Parsing English with a Link Grammar

Daniel D. Sleator[*] and Davy Temperley[†]

[*]School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
email: `sleator@cs.cmu.edu`

[†]Music Department
Columbia University
New York, NY 10027
email: `dt3@cunixa.cc.columbia.edu`

Abstract

We define a new formal grammatical system called a *link grammar*. A sequence of words is in the language of a link grammar if there is a way to draw *links* between words in such a way that (1) the local requirements of each word are satisfied, (2) the links do not cross, and (3) the words form a connected graph. We have encoded English grammar into such a system, and written a program (based on new algorithms) for efficiently parsing with a link grammar. The formalism is lexical and makes no explicit use of constituents and categories. The breadth of English phenomena that our system handles is quite large. A number of sophisticated and new techniques were used to allow efficient parsing of this very complex grammar. Our program is written in C, and the entire system may be obtained via anonymous ftp. Several other researchers have begun to use link grammars in their own research.

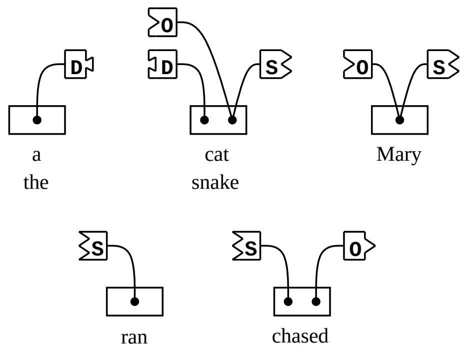
1 Introduction

Most sentences of most natural languages have the property that if arcs are drawn connecting each pair of words that relate to each other, then the arcs will not cross [10, p. 36]. This well-known phenomenon, which we call *planarity*, is the basis of *link grammars* our new formal language system.

A link grammar consists of a set of *words* (the terminal symbols of the grammar), each of which has a *linking requirement*. A sequence of words is a *sentence* of the language defined by the grammar if there exists a way to draw *links* among the words so as to satisfy the following conditions:

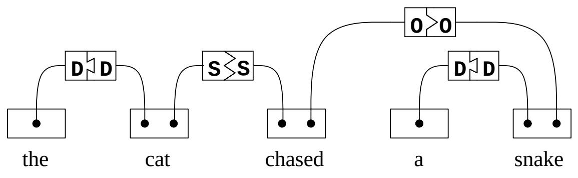
Planarity:	The links do not cross (when drawn above the words).
Connectivity:	The links suffice to connect all the words of the sequence together.
Satisfaction:	The links satisfy the linking requirements of each word in the sequence.

The linking requirements of each word are contained in a *dictionary*. To illustrate the linking requirements, the following diagram shows a simple dictionary for the words *a*, *the*, *cat*, *snake*, *Mary*, *ran*, and *chased*. The linking requirement of each word is represented by the diagram above the word.

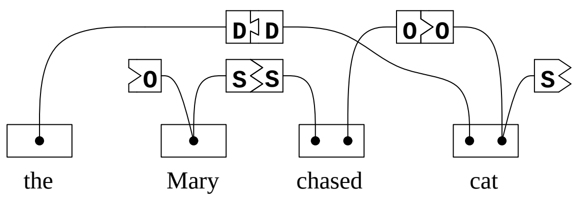


Each of the intricately shaped labeled boxes is a *connector*. A connector is satisfied by matching it to a compatible connector (one with the appropriate shape, facing in the opposite direction). Exactly one of the connectors attached to a given black dot must be satisfied (the others, if any, must not be used). Thus, *cat* requires a D connector to its left, and either an O connector to its left or a S connector to its right. Plugging a pair of connectors together corresponds to drawing a link between that pair of words.

The following diagram shows how the linking requirements are satisfied in the sentence *The cat chased a snake*.

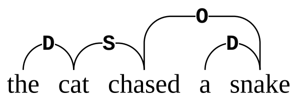


(The unused connectors have been suppressed here.) It is easy to see that *Mary chased the cat*, and *the cat ran* are also sentences of this grammar. The sequence of words: *the Mary chased cat* is not in this language. Any attempt to satisfy the linking requirements leads to a violation of one of the three rules. Here is one attempt:



Similarly *ran Mary*, and *cat ran chased* are not part of this language.

A set of links that prove that a sequence of words is in the language of a link grammar is called a *linkage*. From now on we'll use simpler diagrams to illustrate linkages. Here is the simplified form of the diagram showing that *the cat chased a snake* is part of this language:



We have a succinct, computer-readable notation for expressing the dictionary of linking requirements. The following dictionary encodes the linking requirements of the previous example.

words	formula
a the	`D+`
snake cat	`D- & (O- or S+)`
Mary	`O- or S+`
ran	`S-`
chased	`S- & O+`

The linking requirement for each word is expressed as a formula involving the operators `&`, and `or`, parentheses, and connector names. The `+` or `-` suffix on a connector name indicates the direction (relative to the word being defined) in which the matching connector (if any) must lie. The `&` of two formulas is satisfied by satisfying both the formulas. The `or` of two formulas requires that exactly one of its formulas be satisfied. The order of the arguments of an `&` operator is significant. The farther left a connector is in the expression, the nearer the word to which it connects must be. Thus, when using *cat* as an object, its determiner (to which it is connected with its `D-` connector) must be closer than the verb (to which it is connected with its `O-` connector).

We can roughly divide our work on link grammars into three parts: the link grammar formalism and its properties, the construction of a wide-coverage link grammar for English, and efficient algorithms and techniques for parsing link grammars. We now touch briefly on all three of these aspects.

