Pattern Matching in a Linguistically-Motivated Text Understanding System

Damaris M. Ayuso and the PLUM Research Group

BBN Systems and Technologies
70 Fawcett St.
Cambridge, MA 02138
dayuso@bbn.com

ABSTRACT

An ongoing debate in text understanding efforts centers on the use of pattern-matching techniques, which some have characterized as "designed to ignore as much text as possible," versus approaches which primarily employ rules that are domain-independent and linguistically-motivated. For instance, in the message-processing community, there has been a noticeable pulling back from large-coverage grammars to the point where, in some systems, traditional models of syntax and semantics have been completely replaced by domain-specific finite-state approximations.

In this paper we report on a hybrid approach which uses such domain-specific patterns as a supplement to domain-independent grammar rules, domain-independent semantic rules, and automatically hypothesized domain-specific semantic rules. The surprising result, as measured on TIPSTER test data, is that domain-specific pattern matching improved performance, but only slightly, over more general linguistically-motivated techniques.

1. Introduction

Virtually all systems which participated in the Fifth Message Understanding Conference, MUC-5 [1], used finite-state (FS) pattern matching to some extent. Two useful tasks that this approach is well suited for are:

1. treating application-specific simple constructions that may not belong in a general grammar of the language, and
2. detecting constructions which, though grammatical, may be found more reliably using domain-specific patterns.

For example, special-purpose FS subgrammars were used widely to efficiently and reliably recognize equipment names and company names. This illustrates one (1) above. An illustration of (2) appears in the complex sentence below:

> Daio Paper Corp. said it will *set up* a cardboard factory in Ho Chi Minh City, Vietnam, *jointly* with state-run Cogido Paper Manufacturing Company.

It is easy for any parser to err in not attaching the modifier "jointly" to "set up," and thereby miss the fact that a joint venture is being reported. One might argue that the sentence includes a discontiguous constituent ("set up ... jointly"). Nevertheless, it is easy to write a general pattern to deal with the discontiguous constituent correctly for this domain.

Finite-state parsers perform simple operations, and they are fast. In data-extraction applications, where much of the input can be safely ignored, they provide an easy means to skip text without deep analysis. Some of the best-performing systems in MUC-5 relied heavily on the use of finite-state pattern-matching in crucial system components.

However, there are several advantages in maintaining broad linguistically-based coverage of a language, even in a data-extraction task. First, it allows for well-defined linguistic structures to be recognized and represented in a domain independent way. This provides a level of linguistic representation that can be used by other general linguistic components such as a domain-independent discourse processor. In fact, this is a representational level which will probably be evaluated in the next Message Understanding Conference, MUC-6.

Secondly, general linguistic coverage provides application independence. Different applications, such as data detection (information retrieval) can use the linguistic representations for various purposes. Achieving a synergistic operation of data-extraction and data-detection systems is one of the key goals of ARPA's TIPSTER Phase II project.

Another intuitive advantage is portability. When porting a system to a new application, a base level of understanding is achieved very quickly before having to add domain-specific patterns. This is possible because the bulk of the processing work is done by the domain-independent rules.

BBN's data-extraction system, PLUM [2], showed consistently high-ranking performance in the MUC-3 [3], MUC-4 [4], and MUC-5 evaluations. We added two new finite-state pattern-matching modules to PLUM between MUC-4 and MUC-5, expecting a substantial payoff in performance. The surprising result, as measured on TIPSTER test data, was that although domain-specific pattern matching improved performance, in the English domains it was only a slight improvement over more general, linguistically-motivated techniques.

In the next section we further discuss the movement towards FS approximations in the community. We then describe the role of finite-state pattern-matching in BBN's PLUM system in more detail. Finally we present experiments used to measure the resulting effect in PLUM.

2. A Shift in the Community

Text processing systems participating in the MUC evaluations (most recently, MUC-5) perform linguistic processing to various levels. Some systems may attempt to do a deep level of understanding whenever possible [5], whereas others use more shallow

"skimming" techniques [6], focusing only on information of interest and ignoring all other text. Similarly, systems span the spectrum in their use of finite-state pattern-matching instead of the more traditional, general, syntactic and semantic processing.

There are several reasons for the recent shift to increased use of FS approximations. Work was published on deriving finite-state approximations from more general grammars [7]. Then in MUC-3 it became evident that, in certain data-extraction tasks, a system which ignored much of the input text but focussed attention on the items of interest could perform as well as other systems which emphasized deeper understanding of all the text. Once the problem of data-extraction was perceived to only require the understanding of small fractions of the input text, some systems evolved to do more shallow processing and the use of finite-state approximations increased.

