

Introduction to Information Extraction Technology

A Tutorial Prepared for IJCAI-99

by

Douglas E. Appelt

and

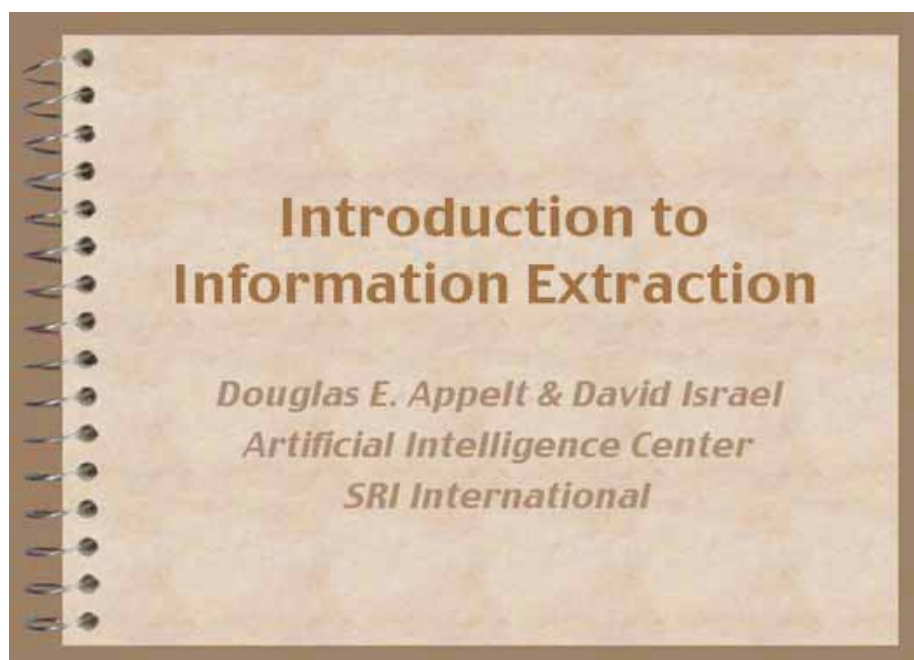
David J. Israel

Artificial Intelligence Center

SRI International

333 Ravenswood Ave.

Menlo Park, CA



We have prepared a set of notes incorporating the visual aids used during the Information Extraction Tutorial for the IJCAI-99 tutorial series. This document also contains additional information, such as the URLs of sites on the World Wide Web containing additional information likely to be of interest. If you are reading this document using an appropriately configured Acrobat Reader (available free from Adobe at <http://www.adobe.com/prodindex/acrobat/readstep.html>) is appropri-

ately configured, you can go directly to these URLs in your web browser by clicking them.

This tutorial is designed to introduce you to the fundamental concepts of information extraction (IE) technology, and to give you an idea of what the state of the art performance in extraction technology is, what is involved in building IE systems, and various approaches taken to their design and implementation, and the kinds of resources and tools that are available to assist in constructing information extraction systems, including linguistic resources such as lexicons and name lists, as well as tools for annotating training data for automatically trained systems.

Most IE systems process texts in sequential steps (or “phases”) ranging from lexical and morphological processing, recognition and typing of proper names, parsing of larger syntactic constituents, resolution of anaphora and coreference, and the ultimate extraction of domain-relevant events and relationships from the text. We discuss each of these system components and various approaches to their design.

An Important URL To Remember

<http://www.ai.sri.com/~appelt/ie-tutorial/>

- * Here you will find
 - A downloadable .pdf file of the tutorial notes
 - Links to information extraction research
 - Links to system building resources
 - Other useful links

Another Interesting Site

<http://www.ai.sri.com/~appelt/TextPro/>

- * Downloadable IE System for Power-PC Macintoshes
- * Uses TIPSTER Technology
 - TIPSTER Architecture
 - Common Pattern Specification Language
- * It's free!
- * Includes a name recognizer grammar for Wall Street Journal Texts

In addition to these tutorial notes, the authors have prepared several other resources related to information extraction of which you may wish to avail yourself. We have created a web page for this tutorial at the URL mentioned in the Power Point slide in the next illustration. This page provides many links of interest to anyone wanting more information about the field of information extraction, including pointers to research sites, commercial sites, and system development tools.

We felt that providing this resource would be appreciated by those taking the tutorial, however, we subject ourselves to the risk that some interesting and relevant information has been inadvertently omitted during our preparations. Please do not interpret the presence or absence of a link to any system or research paper to be a positive or negative evaluation of the system or any of its underlying research.

Also, those who have access to a Power-PC based Macintosh system may be interested in the web site devoted to the TextPro system, written by Doug Appelt. TextPro is a simple information extraction system that you can download and experiment with on your own. The system incorporates technology developed under the recently concluded DARPA-sponsored TIPSTER program. The TIPSTER program provided much of the support for the recent development of IE technology, as well as the development of techniques for information retrieval over very large data collections. In addition to the basic system, the TextPro package includes a complete name recognizer for English Wall Street Journal texts, and a finite-state grammar for English noun groups and verb groups. You can learn a lot about building rule-based information extraction systems by examining these rules written in the Common Pattern Specification Language developed under the TIPSTER program, even if you do not have appropriate hardware to actually run the system.

Introduction

Look at your directory; it is full of files. With a few exceptions, with extensions like “.gif” and “.jpeg,” they are text files of one kind or another. A text file is simply a data structure consisting of alphanumeric and special characters. Your operating system, whatever it is, will have a host of built-in commands for handling such files. None of these commands need be especially smart. In particular, none of them need know anything about the lexical, syntactic or semantic-pragmatic structure of the language, if any, the files are in. Consider a “word count” program. It simply counts the number of sequences of characters that are separated by some conventional delimiter such as a SPACE. Or Consider UNIX grep: it searches a file for a string or regular expression matching an input string or pattern. The query string need not be a meaningful expression of any language; the same, of course, for

Varieties of Text Processing

- * Grep, word count: files as sequences of (ASCII) characters
- * Information/Document Retrieval: files as sequences of perhaps meaningful units-- words
- * Information Extraction: files as containing meaningful phrases/clauses, relevant to a particular topic
- * Text Understanding: files as articles, essays, stories, novels,...

Information Extraction: The Nature of the Task

- * Delimited criteria of relevance/topics are specified in advance
- * Fixed and limited representational format
- * Clear criteria of success are at least possible
- * Corollary features:
 - Typically only parts of the text are relevant
 - Often only part of a relevant sentence is really relevant
 - Can be targeted at large corpora

Text Understanding: The Nature of the Task

- * No predetermined specification of, or limit to the semantic and communicative aspects of interest
- * Representation of meaning must be rich and flexible enough to capture all the meaning (?) of the text.
- * No clearly defined criteria of success
- * Corollary features:
 - Every bit of the text is relevant
 - Can't (yet) be applied to large bodies of text

any matches. If the query string is a meaningful expression of, say, English, that fact is quite irrelevant to grep.

We speak of text processing only when the programs in question reflect, in some way or other, meaningful aspects of the language in which the texts are written. (We should note that Information Retrieval, Information Extraction, Natural Language Understanding can all be applied to spoken language, that is to nontext, audio files; in this tutorial, however, we shall focus solely on Text Processing applications.) The simplest IR programs simply perform a variant of grep. More advanced IR methods, on the other hand, take into account various aspects of morphology, phrasal structure, etc. Thus, with respect to morphology, a query searching for documents about “information extraction” might match on texts containing various form of the root verbs “inform” and “extract.” In the IR world, this is known as stemming. (For more on IR, visit <http://www.cs.jhu.edu/~weiss/ir.html>)

One cannot draw a clear boundary separating Information Retrieval from Information Extraction in terms of the complexity of the language features embodied in the program. The same is true on the

Targets of opportunity for IE

- * Texts whose function is the communication of factual information
- * Intended audience for texts is broad
- * No requirement for modeling special character of either author or audience
- * Prime example: News stories

MUC: Message Understanding Conference

- * Corpus of training texts
- * Specification of the IE task
- * Specification of the form of the required output
- * Keys: ground truth-human produced responses in output format
- * Evaluation procedure:
 - blind test test
 - system performance automatically scored against keys

other side of this spectrum, dividing Information Extraction from (full) Text Understanding. The best way to characterize the different methodologies is in terms of functionality—at least functionality aimed at, if not achieved—and task requirement. The task of IR is to search and retrieve documents in response to queries for information. The fewer irrelevant documents retrieved, other things being equal the better; the fewer relevant texts missed, the better. The functionality of Text Understanding systems cannot be so easily characterized; nor, correlatively, can the task requirements or criteria of success. What counts as (successfully) understanding a text?

The case of Information Extraction is more like that of IR, at least with respect to having, or at least allowing, fairly determinate task specifications and criteria of success. Though we should note that, as with IR, interpersonal agreement on what counts as success is not necessarily easy to come by. The specification can be presented in two different forms: (i) something very like an IR query—a short description of the kind of information being sought, or (ii) a database schema or template, specifying the output format. Here's an example of a query or narrative, taken from MUC-7:

“A relevant article refers to a vehicle launch that is scheduled, in progress or has actually occurred and must minimally identify the payload, the date of the launch, whether the launch is civilian or military, the function of the mission and its status.”

MUC Tasks

- * MUC-1 ('87) and MUC-2 ('89)
 - Messages about naval operations
- * MUC-3 ('91) and MUC-4 ('92)
 - News articles about terrorist activity
- * MUC-5 ('93)
 - News articles about joint ventures and microelectronics
- * MUC-6 ('95)
 - News articles about management changes
- * MUC-7 ('97)
 - News articles about space vehicle and missile launches

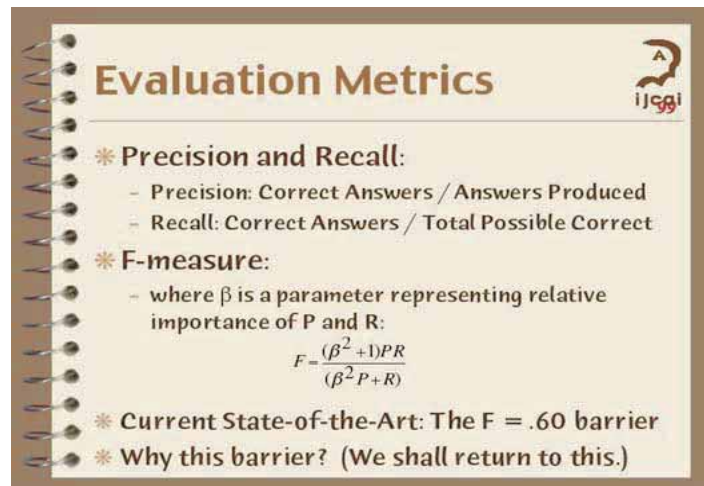
The IE Task: A Sample

- * MUC7 task:
 - Launch Event:
 - Vehicle:
 - Payload:
 - Mission_Date:
 - Mission_Site:
 - Mission_Type: {Military, Civilian}
 - Mission_Function: {Test, Deploy, Retrieve}
 - Mission_Status: {Succeeded, Failed, In_Progress, Scheduled}

What is (a) MUC? A MUC is either a Message Understanding Conference or a Message Understanding Competition. MUCs were instituted by DARPA in the late '80s in response to the opportuni-

ties presented by the enormous quantities of on-line texts. In a very real sense, DARPA created the field of Information Extraction, in part by focusing in on a certain kind of task.

As sketched above, IE is not a stand-alone task that human analysts typically engage in. It is an abstraction from such tasks (note, for instance, that the task is usually specified in a way that either precludes or discourages rich domain-specific inference) -- an abstraction intended to be achievable without human intervention.



Evaluation Metrics

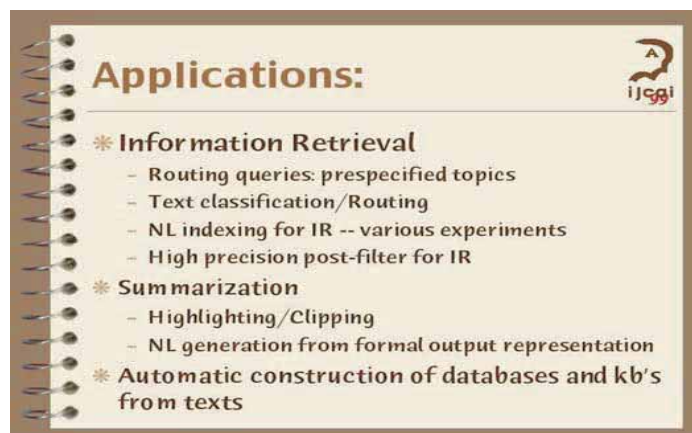
- * **Precision and Recall:**
 - Precision: Correct Answers / Answers Produced
 - Recall: Correct Answers / Total Possible Correct
- * **F-measure:**
 - where β is a parameter representing relative importance of P and R:
$$F = \frac{(\beta^2 + 1)PR}{(\beta^2 P + R)}$$
- * **Current State-of-the-Art: The F = .60 barrier**
- * **Why this barrier? (We shall return to this.)**

The experience of the MUCs has demonstrated that IE is a difficult task. We noted that, unlike Text Understanding, IE allows for fairly precise metrics of success. The field adopted and adapted a version of standard signal processing measures, keyed to counts of true and false positives and true and false negatives. The names of the two basic measures, however, have been changed to Precision and Recall. Interestingly enough, one of the first things to be discovered was how difficult the task is for human beings, even for trained analysts. The natural way to measure this difficulty is by measuring interannotator

agreement. For various aspects of the Information Extraction tasks, interannotator agreement has usually been in the 60-80% range. This gives some idea of how difficult a task it is. Another bit of evidence, of course, is how well the competing systems have done at MUCs. The state-of-the-art seems to have plateaued at around 60% of human performance, even after fairly intensive efforts ranging from a month to many person-months.

It seems safe to conclude that variations in this number across variations in applications is a crude measure of the relative difficulty of the applications. This latter in turn is determined by (i) the nature of the texts (ii) the complexity and variety of the kinds of information sought and (iii) the appropriateness of the chosen output representation to the information requirements of the task. We should note that not much is known in a systematic way about any of these factors. Still, there is a general consensus that the 60% figure represents a rough upper bound on the proportion of relevant information that "an average document" wears on its sleeve, that is the proportion that the creator(s) of the document express in a fairly straightforward and explicit way, and that doesn't require either complex syntactic processing or, more usually significant use of domain-specific knowledge assumed to be accessible to the document's human readers.

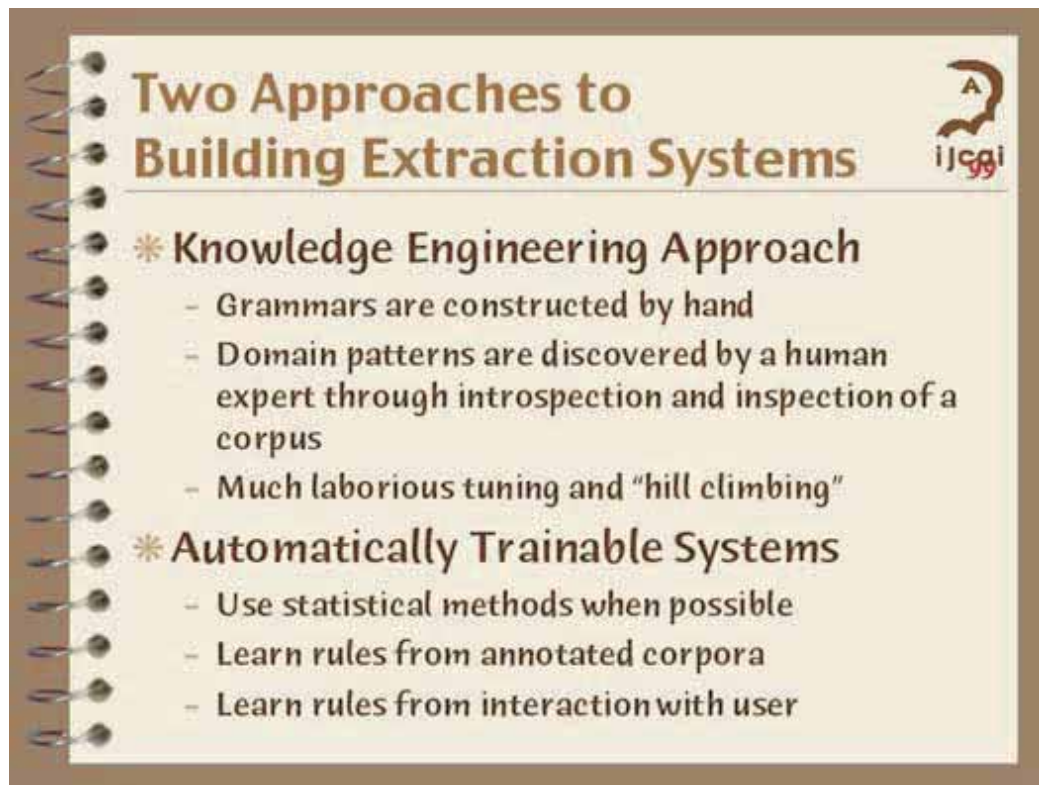
Whatever the source of the difficulty, there are questions to be asked about the 60% plateau. One, that we will not discuss here, is whether and to what extent that number is an artefact of the fairly complex nature of the target output representation and of the correspondingly complex scoring algorithm developed for MUC. Another is whether there are uses for a 60% technology, assuming that automatic creation of data bases from texts is not a likely candidate application for the foreseeable future---not at 60% accuracy. If one



Applications:

- * **Information Retrieval**
 - Routing queries: prespecified topics
 - Text classification/Routing
 - NL indexing for IR -- various experiments
 - High precision post-filter for IR
- * **Summarization**
 - Highlighting/Clipping
 - NL generation from formal output representation
- * **Automatic construction of databases and kb's from texts**

thinks of Information Retrieval in this light, perhaps the answer is, "Yes". Indeed, a number of experiments on using IE for IR have been and are being pursued. We shall return to these at the end of the tutorial.



