Transforming Examples into Patterns for Information Extraction

Roman Yangarber and Ralph Grishman
Dept. of Computer Science
New York University
715 Broadway, 7th Floor
New York, NY 10003, USA
`roman, grishman@cs.nyu.edu`

Abstract

Information Extraction (IE) systems today are commonly based on pattern matching. The patterns are regular expressions stored in a customizable knowledge base. Adapting an IE system to a new subject domain entails the construction of a new pattern base — a time-consuming and expensive task. We describe a strategy for building patterns *from examples.* To adapt the IE system to a new domain quickly, the user chooses a set of examples in a training text, and for each example gives the logical form entries which the example induces. The system transforms these examples into patterns and then applies *meta-rules* to generalize these patterns.

1 Introduction

The task of Information Extraction (IE) as understood in this paper is the selective extraction of meaning from free natural language text.[1] This kind of text analysis is distinguished from others in Natural Language Processing in that "meaning" is understood in a narrow sense - in terms of a fixed set of semantic objects, namely, *entities*, *relationships* among these entities, and *events* in which these entities participate. These objects belong to a small number of *types*, all having fixed regular structure, within a fixed and closely circumscribed subject domain, which permits the objects to be stored in a database (e.g., a relational data base). These characteristics make the IE task both simpler and more tractable than the more ambitious problem of general text understanding. They allow us to define the notion of a "correct answer", and, therefore, to conduct quantitative *evaluation* of performance of an IE system. A series of formal evaluations — the Message Understanding Conferences (MUCs),[2] conducted over the last decade — are described in [8, 6].

The MUCs have yielded some widely (if not universally) accepted wisdom regarding IE:

- *Customization* and portability is an important problem: to be considered a useful tool, an IE system must be able to perform in a variety of domains.
- Systems have *modular* design: Control is encapsulated in immutable core engines, which draw upon domain- or scenario-specific information stored in knowledge bases (KB) which are customized for each new domain and scenario.
- Text analysis is based on *pattern matching*: regular expression pattern matching is a widely used strategy in the IE community. Pattern matching is a form of deterministic bottom-up partial parsing. This approach has gained considerable popularity due to limitations on the accuracy of full syntactic parsers, and the adequacy of partial, semantically-constrained, parsing for this task [2, 1, 5].
- Building effective patterns for a new domain is the most complex and time-consuming part of the customization process; it is highly error-prone, and requires detailed knowledge of system internals.

In view of these findings, it becomes evident that having a disciplined method of customizing the pattern

[1]For a review of a range of extraction systems, the reader is referred to [9].

[2]In this paper we will use IE terminology accepted in the MUC literature. A subject *domain* will denote the class of textual documents to be processed, such as "business news," while *scenario* will refer to the set of facts to be extracted, i.e., the specific extraction task that is applied to documents within the domain. One example of a scenario is "executive succession", the task tested in MUC-6, where the system seeks to identify events in which corporate managers left their posts or assumed new ones.

