4-24-2003

P-FASTUS: Information Extraction System Implemented in a Constraint Programming Language -SICStus Prolog

Rajen Subba '03
Illinois Wesleyan University

P-FASTUS
Information Extraction system implemented in a Constraint Programming Language - SICStus Prolog

Rajen Subba
Supervisor: Dr. Hans-Joerg Tiede
Dept. of Mathematics and Computer Science

April 24, 2003

Contents

1 Introduction

P-FASTUS is an Information Extraction system developed in SICStus Prolog based on the implementation of FASTUS. It is program that extracts prespecified information such as the name of the company, location and the position being advertised from "Job Postings" in text files. The system is composed of different levels of processing phases that are implemented using finite-state transducers.

The next section gives a statement of Information Extraction (IE). Section 3 describes Prolog, particularly SICStus Prolog, a constraint programming language and outlines the appropriateness of its use for natural language processing. The rule-based approach for natural language processing is discussed in section 4. Section 5 summarizes the FASTUS system and the finite state approach to build an IE system. Sections 6 describes the P-FASTUS system that I have developed. Results and conclusions follow the description of P-FASTUS.

The following is an example of the text that P-FASTUS takes in as input:

```
The Estridge Group, the premier homebuilder in the Indianapolis
area, has a Sales Consultant position available at our Greystone Village
Community in Cicero, Indiana.  We are seeking a highly energetic individual
with excellent communication skills.
```

The IE system generates the following output:

Information extracted from file: ./testdata/test2.txt

company : Estridge Group
position : Sales Consultant
location : Greystone Village Community Cicero Indiana

2 What is Information Extraction?

The majority of the information held by businesses, government agencies and individuals are stored in text files. Only a handful of the information is stored in databases, in which case the information is structured. With the advent of the internet, the amount of textual information in the form of natural languages has been growing exponentially. Searching for documents containing relevant information on the web has become a fairly daunting task. Reading through thousands of documents to obtain the information that you require can be cumbersome. In order to address this issue, researchers have been developing Information Extraction systems using techniques of natural language processing.

The goal of Information Extraction is to extract from a set of documents, prominent facts about pre-specified types of events, entities or relationships. These facts are then usually entered into a database, which may then be used to analyze the data for trends, to give a natural language summary, or simply to serve for online access. Extracting information on relationships, events and entities requires a certain level of semantic analysis of the text. Therefore natural language processing techniques can be used. Sentences in the text can be parsed. Parsing of sentences involves the process of determining its phrasal structure. The different parts of speech that constitute the sentence have to be identified. This is similar to the task of parsing programming languages such as Java or C. The objective is to simply determine the grammatical structure of the sentences. For example, the sentence, "John will play baseball in college" would be parsed as follows:

John - `noun (or proper noun)`
will - `verb`
play - `verb`
baseball - `noun`
in - `preposition`
college - `noun`

In essence all languages, including English, are bound by a set of grammatical rules that are used to construct sentences. However, natural languages, unlike programming languages, are context sensitive. In other words, the syntax of programming languages is less complex than that of natural languages. Programming languages such as C, C++ and Prolog are based on a set of EBNF rules that have to be strictly adhered. Natural languages, on the other hand, are not deterministic. The fact that natural languages are context sensitive makes it difficult to develop systems that can accurately understand arbitrary texts. Information Extraction does not attempt to understand the texts it analyzes. Instead it simply searches for information that is being sought for that particular domain in interest (Appelt 1995).

In the United States, Information Extraction has been heavily influenced by

the Message Understanding Conferences (MUCs) which have been funded primarily by the Defense Advanced Research Projects Agency (DARPA). These conferences were instituted by DARPA in the late 80's. Seven MUCs were held, of which the last one was held in 1998. The focus of these conferences was on tasks like event extraction, named entities, template elements, coreference and scenario templates. At present, much research is conducted on improving the techniques for each of these tasks at various institutions and universities. One of the drawbacks of IE systems to date is that they are not portable. IE systems are domain specific. This means that each IE system will only extract information from a prespecified domain. During each MUC, IE systems were required to extract different kinds of information from documents on different events, mostly information that has been of interest to the department of defense. For example, for MUC-3 IE systems aimed to extract information on terrorist activities. Such systems would have been used to filter documents and emails to track any terrorist activities. MUC-5 IE systems extracted information on Joint Venture activities. Each of the systems in these MUC's filled templates designed for these specific events with the information from the documents. By MUC-6 researchers had aimed to make IE systems portable or more flexible. NYU's research team worked on an IE system that analyzed and extracted information from documents on Aircraft orders and Labor negotiations. In contrast to MUC-5, relatively simple templates were designed and the "template elements" (for people, organizations, and artifacts) which would apply to a wide variety of different event types were predefined. Although the systems presented at MUC-6 and MUC-7 were more flexible than their predecessors, they are still bound to a certain range or domain of events. Researchers have also been working on improving subparts of the IE system (such as Named Entity taggers) which identify names in text documents. Information Extractions systems have been built for other languages as well, such as Japanese. The MUC-5 conference showcased not only systems for extracting information on Joint Ventures from documents in English but from documents in Japanese as well (MUC-6).

Instituitions from different countries participated in these Message Understanding Conferences, each with their own techniques to handle the task. Most of these systems have been implemented in either Lisp or C. The decision to use these languages over others has been primarily based on the ease and familiarity of the languages. The goal of my research project was to develop an IE system in SICStus Prolog.

3 Constraint Programming: SICStus - Prolog

None of the IE systems have been implemented in Prolog, a language that was developed with an intention to facilitate natural language processing. Covington argues that Prolog may be the most suitable language to use for NLP and outlines his reasons as follows:

1. In Prolog, it is easier to represent syntactic and semantic structures because it is easy to build and modify large complex data structures.

2. The program itself can be examined and modified using built-in functions such as `assert` and `abolish`.

3. It is a language that is designed for knowledge representation and is built around a subset of first order logic.

4. Prolog has a built in parser for natural language processing and a depth first search algorithm.

5. Pattern matching is facilitated by its unification capability. This feature can be used to build data structures step-by-step without worrying about the order of the steps. (Covington 1994)

Languages such as C lack all of the above features that facilitate NLP. Lisp shares the first two features. It is the most widely used programming language for buliding IE systems (Covington 1994). When implementing P-FASTUS, all but the built-in parser of NLP has been used. The reason why the built-in parser for NLP was not used is because P-FASTUS is implemented by using a cascade of Finite-State Automatons.