Link grammars are a new and elegant context-free grammatical formalism[1][2], and have a unique combination of useful properties:

1. In a link grammar each word of the lexicon is given a definition describing how it can be used in a sentence. The grammar is distributed among the words. Such a system is said to be *lexical*. This has several important advantages. It makes it easier to construct a large grammar, because a change in the definition of a word only affects the grammaticality of sentences involving that word. The grammar can easily be constructed incrementally. Furthermore, expressing the grammar of the irregular verbs of English is easy – there's a separate definition for each word.

   Another nice feature of a lexical system is that it allows the construction of useful probabilistic language models. This has led researchers to construct lexical versions of other grammatical systems, such as tree-adjoining grammars [13]. Lafferty and the present authors have also constructed such a probabilistic model for link grammars [11].

2. Unlike a phrase structure grammar, after parsing a sentence with a link grammar words that are associated semantically and syntactically are directly linked. This makes it easy to enforce agreement, and to gather statistical information about the relationships between words.

3. In English, whether or not a noun needs a determiner is independent of whether it is used as a subject, an object, or even if it is part of a prepositional phrase. The algebraic notation we developed for expressing a link grammar takes advantage of this orthogonality. Any lexical grammatical system, if it is to be used by a human, must have such a capability. In our current on-line dictionary the word *cat* can be used in 369 different ways, and for *time* this number is 1689. A compact link grammar formula captures this large number of possibilities, and can easily be written and comprehended by a human being.

4. Another interesting property of link grammars is that they have no explicit notion of constituents or categories. In most sentences parsed with our dictionaries, constituents can be seen to emerge as contiguous connected collections of words attached to the rest of the sentence by a particular type of link. For example in the dictionary above, `S` links always attach a noun phrase (the connected collection of words at the left end of the link) to a verb (on the right end of the link). `O` links work in a similar fashion. In these cases the links of a sentence can be viewed as an alternative way of specifying the constituent structure of the sentence. On the other hand, this is not the way we think about link grammars, and we see no advantage in taking that perspective.

Our second result is the construction of a link grammar dictionary for English. The goal we set for ourselves was to make a link grammar that can distinguish, as accurately as possible, syntactically correct English sentences from incorrect ones. We chose a formal or newspaper-style English as our model. The result is a link grammar of roughly eight hundred definitions (formulas) and 25000 words that captures many phenomena of English grammar. It handles: noun-verb agreement, questions, imperatives, complex and irregular verbs, many types of nouns, past- or present-participles in noun phrases, commas, a variety of adjective types, prepositions, adverbs, relative clauses, possessives, coordinating conjunctions, unbounded dependencies, and many other things.

The third result described in this paper is a program for parsing with link grammars. The program reads in a dictionary (in a form very similar to the tables above) and will parse sentences according to the given grammar. The program does an exhaustive search – it finds every way of parsing the given sequence with the given link grammar. It is based on our own $O(n^3)$ algorithm (n is the number of words in the sentence to be parsed). The program also makes use of several very effective data structures and heuristics

[1]Link grammars resemble dependency grammars and categorial grammars. There are also many significant differences. Some light is shed on the relationship in section 6.

[2]The proof of the context-freeness of link grammars is not included in this paper, but appears in our technical report [14]. Note that context-free systems can differ in many ways, including the ease with which the same grammar can be expressed, the efficiency with which the same grammar can be parsed, and the usefulness of the output of the parser for further processing.

to speed up parsing. The program is comfortably fast – parsing typical newspaper sentences in a few seconds on a modern workstation.

Both our program (written in ANSI-C) and our dictionary are available via anonymous ftp through the internet.[3] Having the program available for experimentation may make it easier to understand this paper.

The organization of this paper

In section 2 we define link grammars more formally and explain the notation and terminology used throughout the rest of the paper. In section 3 we describe the workings of a small link grammar for English. Our $O(n^3)$ algorithm is described in section 4, and the data structures and heuristics that make it run fast are described in section 5. In section 6 we explain the relationship between link grammars, dependency syntax, and categorial grammars. We show how to automatically construct a link grammar for a given categorial grammar. This construction allows our efficient parsing algorithms and heuristics to be applied to categorial grammars. Section 7 mentions several other research projects that are based on link grammars.

Space limitations prevent us from presenting details of a number of other aspects of our work. The following paragraphs mention a few of these. More details on all of these matters are contained in our technical report [14].

There are a number of common English phenomena that are not handled by our current system. Our technical report contains a list of these, along with the reason for this state of affairs. The reasons range from the fact that ours is a preliminary system to the fact that some phenomena simply do not fit well into the link grammar framework.

Coordinating conjunctions such as *and* pose a problem for link grammars. This is because in a sentence like *The dog chased and bit Mary* there should logically be links between both *dog* and *bit* and *chased* and *Mary*. Such links would cross. We have devised a scheme that handles the vast majority of uses of such conjunctions and incorporated it into our program. The existence of such a conjunction in a sentence modifies the grammar of the words in it. The same parsing algorithm is then used on the resulting modified grammar.

Certain other constructs are difficult to handle only using the basic link grammar framework. One example is the non-referential use of *it*: *It is likely that John will go* is correct, but *The cat is likely that John will go* is wrong. It is possible – but awkward – to distinguish between these with a link grammar. To deal with this (and a number of other phenomena), we extended the basic link grammar formalism with a *post-processor* that begins with a linkage, analyzes its structure, and determines if certain conditions are satisfied. This allows the system to correctly judge a number of subtle distinctions (including that mentioned here).

2 Notation and terminology

2.1 Meta-rules

The link grammar dictionary consists of a collection of entries, each of which defines the linking requirements of one or more words. These requirements are specified by means of a *formula* of *connectors* combined by the binary associative operators `&` and `or`. Presidence is specified by means of parentheses. Without loss of generality we may assume that a connector is simply a character string ending in `+` or `-`.