Building Information Extraction Systems

At this point, we shall turn our attention to what is actually involved in building information extraction systems. Before discussing in detail the basic parts of an IE system, we point out that there are two basic approaches to the design of IE systems, which we label as the *Knowledge Engineering Approach* and the *Automatic Training Approach*.

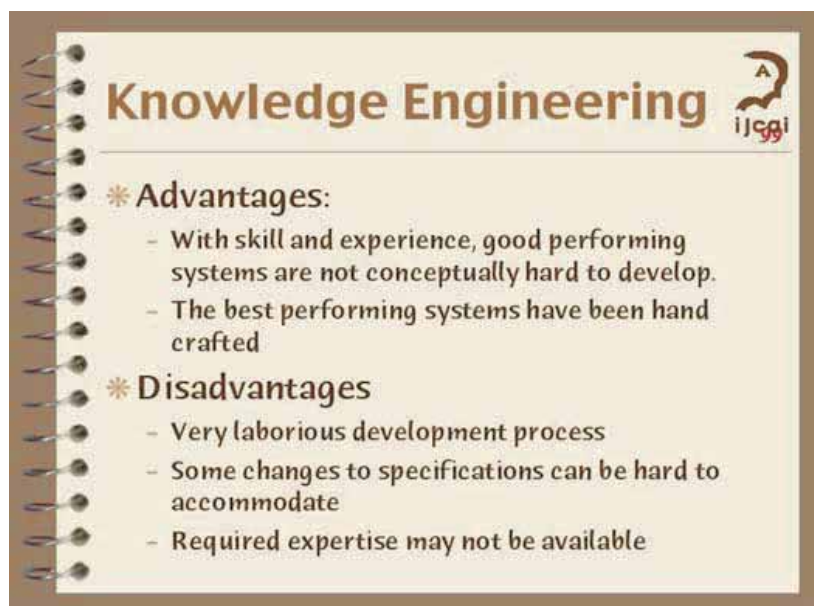
The Knowledge Engineering Approach is characterized by the development of the grammars used by a component of the IE system by a "knowledge engineer," i.e. a person who is familiar with the IE system, and the formalism for expressing rules for that system, who then, either on his own, or in consultation with an expert in the domain of application, writes rules for the IE system component that mark or extract the sought-after information. Typically the knowledge engineer will have access to a moderate-size corpus of domain-relevant texts (a moderate-size corpus is all that a person could reasonably be expected to personally examine), and his or her own intuitions. The latter part is very important. It is obviously the case that the skill of the knowledge engineer plays a large factor in the level of performance that will be achieved by the overall system.

In addition to requiring skill and detailed knowledge of a particular IE system, the knowledge engineering approach usually requires a lot of labor as well. Building a high performance system is usually an iterative process whereby a set of rules is written, the system is run over a training corpus of texts, and the output is examined to see where the rules under- and overgenerate. The knowledge engineer then makes appropriate modifications to the rules, and iterates the process.

The Automatic Training Approach is quite different. Following this approach, it is not necessary to have someone on hand with detailed knowledge of how the IE system works, or how to write rules for it. It is necessary only to have someone who knows enough about the domain and the task to take a corpus of texts, and annotate the texts appropriately for the information being extracted. Typically, the annotations would focus on one particular aspect of the system's processing. For example, a name

recognizer would be trained by annotating a corpus of texts with the domain-relevant proper names. A coreference component would be trained with a corpus indicating the coreference equivalence classes for each text.

Once a suitable training corpus as been annotated, a training algorithm is run, resulting in information that a system can employ in analyzing novel texts. Another approach to obtaining training data is to interact with the user during the processing of a text. The user is allowed to indicate whether the system's hypotheses about the text are correct, and if not, the system modifies its own rules to accommodate the new information.



To a scientist, the automatically trained systems seem much more appealing. When grounded on statistical methods they are backed up by a sound theory, one can precisely measure their effectiveness as a function of the quantity of training data, they hold out the promise of relative domain independence, and don't rely on any factors as imponderable as "the skill of a knowledge engineer."

However, human expertise and intuition should not be short-changed. Advocates of the knowledge engineering approach are eager to point out that higher performance can be achieved by hand-

crafted systems, particularly when training data is sparse.

This can lead to a sterile debate among partisans of the two approaches over which is "superior." Actually, each approach has its advantages and disadvantages, and each can be used to advantage in appropriate situations.

As was pointed out, the knowledge engineering approach has the advantage that to date, the best performing systems for various information extraction tasks have been hand crafted. Although automatically trained systems have come close to performing at the level of the hand crafted systems in the MUC evaluations, the advantage of human ingenuity in anticipating patterns that have not been seen in the corpus, and in constructing rules at just the right level of generality have given those systems a small but significant advantage. Also, experience suggests that given a properly designed system, a bright college undergraduate is capable of competently writing extraction rules with about a week of training, so "IE system expertise" is less of an obstacle than one might expect.

However, the knowledge engineering approach does require a fairly arduous test-and-debug cycle, and it is dependent on having linguistic resources at hand, such as appropriate lexicons, as well as someone with the time, inclination, and ability to write rules. If any of these factors are missing, then the knowledge engineering approach becomes problematic.

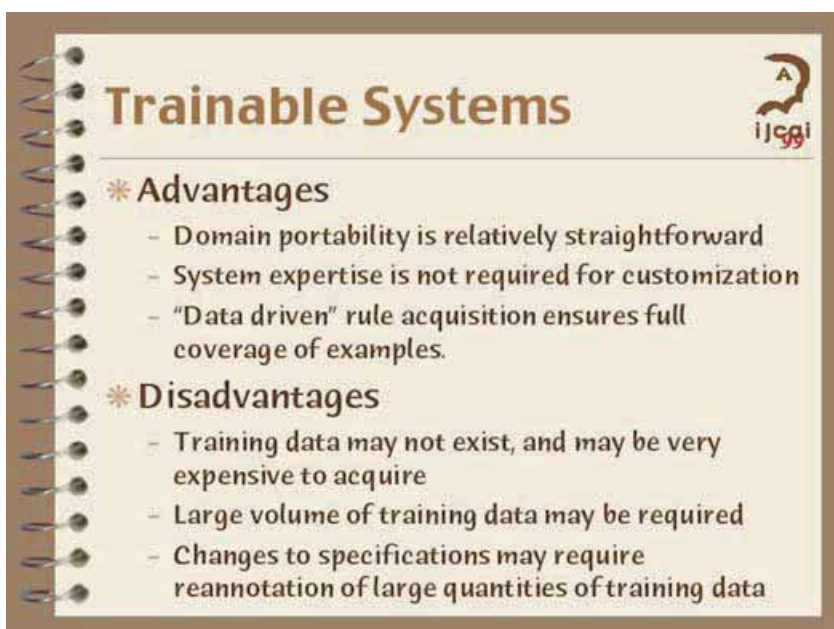
The strengths and weaknesses of the automatic training approach are complementary to those of the knowledge engineering approach. Rather than focusing on producing rules, the automatic training approach focuses on producing training data. Corpus statistics or rules are then derived automatically from the training data, and used to process novel data. As long as someone familiar with the domain is available to annotate texts, systems can be customized to a specific domain without intervention from

any developers. Name recognition is an ideal task for the automatic training approach because it is easy to find annotators to produce large amounts of training data — almost everyone intuitively knows what a “company name” is.

The disadvantages of the automatic training approach also revolve around the fact that it is based on training data. Training data may be in short supply, or difficult and expensive to obtain. Sometimes one may wish to develop an extraction system for a topic for which there are few relevant examples in a training corpus. Such situations place a premium on the human intuition of a good rule designer. If the relations that are sought are complex or technical, annotators may be hard to find, and it may be difficult to produce enough annotated data for a good training corpus.

Even for simple domains like proper names, there is always a vast area of borderline cases for which annotation guidelines must be developed. For example, when annotating company names, are nonprofit entities like universities or the Red Cross considered “companies?” There is no “right” answer to questions like that; the answers must be stipulated and clearly understood by all the annotators. This implies that considerable care must be taken to ensure that annotations are consistent among all annotators. Although evidence suggests that the quantity of data is more important than its quality, it is probably impossible to achieve truly high performance (e.g. F 95 or better on name recognition) with inconsistent training data. This implies that it might be more expensive to collect high-quality training data than one would initially suspect. In fact, for many domains, collecting training data can be just as, if not more expensive in terms of time and personnel, as writing rules can be.

Another problem worthy of consideration is the impact of shifting specifications on the rule writing or training task. It is certainly not the case that specifications for extraction rules will be set in concrete the moment they are thought up. Often, the end users will discover after some experience that they want the solution to a closely related, yet slightly different problem. Depending on exactly how these specifications change, it can impact Knowledge Engineered and Automatically Trained systems differently. Suppose a name recognizer is developed for upper and lower case text, and then the user decides that it is important to process monocase texts. The automatically trained systems can accommodate this change with minimal effort. One need only map the training corpus to all upper case and run the training algorithm again. A rule based system that relies heavily on case heuristics may have to be rewritten from scratch. Now suppose that an initial specification for extracting location names states that names of political jurisdictions are the kind of locations that matter. Later it is decided that the names of mountains, rivers, and lakes should also be recognized. The rule writer can accommodate this change by producing a handful of additional rules and adding them to the rule base. The automatically trained system is faced with a potentially much more difficult task, namely reannotating all the existing training data to the new specifications (this may be millions of words) and then retraining.

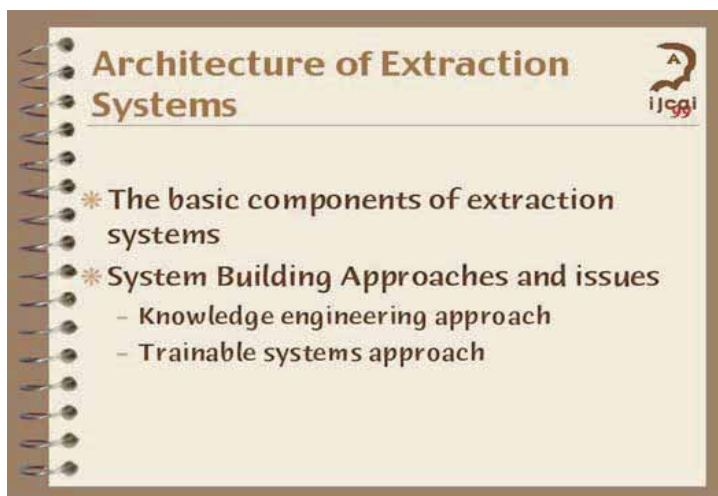


Trainable Systems

- * **Advantages**
 - Domain portability is relatively straightforward
 - System expertise is not required for customization
 - “Data driven” rule acquisition ensures full coverage of examples.
- * **Disadvantages**
 - Training data may not exist, and may be very expensive to acquire
 - Large volume of training data may be required
 - Changes to specifications may require reannotation of large quantities of training data



It is also worth pointing out that not every module of an IE system has to follow the same design paradigm. It is perfectly reasonable to produce a system with a rule-based name recognizer that learns domain rules, or with a statistical name recognizer that operates on hand-generated domain rules when data is scarce. The considerations relevant to deciding which approach to use are summarized on the accompanying slide.

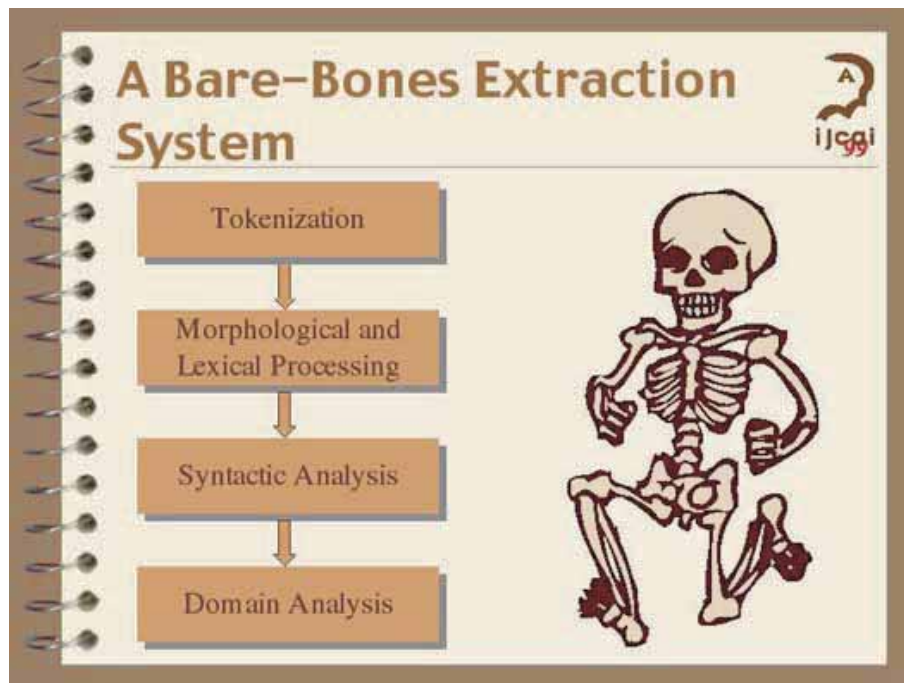


In general one should consider using a knowledge engineered system when linguistic resources like lexicons are available, there is a skilled rule writer available, training data is sparse, or expensive to obtain, it is critical to obtain the last small increment of performance, and the extraction specifications are likely to change slightly over time. Automatically trained systems are best deployed in complementary situations, where resources other than raw text are unavailable, training data can be easily and cheaply obtained, task specifications are stable, and absolute maximum performance is not critical.

Approaches to building both automatically trained and knowledge engineered systems components are discussed in more detail in the tutorial.

The Architecture of Information Extraction Systems

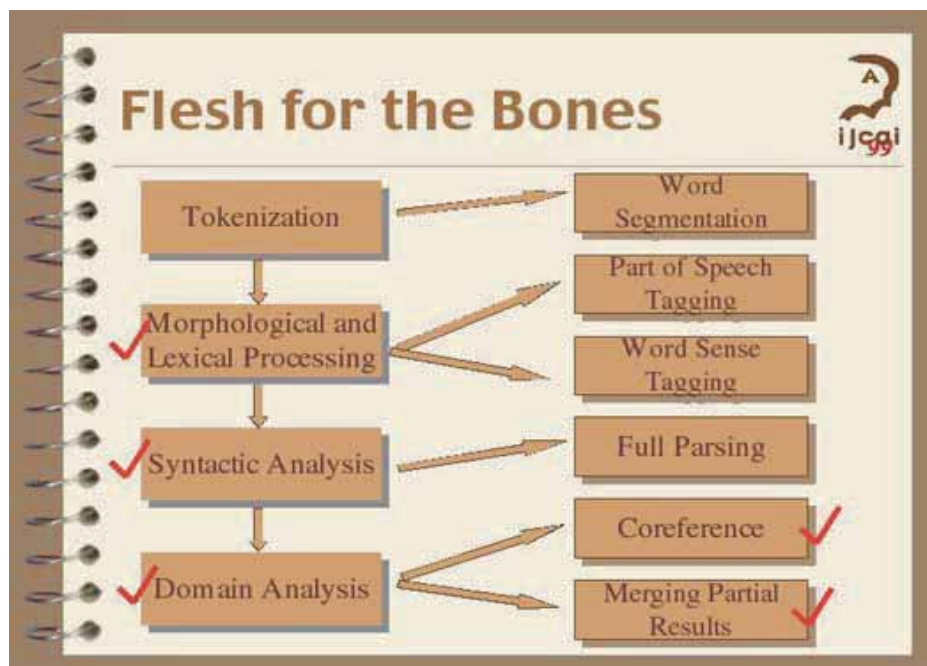
Although information extraction systems that are built for different tasks often differ from each other in many ways, there are core elements that are shared by nearly every extraction system, regard-



less of whether it is designed according to the Knowledge Engineering or Automatic Training paradigm.

