Diploma Thesis

Analysis and Comparison of Existent Information Extraction Methods

by Jun Ying

Supervisor: Prof. Johannes Fürnkranz

Knowledge Engineering Group

Darmstadt University of Technology (TUD)

- Computer Science Department -

Darmstadt, in September 2006

Abstract

Information extraction is initially applied for identification of desired information from natural language documents and conversion of the extracted text into a self-defined presentation. With the rapidly increasing amount of available information sources and electronic documents on the World Wide Web, information extraction is extended for identification from structured and semi-structured web pages.

In the past years, a lot of solutions are described and various information extraction systems are implemented. In this thesis, we present a theoretical analysis and comparison of several information extraction systems in two sub-areas: on the one hand, the systems distinguish from each other in the features used for identification. Thereby, we compare several aspects of different information extraction systems, such as pre-processing for generation of features, various constraints for characterization of target information, and different representations of extraction patterns, namely, how constraints are utilized. On the other hand, in order to reduce human efforts and improve the portability of information extraction systems, diverse machine learning techniques are applied for building information extraction systems. In this thesis, we represent various types of training data and introduce the active learning technique, the boosting algorithm and different rule learning algorithms used in information extraction systems. Most of the information extraction systems mentioned in this thesis employ rule learning techniques. Thereby, the structure, the evaluation heuristics and the pruning methods are compared. In addition, bayesian learning applied in information extraction system is presented as well.

Zusammenfassung

Unter Informationsextraktion versteht man die Verarbeitung, relevante Informationen von gegebenen Texten zu erkennen, zu extrahieren und in einer vordefinierten Form umzusetzen. Ursprünglich wurde Informationsextraktion nur für natürliche Sprachen verwendet, allerdings ist sie mit der rasch zunehmenden Menge der verfügbaren Dokumente im World Wide Web ebenfalls für Webseiten weit verbreitet.

In den letzten Jahren wurden bereits diverse Konzepte und verschiedene Systeme für Informationsextraktion entwickelt. In dieser Diplomarbeit stellen wir eine theoretische Analyse für einige Informationsextraktion-Systeme in zwei Unterkategorie vor: einerseits, unterscheiden sich die Systeme in den Merkmalen, die relevant für die Identifikation sind. Dabei vergleichen wir einige Aspekt in verschiedenen Informationsextraktion-Systemen, nämlich, Pre-processing (Bezeichnung der Eigenschaften von Wörtern), verschiedene Constraints (Identifikation von gewünschten Informationen), und unterschiedliche Darstellungen von Extraktionspattern (Bezeichnung der Zusammenwirkung von Beschränkungen). Anderseits, um menschliche Arbeit zu reduzieren und die Portabilität eines Informationsextraktion-Systems zu verbessern, werden verschiedene maschinelle Lernverfahren für den Aufbau eines Informationsextraktion-Systems angewendet. In dieser Diplomarbeit stellen wir die möglichen Arten der Trainingsdaten für maschinelle Lernverfahren dar, und das Active-Learning, der Boosting-Algorithmus und verschiedene Rule-Learning Algorithmen werden auch analysiert. Bei den Rule-Learning Algorithmen werden die Struktur, die Auswertungsheuristik und das Pruning verglichen. Zusätzlich wird auch Bayessches Lernen dargestellt.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Jun Ying
yingjun.misc@googlemail.com
Darmstadt, den 18. Sep. 2006

Danksagung

Ich möchte mich bei Herr Prof. Fürnkranz bedanken, dass er diese interessante Diplomarbeit ermöglicht hat, und mein Dank gilt allen, die mir bei Fragen und Probleme geholfen haben. Mein Dank gilt auch meiner Familie, die mich im Laufe meines Studiums unterstützt hat, und auch meinem Freund, die mir viel Mut zugesprochen hat.

Contents

Chapter 1

Introduction

1.1 Motivation

Information extraction is originally applied to identify desired information from natural language text and convert them into a self-defined presentation, e.g., a database with particular fields. With the huge and rapidly increasing amount of available information sources and electronic documents on the world wide web, information extraction is extended for identification from structured and semi-structured web pages. Recently, more and more research groups concentrate their attention on development of information extraction systems. Researches on information extraction could be divided into two subareas: the extraction patterns used for identification of target information from given text, and using machine learning techniques to automatically build such extraction patterns for the sake of avoiding expensive construction by hand. Actually, a lot of information extraction systems have been successfully implemented, and part of them perform very well, i.e., they operate much faster than human and have a comparable accuracy with manual works.

Researches focused on the comparison of information extraction systems have been released. For instance, Ion Muslea collected in [Mus99] extraction patterns from various information extraction systems, Line Eikvil described several systems according to the type of texts they deal with in [Eik99], and in [ICC+05] Neil Ireson, Fabio Ciravegna, etc., compared performance of different machine learning techniques used in information extraction systems relying on experimental results.

In this thesis, rather than observing the entire information extraction systems or running systems to obtain experimental data, we pay attention to the main components of extraction patterns and machine learning techniques respectively, and make a theoretical comparison of them.

1.2 Overview

This diploma thesis begins with a brief introduction to information extraction in chapter 2. This is followed by chapter 3, where components of recognizers utilized in different information extraction systems, such as possible preprocessing, miscellaneous constraints, various representations of extraction patterns, etc., are analyzed and compared. Then in chapter 4, we distinguish machine learning methods applied in various systems, e.g., boosting algorithm, rule learning, bayesian learning, and so on. Finally, we conclude with a summary of our findings in chapter 5.

Information extraction systems we use for comparison include: AutoSlog and its new version AutoSlog-TS, CRYSTAL, WHISK, SoftMealy, SRV, RAPIER, STALKER, WIEN, BWI, KnowItAll and Lixto.

Chapter 2

Information Extraction

Information extraction is initially applied for identification of desired information from natural language documents and conversion of the extracted text into a certain format, e.g. a tabular description such as a database. With the expansion of the internet in recent years, information extraction is also widely used for locating target phrases from web pages, which differ from natural language text in their structured or semi-structured format.

In this chapter, we at first compare information retrieval and information extraction. In section 2.2, we introduce the original information extraction used in natural language. Information extraction applied for extraction from web pages is presented in section 2.3.

2.1 Information Retrieval and Information Extraction

According to [CL96, Eik99], information retrieval is a relatively mature technology, which is used to acquire a set of relevant material from a huge amount of collections. A typical example of information retrieval is a keyword based search from world wide web, as a result, web pages containing that keyword are returned. The number of resulting pages could be very large and not all of information contained in a single page is interested. Hence, browsing all resulting pages in order to find the wanted information is sometimes impossible and time-consuming.

In contrast, information extraction addresses this problem by locating the target phrases from documents and transforms them into a structured presentation. Figure 2.1 shows an example task of information extraction. Given documents of seminar announcement, the elements *Date*, *Start-time*, *Location*, *Speaker* and *Topic* could be specified and filled into a predefined pattern.

Hence, in order to efficiently obtain the desired information, we can combine the both techniques, i.e. collect relevant documents from information flood using information retrieval technology at first, then identify relevant information from the collected documents by information extraction.

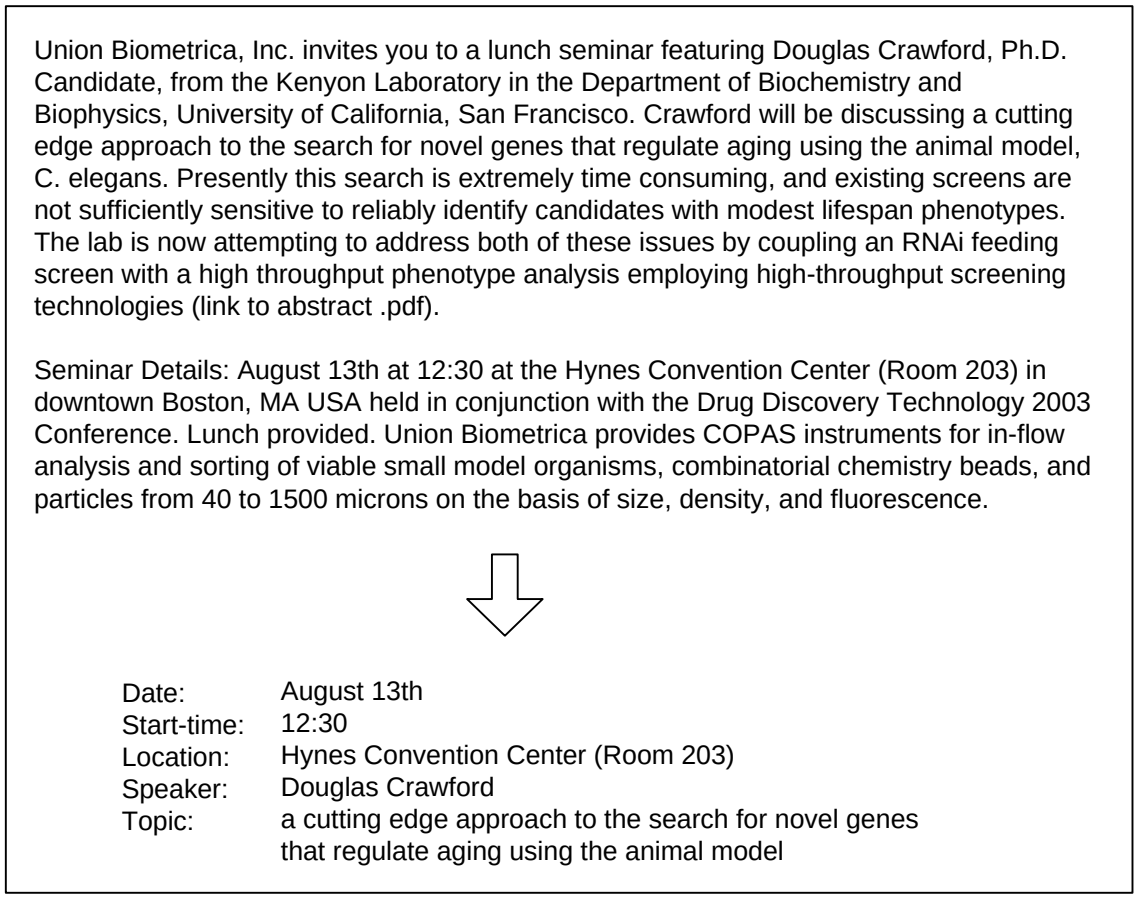


Figure 2.1: Information Extraction for Seminar Announcement

2.2 Information Extraction and Natural Language

Information extraction is originally the task of finding target phrases from natural language text. In this section, the history of information extraction is firstly presented. Then, in section 2.2.2, we describe metrics used to evaluate information extraction systems. In section 2.2.3, we give an overview of the general structure of information extraction system. Finally, two different approaches to building information extraction systems are introduced in section 2.2.4.

2.2.1 The History

As presented in [AI99, Eik99, CL96], information extraction is separated from the field of text understanding, which aims to represent the meaning of given text. Due to the rapid growth of available information and the comparable performance with human work, information extraction systems have attracted much attention.

The development of information extraction benefited enormously from the Message Understanding Conferences (MUCs) [SC93] since the end of the eighties of the last century, which are financially supported by DARPA[1] of the US and provide a forum for several research groups and government agencies to evaluate and discuss process in information extraction. Each MUC has a particular topic. Before a MUC is to take place, each participating group must construct an information extraction system for a same predetermined domain. Various systems are evaluated on same test sets based on same evaluation criteria. MUCs offers a communication platform for researchers of information extraction. The major contribution of MUCs is the evaluation of information extraction systems on a large scale.

The MUCs were held seven times during 1987-1998. In the following, we give an overview of the MUCs:

- 6 research organizations took part in the first MUC (MUC-1) in 1987, which focused on a small number of short narrative texts from naval messages. MUC-1 had either predefined task, nor evaluation criteria.
- MUC-2 that was held in 1989 used a larger training corpus of the same domain as MUC-1. The number of participants increased to 8. The task of MUC-2 was filling Scenario Templates, i.e. to analyze given texts and specify particular fragments, then fill them into corresponding slots of a predefined pattern.
- A total of 15 organizations attended MUC-3 in 1991. The training corpus was comprised of over 1000 newswire texts on the topic Latin American terrorism, while the output pattern contained 17 slots. Furthermore, a scoring program was applied for evaluation in MUC-3.
- MUC-4 in 1992 was concerned with the same domain as MUC-3. However, the output pattern became more complex and consisted of 22 slots. And the total number of attendees increased to 17.
- In MUC-5 held in 1993, the 17 participants from the US, the UK, Canada and Japan concentrated on domains joint ventures and microelectronics. In addition to

[1] the Defense Advanced Research Projects Agency

processing of English articles, information extraction from Japanese texts was also evaluated.

- Distinct to previous MUCs, which focused on the single evaluation task, Scenario Templates, MUC-6[2] in 1995 had three additional tasks: Named Entity Recognition[3], Coreference[4] and Template Elements[5].
- In addition to the evaluation tasks of MUC-6, MUC-7 [MP97] in 1997 had a more task: Template Relation, which recognizes relationships between template elements.

2.2.2 Evaluation Metrics

In this section we introduce evaluation criteria [Eik99] recall and precision that were used in MUCs, which evolved from evaluation used in information retrieval. They are also widely applied as the evaluation metrics at the present day.

Recall in information extraction indicates the proportion of the correct extractions from all possible extractions, i.e. all positive instances in the training set, while precision denotes the proportion of the correct extractions from the extracted information. Formally, recall and precision are defined as:

$$recall = \frac{\#\ correct\ extractions}{\#\ total\ possible\ extractions}$$

$$precision = \frac{\#\ correct\ extractions}{\#\ extracted\ information}$$

In [Sod99], Soderland expressed the metrics with the terms true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN), which denote positive examples considered as positive, negative examples considered as negative, negative examples considered as positive and positive examples considered as negative respectively. Therefore, recall and precision could also describe as:

$$recall = \frac{TP}{TP\ +\ FN}$$

$$precision = \frac{TP}{TP\ +\ FP}$$

[2]source: http://cs.nyu.edu/cs/faculty/grishman/muc6.html

[3]Named Entity Recognition has the task of identification of people names, organization names, place names, etc.

[4]Coreference has the task to identify coreference relations, i.e, the multiple expressions of the same noun phrase.

[5]Template Elements has the task to specify desired information associated with descriptive information such as organization, person and artifact entities.

Furthermore, in order to regard both of recall and precision concurrently, a combination of them so-called F-measure could be used:

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

where P and R indicate precision and recall respectively, while the β denotes the trade-off between precision and recall. If $\beta = 1$, precision and recall are equally weighted.

2.2.3 The Architecture

According to [Car97], although information extraction systems distinct from each other, we can find a general architecture of them. In the following, we show the five major components of a common information extraction system:

- Tokenization and Tagging, which are responsible for normalization of the given text and assignment of part-of-speech tags and semantic tags.
- Sentence Analysis, which is used to identify syntactic constituents such as subject, direct object, prepositional phrase, etc., by syntactic analysis and parsing.
- Extraction, where target phrases will be specified. Note that unlike the first and the second component that are independent from the topic of given text, extraction phase is domain-specific.
- Merging. This Component is applied for identification of coreference relations.
- Template Generation, where slots are filled with the extracted phrases and output patterns are created.

2.2.4 Approaches to Building Information Extraction Systems

In this section, we concentrate on how to construct an information extraction system. As described in [AI99], there are two primary approaches: Knowledge Engineering Approach and Automatic Training Approach.

The Knowledge Engineering Approach relies on knowledge engineers, i.e., human experts, who are skilled not only in the particular information extraction system, but also in the related domain. A person has to read a set of domain-relevant documents, find rules by his observation and construct rules in the grammar of that system by hand. Performance of the system is dependent on the competence of the knowledge engineers. This approach require a lot of human efforts and is time-consuming.

In contrast, no expert knowledge of the information system is necessary for adapting the system for a new domain using Automatic Training Approach. The required human work is reduced to annotation of the training documents. A training algorithm is then invoked the annotated training set and produces extraction rules automatically. Hence, Automatic Training Approach works much faster than Knowledge Engineering Approach and has a better portability, however a relative larger volume of training data is required.

2.3 Information Extraction and Web Pages

At the present day, information extraction is not only used for natural language text, but also commonly applied for web pages. In this section, we firstly distinguish various types of texts to be extracted, such as free text, structured text and semi-structured text. Then, wrappers used for extraction from web pages are introduced.

2.3.1 Free, Structured and Semi-Structured Text

According to [Eik99], information extraction systems for free text are applied to handle natural language texts, i.e., grammatical texts, such as new stories. Prior syntactic processing and semantic tagging are often required for extraction from free text.

The second possible type of given text is so-called structured text. A structured document has a particular format and is typically a result page by querying from database, for instance, the searching result of a certain article from Ebay, where a table including the picture, title, price, remaining time, etc., of all matched articles is returned. For identifying target information from structured texts, the format of the document, e.g., HTML tags, has to be learned.

Semi-structured text is type between structured and free text, which is neither strictly structured nor grammatical. Since semi-structured contains hardly complete sentences, systems for free text relying on syntactic analysis could not be adapted. Similarly, systems using purely format constraints for structured text is either not enough.

2.3.2 Wrappers

A wrapper [Eik99] is a procedure that is built to identify desired information from structured or semi-structured HTML documents and convert the information into a structured format such as database. It works by querying in a certain web source, collecting result pages and extracting target information.

Distinct to information extraction systems for free text, wrappers depend usually primarily on delimiters and separators for identification of targets rather than using linguistic constraints. Syntactic and semantic information are used as supplement especially for extraction from semi-structured documents.

Like building information extraction systems, wrappers could also be generated by hand or automatically using machine learning techniques. As mentioned in section 2.2.4, manual construction of such a system requires human experts and is time-consuming and error-prone. Additionally, new available web pages are frequently brought into internet and also structure of existing web pages could be changed. A new wrapper must be created not only for new pages, but also for documents, whose format is changed. Hence, building wrappers automatically is necessary.

Chapter 3

Recognizers

With the large and increasing amount of information sources and electronic documents available via the WWW, there are more and more researches focus on the information extraction problem. A lot of solutions are presented, which use various features to recognize the target information and different machine learning techniques to automatically construct extraction patterns for new domains.

In this chapter, we first introduce possible pre-processing in information extraction systems, which are used to generate features of words or tokens in given documents. Then various constraints that could be utilized for identifying target phrases are described in section 3.2. We give an overview of different representations for information extraction systems in section 3.3.

3.1 Pre-Processing

Before extraction rules or extraction patterns are used to identify the target information, pre-processing may be required to obtain characteristics of words or tokens in input documents. We distinguish syntactic and semantic pre-processing.

