Answering Definitional Questions Before They Are Asked

by

Aaron D. Fernandes

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2004

Author .....
Department of Electrical Engineering and Computer Science
August 17, 2004

Certified by .....
Boris Katz
Principal Research Scientist
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Answering Definitional Questions Before They Are Asked

by

Aaron D. Fernandes

Submitted to the Department of Electrical Engineering and Computer Science
on August 17, 2004, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Most question answering systems narrow down their search space by issuing a boolean IR query on a keyword indexed corpus. This technique often proves futile for definitional questions, because they only contain one keyword or name. Thus, an IR search for only that term is likely to produce many spurious results; documents that contain mentions of the keyword, but not in a definitional context. An alternative approach is to glean the corpus in pre-processing for syntactic constructs in which entities are defined. In this thesis, I describe a regular expression language for detecting such constructs, with the help of a part-of-speech tagger and a named-entity recognizer. My system, named CoL. ForBIN, extracts entities and their definitions, and stores them in a database. This reduces the task of definitional question answering to a simple database lookup.

Thesis Supervisor: Boris Katz
Title: Principal Research Scientist

"Col. Forbin, I know why you've come here

And I'll help you with the quest to gain the knowledge that you lack."

–Icculus[1]

Acknowledgments

I'd like to thank the following people for making this thesis possible:

- Boris Katz, for giving me this opportunity, and for all his guidance and patience throughout the years.
- Gregory Marton for providing the initial spark (and being my personal sysadmin), and Jimmy Lin for reigniting it.
- Sue Felshin, for her thoughtful editing remarks.
- Prof. Winston, who taught me all the rules for writing a thesis (so I could go ahead and break them).
- Jerome MacFarland, for showing me the Infolab way; Matthew Bilotti, for keeping me company in 822; Federico, Daniel, and Roni, for entertaining me all summer; and all my other fellow Infolabers.
- Jason Chiu, "the human annotator".
- All my friends beyond the Infolab, including, but not limited to the DU boys, the Husky Club, and the Phans. Thanks for keeping me sane.
- Dad, the Bezners, the Morgensterns (no precedence, just alphabetical!), and the rest of my family, for all their love and support. Without you, none of this is possible.

[1]Oh yeah, I'd also like to thank the band that provided these words of wisdom, and a world of enlightenment. This *has* all been wonderful, and you will be missed.

Dedication

In loving memory of Mom, Sitoo, and Grandpa, who never doubted me even when I did. This one's for you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Any web surfer who has issued a search query for a single word or name can relate to the problem of "junk results". These are documents that contain several mentions of the query term, but do not provide relevant information to the target entity. Question answering (QA) systems face a similar dilemma when asked a question that contains only one keyword once stopwords are stripped away. This is because most QA systems use a keyword-based information retrieval (IR) engine to retrieve a manageable document set for further processing. If the document set is cluttered with junk results, the old GIGO axiom (Garbage In, Garbage Out) often applies.

The goal of this thesis is to answer definitional questions without IR in the traditional sense. I will describe an alternative system that extracts informational nuggets from the document corpus in pre-processing, in the form of entity/definition pairs, and stores them for use at runtime. The system, called CoL. ForBIN,[1] collects these pairs by searching for syntactic constructs in which entities are typically defined. CoL. ForBIN is designed to operate on flat text, so its "grammar" is actually a regular language that tries to minimally encapsulate a large subset of English grammar. I will describe my implementation of this system, and demonstrate its near state-of-the-art performance in definitional QA, with speed equivalent to that of a database lookup.

[1] CoL. ForBIN, named after the legendary knowledge seeker of Gamehengian mythology, abbreviates "Colonel Looks For Buried Informational Nuggets

1.1 Question Answering Overview

The Text REtrieval Conference (TREC) Question Answering Track challenges participants to develop systems that retrieve answers from a corpus of news articles in response to natural language questions [16, 18]. Traditionally, systems designed for this track have employed a pipelined architecture [17]. First, a document retrieval module narrows the corpus down to a manageable set of documents. Then, based on these documents, a passage retriever uses some heuristics to extract a set of passages that it deems likely to contain a valid answer. Finally, an answer extractor uses finer-grained heuristics to extract an exact answer from the passages.

Pipelined architectures have proven reasonably successful in the field of QA. In any pipelined architecture, the first element is most crucial. Errors made in the first stages of processing can propagate down the pipe, thus preventing any subsequent modules from achieving their goals. Document retrieval is no exception. Most often, the document retriever is a simple information retrieval (IR) engine that performs a boolean keyword search over the corpus. In the simplest case, the search query is constructed from the question by removing stopwords, and separating the remaining terms with an AND operator. Stopwords include question words ("who", "what", "where", etc.), forms of the verb "to be", and function words that occur frequently in the corpus (e.g. prepositions and articles). This technique is generally sufficient for providing a set of relevant documents to support a generic "factoid" question.[2] However, it is prone to failure when the number of keywords is either too large or too small. If the number is too large, then IR can suffer from poor recall; it will be unable to find enough documents containing all the keywords. If the number is too small, it can suffer from poor precision; it will produce a large number of documents which may or may not be relevant to the question.

Another problem with producing too many documents is that it does not really narrow down our search for an answer. The motivation for doing IR first is that passage retrieval and answer extraction are too computationally expensive to perform

[2]For example, "Who was the first astronaut to do a spacewalk?"

on the entire corpus at runtime. But if an IR query is too general, it may return a document set that is tantamount to the entire corpus, in that it will still take days for the finer-grained modules to do their work. Most systems will attempt to avoid this eventuality by placing a cutoff on the number of documents to be passed along to the passage retriever. This effectually turns a precision problem into a recall problem, because the IR engine can only guess at what the best n documents are among the ones it has found. Usually, it will pick the ones that mention the keywords most frequently, but these are by no means guaranteed to contain the answer we are looking for.

1.2 Definitional Questions

One subclass of natural language questions that can exhibit the phenomenon discussed above is the definitional question. This is a relatively open-ended question that simply asks "What is X?" or "Who is Y?" When faced with such a question, the goal is to extract from the corpus any and all nuggets of information pertaining to the target (denoted here by the variables X or Y). The problem with using a traditional IR approach to answer these questions is that they only contain one keyword.

Take, for example, the question "Who is George W. Bush?" First, it is important to see that, while it may seem as if "George", "W.", and "Bush" are three separate keywords, we really want to treat the entire name "George W. Bush" as one. Otherwise, we would retrieve headlines like: "Bush to Host George Harrison Tribute in W. Va." Next we must realize that a search on "George W. Bush" will produce any document that mentions his name, whether or not in a definitional context. For example, a search on "George W. Bush" in the TREC 12 corpus yields almost 21,000 documents; far too many for any passage retriever to deal with efficiently. We can tell the document retriever to stop looking after it finds the 1000 best documents, but "best" in IR terms means "most hits", not "best definitions". In fact, a quick survey of the top 20 documents returned by the Lucene IR engine [1] in response to the query "George W. Bush" reveals 7 documents that simply list election results, and 8 others

that are equally irrelevant to the definitional question, containing passages like:

> Guests include ... Karen Hughes, press secretary for George W. Bush, ...

1.3 Syntactic Pattern Matching

As mentioned by Katz et al. [9], among others [4, 12], we can avoid the recall problem by precompiling bits of knowledge from the corpus with a technique called syntactic pattern matching. The intuition behind pattern matching is that chunks of text containing a definition usually manifest themselves in one of a few commonly occurring syntactic patterns. Some examples are illustrated below, along with instances from the AQUAINT corpus. In the following examples, D represents a definitional phrase for the entity, X.

1. **copular**: X {is,was,are,were} {a,an,the} D
   example: Cerebral palsy is a term for neurological disorders

2. **appositive**: X, {a,an,the} D,
   example: Vigevano, a town near Milan,

3. **or**: X(s)(,) or D
   example: hypertension, or high blood pressure,

4. **such as**: D, such as X
   example: computer languages, such as Java or C++

5. **occupation**: D X
   example: President of Gambia Yahya Jammeh

But how does one define X and D? In other words, what type of phrase makes a good target entity, and what makes a good definition? Better yet, how do we recognize these phrases in flat text? The Katz et al. system used a part-of-speech (POS) tagger [3] to preprocess the corpus, and a simple regular expression library to recognize certain POS sequences as noun phrases, and thus good targets. The

problem with this approach is that most good targets turn out to be named entities, and these can be difficult to detect based on POS tags alone. Here are some examples where the Katz et al. patterns misidentified a named entity target:

1. **Input**: Squier, Knapp, Ochs & Dunn, a political consulting firm in Washington,
   Target: Dunn

2. **Input**: "Y2K: The Movie" is an "irresponsible" spark ...
   Target: The Movie"

3. **Input**: Though Gore was the clear winner ...
   Target: Though Gore

4. **Input**: After joining the UCLA law faculty Mellinkoff wrote "The ... "
   Target: the UCLA law faculty Mellinkoff

We can't afford to make these types of errors, because the targets we extract become the keys of our database index, and therefore must match the targets extracted from the questions. To alleviate this problem, I've added a named entity recognizer to the preprocessing pipeline and merged its results with the POS tags. I've also designed a more comprehensive regular expression grammar that detects a wider range of noun phrases.

Another shortcoming of the Katz et al. patterns is their reliance on adjacency. For a given target, they will match at most one adjacent nugget, allowing no intervening text, and ignoring any text that follows. This can undermine our goal of maximizing recall, since many newswire sentences are densely packed with informational nuggets pertaining to a given entity. For example, consider the following sentence from the AQUAINT corpus:

> Holt, 50, a physicist, was the assistant director of Princeton Plasma Physics Laboratory.

Here are three good nuggets of information about Holt: he's 50 years old, he's a physicist, and he was the assistant director of Princeton Plasma Physics Laboratory.

But the Katz et al. patterns won't match any of these, because they don't know what to do with the number 50. Even if Holt's age wasn't mentioned, the patterns would still only match the adjacent nugget, "a physicist". CoL. ForBIN on the other hand, employs a completely different matching algorithm. For a given target, it tries to match each pattern in succession, taking as input the text left unconsumed by previous patterns. It also knows how to deal with intervening expressions, such as ages, affiliations, and time expressions.

Chapter 2

Related Work

Syntactic pattern matching is an evolutionary descendent of surface pattern matching techniques that were first described several years ago. Surface patterns attempt to match exact words or character sequences in text, whereas syntactic patterns rely also on syntactic attributes, such as part-of-speech tags. This chapter presents a brief overview of systems that have employed pattern matching techniques in the past three TREC competitions, with special emphasis on definitional questions.

