Finite-state phrase parsing by rule sequences

Marc Vilain and David Day
The MITRE Corporation
202 Burlington Rd.
Bedford, MA 01720 USA
mbv@mitre.org, day@mitre.org

Abstract

We present a novel approach to parsing phrase grammars based on Eric Brill's notion of rule sequences. The basic framework we describe has somewhat less power than a finite-state machine, and yet achieves high accuracy on standard phrase parsing tasks. The rule language is simple, which makes it easy to write rules. Further, this simplicity enables the automatic acquisition of phrase-parsing rules through an error-reduction strategy.

This paper explores an approach to syntactic analysis that is unconventional in several respects. To begin with, we are concerned not so much with the traditional goal of analyzing the comprehensive structure of complete sentences, as much as with assigning partial structure to parts of sentences. The fragment of interest here is demonstrably a subset of the regular sets, and while these languages are traditionally analyzed with finite-state automata, our approach relies instead on the rule sequence architecture defined by Eric Brill.

Why restrict ourselves to the finite-state case? Some linguistic phenomena are easier to model with regular sets than context-free grammars. Proper names are a case in point, since their syntactic distribution partially overlaps that of noun phrases in general; as this overlap is only partial, name analysis within a full context-free grammar is cumbersome, and some approaches have taken to include finite-state name parsers as a front-end to a principal context-free parsing stage (Jacobs *et al.* 1991). Proper names are of further interest, since their identification is independently motivated as valuable to both information retrieval and extraction (Sundheim 1996). Further, several promising recent approaches to information extraction rely on little more than finite-state machines to perform the entire extraction analysis (Appelt *et al.* 1993, Grishman 1995).

Why approach this problem with rule sequences? In this paper we make the case that rule sequences succeed at this task through their simplicity and speed. Most important, they support mixed-mode acquisition: the rules are both easy for an engineer to write and easy to learn automatically.

Rule sequences

As part of our work in information extraction, we have been extensively exploring the use of rule sequences. Our information extraction prototype, *Alembic*, is in fact based on a pipeline of rule sequence processors that run the gamut from part-of-speech tagging, to phrase identification, to sentence parsing, to inference (Aberdeeen *et al.* 1995). In each case, the underlying method is identical. Processing takes place by sequentially relabeling the corpus under consideration. Each sequential step is driven by a rule that attempts to patch residual errors left in place in the preceding steps. The patching process as a whole is itself preceded by an initial labeling phase that provides an approximate labeling as a starting point for rule application.

This patching architecture, illustrated in Fig. 1, was codified by Eric Brill, who first exploited it for part-of-speech tagging (Brill 1993). In the part-of-speech application, initial labeling is provided by lexicon lookup: lexemes are initially tagged with the most common part of speech assigned to them in a training corpus. This initial labeling is refined by two sets of transformations. Morphological transformations relabel the initial (default) tagging of those words that failed to be found in the lexicon. The morphological rules are followed by contextual transformations: these rules inspect lexical context to relabel lexemes that are ambiguous with respect to part-of-speech. In effect, the morphological transformations patch errors that were due to gaps in the lexicon, and the contextual rules patch errors that were due to the initial assignment of a lexeme's most common tag.

Phrase identification: some examples

Sequencing, patching, and simplicity, the hallmarks of Brill's part-of-speech tagger, are also characteristic of our phrase parser. In our approach, phrases are initially built around word sequences that meet certain lexical or part-of-speech criteria. The sequenced phrase-finding rules then grow the boundaries of phrases or set their label, according to a repertory of simple lexical and contextual tests. For example, the following rule assigns a label of ORG to an unlabeled phrase just in case the phrase is ended by the word "Inc."

```
(def-phraser
  label          NONE          ; phrase is currently
                               ; unlabelled
  right-wd-1     lexeme "Inc." ; rightmost word in the
                               ; phrase is "Inc."
  label-action   ORG)          ; change the phrase's label,
                               ; but not its boundaries
```