It should be noted that incorporating finite-state elements can result in advantages that are important for achieving operational data-extraction systems. The simplicity of the finite-state formalism makes FS rules more easily understandable (and thus, modifiable) by non-experts. Since parsing finite-state grammars can be done very efficiently, another advantage is fast processing, which is desirable in many real applications.

In some systems, notably SRI's and GE's, there was a dramatic shift between MUC-3 and MUC-5 towards the use of finite-state pattern-matching in all the critical linguistic components, relying heavily on domain-specific patterns. Development of GE's MUC-5 system, SHOGUN [8], resulted in the complete replacement of their general syntactic parser by a complex FS grammar. This new grammar encodes domain-specific information which was formerly distributed in other components. SRI's new FASTUS [9] relies on a cascade of finite-state transducers; the first stages find simple linguistic structures, and the final and most important stage consists of multiple levels of domain-specific finite-state patterns. Information not matched is ignored.

Although both of these systems (along with BBN's PLUM), were top performers in MUC-5, they now lack a large-coverage domain-independent syntactic and semantic model. Rather, they rely on intensive analysis of domain corpora in order to encode patterns in each new domain.

3. Role in PLUM's Architecture

BBN's PLUM has a traditional and general-purpose processing core, where morphological analysis, syntactic parsing, semantic interpretation, and discourse analysis take place. Purely syntactic parse structures and general semantic interpretations are created during processing. When porting to a new domain, we can use our automatic procedures for learning lexical-semantic case-frame information from annotated data [10] to quickly obtain domain-specific understanding without using finite-state approximations. This then becomes the initial system on which more detailed development is based.

During the development for TIPSTER, we added to the core PLUM system two new optional processing modules which do use finite-state patterns for the following specific purposes:

1. detect domain-specific simple constructions that can be identified on the basis of shallow lexical information, and

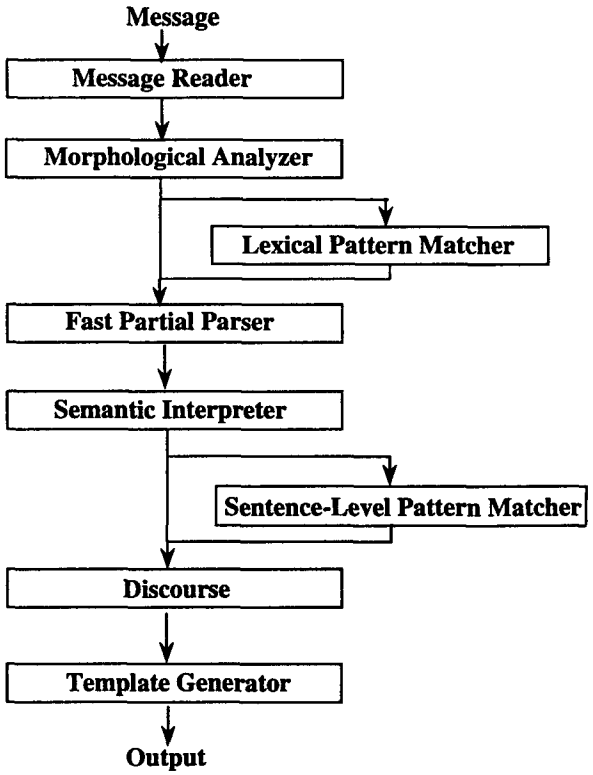

Figure 1: PLUM Architecture

2. as a backup strategy, identify patterns that are likely to have been fragmented during regular processing.

Figure 1 shows PLUM's architecture. Parallel possible paths are indicated where the optional pattern-matching modules appear. The two new modules, the Lexical Pattern Matcher and the Sentence-Level Pattern Matcher, use the same core finite-state pattern-matching processor, SPURT, which is described in the next section.

The Lexical Pattern Matcher operates before parsing but after tagging by part-of-speech to recognize constructions which can be detected based on component words, their parts-of-speech, and simple properties of their lexical semantics. This is used primarily for structures that could be part of the grammar, but can be more efficiently recognized by a finite state machine. Examples are corporation/organization names, person names, and measurements.

The Sentence-Level Pattern Matcher replaced our former fragment combining component which sought to attach contiguous fragments based on syntactic and semantic properties. The new pattern-matching component applies FS patterns to the fragments of the parsed and semantically interpreted input; the matched patterns' associated actions may modify or add new semantic information at the level of the sentence. That is, semantic relationships may be added between objects in potentially widely separated fragments of the sentence, thereby handling the example of the discontiguous constituent presented earlier.

