PARSING AND DEPENDENCY GRAMMAR

Norman M. FRASER

Abstract

Current characterisations of dependency grammar are not equivalent to phrase structure grammar in a number of important respects. As a consequence, parsing algorithms for phrase structure grammars are of questionable relevance to dependency grammars. This paper presents a new parsing algorithm - the "Bonding Algorithm" - which is used to parse input in terms of the dependency-based theory of Word Grammar. A parse stack controls the bonding of syntactic "molecules", where a molecule consists of a head word plus a record of all its dependents, along with details of the "valency" or dependency requirements of the molecule. The special-purpose "visitor" relation and the technique for deriving new dependencies from existing ones are amongst the novel devices embodied in the parser. Some superficial similarities with categorial grammar parsers are noted but these must be treated with caution since the differences appear to outweigh the similarities.

1. Dependency and constituency[1]

Following the findings of Gaifman (1965), it is widely accepted that for every dependency grammar there is a weakly equivalent context-free phrase structure grammar. According to Robinson (1970), every dependency grammar has a strongly equivalent context-free phrase structure grammar and possibly *vice versa*. Gaifman and Robinson both used a characterisation of dependency in which it is true of every well-formed string that:

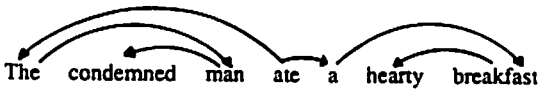
i) there is a single sentence root (one independent element);
ii) all other elements depend on some element;
iii) no element depends directly on more than one other;
iv) each dependent is adjacent to its head.

A dependent is said to be adjacent to its head if it is not separated from that head by anything apart from other dependents of the same head or of the dependent itself. This restriction has much the same effect as the constraint which prohibits the crossing of branches in phrase markers.

---

[1]*This work has been carried out with the support of the Procurement Executive, Ministry of Defence. An earlier version of the paper was presented at the Autumn 1988 (Exeter) meeting of the Linguistics Association of Great Britain. My thanks to Dick Hudson for his many valuable comments on earlier drafts.*

Sentence (1) illustrates these features (arrows point from heads to dependents).

(1) 

In example (1) I have adopted two controversial positions which deserve to be noted although they are not crucial to the argument. Firstly, dependency relations hold between single words and not between larger constituents. This approach has been advanced by Hudson (1984) and incorporated into his dependency-based theory of Word Grammar (WG).

Secondly, I have adopted an analysis in which nouns depend on their determiners rather than the more usual reverse case.

While important issues are involved in both of these points I do not wish to argue them here. Had I chosen another sensible set of assumptions it seems likely that the general sentence processing strategy I am going to present would be equally relevant.

It is possible to map dependency structures onto phrase structures by bracketing each word together with its dependents to form phrases. Using this method we could map the dependency structure in (1) onto the phrase structure bracketing shown in (2).

(2) [[The [condemned man]] ate [a [hearty breakfast]]]

This is a reasonable - though unusual - phrase structure analysis of (1). What is clearly missing from it is any information relating to the direction of dependencies. For example, does *hearty* depend on *breakfast* or *breakfast* depend on *hearty*?

The advent of head-marking phrase structure grammars (Jackendoff, 1977) represents an important step forward in this respect since it is possible to envisage a function which uniquely maps a head-marked phrase structure sentence analysis onto a dependency analysis.

The relatedness of phrase structure and dependency structure systems has obvious implications for parsing. David Hays has observed that "a phrase structure parser can be converted into a dependency parser with only a minor alteration" (Hays, 1966: 79). Since a great deal is known about the formal properties of context free phrase structure grammars and the parsing algorithms available for those grammars (for example see Aho and Ullman, 1972), it is hardly surprising that people wishing to produce dependency analyses of sentences should try to utilise existing phrase structure tools as much as possible. If a dependency structure can be produced by applying a mapping function to the output of a phrase structure parser then it would seem perverse not to adopt this strategy. Alternatively, if it is required that the grammar as well as the analysis be expressed in terms of dependency, this can be achieved by using an existing phrase structure parsing algorithm to construct not constituents but head-dependent structures. In both cases the

grammars can be expressed in terms of collections of rewrite rules (Hays, 1964; Levelt, 1974:135).

These techniques may or may not have been feasible or useful in the past but what can not be questioned is that they presuppose a particular characterisation of dependency structure which is seldom advocated at the present time, namely that given in (i)-(iv) above. Just as the formal definition of phrase structure grammars has been augmented to deal with natural language phenomena (for example, in order to analyse discontinuous constituents it may be necessary to allow crossing branches in trees), so the definition of dependency grammars for natural languages has deviated significantly from the formal standard. Very few people would now advocate the kind of simple dependency system Gaifman and Robinson assumed. A number of refinements introduce clear divergences from all current phrase structure characterisations. We shall briefly consider three of these refinements.

i) Multiple heads

In phrase structure grammars which incorporate the notion "head", a constituent is defined as the phrasal projection of X where X is the head of the phrase containing X and its arguments (if any). One consequence of this definition is that a phrase must contain exactly one head. Another consequence is that an argument can stand in relationship to exactly one predicate, namely the head of the phrase in which it occurs.

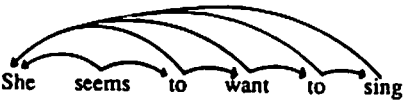
In his dependency-based theory of Lexicase, Starosta analyses "prepositional phrases" as exocentric constructions, having not one but two heads (or "coheads") (Starosta, 1988:232ff). This violates the X-bar one-head constraint.

