Learning information extraction patterns

by

Fajun Chen

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Major Professor: Vasant Honavar

Iowa State University

Ames, Iowa

2000

Graduate College
Iowa State University

This is to certify that the Master's thesis of

Fajun Chen

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I am grateful to the many people who helped me in my research and the writing of this thesis.

I would like to give my special thanks to Dr. Vasant Honavar for his guidance, patience, mentoring, and support during the preparation of this thesis. It is he who led me to the exciting field of information extraction. More importantly, he taught me how to do research. His insightful advice has been invaluable during the entire course my graduate study in Iowa State University. He is a great advisor.

I would also like to thank my committee members for their help and advice: Dr. Les Miller and Dr. Drebb Dobbs.

I am indebted to Dr. John Riley, who patiently reviewed my thesis and helped me correct many grammatical errors. Weiyi Chen helped me review chapters 2 and 3, and made several helpful suggestions.

I would like to thank Rushi Bhatt and Tarkeshwari Trivedi for their help with LaTex for formatting this thesis, and to Arun David Raghavan for helping me with some aspects of Java.

ABSTRACT

The rapid growth of online texts call for systems that can extract relevant information. Many information extraction systems have been developed using the knowledge engineering approach, which is often time-consuming, laborious, and of no portability. A more promising direction is to apply machine learning techniques to information extraction.

A complete Information Extraction (IE) system, IEPlus, has been developed for exploring various design issues. Fine-grained semantic units were defined, and a strategy for semantic resolution was proposed in IEPlus. An enhancement for rule evaluation based on case frame matching was implemented in IEPlus. A rule firing strategy was also presented in IEPlus, which prioritizes the most specific rule in terms of the number of slots extracted. Experiments on the Rental Ads domain demonstrated the effectiveness of the IEPlus system. IEPlus is highly flexible resulting from its object-oriented design, and has the capability of exploring various issues in information extraction system design.

CHAPTER 1. INTRODUCTION

1.1 Information Extraction

1.1.1 Introduction to Information Extraction

We live in an age in which, more than ever before, individuals are overwhelmed by a deluge of data. Much of this data is textual in nature. Some examples include electronic mail (e-mail), articles posted to electronic news groups, documents on the web, scientific articles stored in digital libraries, electronic newspapers, transcripts of radio and television news programs, etc. Modern web browsers or some text retrieval tools or digital assistants can help us perhaps even selectively, proactively, and reactively retrieve such data. However, retrieving the relevant documents is only the first step. Translating our ability to gather, store, and access large amounts of data in digital form into fundamental advances in decision making calls for the development of sophisticated tools for information extraction, and data driven knowledge discovery. In the absence of such tools, information relevant for effective decision making are likely to be overlooked or ignored, potentially at great cost. This motivates us to explore various strategies for automated and customizable extraction of useful information from text. Such information, once extracted, may be used to support decision making, stored in a structured database for future use, summarized, or analysed (using data mining algorithms) to discover useful knowledge about the domain.

1.1.2 The Goal of Information Extraction

The goal of Information Extraction (IE) is to extract from a collection of documents relevant facts such as events, entities, or relationships. For example, IE for rental data may be needed

to extract the information about neighborhood, number of bedrooms, price, etc. IE for job news group might involve identifying job title, skills required, salary, etc. A sample training instance from the Rental Ads domain (Sod99) is shown as follows:

```
@S[
  BALLARD - Deluxe 1 br $550/2 br $600.
  Jim 206-781-1300.  <br>
  <i> <font size=-2> (This ad is from 08/02/97 to 08/03/97.)
  </font> </i> <hr>
]@S
@@TAGS Rental {Neighborhood BALLARD} {Bedrooms 1 br} {Price $550}
@@TAGS Rental {Neighborhood BALLARD} {Bedrooms 2 br} {Price $600}
```

In this example, the text embraced by "@S[" and "]@S" is the raw text to be extracted, and the part beginning with "@@TAGS" is the template annonated by humans. The example contains two case frames. Each case frame consists of three slots, Neighborhood, Bedrooms, and Price respectively.

The origin of IE can be traced to DARPA's MUC program (ARP92), where newspaper and newswire texts such as Latin American terrorist incidents were used to evaluate various IE systems. For each evaluation, each attending team receives a set of texts from a prespecified domain associated with the annotated templates (we call this set a training set). Each template annotated by humans contains relevant information in the corresponding text, which is usually composed of one or multiple case frames. Each attending system is adapted to this training set, generating extraction patterns. After that, these systems are evaluated on the same test set of previously unseen texts. Performance metrics of precision and recall are used for system comparison.

The extracted facts are usually entered automatically into a database, which can then be used for on-line access, data mining, text summary, etc. For example, after applying an IE system as the translator on structured or semi-structured text (it is termed as Wrappers in literature), we can view it as a structured information source and query it using a uniform

query language. IE techniques are finding application in sophisticated web search techniques for heterogeneous information integration, Web knowledge bases (CFMe97), news group query systems (TMT97), weather forecasting (Sod97b), restaurant information systems (MMK99).

1.1.3 The Difficulties of Information Extraction

Information Extraction is a difficult problem which shares some of the challenges of natural language understanding (AI99) including efficient parsing, ambiguity resolution, discourse structure reasoning, language semantics, etc.

There are different ways of expressing the same fact [1]:

- BNC Holdings Inc named Ms G Torretta as its new chairman.
- Nicholas Andrews was succeeded by Gina Torretta as chairman of BNC Holdings Inc.
- Ms. Gina Torretta took the helm at BNC Holdings Inc.
- After a long boardroom struggle, Mr Andrews stepped down as chairman of BNC Holdings Inc. He was succeeded by Ms Torretta.

IE system should be able to generate the same template (ie. tags) out of these different text as follows:

Succession event {Organization: BNC Holdings Inc}
{PersonIn: G Torretta} {Position: chairman}

Notice that in the example above, relevant information is distributed among several sentences. IE system should be able to merge the relevant fractions in different sentences together to form meaningful events.

There might be some subtle nuances of meaning in the text. So IE is difficult even for humans (Sod99; AI99). For various IE tasks, different human annotators may just agree on only 60-80% of the annotations.

[1] http://www.dcs.shef.ac.uk/research/groups/nlp/extraction/

1.1.4 Information Extraction in Context

IE differs from Information Retrieval (IR) and Natural Language Understanding (NLU) (AI99; All95). Typically, IR involves searching and retrieving from a collection of documents a subset which is relevant to a query in terms of keyword matching (or some variants such as stemming). The functionality of NLU is hard to characterize and evaluate, but usually it is more sophisticated. There is no clearly-cut boundary among IE, IR, and NLU. However, IE can be viewed as a task that lies between IR and NLU from the perspective of the complexity of functionality required. In particular,

- IE systems typically have to go beyond naive keyword matching (as in IR) which results in retrieval of entire documents.
- IE systems typically fall short of in-depth text understanding.
- IE systems are more ambitious than IR systems and less ambitious than text understanding systems in terms of the nature of information that provides the end user. They are typically designed with some specific domain in mind (e.g., wall street journal articles, scientific abstracts, rental advertisements) with the goal of understanding text only to the extent necessary to fill the slots in a structured template that captures the relevant information.

1.2 The Architecture of Information Extraction Systems

Although a variety of architectures have been proposed fro IE systems, most of them share the following basic modules for: tokenization, lexical analysis, syntactical analysis, and domain-specific processing. These basic components are shown in Figure 1.1 (Car97; AI99).

First, raw text is tokenized into words, the basic unit of linguistic structure. Lexical analysis usually follows, which identifies Part of Speech (POS) (Bri94), word sense, semantic classes (a kind of complex words) such as date, location, price, and other lexical items.

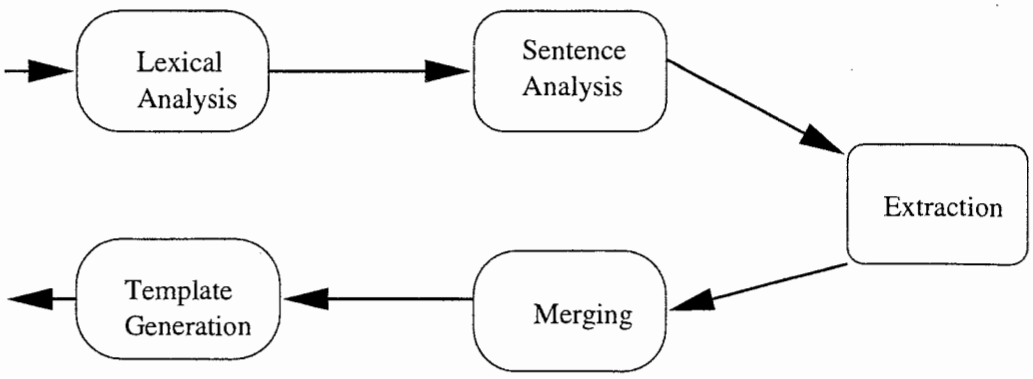

Figure 1.1 Architecture of IE Systems

The second phase involves sentence analysis which is composed of syntactic analysis or parsing. Noun groups, verb groups, prepositional phrases, and other constituents of a sentence are identified, the sentence structure is determined in this phase. Some systems such as FASTUS (Hob97) also recognize more complex noun groups and verb groups for the sake of simplifying extraction pattern (rules) in the subsequent phase. Partial parsing is preferred to full parsing in this step since we only concern the structures around relevant information. Full parsing, although it has been widely used in the systems of natural language understanding, often leads to bad performance in IE systems (HAe90; HAe92).

The third phase involves domain-specific extraction. Extraction pattern is applied to extract relevant information. Extraction patterns are specified by domain experts in some handcrafted systems, while they can also been learned automatically by corpus-based machine learning algorithms. This thesis will focus on extraction pattern learning algorithms.

The next phase is the merging phase, also called coreference resolution, or anaphora resolution. As mentioned in the last section, the entities in different sentences might refer to the same object. The ability to identify and merge the entities distributed in different sentences together might increase the performance of IE systems.

The last phase of IE is to generate the template, which is the output of an IE system. Different events are identified and each event is filled with the information extracted by previous phases.

1.3 Relevant Terminology and Definitions

1.3.1 Semantic Class and Semantic Hierarchy

A semantic class is a set of terms which belongs to an equivalence class, embodying the meaning of a concept. Semantic classes can be represeted in several ways. Disjunctive form and structural form are among the commonly used representations. For instance, The semantic class Bedrooms in the Rental Ads domain (Sod99) can be defined as:

$$\text{Bedrooms} = \text{"brs"}|\text{"br"}|\text{"bdrm"}|\text{"bd"}|\text{"bedrooms"}|\text{"bedroom"}|\text{"bed"}$$

which is a list of disjunctive terms, while semantic class DIGITS is in the structural form as follows according to regular expressions (Sip97):

$$\text{DIGITS} = [0\text{-}9]+$$

A domain-specific semantic class is usually created manually by a domain expert, which may not be a complete or perfect listing, but it helps an IE system generalize its extraction pattern beyond the tokens in its training instances.

Stand-alone semantic classes are not helpful from the point of semantic representation. Hence, semantic classes are typically tied together to form semantic net (RK91) or a semantic hierarchy. If we view a semantic class as a node, then a semantic net is a graph with a set of nodes connected to each other by a set of labeled arcs, which represents the relationships among the nodes.

WordNet (Mil95) is a domain-independent lexical database of about 57,000 words containing a semantic hierarchy in the form of hypernym links. A semantic class is represented by a *synsets* in WordNet. For instance, the English word *board* can signify either a piece of lumber or a group of people assumbled for some purpose. The synsets, {board, plank} and {board,committee} can designate two different concepts. Each word typically has more than one synset to which it belongs. The synsets in WordNet are connected by links of various types, including synonyms, antomyms, meronyms, holonyms, hyponyms, and hypernyms. The semantic hierarchy is formed by hyponym and hypernym links. Hyponym links indicate semantic subclasses while hypernym links indicate semantic superclasses. The semantic hierarchy

implemented in WordNet would greatly faciliate the semantic generalization and specialization in extraction rule learning process.

1.3.2 Design Issues in Knowledge Representation

Knowledge representation is one of the key concerns of Artificial Intelligence (AI) (RN95; RK91). It plays an important role in automated inference and knowledge discovery. The choice of the knowledge representation language determines the types of entities and relationships that can be represented, the efficiency of inference, the comprehensibility of representation and inference, the types of reasoning that are allowed, etc (DSS93). Hence, an appropriate choice of knowledge representation is essential for successful learning of information extraction patterns. Regular languages and first order predicate logic are some of the commonly used knowledge representation formalisms for information extraction patterns.

We define regular languages as the smallest class of languages which contains all finite languages and closed with respect to union, concatenation and Kleene closure. Regular languages are useful tools for recognizing patterns in data (Sip97). It has been used in speech processing and in optical character recognition. Regular language is compact yet expressive that it is suitable for representing the extraction patterns in IE system for many domains although it fails in representing recursive structure inherent in some domains. A limited form of regular language is used in IEPlus and WHISK as a good medium of knowledge representation.

Predicate logic is a logic which concerns not only with sentential connectives but also with the internal structure of atomic propositions. First Order Predicate Logic (FOPL) (RN95; RK91) considers both predicates (or relations) and individual elements. Atomic sentences are constructed by applying predicates to individual elements, and quantification is permitted only over the individual elements.

The languages represented by FOPL is more expressive than regular languages. Thus, richer constraints can be incorporated into an IE system by choosing FOPL as its representation of extraction patterns. For instance, FOPL was used in SRV (Fre98) and Rapier (CM99). The major problem with FOPL is the time and space complexity involved.

1.3.3 Learning Algorithms

A varity of approaches to machine learning are available in the literature (Mit97). Of particular interest are inductive learning approaches that produce general hypothesis from data. Inductive learning differ from each other in terms of choice of knowledge representation (e.g., rules, grammars, etc.) and the search strategy used to identify a hypothesis from data. Compression and covering are two basic strategies used in rule learning systems (Cal98). Systems that use compression conduct a specific to general search, trying to compress the rule set learned.

Systems that use covering strategy include AQ (Mic73),CN2 (CN89),FOIL (Qui90), etc. They work as follows:

1. Start with an empty rule set.
2. Select a positive instance or a set of positive instances from a training set
3. Find the candidate rules which can cover this instance or this set of instances.
4. Select the best rule among these candidate rules according to some criteria which is usually a tradeoff between generality and compactness.
5. Remove the instances covered by the best rule from the training set.
6. If the training set is empty, stop. Otherwise, repeat step 2 to 5.

Covering algorithms tend to be more efficient during search than compression algorithms because they don't learn rules for instances that have been covered by existing rules, but the rules learned by covering algorithms are more specific since there is no process for subsuming existing rules with more general ones. One way to make it up is to design specific rule firing strategy to limit the rules applicable for an instance.

Similar to WHISK(Sod99), IEPlus developed for this thesis uses a covering algorithm with enhancements for rule specialization, rule firing strategy, etc.

1.4 The Goal of this Thesis

This thesis focuses on algorithms for automated learning of information extraction patterns. A review of some existing approaches to learning information extraction patterns is given in **Chapter 2**.

Learning information extraction patterns poses a number of interesting and challenging questions:

- **What kind of preprocessing is required by an extraction pattern learner?**
- **What kind of semantic and syntactic constraints should be used for an extraction pattern learner?**
- **What kind of learning algorithm should be designed for an extraction pattern learner ?**

IEPlus, a specialized descendant of WHISK (Sod99), is designed to explore these interesting problems. IEPlus is distinguished from similar IE systems in the following aspects:

- **The role of lexical analysis:**

  Since most of the online documents are semi-structured, which may not be gramatically correct, syntactic analysis is not always possible for these data. Lexical analyzer in IEPlus is implemented by specialized tools. It is the only component in the system where domain-specific knowledge is hand-coded. Separating domain-specific component from domain-independent components faciliates the exploring of various choices in the phase of lexical analysis.

- **The role of semantic and syntactic contraints:**

  The choice of semantic or syntactic constraints determines the granularity of extractions, the robustness, and the performance of an IE system. Hybrid semantic units and semantic resolution are implemented in IEPlus, and a limited form of regular languages are used to represent syntactic constraints.

- **The role of rule specialization, rule firing, and rule evaluation:**

  The design of rule learning algorithms is of great importance for an IE system. Similar to WHISK, a top-down covering algorithm is used in IEPlus. However, IEPlus enhances it in various aspects including rule specialization, rule firing strategy, and rule evaluation strategy.

