Robust, Finite-State Parsing for Spoken Language Understanding

Edward C. Kaiser
Center for Spoken Language Understanding
Oregon Graduate Institute
PO Box 91000 Portland OR 97291
kaiser@cse.ogi.edu

Abstract

Human understanding of spoken language appears to integrate the use of contextual expectations with acoustic level perception in a tightly-coupled, sequential fashion. Yet computer speech understanding systems typically pass the transcript produced by a speech recognizer into a natural language parser with no integration of acoustic and grammatical constraints. One reason for this is the complexity of implementing that integration. To address this issue we have created a robust, semantic parser as a single finite-state machine (FSM). As such, its run-time action is less complex than other robust parsers that are based on either chart or generalized left-right (GLR) architectures. Therefore, we believe it is ultimately more amenable to direct integration with a speech decoder.

1 Introduction

An important goal in speech processing is to extract meaningful information: in this, the task is *understanding* rather than *transcription*. For extracting meaning from spontaneous speech full coverage grammars tend to be too brittle. In the 1992 DARPA ATIS task competition, CMU's Phoenix parser was the best scoring system (Issar and Ward, 1993). Phoenix operates in a loosely-coupled architecture on the 1-best transcript produced by the recognizer. Conceptually it is a semantic *case-frame* parser (Hayes et al., 1986). As such, it allows *slots* within a particular *case-frame* to be filled in any order, and allows out-of-grammar words between *slots* to be skipped over. Thus it can return partial parses — as *frames* in which only some of the available *slots* have been filled.

Humans appear to perform robust understanding in a tightly-coupled fashion. They build incremental, partial analyses of an utterance as it is being spoken, in a way that helps them to meaningfully interpret the acoustic evidence. To move toward machine understanding systems that tightly-couple acoustic features and structural knowledge, researchers like Pereira and Wright (1997) have argued for the use of finite-state acceptors (FSAs) as an efficient means of integrating structural knowledge into the recognition process for limited domain tasks.

We have constructed a parser for spontaneous speech that is at once both robust and finite-state. It is called PROFER, for Predictive, RObust, Finite-state parsER. Currently PROFER accepts a transcript as input. We are modifying it to accept a word-graph as input. Our aim is to incorporate PROFER directly into a recognizer.

For example, using a grammar that defines sequences of numbers (each of which is less than ten thousand and greater than ninety-nine and contains the word "hundred"), inputs like the following string can be robustly parsed by PROFER:

```
Input:
  first I've got twenty ahhh thirty yaaaaaa
  thirty ohh wait no twenty twenty nine
  hundred two errr three ahhh four and then
  two hundred ninety uhhhhh let me be sure
  here yaaaa ninety seven and last is five
  oh seven uhhh I mean six

Parse-tree:
  [fsType:number_type,
    hundred_fs:
      [decade:[twenty,nine],hundred,four],
    hundred_fs:
      [two,hundred,decade:[ninety,seven]],
    hundred_fs:
      [five,hundred,six]]
```

There are two characteristically "robust" actions that are illustrated by this example.

- For each "slot" (i.e., "_fs" element) filled in the parse-tree's case-frame structure, there were several words both before and after the required word, *hundred*, that had to be skipped-over. This aspect of robust parsing is akin to *phrase-spotting.*
- In mapping the words, "five oh seven uhhh I mean six," the parser had to choose a *later-in-the-input* parse (i.e., "[five, hundred, six]") over a heuristically equivalent *earlier-in-the-input* parse (i.e., "[five, hundred, seven]"). This aspect of robust parsing is akin to *dynamic programming* (i.e., finding all possible start and end points for all possible patterns and choosing the best).

2 Robust Finite-state Parsing

CMU's Phoenix system is implemented as a recursive transition network (RTN). This is similar to Abney's system of finite-state-cascades (1996). Both parsers have a "stratal" system of levels. Both are robust in the sense of skipping over out-of-grammar areas, and building up structural islands of certainty. And both can be fairly described as run-time chart-parsers. However, Abney's system inserts bracketing and tagging information by means of cascaded transducers, whereas Phoenix accomplishes the same thing by storing state information in the chart edges themselves — thus using the chart edges like tokens. PROFER is similar to Phoenix in this regard.