Now, consider the following partially labelled string:

<none>Donald F. DeScenza</none>, analyst with
<none>Nomura Securities Inc.</none>

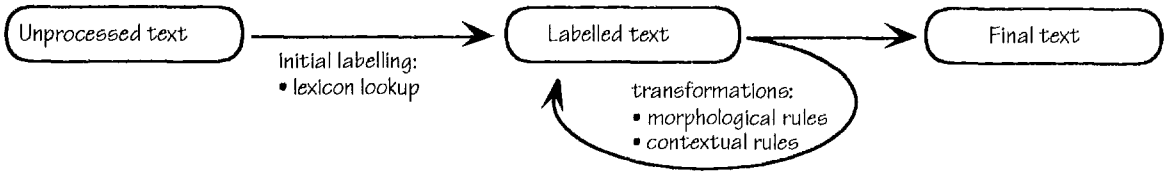


Figure 1: Brill's rule sequence architecture as applied to part-of-speech tagging.

The SGML markup delimits phrases whose boundaries were identified by the initial phrase-finding pass. Of these phrases, the second successfully triggers the example rule, yielding the following relabeled string.

<none>Donald F. DeScenza</none>, analyst with <org>Nomura Securities Inc.</org>

The rule, which seems both as obvious as walking and as fool-proof comes from the name-finding processor we developed for our participation in the 6th Message Understanding Conference (MUC-6). As it turns out, though, the rule is in fact not error-proof, and causes both errors of omission (*i.e.* recall errors) and commission (*i.e.* precision errors). Consider the case of "Volkswagen of America Inc." Because the initial phrase labeling is only approximate, the string is broken into two sub-phrases separated by "of".

<none>Volkswagen</none> of <none>America Inc.</none>

The example rule designates the partial phrase "America Inc." as an ORG, a precision error because of its partiality, and fails to produce an ORG label spanning the entire string (a recall error).

<none>Volkswagen</none> of <org>America Inc.</org>

This problem is patched by a subsequent name-finding rule, namely the following.

```
(def-phraser
  label          ORG              ; this is an organization
  left-wd-1      test country?    ; is the leftmost lexeme
                                  ; in the phrase on a list
                                  ; of country words?
  left-ctxt-1    lexeme "of"      ; to the left of the
                                  ; phrase is the word "of"
  left-ctxt-2    phrase NONE      ; to the left of that is an
                                  ; unlabelled phrase
  bounds-action  MERGE            ; merge the entire left
  labbel-action  ORG)             ; context into the ORG,
                                  ; phrase and all
```

The first two clauses of the rule are antecedents that look for phrases such as "America Inc." The next two clauses are further antecedents that look to the left of the phrase for contextual patterns of form

"<none>...</none> of".

The final two clauses incorporate the left context wholesale into the triggering phrase, yielding:

<org>Volkswagen of America Inc.</org>

This rule effectively patches the errors caused by its predecessor in the rule sequence, and simultaneously eliminates both a recall and a precision error.

The phrase finder

With these examples as background, we may now turn our attention to the technical details of the phrase finding process. As noted above, this process occurs in two main steps, an initial labeling pass followed by the application of a rule sequence.

Initial phrase labeling

The initial labeling process seeds the phrase-finder with candidate phrases. These candidate phrases need not be any more than approximations. In particular, it is not necessary for these candidates to have wholly accurate boundaries, as their left and right edges can be adjusted later by means of patching rules. It is also not necessary for these candidates to be unfragmented, as fragments can be reassembled later, just as with "Volkswagen of America Inc." Further, applications that require multiple types of phrase labels, need not choose such a label during the initial phrase-finding pass.