SICStus is a version of Prolog that has a constraint programming library built in to it. Constraint programming involves using constraints to solve problems. A constraint is simply a logical relation among several unknowns, each taking a value in a given domain. A constraint restricts the possible values that variables can take. It represents some partial information about the variables of interest. Thus, by confining variables of interest to a set of possible value(s), it is possible to generate a set of solution(s). In Prolog variables are constrained to a certain value by using the $\#=$ symbol. For example, $X\#=4$ would entail that the variable X is set the value 4 (Marriott 1999).

4 Rule based NLP approach

IE systems generally process texts in sequential steps or phases ranging from lexical and morphological processing, recognition of proper names, parsing of larger syntactic constituents, resolution of anaphora and coreference, and the ultimate extraction of domain relevant events and relationships from the text. There are two basic approaches one can take when implementing these phases of the IE system. Appelt labels them as the Knowledge Engineering approach and the Automatic Training approach. Technically the Knowledge Engineering approach is based on logic or the use of rules while the Automatic Training approach is based on statistical NLP techniques that use probability (Appelt 1999).

The Knowledge Engineering approach requires that the implementor develop grammars or rules for a component of the system that parses sentences. The engineer who develops these rules must be familiar with the IE system and the formalism for expressing rules for that system. Thus, the engineer has to access a corpus of domain relevant texts and develop rules accordingly. The performance of such systems is dependent on the skill of the knowledge engineer as well (Appelt 1999).

Unlike the Knowledge Engineering Approach, the Automatic Training approach does not require a skilled engineer with detailed knowledge of the IE system. Instead someone with enough knowledge about the domain and the task to annotate a corpus of texts appropriately is required. Annotations would be required for a particular aspect or step of the IE system. For example, the name recognizer would be trained by annotating a corpus of texts with the domain-relevant proper names. Once a suitable training corpus has been annotated, a training algorithm is run, resulting in information that a system can employ in analyzing novel text (Appelt 1999).

Both these systems have their own advantages and disadvantages. Although the best performing systems have been hand crafted using the Knowledge Engineering approach, recent developments in statistical NLP techniques have made Automatic Training approaches as accurate. Since the Automatic Training approach requires a large corpus of training data which is not available, I will implement my IE system using the rule based approach rather than the Automatic Training approach.

5 Finite State Approach and FASTUS

One way to implement a rule based IE system is to use Finite State Machines. IE systems that partially and fully parse sentences have been developed. The parser can be either a bottom-up parser or a top-down parser. A partial parser, what is sometimes referred to as a shallow parser, is a bottom-up parser that does not check whether an input of words or lexicon forms a sentence. For example, in the sentence, "John will play baseball in college", the final phase of the partial parser could end with the following final analysis:

John – noun group
will play – verb group
baseball in college – noun group

On the other hand, a full parser will eventually check that the input, "John will play baseball in college." forms a sentence.

Formerly, it was assumed that there was little use of partially parsing sentences, so NLP researchers generally developed techniques to parse sentences completely. IE systems that fully parse sentences such as SRIs TACITUS and New York Universitys PROTEUS exist but they require a significant amount of computation and time. However, it turns out that, for the purposes of Information Extraction, a complete parse of sentences is not really required. Instead, a partial parsing of sentences can handle the task. Since sentences do not have to be completely parsed, finite state automatons can be used. FASTUS, a system developed by SRI is an example of a system that does partial parsing using finite state non-deterministic automatons. "FASTUS is a (slightly permuted) acronym for Finite State Automata-based Text Understanding System. It is a system for extracting information from free text. Typically, applications mark text with annotations that indicate items of interest, such as names of people or companies, or it fills templates with information that could be entered into a relational database. FASTUS works as a series of cascaded, finite-state automata." (Appelt 1993) FASTUS was one of the most succesful IE systems developed for the MUC conferences. It was among the most accurate systems and at the same time one of the fastest as well. FASTUS's success can be attributed to the very fact that it uses Non-deterministic Finite State Automatons (NFA) or what is also referred to as Finite State Machines (FSM).

So what is a Non-deterministic Finite State Automaton or FSM? Finite State Automatons are basically machines that consists of one or more internal states. They are called 'finite' because they are composed of a finite number of states. They are not physical machines that are subject to any kind of physical resistance such as friction or gravity. FSM are idealized machines that move from one internal state to another. A transition function governs the movement from one state to another. A FSM may have one or more accepting states where the input to the system results in a positive outcome. FSM's can be characterized

as tape machines which takes symbols as input. The symbols are fed to the tape of the machine as it moves from left to right. If the input to the machine does not end in an accepting state then the machine is said the reject the input.

Finite state automatons can either be deterministic (DFA) or non-deterministic (NFA). A DFA can be thought of as a one read-only input tape that can move only from left to right. Basically a DFA cannot change the direction of the machines's movement. Only one input is read at a time. In mathematical notation a DFA is a five-tuple machine:

$$M = (Q, \Sigma, \sigma, q_0, F)$$

where Q is the finite non-empty set of states.

$$\Sigma$$

is the data alphabet,

$$\sigma$$

is the transition function that governs the movement of the machine from one state to another.

$$q_0$$

is the initial state and F is the set of accepting states (Greenlaw 1998).

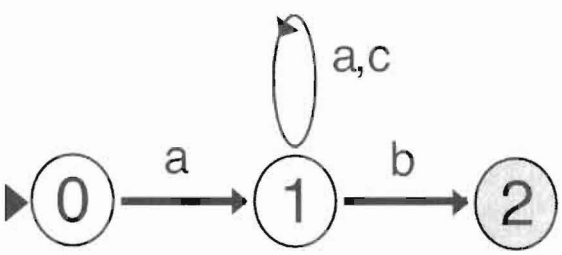

Figure 1: Deterministic Finite State Automata (DFA)

In the figure above, 0 is the initial state and 2 is the accepting state. The input 'a' moves the machine from state 0 to state 1. Only the input 'b' at state 1 moves the machine from state 1 to the accepting state 2. Note that there is only one possible transition for each input.

An NFA is also a five-tuple machine like a DFA with the same specifications.

The difference between a DFA and an NFA is the nature of the transition function. The transition function for an NFA is a finite subset of:

$$Q \times \Sigma \times Q$$

In other words the difference between the two is that a DFA can have only one transition function that moves the machine from one state to only one other state. An NFA on the other hand can have a transition function that can move the machine from one state to any other state for a given input (Greenlaw 1998).

