Proc. of the Internat. Conf. Speech Language Processing, Beijing, China, Oct. 2000.

STATISTICAL RECURSIVE FINITE STATE MACHINE PARSING FOR SPEECH UNDERSTANDING

Alexandros Potamianos and Hong-Kwang J. Kuo

Bell Labs, Lucent Technologies, 600 Mountain Ave., Murray Hill, NJ 07974-0636, U.S.A.
email: {potam,kuo}@research.bell-labs.com

ABSTRACT

In this paper, a statistical framework for semantic parsing is described. The statistical model uses two information sources to disambiguate between rules: rule weights that capture vertical relationships in the parse tree, and concept n-grams that capture horizontal relationships. Rule design consists of simple local mapping rules that non-experts can write, and the rules are implemented as weighted finite state transducers. A general parser for context free grammars is implemented using a finite state machine library. Semantic decoding is implemented by recursively composing the rule transducer with the word-graph automaton produced from the speech recognizer. Detailed metrics for evaluating semantic parse accuracy are proposed. The parser is evaluated on the ATIS travel task with resulting precision and recall rates of over 95%. The proposed finite state transducer formulation allows the incorporation of rules and probabilities in a unified framework and the straightforward combination of acoustic, language, and understanding models.

1. INTRODUCTION

Current speech understanding systems either take a generative approach or a fully statistical approach. The former method suffers from lack of coverage and requires expert knowledge to design the grammars, while the latter requires a large amount of labeled training data. In this paper, a speech understanding system that combines the rich hierarchical representation of a parse tree with the power and robustness of statistical modeling is presented. Our goal is to build an understanding module that requires a small amount of training data and a grammar designer that is not necessarily an expert linguist.

There has been a lot of research effort in syntactic parsing (see [2, 5] for a review). Formal evaluation of these parsing algorithms (typically on the Penn Treebank) can also be found in the literature. Work on semantic parsing, however, is less formal and harder to evaluate due to lack of annotated corpora. Also semantic and syntactic parsing have different goals. In semantic parsing, the goal is to extract relevant semantic fragments, not necessarily to obtain a full parse of the sentence.

In spoken utterances, robust semantic parsing is necessary because spontaneous speech may contain disfluencies and ungrammatical sentences. Robust parsing for spoken language understanding has been a topic of great interest in recent years [11, 7, 12]. A survey of the different systems built for the ATIS task can be found in [4], which describes some of the previous work done on robust parsing. Most systems required a robust parser as a backup even if they tried to use a global syntactic parser. Many systems also used a combination of hand-coded rules and machine learning (including probabilities).

We propose using finite state transducers to encode all hand-coded rules and probabilities in a consistent fashion for robust parsing. Finite state machines have been used extensively in natural language processing and speech recognition [6]. Recently, finite state machines have been successfully applied to the problem of parsing using context free grammars [10, 3]. In this paper, we concentrate on the problem of statistical semantic parsing using recursive application of weighted finite state transducers. Semantic decoding is based on the recursive composition of the rule transducer with the word-graph produced from the speech recognizer's output. The recursion is terminated when the process reaches a fixed point, i.e., the output of the composition remains unchanged and no new parses are added. After convergence is reached, the best semantic path is used to obtain the full parse tree by backtracking. The end result is a hierarchical semantic representation that can be easily encoded into a semantic frame or transformed to a flat application frame. Note that the proposed parser can operate on any context-free grammar and can also be applied to syntactic parsing.

2. STATISTICAL MODELS FOR SEMANTIC PARSE TREES

In statistical syntactic parsing, probabilities are assigned to each rule to help disambiguate between the possible parse trees [5]. Typically, rules are assumed to fire independently of each other. More complex models also allow for lexical dependencies across or within rules [1]. In this section, two models are presented for computing the probability of a semantic tree or branch: a "product of rule probability" model, and a concept n-gram model. The first model describes the vertical dependencies in the tree, while the second describes horizontal dependencies.