The above illustration shows the four primary modules that every information extraction system has, namely a tokenizer, some sort of lexical and or morphological processing, some sort of syntactic analysis, and some sort of domain-specific module that identifies the information being sought in that particular application. (Actually, some extraction systems like name taggers actually stop at the lexical/morphological stage, but we are considering systems targeting events and relationships here.)

Depending on the requirements of a particular application, it is likely to be desirable to add additional modules to the bare-bones system illustrated above.

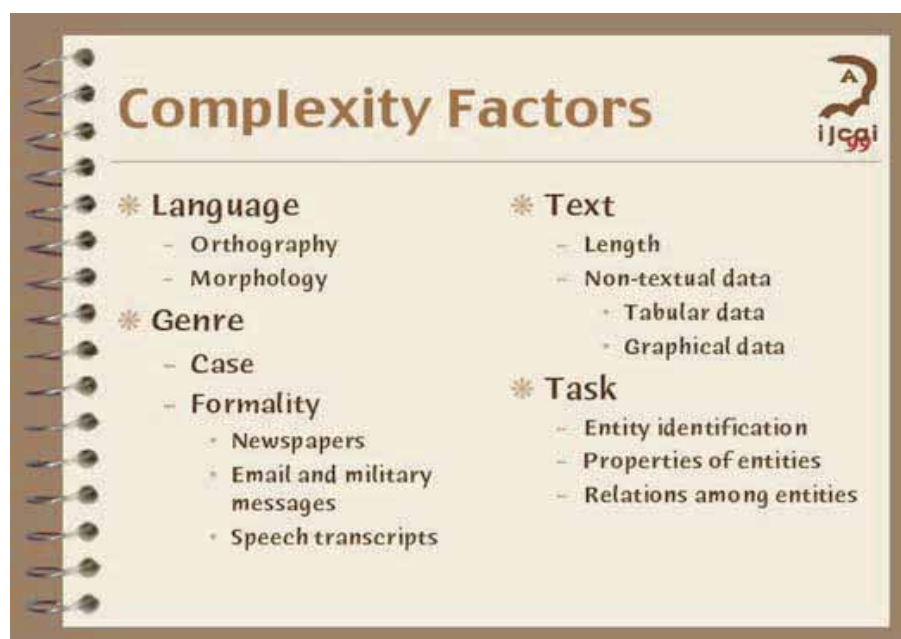


Tokenization is a trivial problem for European languages, requiring only that one separate whitespace characters from nonwhitespace characters. For most newspaper texts, punctuation reliably indicates sentence boundaries. However, in processing some languages like Chinese or Japanese, it is not evident from the orthography where the word boundaries are. Therefore extraction systems for these languages must necessarily be complicated by a word segmentation module.

In addition to normal morphological and lexical processing, some systems may choose to include various feature tagging modules to identify and categorize part of speech tags, word senses, or names and other open-class lexical items.

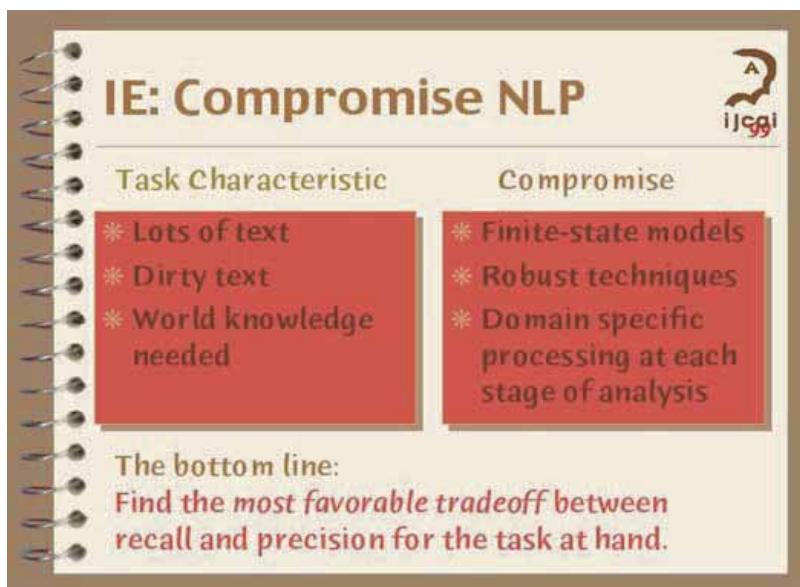
For many domains, a rudimentary syntactic analysis is sufficient to identify the likely predicate-argument structure of the sentence and its main constituents, but in some cases additional parsing, or even full parsing may be desirable.

Although it is possible to design an information extraction system that does not resolve coreferences or merge partial results, in many cases it is possible to simplify the domain phase and increase performance by including modules for that purpose.



In summary, the following factors will influence whether a system needs modules over and above the “bare-bones” system:

- Language of the text. Some languages will require morphological and word segmentation processing that English does not require.
- Genre. Extracting information from speech transcripts requires different techniques than text. For example, one may need to locate sentence boundaries that are not explicitly present in the transcript. Texts with mixed-case letters simplify many problems and eliminate much ambiguity that plagues single-case text. Informal text may contain misspellings and ungrammatical constructs that require special analysis that newspaper text in general does not need.
- Text properties. Very long texts may require IR techniques to identify the relevant sections for processing. Texts that contain images or tabular data require special handling.
- Task. Tasks like entity identification are relatively simple. If one wants to extract properties of entities, then the text needs to be analyzed for fragments that express the property. If the task involves extracting events, then entire clauses may have to be analyzed together.

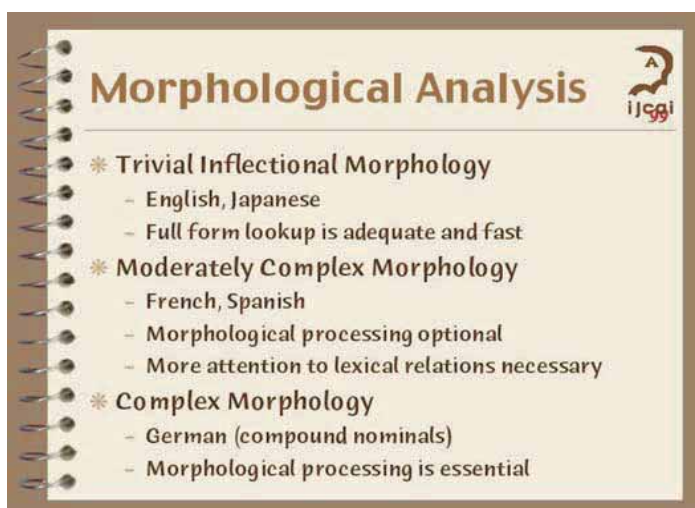


In this tutorial we will discuss how these modules are constructed, however we will not spend much time discussing modules for which the specific application of information extraction has no particular bearing. For example, the application of a part of speech tagger to a text in an information extraction system is no different than the application of a part of speech tagger in any other application. Therefore, we encourage you to read other sources in the literature to find out more about part of speech taggers.

In understanding the influence that the general information extraction problem has on the design of IE systems, it helps to understand information extraction as “compromise natural language processing.” A number of demands are placed on information extraction systems in terms of the quantity and quality of the texts that are processed. To meet these demands, some compromises have to be made in how natural language processing is carried out that are not necessary in other domains like, for example, database question answering.

Typically, IE systems are required to process many thousands of texts in a short period of time. The compromise made to satisfy this constraint is the use of fast but simple finite-state methods. Processing large volumes of real-world texts implies the adoption of robust techniques that will yield acceptable performance even in the face of spelling and grammar errors. Also, the problems to which IE systems are typically applied would typically require a great deal of domain-specific world knowledge to handle properly with a general natural-language processing system. The IE compromise is to build extraction systems that are very highly dependent on their particular domain of application. Although the core of an extraction system contains some domain independent components, domain-dependent modules are the rule. This domain dependence arises through the incorporation of domain-specific extraction rules, or the training of the system on a corpus of domain-relevant texts.

The Components of an Information Extraction System



Morphological Analysis

Many information extraction systems for languages with simple inflectional morphology, like English, do not have a morphological analysis component at all. In English, it is easy to simply list all inflectional variants of a word explicitly in the lexicon, and this is in fact how the TextPro system mentioned earlier works. For languages like French, with more complex inflectional morphology, a morphological analysis component makes more sense, but for a language like German, where compound nominals are agglutinated

into a single word, morphological analysis is essential.

Lexical Lookup

The next task that is usually performed in the lexical/morphological component of an extraction system is lexical lookup. This raises the question of precisely what the lexicon should include. Since information extraction systems must deal with highly unconstrained real-world text, there is a great deal of temptation to try to cover as much of the language as possible in the system's lexicon. In addition, the specific domain of application is likely to introduce a sublanguage of terms relevant to that particular domain. The question arises about whether one should try to broaden the lexicon so that it covers almost any likely domain-specific sublanguage, or augment the lexicon with domain-specific entries for each application.

It is somewhat paradoxical that bigger does not necessarily imply better when lexicons are considered. The larger the lexicon, the more likely it is to contain rare senses of common words. My favorite example is the word "has been" as a noun in the COMLEX lexicon. The problem is, that unless some rare sense is an important part of a domain-relevant sublanguage, the presence of these rare senses in the lexicon will at best complicate parsing, and at worst create a space of possibilities in which incorrect analyses of common phrases are likely to be found and selected. We have had the actual experience of updating an extraction system by doing nothing more than adding a bigger, more comprehensive lexicon. The bottom line performance of the system actually *declined* after the "improvement."

The general lesson to draw from this experiment is to *beware the large list*. Large lists of anything, including person names, locations, or just a large lexicon of ordinary words, tend to have unintended consequences caused by the introduction of unexpected ambiguity. In many cases, augmenting a lexicon or word list with a small quantity of domain-specific information is better than using a large superset of the required vocabulary just because it is available. If large lists are employed, it is almost always necessary to devise a strategy for dealing with the ambiguity introduced thereby.

Part of Speech Tagging

As mentioned earlier, some extraction systems do various kinds of tagging, including part of speech tagging, to make subsequent analysis easier. It might be conjectured that part

Lexical Coverage

- * Does more mean better?
 - The larger the lexicon, the more likely it is to contain rare (domain irrelevant) senses
 - COMLEX includes "has been" as a noun
 - Spurious lexical ambiguity causes errors
 - At best, parsing is complicated
 - At worst, performance declines
- * Beware the large list
 - Domain specific lexicon augmentation may be superior to large general lexicon

Part of Speech Tagging

- * Advantages
 - Ambiguity can potentially be reduced
 - Avoid errors due to incorrect categorization of rare senses.
- * Disadvantages
 - The errors taggers make are often those you would most want them to eliminate
 - High performance requires training tagger on corpus of similar genre to domain texts
 - It takes time

of speech tagging would be a good way to deal with rare word senses, and the spurious ambiguity that can be introduced by large name lists.

Part of speech tagging is certainly useful toward that end, however, if the *only* reason one does part of speech tagging is to avoid incorrect analyses caused by rare word senses, there may be easier and faster ways to accomplish the same ends.

Even the best part of speech taggers, whether rule based or statistically based, are correct about 95% of the time. This may sound impressive, but it turns out that many of the cases where the information provided by a tagger would be most useful

are precisely the cases in which the tagger is most likely to make errors. This eventuality has to be balanced against the fact that part of speech tagging does not come for free. It takes some time to do, and some effort to train, particularly if the texts to be processed by the system are from a highly domain-specific sublanguage, like military messages, for which taggers trained on normal data are less accurate.

If what one is primarily interested in is the elimination of rare word senses, it may make more sense to use simple frequency data to reject an analysis when it employs a rare sense of a word, and competing analyses exist using a common sense. Taggers incorporate the same information by assigning a very low prior probability to the tags for rare senses. This check is very quick and simple, and

while not perfect, is a good compromise between accuracy and efficiency.

Names and Structured Items

One of the most important tasks for the lexical and morphological component of an extraction system to deal with is to assign lexical features to lexical items that may have internal structure, but be too numerous to explicitly enumerate (e.g. dates, times, spelled-out numbers) and proper names. Proper names are particularly important for extraction systems, because typically one wants to extract events, properties, and relations about some particular object, and that object is usually identified by its name.

Although proper names are of critical importance, they present some difficult problems for an extraction system. One problem is that proper names are huge classes and it is difficult, if not impossible to enumerate their members. For example, there are hundreds of thousands of names of locations around the world. Many of these names are in languages other than the one in which the extraction system is designed, and hence are likely to

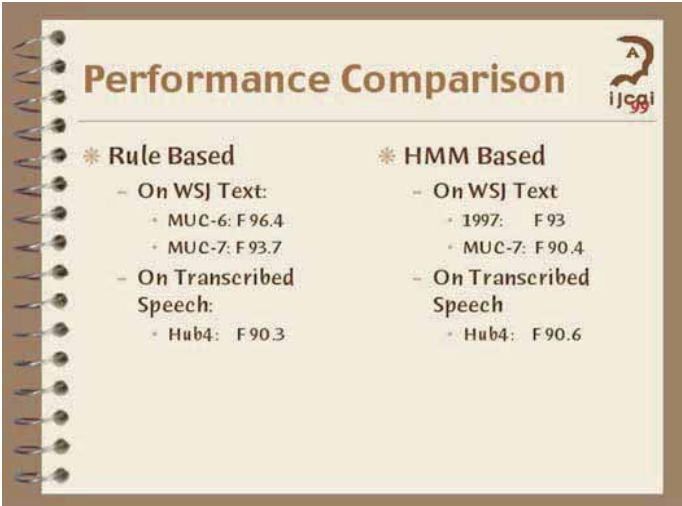
An Effective Part of Speech Compromise

- * **Strategy**
 - Collect a table of unigram tag frequencies
 - Establish a "rareness" threshold
 - Deprecate analyses with rare senses
- * **Results**
 - Very low processing overhead
 - Small increase in correct syntactic analyses
 - Full POS tagging not clearly better, and probably not worth the cost

Dealing with Open Class Lexical Items

- * Classes are very large
- * New members constantly coming into being
- * No strict rules involving coinage
- * **Examples:**
 - Locations
 - Persons
 - Companies
 - Organizations
 - Proprietary products

cause spurious ambiguity with other items in the system's lexicon. Locations tend to be fairly stable over time, however, new companies come into being all the time; in fact many Wall Street Journal articles are about the formation of new companies. These new company names will not be on any preexisting list. Person names from different languages are likely to have different conventions for the order of family and given name. Typically there are no strict rules governing the coinage of new names, and this is particularly true for the names of new products, which can have almost any internal structure. Our favorite example is the name of the margarine, *I Can't Believe It's Not Butter*.



Performance Comparison

* Rule Based	* HMM Based
- On WSJ Text:	- On WSJ Text
- MUC-6: F96.4	- 1997: F93
- MUC-7: F93.7	- MUC-7: F90.4
- On Transcribed Speech:	- On Transcribed Speech
- Hub4: F90.3	- Hub4: F90.6

The difficulty of the name recognition task depends on the type of text one is analyzing. In English, upper and lower case make it relatively easy to recognize the fact that a sequence of words is a proper name. The key problem is determining what kind of a name it is. If the text is all one case, as may well be the situation when one is processing the output of a speech recognizer, simply recognizing that a word is a name may be difficult given the considerable overlap between proper names and ordinary English words.

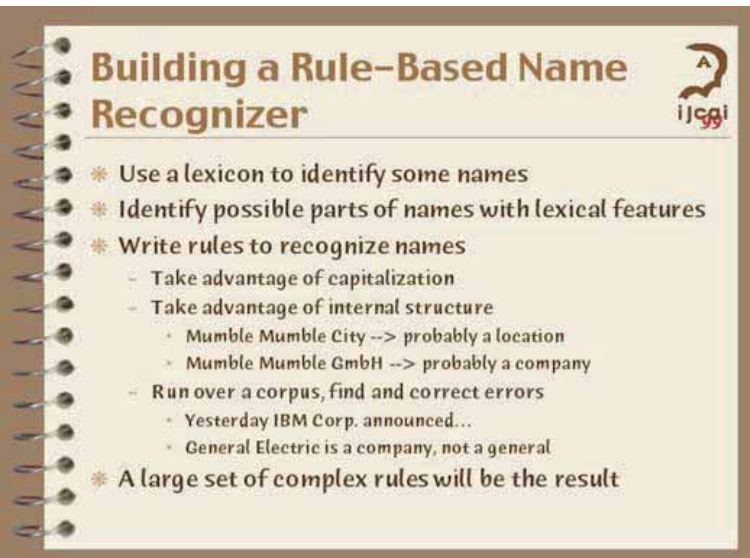
Because of the presence of proper names in almost any text that one would wish to analyze, and because of the importance of proper names in the identification of objects of interest, almost every information extraction system implements some kind of proper name identifier. As we mentioned earlier, it is possible to design most components of an extraction system by approaching the design problem as one of building a hand-crafted rule-based system, or an automatically trained system.