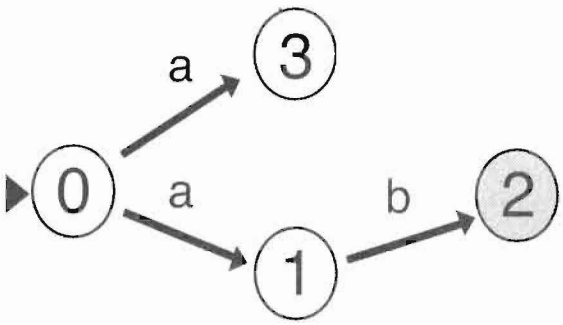

Figure 2: Non-Deterministic Finite State Automata (NFA)

In the figure above, the machine has 4 states, 0 being the initial state and 2 being the accepting state. This machine is non-deterministic since it can move to either state 3 or 1 from the initial state when it takes in the input 'a'. Therefore at state 0, with input 'a', the machine has two possible states to move to instead of just one.

A FSM that takes in an input and produces an output is calles a Finite State Transducer(FST). FASTUS and the P-FASTUS systems are both built using Finite State Transducers. However, I will refer to them as either FSM or finite state automata since they are in essence FSM's.

It is important to note that Finite State Machines are not capable of full natural language processing. This is because, as mentioned previously, natural languages are context-sensitive in nature. English has constructs, such as center embedding, that simply cannot be described by any finite state grammar. Consider the following example:

```
John will play baseball in college.
John the brother of James will play baseball in college.
John the brother of James Laura dated in High School will play baseball
```

in college.

Such recursive constructs cannot be recognized by any finite-state grammar

However, due to the memory limitations in human beings, it is impossible for people to fully expoit the context-freeness of English. This realization of memory limitation led Church to advocate the use of finite state grammars for natural language processing. Further work on using finite state machines for natural language processing was done by Pereira and Wright in 1991 who developed methods for constructing finite state grammars from context-free grammars (Appelt 1994).

As mentioned previously, FASTUS has been implemented by using a cascade of Non-deterministic Finite State Automata. The composite structures or output generated by one phase is passed in as input to the successive stages. The system is modular and consists of five levels of processing, namely the complex words generator, basic phrase generator, complex phrase generator, domain parser and the combiner. The domain parser is the only domain dependent module of the system.

6 Overview of P-FASTUS Architecture

The P-FASTUS system is composed of the following levels of processing which will be described in more detail:

1. Tokenizer and Sentence generation
2. Parts of Speech Tagging (includes name tagging etc.)
3. Simple Phrase generation
4. Complex Phrase generation
5. Domain Events and template generation
6. Merging (Templates - Combiner)

The different levels of processing or phases are similar to those of FASTUS. The difference is in the template generation phase where additional measures have been taken to weed out irrelevant information and superfluity such as determiners, unnecessary adjectives etc. from the results. The system also includes a lexicon that is composed of a subset of words in the English language that covers all the words that occur in the texts that are tested. The lexicon also contains states such as Illinois and New York words which are annotated as states. A domain specific lexicon is also included which contains words that are important in recognizing an event or pattern of interest. The domain specific lexicon is domain dependent while the lexicon is not.

6.1 Tokenizer and Sentence generation

This is the first level of processing. It involves reading characters in as input and forming tokens to be processed for lexical analysis. Each character from the text files containing the job description is read. For example, from the following text:

```
The Estridge Group, the premier homebuilder in the Indianapolis
area, has a Sales Consultant position available at our Greystone Village
Community in Cicero, Indiana.
```

'T', 'h', 'e', etc. would all be read one at a time as characters and stored in a list. A list in prolog is polymorphic. It can store items of any type. Each list would contain a token, also known as a lexical item. For example, 'Estridge' would be a lexical item whose characters would be grouped together and stored in a list. In Prolog a character is identified by single quotation marks such as in 'a', where as the letter a in itself, without the quotation marks, would be a symbol. Each lexical item is recognized by the use of delimeters such as a non-breaking space, a period and a new line character. For the example given above, the output generated by the tokenizer is depicted below:

```
[['T','h','e'],['E','s','t','r','i','d','g','e'],['G','r','o','u','p'],
[','],['t','h','e'],['p','r','e','m','i','e','r'],
['h','o','m','e','b','u','i','l','d','e','r'],['i','n'],
['t','h','e'],['I','n','d','i','a','n','a','p','o','l','i','s'],
['a','r','e','a'],[','],['h','a','s'],['a'],['S','a','l','e','s'],
['C','o','n','s','u','l','t','a','n','t'],['p','o','s','i','t','i','o','n'],
['a','v','a','i','l','a','b','l','e'],['a','t'],['o','u','r'],
['G','r','e','y','s','t','o','n','e'],['V','i','l','l','a','g','e'],
['C','o','m','m','u','n','i','t','y'],['i','n'],['C','i','c','e','r','o'],
[','],['I','n','d','i','a','n','a'],['.']]
```

Every input is read as a character input including numbers. The list is traversed and characters that are numbers are converted into numbers. Non-breaking spaces are discarded. Other characters such as the period '.', comma ',', etc. are read in as single tokens as well. The output of the tokenizer, as seen above, is a list containing a list of tokens, which in turn is a list of characters or a number. At this level company suffixes such as Inc., Corp., etc. and salutations such as Mr., Ms., etc. are recognized. The period at the end of these suffixes is discarded to ease further processing of the lexical items. Words that contain a hyphen, '-', or a dot, '.', between them are also taken care of at this level. For example domain names such as google.com would be read in as ['g','o','o','g','l','e','.','c','o','m']. This is accomplished by the use of a simple finite state machine. The FSM reads in the tokens from the list. It moves from state 0 to state 1 when it encounters a token and from state 1 to state 2 which is the accepting state when it encounters a hyphen or a dot. It stays in state 2 if another token is encountered and returns to state 1 if another hyphen or

dot follows the next token read in. If the machine stops at the accepting state, then it combines the tokens together. The tokenizer also removes any text contained within braces such as "(inside)" from the input since they are deemed unimportant for the purposes of information extraction. One might argue that some text may contain relevant information within these braces. However, the frequency of such instances is low enough to allow us to simply disregard such linguistic constructs.

