In Jacobs (ed.) *Text-Based Intelligent Systems: Current Research and Practice in Information Extraction and Retrieval*, Lawrence Erlbaum, 1992, 83-99.

Robust Partial-Parsing through Incremental, Multi-Algorithm Processing [1]

David D. McDonald

14 Brantwood Road, Arlington, MA 02174
(617) 646-4124 mcdonald@brandeis.edu

The problem that inspired this volume is how to develop intelligent computer systems that are "text-based"—systems that acquire their knowledge by assimilating massive amounts of ordinary natural language text, rather than having to be spoon-fed rules handcrafted by knowledge engineers.[2] A mature text-based systems would keep up with current events by simply reading the newspaper like we do. Conceivably it might even learn new fields by reading the text books.

The sheer number of things to be learned, and the pace with which they change, requires us to try to develop systems that can assimilate the information in a text with only a minimum of human oversight. The ready availability of candidate texts in electronic form: encyclopedias, newswires, daily papers, magazines, printer's tapes, etc. gives us every incentive to try.

An example may help to make it clear just what this enterprise is. Below is an article from the Wall Street Journal, shown exactly as it appears when downloaded electronically from the Dow Jones News Retrieval service.

---

[1] This paper is an expanded and revised version of a paper originally presented at the AAAI Spring Symposium on "Text-based Intelligent systems", March 1990 at Stanford University.

[2] An excellent example of the "handcrafted" approach is Lenat's CYC project at MCC (Lenat & Guha 1990), an out-growth of the Atari Encyclopedia project initiated by Allan Kay (Borning et al. 1983). This kind of careful deliberate work will surely be needed to bootstrap systems to a minimal competence. Large-scale intelligence and currency, however, will only come through the systems' reading what is written for people.

AN 910214-0090
HL Who's News: Goodyear Tire & Rubber Co.
DD 02/14/91
SO WALL STREET JOURNAL (J), PAGE B8
CO * GT WNEWS
IN PIPELINE OPERATORS (PIP) PETROLEUM (PET) AUTO PARTS AND EQUIPMENT INCLUDING TIRES (AUP)
TX GOODYEAR TIRE & RUBBER Co. (Akron, Ohio) -- George R. Hargreaves, vice president and treasurer of Goodyear, will become president and chief executive officer of the Celeron Corp. unit, a holding company for Goodyear's All American Pipeline. Mr. Hargreaves, 61, will assume the post effective March 1 and will retain his current posts. Robert W. Milk, Celeron's current president and chief executive, as well as an executive vice president for Goodyear, will be on special assignment until he retires April 30.

The availability of such "Who's News" articles prompted me to consider the task of automatically extracting information about people's change in position. (The results of this experiment are given in Section 4.) This means that the article is analyzed by a computer program to recover a set of four tuples consisting of the action, person or persons affected, their position (title), and the company or subsidiary. The tuples are then entered into a database. Below is an example of the first such tuple in this article.

```
#<edge75  80 Job-event 105
  event: #<event-type become-title>
  title: (  #<title "president"
            #<title "chief executive officer">  )
  person: #<person Hargreaves, George R.>
  company: #<subsidiary
               of: #<company Goodyear Tire & Rubber Company>
               name:  #<co-name "Celeron Corporation">>>
```

In this paper we examine the challenges that the effort to develop such a system will pose for research on natural language understanding. We begin by identifying the two central problems that this research must solve, and then move to describing the approach taken by the "Sparser" system, a mature program for extracting information in specific domains from unrestricted news articles.

1. Two Central Problems

Obviously, if a computer program is to learn anything from an English text, it must have some procedure for determining the information the text contains—some kind of natural language understanding system. But while many understanding systems have been developed over the last forty years (see Grosz et al., 1985, for a definitive set of early reference papers), none can yet meet the needs of the kind of text-based intelligent system we all envision. I believe that before this can happen, two large problems must be solved.

The first problem is that the information literally present in any naturally occurring text is only a fraction of the information that is conveyed to the person who reads it. These "omissions" must be recognized and the gaps filled before a program will understand from a text what a person would. This is a problem requiring both general inference and very specific textual inferences. Instances of these problems are ubiquitous since almost nothing in a text can be taken at face value: Simple numbers (*43*) may actually denote amounts in millions of dollars; unknown proper names (*April Wednesday*) must be categorized from context; the amount a value has *plunged* will be orders of magnitude different whether it is the value of a stock or of a currency in the international markets. (The first two inferences can be guided by linguistic clues from the rest of the text, the third requires general knowledge.)

