DAVID MILWARD

DYNAMIC DEPENDENCY GRAMMAR*

1. INTRODUCTION

Dynamics is the formal study of systems involving states and transitions between states. A natural application of dynamics is to the study of language processing, where words or morphemes can be thought of as actions which perform transitions between states of the language processor. The paper concentrates on sentence as opposed to text processing, and shows how dynamics can lead to novel parsing algorithms and to new possibilities for the formal description of syntactic constructions, in particular, for the description of non-constituent coordination.

The use of dynamics in algorithm development will be illustrated by presenting a particular parsing algorithm which was developed for a 'core' lexicalised grammar, *Lexicalised Dependency Grammar* (Milward 1992) which resembles both simplified HPSG (Pollard and Sag 1993) and dependency grammar as formalised by Gaifman (see Hays 1964). The algorithm is fully incremental, providing a semantic representation word by word. It also exhibits reduced non-determinism relative to other almost-incremental algorithms[1] by using types instead of partial parse trees. These can be packed further using graph structuring (cf. Tomita 1985).

Although dynamics specifies the states of a process and the possible mappings between states, it does not specify the control strategy (how the state space is traversed). Suitable languages for dynamics are thus both formal and declarative, and can be used to express linguistic generalisations. The final part of the paper regards the dynamics provided for the core lexicalised grammar as a grammar in its own right, *Dynamic Dependency Grammar*, and considers extending this to deal with constructions which are

*I would like to thank John Carroll, Robin Cooper, Jonathan Ginzburg, Joris Hulstijn, Carl Vogel and the anonymous reviewers for helpful comments on an earlier version of this paper. Thanks are also due to Steve Pulman, Ewan Klein, David Beaver and Guy Barry for discussion during the early stages of the work, and to other members of the University of Edinburgh Centre for Cognitive Science and the University of Cambridge Computer Laboratory. The research was supported by the British Science and Engineering Research Council (Research Fellowship B/90/ITF/288, and Research Grant RR30718)

[1]Most incremental algorithms fall into two camps, those that process word by word, but are incomplete e.g. Moortgat's M-System (Moortgat 1988) and Stabler's top down parser (Stabler 1991), and those that are complete, but do not always process word by word (e.g. Pulman 1986).

difficult to encode in a purely lexicalist framework. As an example, it briefly describes an analysis of English non-constituent coordination. The analysis has much in common with both grammar based accounts of coordination (e.g. Gazdar 1981, Steedman 1985), and procedural accounts such as SYSCONJ (Woods 1973).

2. DYNAMIC SPECIFICATIONS

This section provides a formal notation for describing the dynamics of a process i.e. the states and the possible transitions between states. The notation is designed to enable modelling of the transitions between abstract characterisations of 'mental' states during language processing.

Traditional automata theory (see e.g. Arbib 1969) provides three 'standard' processing models, the finite state machine (FSM), the push-down automaton (PDA) and the Turing Machine (TM). These include state descriptions of increasing complexity. For FSMs, states are distinguished according to their label, which is chosen from a finite set of symbols. For PDAs, each label is a pair consisting of an element from a finite set, and a list of symbols of arbitrary length (the stack). The symbols on the stack are also chosen from a finite set. In the closely related Recursive Transitions Networks (Woods 1973), each element on the stack is a syntactic category (e.g. **s**, **np** or **vp**). For TMs, each label is a pair of an element from a finite set, and a tape, unbounded to left and right (the tape can be replaced by two arbitrary length lists).

Although TMs have the most complex state labels of the three classes of automata, these labels are still relatively unstructured in comparison to the data structures (such as trees, graphs or charts), which are employed in many psychological or computational models of language processing. Calculations using TMs often involve complicated series of transitions as a consequence of the lack of structure in the state labels.

Specification of the dynamics of a process, *Dynamic Specifications*, can be provided by using simple logics, resembling Floyd-Hoare Logic (Hoare 1969).[2] These logics provide a convenient formalism in which to describe the dynamics of processes where the

[2]An alternative to using logics is to use relational algebra (see e.g. van Benthem 1991 for use of relational algebra in dynamics). However, standard relational algebra seems to be simultaneously too rich and not rich enough. For example, negation is included, but does seem to be required either in the dynamic specifications or the dynamic grammars described later, and the parallelism operator, intersection, would need to be replaced by at least two different operators to deal with the different semantics of **or** and **and**.

complexity is in the state descriptions rather than in the transitions. They are also of sufficient generality to provide descriptions of the dynamics of FSMs, PDAs and TMs, and to form the basis for the *Dynamic Grammars* which will be described later. The full specifications consist of a logic which specifies possible transitions between states, and a set of pairs of states, specifying possible pairings of initial and final states.

The general form of the specifications is given in Figure 1[3] and this will be illustrated by two examples, a specification of a particular FSM, and a specification of a shift reduce recogniser.

