation has the effect of isolating the node in question from its descendents with respect to the particular blocked feature. This mechanism was commonly used to detach a subordinate clause from a main clause with respect to the semantic bits, for example. The setting that had been delivered to the node during the top-down cycle is retained and sent forward during the bottom-up cycle, but not communicated to the node's children. Another special blocking property can be associated with certain *features*, but the block only applies at the point where an [end] node returns a solution to a parent. This is true, for instance, of the mode for the verb.

Along with the syntactic and semantic features, there are also two slots that are concerned with the trace mechanism, and these are used as well for semantic filtering on key information from the past. There are some special operations concerned with filling these slots and unifying semantics with these slots that will be described in more detail in later sections.

Lexical entries contain three-tuple specifications of values for features; the fourth element is irrelevant since there are no separate top-down and bottom-up cycles. Thus a terminal verb node contains vocabulary entries that include settings for verb mode, and for person/number if the verb is finite. The plural form for nouns can be handled through a [pl] morph for the sake of efficiency. This morph *sets* the value of number to plural, regardless of its prior setting. It is the job of a parent node to unify that setting with the value delivered by the left siblings of the noun.

Some examples may help explain how the constraint mechanism works. Consider, for example, the ill-formed phrase "each boats." Suppose the grammar has the three rules, ([np] $\rightarrow$ [det] [noun]), ([noun] $\rightarrow$ [root-noun]), and ([noun] $\rightarrow$ [root-noun] [pl]). The lexical item "each" sets the number to singular and passes this value to the [noun] node. The [noun] node blocks transfer of number to its children. "Boat" sets the number to singular, but the [pl] morph overrides this value, returning a plural value to the parent. This plural value gets unified with the singular value that had been retained from "each" during the top-down cycle. The unification fails and the parse dies. By splitting off the plural morph, singular and plural nouns can share the bulk of their phonetics, thus reducing greatly the redundancy in the recognizer's matching problem. In theory, morphs could be split off for verbs as well, but due to the large number of irregularities this was not done.

Subject-verb agreement gets enforced by default, because the number information that was realized during the parsing of the subject node gets passed along to the predicate and down to the terminal verb node. The lexical item unifies the number information, and the parse fails if the result is zero. Any nonauxiliary verb node *blocks* the transfer of any predecessor person/number information to its right siblings during the bottom-up cycle, reflecting the fact that verbs agree in person/number with their subject but not their object.

Certain nodes set the *mode* of the verb either during the top-down or the bottom-up cycle. Thus, for example, "have" as an auxiliary verb sets mode to PAST-PARTICIPLE during the bottom-up cycle (i.e., for its *right-siblings*). The category [gerund] sets the mode to PRESENT-PARTICIPLE during the top-down cycle (for its *children*). Whenever a [predicate] node is invoked, the verb's mode has always been set by a predecessor.

2.5.1 Gaps. The mechanism to deal with gaps resembles in certain respects the *Hold* register idea of ATNs, but with an important difference, reflecting the design philosophy that no node can have access to information outside of its immediate domain. The mechanism involves two slots that are available in the feature vector of each parse node. These are called the CURRENT-FOCUS and the FLOAT-OBJECT, respectively. The CURRENT-FOCUS slot contains, at any given time, a pointer to the most recently mentioned content phrase in the sentence. If the FLOAT-OBJECT slot is occupied, it means that there is a gap somewhere in the future that will ultimately be filled by the partial parse contained in the FLOAT-OBJECT. The process of getting into the FLOAT-OBJECT slot (which is analogous to the *Hold* register) requires two steps, executed independently by two different nodes. The first node, the *generator*, fills the CURRENT-FOCUS slot with the subparse returned to it by its children. The second node, the *activator*, moves the CURRENT-FOCUS into the FLOAT-OBJECT position, for its children, during the top-down cycle. It also requires that the FLOAT-OBJECT be absorbed somewhere among its descendants by a designated *absorber* node, a condition that is checked during the bottom-up cycle. The CURRENT-FOCUS only gets passed along to siblings and their descendants, and hence is unavailable to activators at higher levels of the parse tree. That is to say, the CURRENT-FOCUS is a feature, like verb-mode, that is blocked when an [end] node is encountered. To a first approximation, a CURRENT-FOCUS reaches only nodes that are *c-commanded* (Chomsky 1977) by its generator. Finally, certain blocker *nodes* block the transfer of the FLOAT-OBJECT to their children.

A simple example will help explain how this works. For the sentence "(How many pies)$_i$ did Mike buy (t$_i$)?" as illustrated by the parse tree in Figure 3, the [q-subject] "how many pies" is a generator, so it fills the CURRENT-FOCUS with its subparse. The [do-question] is an activator; it moves the CURRENT-FOCUS into the FLOAT-OBJECT position. Finally, the object of "buy," an absorber, takes the [q-subject] as its subparse. The [do-question] refuses to accept any solutions from its children if the FLOAT-OBJECT has not been absorbed. Thus, the sentence "How many pies did Mike buy the pies?" would be rejected. Furthermore, the same [do-question] grammar node deals with the yes/no question "Did Mike buy the pies?," except in this case there is no CURRENT-FOCUS and hence no gap.

More complicated sentences involving nested or chained traces are handled straightforwardly by this scheme. For instance, the phrase, "Which hospital was Jane taken to?" can be parsed correctly by TINA, identifying "which hospital" as the object of the preposition "to" and "Jane" as the object of "taken." The phrase "which hospital" gets generated by the [q-subject] and activated by the following [be-question], thus filling the FLOAT-OBJECT slot. When the predicate of the clause is reached, the word "Jane" is in the CURRENT-FOCUS slot, and the phrase "which hospital" is still in the FLOAT-OBJECT slot. The [participial-phrase] for "taken [object]" activates "Jane," but only for its children. This word is ultimately absorbed by the [object] node within the verb phrase. Meanwhile, the [participial-phrase] passes along the original FLOAT-OBJECT ("which hospital") to its right sibling, the adverbial prepositional phrase, "to [object]." The phrase "which hospital" is finally absorbed by the preposition's object.