In Fig. 1(a) an example parse tree is shown for the sentence "D E F." Let this parse tree be denoted by T. The "product of rules" model λ_1 simply computes the probability of the tree as the product of the probabilities of all the rules used to expand each node in the parse tree:

$$P(T|\lambda_1) = P(A)P(A\rightarrow BC|A)P(B\rightarrow D|B)P(C\rightarrow EF|C)$$
$$= P(A)\frac{P(A\rightarrow BC)}{P(A)}\frac{P(B\rightarrow D)}{P(B)}\frac{P(C\rightarrow EF)}{P(C)}, \qquad (1)$$

where each fraction represents the probability of a rule in the tree, and rules are assumed to fire independently of each other. Due to

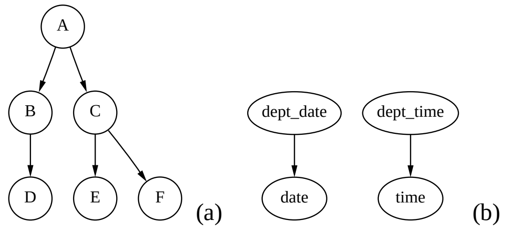


Figure 1: Examples of Semantic Parse Trees

the rule independence assumption, this model does not take into account dependencies across rules.

A concept and word n-gram model λ_2 can be used to model horizontal dependencies within and across rules. This model can be used in addition to λ_1 since it captures dependencies not modeled in the "product of rule" model. Independence is assumed between different levels (horizontal slices) of the tree, and n-grams are used to compute the probability at each tree level. The probability of the tree for the example in Fig. 1(a) using the model λ_2 is estimated as

$$P(T|\lambda_2) = P(A)P(BC)P(DEF) = = P(A)\ P(B)P(C|B)\ P(D)P(E|D)P(F|DE), \quad (2)$$

where $P(A)$, $P(BC)$ and $P(DEF)$ are the probabilities at the root, middle and leaf level of the tree, and a trigram model is used to approximate each probability.

Model λ_1 models the vertical dependencies in the tree. Model λ_2 models horizontal dependencies and provides complementary information when trying to disambiguate between parse trees. For example, consider the case in Fig. 1(b) where a time and a date are specified. Assume that based on rule probabilities (from model λ_1) we are confident that "date" is a departure date (dept_date); "time," however, could be either a departure or an arrival time. In this case, the highly probable dept_date, dept_time bigram can effectively resolve the ambiguity.

The tree probabilities produced by models λ_1 and λ_2 can be assumed to be two independent sources of information and can be combined using exponential weights as follows:

$$P(T|\lambda) \approx P(T|\lambda_1)^{\gamma_1} P(T|\lambda_2)^{\gamma_2}. \quad (3)$$

The exponential weights γ_1 and γ_2 can be computed so that error is minimized on held-out data. The combined statistical model λ is then used for semantic parsing.

2.1. A greedy algorithm

There are cases where there is little training data or a detailed semantic annotation of the corpus is prohibitively expensive. In those cases, a heuristic model can be used to resolve parse ambiguities and compute a crude estimate of $P(ABCDEF|\lambda_1)$. Specifically, rule weights can be hardcoded to values that guarantee greedy behavior, i.e., give preference to rules that transform longer sequences of words and concepts. One simple implementation of this greedy behavior assigns log probabilities to rules that are proportional to the difference in the number of labels before and after the rule is fired, e.g., for the $A \rightarrow BC$ rule, two labels BC get transformed to a single label A, so the rule has a log probability proportional to one. Note that for single to single label rules, e.g., $B \rightarrow D$, some small (non-zero) log probability should be assigned to guarantee that the rules will fire. This simple greedy model can be combined with the n-gram model λ_2 according to Eq. (3) to improve performance.

3. RULE DESIGN AND IMPLEMENTATION

Writing the rules for the semantic grammars is a process that often is both time consuming and requires expert linguistic knowledge. Our goal is to simplify this process by adding rule and n-gram concept probabilities to the semantic decoder so that non-experts can write simple and intuitive local rules without using formal linguistic knowledge. Indeed, probabilistic parsing is more forgiving to ambiguous or general semantic definitions because rule and n-gram probabilities can frequently resolve ambiguity efficiently.