1.5 Organization

The rest of this thesis is organized as follows. **Chapter 2** presents a survey of extraction pattern learning algorithms used in various IE systems, and highlights some of the improvements over WHISK that are incorporated in IEPlus. **Chapter 3** describes some of the key design choices and implementation details of IEPlus. **Chapter 4** presents the experimental evaluation of IEPlus on the Rental Ads domain. **Chapter 5** concludes with a summary and a brief discussion of some directions for further research. The sample training instances, complete JLex specification, and the sample rules generated, are presented in **Appendix A, B, C** respectively.

CHAPTER 2. LEARNING INFORMATION EXTRACTION PATTERNS

2.1 Overview

As defined in (Car97), good extraction patterns are "those that are general enough to extract the correct information from more than one sentence but specific enough to not apply in inappropriate contexts".

Generally speaking, there are two approaches to coming up with information extraction patterns for IE: the Knowledge Engineering Approach and the Machine Learning Approach (AI99). In the Knowledge Engineering Approach, the extraction patterns used in the system are designed and specified by a "knowledge engineer". The design of knowledge base or the so-called expert systems, this knowledge engineering can be labor-intensive, error-prone, and time-consuming. In the Machine Learning Approach, the extraction pattern is generated by a learning algorithm which identifies the essential regularities useful for information extraction from a training set that consist of suitably annotated text. Thus, designing information extraction patterns for different domains reduces to generating suitable training sets for the respective domains. Extraction pattern learning algorithms differ in terms of the nature of the text (e.g., unstructured text, HTML, semi-structured text, etc.) syntactical/semantic features used, the expressive power of extraction patterns, number of training examples required, language preprocessing required, time/space efficiency, effectiveness (recall and precision), etc. However, they do share general architecture as shown in Figure 2.1.

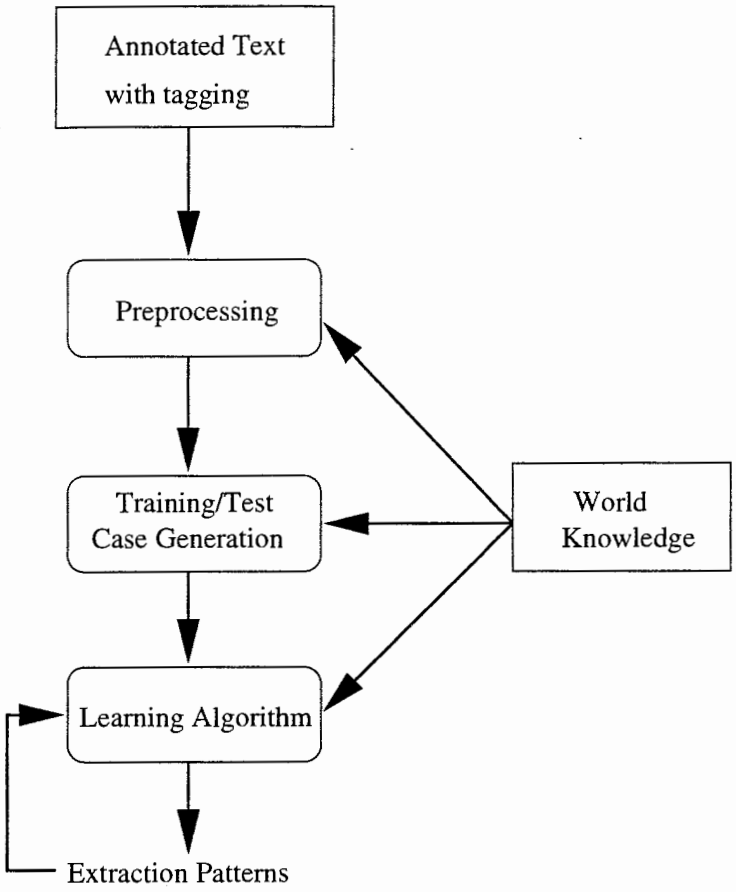

Figure 2.1 IE Extraction Pattern Learning Diagram

2.2 Learning Extraction Patterns for Information Extraction from Free Text

Free text is gramatically complete plain text, so syntactic and semantic preprocessing typically help in improving the performance of IE systems.

2.2.1 AutoSlog

AutoSlog was one of the earliest IE systems for extraction pattern learning (Ril93). One or multiple extraction patterns are generated for each concept to be extracted, which are in the form of concept nodes. Each concept node has a name of the concept to be recognized, a trigger for activating the pattern, a position specifying the syntactical location (subject, direct object, indirect object, etc.) of the concept in an input sentence, the constraint to be satisfied,

and the enabling condition to be met such as active-voice, passive-voice, or other linguistic contexts. For example, the concept node for extracting successor's name from the sentence:

BNC Holdings Inc named Ms G Torretta as its new chairman.

is as shown in Table 2.1.

Table 2.1 AutoSlog Concept Node: an Example

Concept: Position Succession
Trigger = "named"
Position = direct-object
Constraints = (person's name)
Enabling Conditions = (active-voice)

The learning of concept nodes is achieved by specializing one of thirteen domain-independent linguistic patterns. First, the sentence containing the name to be extracted is located. Then, this sentence is fed to a partial parser, which can identify the subject, direct object, and other linguistic constituents. Thirteen linguistic patterns are applied in order to get the parsed result. The first linguistic pattern fired is used to generate the extraction pattern (concept node). In our example, the pattern is "<active-voice-verb> <direct-object>", which can generate the concept node in Table 2.1 as a specialied form of the template concept node (Car97) shown in Table 2.2.

Notice that the input sentence to AutoSlog has to be preprocessed by syntactical analyzer and semantic analyzer, and the output of AutoSlog are single-slot rules (concept nodes), which are passed to human experts for perusal.

Table 2.2 AutoSlog Concept Node Template: an Example

Concept: <slot type> of <target-np>
Trigger = "<verb> of <active-voice-verb>"
Position = direct-object
Contraint = (<semantic class> of <concept>)
Enabling Conditions = (active-voice)

2.2.2 CRYSTAL

CRYSTAL (SFAL95; Sod97a) is able to learn multi-slot rules (concept nodes) from free text. Similar to AutoSlog, CRYSTAL requires precise syntactical analysis and semantic analysis for input text, which can identify and categorize the syntactical constituents (subject, object, prep phrase, etc.) and semantic classes as extraction features.

An example concept node of CRYSTAL for extracting the organization, person, and position from the sentence below:

- BNC Holdings Inc named Ms G Torretta as its new chairman.

is shown in Table 2.3 (following the format in (MMK99)).

Table 2.3 CRYSTAL Concept Node: an Example

Concept type: Position Succession
SUBJECT: Classes include: <Organization name> Terms include: Inc Extract: organization
VERB: Root: NAME Mood: active-voice
DIRECT OBJECT: Classes include: <Person name> Terms include: Ms Extract: person
PREP-PHRASE: Prep: AS Classes include: <Position name> Extract: position

The extraction pattern of CRYSTAL is much more expressive than AutoSlog because the trigger of CRYSTAL is no longer limited to a single trigger word or local linguistic context, it can be any sequence of words or any modifier of semantic class (Car97). CRYSTAL uses a bottom-up covering algorithm, which is a kind of inductive learning algorithms starting from learning specific rules and continuing the generalizing by unification until negative examples are covered or the error threshold is exceeded.

By using Webfoot (Sod97b) as an additional preprocessing step, CRYSTAL can extract relevant information from HTML sources. The trick here is to partition a web page into small sections according to page hierarchy cues. CRYSTAL can process each small section of a web page as free text.

2.2.3 LIEP and PALKA

LIEP (Huf95) can generate multi-slot patterns from free text. LIEP can't extract single slot because its extraction of a slot is with regard to the syntactic context of other slots. For instance, for extracting the organization, person, and position from the following sentence:

- BNC Holdings Inc named Ms G Torretta as its new chairman.

The extraction pattern in Table 2.4 is generated by LIEP following the format in (MMK99).

Table 2.4 LIEP Extraction Pattern: an Example

Event: Position Succession
noun-group(ORGANIZATION,head(isa(Organization name)))
verb-group(VG,type(active-voice),head(named))
noun-group(PERSON,head(isa(Person name)))
preposition(PREP,head(as))
noun-group(POSITION,head(isa(Position)))
subject(ORGANIZATION,VG)
verb-object(VG,PERSON)
object-prep(PERSON,PREP)
prep-object(PREP,POSITION)⇒
event(organization(ORGANIZATION),person(PERSON),position(POSITION))

In Table 2.4, syntactic constraints and semantic constraints are employed. For example, the syntactic form of the sentence is subject-verb-object-prep-object phrase, ORGANIZATION is in the semantic class of "Organization name". One drawback of LIEP is that the rules are induced only from positive training instances, which leads to low performance on negative instances. Some key word filtering is performed for filtering out irrelevant text (negative instances).

PALKA (KM95) is quite similar to AutoSlog except the inclusion of concept hierarchy and semantic class hierarchy, which guide the generalization and specialization of extraction

patterns. An induction learning algorithm which is similar to Mitchell's candidate elimination algorithm is used. Given a new instance, a rule is either generalized to cover the positive instance or specialized to exclude the negative instance. In concept hierarchy, generaliztion is through replacing a semantic class with an ancester, while specialization is through the substitution of a semantic class with its child node.

2.3 Learning Extraction Patterns for Information Extraction from Structured Text

Some text data online are highly structured such as the web pages automatically generated by programs, which often exhibit regular starting and ending delimiters.

An independent branch of research for learning extraction pattern for structured text is wrapper induction, where wrapper is a domain-specific procedure translating a structured information source into the equivalent of database.

2.3.1 WIEN

WIEN (KWD97) is a wrapper induction environment designed for information extraction. It assumes that there is a unique multi-slot rule applicable for all documents. The features used for extraction include only the delimiters right before and after the relevant part of the text. No semantic class is used.

The HLRT wrapper class, which is efficiently learnable yet reasonably expressive, was proposed. A HLRT wrapper class is specified by

$$< h, t, l_1, r_1, \ldots, l_k, r_k >$$

where label h marks the end of the header, label t marks the start of the tail, and each pair of labels l_i and r_i delimits the field to be extracted.

The learning process is to search for appropriate values for the $2K + 2$ parameters in the space of all possible combinations. The search problem can be decomposed into 3 independent subproblems: searching for

- right delimiter
- left delimiter except the first left delimiter l_1
- header, tail delimiter and first left delimiter l_1

Considering the web page in Table 2.5(KWD97):

Table 2.5 A Web Page for Wrapper

$< HTML >< TITLE > CountryCodes < /TITLE >$
$< BODY >< B > CountryCodes < /B >< P >$
$< B > Congo < B >< I > 242 < /I >< BR >$
$< B > Egypt < B >< I > 20 < /I >< BR >$
$< B > Belize < B >< I > 501 < /I >< BR >$
$< B > Spain < B >< I > 34 < /I >< BR >$
$< HR >< B > End < /B >< /BODY >< /HTML >$

The extraction pattern for extracting country name and country code is obtained from a specialized HLRT template, which is detailed below:

$$(h =' < P >', t =' < HR >', l_i \in \{' < B >', ' < I >'\}, r_i \in \{' < /B >', ' < /I >'\})$$

that is,

(1) we skip past the first occurence of $< P >$ in the page,

(2) if the current starting delimiter l_1 is before $< HR >$ in the page, extract the tuples in order according to l_i and r_i.

(3) repeat (2) until it doens't hold.

WIEN can only work for some very structured text such as the web pages generated automatically by programs because of its strong assumption. Also, the features selected are not expressive enough to capture the rich format of HTML text.

2.3.2 STALKER

The information on a web page is often presented in a hierarchical format. For example, each page might contain a list of tuples, each tuple might contain a list of smaller tuples or items, and so on.

STALKER (MMK99) is a single-slot learning system which is capable of extracting information from arbitrarily complex combinations of embedded lists and items.

STALKER proposed Embedded Catalog Tree (ECT) for formalizing the embedded structure in the web pages. A web page is represented as a tree-like structure by an ECT formalism, where a leaf contains relevant data, while an internal node contains congregational data such as a list. For instance, Figure 2.2 shows the ECT formalism for the HTML text in Table 2.5.

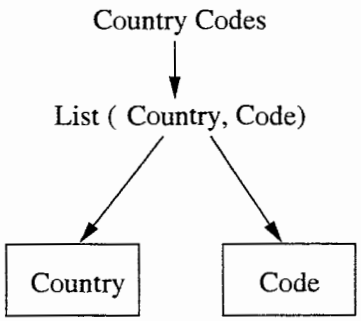

Figure 2.2 ECT description of Country Codes pages

The extractin patterns to be learned by STALKER include the extraction rule for each node in the tree as well as the iteration rule for each LIST node.

A sequential covering algorithm is used in STALKER. Given a set of positive instances, STALKER can learn an ordered list of disjuncts which covers all the positive instances. The ordering of disjuncts is based on the matching statistics of learned disjuncts: the disjuncts with more right matchings (a right matching is the matching which consumes neither too few tokens nor too many tokens among training instances) should be put earlier, the disjuncts with more correct matches are preferred in case of a tie.

The major contributions of STALKER are

- Although STALKER is a single-slot learning algorithm, it can extract multi-slot templates effectively because of the ECT mechanism can group single items extracted together.

- The decomposition of a web page extraction into single node extraction pattern learning is a typical use of "divide and conquer" strategy, which is invariant to the ordering of the items to be extracted.

2.4 Learning Extraction Patterns for Information Extraction from Semi-structured Text

Semi-structured text such as HTML is ungramatical and has no rigid format. The challenge lies in the learning of irregular extraction patterns. For instance, the normal parsing technique which divides a sentence into subject, verb, or objects can't work for semi-structured text because of its irregular format.

Almost all the IE systems we have reviewed so far use very limited features, and the representation of extraction pattern is not expressive enough. SRV and RAPIER incorporated more features and employed the extraction pattern in the form of first order predicate logic (RN95).

2.4.1 SRV

SRV (Fre98) is a multi-strategy single-slot learner for IE. Three different learners are used: rote learner, naive Bayes classifier (Mit97), and a relational rule learner which is similar to FOIL (Qui90).

There are many mature techniques in the field of machine learning. One of the contribution of SRV is to transform a IE problem into a typical machine learning problem. All possible phrases in the text up to a prespecified length are considered to be possible candidates for extraction. Each phrase is assigned by multiple learners a confidence weight indicating the probability of correct extraction as a target slot filler.

Rote learner simply compares a phrase to the correct slot filler in training instances and assigns it some similarity measurement.

Naive Bayes classifier considers each fragment of the text as a candidate hypothesis, which has a vector of TFIDF features associated with it. Naive Bayes algorithm is applied to each fragment of text and a confidence weight can be assigned according to observed data.

The relational learner performs a top-down induction which is similar to FOIL (Qui90). Rote learner and naive Bayes only take into account simple term frequency statistics. The relational learner complements it with structural features such as linguistic syntax, document layout, or simple orthography. The learning proceeds as FOIL: starting with null rule, SRV adds predicates according to FOIL's information gain. Two kinds of predicates are used: simple features mapping a token to a exact value such as *capitalized*? or *noun*?, and relational features such as next-token and prev-token.

2.4.2 RAPIER

RAPIER (CM99) is also a single-slot learning system. RAPIER extraction pattern makes use of both syntactic information such as POS tags and semantic class information such as WordNet (Mil95) links. Thus, POS tagger (Bri94) for preprocessing is required.

The extraction pattern learned by RAPIER is composed of three parts: pre-filler pattern coresponding to left delimiter, post-filler pattern corresponding to right delimiter, and filler pattern specifying the structure in the target slot filler. The pattern in each part is a sequence of elements of pattern items and pattern lists, where a pattern item can match exactly one word for its constraints, while a pattern list can match a set of words and each word satisfies a set of constraints.

RAPIER uses a bottom-up approach, which starts with most specific rules matching a target slot filler, then randomly picks up a pair of rules and conducts a beam search for the least general generalization..

2.5 A Related IE System: WHISK

WHISK (Sod99) is a comprehensive IE system which works for structured, semi-structured, and free text. It can extract both single-slot and multi-slot information. WHISK doesn't require syntactic preprocessing for structured and semi-structured text, and recommend syntactic analyzer and semantic tagger for free text.