What is important is that the initial phrase identification find the *cores* of phrases reliably, even if *complete* phrases are not identified. That is, it must partially align some kind of candidate phrase κ for every phrase ϕ that is actually present in the input. Extending a concept from information retrieval, this amounts to maximizing what we might call initial recall, *i.e.*,

$R_I = |\Phi_I| / |\Phi|$,

where Φ is the set of actual phrases in a test set, K is the set of candidate phrases generated by the initial phrasing passs, and Φ_I is the set of those $\phi \in \Phi$ that are partially aligned with some $\kappa \in K$.

The general strategy we have adopted for finding initial phrase seeds is to look for either runs of lexemes in a fixed word list or runs of lexemes that have been tagged a certain way by our part-of-speech tagger. Different instantiations of this general strategy for initial phrase labeling naturally arise for different phrase-finding tasks. For example, on the classic "proper names" task in mixed-case text, we have achieved good results starting from runs of lexemes tagged with NNP or NNPS, the Penn Treebank proper noun tags. This strategy achieves the desired high initial recall R_I, as these tags are well-correlated with *bona fide* proper names and are reliably produced in mixed-case text by our part-of-speech tagger. This strategy does not yield quite as good initial precision (*i.e.*, it yields false positives) for a number of reasons, such as the fragmentation problems noted above, *e.g.*,

Volkswagen/NNP of/IN America/NNP Inc./NNP

Once again, though, these initial precision errors are readily addressed by patching rules.

Clause type	Syntax	Definition
Contextual tests	left-ctxt-1, left-ctxt-2	Test one place (resp. two places) to the left of the phrase
	right-ctxt-1, right-ctxt-2	Test one place (resp. two places) to the right of the phrase
Phrase-internal	left-wd-1, left-wd-2	Test first (resp. second) word of phrase
tests	right-wd-1, right-wd-2	Test last (resp. next-to-last) word of phrase
	wd-any	Test each word of phrase in succession. Succeeds if any word in the phrase passes the test.
	wd-span	Test entire string spanned by phrase
Label test	label	Test phrase's label
Actions	label-action	Sets the label of the phrase
	bounds-action	Modify the phrase's left or right boundaries

Table 1: Repertory of unary rule clauses.

Phrase-finding rules

A phrase-finding rule in our framework is made up of several clauses. The core of the rule consists of clauses that test the lexical context around a candidate phrase κ or that test lexemes spanned by κ. The repertory of these test loci is given in Table 1. At any given locus, a test may either search for a particular lexeme, match a lexeme against a closed word list, match a part of speech, or match a phrase of a given type. Most rules also test the label of the candidate phrase κ.

The unary contextual tests in the table may also be combined to form binary or ternary tests. For example, combining LEFT-CTXT-1 and LEFT-CTXT-2 clauses yields a rule that tests for the left bigram context. This was done in the ORG defragmentation rule described earlier.

A rule also contains at least one action clause, either a clause that sets the label of the phrase, or one that modifies the boundaries of the phrase. Finally, some rule actions actually introduce new phrases that embed the candidate and its test context; this allows one to build non-recursive parse trees.

Phrase rule interpreter

The phrase rule interpreter implements the rule language in a straightforward way. Given a document to be analyzed, it proceeds through a rule sequence one rule r at a time, and attempts to apply r to every phrase in every sentence in the document. The interpreter first attempts to match the test label of r to the label of the candidate phrase. If this test succeeds, then the interpreter attempts to satisfy the rule's contextual tests in the context of the candidate. If these test succeed, then the rule's bounds and label actions are executed.

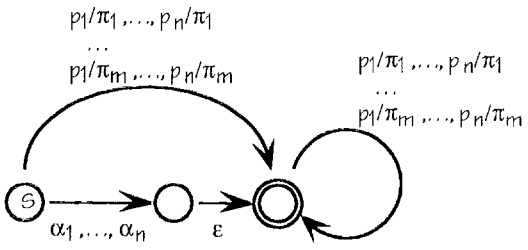
Beyond this, the only real complexity arises with phrase-finding tasks that require one to maintain a temporary lexicon. The clearest such example is proper name identification. Indeed, short name forms (*e.g.*, "Detroit Diesel") can sometimes only be identified correctly once their component terms have been found as part of the complete name (*e.g.*, "Detroit Diesel Corp."). The converse is also true, as short forms of person names (*e.g.*, "Mr. Olatunji") can help identify full name forms (*e.g.*, "Babatunde Olatunji").