When a link connects to a word, it is associated with one of the connectors of the formula of that word, and it is said to *satisfy* that connector. No two links may satisfy the same connector. The connectors at opposite ends of a link must have names that *match*, and the one on the left must end in `+` and the one on the right must end in `-`. In basic link grammars, two connectors match if and only if their strings are the same (up to but not including the final `+` or `-`). A more general form of matching will be introduced later.

The connectors satisfied by the links must serve to satisfy the whole formula. We define the notion of satisfying a formula recursively. To satisfy the `&` of two formulas, both formulas must be satisfied. To satisfy the `or` of two formulas, one of the formulas must be satisfied, and *no* connectors of the other formula may be satisfied. It is

[3]The directory is `/usr/sleator/public` on the host `spade.pc.cs.cmu.edu` (128.2.209.226). Our technical reports [14, 11] are also available there.

sometimes convenient to use the empty formula ("()"), which is satisfied by being connected to no links.

A sequence of words is a *sentence* of the language defined by the grammar if there exists a way to draw links among the words so as to satisfy each word's formula, and the following *meta-rules*:

Planarity: The links are drawn above the sentence and do not cross.

Connectivity: The links suffice to connect all the words of the sequence together.

Ordering: When the connectors of a formula are traversed from left to right, the words to which they connect proceed from near to far. In other words, consider a word, and consider two links connecting that word to words to its left. The link connecting the nearer word (the shorter link) must satisfy a connector appearing to the left (in the formula) of that of the other word. Similarly, a link to the right must satisfy a connector to the left (in the formula) of a longer link to the right.

Exclusion: No two links may connect the same pair of words.

2.2 Disjunctive form

The use of formulas to specify a link grammar dictionary is convenient for creating natural language grammars, but it is cumbersome for mathematical analysis of link grammars, and in describing algorithms for parsing link grammars. We therefore introduce a different way of expressing a link grammar called *disjunctive form*.

In disjunctive form, each word of the grammar has a set of *disjuncts* associated with it. Each disjunct corresponds to one particular way of satisfying the requirements of a word. A disjunct consists of two ordered lists of connector names: the *left list* and the *right list*. The left list contains connectors that connect to the left of the current word (those connectors end in `-`), and the right list contains connectors that connect to the right of the current word. A disjunct will be denoted:

$$((L_1, L_2, \ldots, L_m)\ (R_n, R_{n-1}, \ldots, R_1))$$

Where $L_1, L_2, \ldots, L_m$ are the connectors that must connect to the left, and $R_1, R_2, \ldots, R_n$ are connectors that must connect to the right. The number of connectors in either list may be zero. The trailing `+` or `-` may be omitted from the connector names when using disjunctive form, since the direction is implicit in the form of the disjunct.

To satisfy the linking requirements of a word, one of its disjuncts must be satisfied (and no links may attach to any other disjunct). To satisfy a disjunct all of its connectors must be satisfied by appropriate links. The words to which L_1, L_2, ... are linked are to the left of the current word, and are monotonically increasing in distance from the current word. The words to which R_1, R_2, ... are linked are to the right of the current word, and are monotonically increasing in distance from the current word.

It is easy to see how to translate a link grammar in disjunctive form to one in standard form. This can be done simply by rewriting each disjunct as

$$(\ L_1\ \texttt{\&}\ L_2\ \texttt{\&} \cdots \texttt{\&}\ L_m\ \texttt{\&}\ R_1\ \texttt{\&}\ R_2\ \texttt{\&} \cdots \texttt{\&}\ R_n\ )$$

and combining all the disjuncts together with the `or` operator to make an appropriate formula.

It is also easy to translate a formula into a set of disjuncts. This is done by enumerating all ways that the formula can be satisfied. For example, the formula:

```
(A- or ()) & D- & (B+ or ()) & (O- or S+)
```

corresponds to the following eight disjuncts:

```
  ((A,D)   (S,B))
((A,D,O)   (B))
  ((A,D)   (S))
((A,D,O)   ())
    ((D)   (S,B))
  ((D,O)   (B))
    ((D)   (S))
  ((D,O)   ())
```

2.3 Our dictionary language

To streamline the difficult process of writing the dictionary, we have incorporated several other features to the dictionary language. Examples of all of these features can be found in section 3.

It is useful to consider connector matching rules that are more powerful than simply requiring the strings of the connectors to be identical.

The most general matching rule is simply a table – part of the link grammar – that specifies all pairs of connectors that match. The resulting link grammar is still context-free.

In the dictionary presented later in this paper, and in our larger on-line dictionary, we use a matching rule that is slightly more sophisticated than simple string matching. We shall now describe this rule.

A connector name begins with one or more upper case letters followed by a sequence of lower case letters or `*`s. Each lower case letter (or `*`) is a *subscript*. To determine if two connectors match, delete the trailing `+` or `-`, and append an infinite sequence of `*`s to both connectors. The connectors match if and only if these two strings match under the proviso that `*` matches a lower case letter (or `*`).

For example, `S` matches both `Sp` and `Ss`, but `Sp` does not match `Ss`. Similarly, `D*u`, matches `Dmu` and `Dm`, but not `Dmc`. All four of these connectors match `Dm`.

The formula "`((A- & B+) or ())`" is satisfied either by using both `A-` and `B+`, or by using neither of them. Conceptually, then, the the expression "`(A+ & B+)`" is optional. Since this occurs frequently, we denote it with curly braces, as follows: `{A+ & B+}`.

It is useful to allow certain connectors to be able to connect to one or more links. This makes it easy, for example, to allow any number of adjectives to attach to a noun. We denote this by putting an "`@`" before the connector name, and call the result a *multi-connector*.

Our dictionaries consist of a sequence of *entries*, each of which is a list of words separated by spaces, followed by a colon, followed by the formula defining the words, followed by a semicolon.