Manual semantic labeling of the corpus can be very expensive and may be inconsistent. In addition, minor rule modifications can significantly change the desired annotation of the corpus. Instead of semantic labeling of the corpus, an iterative process for selecting the semantic rules was followed to speed up the design. First, the terminal concepts such as cities, hotels, airports, were automatically extracted from the database. Spoken form variations were manually added for common terminal values, e.g., 'Los Angeles' or 'L A'. Next, n-gram counts were computed for each terminal concept from the training set. Using a graphical user interface, word and terminal combinations were selected from the n-gram list and mapped to higher level concepts. Then the semantic parser was run using the newly selected rules and n-gram counts were computed anew. This bottom-up rule design process continued until all unambiguous terminals were mapped to higher level concepts. In a third (optional) step, rules for ambiguous terminals were written. Typical rules for departure city, date and time respectively include: "<FROMCITY> $\rightarrow$ FROM <CITY>," "<DATE> $\rightarrow$ <month> <daynumbers>," "<TIME> $\rightarrow$ <TIMESPEC> <TIME>" (where <TIMESPEC> is a relative time specifier, e.g., 'before'). If no training data are available, the rules can be manually coded using regular expressions.

Rules are implemented as a finite state transducer (FST) using the AT&T Bell Labs Finite State Machine (FSM) library [9]. To simplify the identification of rule boundaries, segmentation labels are added. For example, if a rule has fewer labels in the left side than the right side, an epsilon-equivalent segmentation label (denoted as <SGM> in Fig. 2) is added to the rule implementation, e.g., "A $\rightarrow$ B C" is implemented as "A <SGM> $\rightarrow$ B C" (label A is always left-aligned). Weights are assigned to each rule according to (conditional) rule probabilities as discussed in Section 2. The rule transducer R_v is obtained by applying the FSM closure and taking the union among all rule transducers. The n-gram model R_h (see Eq. (2)) is also implemented using the FSM library as a weighted finite state automaton. Finally, the rule transducer R used for parsing is obtained as the composition of the rule transducer and the n-gram automaton, i.e., $R = R_v \circ R_h$. [1]

4. SEMANTIC DECODING

Assume that a rule transducer R has been constructed that maps sequences of words or concepts to a sequence of concepts, and a weighted finite state automaton W that encodes all possible word sequences (within a pruning beam) has been produced by the acoustic decoder. Semantic decoding is then achieved by recursive composition of the wordgraph automaton W with the rule transducer R as follows:

$$P_n = ((W \circ \underbrace{R) \circ R) ...) \circ R}_{\text{n times}}). \quad (4)$$

The recursion terminates when a fixed point P_k is reached and further composition with R adds no new semantic parses , i.e., when P_k and P_{k+1} (after removing the arc costs) are equivalent transducers. In practice, checking the equivalence of the FSTs can be computationally expensive. In addition, if beam pruning is applied to the semantic paths during composition P_k and P_{k+1} might no longer be equivalent. A simple and efficient termination condition is when a fixed point is reached in the best path B_k of P_k, i.e., the best paths of P_k and P_{k+1} are identical[2]. As an example, in Fig. 2(a), P_5 is shown for an ATIS sentence "DALLAS TO HOUSTON AFTER TWELVE OH ONE A M" (for simplicity no weights or input labels are shown).

Once a fixed point is found at iteration k and the best path B_k = bestpath$(P_k)_{\text{proj 2}}$ has been identified, the full semantic parse tree can be obtained recursively as follows:

$$T_{n-1} = (P_{n-1})_{\text{proj 2}} \bigcap (R \circ B_n)_{\text{nocost, proj 1}}, \quad (5)$$

[1]The log probabilities in the FSMs are multiplied by their respective stream weights γ_1 and γ_2 in Eq. (3) before the composition is performed.