The interpreter maintains a temporary lexicon on a document-by-document basis. Every time the interpreter changes the label of a phrase ϕ, pairs of form $\langle\lambda, \tau\rangle$ are added to the lexicon, where λ is a lexeme in ϕ, and τ is the label with which ϕ is tagged. This lexicon is then exploited to form the associations between short and long proper name forms (through an extension to the rule repertory defined above).

Correspondence to the regular sets

It is straightforward to prove that this approach recognizes a subset of the regular sets, so we will only sketch the outline of such a proof here. The proof proceeds inductively by constructing a finite state machine μ that accepts exactly those strings which receive a certain label in the phrase-finding process under a given rule sequence Σ. We consider each rule ρ in Σ in order, and correspondingly elaborate the machine so as to reproduce the rule's effect.

To begin with, consider that the initial phrase labeling proceeds by building phrases around lexemes $\alpha_1,\ldots,\alpha_n$ in a designated word list or by finding runs of certain parts of speech $\pi_1,\ldots,\pi_m$. The machine that reproduces this initial labeling is thus

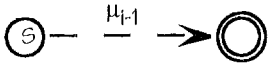


As usual, node labeled "S" is the start state, and any node drawn with two circles is an accepting state. The p_i/π_j arc labels stand for all lexemes in the lexicon that may be labeled with the part of speech π_j.

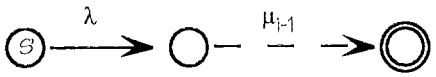
The induction step in the construction proceeds from μ_{i-1}, the machine built to reproduce Σ up through rule ρ_{i-1} in the sequence, and adds additional states and arcs so as to reproduce Σ up through rule ρ_i.

For example, say ρ_i tests for the presence of a lexeme λ to the left of a phrase and extends the phrase's boundaries to include λ. We extend the machine μ to

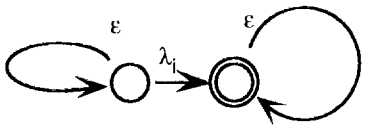
encode this rule by replacing μ's current start state S with a new one S', and adding a λ transition from S' to the former start state S. Thus



becomes



For a rule ρ_i that tests whether a phrase contains a certain lexeme λ_j, we construct an "acceptor" machine that accepts any string with λ_j in its midst.



Noting that the regular sets are closed under intersection, we them proceed to build the machine that "intersects" the acceptor with μ_i.

Other rule patterns are handled with constructions of a similar flavor—space considerations preclude their description here. Note, however, that extending the framework with a temporary lexicon makes it trans-finite-state. Finally, as with all semi-parsers, the machines we construct in this way must actually be interpreted as transducers, not just acceptors.

Learning rule sequences automatically

Our experience with writing rule sequences by hand in this approach has been very positive. The rule patterns themselves are simple, and the fact that they are sequenced localizes their effects and reduces the scope of their interactions. These hand-engineering advantages are also conferred upon learning programs that attempt to acquire these rules automatically.

The approach we have taken towards discovering phrase rule sequences automatically is a maximum error-reduction scheme for selecting the next rule in a sequence. This approach originated with Brill's work on part-of-speech tagging and bracketing (Brill 1993).

Brill's rule learning algorithm

The search for a rule sequence in a given training corpus begins by first applying the initial labeling function, just as would be the case in running a complete sequence. Following this, the learning procedure needs to consider every rule that can possibly apply at this juncture, which itself is a function of the rule schema language. For each such applicable rule r, the learner considers the possible improvement in phrase labeling conferred by r in the current state. The rule that most reduces the residual error in the training data is selected as the next rule in the sequence.