3 An example

Perhaps the best way to understand how to write a link grammar for English is to study an example. The following dictionary does not cover the complete grammar of the words it contains, but it does handle a number of phenomena: verb-noun agreement, adjectives, questions, infinitives, prepositional phrases, and relative clauses.

```
the:      D+;
a:        Ds+;
John Mary:
  J- or O- or (({C- or CL-} & S+) or SI-);
dog cat park bone stick:
  {@A-} & Ds-
  & {@M+ or (C+ & Bs+)}
  & (J- or O- or ({C- or CL-} & Ss+) or SIs-);
dogs cats parks bones sticks:
  {@A-} & {Dm-}
  & {@M+ or (C+ & Bp+)}
  & (J- or O- or ({C- or CL-} & Sp+) or SIp-);
has:
  (SIs+ or Ss- or (Z- & B-))
  & (((B- or O+) & {@EV+}) or T+);
did:
  (SI+ & I+)
  or ((S- or (Z- & B-))
     & (((B- or O+) & {@EV+}) or I+));
can may will must:
  (SI+ or S- or (Z- & B-)) & I+;
is was:
  (Ss- or (Z- & Bs-) or SIs+)
  & (AI+ or O+ or B- or V+ or Mp+);
touch chase meet:
  (Sp- or (Z- & Bp-) or I-)
  & (O+ or B-) & {@EV+};
touches chases meets:
  (Ss- or (Z- & Bs-)) & (O+ or B-) & {@EV+};
touched chased met:
  (V- or M-
    or ((S- or (Z- & B- ) or T-) & (O+ or B-)))
  & {@EV+};
touching chasing meeting:
  (GI- or M-) & (O+ or B-) & {@EV+};
die arrive:
  (Sp- or (Z- & Bp-) or I-) & {@EV+};
dies arrives:
  (Ss- or (Z- & Bs-)) & {@EV+};
died arrived:
  (S- or (Z- & B-) or T-) & {@EV+};
dying arriving:
  (GI- or M-) & {@EV+};
with in by:
  J+ & (Mp- or EV-);
big black ugly:
  A+ or (AI- & {@EV+});
who:
  (C- & {Z+ or CL+}) or B+ or Ss+;
```

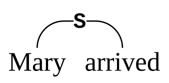
3.1 Some Simple Connectors

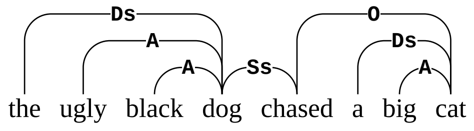
We develop an explanation of how this works in stages. Let's first restrict our attention to the following connectors: `S`, `O`, `A`, `D`. (Imagine the dictionary with all of the other connectors removed.)

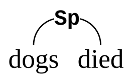
The `S` is used to connect a noun to its verb. The `O` connector is used to connect a verb to its object. The `A` connector is used to connect an adjective to its noun. The `D` is for connecting a determiner to its noun. Notice that this connector is omitted from proper nouns, is optional on plural nouns, and is mandatory on singular nouns. Also notice that the subscripts allow *the* to act as the determiner for both plural and singular nouns, but *a* can only work with the singular nouns. Similarly, the `S` connector is subscripted to ensure verb-noun agreement.

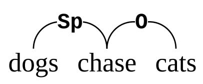
The ordering of the terms in these expressions is often important. For example, the fact that on nouns, the `A-` occurs to the left of the `D-` means that the adjective must be closer to the noun than the determiner.

Here are some judgements that can be rendered by what we have described so far:





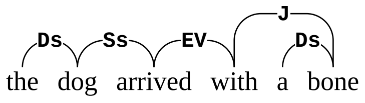


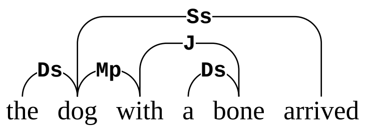


*a dog chase a cat
*black the dog died
*a/*the Mary chased the cat
*a dogs died
*dog died

3.2 Prepositions

The `J`, `M` and `EV` connectors allow prepositional phrases. The `J` connector connects a preposition to its object. Notice that in nouns, the `J-` is an alternative to the `O-`. This means that a noun cannot both be an object of a verb and of a preposition. The `M` connector is used when a prepositional phrase modifies a noun and the `EV` connector is used when a prepositional phrase modifies a verb. The following two examples illustrate this:

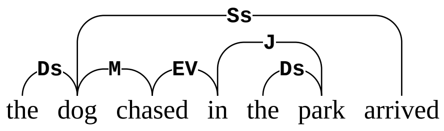


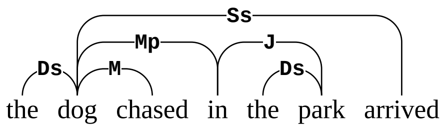


Notice that, as with `A-` connectors on nouns, a `@` is used for `M-` connectors on nouns and `EV-` connectors on verbs, allowing multiple prepositional phrases, such as *John chased a dog in the park with a stick.*

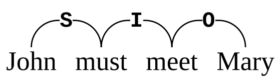
3.3 Participles

The `M-` connector on *chased* allows it to act as a participle phrase modifying a noun, as shown in these examples.





The `I` connector is used for infinitives, as in:



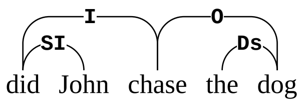
Notice that the `I` connector is an alternative to the `S` connector on plural verb forms. Thus we take advantage of the fact that plural verb forms are usually the same as infinitive forms, and include them both in a single dictionary entry.

In a similar way, the `T` connector is used for past participles. Past participles have a `T-`; forms of the verb *have* have a `T+`. The `GI` connector is used for present participles. Present participles have a `GI-` connector; forms of the verb *be* have a

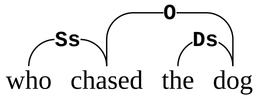
GI+. The AI connector is used for predicative adjectives. Adjectives have a AI- connector; forms of *be* have a AI+ connector.