The example used to illustrate the power of ATNs (Woods 1986), "John was believed to have been shot," also parses correctly, because the [object] node following the verb "believed" acts as both an absorber and a (re)generator. Cases of crossed traces are automatically blocked because the second CURRENT-FOCUS gets moved into the FLOAT-OBJECT position at the time of the second activator, overriding the preexisting FLOAT-OBJECT set up by the earlier activator. The wrong FLOAT-OBJECT is available at the position of the first trace, and the parse dies:

*(Which books)$_i$ did you ask John (where)$_j$ Bill bought (t$_i$) (t$_j$)?

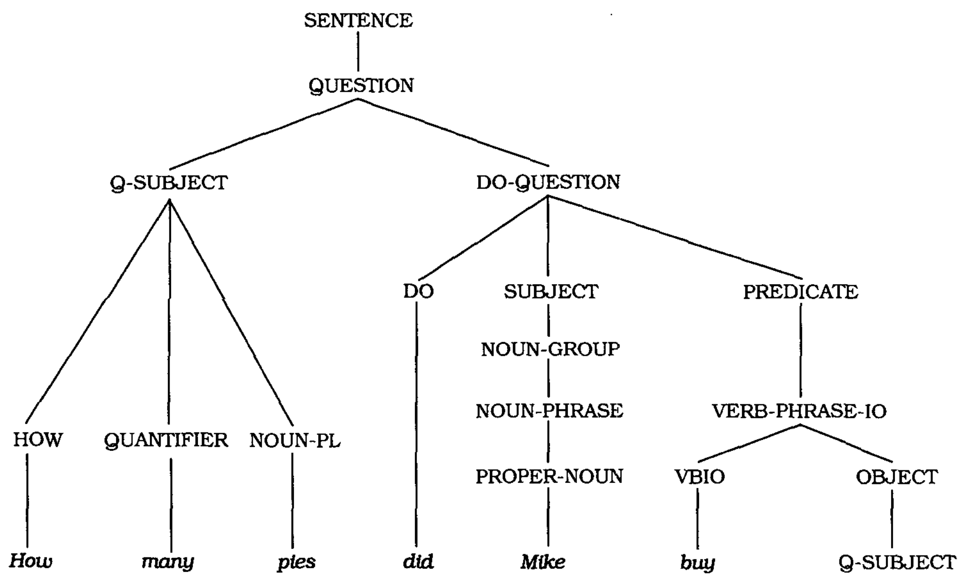

Figure 3
Example of a parse tree illustrating a gap.

The CURRENT-FOCUS slot is not restricted to nodes that represent nouns. Some of the generators are adverbial or adjectival parts of speech (POS). An absorber checks for agreement in POS before it can accept the FLOAT-OBJECT as its subparse. As an example, the question, "(How oily)$_i$ do you like your salad dressing (t_i)?" contains a [q-subject] "how oily" that is an adjective. The absorber [pred-adjective] accepts the available float-object as its subparse, but only after confirming that POS is ADJECTIVE.

The CURRENT-FOCUS has a number of other uses besides its role in movement. It always contains the subject whenever a verb is proposed, including verbs that are predicative objects of another verb, as in "I want to go to China." It has also been found to be very effective for passing semantic information to be constrained by a future node, and it can play an integral role in pronoun reference. For instance, a reflexive pronoun nearly always refers back to the CURRENT-FOCUS, whereas a nonreflexive form never does, unless it is in the nominative case.

2.5.2 Semantic Filtering. In the more recent versions of the grammar, we have implemented a number of semantic constraints using procedures very similar to those used for syntactic constraints. We found it effective to filter on the CURRENT-FOCUS'S semantic category, as well as to constrain absorbers in the gap mechanism to require a match on semantics before they could accept a FLOAT-OBJECT. Semantic categories were

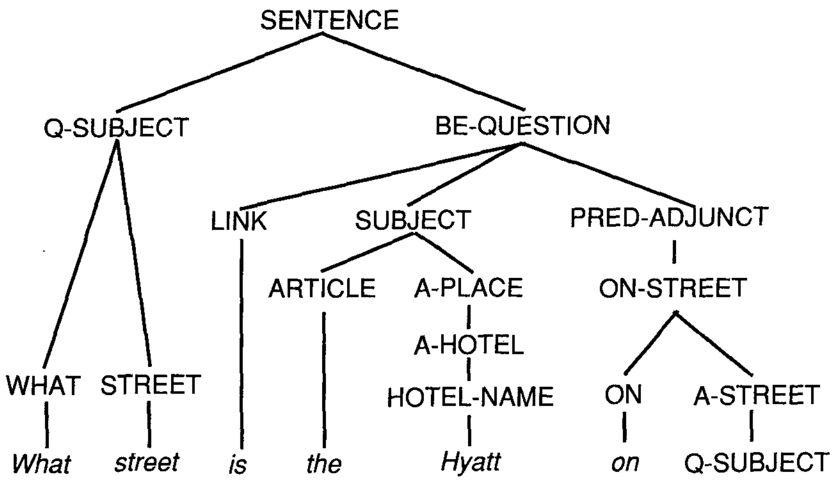

Figure 4
Parse tree for the sentence, "What street is the Hyatt on?"

implemented in a hierarchy such that, for example, RESTAURANT automatically inherits the more general properties BUILDING and PLACE. We also introduced semantically loaded categories at the low levels of the parse tree. It seems that, as in syntax, there is a trade-off between the number of unique node-types and the number of constraint filtering operations. At low levels of the parse tree it seems more efficient to label the categories, whereas information that must pass through higher levels of the hierarchy is better done through constraint filters.

