Extraction Patterns for Information Extraction Tasks: A Survey

Ion Muslea
Information Sciences Institute and Department of Computer Science
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90230, USA
{muslea, minton, knoblock}@isi.edu

Abstract

Information Extraction systems rely on a set of *extraction patterns* that they use in order to retrieve from each document the relevant information. In this paper we survey the various types of extraction patterns that are generated by machine learning algorithms. We identify three main categories of patterns, which cover a variety of application domains, and we compare and contrast the patterns from each category.

Introduction

Information Extraction (IE) is concerned with extracting the relevant data from a collection of documents. For instance, a typical IE task might be to find management changes reported in the Wall Street Journal or to identify the targets of terrorist attacks reported in the newspapers. A key component of any IE system is its set of *extraction patterns* (or *extraction rules*) that is used to extract from each document the information relevant to a particular extraction task. As writing useful extraction patterns is a difficult, time-consuming task, several research efforts have focused on learning the extraction rules from training examples provided by the user. In this paper, we review several types of extraction patterns that are generated by machine learning algorithms. We begin by analyzing the extraction rules used for free text documents, and we continue with the rules that can be applied to more structured types of online documents.

IE From Free Text

In this section we review extraction patterns that are used only to process documents that contain grammatical, plain text. Such extraction rules are based on syntactic and semantic constraints that help identify the relevant information within a document. Consequently, in order to apply the extraction patterns below, one has to pre-process the original text with a syntactic analyzer and a semantic tagger.

AutoSlog

AutoSlog (Riloff 1993) builds a dictionary of extraction patterns that are called *concepts* or *concept nodes*. Each AutoSlog concept has a *conceptual anchor* that activates it and a *linguistic pattern*, which, together with the set of *enabling conditions*, guarantees its applicability. The conceptual anchor is a triggering word, while the enabling conditions represent constraints on the components of the linguistic pattern.

For instance, in order to extract the *target of the terrorist attack* from the sentence

The Parliament was bombed by the guerrillas.

one can use a concept that consists of the triggering word *bombed* together with the linguistic pattern `<subject> passive-verb`. Applying such an extraction pattern is straightforward: first, the concept is activated because the sentence contains the triggering word *bombed*; then the linguistic pattern is matched against the sentence and the `subject` is extracted as the *target* of the terrorist attack.

AutoSlog uses a predefined set of 13 linguistic patterns; the information to be extracted can be one of the following syntactic categories: subject, direct object, or noun phrase. In general, the triggering word is a verb, but if the information to be extracted is a noun phrase, the triggering word may also be a noun.

In Figure 1, we show a sample concept node. The *Name* slot is a concise, human readable description of the concept, and it consists of the information to be extracted (i.e., the *target* of a terrorist attack), the linguistic pattern "`subject passive-verb`", and the triggering word *bombed*. The *Trigger* slot defines the conceptual anchor, while the *Variable Slots* specify that the information to be extracted is the subject of the sentence. Finally, the subject must be a physical target (see "Constraints:"), and the enabling conditions require the verb to be used in its passive form.

LIEP

LIEP (Huffman 1995) is a learning system that generates multi-slot extraction rules. That is, rather than learning one extraction pattern for each item of interest in a sentence (e.g., target and perpetrator), LIEP generates a single rule for all items of interest. For instance, let us again consider the sentence

The Parliament was bombed by the guerrillas.

The LIEP extraction rule from Figure 2 extracts *both* the target and the perpetrator. The pattern consists of two sets of predicate-rules: the *syntactic constraints* (e.g., TRGT is

CONCEPT NODE:

Name:	target-subject-passive-verb-bombed
Trigger:	bombed
Variable Slots:	(target (*S* 1))
Constraints:	(class phys-target *S*)
Constant Slots:	(type bombing)
Enabling Conditions:	((passive))

Figure 1: **Example of AutoSlog concept node.**

TARGET-was-bombed-by-PERPETRATOR:
noun-group(TRGT, head(isa(physical-target))),
noun-group(PERP, head(isa(perpetrator)))
verb-group(VG, type(passive), head(bombed))
preposition(PREP, head(by))

subject(TRGT, VG),
post-verbal-prep(VG, PREP),
prep-object(PREP, PERP)
$\Longrightarrow$ bombing-event(BE, target(TRGT), agent(PERP))

Figure 2: **A LIEP extraction pattern.**

the subject of a sentence that also contains a verb group followed by a prepositional phrase) and the *semantic constraints* (e.g., TRGT is a "physical-target", the verb "bomb" is used in its passive form, and the prepositional phrase starts with "by").