[2]Note that this termination condition only makes sense for weighted rule transducers.

where $\bigcap$ denotes the intersection of two acceptors, 'nocost' removes all weights from the FSM, 'proj 1' or 'proj 2' denotes projection of the FST input or ouput to an FSA. T_{n-1} in Eq. (5) contains all paths from iteration $n-1$ that could have produced the best path B_n at iteration n, according to the rules in R. The best path at iteration $n-1$ can be obtained from T_{n-1} as follows:

$$B_{n-1} = \text{bestpath}(T_{n-1}), \tag{6}$$

where 'bestpath' searches for the path with the smallest cost (or highest probability) through the FSM using the Viterbi decoding algorithm. By iteratively applying Equations (5) and (6), the best path is obtained at each iteration as shown in Fig. 2(b). The parse tree shown in Table 1 can be easily derived from the figure using the rule segmentation marks.

5. EVALUATION

One important issue is how to evaluate the parser. A popular set of measures are the PARSEVAL measures which include the measures of *precision* and *recall* [5]. The parse tree for the candidate parse and for the correct parse are first broken down into brackets which denote the non-terminal nodes and the ranges they span.

Precision measures how many brackets in the candidate parse match those in the correct tree, whereas recall measures how many of the brackets in the correct tree are in the candidate parse. Specifically, let m be the number of matching brackets between the candidate and correct parse. *Precision* is the ratio of m to the number of brackets in the candidate parse. *Recall* is the ratio of m to the number of brackets in the correct tree.

As an example, Tables 1 and 2 show the correct and an erroneous candidate parse of the sentence "DALLAS$_0$ TO$_1$ HOUSTON$_2$ AFTER$_3$ TWELVE$_4$ OH$_5$ ONE$_6$ A$_7$ M$_9$" and the corresponding brackets. There are 11 brackets in the correct parse and 9 brackets in the erroneous parse, of which 6 brackets match. Thus the precision is $6/9$ (66.7%) and the recall is $6/11$ (54.5%).

```
DALLAS TO HOUSTON AFTER TWELVE OH ONE A M
{
   :FROMCITY{
      :CITY "DALLAS"
   }
   :TOCITY{
      :CITY "HOUSTON"
   }
   :FROMTIME{
      :TIME{
         :TIMESPEC "AFTER"
         :TIME{
            :HOUR "TWELVE"
            :MINUTE "OH ONE"
            :AMPM "A M"
         }
      }
   }
}
```

Brackets: *FROMCITY(0,0) CITY(0,0) TOCITY(2,2) CITY(2,2)* FROMTIME(3,8) TIME(3,8) *TIMESPEC(3,3)* TIME(4,6) HOUR(4,4) MINUTE(5,6) *AMPM(7,8)*

Table 1: Correct parse tree.

Besides measuring the precision and recall, it is also helpful to classify the errors as substitution, segmentation, deletion, or insertion errors. We define a substitution error as one where, after all the matched brackets have been taken out, there exists a pair of correct and candidate brackets with the exact same range but different non-terminal labels. In the example, we did not have any substitution error; if there had been a bracket TIME(4,6) in the correct and FLIGHTNO(4,6) in the candidate parse, this would have qualified as a substitution error.

```
DALLAS TO HOUSTON AFTER TWELVE OH ONE A M
{
   :FROMCITY{
      :CITY "DALLAS"
   }
   :TOCITY{
      :CITY "HOUSTON"
   }
   :TIMESPEC "AFTER"
   :FLIGHTNO "TWELVE OH"
   :TIME{
      :HOUR "ONE"
      :AMPM "A M"
   }
}
```

Brackets: *FROMCITY(0,0) CITY(0,0) TOCITY(2,2) CITY(2,2) TIMESPEC(3,3)* FLIGHTNO(4,5) TIME(6,8) HOUR(6,6) *AMPM(7,8)*

Table 2: Parse tree produced when "<MINUTE> → <digit> <digit>" is deleted from the parser's rules.

A segmentation error is defined to be one where the candidate and correct parses have the same non-terminal label with different but overlapping ranges. In the example, there is one segmentation error: TIME(3,8) → TIME(6,8). After the correct matches, substitution errors, and segmentation errors are taken out, what remain are the insertion and deletion errors. In the example, there are 4 deletion errors: FROMTIME(3,8), TIME(4,6), HOUR(4,4), and MINUTE(5,6) and 2 insertion errors: FLIGHTNO(4,5) and HOUR(6,6). Each of these types of errors can be expressed as an error rate, a percentage of the total number of brackets in the correct parses for the whole test set.