Although several approaches are possible, the most well-known name recognition systems are based on Hidden Markov Models. Finite state rule based systems for name recognition have also been developed that attain a very high level of performance. Two such rule systems are publicly available over the Web: SRI's FASTUS and TextPro systems. You can find pointers on the tutorial web site.

As mentioned earlier, rule-based systems tend to have a performance advantage over automatically trained systems in recall and precision. Evaluations over the past few years have supported this

assertion with respect to name recognizers. The most recent performance evaluation, the DARPA-sponsored Broadcast News workshop in March, 1999, shows the rule-based and automatically trained systems with virtually the same performance on transcribed speech. The ability of the BBN *Identifinder* name recognizer to close this gap is due to a combination of having a good underlying statistical model, together with much more training data than was available for earlier evaluations.

There is not a great deal to say about building a rule-based name recognizer,



Building a Rule-Based Name Recognizer

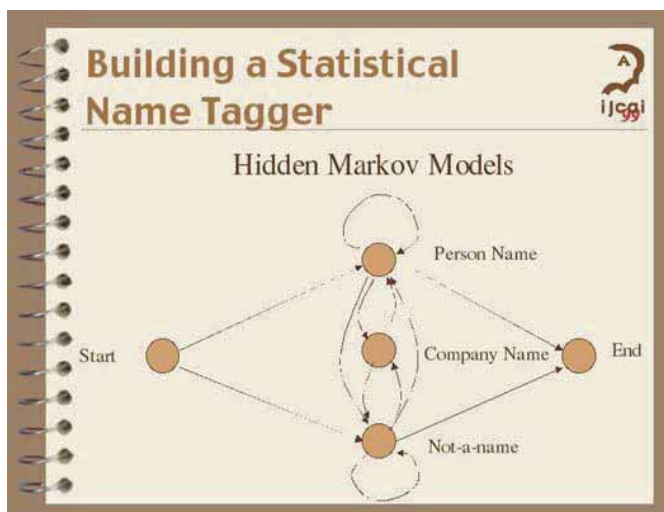
- * Use a lexicon to identify some names
- * Identify possible parts of names with lexical features
- * Write rules to recognize names
 - Take advantage of capitalization
 - Take advantage of internal structure
 - Mumble Mumble City --> probably a location
 - Mumble Mumble GmbH --> probably a company
 - Run over a corpus, find and correct errors
 - Yesterday IBM Corp. announced...
 - General Electric is a company, not a general
- * A large set of complex rules will be the result

other than that it is more a matter of time and effort than exceptional skill and cleverness. One begins with lexicons that categorize names. Often it is easy to recognize a name in mixed-case text because of capitalization, but only by looking the name up in a lexicon is it possible to determine what type of name it is. Fortunately, many names have internal structure. A pattern like “<Word> <Word>, Inc.” almost certainly designates a company. After scanning a corpus of text with several hundred names, it is easy to write rules that cover the kinds of examples seen.

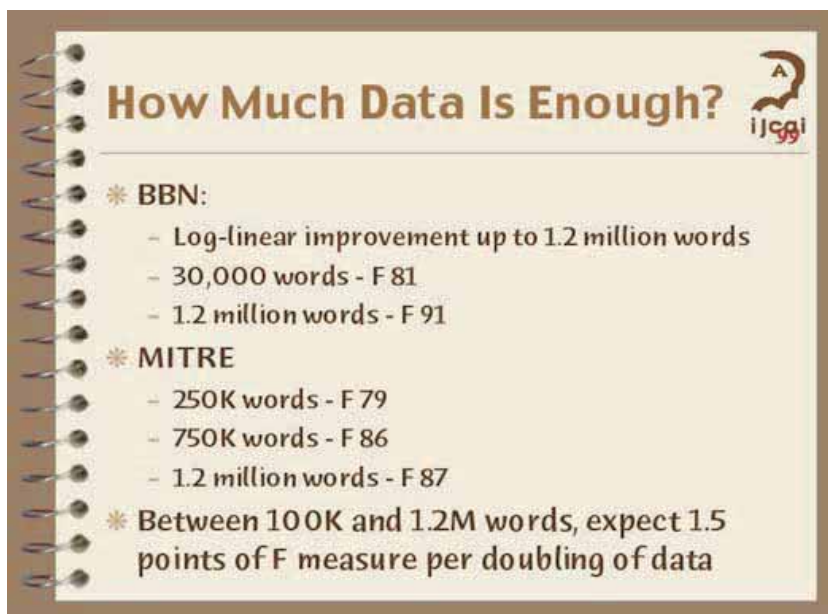
Once some rules have been written, it is good to evaluate the system by running it on a corpus of texts to observe what kinds of names are still missing, and where mistakes are being made. Mistakes that are easy to make are those involving coincidentally capitalized words next to names like “Yesterday IBM announced...” or perhaps tagging “General Electric” as a person with a military title. You will discover many tiresome, ad-hoc facts, like “Michigan State” is the name of a university, while “New York State” is a location.

Assuming that one starts with a lexicon of about 5,000 company names and abbreviations, about 1,000 human first and last names, and about 20,000 domestic and international locations, iterative rule writing and error correcting should yield a solidly performing recognizer with two to three weeks of effort, consisting of around 100 rules.

A particularly good way to construct a trainable name tagger is to base one on Hidden Markov Models. A Hidden Markov Model is a particular kind of probabilistic model based on a sequence of events, in this case, sequential consideration of the words in a text. It is presumed that the individual events are parts of some larger constituents, like names of particular type. Whether a word is part of a name or not is a random event with an estimable probability. For example, the word “John” could refer to a person, and hence be a name, or it could refer generically to the client of a prostitute, and hence be an ordinary common noun. The probability of name versus non-name readings can be estimated from a training corpus in which the names have been annotated. In a Hidden Markov model, it is hypothesized that there is an underlying finite state machine (not directly observable, hence hidden) that changes state with each input element. The probability of a recognized constituent is conditioned not only on the words seen, but the state that the machine is in at that moment. For example, “John” followed by “Smith” is likely to be a person, while “John” followed by “Deere” is likely to be a company (a manufacturer of heavy farming and construction equipment). So constructing an HMM recognizer depends on two things: constructing a good hidden state model, and then examining enough training data to accurately estimate the probabilities of the various state transitions given sequences of words.



-
- Hidden Markov Model**
- * Hidden state transition model governs word sequences
 - * Transitions are probabilistic
 - * Estimate transition probabilities from an annotated corpus
 - $P(s_j | s_{j-1}, w_j)$
 - * At runtime, compute maximum likelihood path through the network
 - * Viterbi algorithm

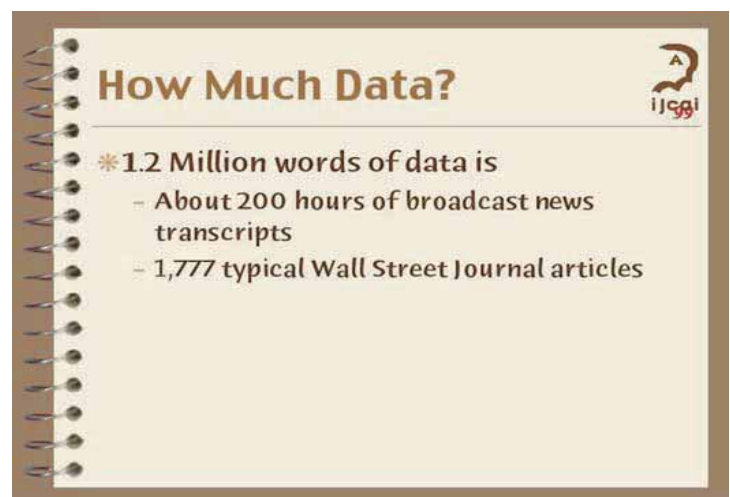


When the recognizer is being run, it computes the maximum likelihood path through the hidden state model for the input word sequence, thus marking spans of input that correspond to names. The search algorithm usually used to find such a path is called the Viterbi algorithm. This algorithm is well explained in literature on speech recognition.

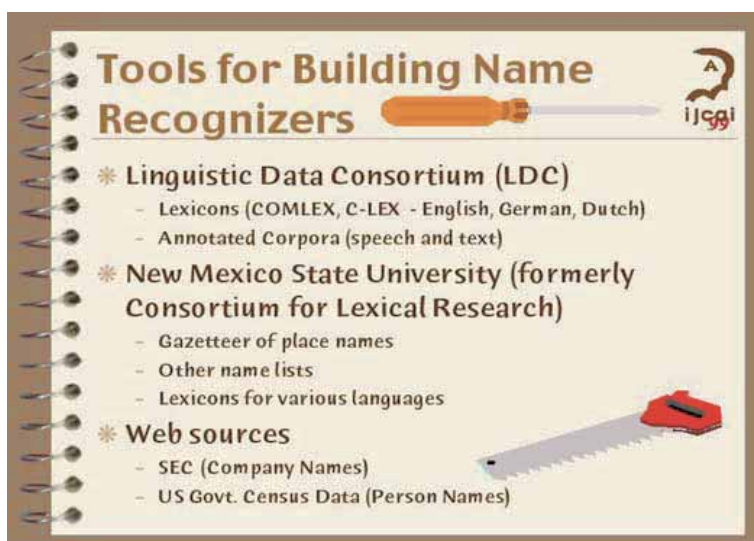
How much data is necessary to train an HMM name recognizer so that it achieves state-of-the-art performance? There is some empirical data about the application of HMMs to recognizing names in

transcribed speech that suggests that the amount of data required is substantial, but not overwhelmingly so.

It is generally true that the performance of trainable modules is a function of the quantity of training data, at least up to a point. Recent experience from two developers of HMM-based name recognizers that system performance bears a roughly log-linear relationship to the training data quantity, at least up to about 1.2 million words of training data, and beyond that point, no data has been publicized. One can expect a rough performance increase of about 1.5 points in F-measure for each doubling in the quantity of training data, depending on precisely what the underlying model being trained is. The Mitre system seemed to top out at a lower level of performance than the BBN system, but the reason for this is not clear. BBN has claimed that the quantity of training data is more important than the precise consistency of the data, however they concede that for very high levels of performance, the accuracy of the training data becomes more critical.



Obtaining 1.2 million words of training data requires transcribing and annotating approximately 200 hours of broadcast news programming, or if annotating text, this would amount to approximately 1,777 average-length Wall Street Journal articles. To guarantee a high degree of accuracy in the training data, it is a good idea to have the annotations reviewed by another annotator. Assuming that annotation of broadcast news data proceeds at approximately double real time, this would imply that it would require about 800 hours, or 20 person-weeks of labor to annotate and check the amount of data required to reasonably expect F-90 level performance from a trainable name-recognizer. This is almost certainly more time than would be required by a skilled rule writer to write a rule-based name recognizer achiev-



ing the same level of performance, assuming all the necessary resources, such as lexicons and name lists are already available.

Assuming that one doesn't have the tools immediately at hand for building a rule-based name recognizer, where might they be obtained? Unfortunately, computational resources for some languages are scarce, but for English, most European languages, as well as Japanese and Chinese, the situation regarding the availability of resources is quite good. If one resource that you do have available is money, then the Linguistic Data Consortium has a great deal to offer. LDC has comprehensive lexicons available for some languages (COMLEX for English, and CELEX for English, Dutch and German), annotated data for training parsers and part of speech taggers (the Penn Treebank), and other data

useful for building information extraction systems, including all the annotated MUC and TIPSTER data. To find out precisely what is offered and the terms under which it may be obtained, you should visit their web site at <http://www ldc.upenn.edu/>.

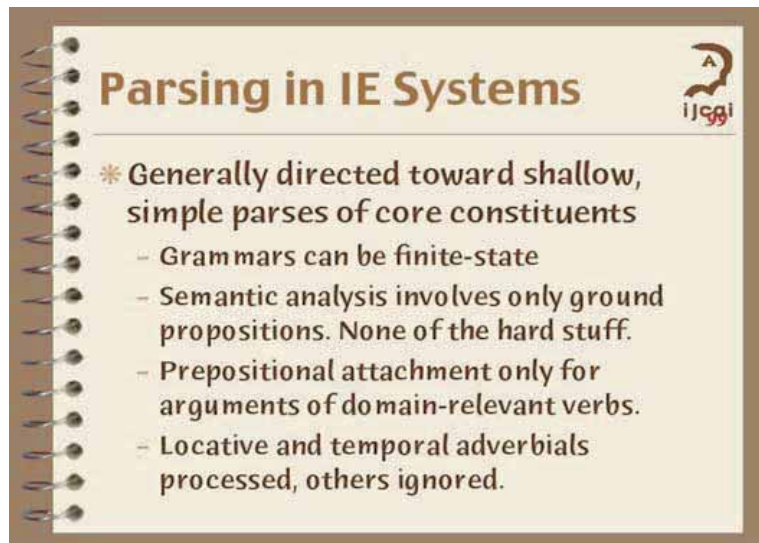


You should also pay a visit to the web site of the Computing Research Laboratory at New Mexico State University (<http://crl.nmsu.edu/Resources/resource.htm>). New Mexico State University used to host the Consortium for Lexical Research about five years ago, which had a number of lexical resources, including lexicons, parsers, and taggers available for a variety of languages. The Comput-

ing Research Laboratory makes these resources publicly available for research purposes. They have also developed a number of software tools that assist in various aspects of the development of information extraction systems.

Finally, you should not ignore the possibility of obtaining information free of charge from various databases on the World Wide Web. The Securities and Exchange Commission's EDGAR database is a good way to get a list of every publicly traded corporation on U.S. stock exchanges (more than 14,000 companies) and the Census Bureau is also a good source of person-name data.

In addition to these general resources, SRI International has published various aspects of its own IE systems. You can examine the SRI FASTUS name recognition and parsing grammars (written in its own native FASTSPEC formalism), and you can download the TextPro system, mentioned earlier, which includes a name recognizer and parser written in the Common Pattern Specification Language (CPSL).



Parsing in IE Systems

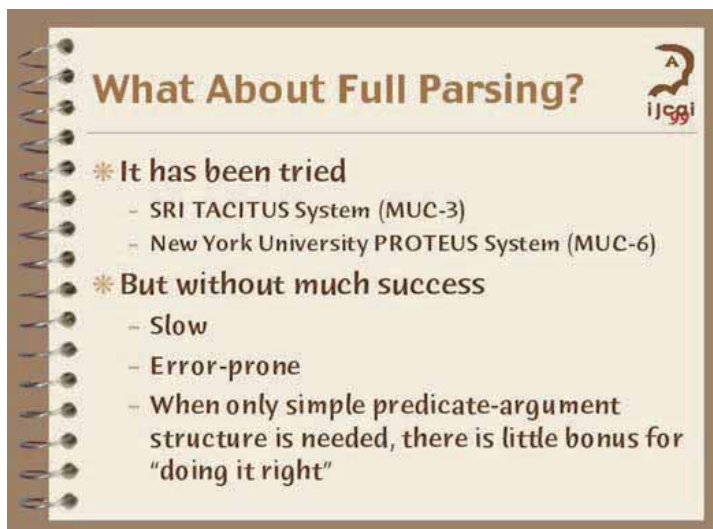
- * Generally directed toward shallow, simple parses of core constituents
 - Grammars can be finite-state
 - Semantic analysis involves only ground propositions. None of the hard stuff.
 - Prepositional attachment only for arguments of domain-relevant verbs.
 - Locative and temporal adverbials processed, others ignored.

Syntactic Analysis in Information Extraction Systems

As mentioned previously, natural language processing in information extraction systems is often a question of making compromises to what one might consider ideal natural-language processing, driven by the need to process large quantities of real-world text within reasonable time constraints. Fortunately, the fact that information extraction is properly directed toward extracting relatively simple relationships among singular entities mitigates the adverse effects of some of these compromises.