PALKA

The PALKA system (Kim & Moldovan 1995) learns extraction patterns that are expressed as *frame-phrasal pattern structures* (for short, FP-structures). As shown in Figure 3, an FP-structure consists of a *meaning frame* and a *phrasal pattern*. Each slot in the meaning frame defines an item-to-be-extracted together with the semantic constraints associated to it (e.g., the target of the bombing event must be a physical object). The phrasal pattern represents an *ordered sequence* of lexical entries and/or semantic categories taken from a predefined concept hierarchy.

The FP-structure combines the meaning frame and the phrasal pattern by linking the slots of the former to the elements of the latter. Applying an FP-structure to a sentence represents a straightforward process: if the phrasal pattern matches the sentence, the FP-structure is activated, and then the corresponding meaning frame is used to actually extract the data.

As opposed to AutoSlog, FP-structures can be activated both by exact match and via the *is_a()* relationship within the predefined concept hierarchy. However, it is interesting to note that PALKA is not strictly more expressive then AutoSlog because FP-structures can express exact word constraints only on verbs, while AutoSlog can also do it on nouns.

CRYSTAL

CRYSTAL (Soderland *et al.* 1995) generates multi-slot concept nodes that allow both *semantic* and *exact word* constraints on *any* component phrase. In order to illustrate the expressivity of the CRYSTAL concept nodes, let us consider now the task of extracting both the *target* and the *name* of the perpetrator in the sentence

The Parliament building was bombed by Carlos.

As one can see in Figure 4, the concept definition includes semantic constraints on both the subject and the prepositional phrase, and it also imposes exact word matching for the verb and the preposition. Furthermore, the "Terms include:" construct introduces an additional exact word matching for the subject of the sentence.

CRYSTAL's extraction patterns are more expressive than the PALKA ones because the latter allow exact word matching only on the verb's root. Furthermore, PALKA rules do not allow semantic constraints on prepositional phrases that do not contain relevant information.

FP-structure = MeaningFrame + PhrasalPattern

Meaning Frame: (BOMBING agent: ANIMATE
target: PHYS-OBJ
instrument: PHYS-OBJ
effect: STATE)

Phrasal Pattern: ((PHYS-OBJ) was bombed by (PERP))

FP-structure:
(BOMBING target: PHYS-OBJ
agent: PERP
pattern: ((target) was bombed by (agent))

Figure 3: **Example of FP-structure.**

Concept type: BUILDING BOMBING

SUBJECT:	Classes include:	<PhysicalTarget>
	Terms include:	BUILDING
	Extract:	*target*
VERB:	Root:	BOMB
	Mode:	passive
PREPOS-PHRASE:	Preposition:	BY
	Classes include:	<PersonName>
	Extract:	*perpetrator name*

Figure 4: **A CRYSTAL concept node.**

CRYSTAL + Webfoot

Webfoot (Soderland 1997) is a preprocessing step that enables CRYSTAL to also work on Web pages: the Web pages are passed through Webfoot, which uses page layout cues to parse the pages into logically coherent text segments. Then one can apply CRYSTAL to text segments as if they were sentences coming from a free text document.

In order to break down a document into text segments, Webfoot uses both domain-independent and domain-specific delimiters. The former are usually pairs of HTML tags (e.g., <table> and </table>, or <title> and </title>),

SEGMENTED DOCUMENT:
<segm> field1: <HEAD> LA Forecast </HEAD> </segm>
<segm> field1: .MONDAY... field2: CLOUDY </segm>
<segm> field1: .TUESDAY... field2: SUNNY </segm>

Concept type: FORECAST
Constraints:

FIELD:	Classes include: <Day>
	Terms include: "." , "..."
	Extract: *day*
FIELD:	Classes include: <Weather Condition>
	Extract: *conditions*

Figure 5: **Concept node for day and conditions.**

while the latter are specific to the extraction domain (e.g., "*line beginning with* **.**").

In Figure 5 we show an example segmented Web page together with a concept node that extracts the day and weather condition from a text segment. There are three main differences between the original CRYSTAL concept definitions and the ones used for Web pages. First, the structural components (e.g., SUBJECT or VERB) were replaced by the neutral component FIELD. Second, the "Terms include:" slot may contain ungrammatical, domain-specific tokens like "...". Last but not least, the semantic constraints imposed on fields might represent domain-dependent constructs like "sunny", or "cloudy with occasional light rain".