2.1 TREC 10 & 11

The TREC 10 competition took place in 2001, followed by TREC 11 in 2002. These were the last two competitions to take place before the rule change that separated questions into three categories: factoid, definition, and list. So, at the time, answers to each type of question were judged by the same standards. The crucial difference between TRECs 10 and 11 was the allowed answer length. TREC 10 entrants were free to return a 50-byte string, which would be judged as correct if it contained the question's answer in full. In TREC 11, however, entrants were required to return an exact answer, which would be judged incorrect if it contained any extraneous information. This remains the standard for factoid questions today. Below, I describe some TREC 10 and 11 systems that employed surface pattern matching techniques.

2.1.1 InsightSoft-M

Soubbotin and Soubbotin of InsightSoft-M [13] describe a story generator that, given a query topic, searches a document set for relevant snippets of text, and attempts to assemble them in a coherent manner. The system, called CrossReader, deems a passage relevant if it matches one of several predefined surface patterns. In 2001, Soubbotin and Soubbotin [14] built a TREC-10 system based on the CrossReader philosophy, treating the question focus as the query topic, and running the matcher at passage retrieval time. Their system operated on pure text strings, attempting to circumvent any notion of language understanding. Thus, the patterns were constructed from simple atomic units, such as letters, digits, and punctuation marks. From these, however, they were able to represent higher syntactic notions, such as the appositive pattern described earlier.

Their entry performed reasonbly well in the competition, achieving a mean reciprocal rank (MRR) of 0.676. They answered 289/472 questions correctly, 193 of which matched one of their patterns. (The rest simply contained question keywords, or other minor indications of a correct response). Recall, however, that the answer guidelines were much more lenient at the time. This allowed the Soubbotin system to return inexact responses without being penalized. Moreover, since they performed their matching in passage retrieval, after IR, they were faced with the recall problem described in the previous chapter.

For TREC 11, Soubbotin and Soubbotin [15] built on their system from the previous year. The key difference was the addition of what they call "Complex Patterns". In addition to matching on simple string tokens, they compiled lists of words corresponding to semantic categories. A reference to one of these lists could then be used as an atomic pattern element. For example, whereas the previous year's system might detect a person as "C C", where "C" denotes a capitalized word, the newer version might instead use "F L", where "F" denotes a list of known first names, and "L" a list of last names. In doing so, they chose to sacrifice recall for precision, which was reflective of the new "exact answer" requirement. However, one can only go so far

with this approach before the lists become large and unwieldy.

2.1.2 ISI's TextMap

Hermjakob et al. [5] of the Information Sciences Institute also employed a version of pattern matching in their TREC 11 system. However, instead of using a collection of predefined patterns built out of string tokens and word lists, they constructed patterns on the fly as reformulations of the question. Their system, called TextMap, would generate up to 30 reformulations of each question, and look for an exact string match in the document collection. Of course, each reformulation, or pattern, would contain a gap reserved for the question's answer. For example, the question "Who invented the telephone?" would be reformulated as "X invented the telephone", "X was the inventor of the telephone", "X's invention of the telephone", and so on, where the substring matching "X" would be returned as an answer.

One drawback to this approach is that each reformulation must be explicitly encoded as a rule. For example, the reformulation "X was the inventor of the telephone" comes from the rule:

```
:anchor-pattern ''PERSON_1 invented SOMETHING_2.''
:is-equivalent-to ''PERSON_1 was the inventor of SOMETHING_2.''
```

This limits the generality of the approach, since the semantics are hand-coded, but is sufficient for reformulating definitional questions, which come in a limited variety of flavors. Another shortcoming of this system is its speed. The TextMap philosophy is to exploit large data sets, such as the World Wide Web, relying on the high probability that its auto-generated patterns will find a match. However, since the patterns are not generated until runtime, the QA turnover is very slow (approximately one question every five minutes).

2.2 Restricting the Domain to Definitional Questions

By 2003, most TREC participants were already dividing the QA task among different subsystems designed to handle the fundamentally different types of questions being asked. The distinction was made official in the TREC 12 guidelines, which defined separate criteria for judging answers to factoid, list, and definition questions, and presented rankings on a per-category basis. As mentioned in the previous section, the desired response to a definitional question is a collection of informational nuggets pertaining to the question topic. Below, I describe some systems tailored specifically around definitional questions.

2.2.1 Fleischman et al.

Fleischman et al. [4], also of ISI, developed an alternative to TextMap with definitional questions in mind. As noted above, definitional questions have only a limited number of reformulations (Fleischman et al. called them concept–instance relations), so they decided to mine the corpus for these at pre-compile time. But, instead of performing an exact string match (which would be impossible without knowing the question target in advance), they preprocessed the text with a part-of-speech tagger, and searched for tag sequences that matched one of two well-defined concept–instance relations. These relations were two types of noun–noun modification, which they called CN/PN (common noun/proper noun) and APOS (appositions), and which I presented as occupation and appositive in the previous chapter. This technique, along with a machine-learned filtering mechanism, yielded results that were fast (three orders of magnitude faster than TextMap) and highly precise. The only problem was poor recall, since many other good concept–instance relations, such as copulae and "such as", were ignored. This is the problem Katz et al. set out to fix.

2.2.2 MIT CSAIL's Intrasentential Definition Extractor

The MIT CSAIL team at TREC 12 employed a three-pronged approach to answering definitional questions. One of these was a direct offshoot of the ISI system described above. The other two were simply backoff mechanisms: dictionary lookup, and keyword search over the corpus [9]. The careful reader will notice that the author was a co-author of this publication, and might correctly surmise that my work on that system gave rise to this thesis. My intuition was as follows: why stop at CN/PN and APOS when there are so many other concept–instance relations (which I shall henceforth refer to as "target/nugget pairs") to glean? This carries an implicit assumption that recall is more important than precision, because we only get one shot to glean all the target/nugget pairs, and in doing so at pre-compile time, we can afford to spend time later on answer filtering. With that in mind, using the sample definition questions and answers provided by NIST before the competition, I manually scoured the AQUAINT corpus for contexts in which these pairs occurred. This gave rise to 15 new patterns that could account for approximately 90 percent of target/nugget recall. I mentioned some of this system's drawbacks in the previous chapter, and will elaborate more on how I addressed them in the next.

2.2.3 State of the Art Alternatives: BBN and Yang

The system that scored highest in the definitional category last year was the group from BBN Technologies, with an F score of 0.555. Xu et. al managed to do reasonably well on the definitional task using a pipelined architecture [19]. First, they identified the question target, to be used as the sole keyword or phrase in an IR query. As an intermediate phase, before passage retrieval, they selected only those sentences from the resulting document set that contained a mention of the question target. Next, they employed a variety of lingustic processing and information extraction techniques to extract *kernel facts*, or information nuggets, about the target. Among the techniques they used were surface pattern matching and semantic role approximation. Finally, the kernel facts were ranked according to how well they matched the "profile" of the

question. The question profile was developed in one of three ways. First, the system would check for an existing definition among several sources, including WordNet and the Merriam-Webster dictionary. If none could be found, the system would back off in one of two ways. For a *who* question, it would derive the centroid of 17,000 short biographies from www.s9.com, essentially creating a generic biographic record. For *what* questions it would take the centroid of all the kernel facts about that target, assuming that the most frequently occurring facts are the relevant ones. The highest scoring facts would be returned as an answer.

Xu et al. noted that when a question received bad F score, the failure was far more likely due to bad recall than bad precision. They even went on to speculate that, assuming perfect recall, their system would have achieved an F score of 0.797. The recall problem is specifically what I intend to avoid by abandoning the pipeline.

The group that finished second in the definitional category, with an F score of 0.319, was Yang et al. from the National University of Singapore. They applied pattern matching, but only as a boosting heuristic after two iterations of document retrieval. The first iteration involves a simple keyword search, and the second one filters out documents that do not contain all the bigrams in the question target. Then the remaining sentences are grouped into positive and negative sets, based on whether or not they contained any part of the search target. Finally, they used frequency metrics to determine the importance of each word pertaining to the target, and tried to select the most "important" sentences as their answer. (Sentences that match certain surface patterns, like the appositive, have their importance rating boosted.)

This approach performed reasonably well, even with two rounds of document retrieval, because it found an alternative solution to the recall problem: they placed no restriction on the size of the resulting document set. But they were still able to reduce it to a manageable size by distilling out documents that didn't contain all the bigrams in the target. It would, however, be intersting to see what "manageable" translates to in a quantitative sense, in terms of the question–answer turnover rate. One advantage that precompiling nuggets will always have over runtime techniques is that of runtime speed.

Chapter 3

CoL. ForBIN: A Pattern Grammar

The Infolab group at MIT CSAIL, of which I am member, has a "strength in numbers" philosophy regarding question answering. This is based on the intuition that, when a user asks a question, she expects a speedy response, and she prefers an exact answer with just enough text to provide relevant contextual information [11]. The first requirement, speed, is why we avoid costly runtime search techniques. The second requirement, precision, is why we prefer to return short strings of text, as opposed to an entire document collection. So, our solution to both of these problems is to accumulate offline vast amounts of data from diverse knowledge sources, and store it in a common representational format: the <**constituent relation constituent**> triple [6, 7, 8]. Then we use natural language processing techniques to translate questions into methods that retrieve and assemble knowledge from our collection.

So how does one go about accumulating this vast collection of data? In the past, we would attempt to identify structured data repositories on the web, and write scripts that glean data from them en masse or at runtime[1], depending on the stability of the source. Of course, finding and annotating such sources requires human labor, and maintenance becomes a major issue as sites change and the web continues to grow. As an alternative, Jimmy Lin wrote a program that automatically extracts relations from semi-structured data sources such as encyclopedia entries, but with limited success [10, 9].

[1]Note that this is not a search, because the scripts know exactly where to look.

In the sections below, I describe a middle-of-the-road solution: a program that performs automatic relation extraction, but only for a small subset of relations (most notably, the "is-a" relation). I claim that this subset alone is sufficient for providing answer support to most definitional questions.

3.1 A Syntactic Regular Expression Library

I have referred to my patterns as syntactic, which implies that they follow the rules of some grammar. However, the pattern-matching infrastructure is designed to operate on flat text, so my "grammar" is actually a regular language that attempts to minimally encapsulate a subset of the English language grammar. In this section, I will describe the regular language in detail, providing examples of strings it recognizes, and counterexamples of ones it does not.

3.1.1 Atomic Elements

At the lowest level of abstraction[2] are three atomic elements: words, parts of speech, and named entities. All expressions in the language are constructed by combining these atomic elements via the concatenation, union, and star operators.

A word is a specific string token,[3] or set of string tokens, with an associated part-of-speech tag. The POS tag can be applied to the whole set, or to individual set members, and need not be specified. Here are some examples from the lexicon:

```
$of  = /of_IN/; # the word ``of''
$the = /the_DT/; # the word ``the''
$do  = /(do_VBP|does_VBZ|did_VBD)/; # forms of ``to do''
$verbs_trans = /(moved|went|returned|came)_[A-Z]+/;
```

The $do element looks for POS tags specific to each conjugation, whereas $verbs_trans applies the generic POS tag[4] to the entire set.