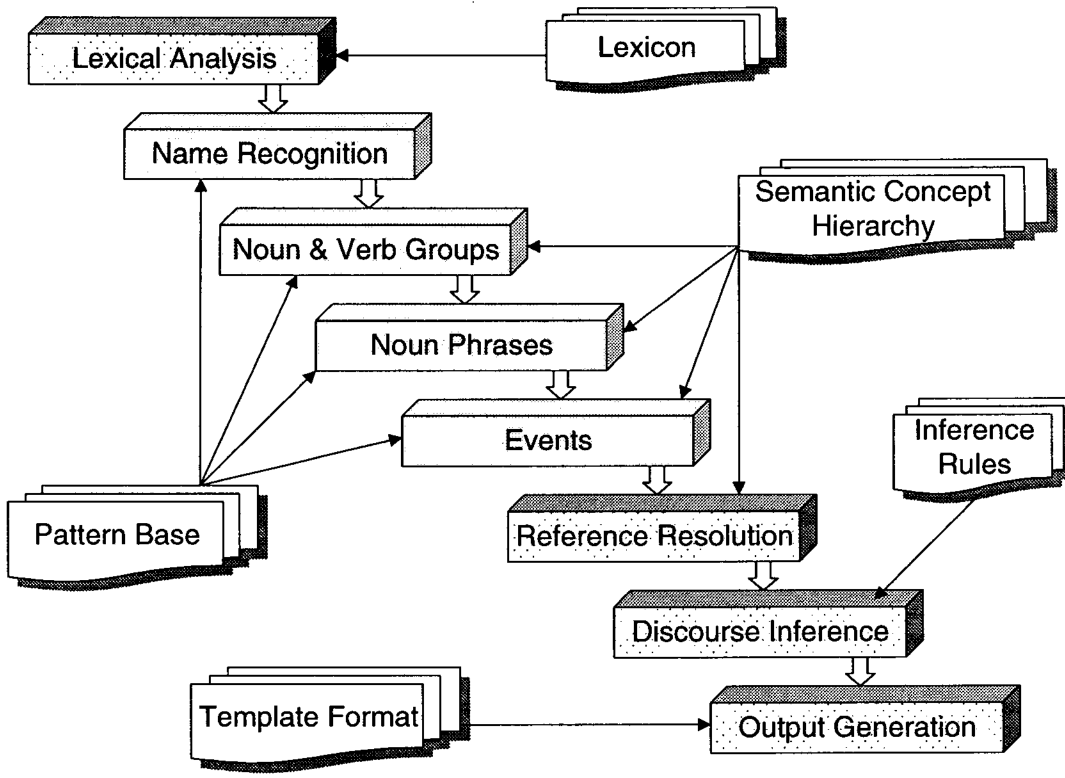

Figure 1: IE system architecture

base is essential. It is particularly valuable if the method allows naive users (i.e., non-developers, unfamiliar with system internals) to customize the system. The current work describes such a method.

2 Structure of the IE System

An outline of our IE system [5, 11, 12] is shown in figure 1. The system is a pipeline of modules: each module draws on attendant KBs to process its input, and passes its output to the next module.

The *lexical analysis* module is responsible for breaking up the document into sentences, and sentences into tokens. This module draws on a set of on-line dictionaries. Lexical analysis attaches to each token a *reading*, or a list of alternative *readings*, in case the token is syntactically ambiguous. A reading contains a list of features and their values (e.g., "syntactic category = *Noun*"). The lexical analyzer incorporates a statistical part-of-speech tagger, which eliminates unlikely readings for each token.[3]

The *name recognition* module identifies proper names in the text by using local textual cues, such as capitalization, personal titles ("Mr.", "Esq."), and company suffixes ("Inc.", "Co.").[4] The next module identifies small syntactic units, such as basic noun groups (nouns with their left modifiers) and verb groups. When it identifies a phrase, the system marks the text segment with a semantic description, which includes the semantic class of the head of the phrase.[5] The next module (incrementally building on information gathered by the earlier modules) finds larger noun phrases, involving (for example) conjunction, apposition, and prepositional phrase modifiers, using local semantic information. The following module identifies scenario-specific clauses and nominalizations.

These four modules operate by matching *patterns* of successively increasing complexity against the input. The patterns are *regular expressions* which trigger associated actions. The actions perform operations on the *logical form* representation (LF) of the processed segments of the discourse. The discourse is

[3]We wish to thank BBN for providing us with their tagger.

[4]Name recognition is a well-researched topic, with the best available systems today reaching 96% accuracy in narrow domains.

[5]These marks are pointers to the corresponding entities which are created and added to the logical form.

Slot	Value
class	C-Company
name	Coca-Cola, Inc.
location	...
...	...