3.1.1 Syntactic Pre-Processing

Syntactic pre-processing is usually used to split input documents into small pieces and give syntactic features of single words, phrases and tokens. In the following, we show at first some sentence analyzers used for information extraction from natural language. Then web page parsers, which are used to segment web page source texts, are introduced. In the end of this section, we describe several simple syntactic taggers.

3.1.1.1 Sentence Analyzers

A prior syntactic processing is usually necessary for an information system, which handles free text like news stories, to specify syntactic relations between words or phrases of a input sentence.

CIRCUS As described in [Ril93, RS95], a conceptual sentence analyzer CIRCUS [Leh91] is used in the AutoSlog system[1].

CIRCUS takes a sentence as input, and segments it into clauses, then parses the clauses into syntactic constituents, such as subject, verb, direct object, indirect object, prepositional phrases.

Sentence:	"A Russian diplomat was killed near the embassy."
Subject:	a Russian diplomat
Verb:	was killed
Prepositional phrase:	near the embassy

Figure 3.1: Syntactic Analysis Using the Sentence Analyzer CIRCUS

In Figure 3.1 we show an example of syntactic analysis done by CIRCUS. The syntactic constituents subject, verb and prepositional phrase of the illustrative sentence are identified.

BADGER Similar to AutoSlog, the CRYSTAL system, which is presented in [SFAL95, Sod97b], also needs a sentence analyzer named BADGER [FSM+95], to extract information from free text.

A version[2] of BADGER performs similarly as CIRCUS, which separates a input sentence into syntactic constituents like subject, verb, object, prepositional phrases, etc. Another version[3] is additionally able to identify the relative clauses of subject, object, prepositional phrase, and so on.

Note that the both versions of BADGER and also CIRCUS generate no parse tree, but a plain list of syntactic constituents for a input sentence. All of the constituents are on the same level.

[1] AutoSlog is designed for information extraction from natural language.
[2] The version used for the Hospital Discharge domain.
[3] The version used for the Management Succession domain.

Sentence:	"Chahram Bolouri, who used to be Nortel's president of global operations, has been named president and CEO by Air Canada Technical Services."
Subject:	Chahram Bolouri
REL-Subject:	who used to be Nortel's president of global operations
Verb:	has been named
Object:	president and CEO
Prepositional phrase:	by Air Canada Technical Services

Figure 3.2: Syntactic Analysis Using the Sentence Analyzer BADGER

As illustrated in Figure 3.2, the example sentence is parsed into subject, the relative clause attached to the subject, verb and two prepositional phrases by the sentence analyzer BADGER.

As introduced in [Sod99], another system that uses BADGER for sentence analysis is called WHISK, which is the first system to handle all text styles: free text, semi-structured text and highly structured text. To extract information from semi-structured and structured text, only minimal preprocessing[4] is required. BADGER is needed when WHISK is used for free text.

Link Grammar Parser A syntactic analysis could be supplied by a link grammar parser, which is described in [ST93], for SRV [Fre98a, Fre98b]. Unlike CRYSTAL and BADGER, the link grammar parser indicates syntactic relationships between words and phrases rather than labeling the syntactic constituent for individual terms of the given sentence.

The link grammar parser offers quite a lot connector to describe various relations, e.g., verbs to direct or indirect objects, preposition to their objects, nouns to relative clauses, verbs with infinitives, etc. Figure 3.3 [Fre98b] shows a part of example sentence parsed by the link grammer paser. The term "Corp" shares a connector S with the term"said", which denotes that "Corp" is the subject-noun to the verb "said". According to parsed connectors, we can conclude the features[5] of each term, which are relevant for identification as well as for learning. For instance, as illustated in figure 3.3, the term "Corp" is a noun and the term, which shares the connector S with "Corp" on the right side, is the "said".

[4]The minimal preprocessing segments the input HTML source text into individual instances using HTML tags. For instance, the HTML tag "<ul>" could be used as delimiters to separate single papers in the source text illustrated in figure 3.11.

[5]Features used in SRV will be introduced in section 3.2.1.2

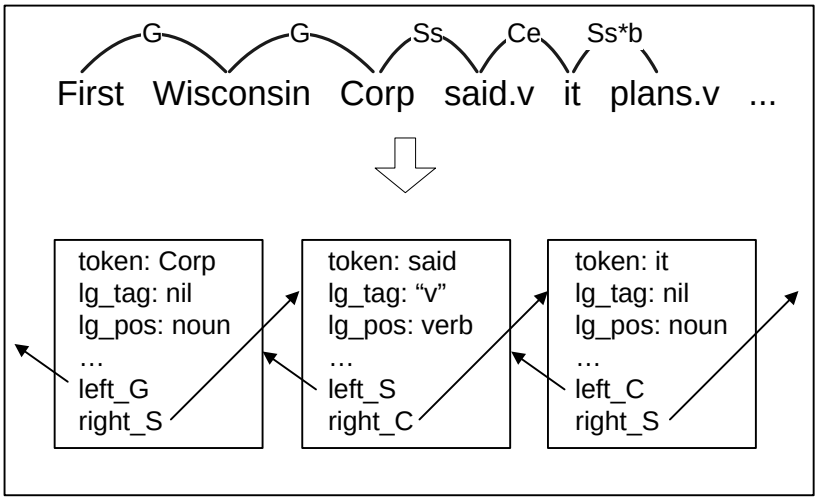


Figure 3.3: An example of link grammar feature derivation [Fre98b]

3.1.1.2 Web Page Analyzers

For specifying the target information from semi-structured and structured text, information extraction systems take HTML source strings as input. In some of the systems, the input string must be at first divided into small units. Since HTML source strings are not grammatical text, sentence analyzer will not work. For this reason, web page parsers are require for parsing HTML source text.

Webfoot As presented in [Sod97a], CRYSTAL is able to be extended to extract information for semi-structured and structured documents with the help of a web page parser named Webfoot.

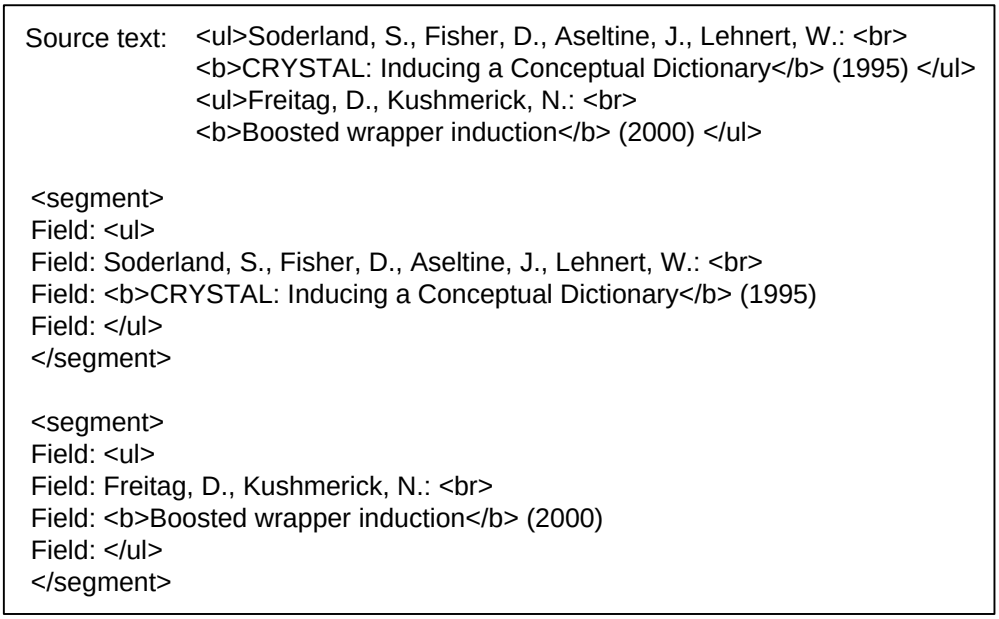
Webfoot begins with the segmentation of a HTML source text using page layout indications. A segment should be a group of logically related information, which is separated from others by domain-independent delimiters of level 1[6] used in Webfoot, and will be further divided by the delimiters of higher levels, until small fields[7] are identified.

Figure 3.4 shows an example source text and its segments and fields parsed by Webfoot. Each segment contains the authors and the title of a single paper and is divided into small fields.

Like CIRCUS and BADGER, Webfoot produces either no tree-like structure, all fields

[6]Delimiters used in Webfoot of various levels are presented in [Sod97a].

[7]If a segment contains less than twenty words without regards of HTML tags, it is to be separated into fields using higher level delimiters.


Source text: Soderland, S., Fisher, D., Aseltine, J., Lehnert, W.:
CRYSTAL: Inducing a Conceptual Dictionary (1995)
Freitag, D., Kushmerick, N.:
Boosted wrapper induction (2000)

Field:
Field: Soderland, S., Fisher, D., Aseltine, J., Lehnert, W.:
Field: CRYSTAL: Inducing a Conceptual Dictionary (1995)
Field:

Field:
Field: Freitag, D., Kushmerick, N.:
Field: Boosted wrapper induction (2000)
Field:

Figure 3.4: Segmentation by the Web Page Parser Webfoot

parsed by Webfoot are on the same level.

Embedded Catalog Formalism According to [MMK98, MMK99], HTML source strings are represented by embedded catalog formalism in STALKER. In contrast to Webfoot, the embedded catalog description has a tree structure. The root is the entire HTML document and each leaf contains an item from a tuple, which denotes significant data that could be extracted. An internal node indicates a tuple, in which an item could be either a leaf or another tuple.

Figure 3.5 shows the embedded catalog description for a searching result from www.hotel.de. The page consists of a list of hotels, each hotel is represented as a 5-tuple that contains *Name*, *Stars*, *Price*, *Address* and *Distance*, which is an embedded list including pairs of *Place* and *Kilometers*.

The content of each node is the corresponding sequence of tokens. For instance, the content of the root is the total token sequence, and generally the content of a certain node is a subsequence of the content of its parent. In order to obtain the slot filler contained in content of a leaf, STALKER identifies the path from root node to the target leaf, and iteratively specifies each node in the path by applying extraction rules to its parent. In this way, STALKER finds the target information by using several simple rules instead of a single complicated rule.

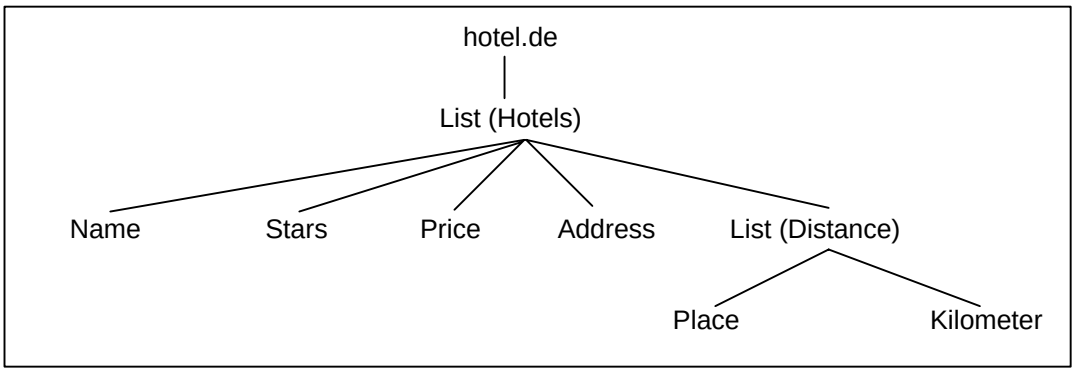


Figure 3.5: Embedded Catalog Description of a Result Page of Searching for Hotels

Java Swing Parser The Lixto[8] system introduced in [BFG01, GKB+04] relies on the Java Swing parser to obtain a HTML/XML DOM tree for a HTML page. Unlike the Embedded Catalog description used in STALKER, whose leaves describe the elements of a data tuple, Swing parser is domain-independent and bases only on HTML hierarchy. On the other hand, Swing parser further differs from the Embedded Catalog in the content of nodes. Rather than a plain sequence of tokens that contained in an Embedded Catalog's node, a node in a HTML tree generated by the Swing parser includes a set of attribute-value pairs, which contain:

- a `name` attribute that indicates the name of the node.
- tag attributes that are appended to the node. For instance, `bgcolor` for a <body> node, `border`, `width` for a <table> node.
- a special `elementtext` attribute, which describes the content of the node. For a non-leaf node, the value of `elementtext` consists of a sequence of the contents of all leaf nodes below it. For instance, the <table> node shown in figure 3.6 is represented with {`(name, table), (border, 1), (width, 75%), (elementtext, 56 K Modem...(041 1) 137)`}. The `elementtext` contains all elements in the table.

An example page and its parse tree with several descriptions of nodes are illustrated in figure 3.6 [BFG01]. Pairs of numbers insides parentheses denote the start and end position of the particular element. Some leaves are notated with a <`content`> element, for which tags such as <`b`>, <`href`> , etc., are considered as attributes and the value of the `elementtext` attribute is the strings contained insides e.g., <td>, <p> tags.

[8]Lixto is an information extraction system that provides a fully visual user interface.

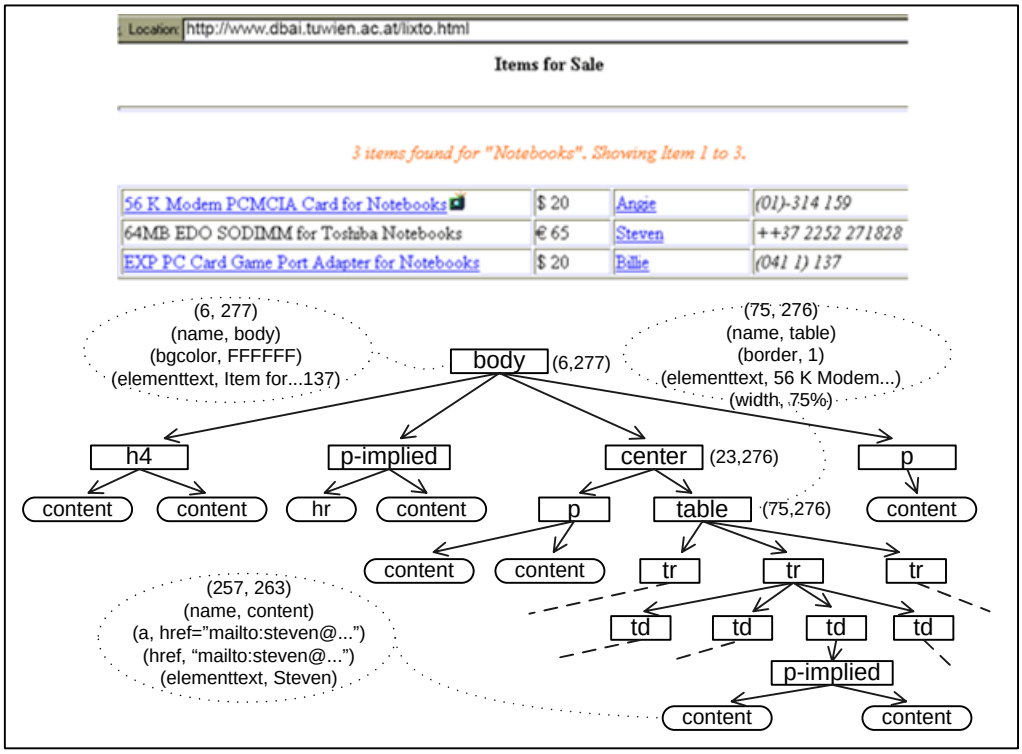


Figure 3.6: A HTML Parse Tree Based on Java Swing Parser [BFG01]

3.1.1.3 Syntactic Taggers

In this section, we introduce syntactic taggers, which label simple syntactic information for individual portions independently, however the taggers provide no relationships between each other. Such a tagger performs usually more rapidly and stably than parsers and the offered syntactic features could be helpful clues for information extraction from both grammatical and ungrammatical texts.

A Part-of-Speech Tagger As introduced in [CM98, Cal98], a part-of-speech tagger is used in RAPIER[9], to label part-of-speech tags for each words and symbols in the input string separately, e.g., whether a word is a noun, a proper noun, a verb, an adjective, a preposition, etc.

The same part-of-speech tagger is applied in KnowItAll [ECD+05] as well for identifying single noun phrases and lists of noun phrases.

Token Classes Used in SoftMealy According to [HD98, Hsu98], a prior segmentation is required in SoftMealy[10] to divide input HTML source strings into tokens. Each token

[9]RAPIER handles information extraction from semi-structured and structured texts.
[10]SoftMealy is used for information extraction from semi-structured and structured text.

is a string v assigned to a token class t and notated as $t(v)$. We show some token classes in SoftMealy as following:

- A string only consists of capital letters, e.g., CAlph(USA)
- A string only consists of small letters, e.g., OAlph(take)
- An capital letter followed by one or more small letters, e.g., C1Alph(Berlin)
- A string only consists of digits, e.g., Num(77)
- Punctuation, e.g., Punc(!)
- Control characters, e.g., NL(1)[11], Tab(3)[12]
- HTML tags, e.g., HTML(<b>)

The classes defined above could be further grouped into *word* and *nonword* token classes, which are significant for rule induction. In evidence, the first four token classes belongs to *word* and as opposite, the last three classes are *nonwords*.

3.1.2 Semantic Pre-Processing

In addition to syntactic pre-processing, semantic pre-processing is used to assign semantic classes for individual words or phrases. Semantic pre-processing could be needed in most information extraction systems that handle free text and semi-structured text.

3.1.2.1 Trigger Words

According to [Ril93, RS95], a trigger word is an anchor point, which is used to activate an extraction rule. A trigger word must be specified in a concept node definition[13] of AutoSlog. Additionally, a set of enabling conditions are defined to restrict the linguistic character of the trigger word, e.g., an active or passive voice that is used for a verbal keyword. For the sake of activating the concept node, the enabling conditions must be satisfied.

As illustrated in Figure 3.7, the concept node is triggered by the word “killed” and will be activated if it has a passive voice, i.e., phrases such as “was killed”, “were killed”, “has been killed”, etc., are able to activate the concept node.

[11] “NL(1)” denotes one newline.
[12] “Tab(3)” denotes three tabs
[13] AutoSlog identifies target information depending on a dictionary of concept node definitions.

Sentence:	"A Russian diplomat was killed near the embassy."
Concept Node:	
Name:	victim-passive-verb-subject-killed
Type:	kill
Trigger:	killed
Syntactic Expectation:	subject
Semantic Constraints:	class victim
Enabling Conditions:	passive-voice
Instantiated Concept Node:	
victim-passive-verb-subject-killed:	a Russian diplomat

Figure 3.7: A Concept Node Definition Used in the AutoSlog System for the Domain Terrorism News Stories

As presented in [ECD+05], each rule in KnowItAll comprises a set of trigger words, which are utilized as search queries to collect web pages containing the words. Then, the rule will be applied to specify the target information from the resulting pages. The trigger words are generate according to the domain-independent generic pattern used in the rule and the label for the target class.