[2] ignoring the Perl regular expression syntax upon which this language is built
[3] as defined by the Penn TreeBank tokenization rules
[4] matching any string of capitalized letters, and thus any POS tag

A part-of-speech element is any token marked with a given POS tag, or set of POS tags. Here are some part-of-speech elements:

```
our $det    = qr{ \S+_DT }ox;
our $noun   = qr{ \S+_NNP?S }ox;
our $verb   = qr{ \S+_MD|VB[DGNPZ]? };
our $comma  = qr{ \S+_, }ox;
```

Note that punctuation marks fall into this category as well, since they are considered by the Brill POS tagger as separate tokens.

Similar to a part-of-speech element is the named entity. A named entity, as its name implies, is a string that refers to a discourse entity by name. These are identified in preprocessing by BBN's IdentiFinder program [2], and are wrapped in double angle brackets to distinguish them from other tokens. Named entities come in several varieties, such as persons, organizations, and facilities, and this meta-data is recorded by IdentFinder as well. Here are some examples:

```
our $named_entity = qr{ <<NE [^>]* >> }ox; # any named entity
our $person = qr{ <<NE-PER [^>]* >> }ox; # any named person
```

3.1.2 Detecting Phrase Boundaries

TREC answers are penalized if they contain too much extraneous information. Likewise, the Infolab philosophy is to answer questions in a succinct and precise manner. For these two reasons, it is important that my patterns detect proper phrase boundaries. But how does one use a regular language to define a noun phrase, for example, when its English language definition is self-referential?[5]

First, we must understand that, while the English language provides support for unlimited recursion, we seldom see more than a few iterations manifest themselves in text. So while the following may be a valid English sentence: "I saw the apple in a bag on the table in the middle of the room with a door to the patio by the yard," it

[5]A phrase is not defined per se, but rather stated in terms of context-free rules. Two of these rules for a noun phrase are NP → NP PP, and NP → NP POS NP, both of which are self-referential.

places high demands on the attention of the reader, and is thus unlikely to be seen in a newspaper article. On the other hand, we can reasonably expect to see a sentence like, "I saw the apple in a bag on the table." So, we can pick a cutoff on the number i of iterations we expect to see (3, for example), and build i levels of phrase patterns with increasing levels of complexity.[6] This is the philosophy behind the design of my pattern grammar, which I describe here.

At the lowest level are proper noun phrases and noun phrase modifiers:

Name	**Pattern**	**Examples**
properNP	(the) {adj,noun}* NE†	Greenpeace the United States the honorable Judge Smails
properDP	the {adj,noun}* NE	the United States the honorable Judge Wapner (same as above, but requires "the")
NPmods	{adj,noun,gerund}*	seven tasty melting ice cream

† This is shorthand for the named_entity element described above.

From these, we can construct modified nouns, then determiner phrase specifiers and modifiers:

[6]For example, the lowest-level NP pattern would match "the table", and the lowest PP would match "on the table". The next higher level would use the lower-level patterns to match "a bag on the table", "in a bag on the table" and so on.

Name	Pattern	Examples
modNoun	(NPmods) noun	seven tasty ice cream cones
possessive	{ his, her, etc. }	his
	{ properNP, time } 's	Aaron's
		last Sunday's
	(det) modNoun 's	the handsome man's
determiner	set union: { a, the, possessive }	
DPMods	(predet) det (number)	all of the seven
		all seven of the
		all seven of last year's

Finally, we can construct a common noun phrase or determiner phrase, and a single unifying low-level noun phrase:

Name	Pattern	Examples
commonNP	(DPmods) modNoun	all seven of the tasty ice cream cones
commonDP	DPmods modNoun	five of the seven tasty ice cream cones
simpleNP	set union: {properNP, commonNP, pronoun }	

Now, with the simple addition of a preposition, we have a low level prepositional phrase. This we can attach to the simple noun phrase to create a second noun phrase of slightly higher complexity:

Name	Pattern	Examples
simplePP	prep simpleNP	like him
		about Aaron
		of the seven tasty ice cream cones
commonNP2	commonNP (simplePP)	guys like him
		the funny story about Aaron
		some of the flavors of the seven ...
secondNP	set union: {properNP, commonNP2, pronoun }	

This gives us all the pieces we need to construct a verb phrase, so let's start building one up. We'll need this for use in a relative clause modifier for our noun phrase. First, we construct some verb modifiers, and a modified verb:

Name	Pattern	Examples
Vmods	adv ((,) (and) adv)*	quickly quickly and quietly deftly, quickly, and quietly
modVerb	(auxVerbs) (Vmods) verb (particle) (Vmods)	could have been being eaten has quickly gone went quietly

And now we *carefully* construct a simple verb argument structure. We have the secondNP at our disposal now, but we elect to use the simpleNP instead. The reasoning behind this is that secondNP consumes a prepositional phrase, but since we're in a verb phrase, we'd prefer that the PP attach to the verb.[7] Also, bearing in mind that we're working up to a relative clause, let's construct a PP-extracted version of the argument string as well:

Name	Pattern	Examples
simpleVargs	{ NP ({NP, PP}), PP (PP) }	the letter the girl the letter the letter to the girl to the store with me to the store
PPextVargs	(NP) (PP) prep	to the letter to to the bar with the letter from the man to

[7]We want the constituent phrases to be as non-greedy as possible, because we don't want them to steal attachments from the higher phrases

Now we can piece them together into an infinitival phrase (and its PP-extracted counterpart), and a reduced relative clause:

Name	Pattern	Examples
InfP	(secondNP) to modVerb (simpleVargs)	to go to the bar her to come to the bar the girl at the bar to come to the club
PPextInfP	to modVerbs PPextVargs	to send the letter to to go to the bar with
reducedRel	(Vmods) participle (Vmods) (simpleVargs)	riding a bicycle eaten quickly by the child

This opens the door to capturing a new level of more complex noun phrases, determiner phrases, and prepositional phrases. Then we can use this higher PP to construct another NP of even higher complexity:

Name	Pattern	Examples
thirdNP	secondNP (reducedRel)	books about language written by Chomsky
thirdDP	secondDP‡ (reducedRel)	some of the candy eaten by the small child
higherPP	prep thirdNP	in the books about language written by Chomsky
fourthNP	commonNP (higherPP)	the first three chapters in the book about language written by Chomsky

‡ This is almost the same as secondNP, but requires a determiner. You'll see later why I'm still making the distinction between noun phrases and determiner phrases, even though determiner phrases are captured by the noun phrase patterns as well.

The commonNP part (above) captures the noun head and pre-modifiers, and the

higherPP part captures all the post-modifiers. This follows the intuition that any PP attachments will come before any reduced relative modifiers, because otherwise, the PPs will prefer to attach themselves somewhere in the embedded verb phrase. Now we can construct some more complex verb phrases for use in a full relative clause:

Name	**Pattern**	**Examples**
Vargs2	(secondNP) (thirdNP) higherPP*	me me the letter me the letter on the table him to an early grave to an early grave me the letter on the table in the room by the door
simpleVP	modVerb {InfP, Vargs2}	sent him to an early grave sent to an early grave showed me the letter on the table in the room by the door

Again, it may appear that I have arbitrarily chosen to use secondNP and thirdNP here, but there is a reason for these selections. I chose secondNP for the first position because thirdNP consumes a reduced relative, which could steal arguments from the main verb. Perhaps this is best illustrated with an example. Consider the sentence: "I showed the woman riding a bicycle the picture of a man". If we allow the first argument to "saw" to be a reduced relative, the reducedRel pattern greedily captures "the woman riding the bicycle the picture". To the regular language, this seems like a perfectly reasonable reduced relative, along the same lines as "the man reading his daughter a story".[8]

What about the thirdNP in the second argument position? Why not use the fourth one there? Well, all the fourth one gives us is an additional prepositional

[8]For the sake of speed, CoL. ForBIN eschews any notion of a feature grammar. Otherwise, it would be able to distinguish the above examples by noting that "read" is a dative verb, while "ride" is not.

phrase argument, but this will also get captured as part of higherPP*. In fact, higherPP* allows us to "cheat" and get infinite recursion, as long as all the trailing noun phrases match thirdNP.

As another example, consider again the sentence: "I saw the apple in a bag on the table in the middle of the room with a door to the patio by the yard." Here there are only one or two constituents attached to the verb (depending on whether you attach "in a bag ..." to "saw" or "the apple"), but the simpleVP pattern will see four[9] and capture them all. This is another way to fake recursion with a regular language.

Now, we're almost done. All we have left to do is construct a relative clause, using the simpleVP above, and attach it to our noun phrase:

Name	**Pattern**	**Examples**
fullRel	{comp, (comp) thirdDP} simpleVP	you need t to look for mold that t sent him to an early grave that Jon sent t to Paris
RC	Set union: {fullRel, reducedRel}	
highestNP	fourthNP {RC, InfP}	tools you need to look for mold words to live by the package that Jon sent to Paris the illness that sent him to an early grave the woman in the picture riding a bicycle who everyone thought was pretty

A few notational comments, regarding the table above: t is a placeholder for the extracted subject or object, comp is a complementizer, such as "who" or "that", and InfP includes the PP-extracted version as well.

There is another intricate detail here, involving the use of DP versus NP. Some-

[9] "the apple in a bag", "on the table in the middle", "of the room with a door", "to the patio by the yard"

thing very ugly happens here if we fail to make this distinction. If we allow the subject of a relative clause to be any noun phrase, we run into problems whenever there is noun–noun modification. Consider the following two examples:

1. I received [the letter John sent me].

2. [Senator John F. Kerry sent me a letter].

The human reader can easily distinguish the first bracketed segment as a noun phrase, and the second one as a sentence. However, these constructions look identical to the RC pattern if we replace thirdDP with some NP. In other words, the second example would be interpretted as "(the) Senator John F. (that) Kerry sent me a letter"[10]. This applies to any sentence with a noun-modified noun phrase as its subject. To avoid this, we require that the subject of a relative clause be a determiner phrase. Of course, this prevents us from capturing noun phrases like "the man women love to hate", but it is rare that we see this formation without a complementizer.

With highestNP as the most complex atomic element, I now have a regular language specification that recognizes a large percentage of noun phrases and verb phrases that one might expect to find in newspaper text.

3.2 Building Patterns

The rationale behind having a library of words and phrases, like the one described above, is to abstract away the details of regular expression syntax from the process of developing patterns that extract relations from text. In this section, I describe nineteen such patterns, treating all items described in the previous section as atomic elements. In defining a pattern, I enclose the target entity in angle brackets, and its defining characteristic in square brackets. I also present positive and false positive instances of each pattern from the AQUAINT corpus.

[10]The human reader would also note that this is syntactically incorrect, but I have chosen to stick with one generic verb argument structure pattern for economy considerations.

3.2.1 Age

The age pattern is designed to capture a person's age when it manifests itself in text as ⟨*person*⟩, [*age*],. In relational terms, this corresponds to the *person* is-of-age *age* relation.