3.4 Questions

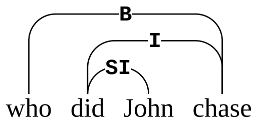
The SI connector is used for questions where there is subject-verb inversion. On nouns SI- is an alternative to S+, and on invertible verbs (is, has, did, must, etc.) SI+ is an alternative to S-. This allows



Wh- questions work in various different ways; only questions involving *who* will be discussed here. For subject-type questions, where *who* is substituting for the subject, *who* simply has an S+ connector. This allows



For object-type questions, where *who* is substituting for the object, the B connector is used. Transitive verbs have B- connectors as an alternative to their O+ connectors. *Who* has a B+ connector. This allows



The following incorrect sentences are rejected:

*Did John chase
*Who did John chase Mary
*John did Mary chase
*Chased John Mary

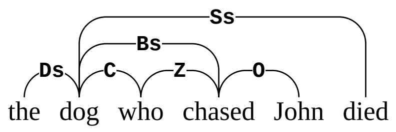
The following incorrect construction is accepted. In our on-line system, post-processing is used to eliminate this.

*Who John chased

3.5 Relative Clauses

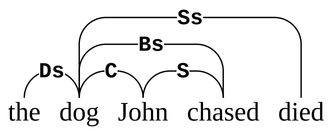
For subject-type relative clauses, where the antecedent is acting as the subject of the clause, a B connector serves to connect the noun to the verb of the relative clause. Nouns have a B+ connector. Notice that this is optional; it is also &ed with the S+, SI-, O+, and J+ connectors, meaning that one of these connectors must be used whether or not the noun takes a relative clause. Verbs have a B- connector which is orred with their S- connectors; if a verb is in a subject-type relative clause, it may not make an S connection as well.

For subject-type relative clauses, the relative pronoun *who* is mandatory. For this purpose, verbs have a Z- connector anded with their B- connector. *Who* has a Z+ connector; therefore it can fulfill this need. However, it also has a C- connector anded with its Z+ connector; this must connect back to the C+ connector on nouns. This allows the following:

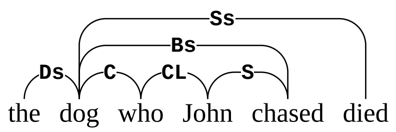


For object-type relative clauses, the same B+ connector on nouns is used. However, this time it connects to the *other* B- connector on verbs, the one which is orred with the O+ connector and which is also used for object-type *wh*- questions.

In this case, the relative pronoun *who* is optional. Notice that nouns have optional C+ and CL- connectors which are anded with their S+ connectors. These are used when the noun is the subject of an object-type relative clause. When *who* is not present, the C+ connector on the antecedent noun connects directly to the C- on the subject of the relative clause:



When *who* is present, the C+ on the antecedent connects to the C- on *who*; this forces the CL+ to connect to the CL- on the subject of the clause:



This system successfully rejects the following incorrect sentences:

> *The dog chased cats died
> *The dog who chase cats died
> *The dog who John chased cats died
> *The dog John chased cats died
> *The dog who chased died

The following incorrect constructions are accepted, but can be weeded out in post-processing:

> *The dog did John chase died
> *The dog who John died Mary chased died

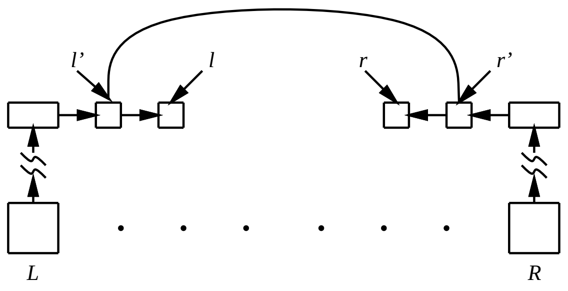
4 The algorithm

Our algorithm for parsing link grammars is based on dynamic programming. Perhaps its closest relative in the standard literature is the dynamic programming algorithm for finding an optimal triangulation of a convex polygon [2, p. 320]. It tries to build up a linkage (which we'll call a *solution* in this section) in a top down fashion: It will never add a link (to a partial solution) that is above a link already in the partial solution.

The algorithm is most easily explained by specifying a data structure for representing disjuncts. A disjunct d has pointers to two linked lists of connectors. These pointers are denoted $\mathit{left}[d]$ and $\mathit{right}[d]$. If c is a connector, then $\mathit{next}[c]$ will denote the next connector after c in its list. The next field of the last pointer of a list has the value NIL.

For example, suppose the disjunct $d =$`((D,O) ())` (using the notation of section 2). Then $\mathit{left}[d]$ would point to the connector `O`, next[left[d]] would point to the connector `D`, and next[next[$\mathit{left}[d]$]] would be NIL. Similarly, $\mathit{right}[d] =$ NIL.

To give some intuition of how the algorithm works, consider the situation after a link has been proposed between a connector l' on word L and a connector r' on word R. (The words of the sequence to be parsed are numbered from 0 to $N-1$.) For convenience, we define l and r to be $\mathit{next}[l']$ and $\mathit{next}[r']$ respectively. The situation is shown in the following diagram:



Here the square boxes above the words L and R represent a data structure node corresponding to the word. The rectangular box above each of these represents one of the (possibly many) disjuncts for the word. The small squares pointed to by the disjuncts represent connectors.

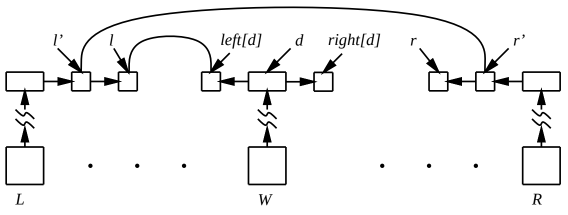
How do we go about extending the partial solution into the region strictly between L and R? (This region will be denoted $(L, \ldots, R)$.) First of all, if there are no words in this region (*i.e.* $L = R + 1$) then the partial solution we've built is certainly invalid if either $l \neq$ NIL or $r \neq$ NIL. If $l = r =$ NIL then this region is ok, and we may proceed to construct the rest of the solution.