The extraction pattern learned by WHISK is in the form of limited regular expression, which is a good representation considering the tradeoff between expressiveness and efficiency. Considering the IE task of extracting neighborhood, number of bedrooms, and price from the text in Table 2.6:

Table 2.6 An Example from the Rental Ads domain

Capitol Hill - 1 br twnhme. fplc D/W W/D. Undrgrnd pkg incl $675. call (206)999-9999 $< br >$ $< i >< fontsize = -2 >$ (This ad last ran on 08/03/97) $< /font >< /i >< hr >$

An example extraction pattern which can be learned by WHISK is as follows:

$$\text{* (Neighborhood) * (Bedroom) * '\$' (Number)} \tag{2.1}$$

where Neighborhood, Bedroom, and Number are semantic classes specified by domain experts. That is, we skip until a token in the semantic class of Neighborhood is encountered, extract this token. The tokens for the number of bedrooms and price can be extracted similarly.

WHISK learns the extraction rules using a top-down covering algorithm. First, a general rule covering the seed instance is learned, then we add terms to specialize it in order to reduce the Laplacian error of the rule. The Laplacian expected error is defined as

$$Laplacian = \frac{e+1}{n+1} \tag{2.2}$$

where n is the number of extractions on the training instances, and e is the number of wrong extractions among them. The candidate terms to be considered include left delimiter, right delimiter, semantic class of a target slot filler, and the additional terms in the seed instance.

WHISK can work in both batch and interactive mood. The algorithm for interactive learning is shown in Figure 2.3, 2.4, 2.5, and 2.6.

In batch mode, all the training instances are fed to the learning algorithm in one pass. While, in interactive mode, only a portion of training instances are used as input in each pass, and users can adjust the learning process according to the feedback from the learner.

```
Cover(UntaggedSet)

Input:
            UntaggedSet: a set of texts without annotated templates
Output:
            RuleSet: a set of extraction rules

            begin
              RuleSet = null
              TrainingSet = null
              repeat at user's request
                 select a subset of untagged texts NewInst
                 user tags NewInst
                 add NewInst to TrainingSet
                 discard rules with errors on TrainingSet
                 for each Inst in
                   for each Tag of Inst
                       if Tag is not covered by RuleSet
                         Rule=Grow_Rule(Inst,Tag,TrainingSet)
                 end for
              end repeat
              Prune RuleSet

              Return RuleSet
            end
```

Figure 2.3 WHISK Learning Algorithm: Top-down Covering

```
Grow_Rule(Inst,Tag,TrainingSet)
Input:
          Inst: a text with an annotated template
          Tag:  the annotated template associated with Inst
          TrainingSet: a set of training instances
Output:
          Rule: an extraction rule

          begin
            Rule = empty_rule(terms replaced by wildcards)
            for i=1 to number of slots in Tag
              Anchor(Rule,Inst,Tag,TrainingSet,i)
            end for
            do until Rule makes no errors on TrainingSet or
                  no improvement in Laplacian error
              Extend_Rule(Rule,Inst,Tag,TrainingSet)
            end do

            return Rule
          end
```

Figure 2.4 WHISK Learning Algorithm: Rule Growing

```
Anchor(Rule,Inst,TrainingSet,i)
Input:
          Rule: the rule before anchoring slot i
          Inst: a text with an annotated template
          TrainingSet: a set of training instances
          i: slot number
Output:
          Rule: the rule after anchoring slot i

          begin
            Base_1 = Rule+ terms just within extraction i
            test first i slots of Base_1 on TrainingSet
            while Base_1 does not cover Tag
              Extend_Rule(Base_1,Inst,Tag,TrainingSet)
            end while

            Base_2 = Rule + terms just outside extraction i
            test first i slots of Base_2 on TrainingSet
            while Base_2 does not cover Tag
              Extend_Rule(Base_2,Inst,Tag,TrainingSet)
            end while

            Rule = Base_1
            if Base_2 covers more of TrainingSet than Base_1
              Rule = Base_2
            end if

            return Rule
          end
```

Figure 2.5 WHISK Learning Algorithm: Slot Anchoring

```
Extend_Rule(Rule,Inst,Tag,TrainingSet)
Input:
          Rule: the rule before anchoring slot i
          Inst: a text with an annotated template
          Tag:  the annotated template associated with Inst
          TrainingSet: a set of training instances
Output:
          Rule:  the rule after entending

          begin
            Best_Rule = null
            Best_L = 1.0
            if Laplacian of Rule within error tolerance
              Best_Rule = Rule
              Best_L = Laplacian of Rule
            end if

            for each Term in Inst
              Proposed = Rule + Term
              test Proposed on TrainingSet
              if Laplacian of Proposed < Best_L
                Best_Rule = Proposed
                Best_L = Laplacian of Proposed
              end if
            end for

            Rule = Best_Rule
          end
```

Figure 2.6 WHISK Learning Algorithm: Rule Extension

2.6 Proposed IE System: IEPlus

IEPlus is a reconstruction of WHISK with improvements over semantic units, semantic resolution, target slot filler location, rule specialization, rule evaluation, rule firing strategy, etc.

IEPlus is composed of lexical analyzer, extraction pattern learner, and template generation module. An optional component is syntactic analyzer. Syntactic analysis is not applicable for semi-structured text because its text may not be gramatically correct, but it would definitely improve the performance of IEPlus on free text because syntactic constituents can be incorporated into the system as additional constraints.

Figure 2.7 shows the architecture of IEPlus.

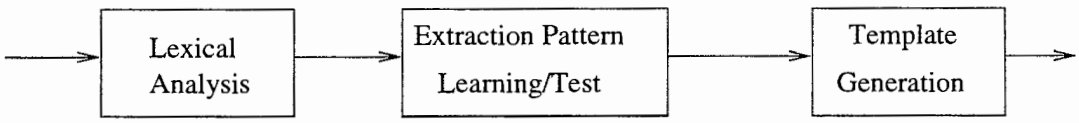

Figure 2.7 The Architecture of IEPlus

2.6.1 Semantic Units

A semantic unit is the smallest fragment of text which has self-contained semantic meaning. For instance, the semantic unit is "word" in RAPIER, syntactic constituent (e.g., subject, direct object) in AutoSlog. Semantic units determine the granularity of extractions, the robustness, and the performance of an IE system. Thus, the semantic units adopted by an IE system is critical for system performance.

Three types of semantic units are used in IEPlus:

- **Semantic Class**

  A semantic class in IEPlus is a set of terms specifying same concept. There could be domain-specific or domain-independent semantic classes. Domain-specific semantic classes are specified by domain experts in a domain. For example, the semantic class

for Bedrooms is Bdrm = ('brs' ||'br'||'bds'||'bdrm'||...). Domain-independent semantic classes can be obtained from some linguistic resources such as WordNet (Mil95). A semantic class corresponds to a synset in WordNet. Only domain-specific semantic classes are used in the current implementation of IEPlus.

- **Structural Phrase**

  Some fraction of text has predictable structure and would convey wrong information if we view it separately. For instance, 572-4496 indicates a phone number rather than two numbers connected by '-'. Structural phrases are used in IEPlus to capture the predictable structure inherent in a text.

- **Word**

  A fragment of text which can't be tokenized into semantic class and structural phrase is considered to be a word in IEPlus.

Fine grained semantic units in IEPlus differentiate similar text fragments from each other. For instance, there may be several numbers in a text, but only the one right after '$' (structural phrase) can be considered as a price.

2.6.2 Semantic Resolution

Let's consider a text from the Rental Ads domain in Table 2.7.

Table 2.7 A Rental Ads Text for Semantic Resolution

First Hill - Large 2 & 3 BR, great views, new kitchens, hardwd flrs. $750-900. call 206-999-9999. < *br* > < *hr* >

There are two digits in the text. The first digit can be tokenized into a semantic class Digit, and the second digit combined with 'BR' into a structural phrase representing the number of bedrooms. However, the interpretation of the first digit in this way is not precise. In fact, these two digits should be tokenized into same semantic category.

We propose to resolve it using context. Some contextual patterns can be specified by users. For instance, the grammar:

$$\text{Digit} < ParallelOperator > Bedrooms \Rightarrow Entry_{digit} \leftarrow \text{Bedrooms}.$$

where ParallelOperator=('&'|'–') is a semantic class, and $Entry_{digit}$ is the entry for the first digit in a symbol table. A lexer or a partial parser can resolve this semantic ambiguity by entering additional patterns. Similarly, the '900' in the text can be interpreted as a price rather than a number by context.

Languages cannot be understood without considering the knowledge shared by speakers. The adding of contextual knowledge enhances the capability of semantic resolution in the IEPlus system.

2.6.3 Target Slot Filler Location

One of the difficulties in extraction pattern learning as mentioned in (Car97) is that "the output templates indicate which strings should be extracted, and how they should be labeled but say nothing about which occurrence of the string is responsible for the extraction when multiple occurrences appear in the text".

For each target slot in the annotated templates, the corresponding target slot filler in the text needs to be located. Since a fragment of text can appear zero, one, or many times in the annotated template, it is not easy to find a corresponding filler for a target slot.

The heuristic guiding the slot filler location in IEPlus is that the tokens (i.e., word, semantic class, structural phrases) in a text are tried in order. If there is no matching token following current position, then IEPlus searches for candidate tokens from the beginning of a text.

2.6.4 Rule Specialization

A good extraction rule should be specialized such that it reflects the unique structure in the instances it covers. In some domains, it is easy to learn a general extraction rule which covers most of the instances. However, further improvement of an IE system is highly dependent on specialized small disjuncts (HAP89)

Let's consider which features in an instance make it saliently different from other instances. Although left delimiter, right delimiter, and semantic class of a slot filler are ideal candiate features for rule specialization, they are not enough. WHISK tries the terms in an instance in order until the error tolerance is satisfied.

A better heuristic is proposed to specialize a rule. Generally speaking, the terms near a target slot filler are usually more informative than the terms far from it according to local context, so a rule is specialized by using the terms nearby first, then the terms far away, until the error tolerance is met.

2.6.5 Rule Evaluation

Quite different from concept learning, the template generated by an extraction rule can be partially correct. Since IEPlus requires the evaluation of candidate extractin rules in the intermediate phases to guide its hill climbing process, reasonable evaluation of an extraction rule is the key to system performance.

Each extraction rule is evaluated based on its Laplacian error on a set of test instances. Precisely, the Laplacian error of a rule on a set of test instances is defined in Equation 2.3.

$$Laplacian_{rule} = \frac{\text{Number of wrong slots extracted}}{\text{Number of slots extracted}} \tag{2.3}$$

Only the slots in a case frame which has a matching case frame in the corresponding annotated template are considered to be correct. Thus, case frame matching is of great importance for rule evaluation.

For two matching case frames, they don't have to be exactly the same. The case frame extracted by IEPlus containing more slots can't match any case frame in the annotated template with fewer slots for the sake of assigning more credit to better rules, but it can be composed of less slots considering that the case frame might be generated by partially formed candidate rules in the training phase. However, the slots in the shared positions between two case frames have to be the same.

Consider the comparison of the following two templates.

```
@@TAGS Rental{Neighborhood UNIVERSITY} {Bedrooms 1 br} {Price $515} {Price $550}
@@TAGS Rental{Neighborhood UNIVERSITY} {Bedrooms 2 br} {Price $975}
```

Template Extracted by IEPlus

```
@@TAGS Rental{Neighborhood UNIVERSITY} {Bedrooms 1  br} {Price $515}
@@TAGS Rental{Neighborhood UNIVERSITY} {Bedrooms 2  br} {Price $975}
```

Template Annotated by Human

The second case frame in the template extracted by IEPlus matches the first case frame in the template annotated, while the first case frame has no match. Thus, the number of correct slots extracted from this text is 3, and the total number of slots extracted from this text is 7.

The matching between two case frames should be invariant to their positions in the templates. They should be considered to be matching as long as the same amount of information is presented.

This definition of case frame matching is coincident with commone sense. It guides IEPlus to search for better extraction rules through refined mechanism of case frame matching and rule evaluation.

2.6.6 Rule Firing Strategy

There may be multiple extraction rules matching a centain instance. What kind of strategy should be employed to generate the output template?

WHISK merges all the case frames generated by these rules when there are multiple rules matching an instance, which may reduce the precision because too many case frames could be generated.

Following the pattern matching strategy in JLex, the most specific rule matching an instance is prioritized in IEPlus. The more slots can be extracted by a matching rule, the more specific this rule is. The rule with low Laplacian error is chosen whenever there is a tie in terms of the number of slots matched.

This rule firing strategy is based on the assumption that the most specific rule has more chance to be correct than general ones.

2.6.7 Extraction Pattern Learning Algorithms

Figure 2.8 and 2.9 shows the pseudo code of IEPlus. Similar to WHISK, a top-down covering algorithm is used in IEPlus. However, IEPlus differs from WHISK in the rule specialization. IEPlus adds additional terms to specialize a rule only if a candidate rule has been anchored from slot filler, and its Laplacian error is greater than the pre-pruning threshold.

```
Cover(TrainingSet,PreThreshold,PostThreshold)
Input:
          TrainingSet:   a set of training instances
          PreThreshold:  a threshold for pre-pruning
          PostThreshold: a threshold for post-pruning
Output:
          RuleSet: a set of extraction rules

          begin
            EvaluationSet = a clone of TrainingSet
            RuleSet = null
            begin loop
              SeedInst = the first instance in TrainingSet
              aRule = Grow_rule(SeedInst, EvaluationSet, PreThreshold)
              if(Laplacian_error(aRule,EvaluationSet) < PostThreshold)
                add aRule to RuleSet
                remove the instances covered by aRule from TrainingSet
              end if
            end loop until(TrainingSet == null)

            return RuleSet
          end
```

Figure 2.8 IEPlus Learning Algorithm: Top-down Covering

```
Grow_rule(SeedInst, EvaluationSet, PreThreshold)
Input:
            SeedInst:  an instance where a rule will be inducted
            EvaluationSet: a set of training instances
                           for evaluating candidate rules
            PreThreshold:  a threshold for specifying the upper
                           bound of Laplacian error
Output:
            aRule: an extraction rule
            begin
              aRule = null
              aTemplate = the annotated template of SeedInst
              for i = 1 to number of case frames in aTemplate
                for j = 1 to number of slots in case frame i
                  locate target slot filler for slot j
                  store the position in an ordered set aSet
              end for
              for each position aPos in aSet
                candidate_1 = aRule + terms in the slot filler
                candidate_2 = aRule + left/right delimiter
                error_1 = Laplacian_error(candidate_1, EvaluationSet)
                error_2 = Laplacian_error(candidate_2, EvaluationSet)
                if(error_1 <= error_2)
                  BestRule = candidate_1
                  MinError = error_1
                else
                  BestRule = candidate_2
                  MinError = error_2
                end if
              end for
              if(MinError > PreThreshold)
                 begin loop
                   candidate = BestRule + additional terms
                   error = Laplacian_error(candidate, EvaluationSet)
                 end loop until (error < pre_threshold) or
                                no improvement in Laplacian error
                 BestRule = candidate
                 MinError = error
              end if
             return BestRule
           end
```

Figure 2.9 IEPlus Learning Algorithm: Rule Growing

2.7 Remarks on IE Systems

The IE systems for free text have the following drawbacks which limit their applicability in semi-structured text:

(1) They are highly dependent on the regular format of free text. For instance, AutoSlog can only extract relevant information from the text which follows its prespecified linguistic patterns.

(2) The features used are limited to local syntactic or semantic constraint, which is far from expressive enough to represent the rich extraction patterns required by semi-structured text.

The IE systems for structured text lack flexibility and are limited to very regular patterns in structured text. For example, WIEN can only extract relevant information from tabular web pages.

SRV and RAPIER use a rich set of features as constraints which could help extract the irregular patterns in semi-structured text with the cost of increased time complexity. They can't extract multi-slot information, which is very critical in some applications.

WHISK is the first IE system to learn single-slot or multi-slot extraction rules for the text styles ranging from structured to semi-structured to free text. As Wrapper inductin algorithms such as WIEN, WHISK can transform a structured text into relational database entries with high precision. Like SRV and RAPIER, WHISK requires no syntactic analysis for semi-structured text. Compared with CRYSTAL and AutoSlog, WHISK can extract information with finer granularity for free text.