Pattern	⟨*person*⟩ , [number],
Instances	pop sensation ⟨*Christina Aguilera*⟩, [19], ⟨*Kelly McConnell*⟩, [13], of Paris, Texas
False Positives	N/A

This pattern is very precise. I'm sure it generates some false positives, but I was unable to find any in a subset of 1000 AQUAINT documents.

3.2.2 Affiliation

The affiliation pattern captures a person's affiliation or place of origin, when stated as ⟨*person*⟩ of [*affiliation*]. The relation captured here is ⟨*person*⟩ is-a-constituent-of [*affiliation*].

Pattern	⟨*person*⟩ of [{organization,location}]
Instances	⟨*Jim Crocker*⟩ of [Johns Hopkins University] ⟨*Bert Ely*⟩ of [Alexandria, Va.]
False Positives	the ⟨*Homer Simpson*⟩ of [the NFL], Terry Bradshaw,

This pattern is also very precise, but can be fooled by idiomatic expressions like the one above.

3.2.3 Also Known As (AKA)

The "aka" pattern is not as restrictive as its name implies. It captures several variants of the "[Y], also know as ⟨X⟩" construction, all of which express the ⟨X⟩ is-a [Y] relation. For entity-second patterns, such as this one, I found that the simplest noun

phrase pattern was generally sufficient for capturing the target entity.[11]

Pattern	[highestNP] (,) be {(more) commonly, also, often, sometimes} {known, referred to} as ⟨*simpleNP*⟩
Instances	The [detainees] are known as ⟨*bidoun*⟩ [the pro-independence Albanian militants], known as ⟨*Kosovo Liberation Army*⟩ [MDMA], commonly known as ⟨*ecstasy*⟩
False Positives	[This method], known as ⟨*surveillance*⟩, [This process] is known as ⟨*cryotherapy*⟩.

Most false positives occur when the definitional term Y is too broad or vague.

3.2.4 Also called

Similar to the AKA pattern is "also called". This is another embodiment of the "is-a" relation where the target entity follows the defining phrase.

Pattern	[highestNP] (,) ({also, sometimes, often}) called ⟨*simpleNP*⟩
Instances	based on [a client-server reporting system] called ⟨*Abacus*⟩ [Maoist rebels], also called ⟨*Naxalites*⟩ [the Internet business], called ⟨*Thaicom Direct*⟩
False Positives	[The group], called ⟨*Central European Initiative*⟩, Annan, in [a separate statement], called ⟨*Pakistan*⟩'s action

In addition to the overgenerality problem, which is common to all entity-second patterns, this pattern can also suffer when the past-tense verb "called" is mistagged as a participle.

[11]Bear in mind that the targets should roughly correspond with targets we'd expect to see in definitional questions. If the target has too many keywords, it's probably best to treat the question as a factoid.

3.2.5 Named

Also along these same lines is the "named" pattern. This one looks like "also called", although it excludes the optional adverb. As its name implies, the target is usually a named entity.

Pattern	[highestNP] (,) named ⟨*simpleNP*⟩
Instances	[a young London-born playwright] named ⟨*Martin McDonagh*⟩ [The unmanned probe], named ⟨*Nozomi*⟩
False Positives	[Deloitte and Touche], named ⟨*trustees*⟩ in the bankruptcy [the complaint] named ⟨*about six former state officials*⟩

The named pattern shares "also called"'s susceptibility to part-of-speech tagging errors, and also generates false positives when named is used in the sense meaning "cited as", or "specified as".

3.2.6 Like

The last two entity-second patterns, "like" and "such as", differ slightly from others. Instead of settling on a single target entity, these patterns will look for multiple targets in a list construction. This follows the intuition that a concept is often demonstrated by listing some of its hyponyms. If a list construction is found in target position, each entity in the list enters a separate "is-a" relation with the common hypernym.

Pattern	[highestNP] (,) like simpleNP(s)
Instances	[a prestigious sporting event] like ⟨*the SEA Games*⟩ [cities with huge populations], like ⟨*Hong Kong*⟩, ⟨*Shenzhen*⟩ and ⟨*Guangzhou*⟩ [Muslim countries] like ⟨*Jordan*⟩, ⟨*Egypt*⟩ and ⟨*Turkey*⟩
False Positives	[a very big noise] like ⟨*lightning*⟩ In [Greece], like ⟨*many countries*⟩, ⟨*parliament members*⟩ have protection

This pattern suffers from multiple types of false positives, due to the different senses of the word "like". For example, the pattern often picks up instances where like is used to convey the meaning "similar to", instead of "such as", as desired.

3.2.7 Such As

The "such as" pattern is nearly identical to "like", and is only kept separate for the sake of simplifying the underlying regular expressions.

Pattern	[highestNP] (,) such as simpleNP(s)
Instances	[donor agencies], such as ⟨*the International Monetary Fund*⟩ [Middle Eastern countries now seeking nuclear capability], such as ⟨*Iran*⟩ and ⟨*Iraq*⟩, [stinging insects] such as ⟨*bees*⟩, ⟨*wasps*⟩, ⟨*hornets*⟩, and ⟨*red ants*⟩
False Positives	[Other aspects], such as ⟨*witness*⟩ and ⟨*expert testimonies*⟩

There are no multiple senses here, but we still see the overgenerality problem.

3.2.8 Occupation

The next few patterns are my implementations of the original concept–instance relations defined by Fleishman et al. Thanks to named entity detection (NED), my patterns will be more precise, because NED removes the guesswork from detecting proper noun phrase boundaries. Below are two versions of my occupation pattern, which correspond to Fleischman's CN/PN pattern designed to capture the ⟨*NE*⟩ has-post [*Y*], or ⟨*NE*⟩ has-title [*Y*] relations.

Pattern	[(properNP) commonNP (and (properNP) commonNP)] ⟨*NE*⟩
Instances	[former dictator] ⟨*Ferdinand Marcos*⟩ [U.S. President] ⟨*Jimmy Carter*⟩ [Pulitzer Prize-winning poet and Hudson resident] ⟨*John Ashbury*⟩
False Positives	[the many layers] ⟨*Chris Cooper*⟩ gave Colonel Fitts

Pattern	[{properNP, commonNP} (of) (the) properNP] ⟨*NE*⟩
Instances	[Former Secretary of Health and Human Services] ⟨*Louis Sullivan*⟩ [Wisconsin Democrat] ⟨*Russ Feingold*⟩
False Positives	on his way to begin a tour of [the U.S.] ⟨*Adams*⟩ has called

False positives occur when the pattern match crosses phrase boundaries; that is when one phrase or clause ends with a common noun, and the next one begins with a named entity.

3.2.9 Appositive

Also a descendent of Fleischman et al., the appositive pattern is my implementation of the APOS concept–instance relation. However, for the sake of precision, I have

restricted the pattern to capture only instances matching ⟨*named entity*⟩, [*non-NE determiner phrase*] construction. I found that ⟨*nounphrase*⟩, [*noun phrase*] was far too wide a net to cast, capturing every pair of NPs in a comma-separated list, for example.

Pattern	⟨*NE*⟩, [thirdDP (and thirdDP)],
Instances	⟨*ETA*⟩, [an acronym that stands for Basque Homeland and Freedom], ⟨*Neta Bahcall*⟩, [a Princeton astronomer and a member of the survey team]. ⟨*El Shaddai*⟩, [a Catholic group].
False Positives	a plot to blow up the ⟨*United Nations*⟩, [a federal building], two tunnels ... no sign points to ⟨*Ur*⟩, [the ziggurat], the tombs ...

As you can see, this pattern still captures some list formations when a common determiner phrase lies adjacent to a named entity.

3.2.10 Copular

A copular expression is an explicit statement of the is-a relation: ⟨*X*⟩ is [*noun phrase*]. This pattern is designed to capture instances of that form.

Pattern	⟨*entity*⟩ be [highestNP (and highestNP)]
Instances	⟨*The Hague*⟩ is [home to the International Court of Justice] ⟨*Ur*⟩ was [the capital of Sumer and an important commercial center in Mesopotamia] ⟨*The Aga Khan*⟩ is [the 49th imam of the Ismaili community]
False Positives	The return of the NBA in ⟨*Toronto*⟩ was [a rush]. Yoshida, the anthropologist, said ⟨*Japan*⟩ is [fertile ground for Nostradamus] ...

False positives occur when the target is the object of a preposition, or when the entire copular expression is embedded in a subordinate clause.

3.2.11 Became

The copular pattern has relatively high recall, and low precision, due to the high term frequency of "to be" conjugations. On the other hand, "became" is a lower recall, higher precision pattern capturing essentially the same relation. Technically speaking, the relation is ⟨X⟩ became [Y], not "is-a".

Pattern	⟨*entity*⟩ became [highestNP (and highestNP)]
Instances	⟨*Egypt*⟩ became [the first Arab country to sign a peace treaty with Israel in 1979]. ⟨*Althea Gibson*⟩ became [the first black tennis player to win a Wimbledon singles title] ⟨*President Roosevelt*⟩ became [the first chief executive to travel through the Panama Canal]
False Positives	ever since ⟨*Bush*⟩ [became governor]

As you can see, this pattern is useful for capturing "the nth X to do Y" constructions. Like the copular pattern, false positives can occur when the entire expression

is embedded.

3.2.12 Relative Clause

The relative clause pattern is a simple juxtaposition of two phrases from the library described in the previous section. It matches a named entity X modified by a non-restrictive relative clause Y. This captures the relations $\langle X \rangle$ did [Y], or $\langle X \rangle$ experienced [Y].

Pattern	$\langle NE \rangle$, [RC],
Instances	$\langle Von\ Gruenigen \rangle$, who [won the world giant slalom title ...], $\langle Pippen \rangle$, who [played his first 11 seasons with the Chicago Bulls], $\langle Iverson \rangle$, whose [nickname is the Answer ...],
False Positives	James Shields: Irish immigrant and opponent of $\langle Abraham\ Lincoln \rangle$, who [is the only person to have been a senator from three states] Jackson said of $\langle Brown \rangle$, who [played for Brown with the Los Angeles Clippers]

A common error, as demonstrated here, is when the relative clause attachment is ambiguous between the head of a noun phrase and the object of its PP modifier.

3.2.13 Was Named

The "was named" pattern is useful for capturing $\langle X \rangle$ held-post [Y], or $\langle X \rangle$ earned-distinction [Y] relations.

Pattern	⟨*NE*⟩ {was, were, {has,have} been} named (as) [highestNP (and highestNP)]
Instances	in 1994 when ⟨*Leon Panetta*⟩ was named [Chief of Staff of the White House]. ⟨*Sanjay Dutt*⟩ was named [best actor for his role as a vendor in the murder tale "Vaastav."]
False Positives	Bernie Williams and ⟨*Hideki Irabu*⟩ were named [player and pitcher of month in the American League for May].

This is another highly precise pattern, but I did find a false positive in the tricky parallel construction noted above. This example demonstrates the shortcomings of CoL. ForBIN's regular language.