6. EXPERIMENTAL RESULTS

The parse results for the 1993 ATIS category A test set are summarized in Table 3. A bigram model and the simple greedy algorithm introduced in Section 2.1 were used to train the rule transducer. Bigrams were trained from the parsed ATIS training corpus (four iterations, unsupervised training). The ATIS category A sentences are independent utterances that did not rely on context information from previous utterances. There are a total of 389 sentences in the 1993 test set, and the correct transcribed sentences were used as inputs to the parser. As shown in Table 3, the precision and recall rates were over 95%. These results were based on text only and not the output of the recognizer.

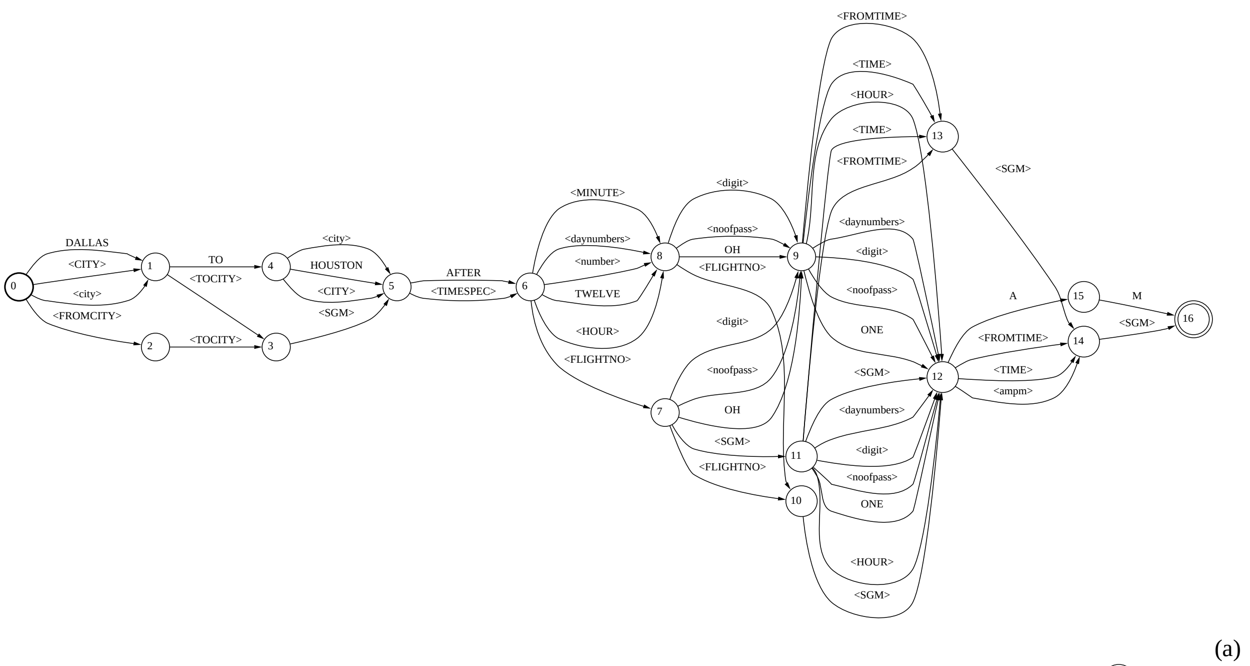
The error rates shown in the table were computed as a fraction of the total number of brackets (2475) in the correct parses for the entire test set. Specifically, there were a total of 7 substitution, 12 segmentation, 95 insertion, and 84 deletion errors. The parser has also been used successfully in the Bell Labs travel reservation system [8].

7. CONCLUSIONS

We have proposed a statistical semantic parser that combines the advantages of the generative and statistical approaches to speech

Precision	95.4%
Recall	95.8%
Substitution Error	0.28%
Segmentation Error	0.48%
Insertion Error	3.84%
Deletion Error	3.39%

Table 3: Parser results for the ATIS 1993 category A sentences.