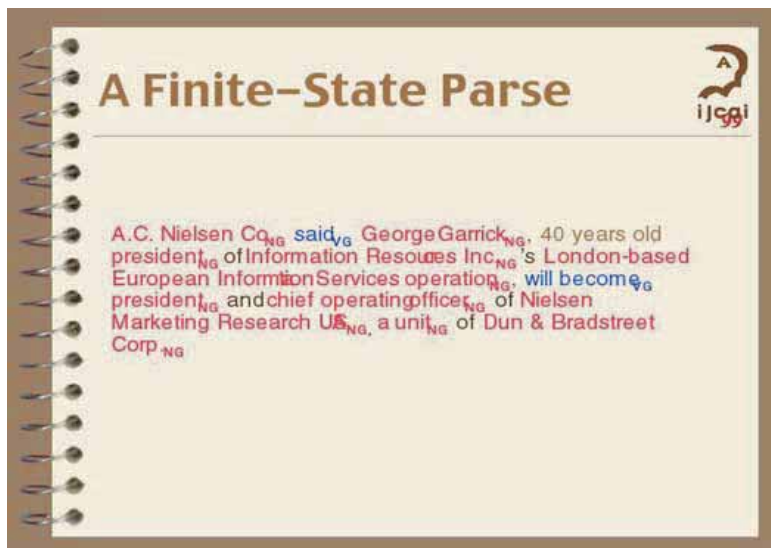
Typically, information extraction systems are designed to produce structurally simple fragment parses. These simple fragments can be described with finite-state grammars that can be processed easily, robustly and quickly. Semantic analysis is usually limited to finding the predicate argument structure of a small set of core propositions. Constructs that cause semantic difficulties, such as quantifiers, modals, and propositional attitudes are treated in a simplified manner, if they are handled at all. One of the factors that complicates parsing is the attachment of modifiers, and constituents such as prepositional phrases. In information extraction systems, these kind of attachment decisions are usually ignored for all except a small set of domain-relevant words. In this simpler set of cases it is possible to apply domain heuristics to getting attachment decisions correct that would be difficult to do in general. Because one has some idea of what the domain relevant types of objects are, it is often possible to distinguish prepositional arguments from locative prepositions for the narrow set of cases of interest. In the cases where the sentences are not domain-relevant, no information would be extracted, so a correct analysis is irrelevant, except to the extent that it might complicate the analysis of domain-relevant clauses in the same sentence.

One may well wonder what the penalty one pays for making the shallow-parsing compromise that one usually makes in information extraction, and if there might be something to gain by really doing full parsing. Experience suggests that it is in fact advantageous to make the compromises



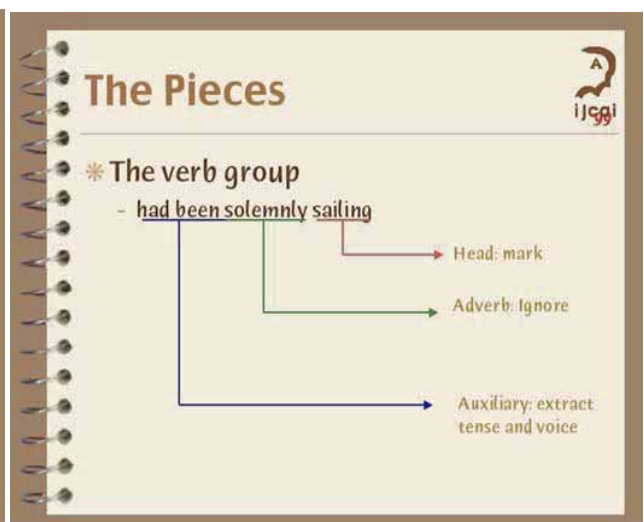
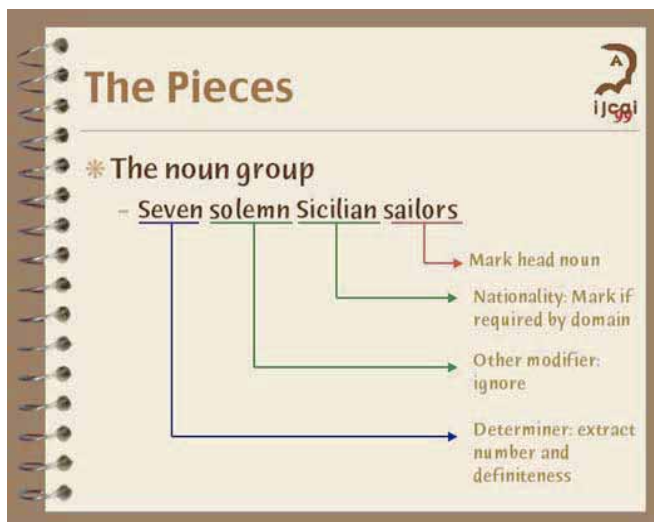
What About Full Parsing?

- * It has been tried
 - SRI TACITUS System (MUC-3)
 - New York University PROTEUS System (MUC-6)
- * But without much success
 - Slow
 - Error-prone
 - When only simple predicate-argument structure is needed, there is little bonus for "doing it right"



outlined above. Full parsing, particularly without the benefit of domain-specific guidance, is error prone, and the combinatorial explosion on long sentences mean that full parsing is in general slow. It is not unusual in newspaper text to encounter sentences that are fifty words long, and even hundred word sentences are not unheard of. Because the information targeted by extraction systems is simple, and confined to a small set of domain-relevant events and relationships, the shallow analysis typical of information extraction systems is adequate most of the time. Even if a grammar and full parser were capable

in theory of finding a correct analysis, there is no guarantee that it will find its way through the competing ambiguities. Therefore, the experience has been that full parsing offers very little gain that



comes with a large price tag. A consensus has therefore emerged that shallow, finite state analyses are the superior alternative.

The illustration at left shows what a typical finite-state analysis of a fairly complex sentence would look like. Noun groups and verb groups are marked, as well as “particles,” a category that includes prepositions, conjunctions, and relative pronouns. The analysis does not, however, attempt to make any decisions about how these particles form constituents and attach to larger structures. To the extent that these decisions have to be made at all, they are made during domain-specific analysis.



The above illustrations show what kinds of meaningful information is extracted from the basic constituents. This particular analysis is what is produced by the SRI International FASTUS system, but it is fairly typical of the kind of shallow analysis used by IE systems. In general, the heads of constituents

are marked. Number and definiteness are extracted from the determiner of noun groups, and information about tense and voice is extracted from the auxiliary in verb groups. Typically, the semantics of adjectival and adverbial modifiers are ignored, except for a small number of domain-relevant cases. Particles constitute their own constituents, and are further analyzed by domain rules, provided they occur in domain-relevant contexts.

Of course, it is not possible for IE systems to completely ignore complex syntactic structures because sometimes relevant information is conveyed in complex sentences. However, because of the shallowness of the semantic analysis, it is often possible to use expedient techniques that extract the correct information from complex sentences, even if such analyses would be inadequate in general.

Relative clauses attached to the subject of a clause can be analyzed by a nondeterministic domain rule, assuming both clauses contain domain-relevant information. In this case, the subject noun group is the subject of both the main clause and the relative clause, and it requires only that a rule skip a relative pronoun to attach the subject to the relative clause. To get the connection between the subject and the main clause, the domain pattern must be written to optionally skip a relative clause.

If both rules work together, then the information from the entire sentence is correctly recovered. Relative clauses attached to objects are harder to handle, because they are more likely to contain gaps. There is no easy way to handle these sentences, and sometimes they can just be ignored, because they are less common in formal writing. The penalty for missing these sentences is surprisingly small.

Coordination presents what is probably the most difficult problem in syntactic analysis for any natural-language processing system. There is no single approach to handling coordination in shallow parsing — the particular approach taken depends on just what is being coordinated, as well as domain-relevant properties of the sentence.

What About Complex Syntactic Structures?

- Subject Relatives: exploit domain phase nondeterminism:
The company that formed a joint venture with Ford invested in another venture in Japan.
- Object-position gaps are harder to handle:
The company that Ford formed a joint venture with • invested in another venture in Japan.
- Domain patterns optionally skip relative clauses

Coordination

- * Sentential - ignore it
- * VP conjunction - exploit nondeterminism
- The company invested \$2 million and set up a joint venture.
- * NP conjunction - handle domain specific cases in a second parser phase
- * N, Adj, Det conjunction - build into parser grammar.
- * Other conjunction - hard to impossible

Prepositional Phrases

- * Second parsing phase can attach common close-attaching PPs (of, for)
- * Second parsing phase can also attach locatives to domain-relevant noun groups
- * Argument PPs can be analyzed by domain phase rules triggered by particular domain-relevant verbs
- * All other PPs are treated as adverbial adjuncts

IE systems typically do not handle disjunction except in some limited cases, because simple semantic representations such as template structures are not sufficiently powerful to represent disjunction in general. Therefore, we will confine our discussion to conjunction.

Sentential conjunction is easily handled because at the level of analysis typical of IE systems, it can be ignored. Conjunction is usually implicit in all the information extracted. Verb phrase conjunction requires a technique similar to the handling of relative clauses outlined above. A domain rule recognizes the subject,

then optionally skips the first conjunct of the verb phrase before recognizing what is probably the second conjunct. If the rule is allowed to apply twice, then both verb phrases will be properly associated with the subject.

Conjunction of simple noun groups is best handled by additional syntax phases. If two noun groups are separated by a conjunction, particularly if the two potential conjuncts refer to objects of the same domain-relevant type, a rule can combine them into a single noun group. Conjunctions of lexical items like nouns, adjectives, and prepositions are best handled directly by grammar rules.

Conjunctions of non-constituents is essentially impossible to handle with any “tricks,” because the early syntactic phase cannot produce any non-constituents as output.

The attachment of prepositional phrases is another problem for shallow parsing systems. Certain prepositions, notably “of” and “for” are usually close-attaching, and the second syntactic analysis phase can often handle these prepositions. If the prepositions are subcategorized for by domain relevant verbs, and the objects are domain-relevant, then the domain phase can interpret the arguments appropriately. Otherwise, all prepositional phrases are treated as adverbial adjuncts. Temporal and locative adjuncts are typically interpreted by a domain phase, assuming the event that they modify is domain-relevant. Even if not, the domain phase may track the temporal and locative modifiers in case they can be merged with other events. Prepositional phrases with no discernible domain relevance are treated like adverbials, although since they make no semantic contribution to the analysis, it does not really matter precisely how they are treated.

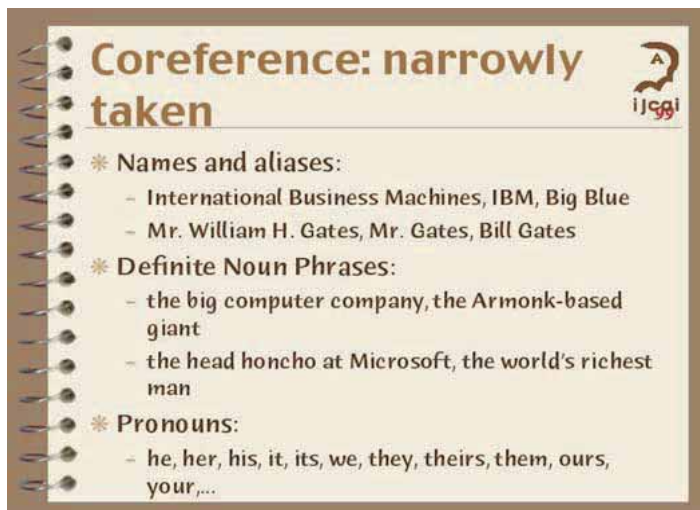
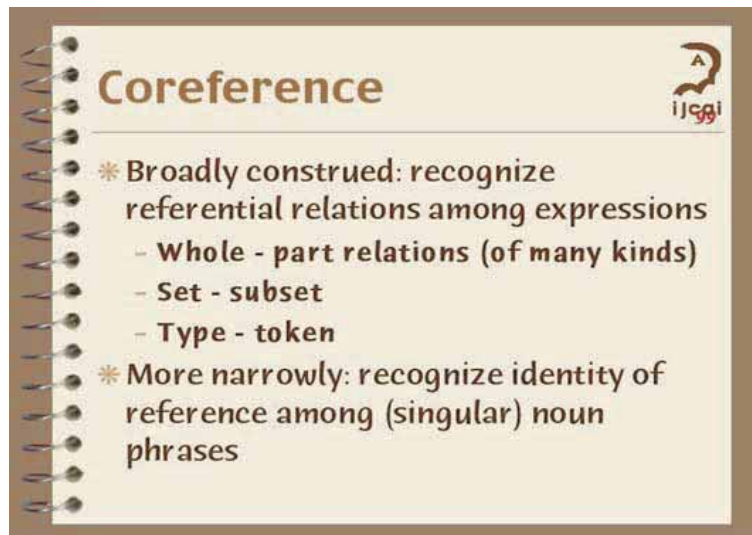
The Bottom Line

- * Shallow parsing makes mistakes. Get used to it.
- * Most errors in final output are due to factors other than parsing correctness
- * Need to robustly exploit partial and incorrect analyses to extract partial information.

The important fact to bear in mind about parsing in information extraction systems, is that fast, robust techniques are necessary, and this necessarily implies simple, shallow analyses. The result is that errors will be made in parsing, and that some of the analyses will blur syntactic and semantic distinctions that would be made in a linguistically correct analysis. However, as long as the domain is a problem suitable for the application of information extraction techniques, the price one pays for making errors in syntactic processing is small. Most of the errors will creep into the analysis from other sources, including named entity identification, coreference, and inadequate domain coverage.

Coreference in Information Extraction Systems

MUC-6 introduced a new task on which systems were to be evaluated: the Coreference Task. In a very real sense, though, IE systems had always been evaluated on this task, if only implicitly. The reason for this is simply that application relevant entities will be referred to in many different ways throughout a given text and thus, success on the IE task was, to a least some extent, conditional on success at determining when one noun phrase referred to the very same entity as another noun phrase. In what is perhaps the simplest kind of case, this might mean recognizing full identity of strings. But consider what might be thought of as the next simplest case: recognizing the various forms of a personal proper name. Thus an article on Microsoft might include “William H. Gates,” “Mr. Gates,” “William Gates,” “Bill Gates,” “Mr. Bill H. Gates,” even “Billy Gates” (??). This problem of recognizing different forms of a personal proper name gets worse when you consider that Gates might, for all we know, be a place name (consider “Lincoln,” “Washington”) or even a company name (consider “Ford,” “Dupont”). How about book titles, e.g., “David Copperfield?” And, of course, there are the problems posed directly by recognizing the different forms of company or other organizational names.



Consider “International Business Machines” and “IBM;” “Atlantic Richfield Company” and “ARCO”, “General Motors Corporation,” “General Motors” (a major nuisance), and “GM.” Such cases suggest the following routine: assume that name-recognizer does an adequate job on the full form of such names. Suppose the system has recognized a word or sequence of words with (at least) initial letter capitalized. Check such words to see if they might be acronyms or aliases for a previously recognized name in the given text. This check can be done as follows: If the unknown name is a single word,

Role of Coreference in IE 

- * Objects involved in relevant events and relationships are referred to in many different ways and often at widely separate locations in a text

Motor Vehicles International Corp. announced a major management shakeup. **MVI** said **its** CEO had resigned. **The big automaker** is attempting to regain market share. **It** will announce significant losses for the 3rd quarter. **A company** spokesman said **the company** will be moving **their** operations...**MVI, the first automaker to announce quarterly results, is the biggest American auto exporter to Latin America.**

check to see whether it can be formed by taking the initial letters of a full, multiword name occurring elsewhere in the text. (All caps is often an indication that the item is an acronym or abbreviation of such a name.) In other cases, check whether the item contains substrings from such a name.

Before moving on to consider other types of coreference, a brief word on temporal references is in order. It is easy enough to imagine IE applications for which it would be essential to determine either the absolute or at the very least the relative temporal location of events, where by absolute temporal location we mean that given by a calendar date and clock time. But of course, in texts we also encounter references to such temporal locations in other forms, e.g., "today", "four days ago", "three weeks from Monday," etc. To determine coreference here one must (i) determine the date (of (publication of production) of the text and (ii) do a little calendar arithmetic.

The special case of temporal coreference illustrates the point that reference and hence coreference can be to all manner of entities. Just to give a flavor of the thing: (i) groups or collections, as in "The jury deliberated for only 15 minutes after they heard Doug testify about the IE system. It quickly decided ..." (ii) to events, as in "The testimony was riveting; it completely destroyed the prosecution's case." (iii) to abstract entities of indeterminate nature, as in "The case, its weakness apparent to anyone who dared look at it, ..." Then there are references to types or kinds: "Juries usually consist of 12 people; but in some jurisdictions, they can have 6 or 9 members. A jury is often unable to agree on where to eat lunch."

Reference to groups or collections, in particular, is the most obvious case in which the importance of capturing a referential relationship beyond identity manifests itself. We often introduce a group and then refer to its members or to subgroups, as in "The jurors decided where to eat. All liked sushi, but five preferred sashimi; the one who was chosen foreperson wanted Macdonald's." In this tutorial, we will simply ignore all such relationships.