3.2.14 Other Verb Patterns

So far we've restricted ourselves to a fairly small subset of relations. There are, of course, many other relations that make good defining characteristics, and we'd like to capture some of those while maintaining a reasonable balance between precision and recall. In this spirit, last year our own Wes Hildebrandt compiled term frequency statistics from biography.com, from which we were able to identify the 32 most frequently occurring verbs in biographic entries. I grouped these verbs according to their argument structure, and used them in five new patterns. Here are verb categories:

NP argument, animate subject: found, formed, retired, discovered, studied, wrote, taught, published, married

NP argument, generic subject: won, led, made, began, joined, founded, established, received, developed, produced, introduced

PP argument: lived, died, worked, served

transitional: moved, went, returned, came

passive: born, killed, appointed, elected

This is what they look like in the context of their respective patterns:

animateNPverb: **Example:**	⟨*person*⟩ verb [highestNP ...] ⟨*Napoleon*⟩ [wrote fantastically fraudulent dispatches of military exploits].
genericNPverb: **Example:**	⟨*NE*⟩ verb [highestNP ...] ⟨*Pittsburgh*⟩ [produced steel in prodigious quantities].
PPverb: **Example:**	⟨*person*⟩ verb [higherPP ...] ⟨*Nelson A. Rockefeller*⟩ [died in New York at age 70].
transitional: **Example:**	⟨*person*⟩ verb {to, from} [highestNP ...] ⟨*Stokes*⟩ [came from a family long active in social causes and public service].
passive: **Example:**	⟨*person*⟩ [{was, were} verb ...] ⟨*Ezra Pound*⟩ [was born in Hailey, Idaho].

3.3 Putting it all Together

In Chapter 1, I discussed two major shortcomings of the Katz et al. pattern matching system. The first section of this chapter explains how I addressed the precision, or boundary detection, problem with "syntactic" regular expressions, and the second section shows how my patterns make use of them. In this section, I describe how I addressed the adjacency problem.

The patterns in the previous section only tell part of the relation extraction story. As presented above, they will still only capture one nugget in the aforementioned example:

> Holt, 50, a physicist, was the assistant director of Princeton Plasma Physics Laboratory.

What we need to do is stop treating each pattern as an independent atomic unit, and instead, unite them all in a single pattern matching architecture.

As it turns out, I divided this "single" architecture into three independent modules. There are two reasons for this. First, I found that the patterns fall into three

natural classes, and secondly, it is always useful to parallelize when processing such massive amounts of data. I describe these three modules in the sections below.

3.3.1 Entity-First Patterns

A natural way to group the nineteen syntactic patterns is according the order in which the target entity and definitional nugget occur. The larger of these two groups, with twelve members, is the entity-first category:

entity-first patterns: age, affiliation, appositive, copular, became, relative clause, was named, animateNPverb, genericNPverb, PPverb, transitional, passive

A thorough scan of the positive matches returned by Katz et al.'s entity-first patterns revealed that a vast majority (approximately 95%) of the targets detected were named entities. Now that I have IdentiFinder to show me the NE boundaries, I can refine these patterns by searching only for named entity targets.

The question remains, however, what do we do about nuggets that are separated from the target by some intervening clause? Also, as a corollary, how can we be sure that the long-distance nugget corresponds to our target, and not some other entity mentioned later in the text? I chose to adopt the following general solution: for each target, look at the adjacent text and see if it matches one of our patterns. If so, record the match, consume the matching text, and repeat.

There is one disclaimer however: we must be wary of the order in which we iterate over the pattern set. We would not, for example, expect to see the appositive pattern in the text following a copular. In fact, this would open the door to many false positives of the form:

> ⟨ Mr. X ⟩ is the president of Company Y., [the world's largest producer of Z's].

As demonstrated here, some patterns are inherently more adjacent than others, and should thus appear earlier in the iteration. Also, for efficiency's sake, I will assume the patterns form a natural precedence ordering (in terms of adjacency, or "target

affinity"), and thus only one iteration is required to capture all relevant nuggets. Here is the order I propose:

age > affiliation > appositive > relative clause > any verb pattern

It turns out this ordering is only violated in rare, pathological cases. Now, the famous Mr. Holt gets three nuggets instead of one. Here are some more examples of multiple-nugget sentences from the AQUAINT corpus:

Multiple-Nugget Sentences
⟨*John Stevens*⟩, [80], [a lawyer from Muscatine, Iowa],
⟨*Iverson*⟩, [22], was [last year's top draft pick].
⟨*Luci Baines Johnson Turpin*⟩, [daughter of President Johnson], [is 52]
⟨*Anwar Sadat*⟩, who [was Egyptian president at the time], was [a staunch U.S. ally].
⟨*The Notorious B.I.G.*⟩, whose [real name was Christopher Wallace], [was killed in a drive-by shooting]

3.3.2 Entity-Second Patterns

The remaining seven patterns look for a definitional nugget before the target entity. Unlike the entity-first patterns, these are generally not chained, so we can run them in succession on the same input text. The only tricky part here is detecting the correct nugget phrase boundary. For example, consider the following two instances of the "also called" pattern:

1. a book of rhymed prophecies, called "Centuries"

2. a spinoff of a Microsoft program called Libraries Online

In the first example, the highestNP captures the correct nugget, because "Centuries" is indeed a book of rhymed prophecies. However, in the second example, Libraries Online is a program (not a spinoff) so the highestNP nugget, "a spinoff of a Microsoft program", is inexact in this case.

How do we decide what to return here, then? It is nearly impossible to determine the appropriate nugget of information based on syntax alone, so I adopt the following solution: return all possibilities, and let the user decide! But is that really the best information CoL. ForBIN can provide, based on its syntactic knowledge? Not quite. I can also tell you the head noun in each possible nugget phrase, and its relative distance from the target.

Let's walk through an example of this procedure. Consider the following excerpt: "the criminal use of justice by a fundamentalist prosecutor called Kenneth Star" From a purely syntactic standpoint, this snippet has three possible phrases in nugget position: "a fundamentalist prosecutor", "justice by a fundamentalist prosecutor", and "the criminal use of [that justice]". We can't just choose the smallest one, or the second smallest one, as demonstrated in the examples above, so we return all the information we can:

Nugget	**Head**	**Distance**
a fundamentalist prosecutor	prosecutor	1
justice by a fundamentalist prosecutor	justice	2
the criminal use of justice by a fundamentalist prosecutor	use	3

Now the user, who is presumably better informed semantically, can choose which one is best. Perhaps she already has strong evidence that Kenneth Star is a lawyer of some kind. Then she will notice that the head noun "prosecutor" is a specialization of "lawyer", and use this information to decide on the first nugget. Or perhaps she will query WordNet for the lowest common hypernym between the target and each nugget head (normalizing all named entities to their subtype, so Kenneth Star would become "person"), and choose the nugget with the shortest hypernym path. Let's try this approach, and see what we come up with here:

```
wn person -hypen
Sense 1
person, individual, someone, somebody, mortal, human, soul
```

```
       => organism, being
           => living thing, animate thing
               => object, physical object
                   => entity
       => causal agent, cause, causal agency
           => entity

wn prosecutor -hypen
Sense 1
prosecutor, public prosecutor, prosecuting officer, prosecuting attorney
       => lawyer, attorney
           => professional, professional person
               => adult, grownup
                   => person, individual, someone, somebody, mortal, human
                       => organism, being
                           => living thing, animate thing
                               => object, physical object
                                   => entity
                       => causal agent, cause, causal agency
                           => entity

wn justice -hypen
Sense 1
justice, justness
       => righteousness
           => morality
               => quality
                   => attribute
                       => abstraction
```

"Justice", in the sense used here, does not share a common hypernym with "person", so "prosecutor" wins again. However, without word sense disambiguation, we must consider all senses equally, so the third sense of "justice" will actually give us the shortest hypernym path:

```
Sense 3
judge, justice, jurist, magistrate
       => adjudicator
           => person, individual, someone, somebody, mortal, human, soul
               => organism, being
                   => living thing, animate thing
                       => object, physical object
                           => entity
               => causal agent, cause, causal agency
                   => entity
```

In retrospect, "shortest hypernym path" probably isn't the best heuristic. What we really want is a short path on the target side, and a long one on the nugget side, because this will give us the most specialized description. By this heuristic, "prosecutor" wins over both senses of "justice". Should this approach fail, then the user can make use of the distance field, perhaps defaulting to the closest possible nugget.

3.3.3 Occupations

Not fitting quite well into either of the above categories are the two occupation patterns. Like the entity-first patterns, their target is strictly a named entity, but like the entity-second patterns, the nugget comes before the target. However, unlike the entity-second patterns, occupations do not generally require the sophisticated nugget disambiguation techniques described above. Furthermore, the occupation patterns generate the most matches by far (almost an order of magnitude greater than any other pattern), and thus require considerably more time to run than the rest. The

combination of these factors led me to treat occupations as a separate class of patterns, with the simplest matching algorithm of all. That is: match the target and nugget. If the nugget ends in a plural noun, check for multiple targets (similar to the approach taken by the "like" and "such as" patterns), and record them all.

Chapter 4

Experimental Results

4.1 Generating Ground Truth

My first approach to fixing the boundary detection problem was to start with the patterns of Katz et al. [9], and substitute the target and nugget regular expressions with my newly created highestNP pattern. This approach saw mixed results. For every good nugget captured by the new patterns (e.g., "a book of rhymed prophecies"), there was a counterexample where they matched too greedily (e.g., "a spinoff of a Microsoft program"). Also, it became apparent that the highestNP pattern was overly sophisticated for target detection. Most correct targets were either named entities or instances of the simpleNP pattern.

In addition to providing such insights, these initial trials also gave rise to a testbed for human evaluation, which led to the generation of ground truth elements. Based on user input provided through a form on the web, I was able to collect 50-500 positive and negative instances of each pattern from a subset of 886 AQUAINT documents. Incidentally, the subset I chose was comprised of the supporting documents for all the definitional nuggets deemed valid for the TREC 12 competition. I chose this set because it was a reasonable size for testing, and because it was guaranteed to contain answers to last year's definitional questions (my initial benchmark for evaluating the new patterns). I was also able to collect 500 instances of valid entity/definition pairs from 50 randomly chosen AQUAINT documents. These did not necessarily

correspond to any surface patterns, and were meant to serve as a basis for evaluating total pattern recall.

4.2 Testing Against Truth

4.2.1 Generating Benchmark Statistics

As noted above, ground truth items were generated by manually annotating old pattern-matching results. Initially, two sets of results were annotated by hand. These were the patterns of Katz et al. in their original form, and with target and nugget regular expressions substituted with highestNP. I will refer to the original patterns as "trec12" and to the substituted ones as "baseline"[1]. Table 4.1 contains the human-judged results of the trec12 and baseline experiments. Precision is defined in terms of both accuracy and relevance, so in order to be judged correct, a nugget must have correct bounds, and must be relevant to the target. The formula I used to calculate precision is *hits/matches*, where *matches* is the number of pattern instances found, and *hits* is the number of correct instances, as defined above.