As an example, consider the sentence, "(what street)$_i$ is the Hyatt on (t$_i$)?" shown in Figure 4. The [q-subject] places "What street" into the CURRENT-FOCUS slot, but this unit is activated to FLOAT-OBJECT status by the subsequent [be-question]. The [subject] node refills the now empty CURRENT-FOCUS with "the Hyatt." The node [a-street], an absorber, can accept the FLOAT-OBJECT as a solution, but only if there is tight agreement in semantics; i.e., it requires the identifier *Street*. Thus a sentence such as "What restaurant is the Hyatt on?" would fail on semantic grounds. Furthermore, the node [on-street] imposes strict semantic restrictions on the CURRENT-FOCUS. Thus the sentence "(What street)$_i$ is Cambridge on (t$_i$)?" would fail because [on-street] does not permit *Region* as the semantic category for the CURRENT-FOCUS, "Cambridge."

One place where semantic filtering can play a powerful role is in subject/verb relationships. This is easily accomplished within TINA's framework because the CURRENT-FOCUS slot always contains the subject of a verb at the time of the verb's instantiation. This is obvious in the case of a simple statement or complete clause, since the [subject] node generates a current-focus, which is available as the subject of the terminal verb node in the subsequent [predicate]. The same [subject] current-focus is also available as the subject of a verb in a predicative object of another verb, as in "I want to *go* to

China." For the case where a verb takes an object and an infinitive phrase as arguments, the [object] node replaces the current-focus with its subparse, such that when the verb of the infinitive phrase is proposed, the correct subject is available. This handles cases like "I asked Jane to *help*." With this mechanism, the two sentences "I want to go" and "I want John to go" can share the same parse node for the verb *want*.

Certain sentences exhibit a structure that superficially resembles the verb-object-infinitive-phrase pattern but should not be represented this way, such as "I avoid cigarettes to stay healthy." Here, clearly, "I" is the subject of "stay." This can be realized in TINA by having a top-level rule, ([statement] → [subject] [predicate] [adjunct-why]). The [object] node for "cigarettes" replaces the CURRENT-FOCUS, but the replacement does not get propagated back up to the [predicate] node (since a current-focus is passed only to siblings and children, but not to parents). Thus, the CURRENT-FOCUS "I" is passed on from the predicate to the adjunct, and eventually to the verb "stay."

Finally, in the case of passive voice, the CURRENT-FOCUS slot is empty at the time the verb is proposed, because the CURRENT-FOCUS which was the surface-form subject has been moved to the float-object position. In this case, the verb has no information concerning its subject, and so it identifies it as an unbound pronoun.

Semantic filters can also be used to prevent multiple versions of the same case frame (Fillmore 1968) showing up as complements. For instance, the set of complements [from-place], [to-place], and [at-time] are freely ordered following a movement verb such as "leave." Thus a flight can "leave for Chicago from Boston at nine," or, equivalently, "leave at nine for Chicago from Boston." If these complements are each allowed to follow the other, then in TINA an infinite sequence of [from-place]s, [to-place]s and [at-time]s is possible. This is of course unacceptable, but it is straightforward to have each node, as it occurs, *or in* a semantic bit specifying its case frame, and, in turn, fail if that bit has already been set. We have found that this strategy, in conjunction with the capability of erasing all semantic bits whenever a new clause is entered (through the meta level "detach" operation mentioned previously) serves the desired goal of eliminating the unwanted redundancies.

Thus far, we have added all semantic filters by hand, and they are implemented in a hard-fail mode, i.e., if the semantic restrictions fail, the node dies. This strategy seems to be adequate for the limited domains that we have worked with thus far, but they will probably be inadequate for more complex domains. In principle, one could parse a large set of sentences with semantics turned off, collecting the semantic conditions that occurred at each node of interest. Then the system could propose to a human expert a set of filters for each node, based on its observations, and the human could make the final decision on whether to accept the proposals. This approach resembles the work by Grishman et al. (1986) and Hirschman et al. (1975) on selectional restrictions. The semantic conditions that pass could even ultimately be associated with probabilities, obtained by frequency counts on their occurrences. There is obviously a great deal more work to be done in this important area.

3. Evaluation Measures

This section addresses some performance measures for a grammar, including coverage, portability, perplexity, and trainability. Perplexity, roughly defined as the geometric mean of the number of alternative word hypotheses that may follow each word in the sentence, is of particular concern in spoken language tasks. Portability and trainability concern the ease with which an existing grammar can be ported to a new task, as well as the amount of training data necessary before the grammar is able to generalize well to unseen data.

To date, four distinct domain-specific versions of TINA have been implemented. The first version (TIMIT) was developed for the 450 phonetically rich sentences of the TIMIT database (Lamel et al. 1986). The second version (RM) concerns the Resource Management task (Pallett 1989) that has been popular within the DARPA community in recent years. The third version (VOYAGER) serves as an interface both with a recognizer and with a functioning database back-end (Zue et al. 1990). The VOYAGER system can answer a number of different types of questions concerning navigation within a city, as well as provide certain information about hotels, restaurants, libraries, etc., within the region. A fourth domain-specific version is under development for the ATIS (Air Travel Information System) task, which has recently been designated as the new common task for the DARPA community.