Phoenix performs a *depth-first* search over its textual input, while Abney's "chunking" and "attaching" parsers perform *best-first* searches (1991). However, the demands of a tightly-coupled, real-time system argue for a *breadth-first* search-strategy, which in turn argues for the use of a finite-state parser, as an efficient means of supporting such a search strategy. PROFER is a strictly sequential, *breadth-first* parser.

PROFER uses a regular grammar formalism for defining the patterns that it will parse from the input, as illustrated in Figures 1 and 2.

Net name tags correspond to bracketed (i.e., "tagged") elements in the output. Aside from

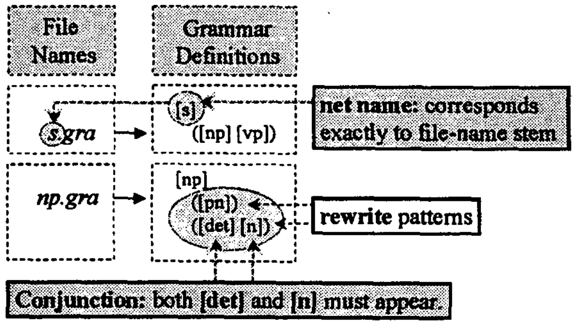

File Names
Grammar Definitions
s.gra
[s]
([np] [vp])
net name: corresponds exactly to file-name stem
np.gra
[np]
([pn])
([det] [n])
rewrite patterns
Conjunction: both [det] and [n] must appear.

Figure 1: Formalism

net names, a grammar definition can also contain non-terminal *rewrite* names and *terminals. Terminals* are directly matched against input. Non-terminal *rewrite* names group together several *rewrite patterns* (see Figure 2), just as *net* names can be used to do, but *rewrite* names do not appear in the output.

Each individual *rewrite pattern* defines a "conjunction" of particular terms or subpatterns that can be mapped from the input into the non-terminal at the head of the pattern block, as illustrated in (Figure 1). Whereas, the list of patterns within a block represents a "disjunction" (Figure 2).

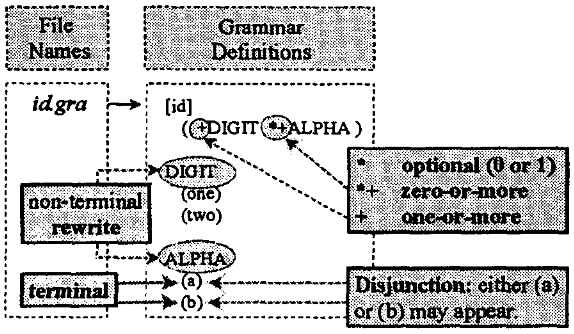
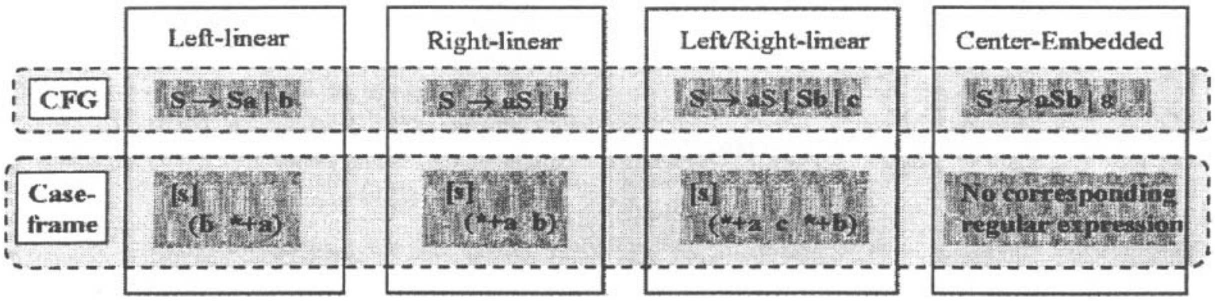