We can also evaluate the entire pattern set in terms of nugget recall. That is, out of all the valid target/nugget pairings in the corpus, how many can we extract using syntactic patterns alone? Of course, it is nearly impossible to hand-annotate the 3 Gb AQUAINT corpus with all valid target/nugget pairs. The best we can do is choose a random subset of documents to annotate, and see how many ground truth pairs are extracted by the patterns as well. Table 4.2 shows the recall statistics for the trec12 patterns [2], and table 4.3 shows how much of the recall can be attributed to each pattern.

A match is ruled inexact, but acceptable, if the targets overlap by more than 50%, and the matched nugget contains all the information in the true nugget, along with a

[1] since the official ground truth items were derived from the substituted ones

[2] I have chosen to exclude the baseline patterns here, because they did not include the occupation patterns, which account for over half of the recall

Pattern	Trec12		Baseline	
	hits/matches	precision	hits/matches	precision
e1_aka	7/28	25.00	3/29	10.34
e2_aka	4/18	22.22	11/29	37.93
e1_also_called	2/72	02.78	2/65	03.08
e2_also_called	5/32	15.63	27/65	41.53
e1_appositive	41/206	19.90	160/1014	15.78
e2_appositive	28/206	13.59	73/1014	07.20
e1_became	26/72	36.11	50/95	52.63
e1_is	29/137	21.17	97/478	20.29
e2_like	6/67	08.69	70/378	18.52
e1_named	7/18	38.89	7/13	53.85
e2_named	3/18	16.67	6/26	23.08
e1_rel_clause	23/94	24.46	46/179	25.70
e1_verb_np_generic	35/207	16.90	141/492	28.66
e1_verb_np_person	5/51	09.80	40/143	27.97
e1_verb_passive	13/41	31.70	22/41	53.66
e1_verb_pp	8/44	18.18	40/72	55.56
e1_verb_transitional	2/35	05.71	32/98	32.65
Weighted Average Precision (Trec12):				18.95
Weighted Average Precision (Baseline):				19.55

Table 4.1: Human-judged precision statistics for the trec12 and baseline patterns. The averages are weighted on the number of matches, which is roughly proportional to recall, but not quite (see the discussion at the end of this chapter). Notice that several patterns appear to be listed twice. This is because, previously, many patterns had both an entity-first and entity-second version. In each case, however, one of these versions was phased out, because its precision was considerably lower than the other. The one exception to this rule is the named pattern, which maintained its entity-second version, and spawned the entity-first "was named" pattern.

Truth Items	Matches		Recall	
	exact	inexact	exact	inexact
483	144	175	29.81	36.23

Table 4.2: Recall numbers for the trec12 patterns.

Pattern	Matches	Percent
e2_aka	1	00.69
e1_appositive	55	38.19
e1_became	1	00.69
e1_is	5	03.47
e2_like	1	00.69
e1_relative	6	04.17
e2_occupation	75	52.08

Table 4.3: Trec12 recall breakdown by pattern (exact).

limited amount of extraneous text.[3] As you can see, much of the recall[4] is still coming from the appositive and occupation patterns, the modified versions of Fleishmann's APOS and CN/PN relations.

4.2.2 Comparison to Benchmark

In this section, I describe my method for comparing the new pattern results to the benchmark statistics. As mentioned above, I was able to collect 50-500 positive and negative instances of each pattern from the training set of documents. The problem with this collection is that it is by no means comprehensive. This is because the human annotator only had access to the specific *sentences* that contained an instance of the old, adjacency-dependent patterns. Thus many positive and negative instances were ignored, because the annotator never saw them. To account for this flaw, I will throw out any new matches that do not roughly correspond[5] to any items in the positive or negative training sets. Precision will then be defined as $p/(p+n)$, where p is the number of positive examples captured, and n the number of negative examples.

As you can see in table 4.4, CoL. ForBIN's patterns are significantly more precise,

[3]The total match length must be no greater than twice that of the true nugget.

[4]In Chapter 2, I claimed this number was close to 90%. This turned out to be a classic case of

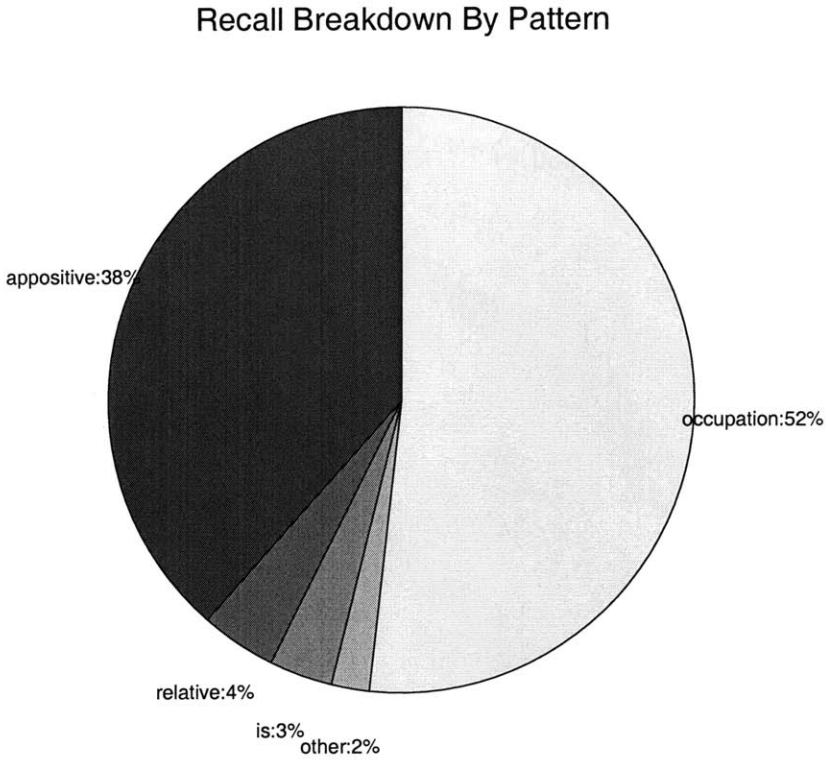

Figure 4-1: Pie chart view of the trec12 recall breakdown.

but at what cost? Usually, there is a tradeoff between precision and recall, but since recall is so important here, my goal was to keep it relatively consistent with that of trec12 patterns. As it turned out, my recall numbers actually *improved*, in terms of both exact matches and inexact matches. (See Table 4.5.) This was due largely to the new long-distance matching architecture.

Table 4.6 and Figure 4-2 show the recall breakdown by pattern. APOS and CN/PN still dominate, but that's okay now that they are much more precise. Also, contribution from other patterns is up from 10 to nearly 17%. This is a direct reflection of the new long-distance matching strategies.

---

overfitting.

[5]Refer to the inexactness discussion in the previous section.

Pattern	Matches		Precision	
	positive	negative	exact	inexact
e1_appositive	67	10	70.13	87.01
e1_became	40	4	79.55	90.91
e1_is	50	11	60.66	81.97
e1_rel_clause	13	12	44.00	52.00
e1_verb_np_generic	67	82	33.56	44.97
e1_verb_np_person	17	25	38.10	40.48
e1_verb_passive	27	4	74.19	87.10
e1_verb_pp	42	5	72.34	89.36
e1_verb_transitional	32	22	46.30	59.26
e1_was_named	8	1	66.67	88.89
e2_aka	10	0	100.00	100.00
e2_also_called	7	0	100.00	100.00
e2_like	29	5	79.41	85.29
e2_such_as	22	7	66.67	81.48
e2_named	3	0	100.00	100.00
Weighted Average Precision (Exact):				58.12
Weighted Average Precision (Inexact):				68.67

Table 4.4: New pattern precision, based on ground truth

Version	Truth Items	Matches		Recall	
		exact	inexact	exact	inexact
trec12	483	144	175	29.81	36.23
forbin	483	156	186	32.30	38.51

Table 4.5: Forbin pattern recall compared with trec12.

Pattern	Matches	Percent
e1_affiliation	1	00.64
e2_aka	2	01.28
e1_appositive	40	25.64
e1_is	4	02.56
e2_like	17	10.90
e2_named	1	00.64
e2_occupation	82	52.56
e1_rel_clause	7	04.49
e1_verb_np_generic	1	00.64
e1_verb_pp	1	00.64

Table 4.6: New recall breakdown by pattern.

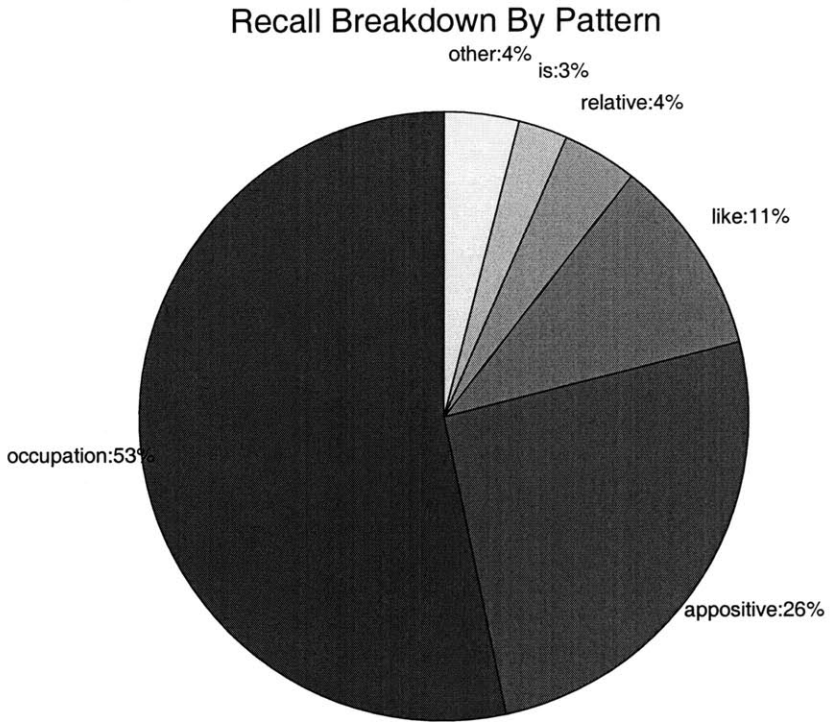

Figure 4-2: Pie chart view of the recall breakdown (new).

4.3 Human Judgements

As noted above, the process of generating truth was somewhat myopic, in the sense that I had not foreseen the advent of the long-distance pattern matching architecture. Thus the human annotator was not asked to look for long-distance nuggets of the form detected by the entity-first module. To correct this error, the same human annotator was asked to re-evaluate the new entity-first results on a per-pattern basis. These results are more reflective of the patterns' actual precision, and are displayed in Table 4.7. Also, as a sanity check and for completeness, I asked the annotator to evaluate the entity-second patterns. Those results can be found in Table 4.8. Note that, even with the larger sample space and human evaluator, these results are reasonably consistent with those reported in the previous section.