3.1 Portability

We tested ease of portability for TINA by beginning with a grammar built from the 450 TIMIT sentences and then deriving a grammar for the RM task. These two tasks represent very different sentence types. For instance, the overwhelming majority of the TIMIT sentences are statements, whereas the RM task is made up exclusively of questions and requests. The process of conversion to a new grammar involves parsing the new sentences one by one, and adding context-free rules whenever a parse fails. The person entering the rules must be very familiar with the grammar structure, but for the most part it is straightforward to identify and incrementally add missing rules. The parser identifies where in the sentence it fails, and also maintains a record of the successful partial parses. These pieces of information usually are adequate to pinpoint the problem. Once the grammar has been expanded to accomodate the new set of sentences, a subset grammar can be created automatically that only contains rules needed in the new domain, eliminating any rules that were particular to the original domain. It required less than one person-month to convert the grammar from TIMIT to the RM task.

3.2 Perplexity and Coverage in RM Task

A set of 791 sentences within the RM task have been designated as training sentences, and a separate set of 200 sentences as the test set. We built a subset grammar from the 791 parsed training sentences, and then used this grammar to test coverage and perplexity on the unseen test sentences. The grammar could parse 100% of the training sentences and 84% of the test sentences.

A formula for the test set perplexity (Lee 1989) is:[13]

$$Perplexity = 2^{-\frac{1}{N}\sum_{i=1}^{N}\log_2 P(w_i \mid w_{i-1},\ldots w_1)}$$

where the w_i are the sequence of all words in all sentences, N is the total number of words, including an "end" word after each sentence, and $P(w_i \mid w_{i-1},\ldots w_1)$ is the probability of the ith word given all preceding words.[14] If all words are assumed equally likely, then $P(w_i \mid w_{i-1},\ldots w_1)$ can be determined by counting all the words that could follow each word in the sentence, along all workable partial theories. If the grammar contains probability estimates, then these can be used in place of the equally

13 The appendix includes an example for computing test set perplexity.

14 In the case of TINA, all words up to the current word within each sentence are relevant.

Table 1
Summary of perplexity and coverage within the Resource Management domain, for the 200 designated test sentences.

Vocabulary Size	Coverage	Perplexity No Probabilities	Perplexity With Probabilities
985	84%	368	41.5

Table 2
Ranking of first reasonable parse in the Resource Management task.

	Top 1	Top 2	Top 3	Top 7
Training	88%	96%	98%	100%
Test	90%	96%	99%	99%

likely assumption. If the grammar's estimates reflect reality, the estimated probabilities will result in a reduction in the total perplexity.

An average perplexity for the 167 test sentences that were parsable was computed for the two conditions, without (Case 1) and with (Case 2) the estimated probabilities. The result was a perplexity of 368 for Case 1, but only 41.5 for Case 2, as summarized in Table 1. This is with a total vocabulary size of 985 words, and with a grammar that included some semantically restricted classes such as [ship-name] and [readiness-category]. The incorporation of arc probabilities reduced the perplexity by a factor of nine, a clear indicator that a proper mechanism for utilizing probabilities in a grammar can help significantly. An even lower perplexity could be realized within this domain by increasing the number of semantic nodes. In fact, this is a trend that we have increasingly adopted as we move to new domains.

We didn't look at the test sentences while designing the grammar, nor have we yet looked at those sentences that failed to parse. However, we decided to examine the parse trees for those sentences that produced at least one parse to determine the depth of the first reasonable parse. The results were essentially the same for the training and the test sentences, as shown in Table 2. Both gave a reasonable parse as either the first or second proposed parse 96% of the time. Two of the test sentences never gave a correct parse.

3.3 Experiments within the VOYAGER domain

We have recently developed a subdomain for TINA that has been incorporated into a complete spoken language system called VOYAGER. The system provides directions on how to get from one place to another within an urban region, and also gives information such as phone number or address for places such as restaurants, hotels, libraries, etc. We have made extensive use of semantic filters within this domain, in order to reduce the perplexity of the recognition task as much as possible.

To obtain training and test data for this task, we had a number of naive subjects use the system as if they were trying to obtain actual information. Their speech was recorded in a simulation mode in which the speech recognition component was

Table 3
Perplexity and coverage data for test and training samples within the VOYAGER domain.

Data set:	Test	Test	Training
System:	initial	expanded	expanded
No Prob:	20.6	27.1	25.8
Prob:	7.1	8.3	8.1
Coverage:	69%	76%	78%

excluded. Instead, an experimenter in a separate room typed in the utterances as spoken by the subject. Subsequent processing by the natural language and response generation components was done automatically by the computer (Zue et al. 1989). We were able to collect a total of nearly 5000 utterances in this fashion. The speech material was then used to train the recognizer component, and the text material was used to train the natural language and back-end components.

We designated a subset of 3312 sentences as the training set, and augmented the original rules so as to cover a number of sentences that appeared to stay within the domain of the back-end. We did not try to expand the rules to cover sentences that the back-end could not deal with, because we wanted to keep the natural language component tightly restricted to sentences with a likely overall success. In this way we were able to increase the coverage of an independent test set of 560 utterances from 69% to 76%, with a corresponding increase in perplexity, as shown in Table 3. Perplexity was quite low even without probabilities; this is due mainly to an extensive semantic filtering scheme. Probabilities decreased the perplexity by a factor of three, however, which is still quite significant. An encouraging result was that both perplexity and coverage were of comparable values for the training and test sets, as shown in the table.

3.4 Generation Mode

As mentioned previously, generation mode has been a very useful device for detecting overgeneralization problems in a grammar. After the addition of a number of semantically loaded nodes and semantic filters, the VOYAGER version of the grammar is now restricted mainly to sentences that are semantically as well as syntactically legitimate. To illustrate this point we show in Table 4 five examples of consecutively generated sentences. Since these were not selectively drawn from a larger set, they accurately reflect the current performance level.