In sentence (3a) there are good reasons for recognising *John* as the subject of both *keeps* and *snoring*. It is John who is doing the keeping and the snoring. (3b) would have the same structure as (3a). Here it should be clear that *raining* rather than *keeps* selects *it*. *John* is a suitable subject in (3a) but not in (3c). Replacement of the proper noun *John* by the pronoun *he* (3d) does not change the result.

(3) a. 

b. It keeps raining
c. * John keeps raining
d. * He keeps raining

This kind of double dependency analysis is adopted in Hudson's WG (1984:82ff). It is difficult to see how such an analysis could be translated into a labelled bracketing and even if it could the subject would have to act as argument to two predicates at the same time. Much more complicated structures can also be generated. (4) shows the WG analysis of a sentence in which one word is the subject of five other words at the same time.

(4) 

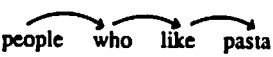
It should be clear that this kind of construction has no parallel in phrase structure grammar.

ii) Interdependence

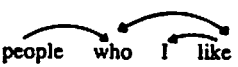
It can sometimes happen that two separate dependency relations, licensed by two separate rules in the grammar, can involve only one pair of words. For example, a dependency grammar might include a rule which states that the verb *like* requires an object as in (5).

(5) 

There might be another rule which states that the dependent of a relative pronoun is a finite verb (ie, the main verb of the relative clause), as in (6).

(6) 

The interaction of these two rules can occasionally lead to the verb being the complement of the relative pronoun while the relative pronoun is the object of the verb. Sentence (7) illustrates this interdependence.

(7) 

The different grammatical relations signify different things and their occasional coincidence in constructions like this need not be regarded as undesirable.

This kind of interdependence can not be represented in a phrase structure grammar; in particular, it can not be represented in a head-marking phrase structure grammar since both related words are simultaneously heads and dependents.

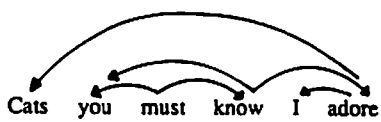
iii) Revised adjacency

The version of adjacency given above prohibits multiple-head constructions like the one shown in (8).

(8) 

However, all of the dependency relations here are well motivated within WG.

Sentence (9) represents another kind of useful construction which is prohibited by the adjacency principle since the extracted item *cats* is not adjacent to its head *adore*.

(9) 

Simply abandoning the notion of adjacency to allow sentences like these would lead to undesirable constructions (such as (10)) being allowed.

(10) 

The WG solution is to reformulate the adjacency principle to allow sentences like (8) and (9) but not (10).

THE ADJACENCY PRINCIPLE

> A word must not be separated from its head by anything other than a subordinate of one of its heads. (Hudson, forthcoming).

SUBORDINATES

> A is a subordinate of B if A is B, or A is a dependent of a subordinate of B (Hudson, 1988:192).

Although phrase structure systems can be augmented to cope with discontinuous constituents, it is clear that some significant differences distinguish their analyses of sentences (8) and (9) from the dependency analysis given here.

Current versions of phrase structure grammar and dependency grammar can not be said to be equivalent. Contrary to Hays' observation, a phrase structure parser can not be "converted into a dependency parser with only a minor alteration". Neither is it appropriate to assume that a dependency parser can be constructed by taking an available phrase structure parser and adding a post-processor to map phrase structures onto dependency structures. If dependency structures are going to be used in linguistic analysis - and this is a separate question - then it is vital that more effort be focused on dependency parsing as an autonomous discipline and not simply as a minor variant of phrase structure parsing.

2. Dependency parsing

In comparison with phrase structure parsing, very little work has been done specifically on dependency parsing. Much of that is being carried out in West Germany and Eastern Europe where dependency remains the dominant tradition in syntax. In Heidelberg, Hellwig (1980) has been working on the PLAIN system which has a shared representation for dependency and constituency information. Kunze (1982) has been leading a research team in East Berlin for a number of years and in Prague, Sgall and his colleagues (1986) have developed Functional Generative Description which is based on dependency but also employs many other non-standard features. Some of the most impressive work has been done at the SITRA Foundation in Helsinki (Valkonen et al., 1987) where the special strengths of dependency in capturing generalisations in variable word order languages are being exploited in parsing systems. Covington in Georgia, USA, has also focused on problems of variable word order in his prototype system. At NEC in Japan, Fukumochi and his colleagues are building dependency-based computer systems for parsing Japanese, as is Starosta in Honolulu (Starosta, 1988; Starosta & Nomura, 1986). Finally, in London, Fraser (1985) and Hudson (1986) have implemented a number of WG parsers.

The basic idea in dependency parsing is not to combine constituents to form larger constituents but rather to satisfy the valency requirements of individual words. A word can have a number of slots which require fillers of particular kinds. A word can itself be the filler for slots belonging to other words. Most of the dependency parsers reported in the literature adopt a blackboard methodology. In artificial intelligence (AI) a blackboard is a shared data structure which is used as the only means of communication between different processes. Roughly speaking, parsing proceeds as follows:

a) Find the current word's valency (usually recorded in the lexicon).
b) See if the valency requirements of the current word are satisfied by any word on the blackboard. If so, check that the two words are adjacent. Adjust the blackboard and the current word's record accordingly.
c) Broadcast on the blackboard the most recent information about the current word.
d) Get the next word and repeat the same procedure for it.

Blackboard-based processing is used successfully in a wide range of AI applications. However, its very generality can be seen as a hindrance to

computational efficiency and conceptual clarity. The rest of this paper will introduce a dependency parsing algorithm which is both efficient and simple.

3. The bonding algorithm

The work reported here has been carried out within the framework of WG. More detailed introductions to the theory can be found in Hudson, 1984; Hudson, 1988; Hudson & van Langendonck, forthcoming.