Pattern	Matches	Precision	
		exact	inexact
e1_affiliation	94	0.83	0.85
e1_age	11	1.00	1.00
e1_appositive	232	0.87	0.89
e1_became	20	0.70	0.85
e1_is	197	0.63	0.68
e1_rel_clause	190	0.53	0.55
e1_verb_np_generic	63	0.43	0.49
e1_verb_np_person	31	0.42	0.48
e1_verb_passive	37	0.46	0.57
e1_verb_pp	28	0.75	0.93
e1_verb_transitional	78	0.35	0.36
e1_was_named	44	0.27	0.55
e2_occupation	132	0.73	0.75
Weighted Average Precision (Exact):			64.25
Weighted Average Precision (Inexact):			68.90

Table 4.7: Entity first pattern precision, as judged by a human evaluator.

Pattern	Matches	Precision	
		exact	inexact
e2_aka	25	68.00	68.00
e2_also_called	47	70.21	70.21
e2_like	234	39.32	42.30
e2_named	11	63.64	63.64
e2_suchas	121	46.28	47.11
Weighted Average Precision (Exact):			46.58
Weighted Average Precision (Inexact):			48.40

Table 4.8: Entity second pattern precision, as judged by a human evaluator.

Overall Weighted Average Precision	
exact	inexact
59.90	63.27

Table 4.9: Overall weighted average precision calculated over all the patterns.

4.4 Testing in QA Context

I have shown that my patterns perform better than the trec12 ones in terms of both precision and recall, but I still have not addressed the topic of this thesis: that is, answering definitional questions. Bearing in mind that CoL. ForBIN is not intended to be a standalone QA system, but rather part of a multi-pronged approach, I decided to "ask"[6] it for nuggets pertaining to some of the TREC 12 definitional question topics.

I have grouped ForBIN's responses into four categories. The first one, vital responses, correspond to nuggets that were deemed vital by NIST for TREC 12. I have also included a duplicates category for paraphrases of previously returned vital responses. Duplicates are considered by NIST as extraneous text, and should thus be avoided.[7] The other two categories are "okay" and "no judgement". NIST defined its ground truth as the set union of all nuggets submitted by participants that pertain to the topic entity. Those that were not considered "vital" defining characteristics were classified as "okay". I have included a separate category, "no judgement", for nuggets that were not submitted by anyone as TREC 12 answers. I leave it as a task for the reader to determine their status.

First, let's examine some "Who is" questions. Since most of the patterns match exclusively named-entity targets, we expect these to have the most database hits. We also expect overall precision to be roughly equivalent to the weighted average precision figures reported above, so 60-65%. I chose the following "Who is" targets at random from the TREC 12 question set: "Andrew Carnegie", "Alberto Tomba", "Aaron Copland", and "Vlad the Impaler". See Tables 4.10, 4.11, 4.12, and 4.13 for CoL. ForBIN's responses regarding these topics.

Next we examine some "What is" questions. (See Tables 4.14 and 4.15.) The first topic, "golden parachute" is not a named entity, so we expect significantly lower recall here. Remember that only the entity-second patterns allow targets that are not named entities, and these only account for approximately 13% of total pattern recall.

[6] In other words, query the database.

[7] But ForBIN's primary responsibility is still recall. It remains the task of the user to filter duplicates.

Qid 2224: Who is Andrew Carnegie?
Vital responses (1 missed)
built a steel empire made gifts amounting to $350 million before he died in 1919
Vital duplicates
the steel tycoon money built more than 1,600 public libraries in the United States in the early part of the century
No Judgement
the self-made man rose from poverty to the role of captain of industry a name that crops up frequently alongside Gates' these days an immigrant from Scotland

Table 4.10: Sorted nuggets from CoL. ForBIN pertaining to "Andrew Carnegie"

Qid 1907: Who is Alberto Tomba?
Vital responses (1 missed)
Italian Olympic and World Alpine skiing champion settled this year after courts set a date for the ski champ's tax trial
Okay responses
won his only World Cup overall title and two discipline titles in the cup finals here in 1995
Vital duplicates
Skiing superstar Olympic gold medalist Italian top skier
No Judgement
her compatriot Local hero came from far off the pace with a blistering second run to momentarily overtake first-run leader Stangassinger, but the Austrian won

Table 4.11: Sorted nuggets from CoL. ForBIN pertaining to "Alberto Tomba"

Qid 1901: Who is Aaron Copland?
Vital responses (2 missed)
composer "American-sounding" music was composed by a Brooklyn-born Jew of Russian lineage who studied in France and salted his scores with jazz-derived syncopations
Okay responses
the other great historic figurehead at Tanglewood
Vital duplicates
divided composers like himself who "play at writing operas" from those "hopelessly attracted to this 'form fatale."'
No Judgement
vastly preferred Stravinsky to Wagner or Brahms.

Table 4.12: Sorted nuggets from CoL. ForBIN pertaining to "Aaron Copland"

Qid 1933: Who was Vlad the Impaler?
Vital responses (0 missed)
the 16th-century warrior prince inspired Bram Stoker's 1897 novel Dracula was known for piercing his victims on spikes while he ate his dinner

Table 4.13: Sorted nuggets from CoL. ForBIN pertaining to "Vlad the Impaler"

Qid 1905: What is a golden parachute?
Vital responses (2 missed)
severance packages

Table 4.14: Sorted nuggets from CoL. ForBIN pertaining to "golden parachute"

Qid 2201: What is Bollywood?
Vital responses (3 missed)
The Bombay-based film industry

Table 4.15: Sorted nuggets from CoL. ForBIN pertaining to "Bollywood"

The second topic, "Bollywood", is a named entity, but it has a low term frequency so we expect low recall for this one as well. ForBIN's responses confirm our expectations.

The last topic, "Hague", has a high term frequency, so we'd expect higher recall and lower precision with this one. But, assuming IdentiFinder can tell us that it's "the Hague", the GPE we're interested in, and not "Hague", the person, we come up with some reasonably good answers. (See Table 4.16.)

Here, we see a truly false nugget: "a defense lawyer". This was a case of ambiguous appositive modification: "In the Hague, a defense lawyer, Michail Wladimiroff, said ...". What about the rest of the "no judgment" nuggets? Table 4.17 shows how they were classified by our human annotator.

So, discounting duplicates, precision is even higher than expected - about 80%. Likewise, our vital nugget recall is over 60%. Of course, this is a very small sample, and we could have just been lucky. These examples were merely meant to illustrate CoL. ForBIN in action.

4.5 Discussion

The skeptic will be quick to point out that the data set I used for benchmark comparison is quite impoverished. Especially deceiving are the high precision numbers attributed to the entity-second patterns. I mentioned several times that I had collected 50-500 positive and negative examples of each pattern, but the greatest number

Qid 2158: What is the Hague?
Vital responses (0 missed)
the Dutch cap. the Dutch city that is the headquarters for the International Court of Justice site of the World Court venue of the International Criminal Tribunal for the former Yugoslavia
Okay responses
handles disputes between nations
Vital duplicates
the Dutch political center home to the International Court of Justice last week indicted Milosevic for orchestrating the forced explusion of some 800,000 ethnic Albanians from Kosovo
No Judgement
a defense lawyer an instrument of American political goals

Table 4.16: Sorted nuggets from CoL. ForBIN pertaining to "the Hague" (as a GPE)

Vital
the self-made man rose from poverty to the role of captain of industry an immigrant from Scotland
Okay
Local hero vastly preferred Stravinsky to Wagner or Brahms an instrument of American political goals
False Positive
a name that crops up frequently alongside Gates' these days her compatriot came from far off pace with a blistering second run to momentarily ... a defense lawyer

Table 4.17: Our judgements of the nuggets for which there was no NIST judgement

you see in either of the matches column is 82. This is because, as it turned out, there was very little overlap in the sets of target/nugget pairs captured by the old and new systems. In other words, the new patterns filtered out many bad nuggets, but missed several good ones as well.

This appears to be a classic tradeoff of precision for recall, yet I claim that my recall numbers improved as well! How is this possible? Well, thanks to long-distance matching, ForBIN was able to get recall from elsewhere in the text. Unfortunately, the human annotator was never asked to look elsewhere in the text for positive and negative instances of the patterns, so we had no means of evaluating these new results programatically.

Thus another round of manual annotation was warranted. Due to time constraints, we were unable to annotate a data set as large as the one used in the baseline experiments. Instead, using the same collection of 886 documents, we annotated the results for each pattern until we saw them all (in the low recall cases), or until we felt we had seen a large enough accurate sampling of the data (in the high recall cases). For most of the high recall cases, the cutoff was around 200 items. The only problem with this approach is that it distorts the weighted averages, by making the pattern distribution seem more uniform. A better weighing metric is percent of total pattern recall. If we weigh on this instead, we get an average precision of 72.14% (73.67%, counting inexact matches), thanks to large contributions from appositive and occupation.

How does this translate into a TREC F score? Well, it's difficult to judge, because NIST's notion of precision is slightly different from the one presented here. Theirs includes an answer length allotment, and a penalty for duplicate responses. However, if we make certain assumptions, we can estimate CoL. ForBIN's F scores for varying values of β. In particular, let's assume that every acceptable response uses exactly its length allotment, and that every incorrect answer and duplicate is that same length as well. Also, since we're using the entire allotment - not just the exact answer - we can assume that the inexact (better) precision and recall calculations apply here. The only factor left to consider is the number (or percentage, rather) of duplicate responses. This is something I have not analyzed in detail, so I must treat it as a

δ	**F($\beta = 5$) score**
0.9	0.3312
0.8	0.3626
0.7	0.3744
0.6	0.3806
0.5	0.3844
0.4	0.3870
0.3	0.3888
0.2	0.3903
0.1	0.3914
0.0	0.3923

Table 4.18: CoL. ForBIN's estimated F($\beta = 5$) score, as δ varies.

δ	**F($\beta = 3$)**	**F($\beta = 2$)**	**F($\beta = 1$)**
0.9	0.2707	0.2087	0.1237
0.8	0.3316	0.2911	0.2131
0.7	0.3585	0.3353	0.2808
0.6	0.3736	0.3628	0.3339
0.5	0.3833	0.3816	0.3765
0.4	0.3901	0.3953	0.4116
0.3	0.3951	0.4056	0.4409
0.2	0.3989	0.4138	0.4658
0.1	0.4019	0.4203	0.4872
0.0	0.4044	0.4257	0.5058

Table 4.19: CoL. ForBIN's estimated F score, varying with β and δ.

variable, δ. Table 4.18 shows CoL. ForBIN's estimated F($\beta = 5$) score for varying values of δ. What this tells us is that, even if only 10% of our responses are unique ($\delta = 0.9$), we achieve an F($\beta = 5$) score of 0.3320 - good enough for second place in TREC 12. If half are unique, we get 0.3846 - less than 0.01 away from the assymptotic upper bound.