Now suppose that the region between L and R contains at least one word. In order to attach the words of this region to the rest of the sentence there must be at least one link either from L to some word in this region, or from R to some word in this region (since no word in this region can link to a word outside of the $[L, \ldots, R]$ range, and something must connect these words to the rest of the sentence).

Since the connector l' has already been used in the solution being constructed, this solution must use the rest of the connectors of the disjunct in which l' resides. The same holds for r'. The only connectors of these disjuncts that can be involved in the $(L, \ldots, R)$ region are those in the lists beginning with l and r. (The use of any other connector on these disjuncts in this region would violate the ordering requirement.) In fact, all of the connectors of these lists must be used in this region in order to have a satisfactory solution.

Suppose, for the moment, that l is not NIL. We know that this connector must link to some disjunct on some word in the region $(L, \ldots, R)$. (It can't link to R because of the exclusion rule.) The algorithm tries all possible such words and disjuncts. Suppose it finds a word W and a disjunct d on W such that the connector l matches *left*$[d]$. We can now add this link to our partial solution.

The situation is shown in the following diagram.



How do we determine if this partial solution can be extended to a full solution? We do this by solving two problems similar to the problem we started with. In particular, we ask if the solution can be extended to the word range $(L, \ldots, W)$ using the connector lists beginning with *next*$[l]$ and *next*[left[d]]. We also ask if the solution can be extended to the word range $(W, \ldots, R)$ using the connector lists beginning with *right*$[d]$ and r. Notice that in the latter case, the problem we are solving seems superficially different: the boundary words have not already been connected together by a link. This difference is actually of no consequence because the pair of links (L to R and L to W) play the role that a direct link from W to R would play: (1) they separate the region $(W, \ldots, R)$ from all the other words, and (2) they serve to connect the words W and R together.

We need to consider one other possibility. That is that there might be a solution with a link between words L and W *and* a link between words W and R. (This results in a solution where the word/link graph is cyclic.) The algorithm handles this possibility by also attempting to form a link between *right*$[d]$ and r. If these two match, it does a third recursive call, solving a third problem analogous to our original problem. In this problem the word range is $(W, \ldots, R)$ and the connector lists to be satisfied begin with *next*[right[d]] and *left*$[r]$.

A very similar analysis suffices to handle the case when l is NIL.

The algorithm described has an exponential worst-case running time as a function of N, the number of words in the sequence to be parsed. This can easily be transformed into an efficient dynamic programming algorithm by using *memoization* ([2, p. 312]).

The running time is now bounded by the number of different possible recursive calls multiplied by the time used by each call. A recursive call is completely determined by specifying the pointers l and r. (These uniquely determine L and R.) The cost of a given call is bounded by the total number of disjuncts in the sequence of words.

If we let d be the number of disjuncts and c be the number of connectors, then the running time is $O(c^2 d)$. For a fixed link grammar, $d = O(N)$ and $c = O(N)$, so the running time is $O(N^3)$.

Our technical reports describe this algorithm in more detail. They contain pseudo-code for the algorithm [14], an argument for it's correctness [14] and an elegant recurrence for the number of linkages of a sentence [11].

After the algorithm above was implemented, we were interested in seeing how well it would work on sentences taken from newspapers and other natural sources. It quickly became clear that something else was needed to make the algorithm run faster on long sentences.

5 Speeding it up

As pointed out in the introduction, in a link grammar dictionary with significant coverage of English grammar the number of disjuncts on many words gets rather large. Thus, the constant d in the analysis at the end of the last section is quite large. We devised and implemented several time-saving schemes that run in conjunction with the algorithm of the previous section.

5.1 Pruning

Our first approach is based on the following observation: In any particular sequence of words to be parsed, most of the disjuncts are irrelevant for the simple reason that they contain a connector that does not match any other connector on a word in the sequence. To be more precise, suppose that a word W has a disjunct d with a connector `C` in its right list. If no word to the right of W has a connector (pointing to the left) that matches `C`, then the disjunct d cannot be in any linkage. This disjunct can therefore be deleted without changing the set of linkages. Deleting such a disjunct is called a *pruning step*. *pruning* consists of repeating the pruning step until it can no longer be applied.

The set of disjuncts left (after pruning is complete) is independent of the order in which the steps are applied. (The pruning operation has the Church-Rosser property.) We therefore choose an ordering that can be efficiently implemented. It would be ideal if we could achieve a running time for pruning that is linear in the number of connectors. The scheme we propose satisfies no useful a-priori bound on its running time, but in practice it appears to run in linear time.

A series of sequential passes through the words is made, alternately left-to-right and right-to-left. The two types of passes are analogous, so it suffices to describe the left-to-right pass. The pass processes the words sequentially, starting with word 1. Consider the situation after words $1, \ldots, W-1$ have been processed. A set S of connectors has been computed. This is the set of connectors that exists on the right lists of the disjuncts of words $1, \ldots, W-1$ that have not been deleted. To process word W, we consider each disjunct d of W in turn. For each connector c on the left list of d, we search the set S to see if it contains a connector that matches c. If one of the connectors of d matches nothing in S, then we apply the pruning step to d (we remove d). Each right connector of each remaining disjunct of W is now incorporated into the set S. This completes the processing of word W.

The function computed by this left-to-right pass is idempotent, which is another way of saying that doing the operation twice in a row will be the same as doing it once. Therefore if (as we alternate left-to-right and right-to-left passes) a pass (after the first one) does nothing, then all further passes will do nothing. This is how the algorithm decides when to stop.