We also used generation mode to construct a word-pair grammar automatically for the recognizer component of our VOYAGER system. To do this, over 100,000 sentences were generated, and word-pair links were established for all words sharing the same terminal category (such as [restaurant-name], for all category-pairs appearing in the generated sentences. We could test completion by continuing until no new pairs were found. The resulting word pair grammar has a perplexity of over 70, in contrast to a perplexity of less than nine for the grammar used to construct it. This difference reflects the additional constraint of both the probabilities and the long-distance dependencies.

Table 4
Sample sentences generated consecutively by the VOYAGER version of TINA.

- Do you know the most direct route to Broadway Avenue from here?
- Can I get Chinese cuisine at Legal's?
- I would like to walk to the subway stop from any hospital.
- Locate a T-stop in Inman Square.
- What kind of restaurant is located around Mount Auburn in Kendall Square of East Cambridge?

4. Interfaces with the Recognizer and the Back-End

At present, we have available at MIT two systems, VOYAGER and ATIS, involving specific application domains in which a person can carry on a dialog with the computer, either through spoken speech or through text input. In both of these systems, TINA provides the interface between the recognizer and the application back-end. In this section, I will describe our current interfaces between TINA and the recognizer and our future plans in this area. In addition, I will describe briefly how we currently translate the parse tree into a semantic frame that serves as the input to database access and text response generation. This aspect of the system is beyond the scope of this paper, and therefore it will not be covered in detail.

The recognizer for these systems is the SUMMIT system (Zue et al. 1989), which uses a segmental-based framework and includes an auditory model in the front-end processing. The lexicon is entered as phonetic pronunciations that are then augmented to account for a number of phonological rules. The search algorithm is the standard Viterbi search (Viterbi 1967), except that the match involves a network-to-network alignment problem rather than sequence-to-sequence.

When we first integrated this recognizer with TINA, we used a "wire" connection, in that the recognizer produced a single best output, which was then passed to TINA for parsing. A simple word-pair grammar constrained the search space. If the parse failed, then the sentence was rejected. We have since improved the interface by incorporating a capability in the recognizer to propose additional solutions in turn once the first one fails to parse (Zue et al. 1991) To produce these "N-best" alternatives, we make use of a standard A* search algorithm (Hart 1968, Jelinek 1976). Both the A* and the Viterbi search are left-to-right search algorithms. However, the A* search is contrasted with the Viterbi search in that the set of active hypotheses take up unequal segments of time. That is, when a hypothesis is scoring well it is allowed to procede forward, whereas poorer scoring hypotheses are kept on hold.

We have thus far developed two versions of the control strategy, a "loosely coupled" system and a "tightly coupled" system. Both versions begin with a Viterbi search all the way to the end of the sentence, resulting in not only the first candidate solution but also partial scores for a large set of other hypotheses. If this first solution fails to parse, then the best-scoring partial theory is allowed to procede forward incrementally. In an A* search, the main issue is how to get an estimate of the score for the unseen portion of the sentence. In our case, we can use the Viterbi path to the end as the estimate of the future score. This path is guaranteed to be the best way to get to the end; however, it may not parse. Hence it is a tight upper bound on the true score for the rest of the sentence. The recognizer can continue to propose hypotheses until one successfully parses, or until a quitting criterion is reached, such as an upper bound on N.

Whereas in the loosely coupled system the parser acts as a filter only on completed candidate solutions (Zue et al. 1991), the tightly coupled system allows the parser to discard partial theories that have no way of continuing. Following the Viterbi search, each partial theory is first extended by the parser to specify possible next words, which are then scored by the recognizer. We have not yet made use of TINA's probabilities in adjusting the recognizer scores on the fly, but we have been able to incorporate linguistic scores to resort N-best outputs, giving a significant improvement in performance (Goodine et al. 1991). Ultimately we want to incorporate TINA's probabilities directly into the A* search, but it is as yet unclear how to provide an appropriate upper bound for the probability estimate of the unseen portion of the linguistic model.

Once a parser has produced an analysis of a particular sentence, the next step is to convert it to a meaning representation form that can be used to perform whatever operations the user intended by speaking the sentence. We currently achieve this translation step in a second-pass treewalk through the completed parse tree. Although the generation of semantic frames could be done on the fly as the parse is being proposed, it seems inappropriate to go through all of that extra work for large numbers of incorrect partial theories, due to the uncertainty as to the identity of the terminal word strings inherent in spoken input.

We have taken the point of view that all syntactic and semantic information can be represented uniformly in strictly hierarchical structures in the parse tree. Thus the parse tree contains nodes such as [subject] and [dir-object] that represent structural roles, as well as nodes such as [on-street] and [a-school] representing specific semantic categories. There are no separate semantic rules off to the side; rather, the semantic information is encoded directly as names attached to nodes in the tree.

Exactly how to get from the parse tree to an appropriate meaning representation is a current research topic in our group. However, the method we are currently using in the ATIS domain (Seneff et al. 1991) represents our most promising approach to this problem. We have decided to limit semantic frame types to a small set of choices such as CLAUSE (for a sentence-level concept, such as *request*), PREDICATE (for a functional operation), REFERENCE (essentially proper noun), and QSET (for a set of objects). The process of obtaining a completed semantic frame amounts to passing frames along from node to node through the completed parse tree. Each node receives a frame in both a top-down and a bottom-up cycle, and modifies the frame according to specifications based on its broad-class identity (as one of noun, noun-phrase, predicate, quantifier, etc.). For example, a [subject] is a noun-phrase node with the label "topic." During the top-down cycle, it creates a blank frame and inserts it into a "topic" slot in the frame that was handed to it. It passes the blank frame to its children, who will then fill it appropriately, labeling it as a QSET or as a REFERENCE. It then passes along to the right sibling the same frame that was handed to it from above, with the completed topic slot filled with the information delivered by the children.