IEPlus is a reconstruction of WHISK with improvements over semantic units, semantic resolution, target slot filler location, rule specialization, rule evaluation, rule firing strategy, etc. Its design is object-oriented, modular, extensible, and portable.

CHAPTER 3. DESIGN AND IMPLEMENTATION OF IEPLUS

Two major functions are implemented in IEPlus: a lexical analyzer, and an iterative parser. The lexical analyzer (ASU86; App95; App98) takes a stream of characters and produces a stream of basic semantic units including semantic classes, structural phrases, words, and punctuation marks. The iterative parser is composed of a set of regular expressions, which stand for the extraction patterns for various texts with different syntactic structure. The interaction between a lexical analyzer and a parser can be shown schematically in Figure 3.1 (ASU86).

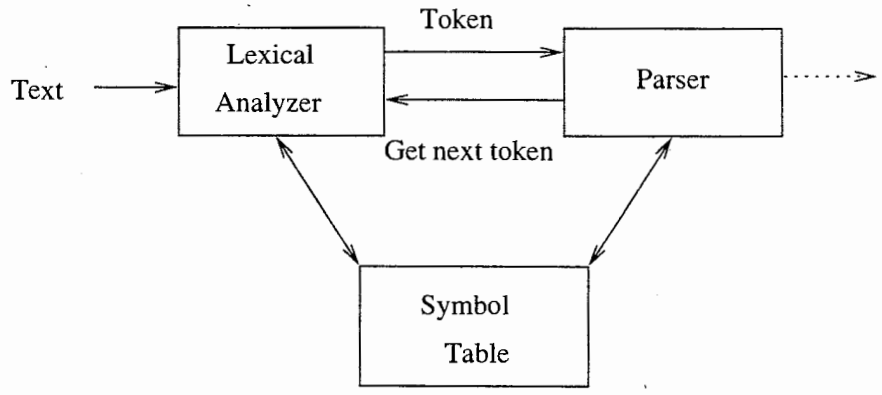

Figure 3.1 Interaction of lexical analyzer with parser

3.1 Lexical Analysis

The lexical analyzer is the first phrase of an IE system design. For the sake of simplicity, efficiency, and portability, specialized tools such as Lex (Les75) and Yacc(Joh78) have been designed to automate the process of lexical analyzer generation.

3.1.1 JLex Lexical Analyzer Generator

JLex [1] is a tool for generating a lexical analyzer for Java. It was developed by Elliot Berk at Princeton University.

The JLex utility follows the Lex (Les75) lexical analyzer generator model. JLex takes a input specification file which contains the details of a lexical analyzer, then creates a Java source program as the table-driven lexer.

3.1.1.1 How JLex Works

The way JLex works is shown in Figure 3.2.

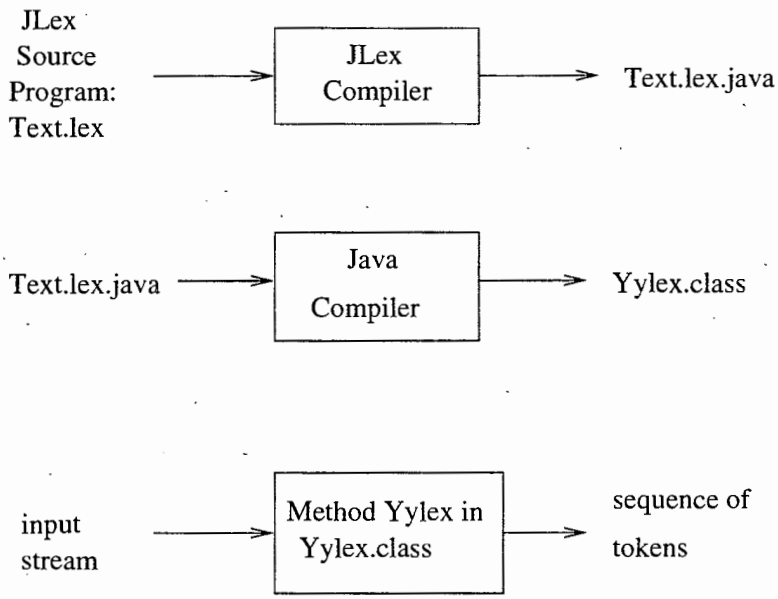

Figure 3.2 JLex Usage

- First, a specification file Text.lex is written in JLex language.
- Then, Text.lex is fed into the JLex compiler to produce a Java source program Text.lex.java. This source program contains a class named Yylex. The constructor of this class requires the input stream to be tokenized as an argument.

[1] http://www.cs.princeton.edu/ appel/modern/java/JLex/

- Third, Text.lex.java is run through the Java compiler to produce a lexical analyzer class file Yylex.class.

- Finally, an input stream is transformed into a sequence of tokens through the call of method Yylex in Yylex class.

3.1.1.2 JLex Specification

The JLex input specification is organized in three sections, which are separated by double-percent directives ("%%") at the beginning of an empty line.

- **User Code**

  User code section is the section where users can write Java code for being used by the lexical analyzer. Packages, classes, variables and return types for the lexer can be defined here.

  The code is optional and must be situated before the first %% delimiter. It will be copied verbatim into the beginning of the Java source program generated by JLex.

- **JLex Directives**

  The lexical analyzer directives are defined in this section. It begins after the first "%%" delimiter and continues until the second "%%" delimiter. Each JLex directive should start from the beginning of a line and be contained in a single line only. For instance, %ignorecase directive can be given to generate case-insensitive lexers.

  JLex directives section can also define macros and lexical states. The format of a micro definition is as follows:

  $$< macro\_name >=< definition >$$

  The definition above should be a valid regular expression.

  States can be defined by %state directive. The default JLex lexical state is YYINITIAL, which is the starting state of a lexical analyzer.

- **Regular Expression Rules**

  The third part of the JLex specification consists of a set of pattern-action pairs. These patterns are regular expressions, which are associated with actions consisting of Java source code.

  Each rule is composed of three distinct parts: the optional state list, the regular expression, and the associated action. Each rule has the following format:

  $$[< states >] < expression > < action$$

  - <states> is an optional state list which specifies the initial states under which the rule can be matched. If no state is specified, it is matched against all possible lexical states.
  - <expression> specifies the pattern to be matched. Strings from an input should be matched with at least one expression, otherwise an error will be generated. If more than one rule match strings from an input, the generated lexer chooses the rule matching the longest string. If more than one rule match strings of the same length, the lexer will choose the rule that is given first in the JLex specification. Therefore, rules appearing earlier in the specification are given a higher priority by the generated lexer.
  - <action> is the action associated with a rule. It contains the Java code which is copied verbatim into the lexical analyzer class. State transitions can be realized by function call *yybegin(state)*. The generated lexer remains in its initial state until a transition is made.

3.1.2 JLex Specifications for Rental Ads

The Rental Ads domain was collected from the Seattle Times on-line classified ads. The relevant information in Rental Ads includes neighborhood, number of bedrooms, price, phone number, etc.

Table 3.1 lists the regular expressions given as specifications to JLex to identify relevant tokens in Rental Ads domain. {BEDROOMS},{PRICE}, {AREACODE} , and {LOCALPHONE} are micro defintions as defined in the table. {BEDROOMS} represents the semantic class of Bedrooms, {PRICE} represents the semantic class of Price, {AREACODE} stands for the semantic class of Area Code, and {LOCALPHONE} stands for the semantic class of Local Phone Number. Some regular expressions in the table are presented in a different way from JLex source program for a self-contained explanation. The complete JLex source program for Rental Ads is in **Appendix B**.

The fine-grained semantic tokens increase the accuracy of target slot filler locating, the precision of extraction rules generated, the correct matching of an extraction rule against input text, etc.

Table 3.1 Patterns for Identifying Semantic Tokens in Rental Ads

Regular Expression given as JLex Specification	**Description**	**Example**
("West Seattle"|"Seattle Center"|"Capitol Hill"|...)	Neighborhood	Seattle Center
("br"|"bds"|"bdrm"|"bd"|...)	Bedrooms	br
[0-9](" "|"+")" "*{BEDROOMS}	# Bedrooms	1 br
("one"|"two")(" "|"+")" "*{BEDROOMS}	# Bedrooms	one br
("Studio"|"Studios")	# Bedrooms	Studio
[0-9]" "*("&"|"-")" "*[0-9](" "|"+")" "*{BEDROOMS} [a]	# Bedrooms	1 & 2 br
"$"[0-9]+	Price	$500
{PRICE}" "*"-"" "*[0-9]+ [b]	Price	$500-600
$[0-9][0-9][0-9][0-9][0-9][0-9][0-9]$	Local Ph.	6342521
$[0-9][0-9][0-9]'' -'' [0-9][0-9][0-9][0-9]$	Local Ph.	634-2521
$[0-9][0-9][0-9]$	Area Code	206
"("{AREACODE}")"{LOCALPHONE}	Domestic Ph.	(206)634-2521
{AREACODE}("-"|" "|"/"|".){LOCALPHONE}	Domestic Ph.	206-6342521

[a] Two Bedrooms tokens are generated
[b] Two Price tokens are generated

3.2 Iterative Parser Design

Each style of text has rules that prescribe its syntactic structure. Parsing is the process of determining if a sequence of tokens matches the syntactic structure specified by a grammar. Given a grammar, what a parser does is to construct a parse tree which matches the grammar.

There are a lot of parsing techniques (ASU86) if grammar rules are known. The challenge for the IE system is that the grammar rules which represent extraction patterns are not available apriori and are changing over the learning process. An extraction pattern learning algorithm tries to iteratively refine grammar rules for the sake of discovering the correct syntactic structure around relevant information. Thus, the design of an iterative parser is challenging and critical. Fortunately, the parsing required by IE system is usually partial parsing. Partial parsing concerns only the syntactic structure around relevant fragmants of text. Thus the parser for IE system is simpler and more efficient than full parsing which is usually used by the compilers of programming languages and text understanding systems.

There are several design alternatives to construct an iterative parser for an IE system. One possible choice is to use parser generators such as Yacc (Joh78) and CUP [2]. However, it is very awkward to construct an iterative parser in this way. Since the grammar is ever changing, you have to generate a specification for parser generator every time the rule is updated. This design choice will be less efficient and hard to extend.

A better design choice is to use Intepreter pattern (GHJV95).

3.2.1 Interpreter Design Pattern

The Interpreter pattern describes how to define a grammar for simple languages, represent strings in the language, and interpret these strings. The Interpreter pattern is applicable when there is a language to interpret, the grammar of the language is simple, and can be represented as abstract syntax trees.

Figure 3.3 shows the structure of the Interpreter pattern.

[2] http://www.cs.princeton.edu/ appel/modern/java/CUP/

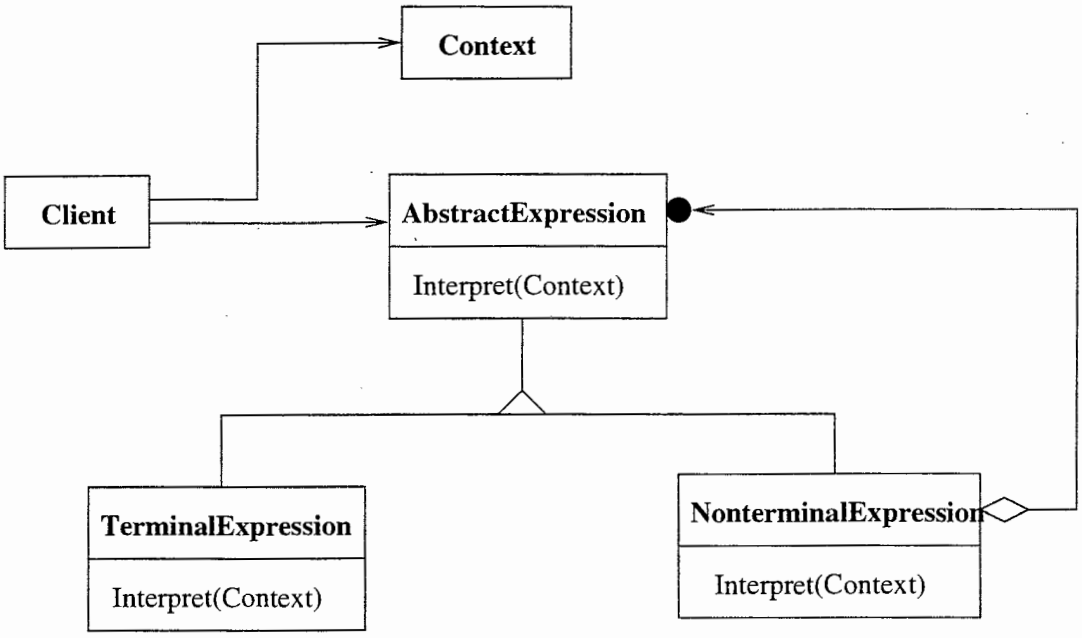

Figure 3.3 The Structure of Interpreter Design Pattern

- **AbstractExpression** defines the abstract Interpret operations common to all nodes in the abstract syntax tree.

- **TerminalExpression** implements the Interpret operations for the terminal symbols associated with the grammar. An instance is required for each terminal symbol in a sentence.

- **NonterminalExpression** represents the syntactic structure in each rule of the grammar. For instance, there are Alternative Expression, Repetition Expression, Sequence Expression for regular languages. One NonterminalExpression class is required for each rule in the grammar. For each rule $R ::= R_1 R_2 \ldots R_n$, instance variables of type AbstractExpression should be maintained for each symbol R_1 through R_n. Similarly, the Interpret operation is also implemented in the class which typically involves the recursive invocation of the operations in the variables representing R_1 through R_n.

- **Context** contains the global information shared by all classes. Context may be updated by the Interpret operations in each class.

- **Client** constructs an abstract syntax tree by using instances of the NonterminalExpression and TerminalExpression classes, and calls the Interpret operation for specific application.

The advantages (GHJV95) of Interpreter pattern are summarized as follows:

- *It is easy to change and extend the grammar.* Because grammar rules are represented by classes, they can be changed or extended easily by using inheritance or composition.
- *Implementing the grammar is easy.* Because the classes in an abstract syntax tree have similar implementations, they are easy to code.
- *Adding new ways to interpret is easy.* The class hierarchy makes it easier to interpret an expression in a new way. Visitor design pattern may be used for the application which requires creating new ways of interpreting frequently.

The extensibility and portability makes the Interpreter pattern fit nicely into extraction pattern learning in IE systems which requires dynamic rule generation and iterative rule evaluation.

3.2.2 Application of Interpreter pattern in IEPlus Implementation

The representation of extraction patterns in IEPlus employs a limited form of regular languages. The major difference lies in the pattern matching mechanism. IEPlus language only considers the first token matching each terminal expression rather than taking all possible matchings into consideration. The language in IEPlus is less expressive than normal regular languages, but it is more efficient because it doesn't need to keep track of all possible matchings.

The following grammar defines the limited regular language in IEPlus:

```
RegularExpr ::= StartExpr|EndExpr|Term|SequentialExpr|SkipExpr
SequentialExpr ::= RegularExpr RegularExpr ...
SkipExpr ::= * RegularExpr
```

where StartExpr, EndExpr, and Term are terminal expressions, StartExpr is the pattern indicating the beginning of a string, while EndExpr indicates the end of a string. Each Term corresponds to each token obtained from lexical analysis, and there is an associated list of (Tag,Slot) coordinates which shows the locations where the matching term will be extracted. SequentialExpr and SkipExpr are nonterminal expressions. SequentialExpr contains a list of regular expressions. It matches a string only if this string matches these regular expressions in order. SkipExpr is composed of one regular expression. Given a string, it skips to check if any subsequent tokens match the regular expression. We can specify to extract the skipped tokens or matched tokens by assigning a list of (Tag,Slot) coordinates with a corresponding part. The '*' in SkipExpr is different from the Kleen Star in regular languages, it stands for wild card.

The grammar above is represented by six classes. an abstract class RegularExpr and its five subclasses StartExpr, EndExpr, Term, SequentialExpr, and SkipExpr. Figure 3.4 shows the class diagram according to the notation in (GHJV95).