As we have seen, one of the central claims of WG is that a grammar need only refer to word-sized items. However, there is no theoretical reason why a WG parser should share the same restrictions as the grammar. Indeed, it is only to be expected that the procedural part should use different structures from the declarative part of any system. I propose that, while a grammar may only refer to word-sized items, a parser should be allowed to refer in addition to two other kinds of data structure: **molecules** and **stacks**.

To help understand the approach to parsing introduced here, we will draw an analogy with chemistry and the process of molecular bonding. An atom with a positive or negative charge is called an **ion** and its overall charge is called its **valency**. Similarly, a single word is said to have a valency. Where ions have a positive charge, words have a requirement for dependents; where ions have a negative charge, words have a requirement for a head. When a positively charged ion meets a negatively charged ion (and other factors permit) the two ions bond to form a single molecule. Any imbalance in charge between the two ions remains as a property of the molecule (although ultimately it is the property of a single nucleus). In like fashion, a word which requires (or allows) a dependent can bond with a word which requires a head to form a molecule unless any constraints prevent it. Any slots (charges) not involved in this bond remain as properties of the molecule. Molecules can bond with other molecules. Grammaticality is analogous to molecular stability in chemistry - in our model a molecule with saturated valency can serve as a sentence. This is important since the largest structure a WG grammar refers to is the word.

For obvious reasons, the scheme presented here is called the **bonding algorithm**.

3.1 Molecules

A molecule is a structure consisting of a root word plus all its subordinates. We shall characterise molecules as four-element lists of the form shown in (11).

(11) [Negative-list, Positive-list, Subordinates, Derivable]

Negative-list = a list of unfilled head slots. A typical Negative-list entry would look like (12):

(12) [2, common-noun, head, word, either]

Here, 2 signifies that the word which licensed the slot is the second word in the sentence. The next item tells us that it was a common-noun. After that comes the slot label: head. The next item says that the filler can be any class of

word and the final item indicates that word-2 can either precede or follow its head.

Positive-list = a list of unfilled dependent slots. A typical Positive-list entry is shown in (13).

(13) [3, finite-verb, subject, noun, before]

Here, 3 signifies that the word which licensed the slot is the third word in the sentence. The next item tells us that it was a finite verb. After that we have the information that the dependency relation involved is the subject relation. The next item constrains the possible subject fillers to nouns and the final item shows that the subject must come before word-3 in linear order. Notice that here we specify the position of the slot-filler relative to the word which licenses it, whereas in the case of Negative-lists the position specified is that of the licensing word in relation to the slot-filler.

There is, of course, a certain amount of arbitrariness in the association of positive charge with dependency requirements and negative charge with head requirements rather than vice versa. The significant point is that they are mutually attractive opposites.

Subordinates = a list structure containing records of all subordinate words and their dependency relations. We shall see how this works in section 3.4.

Derivable = a list of slots and information detailing how to derive the fillers from existing relations. Again, it is not necessary to know its internal structure at this point. We shall learn more of its structure and function in section 3.8.

3.2 Stacks

A stack is a list which can only be accessed from one end so that additions and subtractions must take place from that point. This produces storage and retrieval properties of the "last in, first out" variety. Stacks are normally visualised as vertical structures so items are said to be stored and retrieved at the top of the stack; the opposite end is the bottom. When an item is stored, it is said to be pushed onto the stack; when it is retrieved, it is popped off the stack.

In our parsing algorithm, a single parse stack is used, and only molecules may be pushed onto it.

3.3 Preliminaries

The parser works in a left-to-right, one-pass manner. It would be inaccurate to describe it as either top-down or bottom-up since these are expressions which are rooted in phrase structure grammar. In spirit, however, it is a bottom-up parser since it begins with the words of the sentence and

attempts to build a structure. Of course, the structure is built between the words and not above them.

The parser reads one word at a time, constructing for each word a frame of slots and constraints on fillers. The information which is used to build a frame is obtained from the grammar. Since all rules in a WG refer to word-sized units, there is no obvious distinction between grammar and lexicon. Grammatical information is expressed in terms of simple propositions which describe the structure of a **generalisation hierarchy**. The lexical entry for a given word is produced on demand by a process of **property inheritance** (Fraser, 1985). This kind of hierarchically structured lexicon is currently being advocated by a number of linguists working on other theories (eg. HPSG: Flickinger, 1987, Pollard & Sag, 1987; Cognitive Grammar: Langacker, forthcoming; Gazdar, 1988). However, for the purposes of the present argument it is not necessary to be familiar with these notions. It is sufficient to know that some mechanism exists for using a lexicon/grammar to construct a word frame appropriate for each word.

Some slots in the frame may be filled immediately by observation (eg. those pertaining to morphology, phonology, and the like) but others will require a certain amount of processing before they can be filled. These are the standard valency slots and the slots with derivable fillers.

Let us consider the familiar example given in (14). For the sake of simplicity we shall only mention relevant properties.

(14) 

This would yield the frame for word-1 given in (15) (expressed in propositions of the WG metalanguage).

(15) word-1 isa proper-noun

word-1 has (a head)
word-1 has (mano pre-adjunct)
word-1 has (ano post-adjunct)

(pre-adjunct of word-1) is (a adjective)
(post-adjunct of word-1) is (a preposition)
(pre-adjunct of word-1) precedes word-1
(post-adjunct of word-1) follows word-1

Propositions which license slots contain a quantifier which indicates the status of the slot. We shall only consider the meanings of the three most common quantifiers. These are given in (16):

(16) **a** this indicates that the slot must be filled obligatorily.

ano this indicates that the slot may or may not be filled (ie. it is an optional slot).

mano this indicates that any number (from zero upwards) of specified slots is licensed. So, in English, a noun may be preceded by any number of adjectives including none.