Predicate:	City
Pattern:	NP1 “such as” NPList2
Constraints:	head(NP1) = “cities”
	properNoun(head(each(NPList2)))
Bindings:	City(head(each(NPList2)))
Keywords:	“cities such as”

Figure 3.8: An extraction rule used in KnowItAll [ECD+05]

For instance, the class `City` is labeled with “city” and “town”. Figure 3.8 [ECD+05] shows one possible rule for the extraction of cities based on the generic pattern `NP1 "such as" NPList2`. The trigger word “cities such as” is composed of the label “city” in plural form and the context string “such as” in the pattern. Analogously “towns such as” could be another search query for finding the pages including cities to be extracted.

Consequently, trigger words in KnowItAll are used not only to anchor the segments containing slot fillers in documents as in AutoSlog, but also to obtain the documents automatically.

3.1.2.2 Semantic Tagging

Semantic classes are domain-specific and could be assigned by a semantic lexicon or defined by user. For instance, in the domain terrorism news stories semantic classes <Physical Target>, <Victim>, <Perpetrator>, etc., could be defined, while the domain management succession uses semantic classes <Person Name>, <Corporate Post>, <Organization>, and so on.

With the help of semantic classes, an extraction rule or an extraction pattern is able to adapt a group of words, which share the same semantic features, instead of a single exact word. The usage of semantic classes allows an extraction rule or an extraction pattern to be more general.

A prior semantic tagging is essential for the AutoSlog [Ril93, RS95] system, as well as for CRYSTAL [SFAL95, Sod97b]. A difference in the semantic class assignment of AutoSlog and CRYSTAL is that, all semantic classes in AutoSlog are defined on the same level, in contrast, a semantic hierarchy can be given for CRYSTAL, i.e., a semantic class can cover several subclasses and itself could also be covered by another class. For example, in the domain terrorism new stories, the semantic classes <Human Target> and <Physical Target> could be subclasses of <Target>. The hierarchy allows CRYSTAL to merge concept node definitions[14] by relaxing the level of semantic classes for rule induction.

RAPIER [CM98, Cal98] and SRV [Fre98a, Fre98b] could make use of a domain-independent lexicon called WordNet [MBF$^+$90] for semantic tagging, which is a large lexical database with a great amount of words and also provides semantic hierarchy.

According to [Sod99], a semantic tagging is not indispensable but a helpful preprocessing for WHISK, in particular when it is used for natural language. WHISK integrates <Digit> and <Number> as special semantic classes and additionally allows user to define semantic classes in the form of a list of words that have the same semantic meaning for the domain. As illustrated in [Sod99], a semantic class <Bdrm> of the Rental Ad domain can be defined as a disjunction of various abbreviations or notations of the word “bedroom”:

<Bdrm> = (brs | br | bds | bdrm | bd | bedrooms | bedroom | bed)

Unlike CRYSTAL and RAPIER, no semantic hierarchy is used in WHISK.

[14] A concept node definition is an extraction rule used in CRYSTAL.

3.2 Constraints

In the following, we present various constraints that could be applied for identification of desired information, such as syntactic constraints, semantic constraints, exact word constraints, wildcards and negated constraints in section 3.2.1 to 3.2.5 respectively. Then in section 3.2.6, we distinguish visible and invisible description for boundaries of targets and their contexts.

3.2.1 Syntactic Expectation

Syntactic tags, which are supplied by syntactic analyzer, play an important role in extraction patterns that are used for information extraction from grammatical text. In this section, we describe at first information systems that use syntactic constituents, such as subject, object, etc., for the identifying target text. Then systems, which use only limited syntactic information like capitalization, part-of-speech, will be presented.

3.2.1.1 Syntactic Constituents

AutoSlog [Ril93, RS95] and CRYSTAL [SFAL95, Sod97b] are systems, which rely on syntactic constituents for information extraction from free text. A slot filler[15] specified by them must be a syntactic field, in which particular constraints are satisfied. In contrast, the usage of syntactic constituents is optional for the WHISK system [Sod99], which is also able to identifies target texts from natural language. It is allowed to make use of syntactic labels offered by BADGER or ignore them in WHISK.

Figure 3.9 shows an example rule used in WHISK for the succession management domain. The syntactic tag "{PP" requires that the slot filler[16] *Organization* should be contained in a prepositional phrase, while the syntactic tags for the slot fillers *Person_In* and *Position* are ignored. Moreover, the syntactic tag "@passive" demands a verb of passive voice in the input sentence.

In AutoSlog and CRYSTAL, various extraction patterns are required to identify a significant phrase, if it is not the same syntactic constituents of different given texts. However, WHISK makes it possible to cover some of them with a single rule by ignoring the syntactic tags.

[15] A slot is a field to extract information; a slot filler is an instance of the field. For example, a task of information extraction is to collect the name of research groups in Darmstadt University of Technology (TUD), the *Name* is a slot and "Knowledge Engineering" is a slot filler of the slot *Name*.

[16] WHISK denotes a slot filler with parentheses, which are not inside single quotes.

Sentence: "Chahram Bolouri, who used to be Nortel's president of global operations, has been named president and CEO by Air Canada Technical Services."

Pattern: * (<Person>) * '@passive' *F 'named' (<Corporate Post>) * {PP *F (<Organization>)
Output: Succession {Person_In $1} {Position $2} {Organization $3}

Succession Event:
Person_In: Chahram Bolouri
Position: President and CEO
Organization: Air Canada Technical Services

Figure 3.9: An Extraction Rule for Information Extraction from Free Text Used in WHISK

3.2.1.2 Simple Syntactic Constraints

SoftMealy [HD98, Hsu98], STALKER [MMK98, MMK99] and BWI[17] [FK00] employ token classes to describe simple syntactic features, whether a token is a capitalization, number, HTML tag, punctuation, control character, and so on. The syntactic constraints for all of SoftMealy, STALKER and BWI are only used for the representation of delimiters or separators outside the slot filler, but not for slot filler itself.

Separators[18] used in SoftMealy is in the form of a sequence of tagged tokens and token classes, while token classes in STALKER could be contained in landmarks[19], which are used for arguments of functions, such as:

- `SkipTo(landmark)` denotes that all tokens including the landmark itself are to be skipped.
- `SkipUntil(landmark)` similarly indicates the tokens that should be jumped. However, the specified landmark is not to be skipped.
- `NextLandmark(landmark)` is used to confirm that the next landmark is matched. Like the function `SkipUntil()`, the landmark itself should not be consumed.

According to [CM98, Cal98], syntactic constraints that used in RAPIER are part-of-speech tags, which must be satisfied for a successful extraction. Unlike SoftMealy and STALKER, part-of-speech tags could be used in both patterns immediately before and

[17]Boosted Wrapper Induction
[18]The invisible separators used in SoftMealy will be introduced in section 3.2.6.
[19]Landmarks used in STALKER is a sequence of tokens or token classes.

after the slot filler and the pattern of the slot filler. Furthermore, a pattern could contain pattern items that is defined to match just one word or one symbol, or pattern lists, for which a maximum length N could be identified to restrict that the pattern must contain 0 to N words or symbols. Hence, in addition to the part-of speech tags, a limitation for the number of words is also probably to be applied in RAPIER. Note that both of them are able to be used for the slot filler and as well as its context.

SRV [Fre98a, Fre98b] defines predicates for finding target texts, which describe the features that are expected for the slot filler and its contexts as well, like in the RAPIER system. SRV provides quite a few token-oriented features in two categories:

- Simple features that assign a certain value for a token, which is often a boolean value. A simple feature could be part-of speech that distinguish e.g. verb tokens and noun tokens, or the length of words, which are utilized in RAPIER as well. In addition, as in SoftMealy and STALKER, features of character type, which identify e.g. numbers and punctuation, and orthography, such as capitalization are also allowed in SRV. For instance, `capitalized(token1) = false`.

  In order to extract information from HTML source strings, HTML features are extended, e.g., the `in` features such as `in_title`, `in_b`, which have the value true, if the tokens are inside title tag and b tag respectively.

- Relational features that map a token to another. For example, if `token2` appears direct following the `token1`, the relational features between them are:

  ```
  next_token(token1) = token2
  pre_token(token2) = token1
  ```

  Similarly, relational features such as `table_next_col`, `table_pre_col`, etc., could be added to respectively capture the next and the last column in the same HTML table.

Unlike RAPIER and SRV, KnowItAll [ECD+05] allows part-of-speech constraints only for slot fillers. The constraints could identify the part-of-speech tags that should be labeled for the whole phrase, the head[20] of the phrase in the slot, or the head of each phrase, when the slot contains multiple slot fillers. For instance, the constraints `properNoun(head(each(NPList2)))` illustrated in figure 3.8 means that the head of each phrase in a NPList must be a proper noun.

[20] The head of a noun phrase is a noun or a pronoun in the phrase without regard to its determiners, adjectives, complements, etc.

3.2.1.3 Attribute Conditions for HTML Tags

Unlike the syntactic constrains we have mentioned previously, which are operated on words or terms in the input sentences or strings, the tree extraction mechanism[21] in Lixto [BFG01, GKB+04] specifies attribute conditions to describe expected attribute-value pairs for particular nodes in a HTML parse tree.

An attribute condition is 3-tuple that contains:

- the name of the attribute that should be included in the node.
- the wanted value in the form of a string or a regular expression.
- a parameter *exact*, *substr* or *regvar*, which denotes that the attribute is equivalent to the given string, the attribute contains the given string, or the attribute matches the given regular expression respectively.

A set of attribute conditions could be included in an element path definition (epd) together with a tree path. For instance, an possible element path definition `(*.td`[22]`, [(bgcolor, "yellow",` *exact*`), (width, "%",` *substr*`)])` indicates all table cells having a yellow background and using percentage for the width.

Element path definitions could be then used in Elog's[23] predicates such as:

- `extraction definition predicates`, to identify the instances to be extracted. For example, `subelem(s, epd, x)` has the value `true`, if `x` is a tree region[24] rooted at `x` contained in `s`, moreover, the element path definition `epd` is adapted.
- `context condition predicates`, to describe particular elements that must or must not occur before or after the target. For instance, `notbefore(s, x, epd, d)` is `true`, if there is no element that matches the `epd` and precedes the tree region `x` no more than the distance `d`.
- `internal conditions predicates`, to specify elements that must or must not be contained in the target. E.g. `contains(x, epd, y)` is equal to `true`, if a subtree `y`, whose root matches the `epd`, is included in the tree region `x`.

[21] Lixto provides two kinds of extraction mechanisms: tree extraction and string extraction

[22] The wildcard "*" used in a tree path will be introduced in section 3.2.4

[23] Elog [GK02] is a datalog-like language for data extraction. Elog is invisible for a Lixto user.

[24] A tree region is a part of a tree containing an internal node as root and a set of consecutive children of this root node as well as their descendants. In contrast, a subtree is rooted at an internal node of the original tree, and all descendants of the node are included in the subtree.

Consequently, like CRYSTAL, RAPIER, SRV and WHISK, constraints are able to be used for the desired target as well as the surrounding context.

3.2.2 Semantic Constraints

As long as a prior semantic tagging is done and semantic classes are labeled, semantic constraints are able to be used for helping information extraction systems to specify the desired text.

The AutoSlog system [Ril93, RS95] allows a set of hard and soft semantic constraints to restrict semantic features of the slot filler. Hard constraints must be satisfied in order to identify the filler, while soft constraints are presented as suggestions and could be violated for a successful extraction.

According to the semantic constraint[25] specified for the sentence shown in Figure 3.7, the subject to be extracted must be a word or words of the semantic class <victim>, otherwise, the concept node fails for the example sentence.

One limitation of AutoSlog is that semantic constraints are only allowed for the slot filler. The semantic features of other parts of the input sentence are worthless to identify the target information.

In contrast, CRYSTAL [SFAL95, Sod97b] and WHISK [Sod99] use semantic constraints for both slot fillers and the surrounding contexts to specify the desired information. Note that semantic constraints in CRYSTAL must be employed for syntactic constituents parsed by BADGER, while WHISK allows semantic constraints for any portion of input text.

A concept node definition of CRYSTAL, which is applied to the domain management succession, is illustrated in figure 3.10. Semantic constraints on the subject, object and prepositional phrase are included in the concept node definition, which require a word or words of semantic classes <Person Name>, <Corporate Post>, <Organization> in the subject, object and prepositional phrase respectively.

We show in figure 3.11 an extraction rule used in WHISK for information extraction from a publication page. The semantic constraints <Person> and <Year> in parentheses indicate that the slot fillers *Authors* and *Year* must contain words or phrases of the corresponding semantic classes. The classes <HTML_text_open> and <HTML_text_close>, which could be defined as:

<HTML_text_open> = (| <i>)

[25] We consider it a hard semantic constraint.

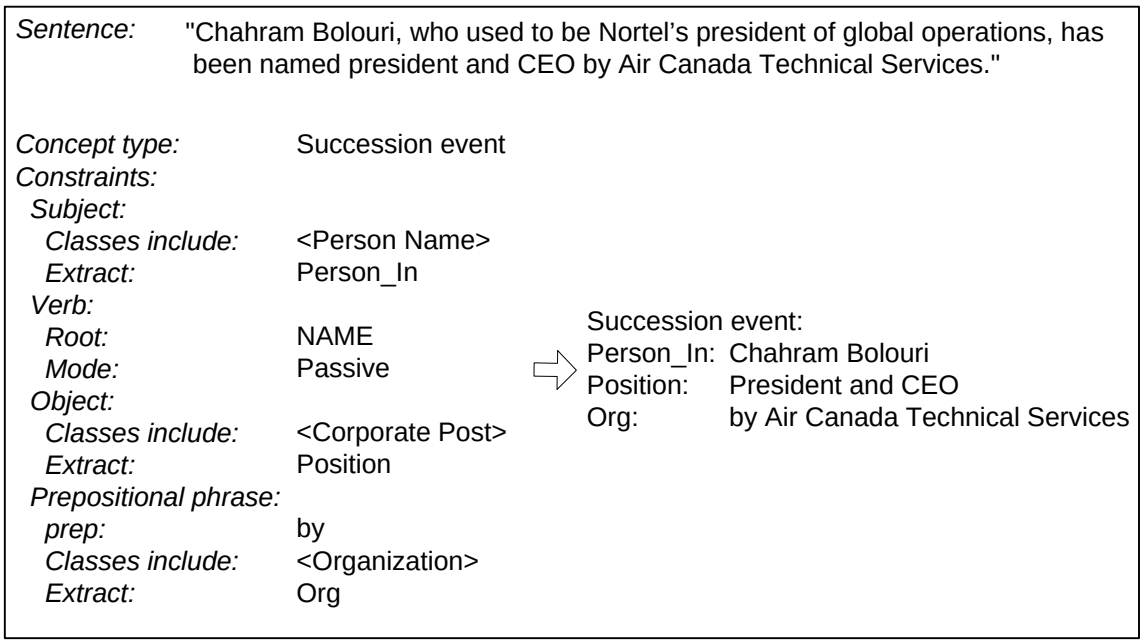


Figure 3.10: A Concept Node Definition Used in the CRYSTAL System for the Domain Management Succession

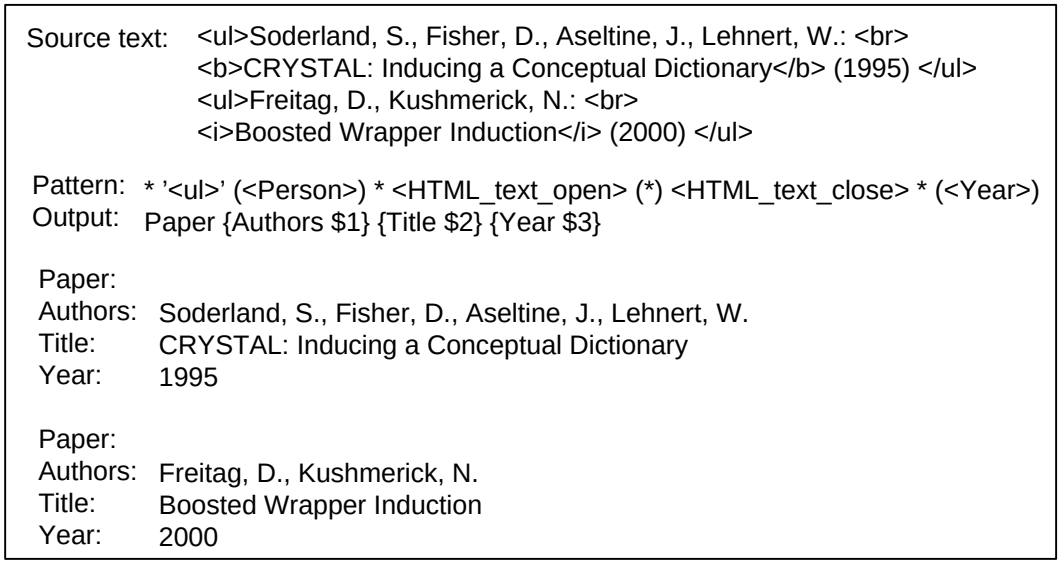


Figure 3.11: An Extraction Rule for Information Extraction from Structure Text Used in WHISK

<HTML_text_close> = (| </i>)

are used to specify the context of the target information.

Similar to CRYSTAL and WHISK, RAPIER, SRV and Lixto also allow semantic constraints for both the desired target and the context. However, unlike WHISK, RAPIER and SRV handle only single-slot extraction and the constraints are operated on the patterns for actual target information and the closest context before and after it, rather than on any part of the input text.

Semantic constraints could straightforward contained in patterns in RAPIER [SFAL95, Sod97b], while SRV [Fre98a, Fre98a] uses the simple feature `wn_word` to specify semantic constraints for a token. Unlike the simple features introduced in section 3.2.1.2, the `wn_word` feature returns a set of words that have the same semantic meaning rather than a boolean value. As presented in [BFG01, GKB+04], Lixto employs `concept condition predicates` to indicate the semantic features required for a corresponding element, e.g., `isCountry(x)`, where `x` is a string included in the target element or its context. The predicate has the value `true`, if `x` describes a country.

3.2.3 Exact Word Constraints

In addition to semantic constraints, many information extraction systems allow exact word constraints to identify the target text. In some cases, exact word constraints offer useful clues for the information extraction task. For instance, the verb "hire" or "name" is a good hint for the slot *Person_In* in the management succession domain.

As mentioned in section 3.1.2.1, AutoSlog [Ril93, RS95] requires an exact trigger word for finding the sentence containing the slot filler. Note that only the trigger word is able to be restricted by an exact word constraint in AutoSlog, other terms are all rejected from exact word constraints.

In contrast to the limitation of AutoSlog, exact word constraints in CRYSTAL, WHISK, RAPIER, SRV and KnowItAll can be used for both target texts and the surrounding contexts. Moreover, an extraction rule without any exact word constraint is also allowed in these systems.