This kind of ability to make inferences from what is given literally in a text – "to read in between the lines" – is not a problem for people since we unconsciously recover the missing information from the context provided by the rest of the article, from what we know about the conventions of the author or the publication, and from the general stock of information we already have. In fact, most of the information people see in a text will be facts that they already know. These will be matters of common sense (e.g. all the information needed to make the passage sensible to, say, a three year old---or better, a martian), as well as facts and descriptions they have heard before. All that reading the passage does is bring the older information to mind and add or modify particulars.

In short, as they read the literal information a text contains, people bring a wealth of general and situational knowledge to bear, dramatically expanding the total amount of information the text conveys. Computer programs must do the same if they are to take away from what they read all that a human author intends.

The second problem, and the focus of this paper, is the problem of parsing. For present purposes I will define "parsing" as what a program does to determine the literal information a text contains, i.e. the information against which it can then bring its wealth of general knowledge to bear. This is a simpler problem than what some natural language understanding systems have taken as their task. Dyer's BORIS system (1983), for example, both recovered the literal information in the text and drew on background information that enabled it to infer much of what people reading the same passages are able to conclude by "reading between the lines".

There is, however, no principled limit on the scope of the inferences that can be drawn from reading a text.[3] This can greatly complicate the design of a system, since the kinds of knowledge involved in general inferences are very different from those needed to analyze linguistic structure, since they are dependent on just what background information is available and on how many assumptions one is willing to make in coming to conclusions. If we want to distinguish between "parsing", which can apply generally to any text, and "inference", which will vary according to the knowledge of the reader, then we must draw a line at some point. A logical place is just at the limit of extracting the information that is literally present, given knowledge of the meaning of the words and grammatical relationships.

---

[3] This was demonstrated in the thesis of Chuck Rieger (1974). The problem is that some inferences seem obvious and lexically driven, for instance that upon reading that "*John Doe was appointed CEO, succeeding Mary Roe, who retired*", we all know what position Ms. Roe used to hold. However, what principle would allow us to distinguish that kind of inference – on the basis of structural properties alone – from inferences such as that she no longer receives a salary from her old company, assumptions about her probable age, her possible interest in trust funds, etc. ?

As mentioned above, there has already been a vast amount of research on the parsing problem. But I will argue here that, with exceptions, the parsers that this research has produced are inadequate for the task, and that new parsers must be developed along very different lines.

All parsers, from Earley to Riesbeck to Marcus,[4] draw on roughly the same "surface linguistic" knowledge to do their analyses, i.e. knowledge of how words combine into phrases because of their positions, their morphology, identities, categories, etc. This much is not controversial; the real issue is what the output of the parser should be. Most parsers stop just with a structural description: a tree of nodes that dominate words that have been identified just by their part of speech. These nodes are labeled with syntactic categories (e.g. noun phrase, sentence), and information will be available in the tree about the grammatical relations among the nodes (e.g. subject, prepositional phrase adjunct, theme).

A few parsers, however, go further and recover information about "who did what to whom" and identify the intended sense of each word. Only this second kind of parser—one that recovers the semantic content of a text—will be of use to a text-based intelligent system.

To a certain extent, of course, this distinction in parsers is simply terminological. A parser that stops with a structural description can obviously be coupled with a semantic interpretation component that analyzes the structural description and the words to arrive at the same information as the second kind of parser produces directly (i.e. some set of concepts and the relations among them).

However, if one looks at the research goals of people working on the first kind of parser, for example Berwick and Weinberg (1984), they tend to be studying the consequences of using different theories of grammar or how to constrain structural syntactic ambiguities—not how information can be extracted from unrestricted, naturally occurring text to support a text-based intelligent system. Since different goals lead people to be interested in different problems and to adopt different approaches, it seems only realistic to expect that only research directly on how to understand unrestricted texts, and not simply on how to describe them linguistically, is most likely to yield an efficient design for a text-based intelligent system.

---

[4] These three people have each had a strong influence on the design of parsers. Jay Earley (1970) invented a standard algorithm for parsing with general context free grammars, establishing the basic efficiency parameters of the process and introducing a set of techniques that widely influenced the design of later parsers, even though it was oriented more towards computer languages than human, "natural" languages. Chris Riesbeck (1975) invented the first "conceptual analyzer", specifically for reading simple English stories. A conceptual analyzer uses heuristics to try to recover the concepts and relations expressed in a text, without particular concern for its syntactic form. Dyer's system is a quite sophisticated conceptual analyzer in terms of the kinds of conceptual knowledge it could employ, but its underlying parsing mechanisms are essentially the same as Riesbeck's and reflect progress within a unified school of thought. Mitch Marcus (1980) invented the so-called "wait and see" parser. His parser was "deterministic", in the special sense that it constructed only one analysis and never retracted any of its decisions (this is the notion of "indelibility", see McDonald 1984). It achieved this by waiting on its judgements until it has accumulated enough evidence to be certain. He recovered a description of the syntactic form of a text, designing his parser so that it could account for some of the psychological observations about the human parsing process.