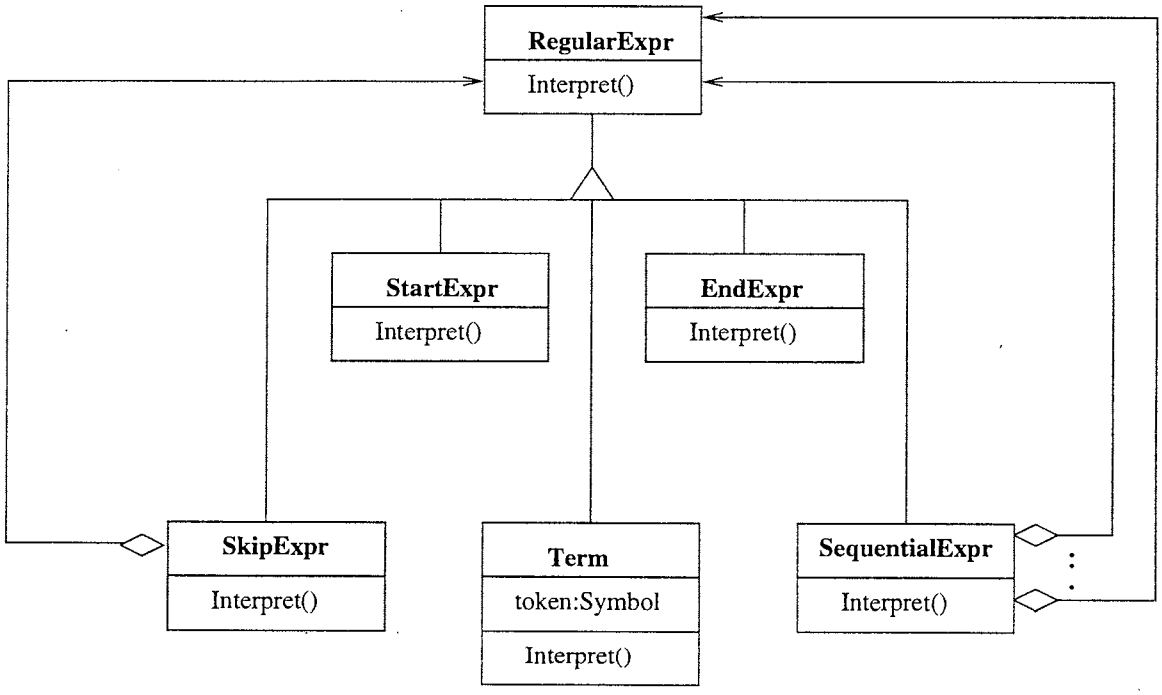

Figure 3.4 The Class Diagram for IEPlus Grammar

The matching strategy is implemented in each class. The composition of the instance of these classes can generate various extraction rules. For instance, the following rule:

* (NEIGHBORHOOD: 0,0) * (BEDROOMS: 0,1) * (PRICE: 0,2)

is an instance of SequentialExpr which is composed of three instances of SkipExpr, and each instance of SkipExpr contains an instance of Term associated with the (Tag,Slot) coordinate for matching tokens.

We can change matching strategy and extend the grammar easily by using object-oriented mechanisms such as inheritance and composition. For example, if we want to add additional constraints such as the length of matching tokens into SkipExpr, we only need to modify the Interpret operation in class SkipExpr. We can also extend the language in IEPlus easily by adding more classes in the class diagram through inheritance and composition.

3.3 The Collection Classes in IEPlus

There are Collection classes in Java (Fla99), but we implemented specialized Collection classes in IEPlus in order to incorporte more sophisticated operations. Figure 3.5 shows the class diagram of the Collection design for IEPlus following UML (BRJ99).

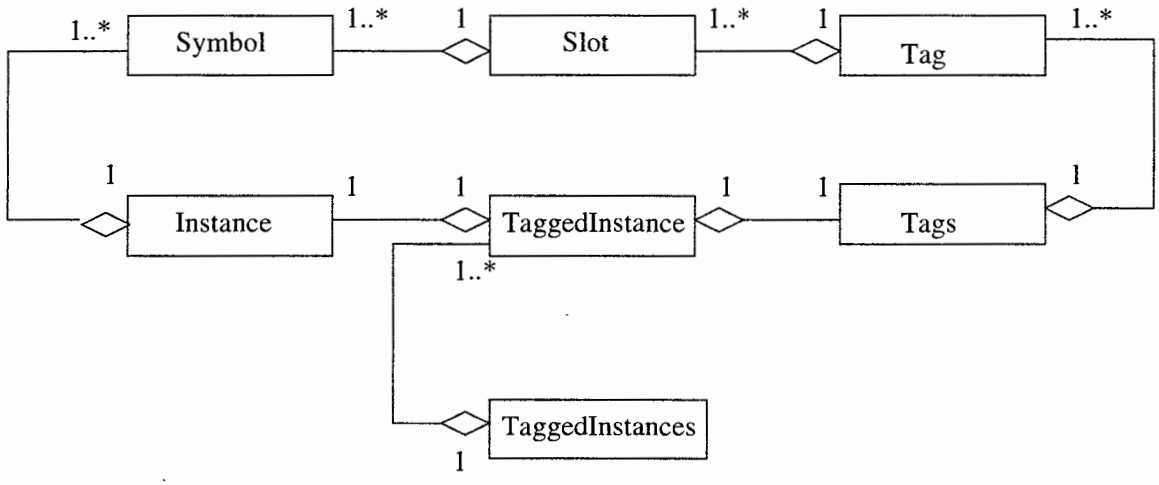

Figure 3.5 The Collection Class Diagram for IEPlus

Let's consider a sample tagged text for extracting from the Rental Ads domain.

```
@S[
  BALLARD - Deluxe 1 bed $550/2 bed $600. Jim 206-781-1300.  <br>
  <i> <font size=-2> (This ad is from 08/02/97 to 08/03/97.) </font> </i> <hr>
]@S
@@TAGS Rental {Neighborhood BALLARD} {Bedrooms 1 br} {Price $550}
@@TAGS Rental {Neighborhood BALLARD} {Bedrooms 2 br} {Price $600}
@@ENDTAGS
```

We model this tagged text as an instance of TaggedInstance class. It is composed of untagged text beginning with "@S[" and ending with "]@S" and a set of case frames in the output template. Each case frame starts with "@@TAGS" at the beginning of a line. We model the untagged text as an instance of Instance class, and each case frame as an instance of Tag class. Each case frame consists of a list of target slots. Each slot is embraced by "{" and "}", and it is an instance of Slot class. Each slot contains a list of tokens, which are instances of Symbol class. The set of case frames is stored in an instance of Tags class. There might be many tagged texts like this in a training set, we model it by defining class TaggedInstances.

CHAPTER 4. EXPERIMENTAL EVALUATION OF IEPLUS

Our objectives in developing IEPlus included: investigating the feasibility of learning information extraction patterns using of a limited form of regular languages for representing extraction patterns; evaluation of the performance of IEPlus under various settings; and comparing IEPlus with other IE systems.

4.1 Domain Description

The domain Rental Ads (Sod99) was collected by Stephen Soderland from Seattle Times on-line classified ads. The template contains three types of slots: Neighborhood, number of Bedrooms, and Price. Each type of slot might appear multiple times in an template. There might be multiple case frames in a template. The instances are generated from a HTML file separated by the HTML tag $< hr >$. 362 training instances are used in the experiments after filtering out the instances without relevant information. A test set of 350 instances obtained from the same source six weeks later is used for testing IEPlus. The template format is slightly different from that for WHISK. For instance, the slot for the number of Bedrooms is {Bedrooms 1 br} rather than {Bedrooms 1} for the sake of self-contained lexical analysis.

4.2 Performance Metrics

The experiments are designed to achieve two major goals: system evaluation and system comparison. The purpose of system evaluation is to determine the relationship between various parameters and the performance of IEPlus, while system comparison is to compare IEPlus with WHISK and some other systems.

Similar to information retrieval, precision and recall are used as the performance metrics of IE systems. Precision is the percentage of information extracted by the system which is correct, while recall is the percentage of relevant information which can be extracted correctly by the system.

We define the precision and recall formally in Equation 4.1 and 4.2.

$$Precision = \frac{\text{Number of correct slots extracted}}{\text{Number of slots extracted}} \tag{4.1}$$

$$Recall = \frac{\text{Number of correct slots extracted}}{\text{Number of slots in annotated templates}} \tag{4.2}$$

Since case frame is the minimal unit representing a relationship, a slot is considered to be correct only if it is in a correct case frame. There will be a detailed discussion about the similarity computation between the extracted case frame and the annotated case frame in next section.

For the ease of performance comparison, the MUC conference also proposed a F-measure (ARP92), which combines precision and recall into a single measurement for information extraction. The F-measure is defined in Equation 4.3.

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \tag{4.3}$$

where P stands for precision, R for recall, and β is a parameter weighting the relative importance of precision and recall. β is normally assigned to 1. The F-measure of current state-of-art IE systems is around 0.6 (AI99).

To some extent, the number of extraction rules induced from a training set can reflect the complexity of extraction patterns generated and the structures inherent in the training set. Compact extract rules can help humans understand the regularity underlying the domain. Thus the number of rules is reported as additional performance metrics when appropriate.

4.3 System Evaluation and Fine Tuning

IE system is more of an experimental system. Its performace is susceptible to the number of training instances, similarity measurement, error estimation function, granularity of lexical

analysis, rule firing strategy, rule pruning strategy, and various other parameters or settings. The motivation underlying system evaluation is to understand the influence of various parameters or setting on the performance of IEPlus such that we can fine tune the system to its optimal performance.

4.3.1 The Number of Training Instances

In order to test the effect of the number of training instances on the performance of IEPlus, a subset of a training set is obtained by random sampling. The size of the subset starts from 25 instances up to all the instances in the training set. For each training size, we reported the average and standard deviation of 5 trials on a separate test set with 350 instances. The performance metrics reported include precision, recall, and the number of rules. Table 4.1 shows the performance of IEPlus as the number of training instances increases.

Table 4.1 The Performance of IEPlus for the Rental Ads domain

Training Size	Prec +/- StdDev (%)	Recall +/- StdDev (%)	#Rules +/- StdDev
25	82.85 +/- 1.56	79.42 +/- 2.08	7.6 +/- 1.151
50	87.17 +/- 1.26	85.11 +/- 1.13	13.2 +/- 1.084
75	89.72 +/- 1.31	84.39 +/- 1.25	13.4 +/- 0.975
100	88.81 +/- 0.79	85.46 +/- 1.14	20.8 +/- 1.342
125	91.69 +/- 0.53	88.39 +/- 1.34	20.6 +/- 1.823
150	92.62 +/- 0.54	90.70 +/- 0.71	24.0 +/- 2.915
175	92.60 +/- 0.73	90.36 +/- 1.07	28.6 +/- 1.823
200	92.34 +/- 0.52	90.99 +/- 1.43	31.0 +/- 1.768
225	92.66 +/- 1.02	91.07 +/- 1.22	34.0 +/- 1.658
250	93.38 +/- 0.24	91.63 +/- 0.97	35.4 +/- 1.997
275	93.46 +/- 0.24	92.82 +/- 0.16	37.8 +/- 2.074
300	93.31 +/- 0.18	92.93 +/- 0.21	39.8 +/- 0.652
325	93.73 +/- 0.08	93.20 +/- 0.08	43.4 +/- 0.758
350	93.85 +/- 0.00	93.31 +/- 0.00	44.4 +/- 0.447
362	93.80 +/- 0.00	93.30 +/- 0.00	46.0 +/- 0.000

One interesting point is that what really counts for an IE system is not the number of training instances, but the number of training instances with different syntactic structures surrounding relevant fragments of text. This can explain the nonmonotonicity in Table 4.1 as training size increases.

The precision of IEPlus increased from 0.8285 to 0.938, and the recall from 0.7942 to 0.933 when the training size increased from 25 instances to 362 instances. Compared with WHISK without post-pruning, whose precision increased from 0.85 to 0.91, and the recall increased from 0.83 to 0.94 when the training size increased from 25 instances to 400 instances. Considering that IEPlus simply used random sampling, while WHISK used selective sampling (Sod99)(WHISK selects one third from instances covered by the current rule set, near misses of the rules, and instances not covered by any rule in each step), the performance of IEPlus is competitive with that of WHISK.

4.3.2 Pruning

An extraction rule without any error might be so specific that it can only cover very few instances. IEPlus uses pre-pruning to avoid overfitting. Given a seed instance, IEPlus first generates a candidate rule where only slot filler and left/right delimiters are used, then the Laplacian error of this rule is compared with a pre-pruning threshold. The rule is further specialized by adding additional terms if its Laplacian error is greater than the pre-pruning threshold.

Since not all the rules learned have low Laplacian expected error, adding the rules with large Laplacian expected error might pollute the rule set. Post-pruning could be used to discard such rules by assigning a threshold. Rules with Laplacian error greater than the threshold are removed from the rule set. The effect of post-pruning on the performance is shown in Table 4.2.

The best precision of IEPlus with post-pruning is 0.944, and the best recall is 0.933, while the best precision of WHISK with post-pruning is between 0.94 and 0.98, and the recall is 0.92. The performance of IEPlus with post-pruning is still comparable to WHISK with post-pruning.

Table 4.2 The Effect of Post-Pruning

Threshold	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Precision	0.0	0.780	0.804	0.831	0.840	0.854	0.887	0.944	0.944	0.938
Recall	0.0	0.734	0.756	0.819	0.830	0.844	0.876	0.933	0.933	0.933
#Rules	0	3	4	7	30	32	37	39	40	46

Threshold 0.8 and 0.9 result in the best system performance, but they are close to the system without post-pruning corresponding to threshold 1.0. This could be attributed to the specific rule firing strategy in IEPlus. Rules with the largest number of slots extracted are fired when there are multiple matching rules, and the rule with low Laplacian error is chosen when there is a tie on the number of slots extracted. Thus the rules with large Laplacian error may have very few chance of being fired.

Another possible explanation is that the rules filtered out by post-pruning might contain extraction patterns which are not covered by other rules with low Laplacian error. These rules play the important role of small disjuncts.

4.3.3 Rule Evaluation

Similar to WHISK, IEPlus learns rules in greedy fashion. It needs to evaluate two candidate rules for each slot filler in the training phase, and chooses the one with low Laplacian error. For each case frame in the output template extracted by candidate rules, it has to be determined whether there is a matching case frame in the template annotated by humans.

One refinement which has been implemented in IEPlus is its order-invariance for case frame matching. For instance, the cases frames in the two templates below are in different order, but the same amount of information is presented. Therefore, each case frame in the extracted template is considered to have a matching one in the annotated template.

```
@@TAGS Rental {Neighborhood UNIVERSITY}{Bedrooms 2 br}{Price $975}
@@TAGS Rental {Neighborhood UNIVERSITY}{Bedrooms 1 br}{Price $515}
```

Template Extracted by IEPlus

```
@@TAGS Rental {Neighborhood UNIVERSITY } {Bedrooms 1  br} {Price $515 }
@@TAGS Rental {Neighborhood UNIVERSITY } {Bedrooms 2  br} {Price $975 }
```

Template Annotated by Human

For two matching case frames, they don't have to be exactly the same. The case frame extracted by IEPlus containing more slots can't match any case frame in the annotated template

with fewer slots for the sake of assigning more credit to better rules, but it can be composed of less slots considering that the case frame might be generated by partially formed candidate rules in the training phase. However, the slots in the shared positions between two case frames have to be the same.

This fine-tuning of case frame matching is critical for the performance of IEPlus, which boosts the precision of IEPlus from 0.906 to 0.938, and the recall from 0.901 to 0.933. This more reasonable rule evaluation guides IEPlus to discover the extraction rules which better reflects the syntactic and semantic structure of the domain.

4.3.4 Lexical Analysis

A lexical analyzer decides the semantic units used, the granularity of tokenizations. Wrong tokens lead to misleading extraction rules. Thus, lexical analyzer fine tuning is indispensible for a state-of-art IE system.

The experiments designed in this section are used to answer the following questions in lexical analysis:

- *Is it better to use case sensitive matching or case insensitive matching in lexical analysis?*
  Since IEPlus is highly dependent on the correct categorization of semantic classes, and it is time-consuming if not impossible to enumerate all possible cases, a case insensitive lexer would be more able to handle unseen instances. Thus, the current optimal setting of IEPlus used the case insensitive lexer. The experiments on 362 training instances showed that the precision would be reduced from 0.938 to 0.905, and the recall from 0.933 to 0.892 if a case sensitive lexer is used.