Like semantic constraints, exact word constraints in CRYSTAL [SFAL95, Sod97b] are also based on syntactic fields. A concept node definition in CRYSTAL is defined in figure 3.10 and specifies that the exact words "name" and "by" must be contained in the verb and the prepositional phrase respectively, in order to identify the slot fillers.

The WHISK [Sod99] system allows exact words, HTML tags or punctuation constraints

for both tokens and syntactic constituents, e.g., the example rule shown in figure 3.9 requires the exact word "named" is required in the syntactic field verb in a passive voice, and the rule in figure 3.11 specifies the HTML tag "<ul>" as the left border for the slot filler *Authors*.

As opposite to CRYSTAL, RAPIER, SRV and KnowItAll operate on tokens or words rather than syntactic fields. RAPIER [CM98, Cal98] identifies the word or a sequence of words that should be encountered in a pattern and SRV [Fre98a, Fre98b] takes the simple feature `word` to indicate a particular exact word that should occur for a token. In KnowItAll [ECD+05], in addition to the exact strings that are specified, context strings in the particular pattern could be considered as implicit constraints. All exact constraints will be contained in the trigger words.

In the WIEN[26] system, which is described in [KWD97, Kus00] by Kushmerick, only extract word constraints are allowed. No syntactic and semantic information are used. Note that unlike CRYSTAL, WHISK, RAPIER, SRV and Lixto, the exact constraints in WIEN is only used for delimiters immediately outside the slot fillers, but not for terms inside.

Similar to WIEN, SoftMealy [HD98, Hsu98], STALKER [MMK98, MMK99] and BWI [FK00] either do not allow exact word constraints for the terms of slot fillers, but only separators and landmarks surrounding slot fillers are able to be specified. The exact token could be included in the representation of the context.

3.2.4 Wildcards

Wildcards are used to describe zero, one or more characters in a representation. With the usage of wildcards, an extraction rule or an extraction pattern is able to ignore unimportant information and present multiple cases with a single specification.

As presented in [Sod99], WHISK uses wildcards to skip irrelevant texts. WHISK allows two kinds of wildcards:

- "*" denotes that all characters should be skipped until the next term is found, without regards of syntactic fields, i.e., the boundaries of syntactic fields could also be skipped.

  Note that the "*" does not allow an unlimited number of matches, but only to skip to the nearest matched term. Therefore, WHISK identifies slot fillers, which

[26] Wrapper Induction ENvironment

are most likely related. Moreover, WHISK is protected from a great number of combination, when multiple wildcards are applied in an extraction rule.

- "*F" means to skip all characters inside the same syntactic field until the following term of the rule is matched.

As illustrated in figure 3.9, the first wildcard "*" in the extraction rule means that all characters involving borders of syntactic fields should be jumped until words or phrases of semantic class <Person> is found. According to the second "*", the extraction rule skips until the term "@passive", which requires a verb in a passive voice. Then the wildcard"*F" means to skip characters in the verb field to find the term "named". As the semantic class <Corporate Post> is matched, the rule skips again to look for a prepositional phrases. The last wildcard "*F" requires to skip in the prepositional phrase until phrases of <organization> is matched.

Furthermore, the wildcards used in WHISK are permitted in both slot fillers and the contexts. The "*" in parentheses illustrated in figure 3.11 identifies that all characters between the classes <HTML_text_open> and <HTML_text_close> should be extracted for the slot *Title*, while the other wildcards in this example are all used to specify the delimiters, which mean to skip until the next expected term occurs.

Wildcards are also used in SoftMealy [HD98, Hsu98], STALKER [MMK98, MMK99] and BWI [FK00] in the form of token classes. In the SoftMealy system, $t(\_)$ is used to indicate any token of the token class t, e.g., Num(_) for all numbers, Punc(_) for all punctuation, HTML(_) for all HTML tags, etc. Similarly, *Number*, *Sign*, *HtmlTag*, etc., denote the corresponding token classes in STALKER. Compared with SoftMealy and STALKER, BWI provides no wildcard for the HTML Tags, however, in addition to the capitalization, character type and punctuation, BWI allows the wildcard "*" to describe any token as in the WHISK system.

Due to the absence of expression of slot fillers in SoftMealy, STALKER and BWI, wildcards are only used for identifying separators and landmarks respectively. Wildcards, which enable a more general representation of separator and a larger coverage of instances, play a major role in rule induction.

According to [BFG01, GKB+04], the Lixto system allows wildcards to describe tree paths. A plain tree path consists of all nodes in the path from the top down and identifies a single element. In contrast, with the help of the wildcard "*", a set of matched elements are able to be specified. For example, the tree path `*.td` presents all paths up to <td> nodes. Furthermore, regular expressions are utilized in Lixto. In addtion to the tree extraction mentioned in section 3.2.1.3, Lixto [BFG01, GKB+04] offers another possibility

for information extraction, which depends on strings rather than attributes conditions. We consider a substring contained in the value of the attribute `elementtext` for each node as a string source. Lixto identifies a regular expression that is required for a string source by string path definitions (spd). Like element path definitions, string path definitions are also used in predicates we have introduced above. For instance, `extraction definition predicate` such as `subtext(s, spd, x)` is equal to `true`, if a string source `x` is contained in `s` and matches the `spd`, i.e., the given regular expression must be satisfied, where `s` is a tree region or a string source. Analogously, string path definitions could also be used in `context condition predicates` and `internal condition predicates`.

3.2.5 Negated Constraints

The constraints that have been presented in the previous sections, specify e.g. semantic features, syntactic tags, exact words, HTML tags, which must be encountered for the sake of successful targeting of slot fillers. In this section, we introduce on the other hand negated constraints, which make it possible to restrict the features that must not occur in the slot filler or the surrounding contexts.

As mentioned in section 3.2.1.2, simple features used in SRV [Fre98a, Fre98b] that provide limited syntactic information, such as character type, orthography, etc., could have a boolean value `true` or `false`, which denotes that the feature must appear or must not appear in the particular terms respectively. For instance, the predicate[27] `every(numeric false)` illustrated in figure 3.15 indicates that no numbers should be included in the slot *Position*. Negated constraints are utilized to avoid unsuitable terms for identification.

Another system that permits negated constraints is the Lixto [BFG01, BFG01]. The Elog language it used allows `notbefore/notafter conditions` to exclude a particular element in a certain range.

3.2.6 Visible vs. Invisible Borders

In this section, we show that information extraction systems could differ from each other in the boundaries they use to identify the target information. We distinguish between visible borders and invisible borders.

Visible borders are used in information extraction systems such as WHISK, WIEN, RAPIER, etc. These systems identify delimiters, which contain the context immediately preceding or following the slot filler. For instance, figure 3.13 shows that WIEN

[27] Predicates of different types used in SRV will be introduced in section 3.3.1.1.

defines the Left and Right delimiters for each slot. As presented in [HD98, Hsu98], the usage of visible borders could cause fail to specify desired texts in following cases:

- If a system has an information extraction task to collect *Authors*, *Title*, *Year* from a publication page as illustrated in figure 3.12. After the slot *Authors* is extracted, the delimiters "
" and "" are adapted for identifying the next slot filler, which could be *Year* or *Title* of a paper. It is unable to differentiate the permutations of the slots using such delimiters.
- To extract slot fillers from a string without space, visible delimiters are difficult to be specified, e.g., to obtain *Airline company* from flights like "CA935", "LH729". Such a situation is common for documents in Asian language, e.g., in Chinese.

```
<p>
<ul>Soderland, S., Fisher, D., Aseltine, J., Lehnert, W.: <br>
<b>CRYSTAL: Inducing a Conceptual Dictionary</b></ul>
<ul>Freitag, D., Kushmerick, N.: <br>
<b>2000</b> <b>Boosted Wrapper Induction</b></ul>
<ul>Nicholas Kushmerick: <br>
(2000) <b>Wrapper Induction: Efficiency and Expressiveness</b></ul>
<ul>Stephen G. Soderland: <br>
<b>Learning Text Analysis Rules for Domain-Specific Natural Language Processing</b></ul>
</p>
```

Figure 3.12: An Example HTML Source Code of a Publication Page Having Multiple Permutation of Slots

In contrast, SoftMealy [HD98, Hsu98] uses invisible borders and Finite-State Transducer[28] to extract target information from pages having multiple permutations of slots, missing slots and slots with multiple values.

SoftMealy defines contextual rules to describe separators, which identify the invisible boundary between two contiguous attributes. A separator is composed of a left context and a right context. We distinguish in SoftMealy attributes and dummy attributes. An attribute is a slot and a dummy attribute indicates tokens between two slots. For instance, an attribute is e.g., the slot *Authors* and a dummy attribute is e.g., tokens after *Authors* and before the next slot. We notate the dummy attribute as $\overline{Authors}$.

In the following we show an example contextual rule, which characterizes the separator between $\overline{Authors}$ and *Year* for the papers illustrated in figure 3.12. To describe various contexts for the same separator, a disjunction could be used. For example:

[28]The Finite-State Transducer used in SoftMealy will be introduced in section 3.3.1.2.

$s < \overline{Authors}, Year >^{L}$ = HTML(
) HTML(<b>) | HTML(
) Punc(()

$s < \overline{Authors}, Year >^{R}$ = Num(4)

Similar to SoftMealy, BWI [FK00] employs boundary detectors in the form $d =< p, s >$, where p and s are the prefix pattern and the suffix pattern that indicate the left context and the right context for the boundary respectively as in SoftMealy. Each pattern consists of a sequence of tokens and token classes. Hence, the boundary that describes the start of the slot *Year* in figure 3.12 could be `<[<br><b>], [</b>]>` or `<[<br><Punc>], [<Punc>]>`.

3.3 Representation

In this section, we firstly give an overview of different structures of information extraction systems in section 3.3.1. Then systems employ a single rule and a set of rule for extraction are contrasted. Finally, in section 3.3.3, we distinguish systems handling single-slot and multi-slot extraction.

3.3.1 Frameworks

We have introduced various constraints that could be used to identity desired information. How can we apply these constraints in information extraction systems for identification? In this section, we give an overview of possible structures of the systems, such as pattern-match rules, finite automata and classifier.

3.3.1.1 Pattern-Match Rules

Quite a few information extraction systems, such as AutoSlog, CRYSTAL, WHISK, WIEN, RAPIER and SRV make the use of pattern-match rules to acquire desired information. All constraints specified in a rule have to be satisfied for the sake of a successful identification from the input text using the rule.

As mentioned in section 3.2.1.1, extraction patterns used in AutoSlog and CRYSTAL are based on syntactic constituents.

AutoSlog [Ril93, RS95] identifies the possible slot filler with a trigger word and its enabling conditions together with the syntactic expectation in a concept node definition. Then, in order to obtain the target text successfully, the semantic constraints for the expected syntactic field must be satisfied. For instance, according to the concept node definition

illustrated in Figure 3.7, AutoSlog searches the trigger word “killed” having a passive voice at first, once it is encountered, the subject of the input sentence is to be extracted when the constraints are satisfied, i.e., the subject is a phrase of the semantic class <victim>. As a result, the string “a Russian diplomat” is identified.

In the CRYSTAL [SFAL95, Sod97b] system, as shown in figure 3.10, a concept node definition contains several syntactic fields, on which semantic constraints and exact word constraints are applied. The example concept node definition specifies that a subject, which contains words of semantic class <Person Name>, of the verb “name” having a passive voice is to be extracted as a *Person_In*, a prepositional phrase with the preposition “by” and words of <Organization> is to be extracted as an *Organization*. It shows that the concept node definition contains semantic constraints on the subject and the prepositional phrase, additionally, it also requires exact word constraints on the prepositional phrase and the verb. Note that a concept node definition in CRYSTAL does not require an explicit ordering of syntactic fields, but there is an implicit ordering according to the linguistic custom[29].

Both of AutoSlog and CRYSTAL are unable to specify exact borders of the slot fillers and therefore they are not capable of finding the exact target phrase but can only specify an entire syntactic field which contains the target information. For this reason, postprocessing is required for discarding irrelevant information of the extracted field.

In contrast, WHISK, WIEN, RAPIER and SRV are systems, which identify exact delimiters of relevant phrases rather than general syntactic fields. Thus the extracted slot fillers are accurate, and no postprocessing is needed to throw out the insignificant words.

WHISK [Sod99] constructs a rule in the form of regular expression, i.e., a sequence of terms including exact words, HTML tags, syntactic tags, semantic classes and wildcards, that should be satisfied in order to identify target information.

The “HLRT” rule used in WIEN [KWD97, Kus00] to obtain slot fillers consists of a Head delimiter, a set of Left and Right delimiter pairs for each slot and further a Tail delimiter, which are all defined with exact strings of HTML tags and punctuation.

In figure 3.13 we show an example rule used in WIEN for the given HTML source string. The Head delimiter “ ” indicates the end of the head of the input page. The Tail delimiter “ ” hints that there are no more slot fillers contained in the page. “<ul>”, “:
”, “”, “”, “(” and “)” are identified as the Left and Right delimiters for the slot fillers *Authors*, *Title* and *Year* of a single paper respectively. However, if the title of the second paper has another text style, e.g., italic instead of bold, the rule will

[29]For example, a subject is mostly before a verb.

Source text:

Soderland, S., Fisher, D., Aseltine, J., Lehnert, W.:
CRYSTAL: Inducing a Conceptual Dictionary (1995)
Freitag, D., Kushmerick, N.:
Boosted Wrapper Induction (2000)

<h, t, l1, r1, l2, r2, l3, r3>
<' ', ' ', '<ul>', ':
', '', '', '(', ')'>

Paper:
Authors: Soderland, S., Fisher, D., Aseltine, J., Lehnert, W.
Title: CRYSTAL: Inducing a Conceptual Dictionary
Year: 1995

Paper:
Authors: Freitag, D., Kushmerick, N.
Title: Boosted Wrapper Induction
Year: 2000

Figure 3.13: An "HLRT" Rule used in WIEN

be failed. In contrast, as shown in figure 3.11, a single rule is able to extract the target texts successfully with the help of semantic classes defined in section 3.2.2.

According to [CM98, Cal98], a extraction rule of RAPIER contains a pre-filler pattern, a slot filler pattern and a post-filler pattern, which are specified for the text immediately before the target information, the target itself and the text immediately after it, respectively. As mentioned in section 3.2.1.2, each pattern could be a sequence of pattern items or a pattern lists, in which exact words or symbols, part-of-speech tags or semantic classes are identified to be matched for exact one word or a list of words.

We show in figure 3.14 a RAPIER's extraction pattern that could be used to identify the slot *Post* for the example sentence. The pre-filler pattern specifies that the exact word "named" is required as the left context for the target. The filler pattern requires a sequence of words for the slot filler: a word of the semantic class <Corporate Post>, then a word with the syntactic tag "CC", which denotes a coordinating conjunction, and finally a word of <Corporate Post> again. According to the post-filler pattern, the word "by" and a following list that contains maximal four words of the semantic class <Organization> are wanted for the right context. Consequently, the phrase "president and CEO" is to be extracted.

In SRV, several predicate types are pre-defined to restrict features of the slot filler and the surrounding contexts for targeting:

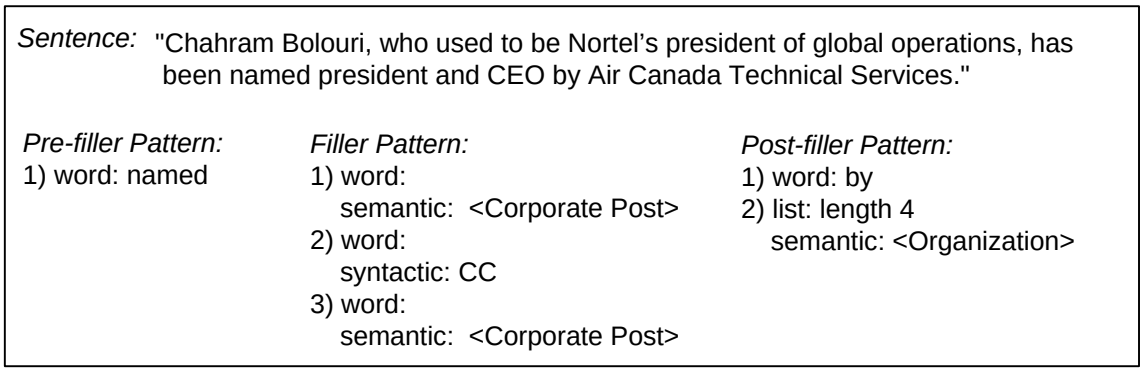


Figure 3.14: An Extraction Pattern Used in RAPIER

- `length(Relop N)`: `N` is denoted for a number. Relop $<$, $>$ or $=$ notates that the number of tokens should be less than, greater than or equal to `N` respectively.

- `some(Variable Path Feature Value)`: The path consists of a sequence of relational features. If the path is empty, the predicate requires some token insides the matching sequence to have the given value for the particular feature. Otherwise, the given feature-value pair should be matched for some token that is located by the path. For instance, the predicate `some(?A [ ] noun true)` indicates that some noun token should be contained in the matching sequence, and according to the predicate `some(?A [next_token] capitalized true)`, there should be some token that is followed by a capitalized token in the slot filler.

- `every(Feature Value)`: The feature-value pair should be adapted for each token that will be extracted.

- `position(Variable From Relop N)`: `From` contains `fromfirst` and `fromlast`, which denotes the beginning and the end of the sequence respectively. The predicate specifies the relative position of the token that is specified in `some` predicate.

- `relpos(Variable Variable Relop N)`: The predicate identifies the distance between two tokens.

A rule used in SRV is structured as a unordered set of predicates of various types, which must be satisfied for targeting the slot filler. Figure 3.15 shows a rule for identifying the slot *Position* from the given sentence. The slot filler should include less than four tokens. Numbers are not permitted. A token in the slot filler, which has the semantic meaning <Corporate Post>, is expected. Moreover, the token should be preceded by the exact

Sentence: "Chahram Bolouri, who used to be Nortel's president of global operations, has been named president and CEO by Air Canada Technical Services."

Position:
length (< 4),
every (numeric false),
some (?A [ ] wn-word "Corporate Post"),
some (?A [prev_token] word "named")

Figure 3.15: A Rule used in SRV

word "named". Hence, the string "President and CEO" is to be specified for the slot *Position*.