HASTEN

The extraction patterns generated by HASTEN (Krupka 1995) are called Egraphs, and they can be seen as lists of *(SemanticLabel, StructuralElement)* pairs. Except for the ANCHOR semantic label, which is similar to AutoSlog's conceptual anchor, all other semantic labels represent *potential* items of interest. The *structural element* associated to the semantic label represents a set of syntactic and semantic constraints. For instance, in Figure 6, the semantic label TARGET must be a *noun phrase* that denotes a physical object. In the extraction phase, HASTEN uses a *similarity metric* to compare an Egraph with the input text. In a first step, the system matches the structural elements and binds the semantic labels of the successfully matched structural elements. Then it uses a set of fixed *weight factors* to compute the percentage of the matched Egraph, and it compares the final score with a predefined threshold value.

A few remarks

All the extraction patterns above are used to extract relevant data from grammatical, free text. Even though all of them use syntactic and semantic constraints to identify the items of interest, there are several important differences between them.

First, the granularity of the extraction is not the same: LIEP and HASTEN identify the exact phrase of interest, while AutoSlog, PALKA, and CRYSTAL determine only the syntactic field that contains the target phrase. Second, except

BOMBING:

TARGET:	NP "semantic = physical-object"
ANCHOR:	VG "root = bomb"
PERPETRATOR:	NP "semantic = terrorist-group"

Figure 6: **Example of Egraph used by** HASTEN**.**

for CRYSTAL, all other systems allow semantic class constraints only on the slots to be extracted (for the other sentence elements they allow exact word and verb root constraints). Third, PALKA, CRYSTAL, and HASTEN can generate both single- and multi-slot rules, while AutoSlog learns only single-slot rules, and LIEP can not induce single-slot rules. Last but not least, AutoSlog, PALKA, and CRYSTAL were designed to *always* use the syntactic context; that is, if a relevant item can appear either in the subject or in a prepositional phrase, they must create two distinct extraction patterns. LIEP and HASTEN do not suffer this limitation, and, consequently, they can create a single rule that covers both cases.

IE from online documents

With the expansion of the Web, users can access collections of documents that consist of a mixture of grammatical, telegraphic, and/or ungrammatical text. Performing IE tasks on corpora of job postings, bus schedules, or apartment rentals has immediate practical applications, but the IE techniques for free text are not fit for online documents. In order to handle such documents, the three types of extraction rules presented in this section combine syntactic/semantic constraints with *delimiters* that "bound" the text to be extracted.

WHISK

WHISK (Soderland 1998) is a learning system that generates extraction rules for a wide variety of documents ranging from rigidly formatted to free text. The WHISK extraction patterns are a special type of regular expressions that have two components: one that describes the *context* that makes a phrase relevant, and one that specifies the exact *delimiters* of the phrase to be extracted. Depending of the structure of the text, WHISK generates patterns that rely on either of the components (i.e., context-based patterns for free text, and delimiter-based patterns for structured text) or on both of them (i.e., for documents that lay in between structured and free text).

In Figure 7 we show a sample WHISK extraction task from online texts. The sample document is taken from an apartment rental domain that consists of ungrammatical constructs, which, without being rigidly formatted, obey some structuring rules that make them human understandable. The sample pattern in Figure 7 has the following meaning: ignore all the characters in the text until you find a digit followed by the "br" string; extract that digit and fill the first extraction slot with it (i.e., "Bedrooms"). Then ignore again all the remaining characters until you rich a dollar sign *immediately* followed by a number. Extract the number and fill the "Price" slot with it.

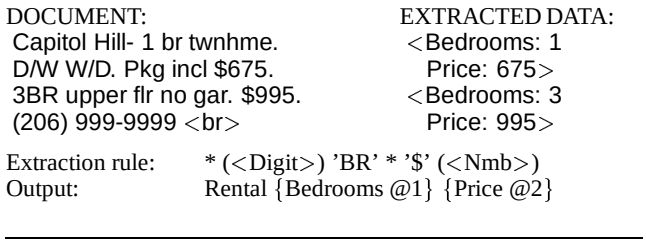

Figure 7: **A WHISK extraction task.**

A more sophisticated version of the pattern could replace the "br" string by the semantic class "`Bedroom`", which is defined as

`Bedroom` ::= (br || brs || bdrm || bedrooms || bedroom)

That is, the semantic class "`Bedroom`" is a placeholder for any of the abbreviations above.