Figure 2: Formalism

Since not all Context-Free Grammar (CFG) expressions can be translated into regular expressions, as illustrated in Figure 3, some restrictions are necessary to rule out the possibility of "center-embedding" (see the right-most block in Figure 3). The restriction is that neither a *net* name nor a *rewrite* name can appear in one of its own descendant blocks of *rewrite patterns.*

Even with this restriction it is still possible to define regular grammars that allow for self-



	Left-linear	Right-linear	Left/Right-linear	Center-Embedded
CFG	$S \rightarrow Sa \mid b$	$S \rightarrow aS \mid b$	$S \rightarrow aS \mid Sb \mid c$	$S \rightarrow aSb \mid \varepsilon$
Case-frame	[s] (b *+a)	[s] (*+a b)	[s] (*+a c *+b)	No corresponding regular expression

Figure 3: Context-Free translations to case-frame style regular expressions.

embedding to any *finite* depth, by copying the *net* or *rewrite* definition and giving it a unique name for each level of self-embedding desired. For example, both grammars illustrated in Figure 4 can robustly parse inputs that contain some number of **a**'s followed by a matching number of **b**'s up to the level of embedding defined, which in both of these cases is four deep.

```
EXAMPLE: nets                EXAMPLE: rewrites
[se]                         [ser]
  (a [se_one] b)               (a SE_ONE b)
  (a b)                        (a b)
[se_one]                     SE_ONE
  (a [se_two] b)               (a SE_TWO b)
  (a b)                        (a b)
[se_two]                     SE_TWO
  (a [se_three] b)             (a SE_THREE b)
  (a b)                        (a b)
[se_three]                   SE_THREE
  (a b)                        (a b)

INPUT:                       INPUT:
  a c a b d e b                a c a b d e b
PARSE:                       PARSE:
  se:[a,se_one:[a,b],b]        ser:[a,a,b,b]
```

Figure 4: Finite self-embedding.

3 The Power of Regular Grammars

Tomita (1986) has argued that context-free grammars (CFGs) are over-powered for natural language. Chart parsers are designed to deal with the worst case of very-deep or infinite self-embedding allowed by CFGs. However, in natural language this worst case does not occur. Thus, broad coverage Generalized Left-Right (GLR) parsers based on Tomita's algorithm, which ignore the worst case scenario, are in practice more efficient and faster than comparable chart-parsers (Briscoe and Carroll, 1993).

PROFER explicitly disallows the worst case of center-self-embedding that Tomita's GLR design allows — but ignores. Aside from infinite center-self-embedding, a regular grammar formalism like PROFER's can be used to define every pattern in natural language definable by a GLR parser.

4 The Compilation Process

The following small grammar will serve as the basis for a high-level description of the compilation process.

```
[s]
  (n [v] n)
  (p [v] p)
[v]
  (v)
```

In Kaiser et al. (1999) the relationship between PROFER's compilation process and that of both Pereira and Wright's (1997) FSAs and CMU's Phoenix system has been described. Here we wish to describe what happens during PROFER's compilation stage in terms of the Left-Right parsing notions of *item-set formation* and *reduction*.

As compilation begins the FSM always starts at state `0:0` (i.e., net 0, start state 0) and traverses an arc labeled by the top-level *net* name to the `0:1` state (i.e., net 0, final state 1), as illustrated in Figure 5. This initial arc is then rewritten by each of its **rewrite patterns** (Figure 5).

As each new *net* within the grammar description is encountered it receives a unique *net*-ID number, the compilation descends recursively into that new sub-*net* (Figure 5), reads in its

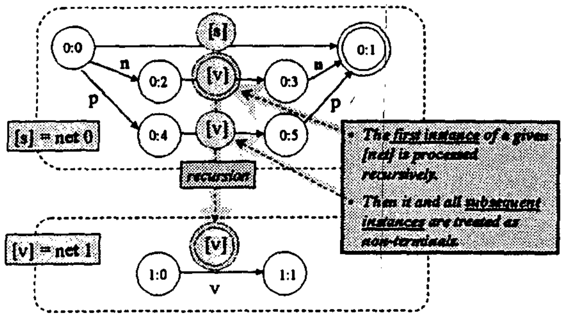

[s]
[v]
[s] = net 0
recursion
[v] = net 1
• The first instance of a given [net] is processed recursively.
• Then it and all subsequent instances are treated as non-terminals.