As distinct from the systems mentioned above, which employ independent extraction rules for targeting slot fillers, patterns in Lixto [BFG01, GKB+04] have a hierarchy order, i.e., each pattern is associated with a parent pattern. A pattern is composed of a set of Elog's rules having the form:

```
New(S, X) ← Par(_, S), Ex(S, X), Co(S, X, ...)  [a, b]
```

where `X` and `S` denote the pattern instance variable and the parent instance variable respectively. `New(S, X)` and `Par(_, S)`[30] are `pattern predicates`, the former defines the new pattern and is considered as the head of the rule, while the latter indicates the parent pattern. `Ex(S, X)` and `Co(S, X, ...)` notate the `extraction definition predicate` and the optional `conditions predicates` we have introduced in section 3.2.1.3 and 3.2.4. The `range conditions [a, b]` restricts that the a-th up to the b-th instance from the matched set are finally adapted. Such a rule specifies the instances from the parent according to the `extraction definition predicate`, where the `conditions predicates` are satisfied. For example, an example rule illustrated in [BFG01] for the page shown in figure 3.6:

```
price(S, X) ← record(_, S), subelem(S, (*.td, [(elementtext,
\Var[Y].*, regvar)]), X), isCurrency(Y)
```

means that each `price` is a table cell in a `record` instance and matches the string that begins with a currency.

[30] "_" is notated for an inessential variable.

3.3.1.2 Finite Automata

As mentioned in 3.2.6, SoftMealy [HD98, Hsu98] identifies target texts with the help of Finite-State Transducers (FST), a kind of finite automata. A FST is designed for a single information extraction task and consists of a body transducer and a tuple[31] transducer:

- Given h^L and t^R, which denote the left context of the separator for the body's head and the right context of the separator for the body's tail respectively, a body transducer is able to specify the body of the input, which contains tuples to be extracted, i.e., the string between h^L and t^R.

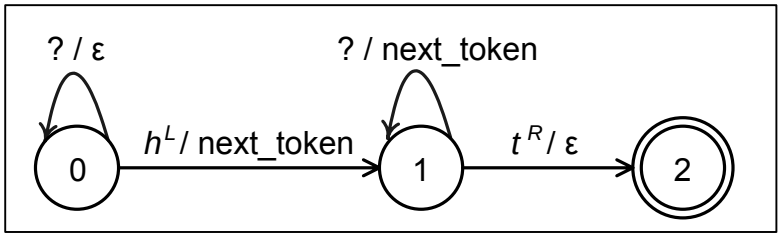


Figure 3.16: A Body Transducer Used in SoftMealy [HD98]

Figure 3.16 shows the diagram of the body transducer used in SoftMealy. For each edge, a wanted separator and a token string for output are identified. The question mark "?" denotes any token that does not meet expectation of any other outgoing transition. The "ϵ" indicates an empty output. Hence, if the input token does not match the h^L, no output will be generated according to the transition from state 0 to itself. The next_token denotes the token string immediately right next to the input separator. Due to the edge between state 0 and 1, once the h^L is encountered, the next right token string is to extracted. For the publication page shown in figure 3.12, h^L and t^R could be respectively defined as "HTML( )" and "HTML( )". The transition from state 1 to 1 indicates that tokens should be appended to the output, until the "HTLM( )" is adapted. As a result, the token strings between "HTML( )" and "HTML( )" will be extracted.

- A tuple transducer then takes the output of the body transducer as input and extract tuples step by step. The states in the tuple transducer comprise all attributes, dummy attributes and additionally a beginning state b and an end state e. If the

[31] A tuple is a sequence of related slot fillers from a single instance. For example, (*Authors*="Freitag, D., Kushmerick, N.", *Year*="2000", *Title*="Boosted Wrapper Induction") is a tuple for the publication page illustrated in figure 3.12.

FST reaches the end state e and there is no remaining input token stings, the extraction is finished. Otherwise, the FST skips to the beginning state b and tries to identify the next tuple.

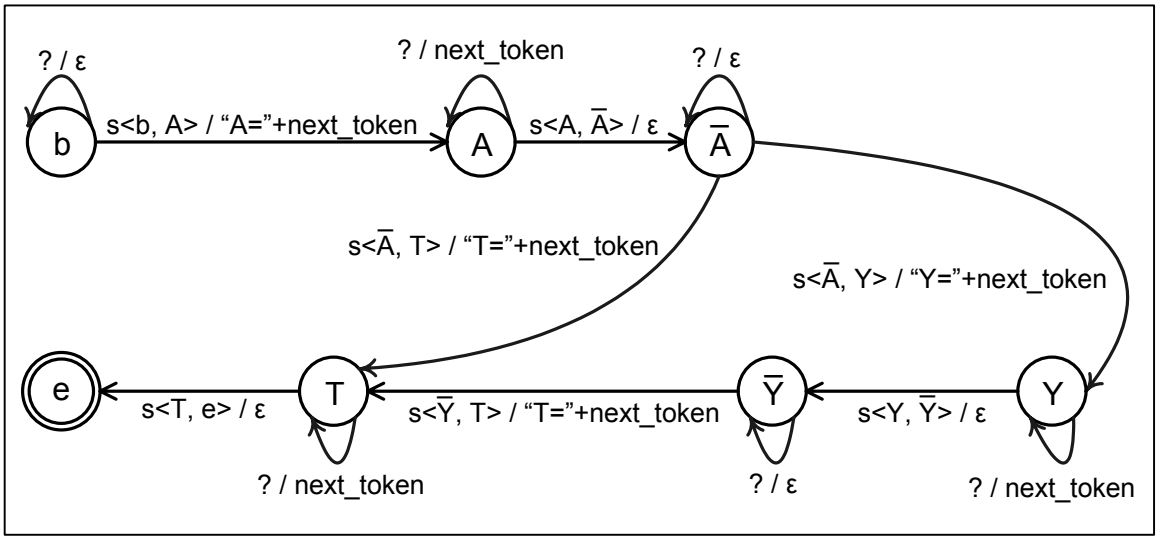


Figure 3.17: A Tuple Transducer Used in SoftMealy

In figure 3.17 we illustrate a tuple transducer for the publication page in figure 3.12. We notate the states A, Y, T for the attributes *Authors*, *Year*, *Title* respectively, and $\overline{A}$, $\overline{Y}$ for the corresponding dummy attributes. As in the body transducer, each transition in a tuple transduce is labeled with an expected input separator and an output token string. When the separator is encountered, the corresponding output is to be produced. The tuple transducer determines whether to skip or to generate output according to the states participating in the transition:

- dummy state $i \rightarrow$ non-dummy state j: The the slot filler j is to be extracted and the output should be "j="+next_token[32].
- non-dummy state $i \rightarrow$ itself: The output next_token is attached for the slot filler i.
- non-dummy state $i \rightarrow$ dummy state j: The token strings should be skipped and no output is produced.
- dummy state $i \rightarrow$ itself: Similar to the last situation, no output is generated.

[32] "+" indicates a chaining of terms before and after it.

A transition from state i to state j implies the neighborhood of the attributes i and j. A possible permutation of attributes could be represented with an individual path in the tuple transducer. Figure 3.17 shows that the tuple transducer could be used to extract (*Authors*, *Title*)[33] for the first and last tuples illustrated in figure 3.12 as well as (*Authors*, *Year*, *Title*) for the second and third tuples.

According to [MMK98, MMK99], STALKER is another system that also employ finite automata for the identification of the target. We describe in the following the rules used in STALKER and the so-called landmark automata.

A STALKER's rule is composed of a sequence of functions that have been mentioned in section 3.2.1.2. We distinguish from start rules and end rules. A start rule is applied to consume the left context of the target from its parent and starts from the beginning of the input tokens, while an end rule is contrarily used to identify the end of the target and begins with the end of the input. Any extraction rule used in STALKER could be presented with a landmark automaton. A transition from state i to state j is tagged with a landmark $l_{i,j}$, which denotes that the transition occurs, when the landmark $l_{i,j}$ is matched in the state i.

Like SoftMealy, disjunctions of rules are permitted in STALKER as well for the sake of adapting various contexts for a same target. Simple Landmark Grammars are used to represent the disjunction. A Simple Landmark Grammar has an initial state S_0 with a branching-factor k. Each of the k branches is a landmark automaton for a single rule in the disjunction and has exactly one accepting state, which indicates the start or the end of a slot filler. Each transition that starts from a non-accepting state could be either to the next state or a loop to itself, which means to skip all tokens until the next landmark is encountered.

Figure 3.18 shows an example Simple Landmark Grammar for specifying the start of the slot *Title* from a publication page. We assume that individual tuples have been divided by repeatedly using the start rule `SkipTo(<ul>)` and the end rule `SkipTo(</ul>)`. The Simple Landmark Grammar is equivalent to the disjunction `R1` or `R2`, where

`R1 = SkipTo(`*`Number`*`)SkipTo())`

`R2 = SkipTo(<br>)SkipTo(<b>)`

are adapted for the last and the first two papers illustrated in figure 3.18 respectively.

As described in [MMK99], the WIEN system represented by Kushmerick could be considered as a 1-disjunctive landmark automaton performing a single `SkipTo()` function,

[33] We ignore the dummy attributes in tuples.

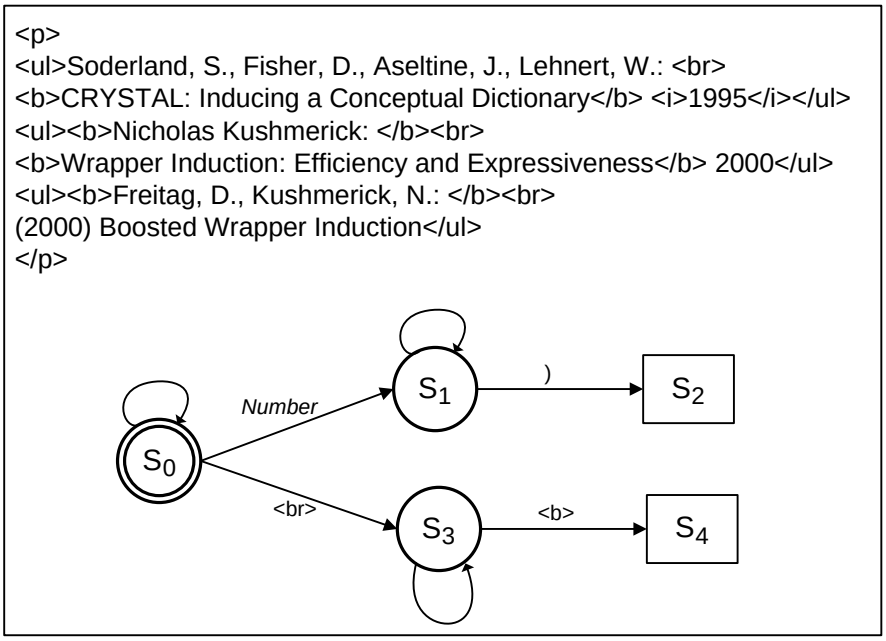


Figure 3.18: A Simple Landmark Grammar used in STALKER

and no wildcards are allowed. In contrast, SoftMealy is to be regarded as a k-disjunctive landmark automaton that makes use of wildcards. However, unlike STALKER, each rule of a disjunction in SoftMealy is either a single `SkipTo()` or a `SkipTo()NextLandmark()`, which does not allow a jump of irrelevant strings between landmarks. Consequently, WIEN and SoftMealy are less expressive than STALKER.

3.3.1.3 Classifier

According to [ECD+05], KnowItAll is comprised of two main components: Extractor and Assessor. Extractor has the task to identify slot fillers using extraction rules. As introduced in section 3.1.2.1, the Extractor at first delivers the keywords of the rule to the search engine, and then targets the slot fillers by applying the rule on the resulting pages. For example, after searching for pages containing the keyword "cities such as" within the rule displayed in figure 3.8, the Extractor specifies that the head of each noun phrase in the NPList2 is a member of the class `City` according to the rule bindings, if all constraints are satisfied.

The Assessor is used as a classifier to determine whether the extracted slot filler is valid based on the statistical computation. The pointwise mutual information[34] (PMI) score:

[34]The PMI-IR algorithm is presented in [Tur01].

$$PMI(I, D) = \frac{|Hits(D + I)|}{|Hits(I)|}$$

where I denotes the slot filler and D indicates a domain-specific discriminator for the corresponding slot, e.g., "is a city" is a possible discriminator for the class `City`, counts the proportion between the hits for the co-occurrence of the slot filler together with a certain discriminator and the single slot filler, which is usually a very very small fraction. The Assessor firstly finds a threshold of the PMI score to distinguish positive and negative instances, then converts each PMI score into conditional probabilities such as $P(PMI > threshold \mid class)$, $P(PMI > threshold \mid \neg class)$, which notate the probability that the PMI score is greater than the given threshold by a correct extraction or an incorrect extraction respectively. Finally, the probabilities are used for the input of a Naive Bayes Classifier [DP97]:

$$P(\phi|f_1, f_2, ..., f_n) = \frac{P(\phi) \prod_i P(f_i|\phi)}{P(\phi) \prod_i P(f_i|\phi) + P(\neg\phi) \prod_i P(f_i|\neg\phi)}$$

where ϕ is a class variable, $P(\phi)$ denotes the probability that ϕ happens, while f_i for $i = 1, ..., n$ are feature variables, and $P(\phi|f_1, f_2, ..., f_n)$ is the conditional probability, which indicates the probability that ϕ happens, given that $f_1, f_2, ..., f_n$ happen. If the value of the probability is greater than a given threshold τ, we consider that an extracted string is a correct instance for the particular slot. Such a threshold makes a trade-off between precision and recall. Increasing the threshold, we have a higher precision and a lower recall, and on the contrary, decreasing the threshold will raise the recall and reduce the precision.

BWI [FK00] is another system that considers the extraction task as a classification problem. An instance is correct, if

$$F(i)A(j)H(j - i) > \tau$$

where i and j denote the start and end boundaries of the instance, F and A notate sets of possible detectors for the boundaries i and j respectively. $F(i) = \sum_k C_{F_k} F_k(i)$ is the fore score, where C_{F_k} indicates the numeric confidence value of the detector k in F, $F_k(i)$ has the value 1 if the detector k is adapted for the start boundary i, the value 0 or else. The fore score presents the probability that the i is a proper start boundary. Analogously, $A(j)$ is the aft score. Further, $H(j-i)$ gives the likelihood that the slot filler has the particular length. The τ is a tradeoff between precision and recall as the probability threshold in KnowItAll.

3.3.2 A Single Rule vs. a Set of Rules

All information extraction systems introduced in this paper, except WIEN presented in [KWD97, Kus00] by Kushmerick, employ a set of extraction rules for identifying the target information.

The WIEN system allows only a single "HLRT" rule to obtain slot fillers. Hence, WIEN is just able to be used for the highly structured pages with uniform delimiters. Such a page is usually automatically generated as a query response and structured in a tabular manner.

If the year "1995" in the figure 3.13 is not inside the parentheses, it fails to extract the slot fillers. In contrast, in the other systems, multiple rules could be identified to cover various instances.

3.3.3 Single-slot vs. Multi-slot Extraction

Some of information extraction systems allow only single-slot extraction, which separately identify information from given text, while others are able to use multi-slot extraction and keep related facts together.

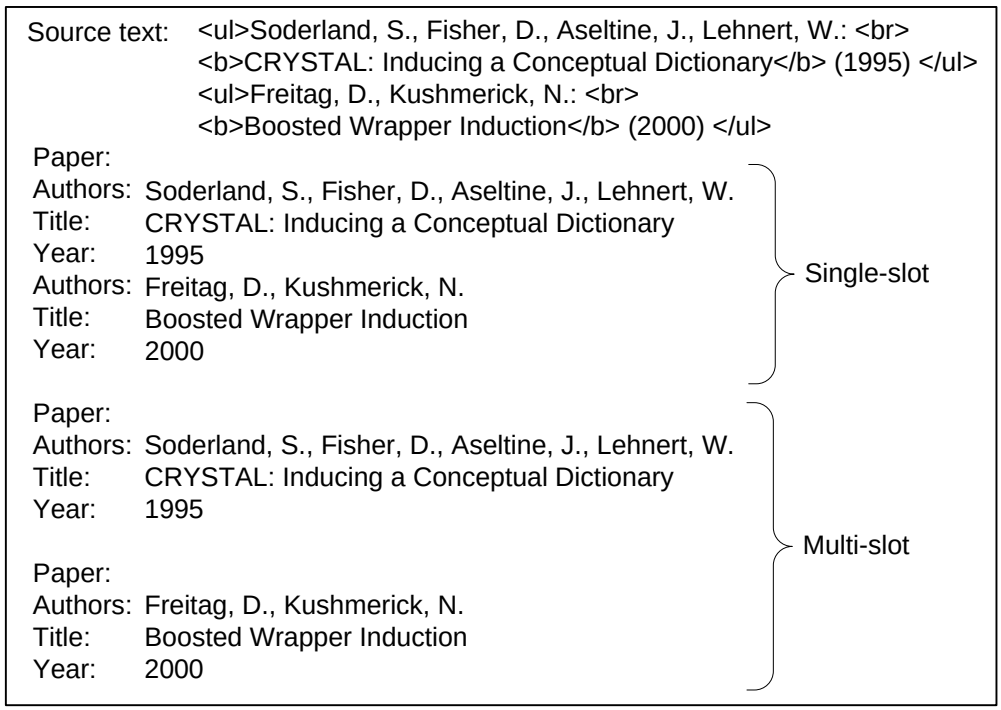


Figure 3.19: Single-slot vs. Multi-slot Extraction

Figure 3.19 shows a comparison of single-slot and multi-slot extraction. With single-slot extraction, the output only shows that, papers "CRYSTAL: Inducing a Conceptual Dictionary" and "Boosted wrapper induction" are listed on the publication page, but it is unable to get to know, which authors, title and year are related respectively. In this case, multi-slot extraction are required to identify the slots *Authors*, *Title* and *Year* as a group for each paper.

However, if only single event is described in the given text, the single-slot extraction is already sufficient. Target information for the same event could be firstly specified from the input text separately, and then combined together. For example, the slots *Name*, *Description*, *Lecturer*, *Time* and *Location*, which are related to the same course, can be extracted from a course page singly.

AutoSlog [Ril93, RS95] defines one slot per concept node definition and therefore is only able to handle single-slot extraction, i.e., the extracted facts are isolated and have no relations of each other.

In contrast, CRYSTAL [SFAL95, Sod97b] allows multiple slots to be specified in a single concept node definition. As illustrated in figure 3.10, the extraction patterns identify "Chahram Bolouri" as a *Person_In* of a *Position* "President and CEO" for an *Organization* named "Air Canada Technical Services" for the input sentence.

The WIEN system [KWD97, Kus00] handles also multi-slot extraction, but it has problem with missing attributes. For example, if the year of the first page illustrated in figure 3.13 is absent, the rule is unable to look for "(...)". Furthermore, WIEN is either not capable of handling various attribute permutation and mutiple attribute values. As mentioned in section 3.3.2, WIEN is only adapt to highly regular pages.