The output from the tokenizer is then passed on to the sentence generator that separates the list of tokens into sentences. Periods ('.'), colons (':'), exclamation marks ('!') and line breaks are used as delimeters for identifying the starting and end points of sentences in the list of tokens. Line breaks are also removed from the list at this level of processing. The output of the sentence generator would be a list of sentences which is in turn a list of the tokens that comprise a sentence. The input list of tokens is traversed and when a delimeter is encountered the list of tokens traversed thus far is inserted/appended to the list of sentences. The output for the sentence generator is as follows:

```
[[['T','h','e'],['E','s','t','r','i','d','g','e'],['G','r','o','u','p'],
[',],['t','h','e'],['p','r','e','m','i','e','r'],
['h','o','m','e','b','u','i','l','d','e','r'],['i','n'],
['t','h','e'],['I','n','d','i','a','n','a','p','o','l','i','s'],
['a','r','e','a'],[',],['h','a','s'],['a'],['S','a','l','e','s'],
['C','o','n','s','u','l','t','a','n','t'],['p','o','s','i','t','i','o','n'],
['a','v','a','i','l','a','b','l','e'],['a','t'],['o','u','r'],
['G','r','e','y','s','t','o','n','e'],['V','i','l','l','a','g','e'],
['C','o','m','m','u','n','i','t','y'],['i','n'],['C','i','c','e','r','o'],
[',],['I','n','d','i','a','n','a'],['.']]]
```

The tokenizer, as well as the sentence generator, is built using very simple finite state machines. For the tokenizer, the machine moves from the initial state 0 to state 1 when a stream of characters are read in and it moves from state 1 to the accepting state 2 when a delimeter is encountered. At the accpeting state, the input read in thus far that satisfies the pattern is grouped together as a lexical entry. Similarly, for the sentence generator instead of a stream of characters, a list of tokens are read in. The input of tokens moves the machine to state 1 and a token containing a delimeter for sentences moves the machine to the accepting state 2. Words forming a sentence are grouped together as sentences.

6.2 Parts of Speech Tagging

This level of processing tags possible company names, location and the different parts of speech that make up the sentences in the text.

The first step at this level involves the tagging of States such as 'California' and 'Illinois'. The list generated by the sentence generator is traversed and each list of lexical entry is matched with state-words in the lexicon. If they match, then the lexical entry in the list is tagged as a state.

e.g. `[state,['I','n','d','i','a','n','a']]`

At the second step of this level, possible company names, names and location are tagged. First names are tagged. This is done by checking if the a lexical entry starts with an upper case. If the lexical entry with an upper case is the first component of the sentence then the program checks if it is already in the lexicon as a determiner or adverb, etc.. If it is then it is not tagged as a name. A lexical entry in the middle of the sentence that contains upper case letters guarantees that it is a name.

e.g. 'The Estridge Group' is tagged as : `[['T','h','e'],`
`[name,['E','s','t','r','i','d','g','e'],['G','r','o','u','p']],...]`

The name tagging process uses a finite state machine as well. Lexical entries from the sentence list are read in one by one. When a lexical entry starting with an Upper Case is encountered, the machine moves from the initial state 0 to state 1. The machine stays at state 1 when more lexical entries with an upper case are encountered following the first one and it finally moves to the final and accepting state 2 when a any other lexical entry is read in as input.

Once the states and names have been tagged, possible company names and location are recognized and annotated. Finite state machines traverse the list searching for possible company names. An arbitrary number of lexical entries tagged as names followed by a company suffix such as 'Inc' or 'LLC', etc. are grouped together and tagged as company. Those that are not followed by a suffix are left annotated as names. Other patterns such as a name followed by a comma (',') and a suffix are also recognized and tagged as company.

For example, consider the text below:

`Redwood Toxicology Laboratory, Inc. is a Santa Rosa, California based company that specializes in drugs of abuse testing.`

Tagging company names and location yields the following result:

```
[[[company,['R','e','d','w','o','o','d'],['T','o','x','i','c','o','l','o','g','y'],
['L','a','b','o','r','a','t','o','r','y'],['I','n','c']],['i','s'],
['a'],[location,['S','a','n','t','a'],['R','o','s','a'],
['C','a','l','i','f','o','r','n','i','a']],['b','a','s','e','d'],
['c','o','m','p','a','n','y'],['t','h','a','t'],
['s','p','e','c','i','a','l','i','z','e','s'],['i','n'],
['d','r','u','g','s'],['o','f'],['a','b','u','s','e'],
['t','e','s','t','i','n','g'],['.']]]
```

Finally once possible company names, names and locations have been recognized by the IE system, the rest of the lexical items in the list of sentences are also tagged. These words are tagged as determiners, pronouns, verbs, nouns, adjectives, adverbs and prepositions. The list of sentences is traversed and the list of lexical entries are matched with entries in the lexicon. When a match is found then the lexical entry in the list is annotated as the corresponding part of speech.

The final output of the parts of speech tagging process is as follows:

```
[[[det,['T','h','e']],[noun,['E','s','t','r','i','d','g','e'],
['G','r','o','u','p']],[','],[det,['t','h','e']],
[adj,['p','r','e','m','i','e','r']],[noun,
['h','o','m','e','b','u','i','l','d','e','r']],[prep,['i','n']],
[det,['t','h','e']],[noun,['I','n','d','i','a','n','a','p','o','l','i','s']],
[noun,['a','r','e','a']],[','],[verb,['h','a','s']],[det,['a']],
[noun,['S','a','l','e','s'],['C','o','n','s','u','l','t','a','n','t']],
[noun,['p','o','s','i','t','i','o','n']],[adj,['a','v','a','i','l','a','b','l','e']],
[prep,['a','t']],[pro,['o','u','r']],[noun,['G','r','e','y','s','t','o','n','e'],
['V','i','l','l','a','g','e'],['C','o','m','m','u','n','i','t','y']],
[prep,['i','n']],[location,['C','i','c','e','r','o'],
['I','n','d','i','a','n','a']],['.']]]
```

One of the differences between the FASTUS system and the P-FASTUS system that I have developed is that the FASTUS system does not include the parts of speech processing level. This level was implemented by FASTUS but was later bypassed as it turned out to double the run-time without improving the accuracy of the IE system. In Prolog this may not necessarily be the case, especially since I have used lists to store the lexical entries. Since unification of lists takes a longer time than unification of symbols in Prolog, tagging lexical entries might lower the run time in the next level of procesing where the parts of speech are merged into simple phrases. In order to test this hypothesis, the code for the next phase could be rewritten so that the timing with and without parts of speech tagging could be compared. Due to the time constraint, I have not been able to pursue this matter. Nevertheless, it would be worthwhile to develop the code to test this hypothesis in the future.

6.3 Simple Phrase generation

The simple phrase generation level groups the different parts of speech into simple phrases that constitute linguistic constructs such as noun groups and verb groups. Please note that these are not really linguistic constructs such as noun phrases. They are words grouped together as a single constituent of the input by the P-FASTUS system in order to deal with pattern matching in the latter phases. It would be unnecessary and impractical to pattern match for specific information described in a sentence by matching every single word. In short,the words are grouped together to simplify pattern matching for identifying information of interest.