Figure 5: Definition expansion.

grammar description file, and compiles it. Since *rewrite* names are unique only within the *net* in which they appear, they can be processed iteratively during compilation, whereas *net* names must be processed recursively within the scope of the entire grammar's definition to allow for re-use.

As each element within a **rewrite pattern** is encountered a structure describing its exact context is filled in. All *terminals* that appear in the same context are grouped together as a "context-group" or simply "context." So arcs in the final FSM are traversed by "contexts" not *terminals*.

When a *net* name itself traverses an arc it is glued into place contextually with ε arcs (i.e., NULL arcs) (Figure 6). Since *net* names, like any other pattern element, are wrapped inside of a context structure before being situated in the FSM, the same *net* name can be re-used inside of many different contexts, as in Figure 6.

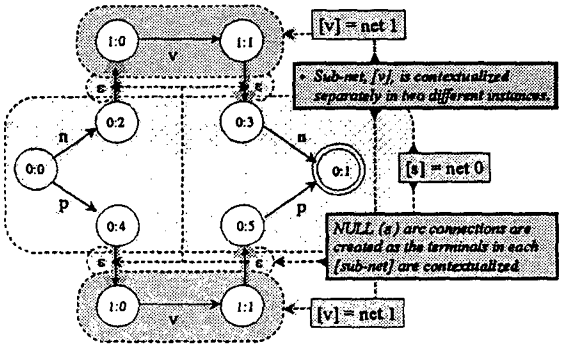

[v] = net 1
• Sub-net, [v], is contextualized separately in two different instances.
[s] = net 0
NULL (ε) arc connections are created as the terminals in each [sub-net] are contextualized
[v] = net 1

Figure 6: Contextualizing sub-nets.

As the end of each **net** definition file is reached, all of its NULL arcs are removed. Each initial state of a sub-net is assumed into its parent state — which is equivalent to *item-set formation* in that parent state (Figure 7 left-side). Each final state of a sub-net is erased, and its incoming arcs are rerouted to its terminal parent's state, thus performing a *reduction* (Figure 7 right-side).

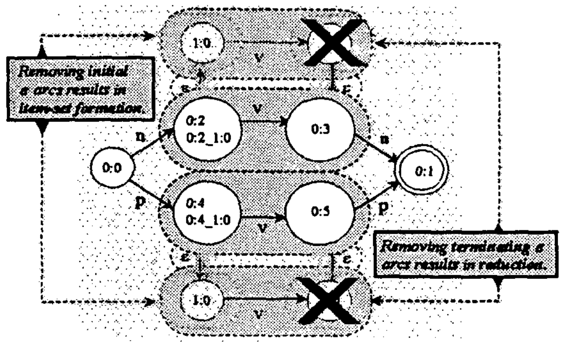

Removing initial ε arcs results in item-set formation.
Removing terminating ε arcs results in reduction.

Figure 7: Removing NULL arcs.

5 The Parsing Process

At run-time, the parse proceeds in a strictly *breadth-first* manner (Figure 8,(Kaiser et al., 1999)). Each destination state within a parse is named by a hash-table key string composed of a sequence of "net:state" combinations that uniquely identify the location of that state within the FSM (see Figure 8). These "net:state" names effectively represent a snapshot of the stack-configuration that would be seen in a parallel GLR parser.

PROFER deals with ambiguity by "splitting" the branches of its graph-structured stack (as is done in a Generalized Left-Right parser (Tomita, 1986)). Each node within the graph-structured stack holds a "token" that records the information needed to build a bracketed parse-tree for any given branch.

When partial-paths converge on the same state within the FSM they are scored heuristically, and all but the set of highest scoring partial paths are pruned away. Currently the heuristics favor interpretations that cover the most input with the fewest *slots*. Command line parameters can be used to refine the heuristics, so that certain kinds of structures be either minimized or maximized over the parse.