Note that these results are for $\beta = 5$, which values recall about five times more highly than precision. But every year, NIST tightens its standards on precision. For example, this year at TREC 13, β was set equal to 3 for defintional questions. How would this affect our F score? For that matter, what if β were set equal to 2 or 1? Table 4.19 shows these results. (See Figure 4-3 for a graphical view.)

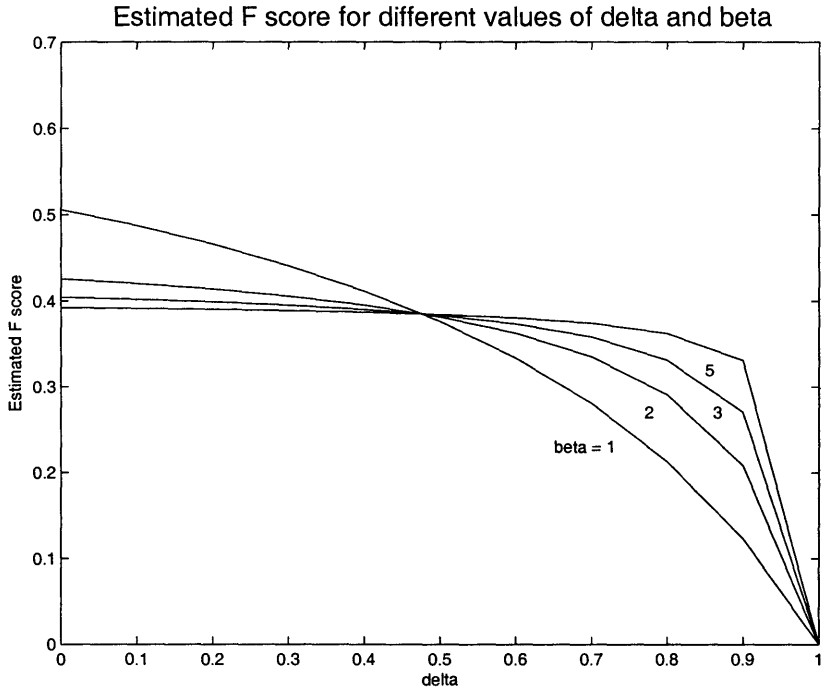

Figure 4-3: F score for different values of β, as a function of δ.

Naturally, as β decreases, δ becomes more of a factor, because we are weighing more heavily on precision. So, with all things equal ($\beta = 1$), a CoL. ForBIN user can achieve an F score above 0.50, or as low as 0.12, depending on how well she is able to filter duplicates. Bear in mind once again that ForBIN is not meant to be a standalone utility, but rather a tool for gleaning information in preprocessing. If one already has an F score close to 0.50 after *preprocessing*[8], one can afford to spend time on more costly runtime techniques aimed at boosting that number.

[8]Remember, BBN won TREC 12 with an F score of 0.555.

Chapter 5

Future Work

As the saying goes, your F score is only as strong as its weakest link. In CoL. ForBIN's case, that link is recall. Below, I ponder the future of CoL. ForBIN, with special emphasis on improving recall.

5.1 Target Expansion

The simplest way to improve recall is with target expansion, or substituting references that are not fully resolved with their fully resolved antecedent. The intuition behind this notion is as follows: When a new discourse referent is introduced, he/she/it is often referred to by name. However, seldom is that full name repeated in the remainder of the discourse. Instead, the name is replaced with an anaphor (e.g. pronoun, nominal reference, or partial name) which the reader is left to resolve. So, for example, if we restrict ourselves to nuggets associated with the string "Allen Iverson", we miss all the nuggets pertaining to "Iverson", "Allen", "the Sixers' point guard", and "he" - all referring to the same Allen Iverson.

Of course, we cannot blindly assume that every "Iverson" in the corpus refers to Allen Iverson (or much worse, every "he"). But we *can* add another stage of preprocessing whereby all anaphoric references are replaced, or at least marked, with their fully resolved antecedents. To do so properly is a costly operation, involving a full parse of the text, but Infolab contributor Federico Mora has developed a low

cost, heuristics-based alternative. Unfortunately it was still incomplete when I began testing CoL. ForBIN, so I was unable to make use of it in my experiments.

5.2 CFG Parsing

While we're on the topic, is parsing really such a bad idea? As machines become faster and memory gets cheaper, the answer tends toward the negative. With that said, the future of CoL. ForBIN has two possible directions: we can either spend time fine-tuning the regular language so as to "fake" more syntactic constructions, or we can translate that language into a compact context-free[1] grammar. While the first path maintains the status quo, it may not be the wisest approach, because the underlying regular expressions are already quite large, and bordering on unwieldy. The second approach involves a substantial change in infrastructure, but the resulting language would be much more expressive and more representative of actual natural language. I foresee the future of CoL. ForBIN proceeding in the latter direction.

5.3 Statistical Methods

Statistical techniques have proven successful in several areas of NLP, ranging from POS tagging to information extraction. In fact, CoL. ForBIN already relies heavily on a statistical tool: the IdentiFinder named entity recognizer. So, if we're planning to endow CoL. ForBIN with a parser (see above), then it follows that we may want its grammar to be probabilistic. In other words, for a given parent node, we would like to know if its possible expansions include any target/nugget pairings, and if so, what is their probability distribution. These parameters are not hand-coded, of course, but must be learned from manually annotated training data. Fortunately, the ForBIN development process has lent us a convenient visual tool designed specifically for annotation, so the process of creating training data will be much less laborious in the future. Given such a training set, we can convert the process of finding pattern

[1] or some variant thereof

instances from a regular expression match to a probabalistic context-free grammar parse.

Chapter 6

Contributions

In CoL. ForBIN, I have described a system that extracts reasonably precise definitional nuggets from newspaper text. It then stores them in a database which is meant to serve as a backbone for a larger definitional question answering system. Although it is not intended for use in isolation, due to its lack of semantic knowledge, the database alone can achieve F scores ranging from 0.40 to 0.50, depending on β and the quality of duplicate filtering. Thus with syntactic clues as its only heuristics, and speed as fast as a database lookup, ForBIN could have placed second in the TREC 12 definitional category.

In a broader sense, this thesis contributes to the field of natural language processing by:

- Identifying certain relations that make good definitions, and the syntactic constructs in which they occur.
- Building a library of regular expressions that resemble context free rules, and thus give us a notion of syntax.
- Assembling syntactic patterns from these expressions designed to capture the aforementioned relations.
- Finding all instances of these patterns in the AQUAINT corpus, and storing them for use in question answering.

Recall remains a limiting factor, but with improvements such as target expansion and incorporating a parser, ForBIN will be an even more highly effective tool for question answering.

Bibliography

[1] Lucene. http://jakarta.apache.org/lucene/docs/index.html.

[2] Daniel M. Bikel, Richard L. Schwartz, and Ralph M. Weischedel. An algorithm that learns what's in a name. *Machine Learning*, 34(1-3):211–231, 1999.

[3] Eric Brill. A simple rule-based part of speech tagger. In *Proceedings of the Third Applied Natural Language Processing Conference (ANLP-92)*, 1992.

[4] Michael Fleischman, Eduard Hovy, and Abdessamad Echihabi. Offline strategies for online question answering: Answering questions before they are asked. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL 2003)*, 2003.

[5] Ulf Hermjakob, Abdessamad Echihabi, and Daniel Marcu. Natural language based reformulation resource and Web exploitation for question answering. In *Proceedings of the Eleventh Text REtrieval Conference (TREC 2002)*, 2002.

[6] Boris Katz. Using English for indexing and retrieving. In *Proceedings of the 1st RIAO Conference on User-Oriented Content-Based Text and Image Handling (RIAO 1988)*, 1988.

[7] Boris Katz. Annotating the World Wide Web using natural language. In *Proceedings of the 5th RIAO Conference on Computer Assisted Information Searching on the Internet (RIAO '97)*, 1997.

[8] Boris Katz, Sue Felshin, Deniz Yuret, Ali Ibrahim, Jimmy Lin, Gregory Marton, Alton Jerome McFarland, and Baris Temelkuran. Omnibase: Uniform access to

heterogeneous data for question answering. In *Proceedings of the 7th International Workshop on Applications of Natural Language to Information Systems (NLDB 2002)*, 2002.

[9] Boris Katz, Jimmy Lin, Daniel Loreto, Wesley Hildebrandt, Matthew Bilotti, Sue Felshin, Aaron Fernandes, Gregory Marton, and Federico Mora. Integrating Web-based and corpus-based techniques for question answering. In *Proceedings of the Twelfth Text REtrieval Conference (TREC 2003)*, 2003.

[10] Jimmy Lin. Indexing and retrieving natural language using ternary expressions. Master's thesis, Massachusetts Institute of Technology, 2001.

[11] Jimmy Lin, Dennis Quan, Vineet Sinha, Karun Bakshi, David Huynh, Boris Katz, and David R. Karger. What makes a good answer? The role of context in question answering. In *Proceedings of the Ninth IFIP TC13 International Conference on Human-Computer Interaction (INTERACT 2003)*, 2003.

[12] Gideon Mann. Fine-grained proper noun ontologies for question answering. In *Proceedings of the SemaNet'02 Workshop at COLING 2002 on Building and Using Semantic Networks*, 2002.

[13] Marin M. Soubbotin and Dimitry Soubbotin. Inteltext: Producing coherent linear texts while navigating in large non-hierarchical hypertexts. In *Human-Computer Interaction, Third International Conference, EWHCI '93, Moscow, Russia, August 3-7, 1993, Selected Papers*, 1993.

[14] Martin M. Soubbotin and Sergei M. Soubbotin. Patterns of potential answer expressions as clues to the right answers. In *Proceedings of the Tenth Text REtrieval Conference (TREC 2001)*, 2001.

[15] Martin M. Soubbotin and Sergi M. Soubbotin. Use of patterns for detection of likely answer strings: A systematic approach. In *Proceedings of the Eleventh Text REtrieval Conference (TREC 2002)*, 2002.

[16] Ellen M. Voorhees. Overview of the TREC 2001 question answering track. In *Proceedings of the Tenth Text REtrieval Conference (TREC 2001)*, 2001.

[17] Ellen M. Voorhees. The evaluation of question answering systems: Lessons learned from the TREC QA track. In *Proceedings of the Question Answering: Strategy and Resources Workshop at LREC-2002*, 2002.

[18] Ellen M. Voorhees. Overview of the TREC 2002 question answering track. In *Proceedings of the Eleventh Text REtrieval Conference (TREC 2002)*, 2002.

[19] J. Xu, A. Licuanan, and R. Weischedel. TREC 2003 QA at BBN: Answering definitional questions. In *Proceedings of the Twelvth Text REtrieval Conference (TREC 2003)*, 2003.