Figure 2: LF for the text: "Coca-Cola, Inc."

thus represented as a sequence of LFs corresponding to the entities, relationships, and events encountered so far in the analysis.

A LF is an object with named slots (see example in figure 2). One slot in each LF, named "class", has distinguished status, and determines the number and type of other slots that the object may contain. E.g., an entity of class "company" has a slot called "name". It also contains a slot "location" which points to another entity, thereby establishing a relation between the location entity and the matrix entity. Events are specific kinds of relations, usually having several operands (example in figure 3).

The subsequent modules operate on the logical forms built by the pattern matching modules. *Reference resolution* merges co-referring expressions, e.g., it links anaphoric pronouns to their antecedents. *Discourse inference* uses inference rules to build more complex event structures, where the information needed to extract a single complex fact is spread across several clauses. Lastly, the output-generation phase formats the resultant LF into the output structure specified by the user, e.g., into a database table.

3 General and Specific Patterns

Before we describe our example-based strategy for building patterns, we examine the organization of the pattern base in more detail. We can group the patterns into "layers" according to their range of applicability:

1. *Domain-independent*: this layer contains the most generally applicable patterns. Included in this layer are many of the patterns for name recognition (for people, organizations, and locations, as well as temporal and numeric expressions, currencies, etc.), and the purely syntactic patterns for noun groups and verb groups. These patterns are useful in a wide range of tasks.

2. *Domain-specific*: the next layer contains domain-specific patterns, which are useful across a narrower range of scenarios, but still have considerable generality. These include domain-specific name patterns, such as those for certain types of artifacts, as well as patterns for noun phrases which express ***relationships*** among entities, such as those between persons and organizations.

3. *Scenario-specific*: the last layer contains scenario-specific patterns, having the narrowest applicability, such as the clausal patterns that capture relevant events.

This stratification reflects the relative "persistence" of the patterns. The patterns at the lowest level, having the widest applicability, are built in as a core component of the system. These change little when the system is ported to a new domain. The mid-range patterns, applicable in certain commonly encountered domains, can be organized as domain-specific ***pattern libraries***, which can be plugged in as required by the extraction task.[6] For example, for the "business/economic news" domain, we have patterns that capture:

- entities – organization, company, person, location;

- relations – person/organization, organization/location, parent/subsidiary organization.

The scenario-specific patterns must be built on a per-scenario basis. This is accomplished through a set of graphical tools, which engage the user only at the level of surface representations, hiding the internal operation of the patterns. The user's input is reduced to

- providing textual *examples* of events of interest,

- describing the corresponding *output structures* (LFs) which the example text should induce.

In the remaining sections we discuss how the system can use this information to

- automatically build patterns to map the user-specified text into the user-specified LF,

- *generalize* the newly created patterns to boost coverage.

[6]To a limited degree, the system is able to adapt to a new domain automatically: given training data in the domain, we can train a statistical proper name recognizer [3], in effect, obviating the need for building domain-specific name patterns.

...Information Resources Inc.'s London-based European Information Services operation has appointed George Garrick, 40 years old, president ...

Field	Value
Position	president
Company	European Information Services
Location	London
Person	George Garrick
Status	In

Figure 3: Succession text and extracted record

```
;;; For <company> appoints <person> <position>
(definePattern Appoint
   "np(C-company)?  vg(C-appoint) np(C-person)
          to-be?  np(C-position):
    company=1.attributes, person=3.attributes,
          position=5.attributes |
...
(defineAction Appoint (phrase-type)
    (let ((person-at (binding 'person))
        (company-entity (entity-bound 'company))
        (person-entity (entity-bound 'person))
        (position-entity (entity-bound 'position))
        new-event)
     ;; if no company slot in position, use agent
...
```

Figure 4: A manually coded scenario pattern

4 Example-based Acquisition

4.1 Objective

Consider a situation where the developer has found a salient text segment and proceeds to extend the IE system to extract the proper information from it. Figure 3 shows a (paraphrased) text segment from the MUC-6 development corpus, with the corresponding extracted event, in the form of a database record. We will use this example to illustrate our methodology. In our earlier system (as in most other IE systems), upon finding a candidate example, the developer had to construct a pattern capable of capturing the example. Such a pattern consists of two parts:

- the *precondition*, which seeks to match an active clause beginning with a *np* of type "company", followed by a verb group (*vg*) of class "appoint", followed by a *np* of class "person", etc.;

- the *action* which fires when the pattern matches, and prescribes the operations to be performed on the sentence fragments and the logical form.