The case of names is often quite an important one, but it is not what most theoreticians have in mind when they speak of the problem of coreference. Typically they have in mind the problems of resolving pronominal and discourse-definite reference. These form the core of what is called "anaphora" or "anaphoric reference"—that is, reference by one term that is dependent in some way on the reference of another, typically earlier term occurrence. Thus, with respect to pronoun reference, con-

Two Approaches to Coreference

- * **Knowledge Engineering Approach:**
 - Based on adapting theoretical work on coreference to the shallow and incomplete parsing strategy of IE
 - Coreference resolution as providing/supporting key to discourse structure
- * **Automatically trained systems**
 - A (small) range of approaches
 - probabilistic and nonprobabilistic

sider: “Bill Gates dropped out of Harvard; history has shown that not graduating didn't hurt him one bit.” By discourse definite reference, we mean such cases as the following: “I saw a man yesterday with a cat in his hat. The cat was a tabby; the man was shabby.” Consider also the following kind of case: “Bill Gates is the richest man in the world....Many people in the software industry fear and respect the guy.” The first definite description “the richest man in the world” does, indeed, refer (let us stipulate) to Gates, but it does so in a sense independently of the earlier occurrence of “Bill Gates;” the second description in contrast, is dependent for its referent on that earlier occurrence.

occurrence.

There is a large and rich literature on the problem of anaphora in the above limited sense. Suffice it to say here (i) that the literature in question presupposes at the very least full syntactic analyses of the texts and (ii) that work on pronominal and discourse-definite coreference in IE system can be seen as attempting to approximate the kinds of algorithms presented in this literature, adapted to the context of very sparse and incomplete syntactic input. Here we sketch a generic IE coreference algorithm.

Knowledge Engineering Approach: Step 1

- * **Mark each candidate referring expression -- each noun phrase -- with:**
 - Sortal information (animacy, sort: company vs. location)
 - Number (singular vs. plural)
 - Gender (masc., fem., neutral)
 - Syntactic features (name, pronoun, definite/indefinite)
 - Perhaps other grammatical features and relation

The Coreference Algorithm: Step 2

- * **For each candidate referring expression:**
 - Determine *accessible antecedents*
 - Filter with *semantic/sortal consistency check*
 - Order (partially or totally) by *dynamic syntactic preferences*

It is assumed that certain syntactic and semantic features are recognized. It may even be assumed that certain surface grammatical relations, especially that of being the subject of a sentence (clause) or direct object of a verb are recognized. The algorithm is structured around the idea that different types of referring expressions have different backward-looking scopes within which to look for possible antecedent referring expressions---that is, to look for “accessible antecedents.” Thus, for instance, the scope of a proper name can be taken to be the whole text. The scope of a discourse definite seems much narrower; that of a pronoun, narrower still. The precise size of these scopes is best determined experimentally; the larger the scope, of course, the more candidates to be checked and the more room for error.

Accessible Antecedents

- * For names: entire preceding text
 - Match using aliases/acronyms
- * Definite noun phrases: part of the preceding text--to be determined experimentally
 - Independent of paragraph structure
- * Pronouns: some smaller part of preceding text
 - Paragraph limits seem reasonable; 1? 2??

Semantic Consistency

- * Check given expressions with accessible antecedents for:
 - Number
 - NB: **The company** moved **their** operations
 - Gender
 - **The ship** swept **her** bow...
 - Sortal consistency
 - Role of sort hierarchies
 - Manual construction
 - Ontology resources: Cyc, Wordnet

As we have seen already, the case of determining coreference between proper names is rather a special one, best handled by the name-recognition module. So we shall focus on the treatment of pronouns and discourse definites.

Once a set of possible antecedents for a given noun phrase has been determined, they can be filtered by checking for semantic consistency. Thus, for example, “Mr. Gates” will be filtered out of the set of antecedents for an occurrence of the pronouns “she” or “it” or “them.” That is, the algorithm checks for consistency in number and gender. Similarly, “Bill Gates” can’t be an antecedent of “the company” or “the jury.” Consider the following: “The big automaker is planning to get out the car business. The company feels that it can never longer make a profit making cars.” To recognize the coreference among “the big automaker,” “it” and “the company,” the system has to recognize that an automaker is a company. This recognition can be realized either by manually constructing an application-specific sort hierarchy or by using a more general, application-independent resource such as Cyc (<http://www.cyc.com/>) or Wordnet (<http://www.cogsci.princeton.edu/~wn/>). The problem with the such resources is that, even assuming that all the required classificatory information is available, one has to figure out how to make use of it.

After filtering out semantically inconsistent accessible antecedents, one then has to choose which of the remaining candidates, if any, are most likely antecedents. To do this, one takes advantage of their relative locations in the text. First one looks for candidate earlier in the same sentence, preferring those that occur earlier in the natural left-to-right order. Thus, in “American Airlines called for mediation in its negotiation with its union,” both “its” corefer with “American Airlines,” not with semantically con-

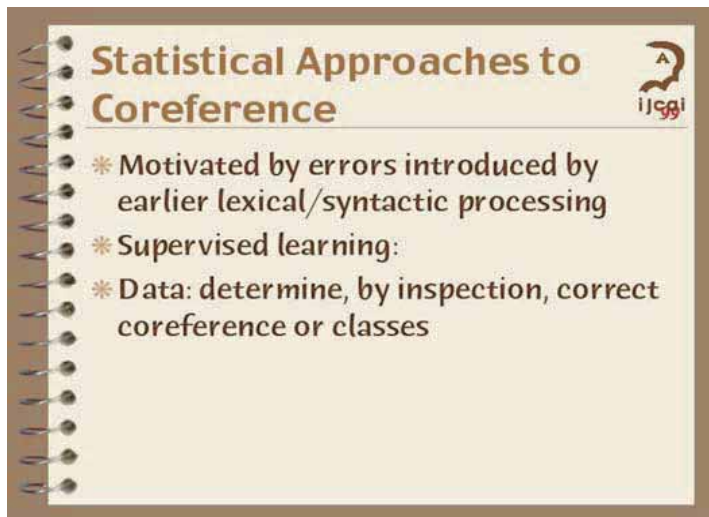
Dynamic Semantic Preferences

- * Order accessible and semantically consistent antecedent by relative location in text:
 - Preceding part of the same sentence (Left-Right)
 - Immediately preceding sentence in same paragraph (Left-Right)
 - Earlier sentences, in order in the same paragraph (Right-Left)
 - Sentences in previous paragraphs (Right-Left)

State of the Art

- * MUC6: Precision = .72, Recall = .63
 - Limited to subset of singular np's
- * MUC7: UPenn's High Precision system
 - Precision = .80, Recall = .30
 - Again on limited test

sistent “negotiation” or “mediation.” This heuristic, though, fails to capture cases in which parallelism between elements, especially in different clauses of a single sentence, plays a role in determining coreference. Thus in “American Airlines called the request for mediation premature and characterized it as a bargaining ploy,” the “it” does not corefer with “American Airlines,” but rather with “the request (for mediation).”



If there are no candidate antecedents earlier in the sentence, look to the immediately preceding sentence, again preferring candidates that occur earlier in that sentence in left-to-right order. If nothing comes up, look back at earlier sentences in the same paragraph, moving backward a sentence at a time, but now, within a given sentence prefer most rightward candidates, that is, those occurring later in the sentence. Finally, if the scope extends back beyond a paragraph boundary, look to sentences in the preceding paragraph, again preferring later (more rightward) to earlier occurrences.

Aside from coreference between names, the above algorithm is silent for the case of

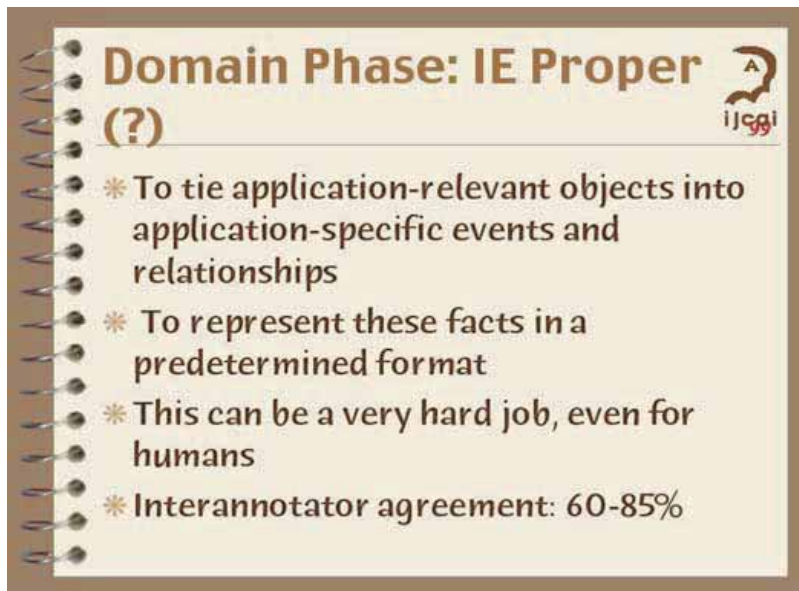
indefinites as well. Indefinites are usually used to introduce entities into texts and thus are not likely to corefer with expressions occurring earlier; but there are exceptions. Sometimes indefinites are used to refer to types or species. Thus one might see: “Whales are large marine mammals. A whale can grow to be 100 feet long and weigh more than 80 tons.”

The fact that IE parsers are incomplete, shallow and not fully reliable motivates looking for some form of statistical or corpus-based approach to coreference. (See, for example, TE-36 under <http://ciir.cs.umass.edu/info/psfiles/tepubs/te.html>.) The basic idea can be sketched as follows: produce, by hand, a key in which all coreference pairs are tagged as such. Determine which system-recognizable features of the individual expressions and of the coreference pairs are relevant to the coreference judgment and apply some learning technique using the resulting feature-vectors. The learning technique can be probabilistic or nonprobabilistic; in the case of UMASS's RESOLVE system, nonprobabilistic decision trees were constructed/trained.

The above idea is based on coreference pairs, but as we have seen above, we may need to group together all coreferring expressions, that is, we may need to construct a set of chains of such coreference pairs or more fully a set of equivalence classes of referring expressions, equivalent precisely in that every member of such a class refers to the same thing. Moreover, certain applications may require that the output of an IE system be probabilistic. For instance, the IE system may feed a downstream system which fuses its output with possibly contradictory information from other sources;



to do this the fusion system needs to know the IE system's degree of certainty with respect to its results and to possible alternatives. A natural way to think of this is to think of a probability space consisting of all referring expressions, whose 'events' in turn consist of all subsets of these—that is all possible



Domain Phase: IE Proper (?)

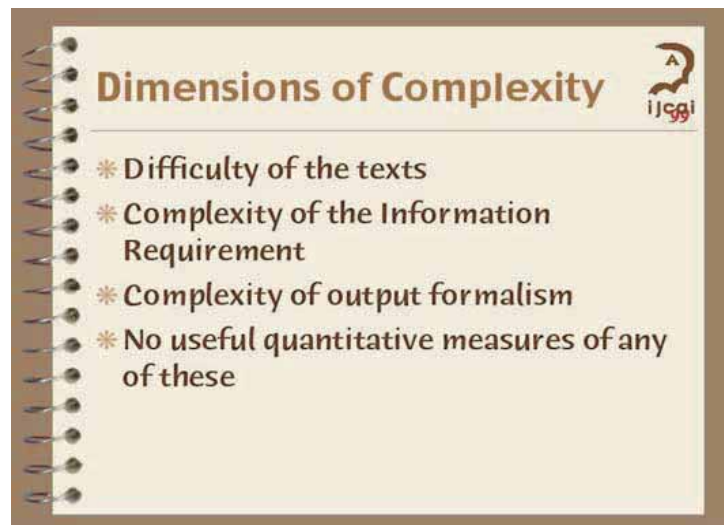
- * To tie application-relevant objects into application-specific events and relationships
- * To represent these facts in a predetermined format
- * This can be a very hard job, even for humans
- * Interannotator agreement: 60-85%

equivalence classes. One wants then to induce a probabilistic model over this space. But this cannot be done by simply aggregating pairwise probabilities of coreference, as these pairwise probabilities need not be consistently combinable into larger equivalence classes. Suppose we have a text with just three referring expressions: A, B and C. Suppose further that given the training corpus it is .505 probable that expression A and B corefer and that it is .504 probable that B and C corefer. Suppose we know that A and C cannot corefer. Then the two pairings of A and B and B and C exclude each other. But then their probabilities sum to

greater than 1. There are various approaches to such problems of combination. We shall not explore these here; it is enough to note that there are significant problems in coming up with an adequate probabilistic treatment of coreference.

Extraction of Domain-Specific Events and Relations

With the exception of name recognition, none of the many intermediate tasks performed by an IE system of the kinds we have been discussing are of much independent interest. The ultimate goal of such a system is to extract information from texts. What information? That is to be determined by the client---not just determined but also clearly specified and not, one hopes, by the client alone, without some input from the IE system developers. We have already said that the kinds of texts most apt for IE systems are those whose primary purpose is the communication of factual information. Even such texts contain hints, if only for the knowledgeable, as to the motives and purposes and perhaps even the emotions of the human agents involved in the events/relationships described. But just as IE systems are not well-suited to discerning the mental state of the writer of the texts, such systems are extremely unlikely to be any good at discerning such features of cognitive state of the agents described in the texts. Second, IE systems may not be well suited for tasks in which almost every sentence in a large corpus, and indeed almost every 'piece of information' in those sen-



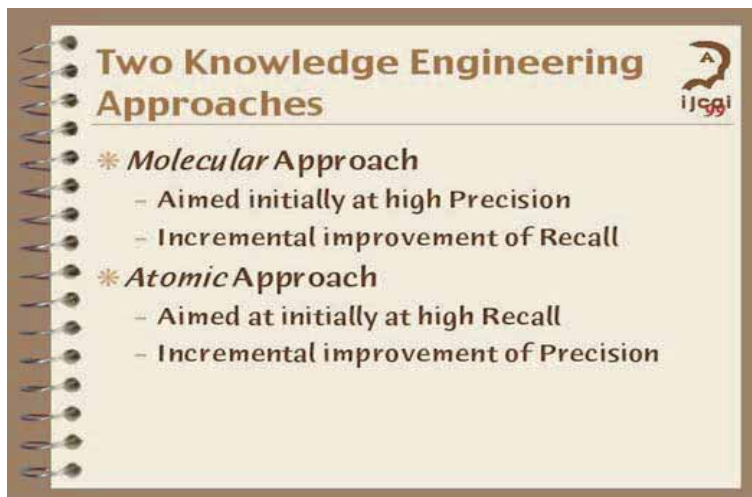
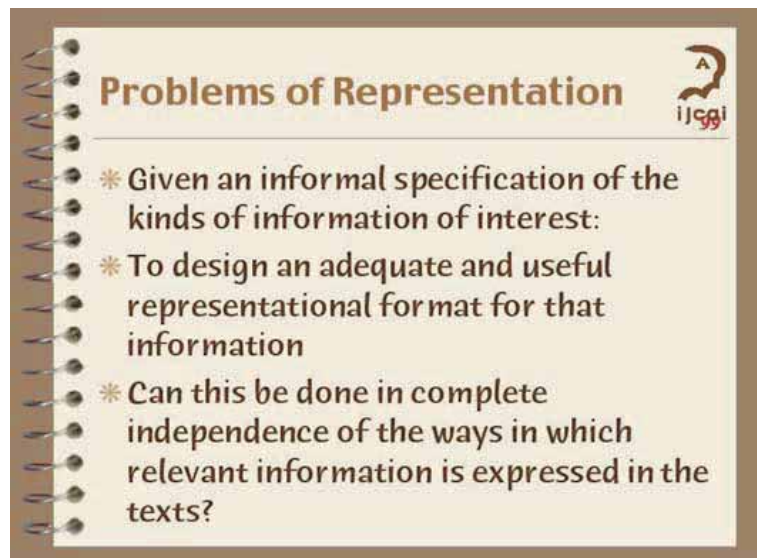
Dimensions of Complexity

- * Difficulty of the texts
- * Complexity of the Information Requirement
- * Complexity of output formalism
- * No useful quantitative measures of any of these

tences is relevant. So it is important that clients/users and system developers agree on a reasonable and feasible task specification. Then an output representation must be decided on and designed.