WHISK [Sod99] and SoftMealy [HD98, Hsu98] are both capable of multi-slots extraction. As opposite to the limitation of WIEN, the both systems are able to deal with extraction task from pages having various permutations of attributes, missing attributes and multiple attribute value. However, a prior training for all possible permutations is required for WIEN as well as for SoftMealy, i.e., they can only handle the permutations they have learned.

Additionally, KnowItAll [ECD+05] is able to handle multi-slot extraction as well, if an extraction rule is based on a n-ary extraction pattern. For instance, `<class 1> "is the" <relation> <class 2>` is a binary pattern, with the relation label "capital of", we can specify rules to extract `capitalOf(City, Country)`.

Unlike CRYSTAL, WIEN, WHISK, SoftMealy and KnowItAll, RAPIER, SRV [Fre98a, Fre98b], STALKER [MMK98, MMK99], Lixto [BFG01, GKB+04] and BWI [FK00] are

only able to handle single-slot extraction separately. They do not depend on the ordering of slot fillers, only the contexts before and after the target are considered. Therefore, it is straightforward to extract desired information from documents having various permutations of attributes or missing attributes using RAPIER, SRV, STALKER, Lixto or BWI. Furthermore, as mentioned in [CM98, Cal98], it is quite easy to extend the RAPIER for multi-slot extraction by adding patterns besides the pre-filler pattern, filler pattern and post-filler pattern. Such a rule could contain one pattern for each slot and in addition context patterns preceding the first slot, between each adjacent slots and following the last slot.

Chapter 4

Machine Learning in IE Systems

Information extraction systems could be constructed by hand, i.e., human experts are required to learn documents and implement extraction patterns for identifying target information. According to [AI99], hand-crafted systems have usually good performance for particular documents, however, they are not readily portable to documents in other formats. Therefore, developing information extraction systems by hand is not adapted for the rapid growing of the web documents in various formats and the possible structure modification for the existing pages. Furthermore, manual building is often time-consuming and error-prone. Consequently, recent information extraction systems employ machine learning techniques to automatically generate and maintain extraction patterns.

In this chapter, we concern ourselves with various machine learning techniques applied in several existent information extraction systems. At first, we present the possibilities for building a training set. Active learning, where instances are intentionally selected for training, is introduced in section 4.2. In section 4.3, we describe boosting algorithm, which repeatedly invokes a given learning algorithm in order to improve its performance. Then, in section 4.4, rule learning algorithms that are widely used for learning extraction patterns are represented. In section 4.5, we introduce bayesian learning for inducing classifier that estimates the validity of an extraction.

4.1 Type of Training Data

As presented in [Alp04], machine learning techniques differ from each other in the type of training data. We distinguish supervised learning and unsupervised learning. The training data for the former consists of input objects and the corresponding target output offered by a so-called supervisor, while for the latter only input data is provided.

4.1.1 Supervised Learning

Supervised learning are applied in most information extraction systems, such as AutoSlog, CRYSTAL, WHISK, SoftMealy, etc. In order to learn extraction patterns, a set of annotated training examples, i.e., sample documents with hand-labeled targets, are required. Generating such a tagged training set is a time consuming task, and sometimes it is not quite straightforward, specifically when labeling for natural language text.

For the sake of reducing the human work, WHISK [Sod99] interleaves tagging of training documents and learning extraction rules rather than annotates all samples in the beginning. In addition, Lixto [BFG01, GKB+04] provides a flexible way for annotation by using mouse and marking the desired phrases in browser-displayed documents.

4.1.2 Unsupervised Learning

In contrast, KnowItAll [ECD+05] employs unsupervised learning technique to bypass the requirement of tagged training set. Rather than annotating for all instances in example documents as in supervised systems, the total human effort required in KnowItAll is to offer class label for target slots. For example, in order to identify the slot *City*, only the class label "city" and "town" must be given, but no city instances are needed to be labeled. In the Bootstrapping step, extraction rules are automatically constructed by given class labels and domain-independent generic patterns. For instance, the extraction rule illustrated in figure 3.8 is generated by replacing `<class1>` with the plural form of the class label "city" in the pattern `<class1> "such as" NPList`. Additionally, no manual collection of sample documents is demanded in KnowItAll. As mentioned in section 3.3.1.3, pages containing target phrases are acquired by querying search engine for the keywords obtained from extraction rules. Moreover, to automatically generate the discriminators that are used in the Assessor for estimating the probability whether an extracted string is correct or not, human intervention is either not necessary. The Bootstrapping obtains a set of training instances by querying keywords, furthermore a set of untrained discriminators based on extraction rules and class labels. We will introduced in section 4.5, how discriminators are induced.

Another system that does not need annotated training corpus is the AutoSlog-TS [RS95, Ril96], a modified version of the AutoSlog system. And instead, a so-called pre-classified training set is provided. Each word in documents must be assigned as either relevant or irrelevant information in a particular domain. The relevant words will be considered for construction of extraction rules.

4.1.3 Semi-Supervised Learning

According to [See00], another possibility of the training set is to combine labeled and unlabeled data, commonly, a great number of unlabeled data, together with a set of labeled data are provided. In this section we introduce the Lixto [BFG01, GKB+04], which provides a visual and interactive user interface and supports user to create extraction patterns semi-automatically by simply highlighting important information with mouse and appending constraints under the guidance of the system. The user of Lixto does not need to learn anything about the internal Elog language. Elog rules are generated automatically by system according to the marked information and the conditions added by user.

We show in figure 4.1 [BFG01] the interactive pattern generation in Lixto. An interactive step by user is labeled with (I), while an automatic step by system is labeled with (A). User is responsible for:

- selecting a parent pattern. As mentioned in section 3.3.1.1, patterns in Lixto have a hierarchical order. Initially is only the root pattern `document` available. As illustrated in figure 3.6, a pattern `price` could be defined as a sub-pattern of `record`.

- selecting an instance of parent pattern containing an instance of target pattern and highlighting the target instance. Lixto creates a basic filter, determines in background the tree path for specifying the target instance and automatically creates the corresponding Elog rule[1].

- checking whether any unwanted examples appear in the result of testing the filter.

- adding conditions interactively, if the filter is too general, i.e. covers negative examples. User first decides what kind of conditions[2] is to be added to the filter. Then, for example, to add an `after condition`, user should mark the desired after element and set the distance tolerance. In background, a corresponding `condition predicate` is appended to the Elog rule.

- checking whether any target examples do not appear in the result of testing the set of induced filters.

[1] The Elog rule for a basic filter does not contain any `condition predicates`, but only the `extraction definition predicate`. However, user can define characteristic attributes for the instance, and the corresponding `attribute conditions` will be contained in `element path definition`.

[2] `Before/after conditions`, `notbefore/notafter conditions`, `internal conditions`, `range conditions` are provided by Lixto.

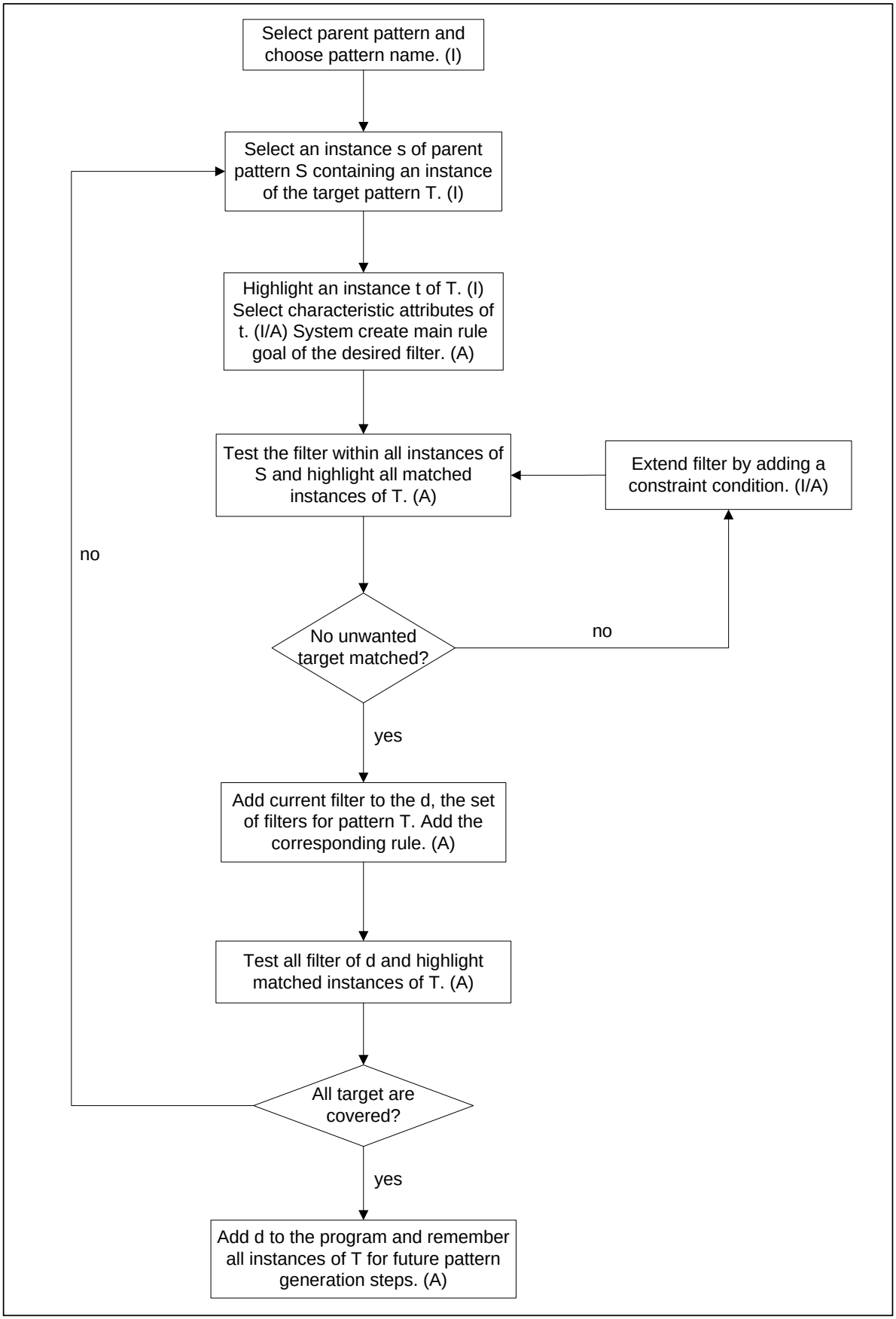


Figure 4.1: Generation a New Pattern in Lixto [BFG01]

User labels a single instance at a time, and observes the coverage of the extraction rule learned from that instance. If there remain uncovered examples considered as target, another instance will be marked for training an additional extraction rule. In this way, only a very small set of instances must be labeled by user, while the unlabeled instances are used for testing.

4.2 Active Learning

According to [CAL94], we distinguish between passive learning that selects training examples randomly, and active learning, in which the examples thought to be helpful are chosen for training.

In section 4.1.1 we have presented that hand-tagging for example documents in supervised learning spends a lot of time and human effort. WHISK [Sod99] reduces the human work by interleaving tagging and learning, i.e., in each iteration a set of unlabeled instances are chosen and passed to the user for annotation. However, randomly selected instances may not contain relevant information or may be already covered by reliable rules. Learning with a such instance does not advance the coverage and the precision of the rule set. WHISK employs selective sampling, a kind of active learning, in order to avoid useless instances and consequently to spare the human effort of worthless tagging.

In the first iteration, i.e., the rule set is empty, WHISK randomly selects unlabeled instances for training. Once a set of rules are induced, untagged instances could be divided into three classes based on the current rule set:

- instances covered by an existing rule. A positive instance increases the confidence of the corresponding rule, while a negative instance provides a counter-example. Therefore, this class of instances is beneficial to the improvement of the accuracy of the rule set.
- instance that are near misses of a rule. A near miss instance is not covered by any existing rule, but it will be covered, if an existing rule is slightly modified. Instances in this class are useful to increase the recall of the rule set.
- instances not covered by any rule. This class contributes to the completeness of the rule set.

WHISK then randomly chooses instances to be annotated from each of these three classes. By default, the three classes are same weighted, and the proportion could be defined by the user.

4.3 Boosting Algorithm

According to [FS96], boosting is a technique for improving the performance of a given simple machine learning algorithm. In each iteration, boosting invokes the give algorithm to learn a hypothesis from the current distribution over the training set, and reweights the training examples as well. A highly accurate hypothesis is then generated by combining the hypotheses induced in iterations.

In this section, we describe the information extraction system BWI [FK00], where boosting method is applied. As introduced in section 3.2.6, BWI employs simple contextual patterns to identify the target information. However, a single pattern can hardly cover all desired phrases, hence, especially in the case of extraction from natural language text, a set of patterns must be created. BWI utilizes the boosting algorithm, `AdaBoost` introduced in [FS95], to generate extraction patterns and combine them for specifying the targets. `AdaBoost` is an abbreviation of Adaptive Boosting, which means that examples learned weakly in previous iterations will be favored in the next steps by the redistribution. Moreover, the `AdaBoost` used in BWI is an improved version presented in [SS99], where confidence values are assigned to hypotheses induced in iterations and the reweight for the next step is related to the confidence of the hypothesis in the current iteration.

Assuming $D_t(i)$ is the distribution in iteration t. Initially, the Distribution D_0 over the training set[3] is uniform, i.e., for each example i, $D_0(i) = 1/m$, where m is the total number of examples in the training set. In each iteration t, a weak learner `LearnDetector` runs and returns the detector d_t and its confidence value C_{d_t}. Then the distribution for the next iteration is computed with:

$$D_{t+1}(i) = \frac{D_t(i)e^{-C_{d_t}d_t(i)(2X(i)-1)}}{N_t}$$

where $X(i) \in \{0, 1\}$ is the label of example i, N_t denotes a normalization factor. And the confidence value is computed as:

$$C_d = \frac{1}{2}ln(\frac{W_d^+ + \epsilon}{W_d^- + \varepsilon})$$

where W_d^+ and W_d^- indicate respectively the sum of the weights of training examples that are correctly and incorrectly classified.

The weak learner `LearnDetector` invoked in each boosting iteration begins with an empty detector and greedily specializes the detector with the best prefix and suffix extensions. At each step of `LearnDetector`, the functions `BestPrefixExtension` and

[3]BWI induces "fore" detector and "aft" detector separately from the training set S and E.

`BestSuffixExtension` search the best extension having a length 1 to L for the current detector, where L is the given look-ahead parameter. Compare the scores of the current detector, the current detector with the best prefix extension and with the best suffix extension, which is estimated with $score(d) = \sqrt{W_d^+} - \sqrt{W_d^-}$. If the current detector has the greatest value, i.e., no extension that increases the score of the current detector is found, the induction stops and returns the current detector. Otherwise, the best extension is to be appended to the current detector and `LearnDetector` continues to search extensions.

Note that unlike the covering algorithm we will introduce in section 4.4.3, BWI reweights the examples rather than removing the examples that are covered by an induced rule from the training set. Therefore, BWI does not stop learning even if all positive examples are covered.

4.4 Rule Learning

Rule Learning is a common used machine learning method in information extraction systems. In the following, we first compare the systems that learn rules inductively and not inductively. Then, in section 4.4.2 the system WIEN that utilizes simple heuristics for rule induction is represented. We contrast compression algorithm with covering algorithm, and learning by generalization with specialization in section 4.4.3 and 4.4.4 respectively. In section 4.4.5 heuristics used for evaluation of a rule is presented. And finally the pruning technique is described.

4.4.1 Inductive vs. No Inductive Learning

The most of information extraction systems make the use of induction to find general rules matched a set of positive examples.

However, according to [Ril93, RS95, Ril96], AutoSlog and its new version AutoSlog-TS do not have automatic inductive steps. AutoSlog searches the first clause containing the labeled target string, while AutoSlog-TS looks for clause that includes the classified relevant text. A conceptual sentence analysis is done by CIRCUS. Then conceptual anchor point heuristics are applied to select a suitable linguistic pattern and identify the anchor point and the corresponding enabling conditions. Some example heuristics are displayed in figure 4.2 [Ril93]. If the clause “a Russian diplomat was killed near the embassy” is found, where the labeled target “a Russian diplomat” is the subject of the verb “killed” in passive voice, the first heuristic in figure 4.2 is matched. Moreover the anchor point

"killed" is specified with the enabling condition that a passive verb is required.

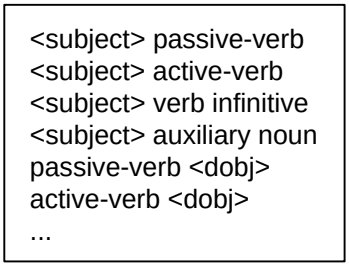


Figure 4.2: Linguistic Patterns Used in AutoSlog [Ril93]

In this way, each concept node is automatically learned from a single positive example without considering its reliability in the whole training set. Hence, many of the constructed concept nodes could be too general and identify not only the target but also irrelevant information. For this reason, post processing is needed to throw out the bad concept nodes.

4.4.2 Learning by Simple Heuristics

In this section, we describe the learning algorithm used in WIEN [KWD97, Kus00], where the output wrapper is determined by simple heuristics.

At first we introduce the algorithm for Learning "LR" wrappers, the basic wrapper presented by Kushmerick. Similar to the HLRT wrapper mentioned in section 3.3.1.1, a "LR" rule is comprised of a set of Left and Right delimiters, and could be expressed as a vector $< l_1, r_1, ..., l_k, r_k >$. Since the $2K$ delimiters for a k-slot extraction are independent of each other, we can induce the delimiters separately. For induction of each Left delimiter, a set of candidates is firstly generated by enumerating all suffixes of the shortest strings that appear to the left of the slot filler in each example. For instance, the candidate set for the Left delimiter of the second slot $Title$ for the example page showed in figure 3.13 is represented as:

$$Candidate_{l_2} = \begin{Bmatrix} ' :< br >< b >', & ' < br >< b >', & 'br >< b >', & 'r >< b >' \\ ' >< b >', & ' < b >', & 'b >', & ' >' \end{Bmatrix}$$

Analogously we create candidates for each Right delimiter by enumerating prefixes of the shortest strings occurring to the right of the slot filler in each example. Note that the Left delimiter for the first slot and the Right delimiter for the last slot are special cases,

the strings between neighboring tuples and the strings before the first tuple, or after the last tuple are to be considered respectively.

In the next step, we identify constraints in order to assure the reliability of delimiters. For instance, as presented in [Kus00],

- a candidate of a Right delimiter r_k must not be a substring of any instance of the slot filler k in any of the example pages,
- a candidate of a Right delimiter r_k must be a prefix of the text that occurs immediately following each instance of slot filler k in every example page,

and analogous constraints are specified for Left delimiters.

The delimiters are induced one by one. For each of them, the system enumerates its candidates and checks the validity of the candidate in each iteration. Once the corresponding constraints are satisfied, the system stops learning the delimiter and returns the candidate.