4. SPURT: A Finite-State Pattern-Action Language

We defined our first version of the FS pattern-matcher and FS grammar syntax for a gisting application [11]. The problem there was to extract key information (e.g., plane-id, command) from the output of a speech recognizer whose input was (real) air-traffic controller and pilot interactions. This initial version of the pattern-matcher was also utilized, for the purpose of detecting company names, in the PLUM configuration used for the initial TIPSTER evaluations.

Before MUC-5 we made the FS grammar syntax more powerful (though still finite-state) to give the rule-writer more flexibility. We also introduced optimizations to the parser and added an action component to the rules. The resulting utility is named SPURT. We first used SPURT for applying sentence-level patterns, and later replaced the simple company name recognizer by SPURT to perform general lexically-based pattern matching of various types of constructions.

SPURT rules are finite-state patterns which can be used to search for complex patterns of information in a sentence and build semantic structures from that information. A SPURT rule has a :PATTERN component which is the expansion (the "right-hand side") of a finite-state grammar rule. It optionally has an :UNDERSTANDING component which states actions to take if the pattern is matched. Examples of SPURT rules are included in subsequent sections.

Rules are either *top-level* rules or *sub-level* (supporting-level) rules. Top-level rules indicate multiple entry points into the grammar defined by the patterns, and may invoke sub-level rules, as in a context-free grammar where the right hand side of a non-terminal may be in terms of other non-terminals. Top-level patterns are iterated over for each sentence, and the actions corresponding to matched rules are executed.

Rules are assigned a *phase*. Rules belonging to phase $n+1$ operate on the input after it is mutated by phase n. So far we have only seen the need for up to 2 phases in our rules.

When the SPURT rules are read in at system load time, they are compiled into a network of nodes and arcs. Arcs coming out of a node indicate multiple possible next states. Nodes contain tests, so that if the test at the end-node of an arc is successful when applied to the input at the pointer, that arc is traversed. The parser simply matches an input against the network, performing a depth-first search, and selecting a path that matches the maximal amount of input. At each decision point, arcs are tried in an order which favors paths that consume a maximal amount of input in a meaningful way (e.g., the parser only follows "don't-care" arcs when other possibilities are exhausted). Once a successful parse of the whole input is found the search is terminated.[1] The resulting path is then traversed to execute the corresponding actions.

4.1. Lexically-based SPURT

The Lexical Pattern Matcher applies SPURT patterns after morphological analysis but prior to parsing. The input consists of word tokens with part-of-speech information. A pattern can test on a token's word component, its part-of-speech, its semantic type, or a top-level predicate in its lexical semantics. When a pattern is matched, the action component identifies substrings of the matched sequence to add to the temporary lexicon. These temporary definitions are active for the duration of the message.

For example, a pattern for recognizing company names could match a sequence such as ("Bridgestone" NP) ("Sports" NPS) ("Co." NP), where NP is the tag for proper nouns, and NPS for plural proper nouns; the pattern's action results in this sequence being replaced by the singular token ("Bridgestone Sports Co." NP), which is, as a side effect, defined as a lexical entry having semantic type CORPORATION.

Figure 2 shows the rules used to match the example above. The first sub-rule, NP-PLUS, finds sequences of tokens that have been tagged as proper nouns. The XXX-CO rule finds sequences of the type {*'the'*} *[proper-noun]*+ {*[proper-noun-plural]*} *[corp-designator]*. The :TERM-PRED operator appearing in this rule allows for other simple tests on the tokens. In this case, the *corp-designator?* test tries to match one of a list of possible company designators, e.g., "Corp.". The CO-INSTANCE rule determines the existence of a company name if one of many company patterns matches. If there is a match, the pattern assigns the tag *tag-string* to the sequence, and the action component creates a lexical entry for it. The lexical entry is assigned type CORPORATION and assigned the predicate NAME-OF relating the entry to a string created out of the words in the matched sequence. Finally, the top-level rule CO finds multiple instances of companies in the input.

```
(def-sub-patt NP-PLUS
    (:pattern (:PLUS tagger::np)))

(def-sub-patt XXX-CO
    (:pattern
      (:SEQ (:OPT "the") (:RULE NP-PLUS)
            (:OPT tagger::nps)
            (:TERM-PRED corp-designator?))))

(def-sub-patt (CO-INSTANCE (:args tag-string))
    (:pattern
     (:tag tag-string
           (:OR ... (:RULE XXX-CO) ...)))
  (:understanding
   ((:type CORPORATION tag-string name)
    (:string STR tag-string)
    (:pred NAME-OF tag-string STR))))

(def-top-patt CO
  (:pattern
   (:seq
    (:plus
     (:seq
      (:star :anyword)
      (:rule CO-INSTANCE corp-string)))
    (:star :anyword))))
```

Figure 2: Lexical Pattern Example

[1] In all-paths mode, the parser can be used to find arc probabilities based on training data. This was used in the gisting application, but has not yet been used in PLUM.