This task can be done well or badly—or anywhere in between. The design of templates, or more generally, abstract data structures, as output forms for automatic information extraction systems must be sensitive to three different but interacting considerations: (i) the template as a representational device, (ii) the template as something to be generated from the text input (iii) the template as input to further processing, by humans or programs, or both. The last consideration, we hope, is clear, though we make no claims to having thought through human-factors issues determining readability/graspability of various structures. The first point is little enough touched on; it is briefly discussed in a paper by Hobbs and Israel <ftp://ftp.ai.sri.com/pub/papers/hlt94.ps.gz> and at greater length in ftp://ftp.ai.sri.com/pub/papers/ord_report.ps.gz. The second point, intimately connected to the first, is that there should be a good match between the output representation of the information to be extracted and the typical mode of expression of that information in the texts.

Consider MUC-6 for example. The task was to extract information about management changes. The template for the task was organized around 'succession events'. These had slots for an organization, a post, a reason for the change and a 'in_and_out' slot with a rather complicated structure. Even without considering any texts, this is a rather unnatural structure given the information requirement. It is even more unnatural given the texts. For example, it is often the case that a single event report (e.g. "John Smith left Microsoft to head a new subsidiary at Apple.") corresponds to multiple succession events--Smith's leaving Microsoft and his entering Apple. Conversely, it is (even more) typical to have a single succession event expressed by multiple sentences (event-reports), often far removed from one another in the text. Also, static information (e.g. "John Smith has been chairman for the last five years.") is often essential to filling the final template, although the succession event structure provides no way of representing this static information. We felt that a more appropriate representation of the domain involved two kinds of structures: states and transitions. A state consists of the association among a person, an organization, and a position at a given point in time. A transition is a ternary relation between states and reasons, associating a start state and an end state with a transition reason. A post processor was written to generate the official MUC-6 templates from



report (e.g. "John Smith left Microsoft to head a new subsidiary at Apple.") corresponds to multiple succession events--Smith's leaving Microsoft and his entering Apple. Conversely, it is (even more) typical to have a single succession event expressed by multiple sentences (event-reports), often far removed from one another in the text. Also, static information (e.g. "John Smith has been chairman for the last five years.") is often essential to filling the final template, although the succession event structure provides no way of representing this static information. We felt that a more

appropriate representation of the domain involved two kinds of structures: states and transitions. A state consists of the association among a person, an organization, and a position at a given point in time. A transition is a ternary relation between states and reasons, associating a start state and an end state with a transition reason. A post processor was written to generate the official MUC-6 templates from

The Molecular Approach

- * The standard Knowledge Engineering approach
- * Read texts
- * Identify common, highly reliable relevant patterns capturable given the chosen class of grammars
- * Write rules to cover these
- * Move to less frequent, less reliable patterns

Molecular Approach

- ComplexNG --> Company | Person | Position
- Person --> Person-Name (Position)
- Position --> Position-Title (and Position-Title) ("of" Company)
- Company --> Company's Company | Company, Company | Company-Description of Company-name

```
[A. C. Nielsen Co.]_co [said]_vo
[George Garrick, 40 years old,
 [president of
 [[Information Resources Inc.]_co]'s
 [London-based European Information Services operation]_co ]_res
 ]_per,
 [will become]_vo
 [president and chief operating officer of
 [[Nielsen Marketing Research USA]_co],
 [a unit of Dun & Bradstreet Corp.]_co ]_res
```

this internal representation. (We have found that even after negotiations with client/users, such a strategy is often necessary.)

The Atomic Approach

- * Determine sortal features of relevant entity types
- * Determine features of possible participants in relevant event types
- * Every occurrence of a noun/verb phrase with these features triggers creation of a candidate event
- * Merge resulting partial templates
- * Filter using application-specific criteria

Conditions of Appropriateness of Atomic Approach

- * Relevant entities have easily determined types
- * Only one or two slots an entity of a given type can fill
- * Only entities of a given type can fill a given slot
- * It helps if events/relations are symmetrical
- * Examples: labor negotiations, MUC-5 microelectronics

Assuming that a reasonable task specification and output format have been arrived at, the next stage in developing a knowledge- or rule-based IE system is to read some texts and identify the most common and most reliably indicative clausal patterns in which relevant information is expressed. One then builds rules to cover these patterns and then moves on to less common, but still reliable patterns, etc. This is the standard, *molecular approach*—molecular because it focus on clausal or even full sentential molecules of text and complete molecules of information that fully specify relevant events or relationships. This approach aims first at high precision, settling for low recall, the expectation being that as one moves to less common, but still reliable clausal patterns, recall will go up without too high a cost in precision. We should note here that yet another thing to get clear about with the clients or intended end-users is what kind of a precision-recall trade-off they are aiming at: Which is more important for their application: avoiding false positives (high precision) or avoiding false negatives (high recall)?

There is another approach to the development of rule-based IE systems. We call it the *atomic* approach; the method here is to start with high recall and low precision, with the expectation that more reliable filters of false positives can be developed at not too great a cost in recall. The basic idea is to assume that every noun phrase of the right sort and every verb of the right type, independently of the syntactic relations obtaining among them, indicates an event/relationship of interest. Such a

design leads to a proliferation of very partial descriptions of possible events/relationships; one then has to merge these descriptions in some way to produce more fully instantiated events/relationships and then to filter the results according to some application-specific criteria.

Merging

- * Linguistic analysis, except coreference, operates within the boundaries of single sentence
- * Event merging vs. entity-level coreference
- * Merging is a form of *unification*
- * Merging uses coreference for slot-by-slot checks

Merging: The Need

A. C. Nielsen Co. said **George Garrick, 40 years old, president of Information Resources Inc.'s London-based European Information Services operation, will become president and chief operating officer of Nielsen Marketing Research USA, a unit of Dun & Bradstreet Corp.** He succeeds John H. Costello, who resigned in March.

Merging: The Abstract Picture

	Person: -- Position: president/coo Org: NMR	==>	Person: Garrick Position: president/coo Org: NMR
+	Person: Costello Position: -- Org: --	==>	Person: Garrick Position: -- Org: --
=	Person: Costello Position: president Org: NMR	==>	Person: Garrick Position: president Org: NMR

This is a reasonable approach for certain extraction tasks, even those tasks for which high recall and low precision is not an acceptable tradeoff. Such tasks are characterized by the following features: (1) entities in the domain have easily determined types and (2) the templates are structured so that there is only one or a very small number of possible slots that an entity of a given type can fill and only entities of a given type can fill those slots. The microelectronics domain of the MUC-5 evaluation was a good example of a domain with these characteristics, and techniques similar to these were successfully applied by at least one system in that evaluation. The same was true for labor negotiations applications used as a training exercise for MUC-6.

The existence of the two approaches raises the question as to whether, and how, results from both, that is, from high recall and high precision systems, can be combined to produce a result that would be better than either system taken on its own. The answer is by no means obvious.

In any event, even in high precision, low recall systems, we must merge partial descriptions of relevant events/relationships. Merging is essentially a unification operation; the precise specifications for merging are provided by the system developer when the domain template is defined. The developer specifies for each slot what type of data is contained in that slot, and for each data type, the system provides procedures that compare two items of that type and decide whether they are identical or necessarily distinct, whether one is more or less general than the other or the two are incomparable. Depending on the results of this comparison, the merge instructions specify whether the objects can be merged, or if not, the candidates should be combined as distinct items of a set, or if the merge should be rejected as inconsistent. The merger makes the assumption that these comparison and

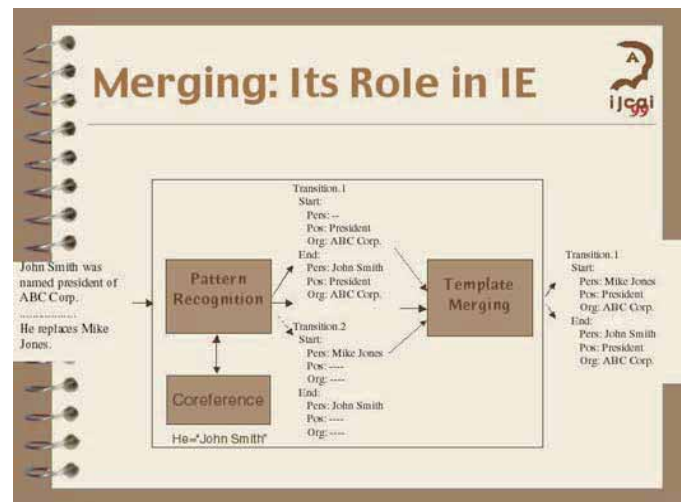
merge decisions are context independent, i.e. it is not necessary to know anything other than the values of the slots to determine whether they merge. One can also allow limited cross-slot constraints in the

form of equality and inequality constraints. Thus, in the case of the MUC-6 management succession application, one type of transition might require that the company in the initial state be distinct from that in the final state; while another, might require that the positions be the same.

With merging as with coreference, the more application-specific domain knowledge one can make effective use of, the better. Consider, in the MUC-4 terrorist activity application, a text containing the following: “The bank was the target of the attack....The lobby was completely destroyed.” In order to merge these two pieces of information about the attack, one has to recognize that lobbies are parts of banks--that is, physical parts of bank buildings, as opposed, for example, to loan departments being functional parts of banks as financial institutions--and hence can be destroyed by attacks on banks. Again, such relationships can be hand coded in some way or recourse can be had to a domain-independent knowledge base, such as Cyc.

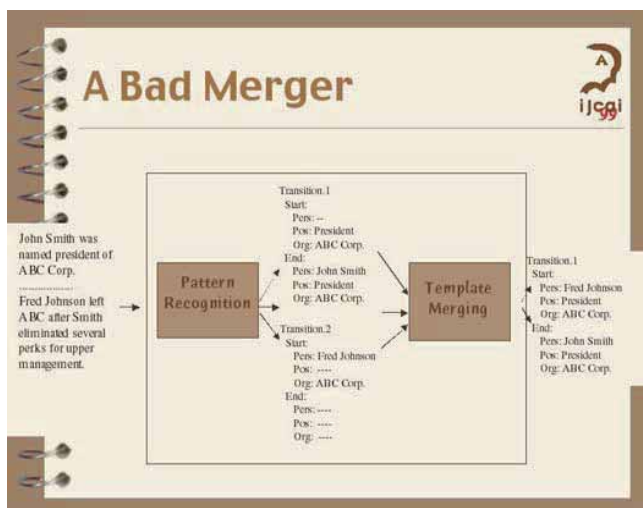
There has been very little work on training mergers or learning merging strategies and what little experience there has been, has not been happy. We shall report on experiments and analysis conducted at SRI. (This is largely the work of Andy Kehler, and more information is available at <http://www.ai.sri.com/~kebler/Papers/MDLP.ps.gz>) There are two ways in which one might attempt to identify better merging strategies. First, one could perform data analyses to identify good merging principles, handcode them, and test the results. Alternatively, one could attempt to have merging strategies be acquired by the system automatically, using some training mechanism.

The first method requires an extensive analysis of merging results. We developed detailed mechanisms for tracing merging behavior and distributed transcripts among several project participants. One would attempt to identify a variety of constraints which appeared to be extremely reliable, in particular, characteristics of templates that were almost always correlated with incorrect merges. These constraints would have to be implemented and tested. The problem is that this might well lead to no significant change in end-to-end performance as measured by F-score. Indeed, this was exactly our experience, which highlights



- ### Merging: A Knowledge Engineering Approach
- * Template slots are typed
 - * For each type, design a procedure that compares two candidates for
 - Inconsistency
 - Coreference
 - Subsumption
 - * Application-specific heuristics almost always necessary

- ### Further Considerations
- * Implicit relationships among entities
 - The bank was the target of the attack...The lobby was completely destroyed by the bomb
 - * Use of external knowledge bases
 - Cyc, WordNet
 - * Identity/Nonidentity constraints on slots
 - When it's known that 2 slots must be filled by same/different entities
 - Mr. Gates left Microsoft and took on the same position at the large potato chip manufacturer.



- ## Supervised Learning
- * Hand annotated a corpus of merges, and learned mergers using Decision Trees and Max Entropy (*Context-based features*)
 - * Both performed well on test sets, making far fewer 'mistakes' than the current merger
 - * However, no improvement in system performance on end-to-end task (as measured by F-score)
 - * **Positive results on specific, human-annotated NLP subtasks do not necessarily translate to improved performance when embedded within systems**

some of the problems with handcoding system improvements. For one, the processes of data analysis, system coding, and testing are labor intensive. One cannot try all possible alternative sets of constraints one might consider, so one can never be sure that other, unattempted constraints would not have fared better. Second, it could be that one will be misguided by the relatively small data sets that one can analyze by hand.

There are other, longer-term considerations for moving away from handcoding merging improvements. For one, the optimal merging strategy is highly dependent on the quality of the input it receives, which is constantly evolving in any realistic development setting, thus requiring continual re-experimentation. Thus, changes that improve performance at one point in system development could potentially decrease performance at another time, or vice versa. Second, a general goal of IE research is to have systems that can be trained for new applications long after the system developers are involved, which precludes experimentation by hand.

These considerations motivate research to determine if merging strategies can be learned automatically. There are several different types of learning, including supervised, unsupervised, and an area in between which one might call indirectly supervised. Researchers at SRI have explored all three. While we are unaware of any other reported research on this task, other work has addressed other MUC-style tasks. Researchers at BBN (Ralph Weischedel, TIPSTER 18-month meeting) report on learned merging strategies achieving good performance on the less complex template entity and template relation tasks in MUC-7, although no comparison with a similar hand-coded system was provided.

In a first set of experiments, the SRI researchers took the approach most commonly pursued in the computational linguistics literature, namely supervised learning. Supervised methods require a set of training data that the learning algorithm can consult in constructing its model. For our initial experiments, we ran the 100 MUC-6 training messages through FASTUS and wrote out feature signatures for the 534 merges that the system performed. The feature signatures were created by asking a set of 50 questions about the context in which the proposed merge is taking place, referencing the content of the two templates and/or the distance between the phrases from which each template was created. Some example questions are:

- ## Context-based Features
- * Articulated 50 questions a learner could ask about a merge
 - Distance between phrases from which templates were created
 - Content: how well populated, subsumption relations, degree of overlap, etc.
 - Combinations of these

(i) SUBSUMED?: true if the contents of one template completely subsume the contents of the other.

(ii) UNNAMED-REFERENCES?: true if either transition has a slot filled with an object lacking a proper name, e.g., “an employee” in the person slot. While these objects can merge with other (perhaps named) entities of the same type, in general they should not.

(iii) LESS-THAN-700-CHARS?: true if the phrases from which the templates are created are less than 700 characters apart in the text.

After the feature signatures were written, we examined the texts and manually encoded a key for each.

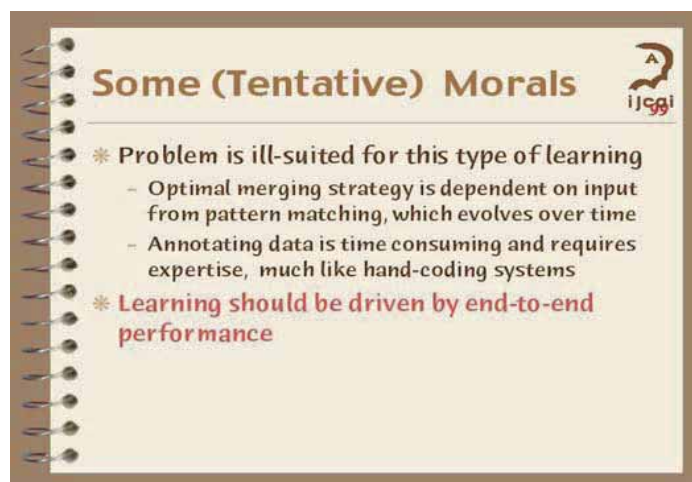
We attempted two approaches to classifying merges using this corpus as training data. The first was to grow a classification tree. At each node, the algorithm asks each question and selects the one

resulting in the purest split of the data. Entropy was used as the measure of node purity. In the second set of experiments, we used the approach to maximum entropy modeling described by Berger et al. The two possible values for each of the same 50 questions (i.e., yes or no) were paired with each of the two possible outcomes for merging (i.e., correct merge or not) to create a set of feature functions, or features for short, which were used in turn to define constraints on a probabilistic model. We used the learned maximum entropy model as a classifier by considering any merge with a probability strictly greater than 0.5 to be correct, and otherwise incorrect.



Out of the available set of questions, each approach selects only those that are most informative for the classifier being developed. In the case of the decision tree, questions are selected based on how well they split the data. In the case of maximum entropy, the algorithm approximates the gain in the model's predictiveness that would result from imposing the constraints corresponding to each of the existing inactive features, and selects the one with the highest anticipated payoff. One potential advantage of maximum entropy is that it does not split data like a decision tree does, which may prove important as training sets will necessarily be limited in their size.