- *Do HTML tags help extract relevant information?*
  Web pages are written in HyperText Markup Language (HTML), which are composed of two types of texts, regular text, and HTML tags describing the text. These markup tags can describe the appearance such as font size, layout such as table, etc. One interesting question in information extraction is whether HTML tags in a text help extract relevant

information though they may not contain relevant information in itself. IEPlus categorizes all HTML tags into the same semantic class, and thus has the precision of 0.938, the recall of 0.933 on 350 test instances using 46 rules learned from 362 training instances with HTML tags. The precision will be reduced to 0.938, the recall to 0.933 on the same test set using 47 extraction rules learned from the same set of training instances without HTML tags. This verified our conjecture that HTML tags play an important role in formatting information, but are not effective delimiters surrounding relevant information on the Rental Ads domain.

4.4 System Comparison

Since IEPlus is a descendant of WHISK, a comparision would be helpful to clarify the underlying differences between IEPlus and WHISK. IEPlus enhances WHISK in the following aspects: lexical analysis, rule evaluation, rule firing strategy, etc.

Table 4.3 summarizes the differences between IEPlus and WHISK.

Table 4.3 The Comparison of Features between IEPlus and WHISK

Feature	IEPlus	WHISK
Structural Phrase	Yes	No
Semantic Resolution	Yes	No
Heuristic for Slot Filler Locating	Yes	Unknown
Case Frame Order-invariance	Yes	Unknown
Rules Applied	Most specific rule	All matching rules
Template Generation	By most specific rule	All templates merged
Implementation Language	Java	Perl

The lexical analysis in IEPlus is much more fine-grained than WHISK. WHISK doesn't consider structural phrases, and the definition of semantic classes is slightly different from IEPlus. For instance, 'br' is tokenized into semantic class Bedrooms in WHISK, while '1 br' is in semantic class Bedrooms in IEPlus. There is no semantic resolution based on local context in WHISK. For instance, WHISK tokenizes '1 & 2 br' into a token DIGIT, a connecting token '&', and a semantic class Bedrooms; While IEPlus tokenizes it into two semantic classes, Bedrooms, and a connecting token '&'. WHISK doesn't consider the issue of case sensitivity.

The use of JLex makes IEPlus much easier to construct fine-grained lexer and to conduct various experiments.

For each target slot in the annotated templates, the corresponding target slot filler in the text needs to be located. Since a fragment of text can appear zero, one, or many times in the annotated template, it is not easy to have an exact matching between target slot and target slot filler. The heuristic guiding the slot filler locating in IEPlus is that the tokens (i.e., word, semantic class, structural phrases) in a text are tried in order. If there is no matching token following current position, then IEPlus searches for candidate tokens for the beginning of a text. There is no corresponding discussion in WHISK.

The case frame matching implemented in IEPlus is order-invariant, while there is no description about it in WHISK.

There may be multiple extraction rules matching a centain instance. Following the pattern matching strategy in JLex, the most specific rule matching an instance is prioritized. The more slots extracted by a rule from an instance, the more specific this rule is. The rule with low Laplacian error is chosen whenever there is a tie in terms of the number of slots extracted. Experiments demonstrated that the rule firing strategy can pick up the right rule coincident with a test instance with high probability. This specific rule firing strategy can make up the less specific rule generated by IEPlus, and increase the precision while keeping the recall of IEPlus. WHISK merges all the case frames generated by these rules when there are multiple rules matching an instance, which may reduce the precision because too many case frames could be generated.

Last but not least, IEPlus is implemented in Java such that it can work in multiple platforms, while WHISK is implemented in Perl which can make use of the pattern matching language features.

Table 4.4 lists the comparison of performance between IEPlus and WhISK. 362 instances used for training IEPlus were obtained by filtering out irrelevant instances from the original 400 instances, while 400 instances for WHISK contain both relevant and irrelevant instances.

Table 4.4 The Comparison of Performance between IEPlus and WHISK

System	Training Size	Precision (%)	Recall (%)
IEPlus without Post-Pruning	25	82.85	79.42
IEPlus without Post-Pruning	362	93.80	93.30
IEPlus with Post-Pruning	362	94.40	93.30
WHISK without Post-Pruning	25	85.00	83.00
WHISK without Post-Pruning	400	91.00	94.00
WHISK with Post-Pruning	400	94.00-98.00	92.00

The performance of WHISK is the average performance of ten runs[1], while the performance of IEPlus is the average performance of five trials.

4.5 Discussion

Similar to WHISK, the performance of IEPlus on the Rental Ads domain demonstrated that extraction rules in the form of regular languages are suitable for structured and semi-structured text, where there is a predictable order of fixed tokens surrounding the relevant information.

One of the bottlenecks for performance boosting in the Rental Ads domain is the inability of IEPlus to differentiate some complementary constituent from a standalone one. For instance, in the text fragment: "Ames, North of Ankeny", "Ankeny" shouldn't be tokenized into a standalone semantic class, it should be combined with direction token to act as the complementary constituent of previous semantic class. Such ambiguity can be resolved if a component of sentence analysis can be incorporated into the system.

The semantic class and structural phrase IEPlus can handle is self-delimiting or with predictable pattern, IEPlus is not capable of recognizing the unstructured portion of the text when the surrounding text is highly variable, which is the problem IEPlus has to solve for better performance on other domains without predictable delimiting patterns.

More constraints may be incorporated into IEPlus such that its precision can be enhanced. For instance, the following rule caused many wrong extractions in the experiments.

[1]The results reported are from the paper about WHISK (Sod99)

* (NEIGHBORHOOD: 0,0) * '/' (*: 0,1) '-' * (BEDROOMS: 0,2) * (PRICE: 0,3)

With the incorporation of the constraint regarding the length of matching tokens for the slot 1 in case frame 0, Precision and recall could be further improved.

4.6 Sample Extraction Rules

One of the rules which covers most of instances in the Rental Ads domain is:

* (NEIGHBORHOOD: 0,0) * (BEDROOMS: 0,1) * (PRICE: 0,2)

This rule means that it ignores any tokens until it encounters semantic class Neighborhood, extracts the tokens corresponding to this semantic class and puts it in slot 0 in case frame 0. It continues skipping up to semantic class Bedrooms, then the tokens associated with this semantic class is extracted to slot 1 in case frame 0. The tokens matching semantic class Price are extracted similarly.

This rule covers most of the instances with high precision. Later rules act as small disjuncts covering instances that can't be fit into this pattern, such as the text with more than one neighborhood, and a list of bedrooms and prices.

Another rule which generates multiple case frames is:

* (NEIGHBORHOOD: 0,0; 1,0; 2,0) * (BEDROOMS: 0,1) * (PRICE: 0,2) *
(BEDROOMS: 1,1) * (PRICE: 1,2) * (BEDROOMS: 2,1) * (PRICE: 2,2)

When it is applied to the instance below, it can generate the same template as annotated below.

```
@S[
  DOWNTOWN 1 br from $825.
  2 br from $1090.
  2 br/2 ba penthse $1785.
  Short term lse & furn apt avail.
  W/D, dw, pkg avl, hlth club.
```

```
  Call for appt 464-0585 <br>
  <hr>
]@S
@@TAGS Rental {Neighborhood DOWNTOWN} {Bedrooms 1 br} {Price $825}
@@TAGS Rental {Neighborhood DOWNTOWN} {Bedrooms 2 br} {Price $1090}
@@TAGS Rental {Neighborhood DOWNTOWN} {Bedrooms 2 br} {Price $1785}
```

For more sample extraction rules, see **Appendix B**.

CHAPTER 5. SUMMARY AND DISCUSSION

This chapter summarizes the major contributions of this thesis. IEPlus, a reconstruction of WHISK, was built to explore the use of machine learning to automate the design of IE systems.

5.1 Contributions

The contributions made by this thesis can be summarized as follows:

- A complete IE system, IEPlus, was developed in Java. Its design is object-oriented, modular, extensible, and portable.
- Fine-grained semantic units were proposed, and a strategy for semantic resolution was suggested.
- Heuristics for target slot filler location was proposed.
- Case frame matching was detailed in IEPlus. Experiments proved its effectiveness in evaluating candidate rules.
- A novel rule firing strategy was proposed, which prioritizes the most specific rule. Experiements proved that it can choose the right rule to fire for most of the instances.

Each of these contributions will be elaborated in order.

5.1.1 Object-oriented System Design

Lexical analysis, iterative parser generation, and instances modeling are three major functions implemented in IEPlus. The function of lexical analysis is to tokenize a text into a

sequence of tokens with correct semantic meaning. Iterative parser generation is a process of learning extraction rules in the form of abstract syntax trees from training instances iteratively. A sequential covering algorithm is used for learning. Instance modeling is simply to manage the text collections of various types.

JLex is chosen to generate domain-specific lexical analyzer, which is separated from domain-independent components in the sytem. This design makes the lexical analyzer highly flexible. Thus, world knowledge can be easily incorporated to guide fine-grained lexical analysis and to adjust the granularity of IEPlus to the right level.

Interpreter design pattern is applied to IEPlus for realizing a limited form of regular language. Because each grammar rule is represented by a class, the extension of a grammar rule is quite easy by using inheritance or composition. Well-defined responsibility in each class facilitates dynamic adding or changing of actions. This flexible design make it feasible to explore various rule representations, and to incorporate more features and constraints into grammar rules in extraction pattern learning.

The flexible design and implementation play an important role for being able to tune IEPlus to its optimal performance.

5.1.2 Semantic Units and Semantic Resolution

The semantic units in IEPlus include word, semantic class, and structural phrase. Semantic classes help IEPlus generalize beyond the tokens in the training instances, and structural phrases capture the tokens with predictable structure in the training instance. The fine-grained semantic units boost the performance of IEPlus in target slot filler location, rule generation, rule evaluation, and template generation.

Another contribution IEPlus made is semantic resolution. It is proposed in IEPlus that the semantic ambiguity of a token can be resolved by considering local context. Patterns and actions can be entered by users for specifying the correct semantic category a token belongs to in a certain syntactic context.

Experiments demonstrated the effectiveness of semantic units and semantic resolution in

IEPlus.

5.1.3 Target Slot Filler Location

For each target slot in the annotated templates, the corresponding target slot filler in the text needs to be located. Since a fragment of text can appear zero, one, or many times in the annotated template, it is not easy to have an exact matching between target slot and target slot filler. The heuristic guiding the slot filler location in IEPlus is that the tokens (i.e., word, semantic class, structural phrases) in a text are tried in order. If there is no matching token following current position, then IEPlus searches for candidate tokens for the beginning of a text.

5.1.4 Case Frame Matching and Rule Evaluation

IEPlus generates multi-slot extraction rules, which could output the template containing one or multiple case frames. Only if a case frame extracted by IEPlus has a matching one in the annotated template could it be considered to be correct, and all the slots in a wrong case frame are considered to wrong extractions.

The case frame matching implemented in IEPlus is order-invariant and coincident with common sense. Experiments with the Rental Ads domain demonstrated that it guides the rule learning algorithm to find the best rule reflecting the regularity inherent in the domain. Please refer to **Appendix C** for a complete set of rules generated from 362 training instances on the Rental Ads domain.

5.1.5 Rule Firing Strategy

WHISK uses merging strategy to combine the extractions made by all firing rules. That is, all the case frames generated by firing rules are considered to be the extractions of the system. It could lead to poor precision if multiple rules are fired and many case frames are generated.

It is proposed in IEPlus to fire the most specific rule when there are multiple rules matching a test instance. The more slots a rule can extract from an instance, the more specific it is.

Since each rule has an associated confidence which is the complement of Lapalician error (i.e., 1- Laplacian) in IEPlus, a rule with low Laplacian error is pickup whenever there is a tie in terms of the number of slots extracted. Experiments on the Rental Ads domain showed that this firing strategy can fire the right rule for most instances.

5.2 Future Work

Further improvement is possible for IEPlus in a number of directions. First, more tests are desirable. More domains of various text styles and more experimental metrics such as the precision and recall of each slot should be tested. Second, more enhancements can be made to IEPlus in text preprocessing, incorporating more features and constraints into the system, and adding more specialized component for specific entity recognition.

5.2.1 More Tests

There is a set of standard test data in the newly created RISE repository [1], including the Seminar Announcements collected by Freitag (Fre98), Jobs collected by Califf(Cal98), LA-Weekly Restaurant used by Muslea (MMK99), etc. The Seminar Announcements data set would test IEPlus's performance in name-entity extraction. The Jobs data set would be good for testing IEPlus in extracting many slots with various difficulties, and the nesting structure in LA-Weekly restaurant would challenge the power of IEPlus's limited regular languages. The experiments on these test sets will facilitate the comparison among IE systems. Also, more domains containing temporal, causal, or other complex relationships among events would be helpful to expose the problems in IEPlus.

Besides the precision and recall over complete system, more experimental metrics can be designed to test the performance of IEPlus. For instance, the difficulty of extractions differs among different slots. It would be better to test the performance of IEPlus on extracting individual slot such that the weaknesses and strengths of the system can be discovered.

[1] http://www.isi.edu/ muslea/RISE

5.2.2 Document Partitioning

An IE system has expertise in extracting relevant information from logically coherent segments of text, but it may not be good at extracting from a raw text. Thus, an indispensible function for a complete IE system is to divide raw text into logically coherent segments according to document hierarchy. These text segments should put logically related facts together and separate unrelated facts. Webfoot (Sod97b) is an example preprocessor which uses page layout cues to divide a web page into sentence-length segments of text. The output of Webfoot is fed into CRYSTAL (SFAL95) such that the segments of text from a web page can be treated as free text.

In order to turn IEPlus into a complete IE system, a preprocessor for automatically partitioning raw text into logically coherent segments is necessary. Since it is not always possible to specify salient delimiters, some unsupervised learning algorithms such as clustering might be able to group related facts together while keeping unrelated facts apart.

5.2.3 Relevance Filtering

Irrelevant instances were filtered out to form a training set of 362 relevant instances for the experiments on the Rental Ads domain. To build a complete IE system, this relevance filtering process should also be automated. There are two possible ways to tackle the problem.

Relevance filtering is essentially a problem of text classification, and there have been a lot of mature machine learning algorithms addressing the problem. Thus, some supervised learning algorithms such as decision tree and naive Bayesian algorithm would work. Another way is to use traditional information retrieval relevance feedback techniques (Sal89).

5.2.4 XML Representation

Extensible Markup Language, or XML[2] for short, is a World Wide Web Consortium Standard for data representation. XML was designed to describe data and to focus on what data is, while HTML was designed to display data and to focus on how data looks.

[2]http://www.w3.org/TR/2000/WD-xml-2e-20000814

XML can be used to describe the data in IEPlus. For instance, a training instance in IEPlus can be represented as:

```
<Document>
        <Advertisement>
              BALLARD - Deluxe 1 br $550
              Jim 206-781-1300.  <br>
             <i> <font size=-2> (This ad is from 08/02/97 to 08/03/97.)
            </font> </i> <hr>
        </Advertisement>
        <Template>
          <Slot>
           <Neighborhood>
                 BALLARD
           </Neighborhood>
           <Bedrooms>
                1 br
           </Bedrooms>
           <Price>
              $550
           </Price>
          </Slot>
        </Template>
</Document>
```

The advantages of using XML in IEPlus include:

- XML makes it easy for data sharing.
- XML facilitates content-based retrieval.

- XML documents are easy to parse because there is Java Simple API for XML (SAX) [3]
- XML documents are easy to be transformed into the format desired using XSLT [4].

5.2.5 Finite-State Transducer Cascade Architecture

A document could have different syntactic structure from different point of view. The finite-state transducer cascade (Abn96) builds up syntactic structure in a series of levels. For instance, for the following sentence:

An NH circle was used to solve problems with CAD.

It can be viewed as a sequence of words (Level 0). It can also be viewed as a list of POS tags (Level 1). Up one level (Level 2), it is viewed as a sequence of base noun phrases. At the top level (Level 3), it is viewed as a list of clauses (subject clause, verb clause, and complement clause). Each level is built upon the previous levels by a transducer, which is a finite-state automaton for mapping input languages to output languages.

The incorporation of finite-state transducer cascade architecture into IEPlus could improve the flat syntactic structure in IEPlus, but it poses challenging problems in parsing semi-structured text, extraction pattern learning from multiple levels of representation, rule pattern matching, etc. However, it is definitely a promising direction for boosting the system performance.

5.2.6 Semantic Class and Semantic Hierarchy

An information extraction learning system can generalize its extraction pattern beyond the tokens in its training instances. The current implementation of IEPlus only incorporates the domain-specific semantic classes, which is far from complete and perfect.