The raw frame that is realized through the treewalk is post-processed to simplify some of the structure, as well as to augment or interpret expressions such as relative time. For example, the predicate modifier in "flights leaving at ten a.m." is simplified from a predicate *leave* to a modifier slot labeled *departure-time*. An expression such as "next Tuesday" is interpreted relative to today's date to fill in an actual month, date, and year. Following this post-analysis step, the frame is merged with references contained in a history record, to fold in information from the previous discourse.

The completed semantic frame is used in ATIS both to generate an SQL (Structured Query Language) command to access the database and to generate a text output to be spoken in the interactive dialog. The SQL pattern is controlled through lists of frame patterns to match and query fragments to generate given the match. Text generation is done by assigning appropriate temporal ordering for modifiers on nouns and for the main noun. The modifiers are contained in slots associated with the QSET frame. Certain frames such as *clock-time* have special print functions that produce the appropriate piece of text associated with the contents.

5. Discussion

This paper describes a new natural language system that addresses issues of concern in building a fully integrated spoken language system. The formalism provides an integrated approach to representations for syntax and for semantics, and produces a highly constraining language model to a speech recognizer. The grammar includes arc probabilities reflecting the frequency of occurrence of patterns within the domain. These probabilities are used to control the order in which hypotheses are considered, and are trained automatically from a set of parsed sentences, making it straightforward to tailor the grammar to a particular need. Ultimately, one could imagine the existence of a very large grammar that could parse almost anything, which would be subsetted for a particular task by simply providing it with a set of example sentences within that domain.

The grammar makes use of a number of other principles that we felt were important. First of all, it explicitly incorporates into the parse tree semantic categories intermixed with syntactic ones, rather than having a set of semantic rules provided separately. The semantic nodes are dealt with in the same way as the syntactic nodes; the consequence is that the node names alone carry essentially all of the information necessary to extract a meaning representation from the sentence. The grammar is not a semantic grammar in the usual sense, because it does include high level nodes of a syntactic nature, such as noun-clause, subject, predicate, etc.

A second important feature is that unifications are performed in a one-dimensional framework. That is to say, features delivered to a *node* by a close relative (sibling/parent/child) are unified with particular feature values associated with that node. The x variable in an x-y relationship is not explicitly mentioned, but rather is assigned to be "whatever was delivered by the relative." Thus, for example, a node such as [subject] unifies in exactly the same way, regardless of the rule under construction.

Another important feature of TINA is that the same grammar can be run in generation mode, making up random sentences by tossing the dice. This has been found to be extremely useful for revealing overgeneralization problems in the grammar, as well as for automatically acquiring a word-pair grammar for a recognizer and producing sentences to test the back-end capability.

We discussed a number of different application domains, and gave some performance statistics in terms of perplexity/coverage/overgeneralization within some of these domains. The most interesting result was obtained within the VOYAGER domain (see Sections 3.3 and 3.4). The perplexity (average number of words that can follow a given word) decreased from 70 to 28 to 8 when the grammar changed from word-pair (derived from the same grammar) to parser without probabilities to parser with probabilities.

We currently have two application domains that can carry on a spoken dialog with a user. One, the VOYAGER domain (Zue et al. 1990), answers questions about places of interest in an urban area, in our case, the vicinity of MIT and Harvard University. The second one, ATIS (Seneff et al. 1991), is a system for accessing data in the Official

Airline Guide and booking flights. Work continues on improving all aspects of these domains.

Our current research is directed at a number of different remaining issues. As of this writing, we have a fully integrated version of the VOYAGER system, using an A* search algorithm (Goodine et al. 1991). The parser produces a set of next-word candidates dynamically for each partial theory. We have not yet incorporated probabilities from TINA into the search, but they are used effectively to resort the final output sentence candidates. In order to incorporate the probabilities into the search we need a tight upper bound on the future linguistic score for the unseen portion of each hypothesis. This is a current research topic in our group. We also plan to experiment with further reductions in perplexity based on a discourse state. This should be particularly effective within the ATIS domain where the system often asks directed questions about as yet unresolved particulars to the flight.

6. Appendix: Sample Grammar Illustrating Probability Calculation and Perplexity Computation

This appendix walks through a pedagogical example to parse spoken digit sequences up to three long, as in "three hundred and sixteen." Included is a set of initial context-free rules, a set of training sentences, an illustration of how to compute the path probabilities from the training sentences, and an illustration of both parsing and perplexity computation for a test sentence.

Since there are only five training sentences, a number of the arcs of the original grammar are lost after training. This is a problem to be aware of in building grammars from example sentences. In the absence of a sufficient amount of training data, some arcs will inevitably be zeroed out. Unless it is desired to intentionally filter these out as being outside of the new domain, one can insert some arbitrarily small probability for these arcs, using, for example, an N-gram back-off model (Katz 1987).

The Grammar:
(parentheses indicate optional elements)

```
number = hundreds-place (tens-place) ones-place
number = tens-place
number = (tens-place) ones-place
hundreds-place = digits (hundred)
hundreds-place = a hundred (and)
tens-place = tens
tens-place = teens  (this overgeneralizes a bit)
tens-place = oh     (as in "four oh five")
ones-place = digits
tens = [twenty thirty forty ...]  (a terminal node with eight
       individual words)
digits = [zero one two three four....]
teens = [ten eleven twelve...]
oh = [oh]
hundred = [hundred]
and = [and]
```

The training sentences: (with spoken form)

```
1: 144 "one hundred and forty four"
```

```
2: 430 "four thirty"
3: 208 "two oh eight"
4:  24 "twenty four"
5: 114 "a hundred fourteen"
```