So at this stage, an adjective followed by a noun forms a noun group. Similarly, a determiner followed by a noun forms a noun group. An adverb followed by a verb forms a verb group as well. There are more combinations of parts of speech that form simple phrases. In essence, a grammatical rule can be established which evaluates whether a group of words forms a simple phrase or not. The following groups of words all form simple phrases:

a (det) highly (adj) qualified (adj) Engineer (noun) -> a highly qualified Engineer (noun group)

will (verb) work (verb) -> will work (verb group)

will (verb) drive (verb) slowly (adv) -> will drive slowly (verb group)

Using Finite State Automata, adjectives are first grouped together as adjective phrases. The machine moves from the initial state 0 to state 1 when it first encounters an adjective. It stays in state 1 if it keeps reading in more adjectives. When it reads in something other than an adjective, it finally moves to state 2 which is the accepting state and the adjectives are grouped together and annotated. Other rules, such as an adjective followed by a conjunction and an adjective forming an adjective group, are also adhered to.

Once the adjectives have been grouped together the noun groups are formed from determiners, nouns and adjective groups. A set of finite state automatons formed from finite state grammars are used to group the different parts of speech into noun groups. Verb groups are also formed from its constituents in a similar fashion.

The output of the simple phrase generator is shown below:

```
[[[noungroup,['T','h','e'],['E','s','t','r','i','d','g','e'],
['G','r','o','u','p']],[','],[noungroup,['t','h','e'],
```

```
['p','r','e','m','i','e','r'],['h','o','m','e','b','u','i','l','d','e','r']],
[prep,['i','n']],[noungroup,['t','h','e'],
['I','n','d','i','a','n','a','p','o','l','i','s'],
['a','r','e','a']],[','],[verbgroup,['h','a','s']],
[noungroup,['a'],['S','a','l','e','s'],
['C','o','n','s','u','l','t','a','n','t'],['p','o','s','i','t','i','o','n']],
[adj,['a','v','a','i','l','a','b','l','e']],[prep,['a','t']],
[pro,['o','u','r']],[noungroup,['G','r','e','y','s','t','o','n','e'],
['V','i','l','l','a','g','e'],['C','o','m','m','u','n','i','t','y']],
[prep,['i','n']],[location,['C','i','c','e','r','o'],
['I','n','d','i','a','n','a']],['.']]]
```

Any lexical entry that is not a part of a simple phrase is left unmodified. For example the word 'at' and the location 'Cicero, Indiana' in the example above are left unchanged.

6.4 Complex Phrase generation

At the fourth level, complex noun groups and verb groups that can be recognized are formed from the output generated by the simple phrase generator. They are recognized from domain-independent syntactic information. A noun group followed by a conjunction and another noun group or location are grouped together as noun group.

Consider the example below:

```
GenSys Software, a life sciences enterprise software products and
services company located in Santa Monica, CA, is looking for a Sr.
Software Engineer to join our team.
```

At this level the phrase 'a life sciences enterprise software products and services company' is grouped together as a noun group.

Prepositional phrases such as 'in' and 'of' are also attached to their head noun groups. In the example on the Estridge group:

```
The Estridge Group, the premier homebuilder in the Indianapolis
area, has a Sales Consultant position available at our Greystone Village
Community in Cicero, Indiana.
```

the phrases "the premier homebuilder in the Indianapolis area" and "Greystone Village Community in Cicero, Indiana" are grouped together as:

```
[[[noungroup,['T','h','e'],['E','s','t','r','i','d','g','e'],
['G','r','o','u','p']],[','],[noungroup,['t','h','e'],
['p','r','e','m','i','e','r'],['h','o','m','e','b','u','i','l','d','e','r'],
['i','n'],['t','h','e'],['I','n','d','i','a','n','a','p','o','l','i','s'],
['a','r','e','a']],[','],[verbgroup,['h','a','s']],[noungroup,['a'],
['S','a','l','e','s'],['C','o','n','s','u','l','t','a','n','t'],
['p','o','s','i','t','i','o','n']],[adj,['a','v','a','i','l','a','b','l','e']],
[prep,['a','t']],[pro,['o','u','r']],[noungroup,['G','r','e','y','s','t','o','n','e'],
['V','i','l','l','a','g','e'],['C','o','m','m','u','n','i','t','y'],
['i','n'],['C','i','c','e','r','o'],['I','n','d','i','a','n','a']],['.']]]
```

Finite State transducers for forming both complex noun groups and complex verb groups complete this task. Patterns are matched. When a noungroup followed by a preposition such as 'in' is followed by another noungroup then they are all grouped together as one single noun group. Similarly when a verb group is followed by more verb groups, that is it follows the rule - (verb group)*, then they are also grouped together as a single verb group. The '*' stands for kleene star which means that the element 'verb group' can occur any arbritrary

number of times. Like the preceeding levels of processing, other lexical entries that are rejected by the transducers are left unchanged at this level. The output at this level is thus a list of sentences that are comprised of complex linguistic structures.

6.5 Domain Event Recognition and Template generation

This is the only level which is domain-dependent. One of the goals of designing FASTUS and P-FASTUS was to make the system portable so that with little modifications the IE system could be used to extract pre-specified information from another domain of interest. Therefore, it is only the code at this level that needs to be modified in order to make P-FASTUS suitable for extraction of pre-specified information from another domain.

At this level, the output generated by the previous level is taken in as input and templates filled with information extracted from the text are built. The input to this level is a list of sentences annotated as complex phrases in the order they were formed. The first step at this level involves the tagging of trigger words, which is further used to identify linguistic constructs that is used to match against domain-specific patterns of interest. The input list from the complex phrase level is traversed, and unification is used to check if any of the words in the list unify with the trigger words that are stored in the domain-specific lexicon. Following are examples of words used as trigger words in the domain-specific lexicon:

1. For identifying linguistic constructs that may contain information on the company advertising the position:

   company, firm, we

2. For identifying linguistic constructs that may contain information on the position being advertised:

   looking, seeking, has, available

3. For identifying linguistic constructs that may contain information on the location of the firm:

   based, located

These words are saved as predicates with two arguments. The first argument is used to identify what type of information the subsuming linguistic construct may contain. The second argument is the word itself which is a trigger entry that is to be unified with lexical entries in the input list.

```
trigger(trig_company,['c','o','m','p','a','n','y']).
trigger(trig_position_verb_based,['b','a','s','e','d']).
trigger(trig_position_verb_located,['l','o','c','a','t','e','d']).
trigger(trig_position_verb_seek,['l','o','o','k','i','n','g']).
trigger(trig_position_open,['o','p','e','n','i','n','g']).
```

The domain-specific lexicon is not complete and trigger words that may exist in texts that have not been tested are not included in the knowledge base. The domain-specific lexicon, like the general lexicon, may never be complete, as there are far too many words in the English language that can be stored for the purposes of Information extraction and not all of it may be used. Furthermore, novel words are often coined. So, technically, no lexicon is ever complete. For the purposes of my project, I have tried to include just the right amount of words I would need for analyzing the texts that I have used. The lexicon can always and easily be expanded by adding more entries to the knowledge base by using `assert`. Although this feature has not been implemented, this feature of Prolog can be used to determine the type of the words not present in the lexicon and can also be used to tackle the problem of ambigious words. For example consider the word 'architect' not present in the current lexicon which could be either used as a noun or a verb. If the preceeding word is a determiner or an adjective, then it is likely that the word 'architect' is being used as a noun in this context. Thus, the assert feature could be used to add 'architect' as a noun in the lexicon or the knowledge base.