An improvement of IEPlus is to support domain-independent semantic class and semantic hierarchy. WordNet (Mil95) is a domain-independent lexical database of about 57,000 words containing a semantic hierarchy in the form of hypernym links. A semantic class is represented

[3] http://www.javaworld.com/javaworld/jw-08-2000/jw-0804-sax-2.html
[4] http://www.w3.org/TR/xslt

by a *synsets* in WordNet. The semantic hierarchy implemented in WordNet would greatly faciliate the semantic generalization and specialization in an extraction rule learning process. Thus, extending IEPlus to support WordNet would definitely boost the performance of IEPlus.

5.2.7 Additional constraints

The constraints in an IE system can be categorized into two types: local constraints and non-local constraints.

Three types of contraints have been incorporated in IEPlus: word, semantic class, and structural phrase. All of them are local constraints. There are many more features which can be added as local constraints. Examples include the length of a slot to be extracted, orthographic constraints (Fre98), POS tag (CM99), syntax constraints such as subject, direct object, prepositional phrase (Ril93) depending on the style of text for processing.

Non-local constraints are often ignored by IE system designers. These constraints embody the non-local relations between sentences such as coreference, recursive structure, etc. If the text is of recursive structure around relevant fragment of text, it is beyond the representation power of the regular language in IEPlus, then context free grammar (Sip97) would be a better knowledge representation.

5.2.8 More Specialized Entity Extraction

IEPlus uses semantic classes and structural phrases to identify an entity. However, some entity in a text may not be enumerable and may not contain predictable structure, and the entities in the same text could be so similar that even human experts can hardly differentiate them. The challenge for an IE system is how to differentiate ambiguous entities in a text. For instance, how to tell a person's name from a company's name if the company was named after a person.

There is no simple solution to this problem. One feasible way is to incorporate more specialized entity extraction components, which could make use of all kinds of hints embodied in the text such as capitalization, formatting tags, local context, coreference, and entity conventions.

APPENDIX A. SAMPLE TRAINING INSTANCES

This appendix contains three types of training instances with different number of case frames in the annotated templates.

```
@S[
  BALLARD - 1 br, covered prkng, strg, N/P $530.
  206-634-2521 <br>
  <i> <font size=-2> (This ad last ran on 08/03/97.) </font> </i> <hr>
]@S
@@TAGS Rental {Neighborhood BALLARD} {Bedrooms 1 br} {Price $530}
@@ENDTAGS
```

```
@S[
  BOULEVARD PARK - Spacious 2 BR, 11/2 BA $585.
  1 BR with view of Seattle skyline $465.
  Pool.
  NO PETS PLEASE.  <br>
  Call 206-767-3806 <br>
  <i> <font size=-2> (This ad last ran on 08/03/97.) </font> </i> <hr>
]@S
@@TAGS Rental {Neighborhood BOULEVARD PARK} {Bedrooms 2 br} {Price $585}
@@TAGS Rental {Neighborhood BOULEVARD PARK} {Bedrooms 1 br} {Price $465}
@@ENDTAGS
```

```
@S[
  Capitol Hill - 1 bdrm w/decks $625; 2 bdrm w/patios $725.
  Both: yard, off st prkng, buses.
  N/S (206) 324-6241 <br>
  <i> <font size=-2> (This ad last ran on 08/03/97.) </font> </i> <hr>
]@S 61
@@TAGS Rental {Neighborhood Capitol Hill} {Bedrooms 1 br} {Price $625}
@@TAGS Rental {Neighborhood Capitol Hill} {Bedrooms 2 br} {Price $725}
@@TAGS Rental {Neighborhood Capitol Hill} {Bedrooms 1 br} {Price $725}
@@ENDTAGS
```

APPENDIX B. COMPLETE JLEX SPECIFICATION

```
package Parse;
import java.util.*;

%%

%implements Lexer
%function nextToken
%type java_cup.runtime.Symbol
%char
%line
%state INSTANCE
%state TAGS
%state TAG
%state SLOT
%state UNIT

%{
private void newline()
{
  errorMsg.newline(yychar);
}
```

```
private void err(int pos, String s)
{
  errorMsg.error(pos,s);
}

private void err(String s)
{
  err(yychar,s);
}

private java_cup.runtime.Symbol tok(int kind, Object value)
{
 return new java_cup.runtime.Symbol(kind, yychar, yychar+yylength(), value);
}

private ErrorMsg.ErrorMsg errorMsg;

private TaggedInstances aTaggedInstances;
private TaggedInstance aTaggedInstance;
private Instance aInstance;
private Tags aTags;
private Tag  aTag;
private Slot aSlot;
private java_cup.runtime.Symbol tt;

//get the tagged instance set after tokenizing
public TaggedInstances getTaggedInstances()
{
```

```
  return aTaggedInstances;
}


Yylex(java.io.InputStream s, ErrorMsg.ErrorMsg e)
{
  this(s);
  errorMsg=e;
}


%}

%init{
      aTaggedInstances = new TaggedInstances();
%init}


%eofval{
    {
      return tok(sym.EOF, null);
    }
%eofval}


ALPHA=[A-Za-z]
DIGIT=[0-9]
DIGITS={DIGIT}+
REAL=({DIGITS}"."{DIGIT}+)|({DIGIT}+"."{DIGITS})
WORD={ALPHA}({ALPHA}|{DIGIT}|"_"|"@"|\\|"'")*
```

```
PRICE=("$"{DIGITS})
AREACODE={DIGIT}{DIGIT}{DIGIT}
LOCALCODE=({DIGIT}{DIGIT}{DIGIT}{DIGIT}{DIGIT}{DIGIT}{DIGIT}
          |{DIGIT}{DIGIT}{DIGIT}"-"{DIGIT}{DIGIT}{DIGIT}{DIGIT})
PHONE=({LOCALCODE}|"("{AREACODE}")"{LOCALCODE}
      |"("{AREACODE}") "{LOCALCODE}
      |{AREACODE}("-"|" "|"/"|"."){LOCALCODE}|"1-"{AREACODE}"-"{LOCALCODE})
HTMLTAG="<"[^>]*">"
WHITE_SPACE_CHAR=[\r\n\ \t\b\012]
BEDROOMS=("brs"|"br"|"bds"|"bdrm"|"bd"|"bedrooms"|"bedroom"|"bed"|"bdr"
         |"BDRM"|"BR"|"Br"|"Bedroom"|"BD"|"apts")
NEIGHBORHOOD=("ALKI BEACH"|"ALKI"|"ADMIRAL"|"Avalon"|"BALLARD"|"Ballard"
     |"Beacon Hill"|"Bellevue"|"Belltown"|"Bel-Sq"|"Bothell"|"BOULEVARD PARK"
     |"Broadview"|"Capitol Hill"|"Central Area"|"Central District"|"Central"
     |"Downtown"|"Eastlake"|"Eastside"|"EDMONDS"|"Fauntleroy"|"FACTORIA"
     |"First Hill"|"Fremont"|"GEORGETOWN"|"Green Lake"|"Greenlake"|"GREENWOOD"
     |"Issaquah"|"Juanita Beach"|"Kirkland"|"Lake City"|"Lake Union"
     |"Laurelhurst"|"Leschi"|"Licton"|"Lincoln Park"|"Madison Park"
     |"Madrona"|"MAGNOLIA"|"Maple Leaf"|"Mt Baker"|"Mt. Baker"
     |"North Seattle"|"Northend"|"Northgate"|"Mercer Is"|"Mercer Island"
     |"Newcastle"|"Oak Tree"|"Phinney Ridge"|"Phinney"|"Queen Anne"
     |"RAINIER VALLEY"|"Ravenna"|"Redmond"|"Regrade"|"Roosevelt"
     |"Sandpoint"|"Seattle, South"|"SHORELINE"|"South Seattle"
     |"UNIVERSITY DISTRICT"|"University Village"|"University"
     |"Volunteer Park"|"Wallingford"|"Wedgewood"|"West Seattle"
     |"Westwood"|"Wedgwood"|"Lincoln Pk"|"Juanita"|"Seattle Center")
MONTH=("Jan"|"January"|"Feb"|"Februrary"|"March"|"Apr"|"April"|"May"
      |"June"|"July"|"Aug"|"August"|"Sep"|"September"|"Oct"|"October"
```

```
      |"Nov"|"November"|"Dec"|"December")
YEAR=({DIGIT}{DIGIT}|{DIGIT}{DIGIT}{DIGIT}{DIGIT})
DATE=({DIGIT}{DIGIT}\/{DIGIT}{DIGIT}\/{YEAR}|{MONTH}" "{DIGIT}{DIGIT}","{YEAR})
SEPARATOR=(","|":"|";"|"."|"?"|\||"_"|"+"|"#"|\/|\\|\(|\)|"<"|">"|\"|"'"|"["
          |"]"|"!"|"+"|"="|"*"|"^"|"%"|"~"|"@"|\`|"$")
PARASEPARATOR=("&"|"-")
%%

<YYINITIAL> \@"S"\[ { yybegin(INSTANCE); aInstance = new Instance();
                      aTaggedInstance = new TaggedInstance(); }
<INSTANCE> {HTMLTAG} { tt = tok(sym.HTMLTAG,yytext());
                       aInstance.add(tt); return tt;}
<INSTANCE> {DATE} { tt = tok(sym.DATE,yytext());
                    aInstance.add(tt); return tt;}
<INSTANCE> {PHONE} { tt = tok(sym.PHONE,yytext());
                     aInstance.add(tt); return tt;}
<INSTANCE> {DIGIT}(" "|"+")" "*{BEDROOMS}
                   { tt = tok(sym.BEDROOMS,yytext().substring(0,1));
                     aInstance.add(tt); return tt;}
<INSTANCE> ("one"|"two")(" "|"+")" "*{BEDROOMS}
                   { tt = tok(sym.BEDROOMS,yytext().substring(0,3));
                     aInstance.add(tt); return tt;}
<INSTANCE> ("Studio"|"Studios") { tt = tok(sym.BEDROOMS,yytext());
                                  aInstance.add(tt); return tt;}
<INSTANCE> {DIGIT}" "*{PARASEPARATOR}" "*{DIGIT}(" "|"+")" "*{BEDROOMS}
            { tt = tok(sym.BEDROOMS,yytext().substring(0,1));
            aInstance.add(tt);
            int i=1;while(yytext().charAt(i)==' ') i++;
```

```
            tt = tok(sym.PARASEPARATOR,yytext().substring(i,++i));
            aInstance.add(tt);
            while(yytext().charAt(i)==' ') i++;
            tt = tok(sym.BEDROOMS,yytext().substring(i,++i));
            aInstance.add(tt);
            tt = tok(sym.COMPLEXBEDROOMS,yytext()); return tt;}
<INSTANCE> {NEIGHBORHOOD} { tt = tok(sym.NEIGHBORHOOD,yytext());
                            aInstance.add(tt); return tt;}
<INSTANCE> {PRICE} { tt = tok(sym.PRICE,yytext().substring(1,yylength()));
                     aInstance.add(tt); return tt; }
<INSTANCE> {PRICE}" "*"-"" "*{DIGITS}
           { int i=yytext().indexOf("-"); i--;
             while(yytext().charAt(i)==' ') i--; i++;
             tt = tok(sym.PRICE,yytext().substring(1,i));
             aInstance.add(tt);
             tt = tok(sym.PARASEPARATOR,"-");
             aInstance.add(tt);
             i=yytext().indexOf("-"); i++;
             while(yytext().charAt(i)==' ') i++;
             tt = tok(sym.PRICE,yytext().substring(i,yylength()));
             aInstance.add(tt);
             tt = tok(sym.COMPLEXPRICE,yytext()); return tt; }
<INSTANCE> {REAL} { tt = tok(sym.REAL,yytext()); aInstance.add(tt); return tt; }
<INSTANCE> {DIGITS} {tt = tok(sym.INT,yytext()); aInstance.add(tt); return tt;}
<INSTANCE> {WORD} { tt = tok(sym.WORD,yytext()); aInstance.add(tt); return tt; }
<INSTANCE> {DIGIT} { tt = tok(sym.DIGIT,yytext()); aInstance.add(tt); return tt; }
<INSTANCE> "{"     { tt = tok(sym.LBRACE,yytext()); aInstance.add(tt); return tt; }
<INSTANCE> "}"     { tt = tok(sym.RBRACE,yytext()); aInstance.add(tt); return tt; }
```

```
<INSTANCE> {SEPARATOR} { tt = tok(sym.SEPARATOR,yytext());
                         aInstance.add(tt); return tt;}
<INSTANCE> {PARASEPARATOR} { tt = tok(sym.PARASEPARATOR,yytext());
                             aInstance.add(tt); return tt; }
<INSTANCE> \]\@"S" { aTaggedInstance.setInstance(aInstance);
                     yybegin(YYINITIAL); }
<YYINITIAL> "@@TAGS" { yybegin(TAG); aTag = new Tag(); aTags = new Tags(); }
<TAGS>      "@@TAGS" { yybegin(TAG); aTag = new Tag(); }
<TAG>  {WORD} { aTag.setName(yytext()); yybegin(SLOT);}
<SLOT>   "{"     { aSlot = new Slot(); }
<SLOT>  {WORD} { aSlot.setName(yytext()); yybegin(UNIT);
                 return tok(sym.WORD, yytext());}
<UNIT>{NEIGHBORHOOD} { tt = tok(sym.NEIGHBORHOOD,yytext());
                       aSlot.add(tt);  return tt;}
<UNIT>{PRICE} { tt = tok(sym.PRICE,yytext().substring(1,yylength()));
                aSlot.add(tt);  return tt; }
<UNIT>{DIGIT}(" "|"+")" "*{BEDROOMS}
                    { tt = tok(sym.BEDROOMS,yytext().substring(0,1));
                                          aSlot.add(tt);  return tt; }
<UNIT>("one"|"two")(" "|"+")" "*{BEDROOMS}
               { tt = tok(sym.BEDROOMS,yytext().substring(0,3));
                 aSlot.add(tt);  return tt; }
<UNIT>  ("Studio"|"Studios") { tt = tok(sym.BEDROOMS,yytext());
                               aSlot.add(tt); return tt;}
<UNIT>{DATE} { tt = tok(sym.DATE,yytext());  aSlot.add(tt);  return tt; }
<UNIT>{PHONE} { tt = tok(sym.PHONE,yytext());  aSlot.add(tt);  return tt; }
<UNIT>{REAL} { tt = tok(sym.REAL,yytext());  aSlot.add(tt);  return tt; }
<UNIT>{DIGITS} { tt = tok(sym.INT,yytext());  aSlot.add(tt);  return tt; }
```

```
<UNIT>{WORD} { tt = tok(sym.WORD,yytext());  aSlot.add(tt);  return tt; }
<UNIT>{DIGIT} { tt = tok(sym.DIGIT,yytext());  aSlot.add(tt);  return tt; }
<UNIT> {SEPARATOR} { tt = tok(sym.SEPARATOR,yytext());
                     aSlot.add(tt); return tt; }
<UNIT> {PARASEPARATOR} { tt = tok(sym.PARASEPARATOR,yytext());
                         aSlot.add(tt); return tt;}
<UNIT>  "}" { aTag.add(aSlot); yybegin(SLOT); }
<SLOT> "@@COVERED_BY" { aTags.add(aTag); yybegin(TAGS); }

<YYINITIAL> "@@ENDTAGS" {aTaggedInstances.add(aTaggedInstance); }
<TAGS> "@@ENDTAGS" { aTaggedInstance.setTags(aTags);
                     aTaggedInstances.add(aTaggedInstance);
                     yybegin(YYINITIAL); }
<YYINITIAL> ({DIGIT}|{DIGITS}|{WORD}) { }
<YYINITIAL,INSTANCE,TAGS,TAG,SLOT,UNIT> {WHITE_SPACE_CHAR}* { }
<YYINITIAL> "@@ENDFILE" {return tok(sym.EOF,null);}
<YYINITIAL,INSTANCE,TAGS,TAG,SLOT,UNIT> .
          {System.out.println("Unrecognized symbol on line " + yyline); }
```