Word frame information can be expressed much more compactly if it is converted into the molecule format. The molecule which would be constructed for word-1 is given in (17).

(17) [[[1,proper-noun,[a,head],word,either]],
[[1,proper-noun,[mano,pre-adjunct],adjective,before],
[1,proper-noun,[ano,post-adjunct],preposition,after]]
[],
[]]

3.4 Molecular bonding

At the heart of the parser lies a process for combining molecules to form larger molecules. In general, if some element of the Positive-list of a molecule can be combined with some element of the Negative-list of another molecule (or *vice versa*) then the second molecule can be merged into the first to produce a new, larger molecule. We illustrate this process by means of a worked example. (18) shows the initial molecule which would be constructed for word-2 of sentence (14).

(18) [[]
[[2, finite-verb, [a, subject], noun, after],
[2, finite-verb, [a, object], noun, after]],
[],
[]]

For the sake of simplicity I have ignored the requirement in English for subject-verb agreement. This can be accommodated in the framework but it would require some digression.

In order to talk about the process of molecular bonding, we will identify the elements of Positive-lists and Negative-lists by means of the names given in (19).

(19) [number, class, [quantifier, slot], slot-class, order]

We will attempt to bond (17) and (18) by trying to unify an element from one Negative-list with an element from the Positive-list of the other molecule. We will say that the two unify if the following conditions hold:

a) A isa B, where A is the Negative-list class and B is the Positive-list slot-class;

b) C isa D, where C is the Positive-list class and D is the Negative-list slot-class;

c) the Positive and Negative orders unify; before unifies with before, after unifies with after, either unifies with anything, but before and after will not unify with each other.

Let us consider the first element of the Positive-list of (18) and the first (and only) element of the Negative-list of (17). These are shown in (20).

(20) +ve [1, proper-noun, [a, head], word, either]
-ve [2, finite-verb, [a, subject], noun, before]

When we try to match these lists we find that:

a) "proper-noun isa noun" succeeds
b) "finite-verb isa word" succeeds
c) either unifies with before

therefore the conditions are satisfied and the molecules can bond. The structure of the resulting molecule is shown in (21).

(21) [[],
[[2, finite-verb, [a, object], noun, after]],
[[subject, 2, 1]],
[]]

Several interesting things have happened here. First of all, the matching elements - a Negative element of (17) and a Positive element of (18) - have collapsed into a single element which is recorded in the Subordinates list (read this element as "the subject of word-2 is word-1"). In addition, two Positive elements of (17) have been deleted. We shall presently discover the reason for this.

3.5 Unification

The basic operation in molecular bonding is **unification** (Shieber, 1986). Unification-based models of a wide range of current linguistic theories have been developed. The list includes Lexical-Functional Grammar, Generalised Phrase Structure Grammar, Head-driven Phrase Structure Grammar, Government-Binding theory, Categorial Grammar, Situation Semantics, Systemic Grammar and, of course, Dependency Grammar. The fact that a single operation can underlie models of different theories is sometimes claimed to be of value in comparing and evaluating theories. However, the very simplicity and generality of unification almost certainly robs the claim of most of its strength. I make no such claims for the model presented here. Unification is exploited in the model because it is a simple and effective low-level matching and structure-building operation.

3.6 Using the stack

The ability to unify molecules is of little value by itself. Structures like (10) must be avoided. This is where the stack comes into play. Only two molecules are available for combining at any one time, namely, the top two molecules on the parse stack. We shall refer to the top-most molecule as M1 and the next one down as M2. To begin with, checks are made to see if M2

can depend on M1 (ie. if some element in M1's Positive-list will unify with some element in M2's Negative-list). If these tests succeed then the two molecules are combined to form a new one. If not, then checks are made to see if M1 can depend on M2. Again, if they unify, then the two molecules bond to form a new molecule. Since the two molecules have combined to form one new one, this becomes the new M1 and the next highest stack element becomes available as M2. If two molecules will not bond, then the stack remains unchanged. The next word of the sentence is then read and a new molecule is constructed and added to the parse stack.

By the end of a sentence, there should be a single molecule left on the stack. If there is more than one then the parser has been unable to find a single dependency structure for the input string.

There are several reasons why a stack-based parser might be preferred to a blackboard-based parser. First of all, the stack-based model is much more efficient since it is only ever possible to consider bonding adjacent molecules, whereas in the blackboard model the adjacency constraint had to be applied after almost all the other processing had been done. This led to a significant amount of effort being wasted exploring options which would never lead anywhere. The reason for the saving is that the stacking model makes much of the adjacency principle implicit to its operation. In fact, the adjacency principle can be said to be a high-level description of the structures which are produced as artifacts of the language processor. This is rather different from the earlier view that the adjacency principle corresponded to some actual component of the language processor.

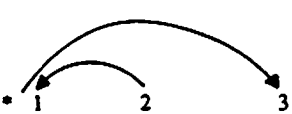
Another strength of this stack-based approach is that it provides neat ways of identifying and closing down impossible search paths as soon as possible. This was a particular problem for blackboard-based models. Recall that when we combined molecules (17) and (18), we produced a new molecule (21). However, in the process, we lost the two slots shown in (22):

(22) [1,proper-noun,[mano,pre-adjunct],adjective,before]
[1,proper-noun,[ano,post-adjunct],preposition, after]

It should be obvious that the first word of a sentence can not possibly have a pre-adjunct. However, we are able to appeal to a more general principle which states that any M1 which has optional slots for dependents with the before order feature, will have these options closed if it is found that the stack is empty; that is to say: there are not and never will be any available fillers. Likewise, if M1 has non-optional slots for preceding fillers and the stack is empty, then no single dependency structure will ever be able to link all of the words of the sentence in a coherent analysis.