Figure 4 shows an excerpt from the pattern code; it is written in Common Lisp, with the precondition specified using a special "pattern language". Clearly, this method of development is quite time-consuming and error-prone.

Instead, we now employ a kind of a "bootstrapping" procedure: the system allows the user to introduce a new example and apply to it the patterns that the system has acquired *so far*. This produces a partial analysis, and builds LFs for the analyzable constituents of the example text. The user then specifies how these LFs, or their sub-parts, combine to yield the LF for the entire example.

4.2 Acquiring Preconditions

To illustrate this method, we first show how the system acquires the pattern for analyzing the portion of our example shown in figure 6. This is a complex NP made up of a series of nouns, names, and other lexical items. The crucial point is that a basic system, which has not been specialized for any domain, will analyze this reduced example as in figure 5 by dint of its built-in patterns for named entities.[7] The analysis also produces the LF entities for each boxed segment. This information is sufficient for the system to *propose* a precondition for a candidate pattern:[8]

n(C-company) **'s** n(C-city) **-based** n(C-company) n(operation)

The system then initiates an interaction with the user in which s/he can operate on the components, modifying them or labeling them as optional (indicated below by a following question mark), to complete the precondition of the pattern:

[n(C-company) 's]? [n(C-city) **-based**]? n(C-company) n(operation)?

4.3 Acquiring Actions

Now the user specifies, again *by example*, what action is to be performed when the precondition of the pattern matches. S/he can select the new type of event

[7]The customization tool supports two methods for analyzing specific noun group structures. The approach described here involves the creation of a semantically-specific noun group pattern. Alternatively, the phrase can first be analyzed by the general, syntactic noun group pattern, with the resulting LF then restructured by a semantically-specific pattern.

[8]For purposes of presentation, we have simplified the form of these patterns to emphasize the parallel with the clause patterns. In the current implementation, each pattern element would involve a conjunction of tests for the syntactic type (*n*) and the semantic class (*C-company*, etc.).

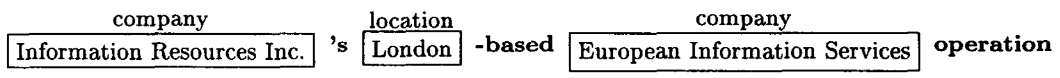

Figure 5: Initial analysis

...Information Resources Inc.'s London-based European Information Services operation ...

Slot	Value
class	C-Company
name	European Information Services
location	entity $\Longrightarrow$ *<London>*
parent	entity $\Longrightarrow$ *<I.R.Inc.>*

Figure 6: A complex NP and corresponding entity LF

or entity to be created, and indicate how the matched constituents (LFs) discovered in the example are to function in the new event. Alternatively, s/he may designate one of the generated entities as the "head" entity (or the matrix entity) for the complex phrase, and designate the remaining entities as *subordinate* to the matrix entity, i.e., as standing in some semantic relation to it. To accomplish this, the user can drag-and-drop a subordinate entity into the appropriate slot in the matrix entity (in a simple GUI environment); the slot serves to indicate the relationship of the subordinate entity to the matrix; (see figure 6). The precondition and the action together now constitute a complete pattern which matches a complex NP and produces a LF with relations.