This generate-and-test cycle is continued until a stopping criterion is reached, which is usually taken as the point where performance improvement falls below a threshold, or ceases altogether. Other alternatives include setting a strict limit on the number of rules learned, or cross-testing the performance improvement of a rule on a corpus distinct from the training set.

The rule search space

The language of phrase rules supports a large number of possible rules that the phrase rule learner might need to consider at any one time. Take one of our smaller training sets, in which there are 291 sentences consisting of 6,812 word tokens, with 2,077 unique word types. Considering only lexical rules (those that look for particular words), this means that there are as many as 18,693 possible unary lexical rules (2,077 x 9 rule schemata), and 12,941,787 binary lexical rules ($2{,}077^2$ x 3 simple bigram rule schemata) in the search space. However, by inverting the process, and tabulating only those lexical contexts that actually appear in the training texts, this search space is reduced to 2,219 unary lexical rules and 854 binary lexical rules.

There are two substantively different kinds of rules to acquire: rules that only change the label of a phrase, and those that change the boundary of a phrase. The latter present a problem for accurately estimating the improvement of a rule, since sometimes the boundary realignment necessary to fix a phrase problem exceeds the amount by which a single rule can move a boundary—namely, two lexemes. For these phrases to be fixed there will have to be more than one rule to nudge the appropriate phrase boundaries over. We handle this through a heuristic scoring function that estimates the value of moving a boundary in such cases.

Error estimation methods

A rule that fixes a problem in some cases might well introduce errors in some other cases. This kind of over-generalization can occur early in the learning process, as new rules need only improve over an approximate initial labeling. The extent to which a candidate rule is rewarded for its specificity and penalized for its over-generalization can have a strong effect on the final performance of the rule sequences discovered.

We explored the use of three different types of scoring metrics for use in selecting the "best" of the competing rules to add to the sequence. Initially we made use of a simple arithmetic difference metric, $y - s$, where y (for yield) is the number of additional correct phrase labelings that would be introduced if a rule were to be added to the rule sequence, and s (for sacrifice) is the number of new mistaken labelings that would be introduced by the addition of the rule. This is Brill's original metric, but note that it does not differentiate between rules whose overall improvement is identical, but whose rate of over-generalization is not. For example, a rule whose yield is 100 and sacrifice is 70 is treated as equally valuable as one whose yield is only 30 but which introduces no overgeneralization at all (*sacrifice* = 0). This can lead to the selection of low-precision rules, and while small numbers of precision errors may be patched, wholesale precision problems make subsequent improvement more difficult.

Scoring metric	*Training*			*Test*		
	Recall	*Precision*	*P&R*	*Recall*	*Precision*	*P&R*
Arithmetic *(y-s)*	88.8	81.2	84.8	87.2	79.0	82.9
Log likelihood	81.9	85.7	78.4	81.0	73.4	77.0
F measure, β=0.8	86.3	82.9	84.5	85.0	81.5	83.2

Table 2: Comparative contributions of three scoring measures after 100 learning epochs. (Training on 1495 sentences from the MUC-6 named entities task).

The next measure we investigated was one advocated by Dunning (1993) which uses a log likelihood measure for estimating the significance of rare events in small populations. This measure did not improve precision or recall in the learned sequences.

The third scoring measure we investigated was the F-measure (VanRijsbergen 1979), which was introduced in information retrieval to compute a weighted combination of recall and precision. The F-measure is also used extensively in evaluating information extraction systems at MUC (Chinchor 1995). It is defined as:

$$F = \frac{(\beta^2 + 1)PR}{(\beta^2 + P)R}$$

This measure is conservative in the sense that its value is closer to precision, P, or recall, R, depending on which is lower. By manipulating the β parameter one is able to control for the relative importance of recall or precision. Preliminary exploration shows that a β of 0.8 seems to boost precision with no significant loss in the long-term recall or F-measure of the rule sequences.