WHISK can also be used on free text domains, where it identifies the exact phrase of interest and allows semantic class constraints on all sentence elements. It can generate both single- and multi-slot patterns, and, similarly to LIEP and HASTEN, it can ignore the syntactic context of the relevant phrase. For grammatical text, the WHISK regular expressions provide a set of special constructs that are helpful when dealing with information provided by syntactic analyzers (e.g., the clausal structure of a sentence). For instance, the pattern * (PObj) * @Passive *F 'bomb' * {'by' *F (Person)} extracts the target and the name of the terrorist involved in a bombing action. The "*F" construct is similar to "*", but it skips characters *only* within the current syntactic field, while the "@Passive" requires the verb "bomb" to be in its passive form.

RAPIER

The RAPIER system (Califf & Mooney 1997) learns single-slot extraction patterns that use limited syntactic information (e.g., the output of a part-of-speech tagger) and semantic class information (e.g., hypernim links from WordNet (Miller 1995)). In Figure 8, we show typical RAPIER extraction task: a sample document, the information to be extracted, and the extraction rule for the `area` slot. The RAPIER extraction pattern consist of three distinct slots: the Pre- and Post- "filler patterns" play the role of left and right delimiters, while the "Filler pattern" describes the structure of the information to be extracted.

Each "Filler pattern" consists of a (possibly empty) list of *pattern items* or *pattern lists*. The former matches exactly one word/symbol from the document, while the latter specifies a maximum length N and matches 0 to N words/symbols from the document. The constraints imposed by the pattern items/lists consists of exact match words, parts of speech, and semantic classes. For instance, in the example above, the pre- and post- filler patterns specify that information to be extracted is immediately preceded by the word "`leading`" and is immediately followed either by

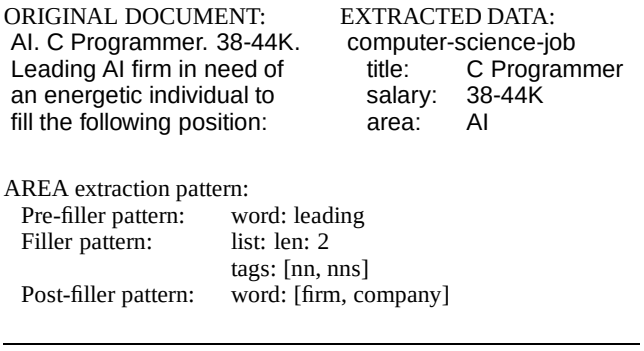

Figure 8: **A RAPIER extraction task.**

DOCUMENT-1: ... to purchase 4.5 mln Trilogy shares at ...
DOCUMENT-2: ... acquire another 2.4 mln Roach shares ...

```
Acquisition:- length( < 2 ),
              some(?A [] capitalized true),
              some(?A [next-token] all-lower-case true),
              some(?A [right-AN] wn-word 'stock').
```

Figure 9: **An SRV extraction task.**

"`firm`" or by "`company`". The "Filler pattern" imposes constraints on the structure of the information to be extracted: it consists of *at most* two words that were labeled "nn" or "nns" by the POS tagger (Brill 1994) (i.e., one or two singular or plural common nouns).

SRV

SRV (Freitag 1998) generates first-order logic extraction patterns that are based on attribute-value tests and the relational structure of the documents. For instance, the pattern from Figure 9 extracts the names of the companies that were the target of an acquisition process. The extraction rule has the following meaning: the company name consists of a single, capitalized word (first two predicates) and is followed by a lower-case word (third predicate).

The most interesting constraints in Figure 9 are imposed by the last predicate, which uses features derived by the link grammar parser (Sleator & Temperley 1993) and WordNet (Miller 1995). The "rightAN" construct refers to the "*right AN link*" in a link grammar, which connects a noun modifier with the noun it modifies. In our example, the information to be extracted (i.e., "`?A`") is connected by "`[right-AN]`" to the word that follows it, which, in turn, is one of the WordNet synset associated with `stock`.