2. Parsing unrestricted text

To be useful, our text-based intelligent system of the hopefully not too distant future will have to deal with the actual texts that people read. A business program should read the Wall Street Journal every day; it should not have to depend on a person to transcribe the Journal into some acceptable computer-pidgin. A concomitant requirement is that there can be no restrictions on what texts the system is prepared to process; it cannot depend on a person going through the Journal and passing it only those articles that it is likely to know how to handle, but must be able to process any text whatsoever, even if it understands nothing in it. (This capacity simply to get through a text without stopping because of a bug or a request to the user is sometimes known as "robustness".) Unrestricted text imposes two key goals for the design of a parser.

Goal #1: It must be a "partial" parser.

While there are a number of parsers today that can produce syntactic descriptions of unrestricted texts,[5] no parser even comes close to understanding everything in a real text such as a news article. While part of the problem is the lack of grammatical analyses for the breadth of constructions found in unrestricted texts, the real, fundamental problems are semantic and conceptual—today's computer programs simply are not able to make sense out of the bulk of the information in texts they are given to read because they lack the needed concepts and referents.

Instead of trying to understand everything, much of the research on parsing today is directed at the problem of extracting very specific information.[6] This has led to the concept of a "partial" parser, one that is specifically designed to recover a particular class of information while ignoring stretches of text on other subjects. A partial parser makes a careful and thorough analysis of the portions of text it is designed for, while skipping over the irrelevant portions. In addition, a partial parser is exceptionally careful that what it does analyze is accurate, i.e. that the analysis would not change if it had been able to made sense of additional parts of the text.

To be concrete, imagine that we have a text-based system that knows something about currency trading and that it sees in the second column of the front page of the Journal the lead text "*The dollar plunged, partly on the Treasury's report of an unexpectedly large deficit...*", (n.b. this is from a real text). It is entirely possible that the system could have the concepts needed to understand the main clause ("*the dollar plunged*"), while not having those needed for the adjunct ("*, partly on ...*"). On the basis of those concepts and referents the parser would be able to link the phrase "*the dollar*" to the object that represents U.S. currency in its domain model, rather than linking it to its unit for the quantity of money worth 100 cents or to some individual dollar bill, etc. Similarly it would be able to determine that the appropriate sense of

---

[5] While these parsers have, indeed, robustly analyzed tens of thousands of words of texts (e.g. Hindle 1983, de Marcken 1990) the analyses they produce are not complete. They produce a forest of phrases rather than complete sentences or larger units, e.g. NPs, verb groups, PPs, minimal clauses, etc. Given that they are making a conventional syntactic analysis, these fragment-sequence analyses are their only way to get a single, definitive analysis. If one insists on full-sentence analyses of unrestricted texts, then one must be prepared to accept potentially enormous ambiguity, sometimes many hundreds of analyses for a single 25 word sentence (Clippinger et al. 1982).

[6] Labels such as "information extraction" or "message processing" are often used to distinguish this research from what had been done before.

"*plunge*" is "fall in value against other nations' currency", rather than "dive into water" or simply "fall in value".

Within the adjunct, if we assume that the parser does not know the idiom "..<event>.. *on the report of...*", it will not be able to completely analyze the whole phrase (though it may well have a partial analysis). Nevertheless, to be effective the parser must still be reasonably sure that the information in the adjunct does not change the meaning of the main clause.

The kind of information just described is an example of what I mean by "literal" information—the proper target of a parser. On the other hand, knowing that the "plunge" by the dollar may have come to just 2 Yen, or that "unexpectedly large" deficit may have been more than 120 billion dollars, is a very different matter, since this information is something that only someone who understood this subject matter would know, and they would have to *already* know it if it is to come to mind when they read this text, since it is not given literally but is inferred—it is not be the parser's problem. These would be instances of the first problem, bringing general and situational knowledge to bear to understand more than the literal content of a text.

Goal #2: Coping with unknown words

As just defined, a partial parser is targeted for a specific kind of information and is designed to identify and analyze such information even when it appears embedded within a text that covers many more kinds of information than just that target. What this means is that we will have on the one hand a set of one or more algorithms and text processing mechanisms that are hopefully fairly general, and on the other a body of rules, a "grammar", that embodies the parser's knowledge of the target information and how it appears in a text.