Once linguistic constructs that contain trigger words are recognized then the list of complex phrases can be used for domain specific pattern matching. Patterns are matched by Finite State Machines or Finite State Grammars. Finite State Grammars are used to extract the following information from the texts:

1. Name of the company or institution that is advertising the job position

Consider the following text:

```
The Estridge Group, the premier homebuilder in the Indianapolis
area, has a Sales Consultant position available at our Greystone Village
Community in Cicero, Indiana.
```

The Estridge group would be the name of the institution that is to be extracted from the text. The pattern above can be approximated by the following Finite State grammar:

```
Noun Group {Relative Clause} <has> Noun Group <available>
```

Here <has> and <available> are the trigger words that help identify that this pattern can be used to extract the name of the institution. The first Noun group that occurs before {Relative Clause} is the phrase or linguistic construct that needs to be extracted. The above pattern or Finite State Grammar can also be used to extract the position that is being advertised in the text. This is not the only FSM used to extract the name of the company or institution. A set of FSMs that approximate patterns are used to identify the information

required.

The following is another example of a FSM that is used to extract the name of the company or institution:

For the text:

```
Lanier Worldwide, Inc.  is a Fortune 500 company
```

Lanier Wordwide Inc is the information to be extracted as the name of the company or institution and the Finite State grammar to extract to match this pattern is as follows:

```
Company <is> <trig_company>
```

Here Lanier Worldwide, Inc. is tagged as company in the POS tagging level since it contains the company suffix. The phrase "a Fortune 500 company" is tagged as a construct containing the trigger word, "company". In any other domain of textual information, we could argue that sentences such as the one above could be referring to another company. However, for the domain of just Job Postings, it is apparant that such sentences refer to only the company of interest. Therefore it is safe to use such patterns in this domain.

2. The job position that is advertised in the text

Consider the following text:

```
GenSys Software, a life sciences enterprise software products and
services company located in Santa Monica, CA, is looking for a Sr.
Software Engineer to join our team.
```

Sr Software Engineer is the information that needs to be extracted from the text. The pattern used to extract this information from the above sentence is as follows:

```
Noun Group {Relative Clause}* <trig_position_verb_seek> Prep Noun Group
```

The trigger word in this sentence is "looking". The Noun Group following the preposition "for" which follows the trigger phrase is the linguistic construct that is extracted. In the above example, Sr. Software Engineer is the Noun Group that is extracted. Other information, such as the name of the company (which in this case is GenSys Software) is also extracted by this Finite State Machine. A set of FSM's approximated from finite-state grammars are also used to extract the position being advertised.

The following is another pattern used to extract the position being advertised:

Consider the example below:

```
RushTrade Group, a Dallas based Direct Access Trading firm, has
openings for Internet (inside) Sales Specialists.
```

In this example, "Internet Sales Specialists" is the information that needs to be extracted. The trigger word in this sentence is "opening" and the following pattern is used to extract the information:

```
Noun Group {Relative Clause}* <has> <trig_position_open> Prep Noun Group
```

Like the previous example, the Noun Group after the preposition is the linguistic structure that is extracted. Here, the verb "has" plays an important role in forming the pattern as well. Information on the company or institution is also extracted by this FSM.

Sentences that begin with a pronoun such as "We" that refers to the institution also hold information on the position that is being advertised. For example:

```
We are looking for a licensed physical therapist to work in our
prime health @ home department.
```

FSM's that handle such occurances of pronouns instead of Noun Groups or possible company or institution names have also been constructed. The following pattern recognizes the information for the position sought in the above example:

```
Pronoun <trig_position_verb_seek> Prep Noun Group
```

The Noun Group at the end of the pattern is the phrase that is extracted. Unlike some of the patterns discussed before, this pattern only recognizes the position that is being advertised and not the name of the company or institution that is advertising.

3. The location of the company or institution

The last item that is extracted by the P-FASTUS system is the location of the company or institution that is advertising the job position. One of the difficulties of this project has been to extract the location of the company or institution. The difficulty lies in the fact that the text may refer to either the location where work needs to be done or the location where the company is based. In most job postings, the location of the work is not specified explicitly. It is implicit in some cases that the location where the company or institution

is based is where the accepted applicant is required to work. Therefore, P-FASTUS does not distinguish between the two.

For sentences that explicitly state the location of the work, consider the example discussed earlier:

```
The Estridge Group, the premier homebuilder in the Indianapolis
area, has a Sales Consultant position available at our Greystone Village
Community in Cicero, Indiana.
```

The pattern that extracts the location, which is this case is "Greystone Village Community in Cicero, Indiana", has been discussed earlier. In this case, the exact location of the work area is obtained.

In contrast, consider the example below:

```
RushTrade Group, a Dallas based Direct Access Trading firm, has
openings for Internet (inside) Sales Specialists.
```

The text that contains this sentence does not explicitly state where the location of the work is. It is implicit from the job posting that the location is Dallas, since the company is based in that city. In order to be able to extract this information, the following FSM was built and coded into this level:

```
{Noun Group | Company} , Noun Group <trig_position_verb_based>
```

In this pattern, the Noun Group in between the comma and the trigger tagged phrase following it is the linguistic construct that is extracted as the location of the company or institution.

In FSM's where relative clauses exist, they are ignored if they do not contain the information that is being sought by that particular FSM. However, if the relative clause does contain information that is of relevance then other FSM's that match that pattern automatically pick up that information and include it into the template. The general template that is filled is as follows:

For the text:

```
The Estridge Group, the premier homebuilder in the Indianapolis area,
has a Sales Consultant position available at our Greystone Village
Community in Cicero, Indiana.  We are seeking a highly energetic individual
with excellent communication skills.

The Estridge Group has been named one of the "Eight Great" builders
to work for in America by Builder Magazine.  This position includes
```

```
a base salary of 36K, an opportunity to earn excellent commissions
and an outstanding benefits package.  We also offer a comprehensive
training program.
```

Template:

File Number: 2
Position: Sales Consultant
Company: Estridge Group
Company: Estridge Group
Position: -
Company: Estridge Group

The cascade of finite-state automatons generates multiple entries of information from different sentences that contain that information, which is then taken care of at the next level.