The data structure used for the set S is simply a hash table, where the hash function only uses the initial upper-case letters of the connector name. This ensures that if two connectors get hashed to different locations, then they definitely don't match.

Although we know of no non-trivial bound on the number of passes, we have never seen a case requiring more than five. Table ?? shows a typical example of the reduction in the number of disjuncts achieved by pruning.

5.2 The fast-match data structure

The inner loop in the algorithm described in section 4 searches for a word W and a disjunct d of this word whose first left connector matches l, or whose first right connector matches r. If there were a fast way to find all such disjuncts, significant savings might be achieved. The fast-match data structure, which is based on hashing, does precisely this. The speed-up afforded by this technique is roughly the number of different connector types, which is roughly 30 in our current dictionary.

5.3 Power pruning

Power pruning is a refinement of pruning that takes advantage of the ordering requirement of the connectors of a disjunct, the exclusion rule, and other properties of any valid linkage. It also interacts with the fast-match data structure in a beautiful way. Unfortunately, these details are beyond the scope of this paper [4]. Table ?? shows outcome of pruning and power pruning on a typical sentence.

Each of the refinements described in this section significantly reduced the time required to do search for a linkage. The operations of pruning, power pruning, and searching for a linkage all take roughly the same amount of time.

6 Dependency and categorial grammars

6.1 Dependency formalisms

There is a large body of work based on the idea that linguistic analysis can be done by drawing links between words. These are variously called *dependency systems* [5], *dependency syntax* [10], *dependency grammar* [3, 4], or *word grammar* [6, 7].

In dependency grammar, a grammatical sentence is endowed with a *dependency structure*, which is very similar to a linkage. This structure, as defined by Meľčuk [10], consists of a set

[4] Although they do appear in our technical report [14].

of planar directed arcs among the words that form a tree. Each word (except the *root word*) has an arc out to exactly one other word, and no arc may pass over the root word. In a linkage (as opposed to a dependency structure) the links are labeled, undirected, and may form cycles, and there is no notion of a root word.

Gaifman [5] was the first to actually give a formal method of expressing a dependency grammar. He shows that his model is context-free.

Meĺčuk's definition of a dependency structure, and Gaifman's proof that dependency grammar is context free imply that there is a very close relationship between these systems and link grammars. This is the case.

It is easy to take a dependency grammar in Gaifman's notation and generate a link grammar that accepts the same language. In this correspondence, the linkage that results from parsing a sentence is the same as the corresponding dependency structure. This means that our algorithm for link parsing can easily be applied to dependency grammars. The number of disjuncts in the resulting link grammar is at most quadratic in the number of rules in the dependency grammar. None of the algorithms that have been described for dependency parsing [3, 15, 7] seem to bear any resemblance to ours. It is therefore plausible to conjecture that our algorithms and techniques could be very useful for directly parsing dependency grammars.

Gaifman's result shows that it is possible to represent a link grammar as a dependency grammar (they're both context-free). But this correspondence is of little use if the parsed structures that result are totally different.

One problem with constructing a dependency grammar that is in direct correspondence with a given link grammar is that a linkage in a link grammar my have cycles, whereas cycles are not allowed in dependency grammar. If we restrict ourselves to acyclic linkages, we run into another problem. This is that there is an exponential blow-up in the number of rules required to express the same grammar. This is because each disjunct of each word in the link grammar requires a separate rule in the dependency grammar.

Gaifman's model is not lexical. The method classifies the words into categories. One word can belong to many categories. Roughly speaking, for each disjunct that occurs in the dictionary, there is a category of all words that have that disjunct. The notation is therefore in a sense orthogonal to the link grammar notation.

We are not aware of any notation for dependency systems that is lexical, or that is as terse and well suited for a natural language grammar as link grammars. There has been work on creating dependency grammars for English [7, 3], but we are not aware of an implementation of a dependency grammar for any natural language that is nearly as sophisticated as ours.

6.2 Categorial grammars

Another grammatical system, known as a *categorial grammar* [1] bears some resemblance to link grammars. Below we show how to express any categorial grammar concisely as a link grammar. It appears to be more difficult to express a link grammar as a categorial grammar.

Just as in a link grammar, each word of a categorial grammar is associated with one or more symbolic expressions. An expression is either an atomic symbol or a pair of expressions combined with one of two types of binary operators: / and \.

A sentence is in the language defined by the categorial grammar if, after choosing one expression associated with each word, there is a *derivation* which transforms the chosen sequence of expressions into `S`, a single expression consisting of a special atomic symbol. The derivation proceeds by combining two neighboring expressions into one using one of the following rules:

$$\frac{e \qquad e\backslash f}{f} \qquad\qquad \frac{f/e \qquad e}{f}$$

Here e and f are arbitrary expressions, and $f\backslash e$ and f/e are other expressions built using e and f. In both cases the two expressions being combined (the ones shown above the line) must be adjacent in the current sequence of expressions. Each combinational operation produces one expression (the one below the line), and reduces the number of expressions by one. After $n-1$ operations have been applied, a sentence of length n has be reduced to one expression.

For example, consider the following categorial grammar [9]:

```
Harry:      NP, S/(S\NP)
likes:      (S\NP)/NP
```

```
peanuts:      NP
passionately: (S\NP)\(S\NP)
```

Here is the derivation of *Harry likes peanuts passionately*.

```
 Harry     likes    peanuts   passionately
S/(S\NP) (S\NP)/NP    NP      (S\NP)\(S\NP)
         ---------------
               S/NP
               ----------------------------
                            S\NP
--------------------------------
               S
```

The set of languages that can be represented by categorial grammars (as they are described here) is the set of context-free languages [1][5] This fact alone sheds no light on the way in which the formalism represents a language. To get a better understanding of the connection between categorial grammars and link grammars, the following paragraphs explain a way to construct a link grammar for a given categorial grammar. The reverse (constructing a categorial grammar from a given link grammar) seems to be more difficult, and we do not know of an elegant way to do this.