We ran experiments using three different such divisions, using each example twice in a training set and once in a test set. In each case the maximum entropy classifier chose features corresponding to either 6 or 7 of the available questions, whereas the decision tree classifier asked anywhere from 7 to 14 questions to get to the deepest leaf node. In each case there was considerable, but not total, overlap in the questions utilized. The maximum entropy classifier made a total of 31 errors, in which 14 correct merges were classified as incorrect and 17 incorrect merges were classified as correct. This is compared to a total of 139



errors out of the 534 merges that the current merger made according to the annotations.

These results may appear to be positive, as it would seem that both methods found some reliable information on which to make classifications. However, our goal here was to improve end-to-end performance on the scenario template task, and thus we wanted to know how much of an impact these improved merging strategies have on that performance.

As an information gathering experiment, we applied FASTUS using the new mergers to the corpus of messages that produced the training data. We would of course expect these experiments to yield better results than when applied to unseen messages. Nonetheless, the results were humbling -- both experiments failed to improve the performance of the overall system, and in fact degraded it slightly. Generally, a point of precision was gained at the expense of a point or two of recall.

Clearly, there is a rift between what one might consider to be good performance at discriminating correct and incorrect merges based on human judgments, and the effect these decisions have on overall performance. Because the baseline FASTUS algorithm merges too liberally, using the classifiers cause many of the incorrect merges that were previously performed to be blocked, at the expense of blocking a smaller number of correct merges. Thus, it is possible that the correct merges the system performs help its end-to-end performance much more than incorrect merges hurt it. For instance, it may be that correct merges often result in well-populated templates that have a marked impact on performance, whereas incorrect merges may often add only one incorrect slot to an otherwise correct template, or even result in templates that do not pass the threshold for extractability at all. In fact, in certain circumstances incorrect merges can actually help performance, if two incorrect templates that would produce incorrect end results are unified to become one.

In any case, it should be clear that improved performance on an isolated subcomponent of an IE system, as measured against human annotations for that subcomponent, does not necessarily translate to improved end-to-end system performance. Add this to the cost of creating this annotated data — which will continually become obsolete as the upstream FASTUS modules undergo development — and it becomes clear that we need to look to other methods for learning merging mechanisms.

Several factors could influence the likelihood of a potential merge within a particular application, and it therefore seems that something tied to the application needs to guide the learning process.

When developing an IE system, one typically encodes (or is given) a moderate-size set of end-to-end development keys for a set of sample messages. These keys need to be encoded only once. We did not use these keys for supervised learning because of the difficulties in aligning the inaccurate and incomplete intermediate templates produced by the system with the (normalized) end results. However, we can use the keys to evaluate the end results of the system, and attempt to tune a merging strategy based on these evaluations. After all, it is improved end-to-end performance that we are seeking in the first place.

Thus, we consider a form of what we are calling indirectly supervised learning. We use the HAC mechanism described in the previous section, but attempt to learn the similarity metric instead of stating it explicitly. The search through the space of possible similarity metrics will be driven by end-to-end performance on a set of training messages.

We start by defining a space of similarity metrics. In a preliminary experiment, we used 7 of the questions that were used in the supervised experiments, coupled with their negations, for a total of 14 questions. These questions are assigned weights, either positive or negative, that get incorporated into a similarity metric when the question is true of a potential merge.

We used an annealing strategy to tune the weights. The algorithm begins by processing the 100-message MUC-6 development set, usually with a randomly selected initial configuration that establishes a baseline F-score. The algorithm then iterates, selecting some of the questions at random (perhaps just one, perhaps all of them) and permuting their weights by a random amount, either positive or

negative. The system is then rerun over the training set and the F-score measured. Any permutation resulting in an F-score that is strictly greater than the current baseline is adopted as the new baseline. To stay out of local maxima, a permutation leading to a decrease in performance may also be adopted. This is the annealing part -- such negative permutations are accepted with a probability that is proportional to a steadily decreasing measure of 'temperature', and inversely proportional to the magnitude of the decrement in performance. Thus, permutations that decrease performance slightly in early stages of the search are likely to be adopted, whereas permutations that decrease performance either significantly or in later stages of the search are not.

The learner was not able to leverage the available features to acquire a much better merging strategy than the one it started with. Perhaps even more surprising, however, is that there were also not lower low points — only iteration 10 achieved a score lower than 58. Because the learner was not given any bias with respect to the permutations it attempted, some of those it considered were intuitively poor (e.g., boosting the weight for phrases that are very far apart, lowering the weight for sparsely filled templates with no overlap). Thus, one might have expected certain of these to devastate performance, but none did. It seems that as long as a certain amount of merging is performed, it matters less which templates are actually merged, and in what order.

In sum, the learned mechanisms were neither significantly better nor worse than a hand-coded merging strategy. The inability to outperform the existing strategy could be attributed to several facts. We suspect that a major problem is the lack of accessible, reliable, and informative indicators for merging decisions. Unlike lower-level problems in natural language processing (NLP) in which local information appears to bear highly on the outcome, including, for instance, part-of-speech tagging and sense disambiguation none of the questions we have formulated appear to be particularly indicative of what effect a potential merge will have on system performance. This suggests that more research is needed to identify ways to access the necessary knowledge from independent sources such as existing knowledge bases, or by mining it from online text corpora using unsupervised or indirectly supervised learning techniques.

The contrast between domain- and application-specific ideas and general domain- and application-independent resources has been a recurring one in this tutorial and in the work on which it reports. Customizing an extraction system to an application has been a long and tedious process. One must determine all the ways in which the target information is expressed in a given corpus, and then think of all the plausible variants of those ways, so that appropriate regular patterns can be written. Given that



Rule Expansion

- * Divide phases into application-independent *meta-rules* and application-dependent *instances*
- * Parameterize the application-independent rules (*macro calls*)
- * Application-dependent instances consist of structured collections of parameter settings to be instantiated to produce actual rules (*macro definitions*)
- * Grammar Compilation: expand macro calls in meta-rules by substituting values of parameters defined in instances

computational linguists have been developing general grammatical systems for a long time, one might be led to believe that systems that are based on linguistically-motivated English grammars would be much easier to adapt to a new domain. But as already noted, it has been the experience of MUC developers that systems based on general grammars have not performed as well as those that have been customized in a more application-dependent manner. The reasons for this are more practical than theoretical. General grammars of English, by virtue of being general, are also highly ambiguous. One conse-

Search for Adaptability/Portability

- * Domain-phase is most application-specific phase
- * Customizing an IE system to a new application by hand takes time
- * Solutions:
 - general purpose text understanding
 - *shallow* all event coverage of a broad domain
 - various learning strategies--in support of or in place of hand-written rules

quence of this ambiguity is that a relatively long processing time is required for each sentence; this implies, in turn, a relatively long develop-test-debug cycle. Moreover, these systems have proved rather brittle when faced with the multitude of problems that arise when confronted by real-world text.

One might naturally wonder whether one can have the advantages of both worlds: tightly defined, mostly unambiguous patterns that cover precisely the ways the target information is expressed, and a way of capturing the linguistic generalizations that would make it unnecessary for an analyst to enumerate all the possible ways of expressing

it.

One approach in this direction is to localize the domain-dependence of the rules to the maximum extent possible. The idea is to divide the rules of the domain phase into domain-dependent and domain-independent portions. The domain-independent part of the domain-phase consists of a number of rules that one might characterize as parameterized macros. The rules cover various syntactic constructs at a relatively coarse granularity, the objective being to construct the appropriate predicate-argument relations for verbs that behave according to that pattern. The domain-dependent rules comprise the clusters of parameters that must be instantiated by the “macros” to produce the ac-

Text Understanding Systems

- * General purpose grammars are highly ambiguous
- * Long processing times per sentence
- * Long develop-test-debug cycle
- * Tend to be brittle when faced with real texts

Open Domain IE

- * Patterns for 200 most common event types in business news
- * Provides a shallow but broad starting point for more specific applications
- * How much can be done within the shallow and limited parsing IE paradigm?

tual rules. These domain-dependent rules specify precisely which verbs carry the domain-relevant information, and specify the domain-dependent restrictions on the arguments, as well as the semantics for the rule.

As for domain-specific rules, these, too, are centered around verbs. In a typical information extraction task, one is interested in events and relationships holding among entities, and these are usually specified by verbs. Verbs, of course, have corresponding nominalizations, so the macros should automatically instantiate nominalization patterns as well.

Supervised Learning: Domain-Phase

- * Requirements: Stiff Barriers to Entry
 - Large training corpus
 - Hand-coded keys
- * Intrinsic difficulty: task lacks local contextual features that are:
 - easy to compute/recognize
 - highly informative

Learning from Examples

- * More modest aim
- * Monitor manual filling out of templates from texts
- * Hypothesize *safe and limited* generalizations of observed cases
- * Present new rules to user/developer for inspection
- * Run system with new rules that pass inspection
- * User/developer corrects new templates

The success of this general approach depends heavily on two prerequisites: reliable coreference resolution and a well-developed parser. The coreference module is necessary because it relieves the

developer of the domain phase of the burden of anticipating all the variations that would result from pronominal and definite reference. Otherwise the developer must see to it that every rule that involves a company as subject also applies to “it,” when it refers to a company, as well as to “the company,” “the concern,” etc. The parser has the responsibility of correctly analyzing appositives and noun-phrase conjunction. This makes it possible for the domain phase to skip complements correctly. If all this work is done, then the specification of domain-specific rules can be a surprisingly simple task.

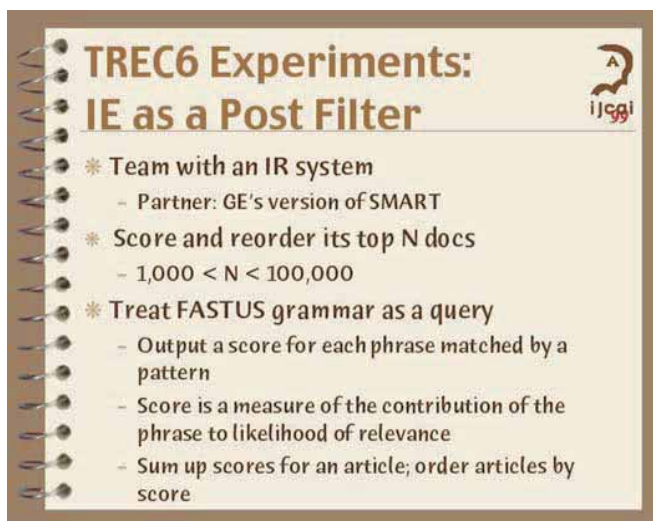
Applications Revisited

- * Information Retrieval:
 - * Highly useful, highly imprecise
- * Ad hoc queries:
 - Keyword/Boolean queries; one-off searches
- * Routing/Filtering queries:
 - Profiles representing more stable information requirements
- * Use IE grammars as a post-filter for routing queries
- * Use IE results as another weight in relevance scoring

In sum, the macro rules facility preserves the ability to write patterns that are tightly constrained to fit the particular relevant sentences of the domain, but with the additional advantage of automatically

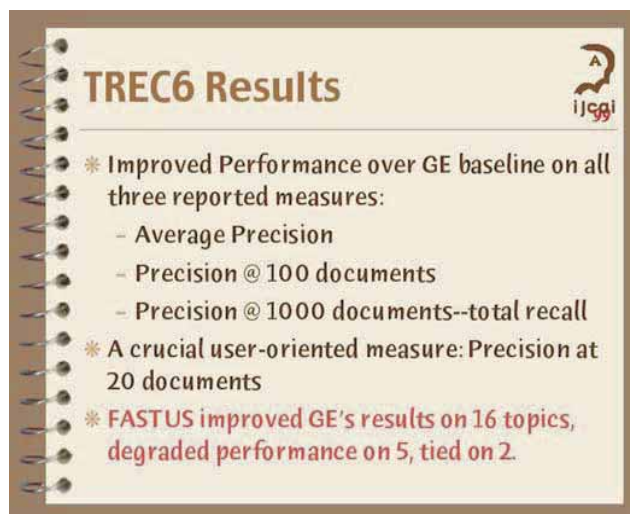
generating all of the possible linguistic variations in an error-free manner. A developer need no longer lament having failed to include a “passive” variant of a particular pattern simply because no instance occurred in the training corpus. Also, the information specified by the domain-dependent rules is relatively straightforward to provide, so that with the help of a suitable user interface, it is possible to imagine an analyst supplying the system with the information needed to customize it to a new extraction task.

The idea naturally arises, then, of a system based on learning from examples supplied by the user/developer. One begins with a system of hand-produced rules for the general domain of the application. The user is provided an interface with texts on one side, for example, and a template on the other. Thue user selects a span of text, hopefully literally a span--that is a contiguous segment and either drops and drags that text into a slot in the template or fills out the template with a seletion from a predetermined set of alternatives on the basis of the text selected. This activity is logged and moniroredby the system, thereby developing a case base of example pairs of texts and template fills. The system generates simple and safe generalizations of its existing rule-base on the basis of these observed cases. These



**TREC6 Experiments:
IE as a Post Filter**

- * Team with an IR system
 - Partner: GE's version of SMART
- * Score and reorder its top N docs
 - $1,000 < N < 100,000$
- * Treat FASTUS grammar as a query
 - Output a score for each phrase matched by a pattern
 - Score is a measure of the contribution of the phrase to likelihood of relevance
 - Sum up scores for an article; order articles by score



TREC6 Results

- * Improved Performance over GE baseline on all three reported measures:
 - Average Precision
 - Precision @ 100 documents
 - Precision @ 1000 documents--total recall
- * A crucial user-oriented measure: Precision at 20 documents
- * **FASTUS improved GE's results on 16 topics, degraded performance on 5, tied on 2.**

might be presented to the user/developer for inspection and editing. In any case, the system is then run with the new rules and the output presented to the user/developer, who then corrects and edits the new templates. This process is iterated.

As in all case- or example-based learning, there are issues as to similarity among cases and issues of dimensions and degrees of generalization to be faced. Unfortunately nothing beyond very preliminary experiments along these lines has been done.

Finally, as for applications: there has been some experimentation with using IE grammars as post-filters applied to the output of standard IR system routing queries. (A full description of this experiment is available at <http://www.ai.sri.com/pubs/papers/Bear9711:Using/document.ps.gz>.)

Researchers have pursued a variety of approaches to integrating natural language processing with document retrieval systems. The central idea in the literature is that some, perhaps shallow variant of the kind of syntactic and semantic analysis performed by general-purpose natural language processing systems can provide information useful for improving the indexing, and thus the retrieval, of documents. The work in this area has seen some success, but significant performance improvements have yet to be demonstrated. We pursued a different hypothesis, that an information extraction (IE) system can be pipelined with a document retrieval system in such a way as to improve performance on routing tasks.

The goal of a document retrieval system, as embodied in the routing task of TREC6, is to consult a large database of documents and return a subset of documents ordered by decreasing likelihood of being relevant to a particular topic. In the TREC6 routing task, a document retrieval system returns the 1000 documents it judges most likely to be relevant to a query out of a database of roughly one million documents. A system performs well if a high proportion of the articles returned, high relative to the ratio of relevant articles in the corpus, are relevant to the topic, and if the relevant articles are ranked earlier in its ordering than the irrelevant ones.

The experimental approach to using NLP techniques for IR was to adapt an IE system, SRI's FASTUS system, to enable the writing of small grammars for many topics and to use those grammars as queries to be run against the top 2000 documents for those topics, as determined by an IR system—in this case GE's version of SMART. As noted above, the output of an IR system for a given topic on the routing task is a list of the documents ordered by decreasing likelihood of relevance. The adaptation of FASTUS involved having each grammar rule that matched some segment of an article assign a score to that segment. One then summed the scores to get a total for the article.

For each of the topics for which grammars were written, FASTUS processed each article in GE's 2000 top articles for that topic and ranked them by score. The highest-scoring articles were ranked first, and importantly, in the case of ties GE's order was used.

Overall we improved the average precision very slightly over our input, from 27% to 27.3%.

FASTUS improved the average precision (non-interpolated) compared to the GE input on 17 of the 23 topics. On 12 of these the resulting average precision was above the median; in seven of these cases, we transformed above median input into an even better ordering. On one of these 7, FASTUS had the best average precision (.6322). There are several possible reasons for this success. The most likely is that there was less training data for this topic than for any of the others: 100 articles instead of (app.) 1000. Our approach may suffer less from this relative scarcity of training data than purely statistical approaches. We only wrote grammars for two of the topics that had 100 or fewer training articles, so this is still conjecture. On the other topic of this kind, we very slightly improved above-median input. Finally, the topic may just be one where the information tends to be expressed in ways that IE patterns can recognize.

On six of the topics, FASTUS lowered the average precision of the input order. A characterization of these cases is instructive. Two of these had 7 and 8 relevant documents in the training sets, respectively. When faced with that little data, we could only guess at the various ways in which relevant information might be expressed and at which patterns would recognize them. Obviously we did not make very good guesses.