Table 2 summarizes the contributions of these three error measures towards learning rule sequences for the MUC-6 named entities task (for task details, see below).

Evaluation

We have applied this rule sequence approach to a variety of realistic tasks. These largely arose as part of our information extraction efforts, and have been either directly or indirectly evaluated in the context of two evaluation conferences: MUC-6 and MET (for Multilingual Entity Tagging). In this paper, we will primarily report on evaluation conducted in the context of the MUC-6 named entities task (Sundheim 1995).[1]

The named entities task attempts to measure the ability to identify the basic building blocks of most newswire analysis applications, e.g., named entities such as persons, organizations, and geographical locations. Also measured is the identification of some numeric expressions (money and percentiles), dates, and times. This task has become a classic application for finite-state pre-parsers, and indeed our work was in part motivated by the success that has been achieved by such systems in past information extraction evaluations.

We have applied a variety of techniques towards this task. The easy cases of dates and times are identified by a separate pre-processor, leaving numeric expressions (also easy) and "proper names" (the interesting hard part) to be treated by the rule sequence processor.

Hand-crafted Rules

We first approached this task as an engineering problem, and wrote a rule sequence by hand to identify these named entities. The rule sequence comprises 145 named-entity rules, 12 rules for expressions of money and percentiles, and 61 rules for geographical complements (as in "Hyundai of Canada"). In addition, the rules refer to a few morphological predicates and some short word lists—one such list, for example lists words designating business subsidiaries, *e.g.*, "unit". The initial phrase labeling for the proper name cases is implemented by accumulating runs of NNP- and NNPS-tagged lexemes. A similar strategy is used for number expressions, using numeric tags.

The performance of our hand-crafted rule sequence is summarized in Table 3, below, which gives component scores on the MUC-6 blind test set. The most interesting measures are those for the difficult proper name cases. Our performance here is high, especially for person names. Our lowest score is on organizational names, but note that the system lacks any extensive organization name list. Aside from ten hard-wired names, all names are found from first principles. On the easy numeric expressions, performance is almost perfect—precision appears poor for percentiles, but this is due to an artifact of the testing procedure.[2]

Machine-crafted Rules

To evaluate the performance of our learning algorithm, we attempted to reproduce substantially the same environment as is used for the hand-crafted rules. The learner had access to the same predefined word lists, including the less-than-perfect TIPSTER gazetteer. Further, we only acquired rules for the hardest cases, namely the person, organization, and location phrases. We cut off rule acquisition after the 100th rule.

The results for this acquired rule set are surprisingly encouraging. As Table 3 shows, these rules achieved higher recall on the very hardest phrase type (organization) than their hand-crafted counterparts, albeit at a cost in precision. Overall, however, the machine-crafted rules still lag behind. When we incorporated them into our information extraction

[1] We have also measured performance on several syntactic constructs, (*e.g.*, the so-called noun group), and on semantic subgrammars, (*e.g.*, person-title-organization appositions).

[2] Our performance vis-a-vis other MUC-6 participants placed us in the top third of participating systems. Except for the absolute highest performer, all these top-tercile systems were statistically not distinguishable from each other.

Phrase type	*N*	*Hand-crafted rules*		*Machine-learned rules*	
		Recall	*Precision*	*Recall*	*Precision*
Organization	419	85	87	87	79
Person	348	94	94	78	79
Location	109	94	87	82	68
Money	74	99	97	—	—
Percent	16	100	67	—	—
All phrases	2150	91	92	88	83
		Overall F=91.2		*Overall F= 85.2*	

Table 3: Performance on the MUC-6 named entities blind test.

system, the machine-learned rules achieved an overall named entities F-score of 85.2, compared to the 91.2 achieved by the hand-crafted rules. It should be noted, however, that the system loaded with these machine-crafted rules still outperformed about a third of systems participating in the MUC-6 evaluation.