Robustness within this scheme is achieved by allowing multiple paths to be propagated in parallel across the input space. And as each such

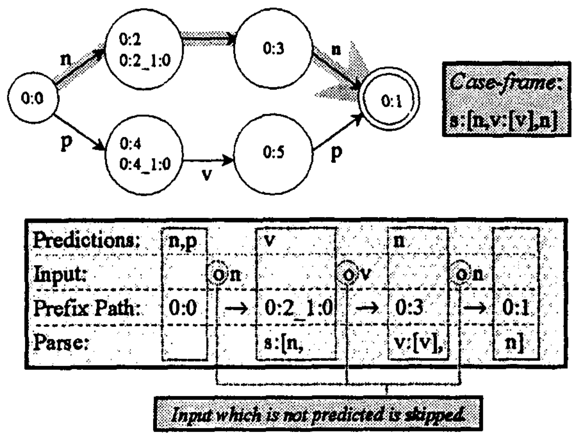


Figure 8: The parsing process.

partial-path is extended, it is allowed to skip-over terms in the input that are not licensed by the grammar. This allows all possible start and end times of all possible patterns to be considered.

6 Discussion

Many researchers have looked at ways to improve corpus-based language modeling techniques. One way is to parse the training set with a structural parser, build statistical models of the occurrence of structural elements, and then use these statistics to build or augment an n-gram language model.

Gillet and Ward (1998) have reported reductions in perplexity using a stochastic context-free grammar (SCFG) defining both simple semantic "classes" like dates and times, and degenerate classes for each individual vocabulary word. Thus, in building up class statistics over a corpus parsed with their grammar they are able to capture both the traditional n-gram word sequences plus statistics about semantic class sequences.

Briscoe has pointed out that using stochastic context-free grammars (SCFGs) as the basis for language modeling, "...means that information about the probability of a rule applying at a particular point in a parse derivation is lost" (1993). For this reason Briscoe developed a GLR parser as a more "natural way to obtain a finite-state representation ..." on which the statistics of individual "reduce" actions could be determined. Since PROFER's state names effectively represent the stack-configurations of a parallel GLR parser it also offers the ability to perform the full-context statistical parsing that Briscoe has called for.

Chelba and Jelinek (1999) use a structural language model (SLM) to incorporate the longer-range structural knowledge represented in statistics about sequences of phrase-head-word/non-terminal-tag elements exposed by a tree-adjoining grammar. Unlike SCFGs their statistics are specific to the structural context in which head-words occur. They have shown both reduced perplexity and improved word error rate (WER) over a conventional tri-gram system.

One can also reduce complexity and improve word-error rates by widening the speech recognition problem to include modeling not only the word sequence, but the word/part-of-speech (POS) sequence. Heeman and Allen (1997) has shown that doing so also aids in identifying speech repairs and intonational boundaries in spontaneous speech.

However, all of these approaches rely on corpus-based language modeling, which is a large and expensive task. In many practical uses of spoken language technology, like using simple structured dialogues for class room instruction (as can be done with the CSLU toolkit (Sutton et al., 1998)), corpus-based language modeling may not be a practical possibility.

In structured dialogues one approach can be to completely constrain recognition by the known expectations at a given state. Indeed, the CSLU toolkit provides a generic recognizer, which accepts a set of vocabulary and word sequences defined by a regular grammar on a per-state basis. Within this framework the task of a recognizer is to choose the best phonetic path through the finite-state machine defined by the regular grammar. Out-of-vocabulary words are accounted for by a general purpose "garbage" phoneme model (Schalkwyk et al., 1996).

We experimented with using PROFER in the same way; however, our initial attempts to do so did not work well. The amount of information carried in PROFER's token's (to allow for bracketing and heuristic scoring of the semantic hypotheses) requires structures that are an order of magnitude larger than the tokens in a typical acoustic recognizer. When these large tokens are applied at the phonetic-level so many are needed that a memory space explosion occurs. This suggests to us that there must be two levels of tokens: small, quickly manipulated tokens at the acoustic level (i.e., lexical level), and larger, less-frequently used tokens at the structural level (i.e., syntactic, semantic, pragmatic level).