The template is stored as a list in the program and it looks like this:

```
[2,[position,[['S','a','l','e','s'],['C','o','n','s','u','l','t','a','n','t']]],
[company,[['E','s','t','r','i','d','g','e'],['G','r','o','u','p']]],
[location,[['G','r','e','y','s','t','o','n','e'],['V','i','l','l','a','g','e'],
['C','o','m','m','u','n','i','t','y'],['C','i','c','e','r','o'],
['I','n','d','i','a','n','a']]],[company,[['E','s','t','r','i','d','g','e'],
['G','r','o','u','p']]],[position,[]],[company,[['E','s','t','r','i','d','g','e'],
['G','r','o','u','p']]]]
```

In order to improve the accuracy of the P-FASTUS system, additional measures were taken at this level. Superfluous words such as determiners, adjectives and other irrevelant words from the information extracted were removed. For example the 'a' and 'position' in the 'a Sales Consultant position' are removed. Rules to remove all occurances of determiners were implemented. In order to remove the word position, 'position' was included in the domain-specific lexicon as a trigger word that needs to be removed from the information that is extracted. Other words include company, candidates and individual. It is important to remove occurances of such words since the FSM's extract linguistic constructs that are completely erroneous.

Consider the sentence below:

```
We are seeking a highly energetic individual with excellent communication
skills.
```

One of the FSM's of the system will read 'highly energetic individual' as the position. In order to correct this the word 'individual' will first be removed and

the remaining list would be checked to make sure that at least one remaining word is a noun. If none of the words are nouns or uppercased, which in this case the remaining words would be 'highly energetic', they are all removed. This helps to improve the precision of the IE system.

6.6 Combiner - Template Merging

The final level of the system involves the process of merging the structures or templates generated by the previous phase. The template generated by the domain dependent template generator may contain structures holding the same information. In some cases, one may hold more information than the other. For example from the sentences of the text below, four templates holding the information on the name of the company or institution are generated.

```
RushTrade Group, a Dallas based Direct Access Trading firm, has openings
for Internet (inside) Sales Specialists.  RushTrade is an Online Brokerage
firm that offers Proprietary Software solutions to active individual
traders around the world.

.....  RushTrade has strategic alliances with Tier
one firms in the online community providing a steady flow of warm inbound
leads.

The Ideal Candidate will possess the following:

.....  .................

RushTrade is offering a competitive compensation package of 50,000+
per year ............
```

The templates generated by the template generation level are as follows:

Template:

File Number: 5
Position: Internet Sales Specialists
Company: RushTrade Group
Location: Dallas
Company: RushTrade Group
Company: RushTrade
Company: RushTrade Group

At this level, these templates are merged and the following combined template is formed:

File Number: 5
Company: RushTrade Group
Position: Internet Sales Specialists
Location: Dallas

The template that contains the word "RushTrade" only is subsumed by the

template that contains the words "RushTrade Group". Once the structures have been merged by the combiner, the final step at this level then prints the content of the list.

The structure of the list that holds the extracted information is as follows:

```
[File_Number,[company, C],[position, P],[location, L]]
```

C, P and L are the lists that contain the lexical entries for each of the respective information sought by the system. Each file is assigned a number which is later used to print the name of the file along with the information extracted.

7 Testdata and Results

7.1 Testdata

The text files that were used to test the system were obtained from the following job sites:

1. Monster.com
2. Hotjobs.com
3. and Craigslist.org

10 files were repeatedly tested to build the FSM's. 10 new text files were used to the test the system's performance on novel texts. The testdata included a total of 20 text files.

7.2 Metric System

In order to compare and evaluate the performance of the IE systems, a metric system was developed. The principal measures used are recall and precision. "Recall is the number answers the system got right divided by the number of possible right answers". Recall measures how comprehensive the system is in its extraction of relevant information. The second measure, Precision, is the number of answers the system got right divided by the number of answers the system generated. In short, Precision measures the accuracy of the system.

In mathematical notation:

$$Recall = \frac{Numberofcorrectanswers}{Numberofpossibleanswers}$$

$$Precision = \frac{Numberofcorrectanswers}{Numberofanswersgeneratedbythesystem}$$

In addition to the two principal measures, a third measure called the F-Score is also computed. The F-Score is a combined measure and is defined as follows:

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

where, P is the Precision, R is the recall and β is a parameter encoding the relative importance of Recall and Precision. $\beta = 1$, means that they are weighted

equally. $\beta > 1$, means that Precision is more significant. $\beta < 1$, means that Recall is more significant. (Jackson 2002)

7.3 Results

From the first 10 test files the following output was generated:

Information extracted from file: ./testdata/test1.txt

company : Redwood Toxicology Laboratory Inc
position : NA
location : Santa Rosa California

Information extracted from file: ./testdata/test2.txt

company : Estridge Group
position : Sales Consultant
location : Greystone Village Community Cicero Indiana

Information extracted from file: ./testdata/test3.txt

company : IndX
position : qualified Controller
location : NA

Information extracted from file: ./testdata/test4.txt

company : Watsonville Community Hospital
position : licensed physical therapist
location : NA

Information extracted from file: ./testdata/test5.txt

company : RushTrade Group
position : Internet Sales Specialists
location : Dallas

Information extracted from file: ./testdata/test6.txt

company : Edison Schools Inc..
position : qualified Client Relationship Coordinator
location : NA

Information extracted from file: ./testdata/test7.txt

company : Lanier Worldwide Inc
position : NA
location : Atlanta

Information extracted from file: ./testdata/test8.txt

company : Lanier
position : NA
location : NA

Information extracted from file: ./testdata/test9.txt

company : NA
position : District Manager
location : NA

Information extracted from file: ./testdata/test10.txt

company : GenSys Software
position : Sr. Software Engineer
location : Santa Monica CA

The system was able to detect 21 correct answers out of a possible 26 correct answers. Out of the 21 answers generated all were correct. Therefore the system yielded a Recall of 80.7% and a Precision of 100% on these test files. However it is important to note that the system was built while testing for these test files. Therefore a new set of 10 files were tested on the system. On the new files, the system generated the following output:

Information extracted from file: ./testdata/untested/test11.txt

company : Major Financial NYC
position : strong C++ developers
location : NA

Information extracted from file: ./testdata/untested/test12.txt

company : NA
position : NA
location : NA

Information extracted from file: ./testdata/untested/test13.txt

company : Successful mut
position : NA

location : NA

Information extracted from file: ./testdata/untested/test14.txt

company : Friday
position : General Dentist
location : NA

Information extracted from file: ./testdata/untested/test15.txt

company : Manpower
position : NA
location : NA

Information extracted from file: ./testdata/untested/test16.txt

company : Our client
position : NA
location : NA

Information extracted from file: ./testdata/untested/test17.txt

company : Our dynamic Concord
position : Mechanical Engineer
location : NA

Information extracted from file: ./testdata/untested/test18.txt

company : real estate industry
position : experienced bookkeeper or assistant controller
location : NA

Information extracted from file: ./testdata/untested/test19.txt

company : Authorize
position : NA
location : NA

Information extracted from file: ./testdata/untested/test20.txt

company : Internet
position : New Media Account Executives
location : NA

The results from these test files were not very encouraging but satisfactory. The system generated only 6 correct answers out a a possible 21 correct an-

swers. The system also generated a total of 14 answers. Recall was about 29% and Precision was about 43%.