4.2. Sentence-Level SPURT

The input to Sentence-Level SPURT is a sentence object which has already been processed through the fragment semantic interpreter. Its fragments' parse nodes have already been annotated with a semantic interpretation. SPURT's parser actually operates on the leaf elements (the nodes corresponding to the terminals, or words) of the fragment parses. The "pointer" can move along the input either at the word level, or at the level of higher structures, achieved by matching nodes that are ancestors of the leaf nodes. Thus patterns can test on words or phrases. When a word is matched, the parse pointer moves to the next word's leaf node; if a phrase is matched, it is moved to the next possible word not spanned by the tree.

A pattern can test both syntactic and semantic information associated with the parse nodes. When a pattern is matched, the action component specifies new semantic information to be added to particular parse nodes (and thus to the fragment in which each node is contained). The new information is allowed to include predicates connecting semantic structures across different fragments–this is something the fragment semantic interpreter is unable to do, as it is a compositional operation on the individual, independent, parse fragments.

Below is an example of a sentence-level rule which will match the example given in the introduction. This pattern matches sequences of the type *[anyword]* [joint-word] [anyword]* [activity-corporation-or-venture-np] [anyword]**, where *[joint-word]* (or *JOINT-WORDS* as specified below) is one of a list of words such as "jointly" and "together". The operator :AND-ENV introduces tests on phrases in a parse tree: :CAT indicates the phrase category; because some phrasetypes are recursive, :LOW (or other values) is used to indicate which level of the recursive structure is the one to be looked at; and :CONCEPT indicates the semantic type that is desired of that phrase. The simple action component of this rule adds the semantic type JOINT-VENTURE to the parse-node where the *joint-word* occurred. In effect this is indicating there is a joint-venture in the sentence. Note that this pattern makes no decisions regarding the role, if any, that *[activity-corporation-or-venture-np]* plays in the joint venture.

```
(def-top-patt JOINT-WORD-TRIGGER
 (:pattern
  (:seq
   (:star :anyword)
   (:tag joint-word-tag (:term *JOINT-WORDS*))
   (:star :anyword)
   (:or
     (:and-env (:cat np :low)
               (:concept JV-ACTIVITY))
     (:and-env (:cat np :low)
               (:concept CORPORATION))
     (:and-env (:cat np :low)
               (:concept POSS-JV)))
   (:star :anyword)))
  (:understanding
   ((:type JOINT-VENTURE joint-word-tag))))
```

Figure 3: Sentence-Level Pattern Example

5. Experiments

In order to measure the impact of the new FS components, we ran our MUC-5 English configurations (for English joint ventures and English microelectronics) on two test sets. The first is test data used for the TIPSTER 18-month evaluation, the second is data that was released for training, but was kept blind. For each pair of domain and test set, we ran experiments in each of 4 modes:

- Baseline: our default configuration, using both FS components;
- No Lexical FS: turn off lexical FS processing, except for the old company-name recognizer;
- No Sentence FS: turn off sentence-level FS processing; and
- No FS: turn off both.

The default configurations in the two domains share the same processing modules, the same general domain-independent grammar and semantic rules, and the same company-name recognizer. Each configuration contains its own set of domain-specific lexical-semantic definitions. A lexical-semantic definition contains the word's semantic type and (optionally) case-frames identifying semantic tests on possible arguments to the word. The semantic interpreter uses these rules in compositionally assigning semantics to parse-trees. For EJV, the initial version of the lexical semantics was automatically generated from training data [10]; it was then modified manually as needed.

Although we tested both domains, we consider the test on EME to be more representative of the effects of the new modules. Most of the EJV development preceded the existence of the modules. In fact, for EJV we added no new rules to the lexical FS component. EME development, however, was able to take advantage of the new utilities almost from the start. It made heavier use of the front-end rules for some of the tricky technical constructions in that domain; it should be noted that even then, the impact of the lexical FS was minimal in that domain.

Table 1 shows the difference in ERR for the various modes. ERR was the primary error measure used in MUC-5; to show improved performance, the goal is to minimize this measure. The new FS components, as evidenced by the Base results, improved ERR by at most 3 percentage points.