The greater part of a topic-specific grammar will consist of rules about words, the "content words" that convey concepts and relationships that can be quite specific to the domain of interest. This will include not only the technical vocabulary that may be unique to the domain, but also subtle shadings to the meanings of ordinary words such as "old" and "new".[7]

If it has good coverage, the parser will also know most of the "function words"—e.g. "to", "does", "who", "a"—whose role in the language is not so much to identify concepts as to indicate grammatical relations. It may also know words from the vocabularies of very common subjects such as dates, measures, numbers, money, etc.

These words are "known" to the parser in the sense that it has specific rules that it applies when they appear in a text: starting a certain kind of phrase, retrieving a certain concept, establishing a certain relationship, etc. By this token, a word is "unknown" if the parser has no rules for handling it. Today, given what subjects are tractable for a parser to extract and what actually appears in a general text source like a newspaper or newswire, the vast bulk of the words a parser will encounter will be unknown.[8]

---

[7] In the domain of employment, consider the differences between "*the <u>new</u> position of assistant vice chairman*" and "*A <u>new</u> president wasn't named*". The new position is one that didn't exist before in the company's job roster. The new president will be new to the position but nothing about him or her per se will be new.

[8] Of course if all one wants from a parser is a description of the text's syntactic structure, then a machine-readable dictionary can supply information about part of speech (noun, verb, adjective, adverb) and a number, possibly a large number, of general definitions in natural language. But this

If all the unknown, off-topic words clustered together into their own portion of the text, then the partial parser's job would be simple: just skip over those parts. Unfortunately, this is anything but the case. Unknown words can appear anywhere, as unknown verbs between known noun phrases, unknown adjectives within noun phrases, etc. Even if all the content words in a sentence are unknown, it will still contain function words (an average of one every 3.5 words in the corpus described in Section 4), and these may activate the parser's mechanisms even though ultimately nothing will be extracted. How a partial parser copes with unknown works will tell a lot about the quality of its algorithms.

3. A Multi-Algorithm Partial Parser

In keeping with this goal of a parser that can accurately extract literal information from unrestricted texts, I have developed a system called "*Sparser*" that is designed to extract selected information from unrestricted text sources such as news articles in the Wall Street Journal.

I began work on this kind of parsing system in late 1987. An in-house production version using different techniques was used extensively at MAD Cambridge from mid-1988 through early September of 1989. The current version is a complete reimplementation done during the summer of 1990 in the course of about three months on a Mac II. Sparser is written in CommonLisp and has been ported to several unix platforms and several different vendor's CommonLisps. On the Mac II it presently runs at between 20 and 200 words per second depending on what layers of the system are being used.

As one would imagine, the design of this parser has much in common with other systems with similar goals, in particular the systems of Jacobs & Rau (1988), Riesbeck & Martin (1985), sublanguage-based systems like Sager's (1981), and in certain respects systems that produce a succession of phrases rather than sentences such as Hindle's (1983) or O'Shaughnessy's (1989).

In the rest of this section I will sketch the particular algorithms Sparser uses and the motivations behind them. A thorough discussion of the special properties of the primary parsing algorithm can be found in (McDonald, 1992).

3.1 Why multiple algorithms

Sparser has multiple parsing algorithms primarily because this makes it easier to write grammars. Information is distributed in a text in many different ways, and algorithms can be tailored to these differences according to the moments and forms with which the information becomes available and the way in which the rules for noticing the information are expressed.

An additional benefit comes from the ways a set of algorithms can be woven together. Given a careful control structure, there is no need to regiment them into a strict sequence or cascade; control can pass between the algorithms in reentrant loops, e.g.

---

leaves the parser with very large problem of ambiguity, and it does not provide meanings for the words in the form information extraction systems need them, since their goal is invariably to feed another computer program, today a database, but tomorrow an intelligent text-based system. Moreover, a great many unknown words will not be in a dictionary, some because of their rarity, but most because they are proper names—an open, seldom cataloged set.

information that is found by a later algorithm can lead to a reexamination of the text by a normally early algorithm, causing it now to draw conclusions that were unavailable to it the first time through.

This is especially true when unknown words are involved. A first pass by one class of algorithm may make observations based on weak phrase-internal evidence, the boundaries of the phrasal segment are then established by a second class of algorithm, and the final classification is made only much later once enough contextual evidence has been accumulated from other parts of the text for the judgement to be certain.