4.4 Semantic Generalization

Consider the final, optional constituent in the precondition of the preceding pattern, *n(operation)*. We would like to broaden the coverage of the pattern, so that it could match any semantically similar noun in that position; in particular, it should also match "concern", "outfit", etc. To this end, our system allows the user to gather semantic concepts in an inheritance hierarchy. For example, s/he can gather all these and more lexemes under the same semantic class, called, e.g., *C-co-descrip*. Similarly, the classes *C-city* for city names and *C-state* for state names would be gathered under a concept *C-location*. The GUI tools then allow the user to perform *semantic generalization* on the individual constituents of the pattern's precondition; its final form becomes:

Slot	Value
class	Predicate-Start-Job
company	entity $\Longrightarrow$ *<E.I.S.>*
person	entity $\Longrightarrow$ *<Garrick>*
position	entity $\Longrightarrow$ *<president>*

Figure 7: Event LF corresponding to a clause

> [n(C-company) 's]? [n(C-location) **-based**]? n(C-company) n(C-co-descrip)?

The semantic hierarchy is scenario-specific. It is built up dynamically through tools that draw on pre-existing domain-independent hierarchies, such as WordNet, as well as domain-specific word similarity measures and co-occurrence statistics [4].

By a similar process, we can now acquire a *clausal* pattern from the example in figure 3 at the beginning of this section. The system proposes the precondition:

> np(C-company) vg(C-appoint) np(C-person) np(president)

Applying semantic generalization to the last constituent yields:

> np(C-company) vg(C-appoint) np(C-person) np(C-title)

where *C-title* is a semantic class that gathers all corporate titles. The user can now fill the slots in the LF for the event as in figure 7.

5 Meta-rules

Consider the following variant of the original example:

> ... George Garrick, *an avowed anti-capitalist,* was appointed *yesterday* president of Information Resources Inc., ...

The basic pattern for an active clause, which we acquired in the preceding section, will not match this paraphrase. There are two essential kinds of variations here:

- *syntactic transformations*; the system needs several related patterns, which capture the corresponding passive clause, relative clause, and other syntactic variants of the example.
- optional, semantically irrelevant *modifiers*, e.g., sentence adjuncts, appositions, etc., as exemplified by the italicized segments above.

The user could, of course, provide *transformed* examples, build patterns individually for each transformation of the original, and insert the optional modifiers to make the patterns as general as possible. However, it is clear that this naive approach quickly leads to a proliferation of patterns with which the user is directly concerned. Instead, we have introduced a *meta-rule* mechanism: after a pattern is accepted, the system generates all related generalizations of the pattern automatically.[9] For example, from the active clause pattern above, a passivizing meta-rule will produce the precondition:

np(C-person) rn? sa? pass-vg(C-appoint) sa?
np(C-title) [by np(C-company)]? [10]

The resulting pattern will match the variant example, and produce the correct event LF. To maximize coverage, the system should contain meta-rules for all clausal variants, including nominalizations; similar meta-rules can be provided to generalize noun-phrase patterns, as discussed in section 4.2.

6 Discussion

We have described a comprehensive methodology for acquiring patterns from examples and automatically expanding their coverage. Other IE systems employ variants of example-based pattern acquisition. One system, developed at University of Massachusetts at Amherst, [10], used unsupervised training to *learn* patterns from the MUC training corpus. However, unsupervised learning can degrade in the face of sparse data; the UMass system seemed to require one more order of magnitude of training data than was available in MUC-6. The HASTEN system, developed by SRA [7], used a somewhat different example-based approach: they seek to broaden coverage by allowing statistically *approximate* matches, a strategy that lacks a syntactic basis, and may result in overgeneration.

[9] A meta-rule mechanism is also included in the SRI FASTUS system[2].

[10] where *rn* is a pre-defined sub-pattern that matches various right noun-phrase modifiers, *sa* is a sentence adjunct, and *pass-vg* is a passive verb group.