To simplify the construction, we'll use a modified definition of a link grammar called a *special link grammar*. This differs from an ordinary link grammar in two ways: the links are not allowed to form cycles, and there is a special word at the beginning of each sentence called *the wall*. The wall will not be viewed as being part of any sentence.

Let d be a categorial grammar expression. We'll show how to build an equivalent link grammar expression $E(d)$. If a word w has the set $\{d_1, d_2, \ldots, d_k\}$ of categorial expressions, then we'll give that word the following link grammar expression:

$$E(d_1)\texttt{or}E(d_2)\texttt{or}\cdots\texttt{or}E(d_k)$$

The function $E(\cdot)$ is defined recursively as follows:

$$\begin{aligned}
E(f/e) &= f/e- \ \texttt{or}\ f/e+ \ \texttt{or}\ (e+ \ \&\ E(f)) \\
E(e\backslash f) &= e\backslash f- \ \texttt{or}\ e\backslash f+ \ \texttt{or}\ (e- \ \&\ E(f)) \\
E(A) &= A- \ \texttt{or}\ A+
\end{aligned}$$

Here A stands for any atomic symbol from the categorial grammar, and A, f/e and $e\backslash f$ are connector names in the link grammar formulas.

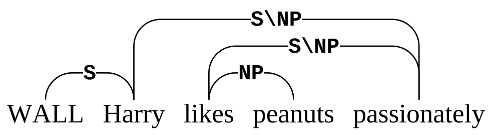
The wall has the formula `S+`.

Here is the link grammar corresponding to the categorial grammar above:

```
WALL:         S+;
peanuts:      NP+ or NP-;
Harry:
  (NP- or NP+)
  or (S/<S\NP>-
     or S/<S\NP>+
     or (S\NP+ & (S+ or S-)));
likes:
  <S\NP>/NP- or <S\NP>/NP+
  or (NP+ & (S\NP- or S\NP+
            or (S- & (NP- or NP+))));
passionately:
  <S\NP>/<S\NP>- or <S\NP>/<S\NP>+
  or (S\NP- & (S\NP- or S\NP+
              or (S- & (NP- or NP+))));
```

(Here we have replaced parentheses in the categorial grammar expressions with brackets when using them inside of a link grammar expression.)

This link grammar gives the following analysis of the sentence shown above:



Notice that in this construction, both the size of the link grammar formula, and the number of disjuncts it represents are linear in the size of the original categorial grammar expressions. This suggests that a very efficient way to parse a categorial grammar would be to transform it to a link grammar, then apply the algorithms and heuristics described in this paper.

7 Remarks

Link grammars have become the basis for several other research projects. John Lafferty [11] proposes to build and automatically tune a probabilistic language model based on link grammars.

[5] There are other variants of categorial grammars which are mildly context-sensitive[9]. Of course the construction presented here does not work for those languages.

The proposed model gracefully encompasses trigrams and grammatical constraints in one framework. Andrew Hunt[6] has developed a new model of the relationship of prosody and syntax based on link grammars. He has implemented the model, and in preliminary tests, the results are much better than with other models. Tom Brehony[6] has modified our parser to detect the kinds of errors that Francophones make when they write in English.

References

[1] Y. Bar-Hillel, Language and information; selected essays on their theory and application. Addison-Wesley, 1964.

[2] Cormen, T. H., C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press and McGraw-Hill, 1990.

[3] Fraser, N., "Parsing and dependency grammar," In: Carston, Robyn (ed.) *UCL Working Papers in Linguistics* 1, University College London, London, 1989 Pages 296–319.

[4] Fraser, N., "Prolegomena to a formal theory of dependency grammar," In: *UCL Working Papers in Linguistics* 2, University College London, London, 1990 Pages 298–319.

[5] Gaifman, H., "Dependency systems and phrase-structure systems," *Information and Control* **8**, 1965, Pages 304–337.

[6] Hudson, R., *Word Grammar*, Basil Blackwell, 1984.

[7] Hudson, R., "Towards a computer testable word grammar of English," In: Carston, Robyn (ed.) *UCL Working Papers in Linguistics* 1, University College London, London, 1989, Pages 321–339.

[8] Hudson, R., *English Word Grammar*, Basil Blackwell, 1990.

[9] Joshi, A. K., "Natural Language Processing," *Science*, Volume 253, No. 5025, (Sept. 13, 1991), Pages 1242–1249.

[10] Mel̀čuk, I. A. *Dependency Syntax: Theory and Practice*, State University of New York Press 1988.

[11] Lafferty, J, D. Sleator, D. Temperley, "Grammatical Trigrams: A Probabilistic Model of Link Grammar," Proc. of the 1992 AAAI Fall Symp. on Probabilistic Approaches to Natural Language, and Technical report CMU-CS-92-181, School of Computer Science, Carnegie Mellon University, Sept 1992.

[12] Oehrle, R. T., E. Bach, and D. Wheeler, Editors *Categorial Grammars and Natural Language Structures* D. Reidel Publishing Company, 1988.

[13] Y. Schabes. "Stochastic lexicalized tree-adjoining grammars." In *Proceedings of COLING-92*, Nantes, France, July 1992.

[14] Sleator, D. D., D. Temperley, "Parsing English with a Link Grammar," Technical report CMU-CS-91-196, Carnegie Mellon University, School of Computer Science, October 1991.

[15] van Zuijlen, J., M., "Probabilistic methods in dependency grammar parsing," Proceedings of the International Workshop on Parsing Technologies, Carnegie Mellon University, 1989, Pages 142–250.

[6] Personal communication.