In the case of Sparser, the algorithms are incorporated into six interleaved components: a tokenizer, a set of transition networks, a context-free phrase structure parser, a context-sensitive parser, a conceptual analyzer, and a heuristic facility that hypothesizes phrasal boundaries using the grammatical properties of function words and partially-parsed phrases.

We will look at each in turn, starting with the simpler and more conventional, and then moving to the more heuristic algorithms designed specifically for coping with unknown words.

3.2 Tokenizing

Over all, a text parser is a transducer from the stream of characters that comprise the text to some abstract structured representation of the information it contains. Since characters are simply a means of representing words and punctuation, the first step in parsing is to segment the characters into orthographically sensible groups and look up the words that correspond to them, a process known as "tokenizing".

Sparser uses a very conservative tokenizing algorithm, grouping together contiguous sequences of characters of the same class, i.e. alphabetics, digits, or identical punctuation. Where most other tokenizers would see "$4.3 million" as two tokens or even one, Sparser sees it as six (including a reified "word" representing the space), leaving it to later stages to combine the minimal tokens into phrases and to categorize them when context can be taken into account. This delay can have considerable advantages. For example, the string "F-14" in one context could be the name of an airplane, in another a function key on a computer keyboard. Some systems make this judgement within their tokenizer, but I see that as a mistake since it commits the system prematurely, i.e. before any of the surrounding context can be taken into account. To Sparser's tokenizer the string is just the capital letter "F", a hyphen with no intervening spaces, and the number fourteen; to the next level of processing it is a still only a "hyphenated letter-number sequence" (in recognition of the fact that there are no spaces between the tokens), and it is left to the phrase structure components to apply contextual knowledge and make the substantive judgements.

The tokenizer is the level that first comes into contact with unknown words, i.e. any character sequence not already in its lookup table. Unknown does not mean without properties however. The tokenizer notices capitalization and the presence of affixes (-s, -ed, -ing). These properties define artificial words that can be reacted to by later algorithms just like a known word.

3.3 Word-level transition nets

When the words are known most phrases will be built up by noticing pairs of adjacent elements: specific words, words labeled by their categories, or already formed

labeled phrases. The question then arises as to what internal structure the phrases should have. Most of the phrases linguists study are best described as binary branching trees where each lower constituent is well-founded semantically and makes a disciplined composition with its neighbor word or phrase. However, other, less studied but extremely frequent phrase types such as proper names, numbers, and some word compounds involve little or no composition, and are best seen as flat structures. For parsing these, Sparser provides hooks for a transition net facility that operates over the stream of words output from the tokenizer.

In principle, the phrase types could be handled with a minimal impact on efficiency by the phrase structure rules that operate at the next stage. The transition nets are used because it is a more natural description for the grammar writer to choose. Similarly, other researchers have used much more elaborate pattern matching facilities at the word level,[9] while I have judged that the kinds of things these complex patterns search for are more easily stated in terms of phrase structure rules and handled at a later stage.

3.4 Phrase structure parsing using semantic labels

The phrase structure parser is driven by a grammar of conventional rewrite rules and constructs trees of phrases stored in a chart. Its algorithm (McDonald 1992) is an unusually efficient deterministic variation on bottom-up parsing, and, like all phrase structure algorithms, it is reacting to patterns of adjacent labeled constituents.

What is usual about Sparser's rewrite rules is that they employ primarily semantic labels on phrases rather than syntactic. That is, rather than label the text "*the dollar plunged*" as a clause consisting of a noun phrase (consisting of a determiner and a noun) followed by a verb phrase consisting of a single verb, Sparser's grammar will label *the dollar* as, e.g., "currency" and *plunged* as a verb of motion.[10] The syntactic labeling is of course correct, but it is not to the point. Sparser's goal is to extract semantically characterized information, not to produce a syntactic structural description.

This approach to the choice of labels integrates the process of disambiguating word senses directly with the process of analyzing grammatical relations and forming phrases: a phrase will only be formed if its elements are semantically, as well as syntactically, consistent. This approach also dramatically reduces the number of structural ambiguities ever considered.

This kind of design was first worked out by Burton and Brown (1979), and while considered an effective means of engineering a grammar for a small domain, it never gained support as a general strategy for syntactic parsers. This historical fact reflects the general trend in computational linguistics to see parsing as an isolated process just concerned with the recovery of a text's structure. In the context of text-based intelligent

---

[9] Often full regular expressions are allowed, with boolean combinations including optional elements, wildcards of stipulated length, variable bindings, etc., even multipass cascades to resolve "meta-tokens"; see e.g. Masand & Duffey 1985.