The presented methodology has been fully implemented as a set of tools that complement our core information extraction engine, and has been tested on three different scenarios. One of the scenarios was successfully implemented by a computational linguist who interacted with the system exclusively by means of the tools, and had no familiarity with the system internals.

Our experience also suggests areas of improvement, which we are currently pursuing. One important question is: where do examples come from? We seek to shift the burden of inventing the examples from the developer to the system. In response to these problems we are building tools that will help the user surf the corpus to help discover patterns.

7 Conclusion

Porting an existing system to a new domain presents an important problem in IE. Effective techniques are needed to minimize the time and complexity of the process, and to extricate the porting process from low-level system details, so that it can be undertaken by non-expert users. In this report, we have described our approach to the problem, based on:

- example-based acquisition of scenario-specific patterns,
- system-aided generalization of acquired patterns, at the semantic and syntactic level.

The experience we have gained from implementing this strategy leads us to believe in its overall usefulness.

References

[1] Douglas Appelt, Jerry Hobbs, John Bear, David Israel, Megumi Kameyama, Andy Kehler, David Martin, Karen Meyers, and Mabry Tyson. SRI International FASTUS system: MUC-6 test results and analysis. In *Proc. Sixth Message Understanding Conf. (MUC-6)*, Columbia, MD, November 1995. Morgan Kaufmann.

[2] Douglas Appelt, Jerry Hobbs, John Bear, David Israel, and Mabry Tyson. FASTUS: A finite-state processor for information extraction from real-world text. In *Proc. 13th Int'l Joint Conf. Artificial Intelligence (IJCAI-93)*, pages 1172–1178, August 1993.

[3] Andrew Borthwick, John Sterling, Eugene Agichtein, and Ralph Grishman. Exploiting diverse knowledge sources via maximum entropy in named entity recognition. In *Proceedings of the Sixth Workshop on Very Large Corpora*, Montreal, Canada, August 1998.

[4] Ido Dagan, Shaul Marcus, and Shaul Markovitch. Contextual word similarity and estimation from sparse data. In *Proceedings of the 31st Annual Meeting of the Assn. for Computational Linguistics*, pages 31–37, Columbus, OH, June 1993.

[5] Ralph Grishman. The NYU system for MUC-6, or where's the syntax. In *Proc. Sixth Message Understanding Conf.*, pages 167–176, Columbia, MD, November 1995. Morgan Kaufmann.

[6] Ralph Grishman and Beth Sundheim. Message understanding conference - 6: A brief history. In *Proc. 16th Int'l Conf. on Computational Linguistics (COLING 96)*, Copenhagen, August 1996.

[7] George Krupka. SRA: Description of the SRA system as used for MUC-6. In *Proc. Sixth Message Understanding Conf. (MUC-6)*, Columbia, MD, November 1995. Morgan Kaufmann.

[8] *Proceedings of the Sixth Message Understanding Conference (MUC-6)*, Columbia, MD, November 1995. Morgan Kaufmann.

[9] Maria Teresa Pazienza, editor. *Information Extraction.* Springer-Verlag, Lecture Notes in Artificial Intelligence, Rome, 1997.

[10] W. Soderland, D. Fisher, J. Aseltine, and W. Lenhert. CRYSTAL: Inducing a conceptual dictionary. In *Proc. Int'l Joint Conf. Artificial Intelligence (IJCAI-95)*, pages 1314–1319, Montreal, Canada, 1995.

[11] Roman Yangarber and Ralph Grishman. Customization of information extraction systems. In Paola Velardi, editor, *International Workshop on Lexically Driven Information Extraction*, pages 1–11, Frascati, Italy, July 1997. Università di Roma.

[12] Roman Yangarber and Ralph Grishman. NYU: Description of the Proteus/PET system as used for MUC-7 ST. In *MUC-7: Seventh Message Understanding Conference*, Columbia, MD, April 1998. Avaliable through the SAIC MUC web site, `http://www.muc.saic.com/`.