The reason for the erasure of the post-adjunct slot is that there is a generalisation which says that when an M2 is combined with an M1 as its head, any optional after slots it may have had are removed. This is because structures of the sort shown in (23) can never occur.

(23)



If the slot had been obligatory then once again we would have known that a fully connected analysis of the sentence could never have been produced so we would have aborted at that point.

It is important to be able to spot "dead-end" analyses as quickly as possible since lexical ambiguity tends to be of such proportions that there are often several paths and many more partial paths through a sentence. If the partial paths can be abandoned as soon as possible, the amount of fruitless processing can be pared down to a minimum.

3.7 A worked example

In order to get a clearer idea of the working of the bonding algorithm let us consider the step-by-step analysis of sentence (24). This appears under (25). The grammar fragment used in this analysis appears as (26).

(24)

(25)

[1] The

STACK — READ WORD 1; No M2 to bond with

[2] big

The

STACK — READ WORD 2; M1 & M2 will not bond

[3] fat

big

The

STACK — READ WORD 3; M1 & M2 will not bond

[4] cat

fat

big

The

STACK — READ WORD 4; M1 & M2 will bond

[5]

big

The

STACK — AFTER BONDING
M1 & M2 will bond

[6]

The

STACK — AFTER BONDING
M1 & M2 will bond

[7]

STACK — AFTER BONDING
No M2 to bond with

[8] likes

STACK — READ WORD 5
M1 & M2 will bond

[9]

STACK — AFTER BONDING
No M2 to bond with

[10] large

STACK — READ WORD 6
M1 & M2 will not bond

[11] meals

large

STACK

READ WORD 7
M1 & M2 will bond

[12]

large meals

STACK

AFTER BONDING
M1 & M2 will bond

[13]

STACK

AFTER BONDING
No M2 to bond with
Input complete

(26) determiner has (a complement)
adjective has (a pre-adjunct)
noun has (mano pre-adjunct)
tensed-verb has (a subject)
w-like has (a object)

(complement of determiner) is (a common noun)
(pre-adjunct of adjective) is (a adverb)
(pre-adjunct of noun) is (a adjective)
(subject of word) is (a noun)
(object of word) is (a noun)

(pre-adjunct of word) precedes word
(post-adjunct of word) follows word

subject isa pre-adjunct
object isa post-adjunct

3.8 Visitors

The stack itself can be treated as a data object; rules can be written to recognise and manipulate particular stack configurations. We have already seen how it is possible to use the absence of an M2 to predict the

fruitlessness of analyses where M1 requires a pre-adjunct. Now we will consider how some apparently problematic stack configurations can be turned to good account.

Let us consider what happens during the parse of a sentence in which the object has been extracted. If we analyse sentence (27), we arrive at a point where the stack can be represented as in (28).

(27) The woman I love

(28) I ← love

The → woman

STACK

At this point, M1 has no more normal slots allowing preceding fillers. We know that no matter what follows, the present M2 will be stranded at the bottom of the stack since *love* is the root of the sentence and a word on one side of the root may not depend on a word on the other side of the root under normal circumstances. A molecule containing the root of a sentence is easily recognisable since its Negative-list will always be empty. There is a rule in WG which says that a finite verb has no head unless it is required to by the presence of some other element (eg. a complementizer).

When just this configuration is encountered - M1 is the root molecule and M2 is a stranded molecule at the bottom of the stack - the special purpose visitor relation is established between the root of M1 and the root of M2. M2 bonds with M1 in the usual way and a new element "[visitor, R1, R2]" is added to the Subordinates list, where R1 is the root of M1 and R2 is the root of M2. WG has a rule which says that where a word has a visitor and requires some other dependent whose slot-filling constraints are met by the visitor, then the visitor relation can be used to mediate the other dependency relation. The one constraint which is ignored when visitors are used to mediate other relations is the order constraint since visitors are used - by definition - when items have been "moved" from their normal position.

Usually dependency relations are established between separate molecules on the stack. Sometimes, however, they are established among sub-structures within a single molecule. Such non-standard dependencies are always derived from existing dependencies. The Derivable part of the molecular structure (see (11)) is reserved for rules which license derivable dependencies. (29) expresses a general rule of WG.

(29) (visitor of word) is (post-dependent of word)

This rule is recorded (in slightly modified form) in the Derivable list. The Derivable list is checked immediately after every bonding operation and any new dependencies which can be derived from that bond are established. Thus, since *love* requires an object (a kind of post-dependent) and it also has a visitor, (29) derives the fact that the object of *love* is identical to the

visitor of *love*. This fact is recorded in the usual way in the Subordinates list ([visitor, word-4, word-1] => [object, word-4, word-1]). The resulting structure is shown in (30).

(30)

Rules in the Derivable list are marked as optional or obligatory. Clearly, (29) would be marked as optional since we would not want it to work in sentence (31) as it did in sentence (30). This parser is just as susceptible to problems of ambiguity and indeterminacy as most other available parsers.

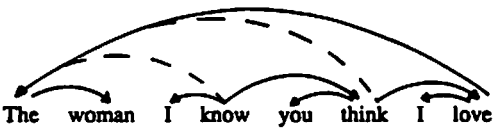
(31) The woman I love to meet

WG analyses sentences like (31) by allowing the visitor relation to "hop" down the dependency chain. The Derivable list contains another rule like the one in (32).

(32) (visitor of word) is (visitor of (complement of word))

This rule allows sentences like (33) to be analysed correctly. The broken lines show intermediate visitor relations which led to the final relation from word-8 to word-1.