The training rules: (excluding terminals)

```
1: number = hundreds-place tens-place ones-place
   hundreds-place = digits hundred and
   tens-place = tens
   ones-place = digits

2: number = hundreds-place tens-place
   hundreds-place = digits
   tens-place = tens

3  number = hundreds-place tens-place ones-place
   hundreds-place = digits
   tens-place = oh
   ones-place = digits

4. number = tens-place ones-place
   tens-place = tens
   ones-place = digits

5. number = hundreds-place tens-place
   hundreds-place = a hundred
   tens-place = teens
```

The training pairs for "hundreds-place" (gathering together all rules in (1, 2, 3, 5) above that have "hundreds-place" on the LHS:

```
from 1: start digits, digits hundred, hundred and, and end
from 2: start digits, digits end
from 3: start digits, digits end
from 5: start a, a hundred, hundred end
```

The count array for "hundreds-place":

	digits	hundred	and	end	a	total
start	3	0	0	0	1	4
digits	0	1	0	2	0	3
hundred	0	0	1	1	0	2
and	0	0	0	1	0	1
a	0	1	0	0	0	1

The probability of a transition from start to digits, within the parent node "hundreds-place," is just 3/4, the ratio of the number of times "hundreds-place" started with "digits" over the number of times it started with anything.

Parsing the phrase "four fifteen" with the trained parser:
The initial stack:[15]

```
Child|Parent, Left Sibling      Path Probability
hundreds-place|number, start       4/5
tens-place|number, start           1/5
```

After "hundreds-place" gets popped and expanded:

```
digits|hundreds-place, start      4/5*3/4
tens-place|number, start           1/5
a|hundreds-place, start           4/5*1/4  (this is a tie score with
                                            the above)
```

After "digits|hundreds-place" is popped and a match with "four" is found:

```
end|hundreds-place, digits         2/3  (given "four" with certainty)
hundred|hundreds-place, digits     1/3  (this is the word "hundred")
tens-place|number, start           1/5
a|hundreds-place, start           4/5*1/4
```

After "end|hundreds-place, digits" is popped, "hundreds-place" has a solution in hand, "four." It now activates its only right sibling, "tens-place." This is a different instance of "tens-place" from the one at the third place in the stack. Its left sibling is "hundreds-place" rather than "start."

```
tens-place|number, hundreds-place   2/3
hundred|hundreds-place, digits      1/3
tens-place|number, start            1/5
a|hundreds-place, start            4/5*1/4
```

After "tens-place" is expanded, we have:

```
tens|tens-place, start             2/3*3/5
hundred|hundreds-place, digits       1/3
tens-place|number, start             1/5
a|hundreds-place, start            4/5*1/4
teens|tens-place, start            2/3*1/5
oh|tens-place, start               2/3*1/5
```

"Tens" and "hundred" will both get popped off and rejected, because there is no match with the word "fifteen." "Tens-place" will also get popped, and eventually rejected, because nothing within "tens-place" matches the digit "four." A similar fate meets the "a" hypothesis. Finally, "teens" will be popped off and matched, and "end|tens-place, teens" will be inserted at the top with probability 1.0. This answer will be returned to the parent, "tens-place," and two new hypotheses will be inserted at the top of the

15 To make the story simpler, I'm ignoring probabilities on the terminal word nodes.

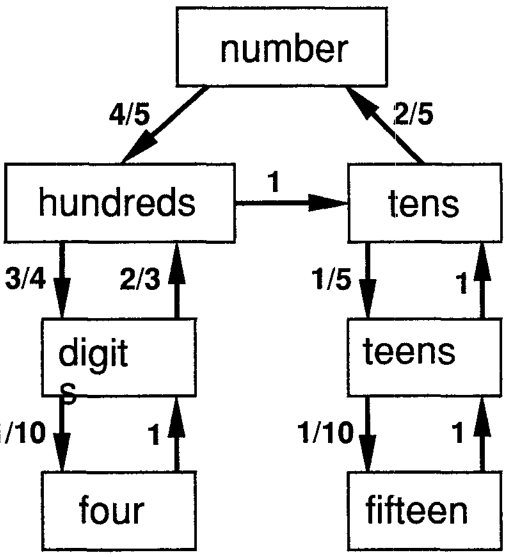

Figure A.1
Paths through the parse tree for the phrase "four fifteen" with associated probabilities derived from the training data.

stack as follows:

```
ones-place|number, tens-place          3/5
end|number, tens-place                 2/5
...
```

After the first one is rejected, the second one finds a completed "number" rule and an empty input stream. The correct solution is now in hand. Notice that because "teens" was a relatively rare occurrence, a number of incorrect hypotheses had to be pursued before the correct one was considered.

Computation of perplexity, for the phrase, "four fifteen:"

$$Perplexity = 2^{-\frac{1}{N}\sum_{i=1}^{N} log_2 P(w_i|w_{i-1},\ldots w_1)}$$

These are the three transitions with associated probabilities, following the appropriate paths in Figure A.1:

Transition		Probability
1: start	→ four	4/5*3/4*1/10
2: four	→ fifteen	1*2/3*1*1/5*1/10
3: fifteen	→ end	1*1*2/5

Thus, for this example test sentence:

$$Perplexity = 2^{-\frac{log_2(\frac{4}{5}\frac{3}{4}\frac{1}{10}) + log_2(\frac{2}{3}\frac{1}{5}\frac{1}{10}) + log_2(\frac{2}{5})}{3}}$$

This comes out to about 14 words on average following a given word, for this particular phrase. This is higher than the norm for numbers given the grammar, again because of the rare occurrence of the "teens" node, as well as the fact that there is no ones-place. This example is a bit too simple – in general there would be multiple ways to get to a particular next word, and there are also constraints which kill certain paths and make it necessary to readjust probabilities on the fly. In practice, one must find all possible ways to extend a word sequence, computing total path probability for each one, and then renormalize to assure that with probability 1.0 there is an advance to *some* next word. It is the normalized probability contribution of *all* paths that can reach the next word that is used to update the log P calculation.