[10] Actually, Sparser's labeling scheme is more complex than this. Every parse node has three "label" fields. The primary label is taken from a semantic vocabulary, but it is shadowed by a conventional syntactic label that is used in default rules, and there is also the category of the phrase's denotation. For the case of *dollar*, its primary label is "currency", but it also has the label "head noun", and a structured ambiguity in its denotation between "physical object", "quantity of money worth 100 cents", and "the value of U.S. currency on international money markets". This ambiguity is resolved by the propagation of semantic constraints as phrases are formed, e.g., only a "value" is able to move and so the last sense is the one taken when a currency is combined with a verb like *plunged*.

systems, however, this view is myopic. The purpose of a parser is to facilitate the recovery the information a text contains, and any technique that speeds that effort is to the point. By folding interpretation and structural analysis together through the use of semantic labels, one is insured that every phrase that is allowed to complete syntactically will never be rejected semantically—a dramatic savings in efficency at the cost of a multiplication in the number of rules in the grammar, which is a tradeoff easily made in today's computational architectures.

3.5 Context-sensitive phrase structure parsing

In a context-free phrase structure rewrite rule, a sequence of labels is matched against the labels of adjacent constituents in the text. If the match succeeds ("completes"), the entire sequence of constituents is composed into a new phrase and given the label that the rule dictates. A context-sensitive rule can be seen as a matching operation against a sequence of adjacent constituents, with the exception that now only one of those constituents is relabeled rather than the whole sequence.

Context sensitive rules are used extensively in Sparser's grammars. They allow information to be accumulated gradually. Early rules can be cautious in their assumptions about what a phrase denotes (as in "hyphenated letter-number sequence" or "name"), waiting for a context to accumulate around the phrase through the action of other rules. Once the context is established, a context sensitive rule will be triggered to enrich the categorization, e.g. name -> person / ____ <was named CEO>.

3.6 Parsing non-adjacent constituents with a conceptual analyzer

The conceptual analyzer algorithm is responsible for composing constituents that are not adjacent (and so are invisible to the phrase structure algorithm), but which can be linked on semantic grounds based on how the constituents are labeled. The primary function of this component of the parser is to make it possible to "skip over" regions of the text that are outside of the system's competence.

Consider the following text from the perspective of a grammar of employment change. In bold are the text segments that the parser understands; a relative clause (plain text) intervenes between the person that is the subject of the sentence and the "appointment" verb phrase that it should combine with.

> .. ***Robert A. Beck, a 65-year-old former Prudential chairman*** *who originally bought the brokerage firm,* ***was named chief executive of Prudential Bache***. ...

Even though the grammar contains a rule for it, the phrase structure algorithms cannot compose these segments because they are not adjacent. Instead, the conceptual analyzer algorithm, triggered by these "stranded" constituents, uses the grammar rule to define a search path to try and join them. In this case, the appointment verb phrase searches leftward through the partial constituents until it finds a person constituent to compose with. The search is constrained to fit the grammatical relation this rule instantiates, namely that of subject to predicate in a clause. This means that it will fail if the search extends beyond the current sentence or across constituents that could not be interposed between a subject and its predicate verb phrase.

The notion of a conceptual analyzer (CA) was developed by Roger Schank and his students through the 1970s at Yale (for a practical summary, see Birnbaum & Selfridge (1981)). Sparser's version is based on the same philosophy—the composition of

phrases based on their semantic ("conceptual") characteristics rather than syntactic—with the significant difference that Sparser's CA is operating in conjunction with a set of other parsing algorithms, and so is not also responsible for assembling the small, syntactically rich phrases where traditional techniques excel. In the traditional CA design, all parsing is done by searching for patterns of semantic elements. This leads to stating simple syntactic facts in a complicated semantic pattern language, again an unnecessary burden on the grammar writer.

3.7 Forming phrases heuristically

Working together, these algorithms can efficiently recover a description of the portion of a text where the words are known. Where they are not, a heuristic facility is drawn on to deduce as much as possible. This facility uses the linguistic properties of items such as inflectional and derivational morphemes (e.g. '+tion', '+ed') and function word vocabulary (e.g. *is*, *from*, *and*) to deduce the boundaries of phrases even though it does not recognize most of their words. For example when the function word *the* is seen it knows that that must be the start of a noun phrase; an auxiliary verb like *does* or a preposition terminates whatever phrase came before it.