(33)



Hudson has presented arguments for a visitor analysis (1984:124ff; 1988), but he has not yet published any procedural explanations of how to establish visitor relations in the first place. I would argue that the stack configuration illustrated in (28) can be recognised easily and used to establish the relation in one of the main cases where visitors are required.

3.9 Establishing double dependencies

We return to the problems raised by double dependency as exemplified by (3) and (4). We can solve the problems by including in the grammar the following propositions:

(34)
- i. derived-subject isa subject
- ii. (predicative of word) has (a derived-subject)
- iii. (incomplement of word) has (a derived-subject)
- iv. (derived-subject of word) is (subject of (head of word))

("Incomplement" is the name of the grammatical relation which links word pairs (2,3), (3,4), (4,5) and (5,6) in sentence (4). For more details of this analysis see Hudson, 1988).

Rule (34iv) can be built into the Derivable list at the time a word is read and a new molecule is first constructed. It serves to establish the most important and frequently occurring double dependencies and accounts for the analyses of (3a-b) and (4).

3.10 Establishing interdependence

The bonding algorithm I have presented so far simplifies the facts in assuming that once two words have bonded by means of some dependency relation they will form a single molecule, thus effectively preventing them from bonding with each other by means of some other dependency relation. As we have seen from example (7), WG theory allows interdependence to occur occasionally. It seems unlikely that this sort of problem is amenable to the range of solutions provided by the Derivable list since interdependence does not appear to be readily predictable from specific grammatical relations or stack configurations. The most effective solution promises to be the introduction of an explicit interdependence checker which operates immediately after every bonding operation. Unlike the other work presented here, this idea has not yet been implemented.

3.11 Implementation

The bonding algorithm has been implemented in a prototype parser written in the Prolog computer programming language and running on a Sun 3/52 workstation. The parser can analyse a wide range of English constructions while maintaining consistently high levels of efficiency. The stack-based algorithm ensures that the amount of work required to link two words remains virtually constant (0.23 - 0.25 secs) regardless of sentence length. In the prototype a vocabulary of approximately 500 lexical items has been used. Comparatively few lexical ambiguities have been included. In a large-scale system including multiple lexical ambiguities it is believed that the parser would run in polynomial time, probably proportional to n^3. However, until such time as a formal proof can be worked out this result must be regarded as tentative.

4. Word grammar parsing and categorial grammar parsing

A number of colleagues have commented on the apparent similarities between the parser presented here and some existing Categorial Grammar (CG) parsers (eg. Haddock, 1987; Pareschi, 1987). This should not be surprising since CG can be regarded as the middle ground between dependency and constituency grammar, incorporating insights from both. To aid comparison we shall briefly review some similarities and differences between the WG parser and most common CG parsers.

4.1 Similarities

We shall consider four main characteristics shared by the parsers.

i) Use of a parse stack;
ii) grammar and lexicon are not distinguished;
iii) reductions are performed on the basis of information local to the stack. It is not necessary to look up a grammar before reducing the top two stack elements;

iv) syntactic analysis and semantic interpretation proceed left-to-right incrementally (Ades and Steedman, 1982; Steedman, 1985).

(i) can not be regarded as significant since many parsers based on a wide variety of theories - including constituent-based theories (Aho & Ullman, 1972) - use parse stacks.

I know of no phrase structure grammar which has carried lexicalism to the lengths expressed in (ii). Of course, this is rightly a property of the theories of grammar on which the parsers are based, rather than of the parsers themselves.

(iii) can not be taken as seriously as it would have been several years ago since an increasing amount of work in phrase structure parsing is being based on the unification of complex feature sets - a move which results in the increased localisation of information and control in parsing.

(iv) observes that WG parsers and CG parsers both permit incremental analysis. This is made possible in CG by the use of the rule of function composition (X/Y + Y/Z => X/Z). WG has no rule corresponding to function composition. It succeeds in producing incremental analyses without special machinery precisely because it does not make use of phrasal constituents.

In the past, serious linguistic theories have failed to take sufficient account of the agreement between psycholinguistic studies and the common intuitions of hearers that interpretation proceeds incrementally (Marslen-Wilson, 1975; Marslen-Wilson & Tyler, 1980). However, now that incremental analysis is on the agenda it would be wrong to suppose that it is a special property of CGs. Any phrase structure grammar can build structure incrementally if analysis proceeds top-down. Even a bottom-up parser can analyse incrementally if a left-branching grammar is used. It is difficult to conceive of a sensible dependency grammar which could not be used to build structure incrementally.

In conclusion it might be said that while CG parsers and the present WG parser have a number of general properties in common, they share most of these properties with many of their contemporaries based on phrase structure grammar.

4.2 Differences

The most obvious and trivial difference between the parsers is that they are based on different theories. There are numerous superficial differences between WG and CG which are reflected in the parser:

- WG refers only to word-sized units, while CG refers directly to phrasal categories as well as words.
- WG refers directly to grammatical relations between words, CG does not.
- In most versions of CG the order of arguments relative to functions is fixed. Thus the function X/Y takes an argument Y which occurs to its right (Forward Application: X/Y + Y => X), while the function X\Y takes an argument Y which occurs to its left (Backward Application: Y + X\Y => X). Direction of application is built into the names of categories. In WG the relative order of

heads and dependents is listed as a property of the words and it may be left unspecified if word order is free. (This problem can be overcome in CG if directionality is separated from category labels in much the same way as immediate dominance and linear precedence have been separated in Generalised Phrase Structure Grammar (Gazdar et al., 1985). This adds complications to the otherwise elegant simplicity of CG).