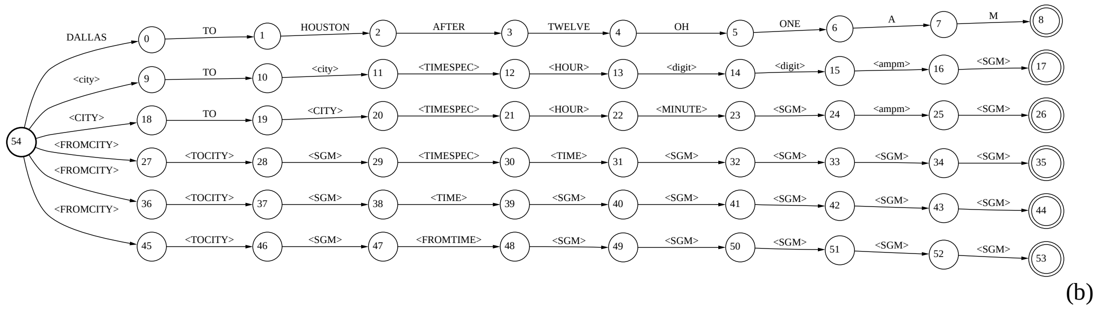


Figure 2: (a) A costless projection of the resulting finite state transducer when a fixed point is reached; recursive composition is performed on the sentence "DALLAS TO HOUSTON AFTER TWELVE OH ONE A M", and (b) Backtracking from the best path: the costless union of the best path at each iteration is shown for the same sentence.

understanding, i.e., the rich hierarchical representation of a parse tree with the power and robustness of statistical modeling. Rules from all knowledge sources and probabilities estimated from training data can be encoded in a consistent framework with finite state transducers. By using finite state machine technology for rule design and decoding, the understanding module has improved efficiency and extensibility, and is easy to integrate with other modules. For example, acoustic, language, and understanding scores can also be easily combined in a word- and concept-graph. Guidelines for designing and implementing rules using finite state transducers were described. The parser was evaluated on a travel reservation application, achieving precision and recall rates of over 95%.

8. REFERENCES

[1] Collins, M. J. *Head-driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, Philadelphia, 1999.

[2] Jurafsky, D., and Martin, J. H. *Speech and Language Processing*. Prentice Hall, Upper Saddle River, New Jersey, 2000.

[3] Kaiser, E. C., Johnston, M., and Heeman, P. A. "PROFER: Predictive, Robust Finite-State Parsing for Spoken Language," in *Proc. ICASSP'2000* (Istanbul, Turkey), Jun. 2000.

[4] Kuhn, R. and De Mori, R., "Sentence Interpretation," in De Mori, R., ed., *Spoken Dialogues with Computers*, Academic Press, New York, 1998.

[5] Manning, C. D., and Schutze, H. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, Massachusetts, 1999.

[6] Pereira, F. C. N. and Riley, M. D., "Speech Recognition by Composition of Weighted Finite State Automata," in E. Roche and Y. Schabes, ed., *Finite-State Language Processing*, MIT Press, Cambridge, Massachusetts, 1997.

[7] Pieraccini, R. and Levin, E., "A spontaneous-speech understanding system for database query applications ," in *ESCA Workshop on Spoken Dialogue Systems - Theories and Applications*, 1995.

[8] Potamianos, A., Ammicht, E. and Kuo H-K. J. "Dialogue management in the Bell Labs communicator system," in *Proc. ICSLP* (Beijing, China), Oct. 2000.

[9] Riley, M., Pereira, F., "Weighted-finite automata tools with applications to speech and language processing," Technical report, AT&T Bell Laboratories, 1995. Most current documentation at http://www.research.att.com/sw/tools/fsm

[10] Roche, E., "Parsing with Finite-State Transducers," in E. Roche and Y. Schabes, ed., *Finite-State Language Processing*, MIT Press, Cambridge, Massachusetts, 1997.

[11] Seneff, S., "The Use of Linguistic Hierarchies in Speech Understanding," in *Proc. ICSLP'98*, (Sydney, Australia), Nov. 1998.

[12] Wang, Y., "A Robust Parser for Spoken Language Understanding," *Proc. Eurospeech'99*, (Budapest, Hungary), Sep. 1999.