Remarks

The three types of extraction rules presented above differ in several ways. First, RAPIER and SRV can generate only single-slot rules, which is a serious limitation for a significant number of domains (e.g., in a document that contains several names and addresses, single-slot rules can not specify which is the address of a particular person). On the other

D1: 1.Joe's: (313)323-5545 2.Li's: (406)545-2020
D2: 1.KFC: 818-224-4000 2.Rome: (656)987-1212

WIEN rule: * '.' (*) ':' * '(' (*) ')'

SoftMeaky rule: * '.' (*) EITHER ':' (Nmb) '-'
OR ':' * '(' (Nmb) ')'

Output: Restaurant {Name @1} {AreaCode @2}

Figure 10: **WIEN and SoftMealy rules.**

hand, even tough WHISK generates multi-slot rules, it does not have the ability to automatically segment the document such that the rules can be applied only to the relevant pieces of text (e.g., for the rental domain, the document had to be manually broken into individual adds in order to avoid spurious matches in the HTML segments that separate the adds). Second, RAPIER and SRV are capable of imposing a richer set of constraints than WHISK: RAPIER makes use of a part-of-speech tagger, while SRV takes advantage of orthographic features, token's length and link grammars. Furthermore, both systems can impose constraints based on the WordNet semantic classes. Third, if the length of the phrase to be extracted varies dramatically from one document to another, RAPIER and SRV may extract either too many or too few words because both systems are likely to enforce phrase length constraints.

Wrapper Induction Systems

Independently of the traditional IE community, the *wrapper generation* field appeared from the necessity of extracting and integrating data from multiple Web-based sources. A typical wrapper application extracts the data from Web pages that are generated based on predefined HTML templates (e.g., electronic commerce, weather, or restaurant reviews pages). The *wrapper induction* systems generate delimiter-based rules that *do not* use linguistic constraints. In order to facilitate the comparison between the various of extraction patterns, all three types of rules discussed below are described based on a WHISK-like syntax.

WIEN (Kushmerick, Weld, & Doorenbos 1997) was the first wrapper induction system, and it generates extraction rules similar to those of WHISK, except that it uses only delimiters that immediately precede and follow the actual data. WIEN also assumes that there is a unique multi-slot rule that can be used for all documents, and does not allow the use of semantic classes. In Figure 10 we show a sample WIEN rule for extracting the restaurant name and area code from a document. The rule has the following meaning: ignore all characters until you find the first occurrence of '.' and extract the restaurant name as the string that ends at the first ':'. Then, again, ignore all characters until you find '(' and extract the string that ends at ')'. In order to extract the information about the other restaurants within the same page, the whole pattern is applied repeatedly until it fails to match. It is easy to see that the WIEN rule can be successfully applied to document D1, but it fails on D2 because of the different phone number formatting.

SAMPLE DOCUMENT:
Name: Taco Bell
 <p>
- LA: 400 Pico; (213)323-5545,(800) 222-1111.
211 Flower; (213) 424-7645.<p>
- Venice: 20 Vernon; (310) 888-1010.<p><hr>

Embedded Catalog Tree:
Document ::= Restaurant LIST(City)
City ::= CityName LIST(Location)
Location ::= Number Street LIST(Phone)
Phone ::= AreaCode PhoneNumber

`Restaurant` extraction rule: * 'Name :' (*) '
'
`LIST(City)` extraction rule: * '
' * '
' (*) '<hr>'
`LIST(City)` iteration rule: * '-' (*) '<p>'
`CityName` extraction rule: * (*) ':'

Figure 11: **A STALKER extraction domain.**

The WIEN rule above is an instance of `LR` class, which the simplest type of WIEN rules. The classes `HLRT`, `OCLR`, and `HOCLRT` are extensions of `LR` that use document *head* and *tail* delimiters, tuple delimiters, and both of them, respectively. WIEN defines two other classes, `N-LR` and `N-HLRT`, but their induction turned out to be impractical.

SoftMealy (Hsu & Dung 1998) is a wrapper induction algorithm that generates extraction rules expressed as finite-state transducers. It allows both the use of semantic classes and disjunctions, which are especially useful when the documents contain several formatting conventions or various orderings of the items of interest. Figure 10 also shows a SoftMealy extraction rule[1] that can deal with the different formatting of the area code. The SoftMealy rule reads as follows: ignore all tokens until you find a '.'; then extract the restaurant name, which is the string that ends before the first ':'. If ':' is immediately followed by a number, extract it as the area code; otherwise ignore all characters until you find a '(' immediately followed by a number, which represents the area code. SoftMealy's extraction patterns are obviously more expressive than the WIEN ones; their main limitation consists of their inability to use delimiters that do not immediately precede and follow the relevant items.

STALKER (Muslea, Minton, & Knoblock 1999) is a wrapper induction system that performs hierarchical information extraction. In Figure 11, we have a sample document that refers to a restaurant-chain that has restaurants located in several cities. In each city, the restaurant may have several addresses, and at each address it may have several phone numbers. It is easy to see that the multi-slot output schema is not appropriate for extraction tasks that are performed on such documents with multiple levels of embedded data. In order to cope with this problem, STALKER introduces the

[1] In fact, the sample SoftMealy rule represents the expansion of all possible paths through the transducer.