This basis for segmenting a text can be combined with the phrase structure rules to allow phrases to be formed that would otherwise be missed. Consider the real example "... ***this*** *gold mining* ***company*** *was* ...". Two unknown words keep the conventional algorithms from forming this noun phrase, but in its context the function-word driven heuristics guarantee that it all four words are in the same phrase. Given that there is a phrase structure rule in the grammar that would have combined *this* and *company* had they been adjacent, we are entitled to span this entire phrase using that same rule. This will give us the label for the phrase, allowing it to be passed to the adjacency-driven algorithms for composition with its neighbors, even though we don't understand all of what it means. These techniques complement the conceptual analyzer, since they make constituents adjacent (and so accessible the phrase structure rules, the backbone of Sparser's operation) by allowing constituents to form even when they include unknown words, while the conceptual analyzer searches across unparsed text segments for which no analysis was possible.

Similar techniques are used by Hindle (1983) and O'Shaughnessy (1989), where, as in this case, the emphasis is on recovering a relatively robust parse of successive phrases, rather than insisting on a full parse of entire sentences, with the concomitant problems of structural ambiguities discussed above.

4. Results in practice

The real test of any parser architecture is in how well the grammars written for it perform in realistic tests. Of course some of the results will be a function of the linguistic skills of the grammar writer, but the relative ease of use of the parser's rule notations will always be a key factor.

Sparser was put to its first significant test in May of 1991 with a relatively large grammar for extracting information on people changing jobs. 203 previously unseen articles from the Wall Street Journal were analyzed by the parser and the results compared against a human analyst's judgements. The articles were literally the second half of all articles in the Journal that were labeled with the tag "WNEWS" during the month of February. They included short articles specifically about job changes, as well

as long columns and even features where the information was incidental. Roughly three man months had gone into preparing the grammar for this domain, resulting in approximately 2300 rules, though these were known to be insufficient to account for many of the cases found in the training set.

The test was to identify all possible instances of the four-tuple: person, company or subsidiary, title, and type of event (e.g. appointment, retirement). In the texts, these corresponded to every clause with a relevant verb, as well as redundant cases of nominalizations and anaphoric references.

The results were strong, considering the amount of time invested. Four out of five of all possible events were correctly identified (597/735). Of the events identified, four out of five were completely correct in all four fields (486/597), with the bulk of the deficit in omissions (empty fields) rather than actual mistakes. The false positive rate was 3%.

5. Directions

It is unlikely that even with a great amount of work extending the present kind of grammar that the success rate on this task could be brought close to 100% (discounting, of course, texts that a person wouldn't understand either). Previously unseen verbs and new titles will always continue to be seen, and even the most carefully edited texts include occasional errors. The only way to surmount these problems would be by adding a new kind of component, one that deduced the category, if not the full meaning, of unknown or mistaken words.

This is not an intrinsically difficult problem (see, for example Hirschman (1986) and other papers in the same volume). But it can be very much easier or harder depending on the architecture of the parser. There must be an explicit model of the relationships among the grammar's rules, and the rules must be applied in a conservative, monotonic manner or else there will not be an adequate representation (the intermediate states of the parser) to run word-induction heuristics over. Sparser is consistent with these properties, and some early experiments have been promising.

Another area of concern is the amount of labor needed to assemble a grammar for a new domain when using semantic instead of syntactic labels. Once the idiosyncratic constructions like dates or money are discounted, one syntactic grammar will cover nearly all texts, and it will need only hundreds of rules rather than the thousands required for *each* topic area when semantic labels are used as the lexical and phrasal categories.

However, to a certain extent, this is just a consequence of moving into the phrase structure rules much of the apparatus that would have had to be in the total system anyway, though with a syntactic grammar it would have appeared in the "semantic interpretation component" rather than in Sparser's rules. As the goal is to understand the text and not simply describe it, the full set of categorial distinctions and composition schemas of the domain is required and having one or two orders of magnitude more rules is inevitable.

Nevertheless, one would like the process of adding a new domain to go more quickly, and we are investigating two possibilities. The first is to tightly couple the grammar writing process to the process of defining the concepts, relations, operations, etc. of the actual semantic model and reasoning methods of the new domain. We have taken some initial steps in this direction (McDonald 1991), at the same time tieing in the

capability of reversing the process and generating texts from the objects of the conceptual model as well as parsing to them.

The second possibility is to collapse many of the rules of linguistic composition together by moving a certain class of the categorial distinctions into the semantic structures that accompany Sparser's parse nodes (see footnote 10), having the differences that only affect interpretation without differentiating the possibilities for composition operate at a different level (though at the same time) as the parsing rules for assembling nodes out of constituents. This work is being done in collaboration with James Pustejovsky using his theory of the generative lexicon (Pustejovsky 1991).