For the total 20 test files that were used, Recall was 57.4% and Precision was 77%. The F-score when Recall and Precision were given equal weight, that is when $\beta = 1$, was 65.77.

P-FASTUS took 48.079 sec to process the 20 texts on a Pentium III 500 MHz processor. The texts contain 4506 words in 356 lines which translates into approximately 5623 words per minute. At the MUC-4 evaluation, FASTUS processed 2375 words per minute on a SPARC-2 station. It took 15.9 minutes to process 100 texts and was capable of processing 9000 texts per day (Roche 1997). It is currently impossible to compare two systems since they operate on different domains of interest and the number of information extracted varies between the two. Morever, they have been tested on different machines that run on processors of different speed. However, the results do indicate that the P-FASTUS system is fairly efficient. It is important to note that the speed of the P-FASTUS system does depend on the size of the lexicon as well. The lexicon for P-FASTUS is relatively small at the moment and the larger it gets the longer the program may take to process the texts. However, better heuristic measures could be taken in the future to ensure faster access to the desired entry in the lexicon. For example, a 'case statement' type of feature could be implemented that would let the IE system avoid the inspection of words that are not bound to unify.

8 Conclusion

The P-FASTUS system did not perform well when tested on new test files. On new files, it yielded a Recall of 29% and Precision equal to 43%. These results are comparable to the scores of the original MUC-5 FASTUS system that yielded a Recall of 34% and Precision equal to 56% (Roche 1997). As mentioned earlier, when constructing an IE system using the knowledge engineering approach, the performance of the system depends heavily on the ability of the knowledge engineer to construct FSM's that can comprehensively capture all or most of the patterns that are prevalent in these "job posting" text files. Developing FSM's, for pattern recognition, that are comprehensive and accurate require time to develop. Unfortunately due to the limited amount of time available to complete this project, I was unable to construct more accurate and comprehensive FSM's to match potential patterns containing relevant information. The template generation or domain event recognizer level is the one that needs further improvement in order to make the system more accurate and comprehensive. With more time, I am confident that the performance of the system can be improved.

The advantages of using Prolog for developing IE systems are clear. Unification allows for a simple way to compare and match word inputs with lexical entries in order to derive their type. It provides an efficient way to match two items of interest and is used in every level of processing in P-FASTUS. Prolog lists are convenient data structures to use for Natural Language Processing as well. Since words are stored as lists of characters, detection of names in the text is easily accomplished by using a FSM that searches for upper case letters in the words. Unification allows for an easy way of matching the contents of the lists. One of the most useful features of Prolog is the depth first search algorithm it uses when searching for a possible answer. This feature has been most useful when implementing the FSMs that match patterns of interest, i.e. the domain dependent level that looks for patterns of interest. Each pattern of interest can be coded as a series of FSMs. When one pattern fails, others are checked for a possible match. This gives the FSMs a non-deterministic characteristic which is required for matching different patterns of interest. The ability to modify the program at run time provides additional tools to make the system more portable and also allows for the possibility of handling the problem of ambiguity better. As mentioned earlier, this feature unfortunately has not yet been implemented due to time constraints.

With respect to the speed of the P-FASTUS system, it is impossible to compare it with FASTUS since they deal with different domains of interest. SRI had originally developed automatic domain learning capabilities for FASTUS and with this in mind I had hoped to obtain a copy of the program so that I could compare the two systems. Unfortunately, I was unable to acquire a copy and the question of whether Prolog, a logic programming language, provides a faster way of extracting information remains unanswered at the moment. However,

we can argue that P-FASTUS is a relatively efficient IE system that processes approximately 5623 words per minute.

P-FASTUS, like FASTUS is not a text understanding system. It uses a cascade of non-deterministic finite state automatons. In essence, it is a pattern matching program implemented in SICStus, a constraint programming language that facilitates natural language processing.

9 Acknowledgements

Special thanks to my supervisor, Dr. Hans-Joerg Tiede, for all the support and guidance. I would also like to thank the other members of the committee: Dr. Susan Anderson Freed, Dr. Paul Kapitza and Dr. Melyvn Jeter.

Thank you!

10 References

1. Appelt, Douglas E.; Hobbs, Jerry; Bear, John; Israel, David; Kameyama, Megumi; Tyson, Mabry. "SRI: Description of the JV-FASTUS System Used for MUC-5". Artificial Intelligence Center, SRI International: 1993.

2. Appelt, Douglas E.; Hobbs, Jerry; Bear, John; Israel, David; Kameyama, Megumi; Tyson, Mabry; Stickel, Mark. "FASTUS: A Cascaded Finite-State Transducer for Extractiong Information from Natural-Language Text". Artificial Intelligence Center, SRI International: 1995.

3. Appelt, Douglas E.; Israel, David. "Introduction to Information Extraction Technology". Artificial Intelligence Center, SRI International: 1999.

4. Covington, Michael A. "Natural Language Processing for Prolog Programmers". Prentice Hall: 1994. pp. 1-12.

5. Dale, Robert; Moisl, Hermann; Somers, Harold. "Handbook of Natural Language Processing". Marcel Dekker, Inc. : 2000. pp. 241-258.

6. Greenlaw, Raymond; Hoover, H. James. "Fundamentals of the Theory of Computation". Morgan Kaufmann Publishers, Inc.: 1998, pp. 85-112.

7. Jackson, Peter; Moulinier, Isabelle. "Natural Language Processing for Online Applications". John Benjamins Publishing Company: 2002. pp. 75-113.

8. Marriott, Kim; Stuckey, Peter J. "Programming with Constraints". The MIT Press: 1999. pp. 11.

9. MUC-6. "The Sixth Message Understanding Confere". <http://www.cs.nyu.edu/cs/faculty/grishman/muc6.html>.

10. Pereira, Fernando; Wright, Rebecca. "Finite-State Approximation of Phrase-Structure Grammars". In 29th Annual Meeting of the Association for Computational Linguistics: 1991.

11. Roche, Emmanuel; Schabes, Yves. "Parsing with Finite-State Transducers". The MIT Press: 1997. pp. 1-62.