APPENDIX C. SAMPLE RULES GENERATED

This appendix contains a set of extraction rules learned from the Rental Ads domain.

```
Rule 0 :  * (NEIGHBORHOOD: 0,0) * (BEDROOMS: 0,1) * (PRICE: 0,2) * (PRICE: 0,3)

Rule 1 :  * (NEIGHBORHOOD: 0,0) * (BEDROOMS: 0,1) * (BEDROOMS: 0,2) * (PRICE: 0,3)
          * (PRICE: 0,4)

Rule 2 :  * (NEIGHBORHOOD: 0,0) * (BEDROOMS: 0,1) * (PRICE: 0,2)

Rule 3 :  * (NEIGHBORHOOD: 0,0; 1,0) * (BEDROOMS: 0,1) * (PRICE: 0,2) *
          '/' (*: 1,1) '600'  * (PRICE: 1,2)

Rule 4 :  * (NEIGHBORHOOD: 0,0) * (NEIGHBORHOOD: 0,1) * (BEDROOMS: 0,2)
          * (PRICE: 0,3)

Rule 5 :  * (NEIGHBORHOOD: 0,0) * (BEDROOMS: 0,1) * (BEDROOMS: 0,2)
          * (BEDROOMS: 0,3)

Rule 6 :  * (NEIGHBORHOOD: 0,0; 1,0) * (BEDROOMS: 0,1) * (PRICE: 0,2) *
          '.' (*: 1,1) 'with'  * (PRICE: 1,2)

Rule 7 :  * (NEIGHBORHOOD: 0,0) * (PRICE: 0,2)

Rule 8 :  * (NEIGHBORHOOD: 0,0; 1,0) * (BEDROOMS: 0,1) * (PRICE: 0,2) *
          '.' (*: 1,1) ','  * (PRICE: 1,2)

Rule 9 :  * (NEIGHBORHOOD: 0,0; 1,0; 2,0) * (BEDROOMS: 0,1; 2,1) * (PRICE: 0,2)
          * (BEDROOMS: 1,1) * (PRICE: 1,2; 2,2)

Rule 10 :  * (NEIGHBORHOOD: 0,0) * (BEDROOMS: 0,1) * (PRICE: 0,2) * (PRICE: 0,3)
           *  '.' (*: 0,4) ';'

Rule 11 :  * (NEIGHBORHOOD: 0,0) *  '/' (*: 0,1) '-'  * (BEDROOMS: 0,2)
           * (PRICE: 0,3)
```

```
Rule 12 :  * (NEIGHBORHOOD: 0,0) *  ('So': 0,1) ('Seattle': 0,1)  * (BEDROOMS: 0,2)
           * (PRICE: 0,3) * (PRICE: 0,4)
Rule 13 :  * (NEIGHBORHOOD: 0,0; 1,0) * (BEDROOMS: 0,1) * (PRICE: 0,2) *
           '-' (*: 0,3) ';' * (BEDROOMS: 1,1) * (PRICE: 1,2) * (PRICE: 1,3)
Rule 14 :  * (NEIGHBORHOOD: 0,0) *  'Affordable' (*: 0,1) '<br>'  * (BEDROOMS: 0,2)
Rule 15 :  * (NEIGHBORHOOD: 0,0) * ('Westlake': 0,1) * (BEDROOMS: 0,2)
           * (BEDROOMS: 0,3) * (PRICE: 0,4) * (PRICE: 0,5)
Rule 16 :  * (NEIGHBORHOOD: 0,0; 1,0; 2,0) * (NEIGHBORHOOD: 0,1; 1,1; 2,1)
           * (BEDROOMS: 0,2; 2,2) * (PRICE: 0,3) *  '<br>' (*: 1,2) 'from'
           * (PRICE: 1,3; 2,3)
Rule 17 :  * (NEIGHBORHOOD: 0,0; 1,0) *  ',' (*: 0,1) ':'  *  '-' (*: 1,1) ':'
           * (PRICE: 0,2) * (PRICE: 1,2)
Rule 18 :  * (NEIGHBORHOOD: 0,0; 1,0) * (BEDROOMS: 0,1) * (PRICE: 0,2) *
           ',' (*: 1,1) '750'  * (PRICE: 1,2) * (PRICE: 1,3)
Rule 19 :  * (NEIGHBORHOOD: 0,0) * (BEDROOMS: 0,1) * (BEDROOMS: 0,2) *
           '.' (*: 0,3) '+'
Rule 20 :  * (NEIGHBORHOOD: 0,0) * (NEIGHBORHOOD: 0,1) * (BEDROOMS: 0,2)
           * (PRICE: 0,3) * (PRICE: 0,4)
Rule 21 :  * (NEIGHBORHOOD: 0,0) * (NEIGHBORHOOD: 0,1) * (BEDROOMS: 0,2)
           * (BEDROOMS: 0,3) * (PRICE: 0,4)
Rule 22 :  * (NEIGHBORHOOD: 0,0) * ('Brdvw': 0,1) * (BEDROOMS: 0,2)
           * (PRICE: 0,3)
Rule 23 :  * (NEIGHBORHOOD: 0,0) *  '/' (*: 0,1) 'Brand'  * (BEDROOMS: 0,2)
           * (PRICE: 0,3)
Rule 24 :  * (NEIGHBORHOOD: 0,0) * (BEDROOMS: 0,1) *  '&' (*: 0,2) '''
           *  '.' (*: 0,3) 'Up'
Rule 25 :  * ('Madison': 0,0) * (NEIGHBORHOOD: 0,1) * (BEDROOMS: 0,2)
           * (BEDROOMS: 0,3) * (PRICE: 0,4) * (PRICE: 0,5)
```

```
Rule 26 :  * (NEIGHBORHOOD: 0,0) * (BEDROOMS: 0,1) * (PRICE: 0,2)
Rule 27 :  *  ('NEAR': 0,0) ('KING': 0,0) ('ST': 0,0)  * (BEDROOMS: 0,1)
           * (PRICE: 0,2)
Rule 28 :  * (NEIGHBORHOOD: 0,0) * (BEDROOMS: 0,1) * (BEDROOMS: 0,2) *
           '&' (*: 0,3) 'from'  * (PRICE: 0,4)
Rule 29 :  * (NEIGHBORHOOD: 0,0) * (BEDROOMS: 0,1) * (BEDROOMS: 0,2) * (PRICE: 0,3)
Rule 30 :  * ('Oaktree': 0,0) * (NEIGHBORHOOD: 0,1) * (BEDROOMS: 0,2) * (PRICE: 0,3)
Rule 31 :  * (NEIGHBORHOOD: 0,0; 1,0; 2,0) * (BEDROOMS: 0,1)
           * (PRICE: 0,2) * '.' (*: 1,1) ','  *  ',' (*: 1,2) '.'
           *  '.' (*: 2,1) '/'  * (PRICE: 2,2)
Rule 32 :  * (NEIGHBORHOOD: 0,0; 1,0) * (BEDROOMS: 0,1) * (PRICE: 0,2)
           * (PRICE: 0,3)  *  '.' (*: 1,1) '525'  * (PRICE: 1,2)
Rule 33 :  * (NEIGHBORHOOD: 0,0) * (BEDROOMS: 0,1) * (PRICE: 0,2)
           * (NEIGHBORHOOD: 1,0) * (BEDROOMS: 1,1) * (PRICE: 1,2)
Rule 34 :  * (NEIGHBORHOOD: 0,0) * (NEIGHBORHOOD: 0,1; 1,1) * (BEDROOMS: 0,2)
           * (PRICE: 0,3) * (BEDROOMS: 1,2) * (PRICE: 1,3) * (NEIGHBORHOOD: 1,0)
Rule 35 :  * (NEIGHBORHOOD: 0,0; 1,0) *  '-' (*: 0,1; 1,1) 'Junction'
           * (BEDROOMS: 0,2) * (PRICE: 0,3) * (BEDROOMS: 1,2) * (PRICE: 1,3)
Rule 36 :  *  ('Trenton': 0,0) ('St': 0,0)  * (BEDROOMS: 0,1) * (PRICE: 0,2)
           *  ('California': 1,0) ('Ave': 1,0)  * (BEDROOMS: 1,1) * (PRICE: 1,2)
           *  (NEIGHBORHOOD: 2,0) ('Ave': 2,0)  * (BEDROOMS: 2,1) * (PRICE: 2,2)
           *  ('Barton': 3,0) ('St': 3,0)  * (BEDROOMS: 3,1) * (PRICE: 3,2)
Rule 37 :  *  @start (*: 0,0) '-'  * (BEDROOMS: 0,1) * (PRICE: 0,2)
Rule 38 :  * (NEIGHBORHOOD: 0,0) *  '/' (*: 0,1) 'Spectacular'  * (BEDROOMS: 0,2)
           * (BEDROOMS: 0,3) * (PRICE: 0,4)
Rule 39 :  * (NEIGHBORHOOD: 0,0; 1,0; 2,0) * (BEDROOMS: 0,1) * (PRICE: 0,2)
           *  ';' (*: 1,1) ','  *  ',' (*: 1,2) '/'  * (PRICE: 1,3)
           * (BEDROOMS: 2,1)  * (PRICE: 2,2) * (PRICE: 2,3)
```

```
Rule 40 :  * (NEIGHBORHOOD: 0,0; 1,0; 2,0) * (BEDROOMS: 0,1) * (PRICE: 0,2)
           * (BEDROOMS: 1,1) * (PRICE: 1,2) * (PRICE: 1,3) * (BEDROOMS: 2,1)
           * (PRICE: 2,2) * (PRICE: 2,3)
Rule 41 :  * (NEIGHBORHOOD: 0,0; 1,0; 2,0) * (BEDROOMS: 0,1) * (PRICE: 0,2)
           * (BEDROOMS: 1,1) * (PRICE: 1,2) * (BEDROOMS: 2,1) * (PRICE: 2,2)
Rule 42 :  * (NEIGHBORHOOD: 0,0) * (BEDROOMS: 0,1) * (BEDROOMS: 0,2)
           * (BEDROOMS: 0,3) *  'fr' (*: 0,4) '.'
Rule 43 :  * (NEIGHBORHOOD: 0,0) *  'Vista' (*: 0,1) '99'  * (PRICE: 0,2)
Rule 44 :  * (NEIGHBORHOOD: 0,0) * ('Efficient': 0,1) * (PRICE: 0,2) * (PRICE: 0,3)
Rule 45 :  * (NEIGHBORHOOD: 0,0) * (BEDROOMS: 0,1) *  '&' (*: 0,2) '''
```

BIBLIOGRAPHY

[Abn96] S. Abney. Partial parsing via finite-state cascades. In *Workshop on Robust Parsing, 8th European Summer School in Logic, Language and Information*, pages 8–15, Prague, Czech Republic, 1996.

[AI99] Douglas E. Appelt and David J. Israel. Introduction to information extraction technology. IJCAI-99 tutorial, 1999.

[All95] James Allen. *Natural Language Understanding.* The Benjamin/Cummings, Redwood City, CA, 1995.

[App95] Andrew W. Appel. *Modern Compiler Implementation in C.* Cambridge University Press, New York, 1995.

[App98] Andrew W. Appel. *Modern Compiler Implementation in Java.* Cambridge University Press, New York, 1998.

[ARP92] Editor. ARPA. Proceedings of the fourth darpa message understanding evaluation and conference. Morgan Kaufman, San Mateo, CA, 1992.

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, Reading, MA, 1986.

[Bri94] E. Brill. Some advances in rule-based part of speech tagging. In *Proceedings of the 12th Annual Conference on Artificial Intelligence*, pages 722–727, 1994.

[BRJ99] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide.* Addison Wesley, Reading, MA, 1999.

[Cal98] Mary Elaine Califf. Relational learning techniques for natural language information extraction. Ph.D. Thesis, Department of Computer Sciences, University of Texas at Austin, 1998.

[Car97] Claire Cardie. Empirical methods in information extraction. *AI Magazine*, 18(4):65–79, 1997.

[CFMe97] M. Craven, D. Freitag, A. McCallum, and etc. Learning to extract symbolic knowledge from the world wide web. Technical report, School of Computer Science, Carnegie Mellon University, 1997.

[CM99] Mary Elaine Califf and Raymond J. Mooney. Relational learning of pattern-match rules for information extraction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 328–334, 1999.

[CN89] P Clark and R Niblett. The cn2 induction algorithm. *Machine Learning*, 3:261–284, 1989.

[DSS93] R. Davis, H. Shrobe, and P. Szolovits. What is a knowledge representation? *AI Magazine*, 14(1):17–33, 1993.

[Fla99] David Flanagan. *ava in a Nutshell : A Desktop Quick Reference (Java Series).* O'Reilly & Associates, Cambridge, MA, 1999.

[Fre98] Dayne Freitag. Information extraction from html: Application of a general machine learning approach. In *Proceedings of the Fifteenth Conference on Artificial Intelligence AAAI-98*, pages 517–523, 1998.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, Reading, MA, 1995.

[HAe90] Jerry R. Hobbs, Douglas E. Appelt, and etc. Interpretation as abduction. Technical Report SRI Technical Note 499, SRI International, Menlo Park, CA, 1990.

[HAe92] Jerry R. Hobbs, Douglas E. Appelt, and etc. Fastus: A system for extracting information from natural-language text. Technical Report SRI Technical Note 519, SRI International, Menlo Park, CA, 1992.

[HAP89] R.C. Holte, L. Acker, and B. Porter. Concept learning and the problem with small disjuncts. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 813–818, 1989.

[Hob97] Fastus: A cascaded finite-state transducer for extracting information from natural-language text. 05/20/97,In Stuart Shieber (Coordinator) Computation and Language E-Print Archive, 1997.

[Huf95] Scott B. Huffman. Learning information extraction patterns from examples. In *IJCAI-95 Workshop on new approaches to learning for natural language processing*, pages 127–142, 1995.

[Joh78] Stephen C. Johnson. Yacc: Yet another compiler compiler. Technical Report Technical Report CSTR 32, AT&T Bell Lab, Murray Hill, NJ, 1978.

[KM95] J. Kim and D. Moldovan. Acquisition of linguistic patterns for knowledge-based information extraction. *IEEE Transaction on Knowledge and Data Engineering*, 7(5):713–724, 1995.

[KWD97] Nicholas Kushmerick, Daniel S. Weld, and Robert Doorenbos. Wrapper induction for information extraction. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 729–737, 1997.

[Les75] M. E. Lesk. Lex - a lexical analyzer generator. Technical Report Technical Report CSTR 39, AT&T Bell Lab, Murray Hill, NJ, 1975.

[Mic73] R. S. Michalski. Discovering classification rules using variable valued logic system. In *Third International Joint Conference on Artificial Intelligence.*, pages 162–172, 1973.

[Mil95] G. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[Mit97] T. Mitchell. *Machine Learning.* McGraw Hill, New York, NY, 1997.

[MMK99] Ion Muslea, Steven Minton, and Craig A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems (Special issue on Best of Agents'99)*, 1999.

[Qui90] J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.

[Ril93] E. Riloff. Automatically constructing a dictionary for information extraction tasks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 811–816, 1993.

[RK91] Elaine Rich and Kevin Knight. *Artificial Intelligence.* McGraw-Hill Inc, New York, NY, 1991.

[RN95] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*, page 33. Prentice Hall, Engelwood Cliffs, NJ, 1995.

[Sal89] g. Salton. *Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer.* Addison-Wesley, Reading, MA, 1989.

[SFAL95] Stephen Soderland, D. Fisher, J. Aseltien, and W. Lehnert. Crystal: Inducing a conceptual dictionary. In *Proceedings of the Fourteen International Joint Conference on Artificial Intelligence*, pages 1314–1321, 1995.

[Sip97] Michael Sipser. *Introduction to the Theory of Computation.* PWS Publishing Company, Boston, MA, 1997.

[Sod97a] Stephen Soderland. Learning text analysis rules for domai-specific natural language processing. Technical Report Technical report UM-CS-1996-087, University of Massachusetts, Amherst, MA, 1997.

[Sod97b] Stephen Soderland. Learning to extract text-based information from the world wide web. In *Proceedings of Third International Conference on Knowledge Discovery and Data Mining*, pages 251–254, 1997.

[Sod99] Stephen Soderland. Learning information extraction rules for semi-structured and-free text. *Machine Learning*, pages 1–44, 1999.

[TMT97] C.A. Thompson, R.J. Mooney, and L.R. Tang. Learning to parse natural language database queries into logical form. In *Proceedings of the ML-97 Workshop on Automata Induction, Grammatical Inference, and Language Acquisition.* Association for Computational Linguistics, Somerset, NJ, 1997.