6. References

Berwick, Robert & Amy Weinberg (1984) *The Grammatical Basis of Linguistic Performance*, MIT Press, Cambridge, MA.

Birnbaum, Larry & Malory Selfridge (1981) "Conceptual Analysis of Natural Language", in Schank & Reisbeck (eds.) *Inside Computer Understanding*, Lawrence Erlbaum, Hillsdale, New York.

Borning, Alan, Doug Lenat, David McDonald, Craig Taylor, & Steve Weyer (1983) "KNOESPHERE: Building Expert Systems with Encyclopedic Knowledge", in the Proceeding of IJCAI-83, Karslruhe, Germany; available from Morgan Kaufmann Publishers.

Burton, Richard R. & John Seeley Brown (1979) "Towards a natural-language capability for computer-assisted instruction", in O'Neil (ed.) *Procedures for Instructional Systems Development*, Academic Press, New York.

Clippinger, John H., Peter Szolovitz, David McDonald, Ken Church, Glen Burke (1982) *Final Report: Project to Develop a Prototype Artificial Intelligence System with a Rapid Automated Intelligence Gathering and Analysis Capability for Natural Language Texts* USACECOM; Contract No. DAAB07-82-J070.

de Marcken, Carl G. (1990) "Parsing the LOB Corpus" Proc. 21st Annual Meeting of the Association for Computational Linguistics, June 6-10, Univ. of Pittsburgh, 243-251.

Dyer, Michael (1983) *In-Depth Understanding*, MIT Press, Cambridge, Massachusetts.

Earley, Jay (1970) "An efficient context-free parsing algorithm" *Communications of the ACM* 13(2) February 1970, 94-102.

Grosz, Barbara, Karen Sparck Jones, Bonnie Webber (eds.) (1985) *Readings in Natural Language Processing*, Morgan Kaufmann, Los Altos, California.

Hindle, Don (1983) "Deterministic Parsing of Syntactic Non-fluencies", Proc. 21st Annual Meeting of the Association for Computational Linguistics, June 15-17, 1983, MIT, 123-128; in reference to his parser *Fidditch*.

Hirschman, Lynette (1986) "Discovering Sublanguage Structures", in Grishman & Kittredge (eds.) *Analyzing Language in Restricted Domains*, Lawrence Erlbaum, Hillsdale, New Jersey.Hindle, Don (1983) "Deterministic Parsing of Syntactic Non-fluencies", Proc. 21st Annual Meeting of the Association for Computational Linguistics, June 15-17, 1983, MIT, 123-128; in reference to his parser *Fidditch*.

Lenat, Doug & R.V. Guha (1990) *Building Large Knowledge-Based Systems*, Addison Wesley, Reading Massachusetts.

Marcus, Mitch (1980) *Theory of Syntactic Recognition for Natural Language*, MIT Press, Cambridge, MA.

Masand, Brij & Roger Duffey (1985) "A Rapid Prototype of an Information Extractor and its Application to Database Table Generation", working paper, Brattle Research Corporation, Cambridge Massachusetts.

McDonald, David (1991) "Reversible NLP by Deriving the Grammars from the Knowledge Base", Proc. Workshop on Reversible Grammar in NLP, June 17, 1991, Berkeley, pp.40-44, available from the ACL.

_____ (1992) "An Efficient Chart-based Algorithm for Partial Parsing of Unrestricted Texts" Proc. 3d Conference on Applied Natural Language Processing (ACL), April 1-3, Trento, Italy.

O'Shaughnessy (1989) "Parsing with a Small Dictionary for Applications such as Text to Speech", *Computational Linguistics* 15(3) June 1989, 97-108.

Pustejovsky, James (1991) “The Generative Lexicon”, *Computational Linguistics* 17(4), pp.409-441.

Rau, Lisa & Paul Jacobs (1988) "Integrating Top-Down and Bottom-up Strategies in a Text Processing System", Proc. 2d Conf. on Applied Natural Language Processing (Assoc. Comp. Ling.), February 9-12, 1988, Austin Tx., 129-135.

Rieger, Charles (1975) “Conceptual Memory and Inference”, in Schank (ed.) *Conceptual Information Processing*.

Riesbeck, Chris (1975) “Conceptual Analysis”, in Schank (ed.) *Conceptual Information Processing*.

_____ & C.E. Martin (1985) “Direct Memory Access Parsing” technical report DCS/RR 354, AI Group, Yale University.

Sager, Naomi (1981) *Natural Language Information Processing*, Addison-Wesley, Reading, MA.

Schank, Roger (ed.) (1975) *Conceptual Information Processing*, American Elsevier, New York.