	Base	No Lex	No Sent	No FS
EJV-1	66	66	68	68
EJV-2	68	68	70	70
EME-1	59	60	61	62
EME-2	62	63	63	63

Table 1: Impact of New FS Components: Numbers are ERR measures

It should be noted that the Japanese domains, JJV and JME, made heavy use of the sentence-level patterns. FS patterns for JJV gave us a quick gain in performance, but the price paid was having little carryover to the JME domain once that development began. We did not test those domains without the FS components. Based on our experience, if multiple Japanese domains are expected, we would

undoubtedly build a robust domain-independent core of semantic rules, which in the long-run maximizes re-usability and minimizes effort for each new domain. We utilized FS patterns because our Japanese expert wanted to explore the capabilities, and limits, of pattern-matching.

6. Conclusion

Finite-state pattern-matching has already shown to be useful and valuable in data-extraction applications. Its full possible impact is still being investigated. For example, several groups are trying to find automatic ways to derive FS patterns in order to surmount the porting problem they pose in systems that heavily depend on them.

However, maintaining a wide-coverage linguistic core can result in excellent data-extraction capability as has been evidenced by PLUM's performance in the government-sponsored MUC evaluations.

Perhaps the most interesting result was that domain-specific patterns, though in principle very powerful, added relatively little to the performance of the linguistically motivated components. Error rate was improved by at most 3 percentage points. Nevertheless, PLUM data extraction system's performance was among the highest of all systems participating in MUC-5.

While this one case study does not prove the relative efficacy of domain-specific patterns versus domain-independent, linguistically motivated processing, it does suggest that more research and development in linguistically motivated syntactic and semantic processing is promising even in the short term, not just in long range research.

7. Acknowledgements

The work reported here was supported in part by the Advanced Research Projects Agency and was monitored by the Rome Air Development Center under Contract No. F30602-91-C-0051. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the United States Government.

The members of the PLUM Research Group are: Ralph Weischedel (Principal Investigator), Damaris M. Ayuso, Sean Boisen, Heidi Fox, and Constantine Papageorgiou (BBN), and Dawn MacLaughlin (Boston University).

References

1. *Proceedings of the Fifth Message Understanding Conference (MUC-5)*, August 1993, to appear.
2. The PLUM System Group. BBN PLUM: MUC-5 System Description. To appear in *Proceedings of the Fifth Message Understanding Conference (MUC-5)*, August 1993.
3. *Proceedings of the Third Message Understanding Conference (MUC-3)*, Morgan Kaufmann Publishers Inc., May 1991.
4. *Proceedings of the Fourth Message Understanding Conference (MUC-4)*, Morgan Kaufmann Publishers Inc., June 1992.
5. Grishman, R. and Sterling J. New York University: Description of the Proteus System as Used for MUC-5. To appear in *Proceedings of the Fifth Message Understanding Conference (MUC-5)*, August 1993.
6. Lehnert, W., McCarthy, J., Soderland, S., Riloff, E., Cardie, C., Peterson, J., and Feng, F. UMASS/HUGHES: Description of the CIRCUS System Used for MUC-5. To appear in *Proceedings of the Fifth Message Understanding Conference (MUC-5)*, August 1993.
7. Pereira, F. Finite-State Approximations of Grammars. *Proceedings of the Speech and Natural Language Workshop*, pages 20-25. Morgan Kaufmann Publishers Inc., June 1990.
8. Jacobs, P. (Contact). GE-CMU: Description of the SHOGUN System Used for MUC-5. To appear in *Proceedings of the Fifth Message Understanding Conference (MUC-5)*, August 1993.
9. Appelt, D., Hobbs, J., Bear, J., Israel, D., Kameyama, M., and Tyson, M. The SRI MUC-5 JV-FASTUS Information Extraction System. To appear in *Proceedings of the Fifth Message Understanding Conference (MUC-5)*, August 1993.
10. Weischedel, R., Ayuso, D., Bobrow, R., Boisen, S., Ingria, R., and Palmucci, J. Partial Parsing: A Report on Work in Progress. *Proceedings of the Speech and Natural Language Workshop*, pages 204-209. Morgan Kaufmann Publishers Inc., Feb 1991.
11. Rohlicek, R., Ayuso, D., Bates, M., Bobrow, R., Boulanger, A., Gish, H., Jeanrenaud, P., Meteer, M., Siu, M., Gisting Conversational Speech" in *Proceedings of International Conference of Acoustics, Speech, and Signal Processing (ICASSP)*, Mar. 23-26, 1992, Vol.2, pp. 113-116.