Embedded Catalog Tree (ECT) formalism to describe the hierarchical organization of the documents. The ECT specifies the output schema for the extraction task, and it is also used to guide the hierarchical information extraction process.

For a given ECT, STALKER generates one extraction rule for each node in the tree, together with an additional iteration rule for each `LIST` node. The extraction process is performed in a hierarchical manner. For instance, in order to extract all `CityNames` in a document, STALKER begins by applying to the whole document the extraction rule for the `LIST(City)`, which skips to the second
 in the page and extracts everything until it encounters a <hr>; then in order to extract each individual `City`, it applies the `LIST(City)` iteration rule to the *content of the list*. Finally, STALKER applies to the content of each extracted `City` the `CityName` extraction rule.

There are two main differences between the rules generated by STALKER and WHISK. First, even though STALKER uses semantic constraints, it does not enforce any linguistic constraints. Second, the STALKER rules are single-slot. However, unlike RAPIER and SRV, the single-slot nature of the STALKER rules does not represent a limitation because STALKER uses the ECT to group together the individual items that were extracted from the same multi-slot template (i.e., from the same ECT parent). Using single-slot rules in conjunction with the ECT has two major advantages. First, to our knowledge, STALKER is the only IE inductive system that can extract data from documents that contain arbitrarily complex combinations of embedded lists and items.[2] Second, as each item is extracted independently of its siblings in the ECT, the various orderings of the items does not require one rule for each existing permutation of the items to be extracted.

Conclusions

With the growth of the amount of online information, the availability of robust, flexible IE systems will become a stringent necessity. Depending on the characteristics of their application domains, today's IE systems use extraction patterns based on one of the following approaches: syntactic/semantic constraints, delimiter-based, or a combination of both. WHISK is the only system that is currently capable of generating multi-slot rules for the whole spectrum of document types. On the other hand, by using RAPIER- or SRV-like rules in conjunction with the Embedded Catalog Tree, one could obtain rules that are more expressive than the ones of WHISK.

References

Brill, E. 1994. Some advances in rule-based part of speech tagging. *Proceedings of the 12th Annual Conference on Artificial Intelligence (AAAI-94)* 722–727.

Califf, M., and Mooney, R. 1997. Relational learning of pattern-match rules for information extraction. *Working Papers of the ACL-97 Workshop in Natural Language Learning* 9–15.

Freitag, D. 1998. Information extraction from html: Application of a general learning approach. *Proceedings of the 15th Conference on Artificial Intelligence (AAAI-98)* 517–523.

Hsu, C., and Dung, M. 1998. Generating finite-state transducers for semi-structured data extraction from the web. *J. of Information Systems* 23(8):521–538.

Huffman, S. 1995. Learning information extraction patterns from examples. *IJCAI-95 Workshop on new approaches to learning for natural language processing* 127–142.

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

Krupka, G. 1995. Description of the sra system as used for muc-6. *Proceedings of the Sixth Message Understanding Conference (MUC-6)* 221–235.

Kushmerick, N.; Weld, D.; and Doorenbos, R. 1997. Wrapper induction for information extraction. *Proceedings of the 15th International Conference on Artificial Intelligence (IJCAI-97)* 729–735.

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

Muslea, I.; Minton, S.; and Knoblock, C. 1999. A hierarchical approach to wrapper induction. *Proceedings of the Third International Conference on Autonomous Agents (AA-99)*.

Riloff, E. 1993. Automatically constructing a dictionary for information extraction tasks. *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-93)* 811–816.

Sleator, D., and Temperley, D. 1993. Parsing english with a link grammar. *Third International Workshop on Parsing Technologies*.

Soderland, S.; Fisher, D.; Aseltine, J.; and Lehnert, W. 1995. Crystal: Inducing a conceptual dictionary. *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)* 1314–1319.

Soderland, S. 1997. Learning to extract text-based information from the world wide web. *Proceedings of Third International Conference on Knowledge Discovery and Data Mining (KDD-97)*.

Soderland, S. 1998. Learning information extraction rules for semi-structured and free text. *To appear in the Journal of Machine Learning*.

[2] As SoftMealy induces the topology of the transducers, it may generate extraction rules that are ambiguous with respect to the grouping of the individual items.