Multilingual evaluation (MET)

After the MUC-6 evaluation, the named entity task was extended in various ways to make it more applicable cross-linguistically. Predictably, this was followed by a new round of evaluations: MET. The target languages in this case were Spanish, Chinese, and Japanese. We applied our approach to all three.

The MET evaluation required actual system performance results to be kept strictly anonymous, which precludes our reporting here any scores as specific as we have cited for English. What we may legitimately report, however, is that we have effectively reproduced or bettered our hand-engineered English results in the Spanish and Japanese tasks, despite having no native speakers of either language (and only the most rudimentary reading skills in Kanji). In both cases, we were able to exploit part-of-speech tagging and some existing word lists for person names and locations.

For Chinese, although we had available a word segmenter, we had neither part-of-speech tagger, nor word lists, nor even the elementary reading skills we had for Japanese. As a result, we had to rely almost entirely on the learning procedure to acquire any rule sequences. Despite these impediments, we came close to reproducing our results with the English machine-learned named entities rule sequence.

Discussion

What is most encouraging about this approach is how well it performs on so many dimensions. We have only reported here on name-finding tasks, but early investigations in other areas are encouraging as well. With rule sequences that parse noun groups, for instance, we hope to reproduce the utility of other rule-sequence approaches to text chunking (Ramshaw & Marcus 1995). We are also excited by the promise of the learning procedure, not just because it learns good rules, but also because the rules it learns can be freely intermixed with hand-engineered rules. This mixed-mode acquisition is unique among natural language learning procedures, and we put it to good use in building our multilingual name-tagging sequences.

Despite results that compare favorably to those of more mature systems, this work is still in its infancy. We still have much to explore, especially with the learning procedure. Indeed, while the learner induces rule sequences that perform well in the aggregate, individual rules clearly show their mechanical genesis. For instance, when the learner must break ties between identically-scoring rule candidates, it often does so in linguistically clumsy ways. At times, the learner may acquire a good contextual pattern, but may be unable to extend it to closely-related cases that would occur naturally to a linguist.

We believe these problems are solvable in the near-term, and we have partial solutions in place already. As our techniques mature, this validates not only our particular approach to phrase-finding, but the whole field of language processing through rule sequences.

References

Aberdeen, J., Burger, J., Day, D., Hirschman, L., Robinson, P., & Vilain, M. 1995. "Description of the *Alembic* system used for MUC-6". In *Prcdgs. of* MUC-6, Columbia MD.

Appelt, D. E., Hobbs, J. R., Bear, J., Israel, D., & Tyson, M. 1993. "FASTUS: A finite-state processor for information extraction from real-world text." In *Prcdgs. of* IJCAI-93, Chambéry, France.

Brill, E. 1993. *A corpus-based approach to language learning*. Doctoral Dissertation, Univ. of Pennsylvania.

Chinchor, N. 1994. "MUC-5 evaluation metrics". In *Prcdgs. of* MUC-5, Baltimore, MD.

Dunning, T. 1993. "Accurate methods for the statistics of surprise and coincidence". *Comput. Ling* **19**.

Grishman, R. 1995. "The NYU system for MUC-6, or where's the syntax?" In *Prcdgs. of* MUC-6, Columbia MD.

Jacobs, P. S., Krupka, G., & Rau, L. 1991. "Lexico-semantic pattern-matching as a companion to parsing". In *Prcdgs. of the Fourth* DARPA *Speech and Nat. Lang. Workshop*, San Mateo, CA: Morgan Kaufman.

Ramshaw, L. & Marcus, M. 1995. "Text chunking using transformation-based learning". In *Prcgs. of 3rd Wkshp on Very Large Corpora*, Cambridge, MA.

Sundheim, B. 1995. "Named entity task definition". In *Prcdgs. of* MUC-6, Columbia MD.

Van Rijsbergen, C.J. 1979. *Information Retrieval*. London: Buttersworth.