The "HLRT" wrapper is learned similarly, however, in contrast to the independence of the delimiters in "LR" wrapper, the Head delimiter h, the Tail delimiter t and the Left delimiter of the first slot l_1 react on each other, i.e., the validity of whichever is dependent on the selection of the other two. Moreover, additional constraints[4] [Kus00] are added to guarantee the performance of the induced wrapper, such as:

- a candidate of the Head delimiter h must be a substring of the head in every example page,
- a candidate of the delimiter l_1 must be a suffix of the portion of the head in each example page after the first occurrence of the Head delimiter h,

and so on. Therefore, these three delimiters must be learned together. The learning stops until a group of the candidates of h, t and l_1 satisfies the constraints. The other delimiters are learned separately as in a "LR" wrapper.

4.4.3 Compression vs. Covering Algorithms

As introduced in [Cal98], rule induction algorithms could be classified into compression and covering based on their structure.

[4] We introduce here only two of the constraints presented in [Kus00].

Compression algorithms start with a set of the most specific rules, one for each positive instance, and try to compress the rule set by construction general rules to substitute the specific ones that are covered, until the rule set is not able to be compressed with any rules.

Covering algorithm [FF05] is a class of machine learning technique that is widely used for rule induction in information extraction systems. The algorithm learns one rule at a time to cover a set of positive examples and reject negative examples. The process ends if all positive instances are covered. It is also named separate-and-conquer strategy, since once a rule is created, all positive instances that are covered by the rule are separated from the training set and will be not regarded in the rule generation for conquering the remaining training instance. Hence, the covering algorithms are more efficient than compression. However, compression avoids learning a rule that is quite specific by permitting the generalization of remaining instances with already induced rules.

Figure 4.3(a)[5] shows a set of original training data, where "+" and "-" denote positive and negative instances respectively. The rule $R1$ matches the positive examples insides the square as displayed in figure 4.3(b). In figure 4.3(c), these examples are removed when the rule $R1$ is generated. Further rules will be induced to cover the remaining positive examples.

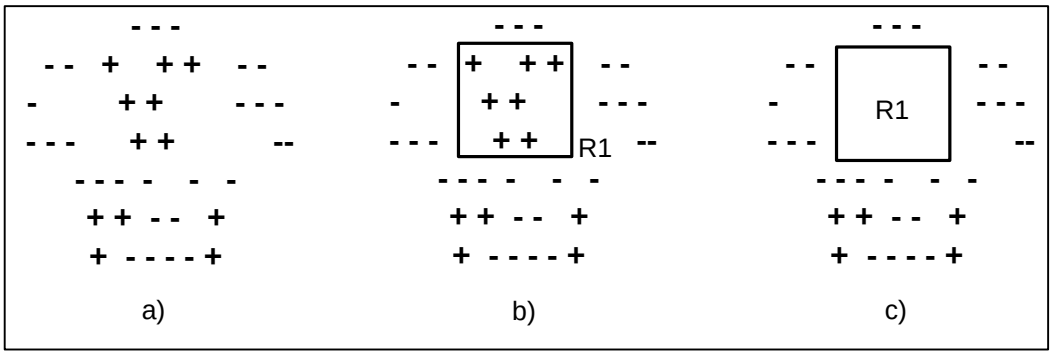


Figure 4.3: Separate-and-Conquer Rule Learning

In the following we will introduce the RAPIER [CM98, Cal98], a compression based system, which randomly selects a pair of rules and then attempts to build the generalizations of them.

CRYSTAL, WHISK, SRV and STALKER are systems that typically rely on covering algorithms. Note that except STALKER, the others are not truly separate-and-conquer,

[5]source: http://www.cs.utah.edu/classes/cs6350/slides/rule-lrn_4.pdf

but the whole training set is needed for the evaluation of a induced rule. Since hand-tagging is expensive and only a limited number of labeled instances are provided for training.

4.4.4 Generalization vs. Specialization

According to the direction of search, we distinguish between bottom-up and top-down algorithms, where a set of rules are induced by a generalization or a specialization respectively.

4.4.4.1 Bottom-Up

Bottom-up algorithms begin with a specific rule that matches a single positive instance, and successively relax the constraints to generalize the rule to cover additional positive instances as illustrated in figure 4.4[6]. In this section, we introduce and compare the information systems CRYSTAL [SFAL95, Sod97b], RAPIER [CM98, Cal98], SoftMealy [HD98, Hsu98], which operate from the bottom up.

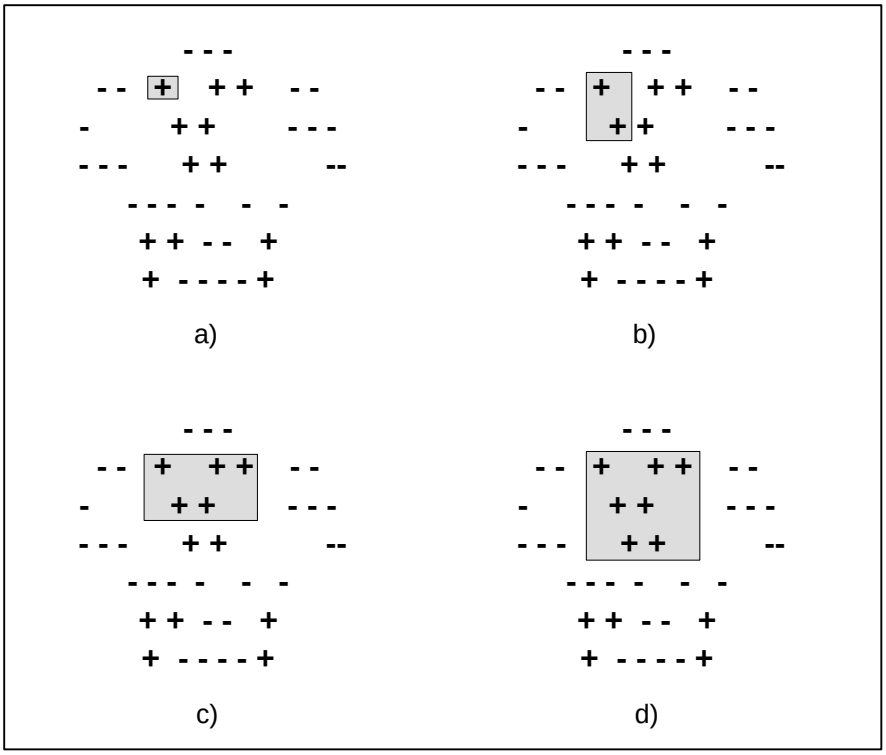


Figure 4.4: Bottom-Up Rule Learning

[6]source: http://www.cs.utah.edu/classes/cs6350/slides/rule-lrn_4.pdf

The First Rule Set CRYSTAL initializes the rule set with concept node definitions created from each positive instance[7] in the training set. Each initial concept node definition is the most specific rule containing all available syntactic and semantic features for the particular instance.

Similar to CRYSTAL, RAPIER creates the most specific rules, one for each positive instance. The filler pattern of a rule consists of word constraints and tag constraints for each word or symbol in the slot filler[8]. Analogously, pre-filler pattern and post-filler pattern of a rule contain all word and tag information describing the full context of the slot filler, i.e., they are comprised of words and symbols with the corresponding constraints for all tokens in the document that precede the filler and follow the filler respectively.

Unlike CRYSTAL and RAPIER, where an integrated rule is learned at a time, contextual rules representing separators between various attribute pairs are induced separately in SoftMealy. SoftMealy collects at first separators labeled for each same attribute pair and an initial contextual rule is described with the context tokens of one separator. In order to limit the number of the available expressions of a contextual rule, a bias is used to restrict that a sequence of tokens consists of arbitrary contiguous *nonword* tokens and at most one *word* token, which are mentioned in section 3.1.1.3. Moreover the context must not exceed the separator of any other attribute pairs. Therefore, we have the contextual rules in the form of $[word]\ nonword^*$ and $nonword^*\ [word]$ for the left and right context respectively. Note, due to the length constraint by the bias, the initial contextual rules for the head separator h and the tail separator t could be too general to correctly extract the body containing tuples of desired information. Hence, before the induction, they must be applied to extract the body for testing the reliability. If it fails, SoftMealy attaches the next left token of h^L to h^L and the next right token of t^R to t^R to specialize the rules and tests again. Until the body is exactly identified, the contextual rules of h^L and t^R can be applied for generalization.

Generalization's Methods For the generalization, CRYSTAL takes an initial concept node definition and attempts to generalize it by finding its similar concept definitions, which identify the same slots from the same syntactic constituents and require the least number of relaxations of constraints in order to merge with the current concept node definition. Exact word constraints are unified by eliminating terms except the common

[7] CRYSTAL considers the labeled instances as positive examples and all unlabeled instances as negative examples. If multiple slots are to be identified, CRYSTAL considers a group of at least two slot fillers as a positive example.

[8] The semantic tag in RAPIER will be thought during generalization, since a single word does not usually have a certain semantic meaning.

ones, while semantic constraints are merged with the common ancestor class. For instance, the word constraints "the former president" and "the president" will be relaxed with "president", and the semantic constraints <Human Target> and <Physical Target> will be unified by the class <Target>. If the unification is valid[9], it is selected as the current rule and CRYSTAL searches the similar definition for it. All concept node definitions covered by the unification are removed from the rule set. If no more similar concept definition is found, the current concept definition is considered as covering all possible similar definitions and will not be improved by more relaxations, it is appended to the rule set. The process repeats until the initial set of concept node definitions is empty.

Similar to the relaxation of semantic classes in CRYSTAL, SoftMealy generalizes contextual rules by searching the common ancestor token class according to the taxonomy trees as shown in figure 4.5 [HD98], where `CnAlph(_)` is a new class for all tokens begin with an uppercase letter. For instance, the tokens `CAlph(USA)` and `C1Alph(Canada)` in the same column could be generalized with `CnAlph(_)`. However, because of the inconsistent length of contextual rules, a preprocessing that aligns *word* tokens together and *nonword* tokens to right for left context and vice versa, is necessary before starting generalization in SoftMealy. The tokens standing in the same column will be induced.

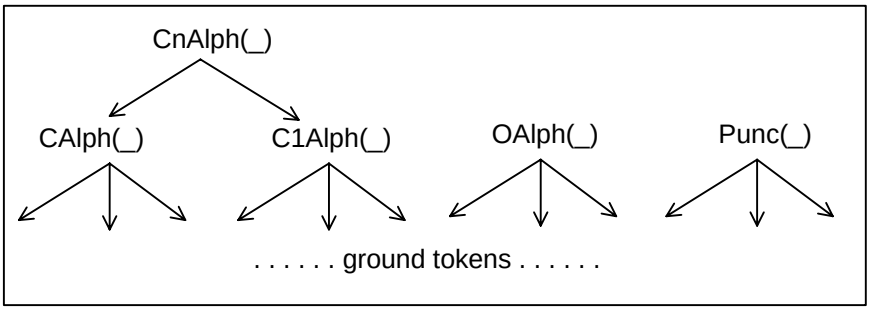


Figure 4.5: Token Taxonomy Trees [HD98]

RAPIER selects randomly two rules from the rule set and tries to find the least general generalization of them. On the principle that the most significant clues for identifying the slot filler are tokens closest to the filler, RAPIER firstly generalizes the filler patterns of the chosen rules without regard to the pre-filler and post-filler patterns, i.e., a rule with empty pre-filler and post-filler patterns is built. Two possible generalizations of filler patterns will be considered, one is a disjunction of the constraints and another is the relaxation without constraints. For instance, the word constraints {`China`} and {`Germany`} are generalized by the disjunction {`China, Germany`} and an empty word

[9]We introduce in section 4.4.6.1 the evaluation heuristics of pre-pruning for CRYSTAL.

constraint. Distinct to the CRYSTAL system, where a generalized rule is continuously to be generalized until the stopping criteria is satisfied, the least general generalization found in RAPIER is straightway added into the rule set and it could be chosen for a posterior generalization with another rule.

Note that RAPIER employs not only the bottom-up approach, but also the top-down technique. In contrast to its outer loop that begins the generalization with the most specific rules, RAPIER obtain the appropriate pre-filler and post-filler patterns using specialization in order to avoid the complex computation of the least general generalization for pre-filler and post-filler patterns due to the different and long length of them. After the filler pattern is created, RAPIER induces the pre-filler and post-filler pattern from the two base rules. RAPIER searches the generalization for three pairs of elements:

- the n tokens closest to the filler in the first base rule and the $n-1$ tokens closest to the filler in the second base rule,
- the $n-1$ tokens closest to the filler in the first base rule and the n tokens closest to the filler in the second base rule,
- the n tokens closest to the filler in the first and the second base rule,

where n is the parameter that denotes the number of tokens that will be considered for the specialization. Each suggested generalizations will be evaluated according to the heuristic represented in section 4.4.5.2. The n increments until the evaluation for the best of the proposed rules does not improve or does not cover any negative examples. In order to ensure the coverage of the best rule, RAPIER employs the rule-value metric introduce in [Coh95]. The best rule, which is satisfied with:

$$\frac{p-n}{p+n} > noiseparameter$$

will be added into the rule set, and all rules it covers will be removed from the rule set.

We will introduce in section 4.4.6 the heuristics for stopping induction of a rule. Additional generalizations are required until there is no remaining positive instance, which is not covered by any learned rule.

4.4.4.2 Top-Down

In contrast to the bottom-up systems we have introduced above, top-down systems shown in figure 4.6[10] start with a most general rule that may cover all positive examples and negative examples as well, then constraints are repeatedly appended to the rule to specialize

[10]source: http://www.cs.utah.edu/classes/cs6350/slides/rule-lrn_4.pdf

it in order to refuse negative instances. In this section, the information systems WHISK [Sod99], SRV [Fre98b], STALKER [MMK98, MMK99] and Lixto [BFG01, GKB+04] which induce extraction rules using top-down algorithms, are presented and compared.

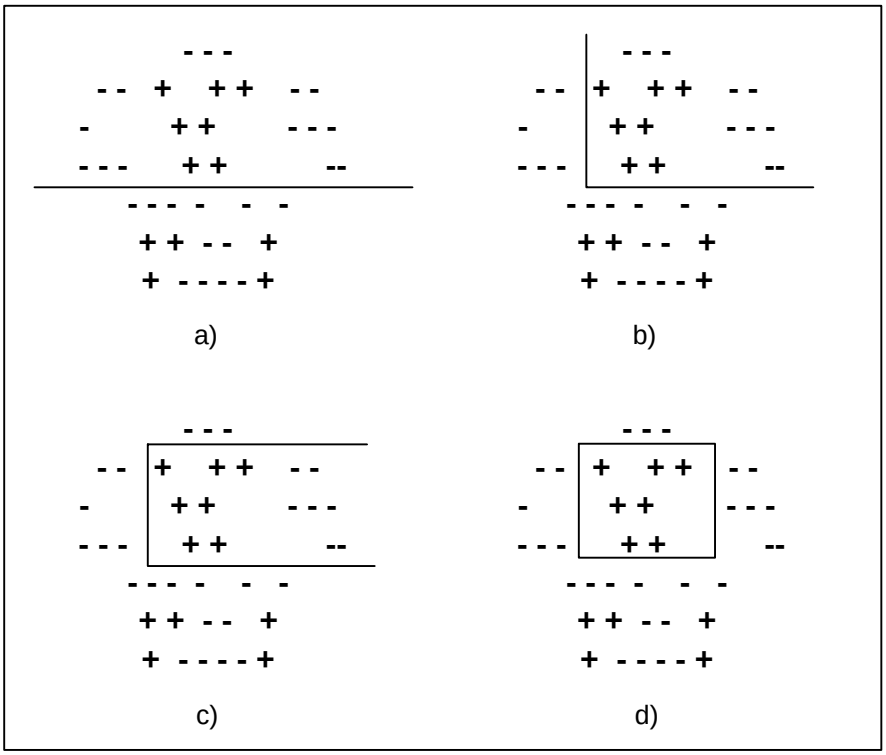


Figure 4.6: Top-Down Rule Learning

The Starting Rule SRV begins with a null rule that does not include any predicates mentioned in section 3.3.1. The null rule covers all positive and negative instances in the training set.

WHISK starts with an empty rule filling with wildcards. For instance, `* (*) * (*) *` is an empty rule for a two-slot extraction. The first wildcard means to skip an empty string, while the second wildcard extracts the whole instance for the first slot and there is no output for the second slot. Note that as distinct from a typically general rule for top-down algorithms, as in SRV, which covers all possible instances, the empty rule in WHISK hardly matches any positive instance.

As mentioned in section 3.3.1.2, each exaction rule in STALKER is equivalent to a landmark automaton, a branch of a Simple Landmark Grammar that presents a disjunction of rules. In order to induce the extraction rules that identify the target slot, STALKER begins with an empty Simple Landmark Grammar, i.e. an empty rule. Like in WHISK, the empty rule does not cover any positive examples.

Specialization WHISK selects an instance including slots that are not covered by any rule and specializes the empty rule by adding terms for identifying one slot at a time in order to successively cover the whole instance. A candidate is either the term describing the slot filler or terms representing the closest left and right contexts of the slot. For a term labeled with a semantic tag, the candidate could be the term itself or the corresponding semantic class. The proposed rules will be applied to extract the slots, which have been already induced, in the training set. The best candidate according to the Laplacian heuristic described in section 4.4.5.1 will be selected by WHISK.

As mentioned in section 3.3.3, SRV handles single-slot extraction rather than multi-slot extraction as in WHISK. SRV builds an extraction pattern by adding one predicate at a time for specifying the target slot. A candidate could be any predicate describing features of tokens. We introduce how does SRV find out the best candidate in each iteration using a gain metric as introduce and when will SRV stop appending predicate to the rule in section 4.4.5.1 and 4.4.6.1 respectively. Once a rule is successfully constructed, all instances it covers will be removed from the training set and SRV tries to build new rules for the remaining.

STALKER learns a new disjunction provided there are still some positive examples[11] in the training set. In each iteration, STALKER first initializes a candidate set with the last token of the prefixes and the corresponding token class, and additionally a terminal set with all tokens and token classes occur in every prefix example. For instances, a initial set of candidates for extraction the *Title* of papers from the example page illustrated in figure 3.18 should be {`<b>, ), HtmlTag, Symbol`}, while the terminal set is described as {`:, <br>, <b>, </b>, Symbol, HtmlTag, Number, Word`}. A best candidate landmark, which is adapted for the greatest number of positive examples or, alternatively, which matches the smallest number of negative examples, is to be selected. In order to reject negative examples the candidate landmark covers, STALKER then specializes the candidate with the tokens in the terminal set. A terminal could be appended either with a landmark refinement or a topology refinement. The former chains the terminal to the beginning the candidate landmark[12], while the latter keeps the landmark unmodified, but extends a new state. The transition from the state S_0 to the new state is labeled by the terminal. If a disjunction is perfect, i.e., the disjunction covers only positive examples, it will be added into the Simple Landmark Grammar and all examples it covers will be discarded.