Acknowledgments
This research has benefited significantly from interactions with Lynette Hirschman and Victor Zue. In addition, Jim Glass, David Goodine, and Christine Pao have all made significant contributions to the programming of the TINA system, for which I am deeply grateful. I would also like to thank several anonymous reviewers for their careful critiques, the outcome of which was a substantially improved document.

References

Boisen, S.; Chow, Y.-L.; Haas, A.; Ingria, R.; Roukos, S.; and Stallard, D. (1989). "The BBN spoken language system." In *Proceedings, DARPA Speech and Natural Language Workshop.* 106–111.

Bresnan, J., ed. (1982). *The Mental Representation of Grammatical Relations.* Cambridge, MA: The MIT Press.

Chomsky, Noam (1977). "On wh-movement." In *Formal Syntax,* edited by P. Culicover, T. Wasow, and A. Akmajian. New York: Academic Press.

De Mattia, M., and Giachin, E. P. (1989). "Experimental results on large vocabulary continuous speech understanding." *ICASSP-89 Proceedings.* 691–694.

Fillmore, C. J. (1968). "The case for case." In *Universals in Linguistic Theory,* edited by E. Bach and R. Harms. New York: Holt, Rinehart, and Winston. 1–90.

Goodine, D.; Seneff, S.; Hirschman, L.; and Phillips, M. (1991). "Full integration of speech and language understanding in the MIT spoken language system." In *Proceedings, 2nd European Conference on Speech Communication and Technology.* Genova, Italy. 24–26.

Grishman, R.; Hirschman, L.; and Nhan, N. T. (1986). "Discovery procedures for sublanguage selectional patterns: Initial experiments." *Computational Linguistics* 12(3): 205–215.

Hart, P.; Nilsson, N. J.; and Raphael, B. (1968). "A formal basis for the heuristic determination of minimum cost paths." *IEEE Transactions of Systems, Science and Cybernetics* SSC-4(2): 100–107.

Hirschman, L.; Grishman, R.; and Sager, N. (1975). "Grammatically-based automatic word class formation." *Information Processing and Management* 11: 39–57.

Jelinek, F. (1976). "Continuous speech recognition by statistical methods." *IEEE Proceedings* 64(4): 532–556.

Katz, S. M. (1987). "Estimation of probabilities from sparse data for the language model component of a speech recognizer." *ASSP-35*: 400–401.

Lamel, L.; Kassel, R. H.; and Seneff, S. (1986). "Speech database development: Design and analysis of the acoustic-phonetic corpus." In *Proceedings, DARPA Speech Recognition Workshop.* Palo Alto, CA. 100–109.

Lee, K. F. (1989). *Automatic Speech Recognition: The Development of the SPHINX System,* Appendix I. Boston: Kluwer Academic Publishers.

Lee, K. F.; Hon, H. W.; and Reddy, R. (1989). "An overview of the SPHINX speech recognition system." *IEEE Transactions on Acoustics, Speech, and Signal Processing* 38(1): 35–46.

Niedermair, G. Th. (1989). "The use of a semantic network in speech dialogue." 1st European Conference on Speech Communication and Technology, Paris, France. 26–29.

Niemann, H. (1990). "The interaction of word recognition and linguistic processing in speech understanding." Invited Lecture, NATO-ASI Workshop on Speech Recognition and Understanding, Cetraro, Italy.

Pallett, D. (1989). "Benchmark tests for DARPA resource management database performance evaluations." In *Proceedings, ICASSP-89.* 536–539.

Seneff, S.; Glass, J.; Goddeau, D.; Goodine, D.; Hirschman, L.; Leung, H.; Phillips, M.; Polifroni, J.; and Zue, V. (1991). "Development and preliminary evaluation of the MIT ATIS system."

Fourth DARPA Speech and Natural Language Workshop, Asilomar, CA. 88–93.

Viterbi, A. (1967). "Error bounds for convolutional codes and an asymptotically optimal decoding algorithm." *IEEE Transactions on Information Theory IT-13*. 260–269.

Woods, W. A. (1970). "Transition network grammars for natural language analysis." *Commun. of the ACM* 13: 591–606.

Woods, W. A. (1986). "Semantics and quantification in natural language question answering." In *Readings in Natural Language Processing*, edited by B. J. Grosz, K. S. Jones; and B. L. Webber. Los Altos, CA: Morgan Kaufmann. 205–248.

Young, S. R. (1989). "The minds system: Using context and dialog to enhance speech recognition." *Proceedings, DARPA Speech and Natural Language Workshop*. 131–136.

Zue, V.; Daly, N.; Glass, J.; Goodine, D.; Leung, H.; Phillips, M.; Polifroni, J.; Seneff, S.; and Soclof, M. (1989a). "The collection and preliminary analysis of a spontaneous speech database." DARPA Speech and Natural Language Workshop. Harwichport, MA. 15–18.

Zue, V.; Glass, J.; Phillips, M.; and Seneff, S. (1989b). "The MIT SUMMIT speech recognition system, a progress report." *Proceedings, DARPA Speech and Natural Language Workshop*. Philadelphia. 21–23.

Zue, V.; Glass, J.; Goodine, D.; Leung, H.; Phillips, M.; Polifroni, J.; and Seneff, S. (1990). "The VOYAGER speech understanding system: Preliminary development and evaluation." *IEEE International Conference on Acoustics, Speech and Signal Processing*. Albuquerque, NM. 73–76.

Zue, V.; Glass, J.; Goodine, D.; Leung, H.; Phillips, M.; Polifroni, J.; and Seneff, S. (1991). "Integration of speech recognition and natural language processing in the MIT VOYAGER system." *IEEE International Conference on Acoustics, Speech and Signal Processing*. Toronto, Ontario. 14–17.