- In some languages (eg. German) it may be necessary to say that a word has a number of complements, all of which follow it but whose order relative to each other is unimportant. In WG this can be expressed simply by specifying the complements and their order relative to the head but leaving their order relative to each other unspecified. In CG it is extremely difficult to do this without proliferating categories, eg. (A/B)/C or (A/C)/B.
- In case-marking languages (eg. Russian) word order can be relatively free. This poses a problem for configurational formalisms such as phrase structure grammar or CG. The formalisms give special value to concatenation while the languages do not. What these languages do is to make the dependency structure of clauses explicit by means of case and agreement markers. This is exactly the information which a dependency grammar expresses. WG and CG differ, then, in expressing different kinds of structural information. As Mel'čuk points out, the main logical operation in constituency-based grammars is set inclusion, while the main logical operation in dependency-based grammars is the establishing of binary relations (Mel'čuk, 1988: 13-14). A neglected argument in the constituency/ dependency debate is the fact that many languages mark dependencies overtly in their morphology while no language - to the best of my knowledge - includes overt morphological markers to signal the boundaries of constituents. (Of course constituent boundaries can sometimes be identified by virtue of the consistent peripheral location of morphologically-marked heads within their constituents but this is surely an artifact of a consistent head-first or head-last dependency system and not the result of overt constituent-boundary marking).

CG parsers can use a variety of rules to reduce the top two items on the stack (function application, function composition, type raising), whereas the WG parser has only a single reduction rule, namely the bonding rule. However, it is allowed to take stack configurations into account and to establish visitor relations and derived dependencies. CG parsers have nothing which corresponds to these.

The discussion so far has proceeded as though there existed a single undisputed version of CG. In fact there are many different versions currently being advanced, some of which display very different properties indeed. The variety which has attracted most attention and which seems to be making the most adventurous claims is the combinatory categorial grammar (CCG) developed by Mark Steedman and others (Steedman, 1987a). It is CCG which has introduced function application and type raising to deal with the kinds of constructions which involve "incomplete constituents", eg. coordination (Steedman, 1985; Dowty, 1987) and parasitic gaps (Steedman,

1987b). CCG makes use of the class of logical operations known as combinators (Turner, 1979). These can be used to define applicative systems such as the lambda calculus without the use of abstraction or bound variables. An interesting consequence of introducing combinatory rules is that for most sentences there are many possible sequences of application and composition which could generate exactly the same semantic structure. Thus, it is possible to parse a sentence so as to produce many syntactic structures but only one semantic structure. Left-to-right incremental interpretation is not "built in" to CCG, rather it so happens that a left-branching tree can be produced among a host of semantically equivalent trees for a sentence. This problem of **derivational equivalence** or **spurious ambiguity**, which has no correlate in WG, has obvious implications for the construction of CCG parsers. Pareschi and Steedman (1987) suggest using a chart parser which checks for semantic equivalence as well as syntactic identity before inserting an edge in the chart. Hepple and Morrill (1989) prefer to contract all equivalent structures to a "normal form" during parsing. Wittenburg (1987) tackles the problem in the grammar before parsing begins by deriving new devices called "predictive combinators" from the existing ones. This compiled version of the grammar is claimed not to produce spurious ambiguities. Other suggestions have been made and published contributions to the debate continue to appear. But it is a literature which has no twin in dependency grammar. The incremental interpretation of CG is bought at a price - spurious ambiguity; the incremental interpretation of WG comes free - because structure-building involves relating exactly two words directly, without the need to build intermediate non-terminal category structures.

It may be observed that the similarities noted between CG and WG parsers can be expressed at comparatively high levels of generality. In some cases it may be said that they reflect not just similar approaches in the two systems but general trends in contemporary linguistics. When we turn to specifics however, we find that both CG parsers and WG parsers make use of devices unique to themselves.

5. Conclusion

We began by considering some differences between phrase structure grammars and dependency structure grammars. We went on to look in some detail at the structure of a dependency parser based on the theory of WG and it is clearly distinct from any current phrase structure parser. CGs hold the middle ground, having some features in common with phrase structure grammars and other features in common with dependency structure grammars. However, we saw that the dependency parser presented here had a number of properties which distinguished it from CG parsers. It may also be observed that CG parsers deliver up what are effectively phrase structure trees. We saw in section 1 that at least some of the analyses of current dependency theories can not be mapped onto phrase structures.

Constituency and dependency are distinct notions. In highly constrained formal systems they may coincide to the extent that a structure expressed in terms of one may be mapped onto a structure expressed in terms of the other. The relaxation of constraints in order to write grammars for natural language renders such mappings impossible. Consequently, constituency

parsing and dependency parsing must develop as autonomous disciplines, each informed by the other but not dependent upon it.

Comparison of syntactic theories is a primitive art rather that an exact science. The comparison of parsers based on different theories may provide suggestive evidence to guide theory comparison. However, there exist few truly effective formal procedures for evaluating and comparing parsers based on the same theory, far less different theories. If the results of parser construction or theory construction are to be quantifiable then a great deal more work needs to be done on techniques of formal comparison in linguistics.

6. References

Ades, A.E. & M.J. Steedman (1982) "On the order of words", Linguistics and Philosophy, 4, 517-558.

Aho, A.V. & J.D. Ullman (1972) The Theory of Parsing, Translation, and Compiling, Vol. 1: Parsing. Englewood Cliffs, NJ: Prentice-Hall.

Covington, M.A. (1988) "Parsing variable word order languages with unification-based dependency grammar". ACMC Research Report 01-0022, University of Georgia, Athens, GA.

Dowty, D. (1987) "Type raising, functional composition, and non-constituent conjunction". In R. Oehrle, E. Bach, & D. Wheeler (eds) Categorial Structures and Natural Language Structures. Dordrecht: Reidel.