[11] If T_i is a sequence of tokens and Idx_i denotes the beginning of the slot filler, the sequence $T_i[1], T_i[2], ..., T_i[Idx_i - 1]$ presenting the prefix of the slot filler is a positive example for STALKER.

[12] We introduce learning for a start rule in STALKER, for an end rule, the terminal will be added to the end of the landmark.

As illustrated in figure 4.1, a training instance in Lixto is directly marked on the example document by user with mouse. A basic filter with its associated Elog rule containing the `extraction definition predicate`, which is decided by the tree path identifying the marked instance and possibly the similar elements as well, is then automatically generated by system. Unlike in WHISK, SRV and STALKER, where a set of candidates are offered, Lixto returns one and only solution and highlights all matched instances of the filter on the example page. If the result includes any undesirable instance, i.e., the filter is too general, restricting conditions will be added interactively by user until the filter covers only positive instances. As mentioned in section 4.1.3, user is responsible for determining the condition to be added. After a filter is successfully induced and added to the pattern, Lixto marks all instances identified by filters in the pattern. If the pattern does not cover all positive examples, user should highlights an uncovered instance and a new filter is to be created. Note that different to other systems, where evaluation heuristics are employed, the coverage of filters in Lixto is estimated by the observation of user.

4.4.5 Evaluation Heuristics

In this section, we introduce various heuristics, which are applied to estimate extraction rules or patterns. During rule learning, sets of proposed rules could be generated and particular evaluation heuristics will be used to determine the best rule from the candidates for the further induction steps.

4.4.5.1 Statistical Measures

Many of information extraction systems find the best rule based on statistical measures.

In WHISK [Sod99], a modified version of Laplacian[13] is employed to select the best rule having the lowest error:

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

where e denotes the number of extraction errors and n indicates the total number of extractions in the training set, or in other words, the rule that covers the most positive examples will be selected for the following refinement.

As presented in SRV [Fre98b], a gain metric as FOIL introduced in [Qui90] is used. In order to obtain the gain of predicates, the function:

[13]The original Laplacian is expressed as $\frac{e+1}{n+2}$.

$$I(S) = -log_2 \frac{P(S)}{P(S) + N(S)}$$

will be computed for the proportion of the positive and negative examples, where $P(S)$ and $N(S)$ denote the number of positive instances and negative instances in the training set S respectively. The gain of the predicate A is defined as:

$$Gain(A) = P(S_A)(I(S) - I(S_A))$$

where S_A indicates the set of instances covered by the predicate A. It indicates the utility of the predicate A. The candidate predicate that has the best gain value, i.e., the candidate has the greatest improvement, will be attached to the extraction pattern. Note that in order to avoid a rule that covers only few positive instances and save the effort on learning such a rule, SRV does not consider any candidate covering less than five positive examples. The number could be defined by user.

4.4.5.2 Description Length

The rule evaluation metric in RAPIER [CM98, Cal98] considers not only the coverage of positive examples in the training set based on statistics, but also the complexity of a rule based on the minimum description length, which is presented in [J.R78, QR89]. The evaluation is estimated as:

$$ruleVal = -log_2(\frac{p+1}{p+n+2}) + \frac{ruleSize}{p}$$

The first part computes the Laplace that evaluates the proportion of the positive examples the rule covers, where p and n indicate the number of correct extraction and the error extraction respectively. The second part calculates the length of the rule[14]. In RAPIER, a simple heuristic for rule size computation is used, e.g., a pattern item counts 2, a pattern list counts 3, etc. If one rule covers as many positive and negative examples as another, RAPIER prefers the simpler one.

4.4.6 Pruning

As described in [Für97], pruning techniques for the covering learning algorithms are applied to keep the induced rules away from overfitting problem, which means that a complicated rule could has a low coverage and only help to turn down negative examples. Most

[14]The division by the number of positive examples that are covered is used to avoid the too much weight on the second part.

of the pruning techniques we introduce in the following are based on statistical measures. The two standard pruning approaches are pre-pruning and post-pruning.

4.4.6.1 Pre-Pruning

Improving a rule until it works perfectly on the training examples and makes no errors could cause the overfitting problem. Pre-pruning approaches are utilized during the learning process to stop the refinement of a rule according to a certain evaluation heuristic, although the rule makes still some errors in the training set. In this section, we represent the pre-pruning techniques used in information extraction systems.

CRYSTAL employs the pre-pruning algorithm. As presented in [SFAL95, Sod97b], each suggested generalization must be tested for the extraction error rate on the entire training set. When the part of the negative examples it covers exceeds the given error tolerance, the relaxation stops and the last generalization within the tolerance is added into the rule set and another initial concept definition will be chosen to be generalized. Otherwise, CRYSTAL keeps on relaxing the current one by searching for a new similar concept node definition.

Pre-pruning is also used in the WHISK [Sod99] system. The Laplacian error rate mentioned in section 4.4.4.2 is applied not only for finding the optimal candidate, but also as the stopping criteria. Each proposed rule must be tested in the training set, if the predefined threshold for the Laplacian error rate is not exceeded, WHISK continues specializing the rule by adding more terms, or else WHISK stops expending the rule. Distinct to the error rate used in CRYSTAL, the Laplacian counts the coverage of negative examples beginning with 1 rather than 0. For instance, a rule covers 19 positive examples and 1 negative example and has the Laplacian 0.095 is considered better than the rule that covers 5 positive examples with no errors and has the Laplacian 0.167. The latter will be thrown away by the error tolerance 0.1. Therefore, Laplacian prefers a higher coverage of positive examples rather than error free.

Similar to WHISK, SRV [Fre98b] uses also the same heuristic for pre-pruning as well as for choosing the best candidate predicate, i.e., the gain metric introduced in 4.4.5.1 is utilized to determine when to stop adding predicates to an extraction pattern. The specialization of the rule stops either when the extraction rule does not cover any more negative instances, or all of the candidates have a negative gain value.

4.4.6.2 Post-Pruning

In contrast to pre-pruning, post-pruning algorithms first ignore the overfitting problem and after all rules are learned, heuristics are used to simplify the complete rule set by discarding unsuitable rules. Thus, post-pruning is thought less efficient than pre-pruning approach, since it spends more time for learning overfitting rules. In this section, information extraction systems that employ post-pruning techniques are introduced.

In section 4.4.6.1 we have described the pre-pruning algorithm used in WHISK [Sod99], in addition, WHISK also makes the use of post-pruning technique. After the whole rule set is created, WHISK throws out the rules, whose Laplacian error rate exceeds the predefined threshold.

As mentioned in section 4.4.1, AutoSlog and its new version AutoSlog-TS [Ril93, RS95, Ril96] do not learn rules inductively, but they create a concept node definition from a single instance. Thus, a lot of the concept node definitions may not be adapted to the identification of target information, e.g., a concept node "X took" is not a good pattern for the domain terrorism news stories, because it is too general and often appears in various texts. In the AutoSlog system, a post-pruning will be done manually by human. A person is responsible for refusing bad concept node definitions. Unlike in AutoSlog, AutoSlog-TS does not request human effort on post-pruning. AutoSlog-TS applies the concept nodes in the training set and calculates the relevancy rate of each concept node that denotes the proportion of the occurrence in relevant texts. The concept node definitions having a high relevancy rate should be appropriate rules for the particular domain.

4.5 Bayesian Learning

We have introduced in section 3.3.1.3, the Assessor of KnowItAll [ECD+05] employs a Naive Bayes Classifier in order to estimate whether an extracted phrase is valid before appending the extraction to knowledge base of KnowItAll. For the estimation, we have to learn the $P(f_i|\phi)$ and $P(f_i|\neg\phi)$.

As mentioned in section 4.1.2, KnowItAll uses unsupervised learning technique that excludes human intervention. The candidate training instances of the target class are obtained by sending the query generated from keywords in rules to search engine. In order to ensure the reliability of instances, only the instances, whose hit count exceeds a given h, will be accepted. These instances are used for training discriminators, for each discriminator additionally a threshold τ for PMI scores that distinguishes positive and negative

examples[15] and the conditional probabilities $P(PMI > \tau|class)$, $P(PMI > \tau|\neg class)$, $P(PMI \leq \tau|class)$ and $P(PMI \leq \tau|\neg class)$.

A set of untrained discriminators are initialized from rules and class names. In the first iteration, $PMI(c, u)$ for each instance c and each untrained discriminator u are calculated. The m instances, which have the best average PMI score, will be used in the first iteration. We take half of the instances and also the same number of negative examples to induce the threshold for PMI score for each discriminator and another half of the m instances are applied to compute conditional probabilities we presented above. The best k discriminators, which clearly differentiate the positive and negative examples are to be selected for Assessor. To retrain the discriminators in the next iterations, the instances, which have highest probabilities of the trained discriminators, are to be chosen for training.

[15]Negative examples of class A could be positive examples of class B.

Chapter 5

Conclusion

Information extraction has the task to locate wanted phrases from given texts and to fill them into structured format. With the rapid growth of the world wide web and the amount of online documents, information extraction is widely applied to handle natural language as well as web pages. In this thesis, we have focused on the two sub-problem of information extraction: features used to identify targets and how to build an information extraction system.

Before extraction patterns are applied for identification, preprocessing such as syntactic and semantic analysis could be required. For instance, sentence analyzers or part-of-speech tags are often necessary for systems handling free text such as new stories, while web pages could be parsed based on HTML tags, either into a tree-like structure or a plain list. Semantic classes are either defined by user or determined by existing lexicons. Once syntactic and semantic tags are labeled, syntactic and semantic constraints are able to be used. Moreover, exact word constraints, wildcards, negated constraints are also helpful clues to describe features of the target. Additionally, we have distinguished visible and invisible boundaries that are used to separate target phrases and the contexts. Various combinations of features, such as pattern-match rules, finite automata and classifiers have been differentiated as well. We have also compared systems using a single rule and a set of rules for extraction, and systems handle single-slot and mulit-slot extraction as well.

Due to frequent changes on world wide web, flexible building information extraction system is essential, in order to easily adapt the system for different domains. We have presented three possible learning scenarios: supervised learning with a set of labeled training examples, unsupervised learning where the training set is not labeled, and semi-supervised learning having a limited set of labeled training examples together with a great number of unlabeled data. In order to increase the learning performance, active learning could be used to select examples considered more helpful for training. Moreover, boosting algorithm has been introduced, which advances performance of a given machine

learning algorithm by repeated invoking it. We have also compared information extraction systems relying on rule learning and distinguished their structures: a compression or a covering algorithm, a generalization or a specialization, and evaluation heuristics and pruning techniques they use, as well. In addition, bayesian learning has been described.

List of Figures

Bibliography

[AI99] Douglas E. Appelt and David J. Israel. Introduction to Information Extraction Technology, 1999. A tutorial prepared for IJCAI-99.

[Alp04] Ethem Alpaydin. *Introduction to Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, October 2004.

[BFG01] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual Web Information Extraction with Lixto. In *The VLDB Journal*, pages 119–128, 2001.

[CAL94] David A. Cohn, Les Atlas, and Richard E. Ladner. Improving Generalization with Active Learning. *Machine Learning*, 15(2):201–221, 1994.

[Cal98] Mary Elaine Califf. Relational Learning Techniques for Natural Language Extraction. Technical Report AI98-276, 1, 1998.

[Car97] Claire Cardie. Empirical Methods in Information Extraction. *AI Magazine*, 18(4):65–80, 1997.

[CL96] James Cowie and Wendy Lehnert. Information Extraction. *Commun. ACM*, 39(1):80–91, 1996.

[CM98] Mary Elaine Califf and R. J. Mooney. Relational Learning of Pattern-Match Rules for Information Extraction. In *Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*, pages 6–11, Menlo Park, CA, 1998. AAAI Press.

[Coh95] William W. Cohen. Fast Effective Rule Induction. In *Proc. of the 12th International Conference on Machine Learning*, pages 115–123, 1995.

[DP97] Pedro Domingos and Michael Pazzani. On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning*, 29(2-3):103–130, 1997.

[ECD+05] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S.Weld, , and Alexander Yates. Unsupervised Named-Entity Extraction from the Web: An Experimental Study. *Artificial Intelligence*, (1):91–134, 2005.

[Eik99] Line Eikvil. Information Extraction from World Wide Web - A Survey. Technical Report 945, Norweigan Computing Center, 1999.

[FF05] Johannes Fürnkranz and Peter A. Flach. ROC 'n' Rule Learning - Towards a Better Understanding of Covering Algorithms. *Machine Learning*, 58(1):39–77, 2005.

[FK00] Dayne Freitag and Nicholas Kushmerick. Boosted wrapper induction. In *AAAI/IAAI*, pages 577–583, 2000.

[Fre98a] Dayne Freitag. Information Extraction from HTML: Application of a General Machine Learning Approach. In *AAAI/IAAI*, pages 517–523, 1998.

[Fre98b] Dayne Freitag. *Machine Learning for Information Extraction in Informal Domains*. PhD thesis, Carnegie Mellon University, 1998.

[FS95] Yoav Freund and Robert E. Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. In *European Conference on Computational Learning Theory*, pages 23–37, 1995.

[FS96] Yoav Freund and Robert E. Schapire. Experiments with a New Boosting Algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.

[FSM+95] David Fisher, Stephen Soderland, Joseph McCarthy, Fangfang Feng, and Wendy Lehnert. Description of the UMass System as Used for MUC, 1995.

[Für97] Johannes Fürnkranz. Pruning algorithms for rule learning. *Machine Learning*, 27(2):139–171, 1997.

[GK02] Georg Gottlob and Christoph Koch. Monadic Datalog and the Expressive Power of Languages for Web Information Extraction. In *Symposium on Principles of Database Systems (PODS)*, pages 17–28, 2002.

[GKB+04] Georg Gottlob, Christoph Koch, Robert Baumgartner, Marcus Herzog, and Sergio Flesca. The Lixto Data Extraction Project - Back and Forth between Theory and Practice. In *Proceedings of the Symposium on Principles of Database Systems (PODS-04)*, 2004.

[HD98] Chun-Nan Hsu and Ming-Tzung Dung. Generating Finite-State Transducers for Semi-Structured Data Extraction from the Web. *Information Systems*, 23(8):521–538, 1998.

[Hsu98] Chun-Nan Hsu. Initial Results on Wrapping Semistructured Web Pages with Finite-State Transducers and Contextual Rules. In *Workshop on AI and Information Integration, in conjunction with the 15'th National Conference on Artificial Intelligence (AAAI-98)*, pages 66–73, 1998.

[ICC+05] Neil Ireson, Fabio Ciravegna, Mary Elaine Califf, Dayne Freitag, Nicholas Kushmerick, and Alberto Lavelli. Evaluating Machine Learning for Information Extraction. In *Proc. International Conference on Machine Learning (ICML)*, 2005.

[J.R78] J.Rissanen. Modeling by Shortest Data Description. *Automatica*, 14:465–471, 1978.

[Kus00] Nicholas Kushmerick. Wrapper Induction: Efficiency and Expressiveness. *Artificial Intelligence*, 118(1-2):15–68, 2000.

[KWD97] Nickolas Kushmerick, Daniel S. Weld, and Robert B. Doorenbos. Wrapper Induction for Information Extraction. In *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.

[Leh91] Wendy G. Lehnert. *Symbolic/Subsymbolic Sentence Analysis: Exploiting the Best of Two Worlds*. J. Barnden and J. Pollack, editors, Advances in Connectionist and Neural Computation. Ablex Publishers, 1991.

[MBF+90] George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller. Introduction to WordNet: An On-Line Lexical Database. *International Journal of Lexicography*, 3(4):235–312, 1990.

[MMK98] Ion Muslea, Steve Minton, and Craig Knoblock. STALKER: Learning Extraction Rules for Semistructured, Web-based Information Sources. In *Proceedings of AAAI-98 Workshop on AI and Information Integration*. AAAI Press, 1998.

[MMK99] Ion Muslea, Steve Minton, and Craig Knoblock. A Hierarchical Approach to Wrapper Induction. In *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 190–197. ACM Press, 1999.

[MP97] Elaine Marsh and Dennis Perzanowski. Muc-7 Evaluation of IE Technology: Overview of Results. In *Proceedings of the Seventh Message Understanding Conference*, 1997.

[Mus99] Ion Muslea. Extraction Patterns for Information Extraction Tasks: A Survey. In *Proceedings of AAAI Workshop on Machine Learning for Information Extraction*, 1999.

[QR89] J. Ross Quinlan and Ronald L. Rivest. Inferring Decision Trees Using the Minimum Description Length Principle. *Information and Computation,*, 80:227–248, 1989.

[Qui90] J.R. Quinlan. Learning Logical Definitions from Relations. *Machine Learning*, 5:239–266, 1990.

[Ril93] Ellen Riloff. Automatically Constructing a Dictionary for Information Extraction Tasks. In *Proc. of AAAI-93*, pages 811–816, 1993.

[Ril96] Ellen Riloff. Automatically Generating Extraction Patterns from Untagged Text. In *AAAI/IAAI, Vol. 2*, pages 1044–1049, 1996.

[RS95] Ellen Riloff and Jay Shoen. Automatically Acquiring Conceptual Patterns without an Automated Corpus. In *Proceedings of the Third Workshop on Very Large Corpora*, pages 148–161, 1995.

[SC93] Beth M. Sundheim and Nancy A. Chinchor. Survey of Message Understanding Conferences. In *Proceedings of the workshop on Human Language Technology*, 1993.

[See00] Matthias Seeger. Learning with Labeled and Unlabeled Data. Technical report, Institute for ANC, Edinburgh, UK, 2000.

[SFAL95] Stephen Soderland, David Fisher, Jonathan Aseltine, and Wendy Lehnert. CRYSTAL: Inducing a Conceptual Dictionary. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1314–1319, 1995.

[Sod97a] 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.

[Sod97b] Stephen G. Soderland. *Learning Text Analysis Rules for Domain-Specific Natural Language Processing.* PhD thesis, 1997.

[Sod99] Stephen Soderland. Learning Information Extraction Rules for Semi-Structured and Free Text. *Machine Learning*, 34(1-3):233–272, 1999.

[SS99] Robert E. Schapire and Yoram Singer. Improved Boosting Using Confidence-rated Predictions. *Machine Learning*, 37(3):297–336, 1999.

[ST93] Daniel D. K. Sleator and Davy Temperley. Parsing English with a Link Grammar. In *Third International Workshop on Parsing Technologies*, 1993.

[Tur01] Peter D. Turney. Mining the Web for Synonyms: PMI–IR versus LSA on TOEFL. *Lecture Notes in Computer Science*, 2167:491–502, 2001.

Index