7 Future Work

In the MINDS system Young et al. (1989) reported reduced word error rates and large reductions in perplexity by using a dialogue structure that could track the active goals, topics and user knowledge possible in a given dialogue state, and use that knowledge to dynamically create a semantic case-frame network, whose transitions could in turn be used to constrain the word sequences allowed by the recognizer.

Our research aim is to maximize the effectiveness of this approach. Therefore, we hope to:

- expand the scope of PROFER's structural definitions to include not only word patterns, but intonation and stress patterns as well, and
- consider how build to general language models that complement the use of the categorial constraints PROFER can impose (i.e., syllable-level modeling, intonational boundary modeling, or speech repair modeling).

Our immediate efforts are focused on considering how to modify PROFER to accept a word-graph as input — at first as part of a loosely--coupled system, and then later as part of an integrated system in which the elements of the word-graph are evaluated against the structural constraints as they are created.

8 Conclusion

We have presented our finite-state, robust parser, PROFER, described some of its workings, and discussed the advantages it may offer for moving towards a tight integration of robust natural language processing with a speech decoder — those advantages being: its efficiency as an FSM and the possibility that it may provide a useful level of constraint to a recognizer independent of a large, task-specific language model.

9 Acknowledgements

The author was funded by the Intel Research Council, the NSF (Grant No. 9354959), and the CSLU member consortium. We also wish to thank Peter Heeman and Michael Johnston for valuable discussions and support.

References

S. Abney. 1991. Parsing by chunks. In R. Berwick, S. Abney, and C. Tenny, editors, *Principle-Based Parsing*. Kluwer Academic Publishers.

S. Abney. 1996. Partial parsing via finite-state cascades. In *Proceedings of the ESSLLI '96 Robust Parsing Workshop*.

T. Briscoe and J. Carroll. 1993. Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1):25–59.

C. Chelba and F. Jelinek. 1999. Recognition performance of a structured language model. In *The Proceedings of Eurospeech '99 (to appear)*, September.

J. Gillet and W. Ward. 1998. A language model combining trigrams and stochastic context-free grammars. In *Proceedings of ICSLP '98*, volume 6, pgs 2319–2322.

P. J. Hayes, A. G. Hauptmann, J. G. Carbonell, and M. Tomita. 1986. Parsing spoken language: a semantic caseframe approach. In *11th International Conference on Computational Linguistics, Proceedings of Coling '86*, pages 587–592.

P. A. Heeman and J. F. Allen. 1997. Intonational boundaries, speech repairs, and discourse markers: Modeling spoken dialog. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pages 254–261.

S. Issar and W. Ward. 1993. Cmu's robust spoken language understanding system. In *Eurospeech '93*, pages 2147–2150.

E. Kaiser, M. Johnston, and P. Heeman. 1999. Profer: Predictive, robust finite-state parsing for spoken language. In *Proceedings of ICASSP '99*.

F. C. N. Pereira and R. N. Wright. 1997. Finite-state approximations of phrase-structure grammars. In Emmanuel Roche and Yves Schabes, editors, *Finite-State Language Processing*, pages 149–173. The MIT Press.

J. Schalkwyk, L. D. Colton, and M. Fanty. 1996. The CSLU-sh toolkit for automatic speech recognition: Technical report no. CSLU-011-96, August.

S. Sutton, R. Cole, J. de Villiers, J. Schalkwyk, P. Vermeulen, M. Macon, Y. Yan, E. Kaiser, B. Rundle, K. Shobaki, P. Hosom, A. Kain, J. Wouters, M. Massaro, and M. Cohen. 1998. Universal speech tools: the cslu toolkit". In *Proceedings of ICSLP '98*, pages 3221–3224, Nov..

M. Tomita. 1986. *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems.* Kluwer Academic Publishers.

S. R. Young, A. G. Hauptmann, W. H. Ward, E. T. Smith, and P. Werner. 1989. High level knowledge sources in usable speech recognition systems. *Communications of the ACM*, 32(2):183–194, February.