Flickinger, D.P. (1987) "Lexical rules in the hierarchical lexicon". Stanford University PhD thesis.

Fraser, N.M. (1985) "A Word Grammar Parser". MSc dissertation, University College London.

Gaifman, H. (1965) "Dependency systems and phrase-structure systems", Information and Control, 8, 304-337.

Gazdar, G. (1988) "The organization of computational lexicons". CSRP 99, School of Cognitive Sciences, University of Sussex.

Gazdar, G., E. Klein, G. Pullum & I. Sag (1985) Generalised Phrase Structure Grammar. Oxford: Blackwell.

Haddock, N. (1987) "Incremental interpretation and combinatory categorial grammar", Edinburgh Working Papers in Cognitive Science, 1: Categorial Grammar, Unification Grammar and Parsing, 71-84. Centre for Cognitive Science, University of Edinburgh.

Hays, D.G. (1964) "Dependency theory: A formalism and some observations", Language, 40, 511-525.

Hays, D.G. (1966) "Parsing". In D.G. Hays (ed) Readings in Automatic Language Processing. New York: American Elsevier.

Hellwig, P. (1980) "PLAIN - A program system for dependency analysis and for simulating natural language inference". In L. Bolc (ed) Representation and Processing of Natural Language. Munchen: Carl Hanser Verlag.

Hepple, M. & G. Morrill (1989) "Parsing and derivational equivalence", Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics, Manchester. 10-18.

Hudson, R.A. (1984) Word Grammar. Oxford: Blackwell.

Hudson, R.A. (1986) "A Prolog implementation of Word Grammar", Speech, Hearing and Language: Work in Progress 2, Department of Phonetics and Linguistics, University College London, 133-150.

Hudson, R.A. (1988) "Extraction and grammatical relations", Lingua 76, 177-208.

Hudson, R.A. (forthcoming) "Recent developments in dependency theory". In J. Jacobs, W. Sternefeld, T. Vennemann & A. von Stechow (eds) Syntax. An International Handbook of Contemporary Research. Walter de Gruyer & Co.

Hudson, R.A. & W. van Langendonck (forthcoming) "Word grammar". In F. Droste (ed) Mainstreams in Linguistics.

Kunze, J. (1982) Automatische Analyse des Deutschen. Berlin: Akademie-Verlag.

Jackendoff, R. (1977) $\bar{X}$ Syntax: a study of phrase structure. Cambridge, Mass.: MIT Press.

Langacker, R.W. (forthcoming) "Cognitive grammar". In F. Droste (ed), Mainstreams in Linguistics.

Levelt, W.J.M. (1974) Formal Grammars in Linguistics and Psycholinguistics, Vol. II: Applications in linguistic theory. The Hague: Mouton.

Marslen-Wilson, W. (1975) "The limited compatibility of linguistic and perceptual explanations", Chicago Linguistic Society Parasession on Functionalism, 409-420.

Marslen-Wilson, W. & L.K. Tyler (1980) "The temporal structure of spoken language understanding", Cognition, 8, 1-74.

Mel'čuk, I.A. (1988) Dependency Syntax: Theory and Practice. Albany, N.Y.: SUNY.

Pareschi, R. (1987) "Combinatory grammar, logic programming, and natural language processing", Edinburgh Working Papers in Cognitive Science, 1: Categorial Grammar, Unification Grammar and Parsing, 85-114. Centre for Cognitive Science, University of Edinburgh.

Pollard, C. & I.A. Sag (1987) Information-based Syntax and Semantics, Vol. 1: Fundamentals (CSLI Lecture Notes, 13). Stanford: CSLI.

Robinson, J.J. (1970) "Dependency structures and transformational rules". Language, 46, 259-285.

Sgall, P., E. Hajicova & J. Panevova (1986) The Meaning of the Sentence in its Semantic and Pragmatic Aspects. Prague: Academia.

Shieber, S.M. (1986) An Introduction to Unification-Based Approaches to Grammar (CSLI Lecture Notes, 4). Stanford: CSLI.

Starosta, S. (1988) The Case for Lexicase. London: Pinter.

Starosta, S. & H. Nomura (1986) "Lexicase Parsing: a lexicon-driven approach to syntactic analysis". In M. Nagao (ed) Proceedings of the Eleventh International Conference on Computational Linguistics (COLING '86), Bonn. 127-132.

Steedman, M.J. (1985) "Dependency and co-ordination in the grammar of Dutch and English", Language, 61, 523-568.

Steedman, M.J. (1987a) "Combinators and grammars". In R. Oehrle, E. Bach, & D. Wheeler (eds) Categorial Structures and Natural Language Structures. Dordrecht: Reidel.

Steedman, M.J. (1987b) "Combinatory grammars and parasitic gaps", Edinburgh Working Papers in Cognitive Science, 1: Categorial Grammar, Unification Grammar and Parsing, 30-70. Centre for Cognitive Science, University of Edinburgh.

Pareschi, R. & M.J. Steedman (1987) "A lazy way to chart-parse with categorial grammars", Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics, Stanford. 81-88.

Turner, D.A. (1979) "A new implementation technique for applicative languages", Software: Practice and Experience, 9, 31-49.

Valkonen, K., H. Jappinen, A. Lehtola & M. Ylilammi (1987) "Declarative model for dependency parsing - a view into blackboard methodology", Proceedings of the Third Conference of the European Chapter of the Association for Computational Linguistics, Copenhagen. 218-225.

Wittenburg, K. (1987) "Predictive combinators: a method for efficient processing of combinatory categorial grammar", Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics, 73-80.