1. A set of states (not necessarily finite).
2. A set of axioms of the form,[a]

   $$S_i \ \text{String} \ S_j$$

3. A deduction rule, Sequencing,

   $$\frac{S_i \ \text{String}_a \ S_j \qquad S_j \ \text{String}_b \ S_k}{S_i \ \text{String}_a \bullet \text{String}_b \ S_k}$$

4. A set of pairs of states.

Fig. 1. Dynamic Specifications

[a] Notation: capital letters are used for all variables. A subscripted **S** stands for a state, which may be a structured object which includes free variables. **String** (whether subscripted or not) stands for strings of symbols from a finite set. '•' denotes string and list concatenation. For example, if String_a = "kl" and String_b = "mn", then $\text{String}_a \bullet \text{String}_b$ = "klmn", and if $\text{List}_a = \langle \text{np} \rangle$ and $\text{List}_b = \langle \text{v,s} \rangle$ then $\text{List}_a \bullet \text{List}_b = \langle \text{np,v,s} \rangle$.

Consider the finite state machine, FSM(1), given in Figure 2. This accepts any string which maps from the initial state, 0, to the final state, 3 (i.e. strings of the form: $\mathbf{ab^*cb}$). The dynamic specification is given in Figure 3.

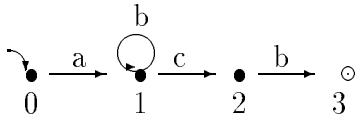


Fig. 2. FSM(1)

[3] An even more general form of specification would leave the deduction rules unspecified. The use of a single deduction rule, Sequencing, corresponds to only considering sequential processing.

1. A set of states, {0,1,2,3}

2. A set of axioms,
   {0 "a" 1, 1 "b" 1, 2 "b" 3, 1 "c" 2}

3. A deduction rule, Sequencing,[a]

$$\frac{S_i\ \mathrm{String}_a\ S_j \qquad S_j\ \mathrm{String}_b\ S_k}{S_i\ \mathrm{String}_a \bullet \mathrm{String}_b\ S_k}$$

4. A set of pairs of states, { (0,3) }

Fig. 3. Dynamic Specification of FSM(1)

[a] $\mathbf{S}_i$ stand for labels from the set {0,1,2,3}, $\mathbf{String}_i$ are strings of letters.

Each transition in the diagram is specified using an axiom of the form $\mathbf{S}_i$ **String** $\mathbf{S}_j$. These provide 'primitive' proofs which can be put together using the Sequencing Rule[4] to form larger proofs. For example, the two axioms, **1 "c" 2** and **2 "b" 3** can be combined to give a proof of the statement, **1 "cb" 3** i.e., a proof that the string of letters **"cb"** performs a transition between states **1** and **3**

$$\frac{1\ \text{“c”}\ 2 \qquad 2\ \text{“b”}\ 3}{1\ \text{“cb”}\ 3}$$

The acceptance of a string is modelled by a proof that a string performs a transition between a specified pair of states (in this case there is a single specified pair, **(0,3)**, where **0** is the initial state of the FSM, **3** the final state). Thus, a string, **Str**, is accepted if and only if it is possible to construct a proof of the statement **0 Str 3** using the axioms and the Sequencing Rule. For example, the string **"abbcb"** has the proof given in Figure 4, amongst others.

$$\cfrac{\cfrac{\cfrac{1\ \text{“b”}\ 1 \quad 1\ \text{“b”}\ 1}{1\ \text{“bb”}\ 1} \qquad 1\ \text{“c”}\ 2}{0\ \text{“a”}\ 1 \qquad 1\ \text{“bbc”}\ 2}}{0\ \text{“abbc”}\ 2 \qquad 2\ \text{“b”}\ 3}$$
$$0\ \text{“abbcb”}\ 3$$

Fig. 4. Example Proof

In this example, the specification of dynamics was constructed from a pre-existing FSM, which could be directly implemented. However, in later parts of the paper the dynamics will be specified first, and from this an implementation will need to be constructed. How can this be done? The obvious implementation of the logic is to use a goal-driven theorem prover. This is fine for the finite state recogniser above, where there is no output, and

[4] The name, Sequencing, is adopted from a similar rule in Floyd-Hoare Logic

only one successful proof need be found. It is not satisfactory if all possible outputs are required for a given input. The problem is the multiple different ways of proving the same statement ('spurious' ambiguity).

Fortunately, in the case of dynamic specifications, there is 'total' spurious ambiguity: a proof can be obtained with left branching structure, right branching structure or any mixture of the two. The order in which strings are combined by the Sequencing Rule is immaterial. It is therefore possible to choose a particular *normal form* (e.g. fully left branching proofs). Alternatively, it is possible to work with a representation for a set of proofs which abstracts from the order of combination. An appropriate representation provides the intermediate states during a proof, but none of the tree structure. For example, the following representation can be used to represent the set of proofs of **"abbcb"**:

$$0 \text{ “a” } 1 \text{ “b” } 1 \text{ “b” } 1 \text{ “c” } 2 \text{ “b” } 3$$

Representations of this sort (equivalence classes of proofs) can be constructed from left to right, by starting from an initial state, and applying the axioms one letter at a time. Alternatively, they can be constructed from right to left starting from a final state. This bidirectionality is due to the fact that the logic does not encode anything about the traversal of the state space. Each axiom merely encodes which states are related to which other states by which letters.

Although the logic does not impose order on the construction of equivalence classes, it is possible to interpret the left to right ordering of the states as sequential order. What is obtained is a trace of the states through which the FSM passes, a *transition sequence*. Transition sequences can be represented by adding directed arrows to the equivalence proofs e.g.

$$0 \overset{\text{“a”}}{\rightarrow} 1 \overset{\text{“b”}}{\rightarrow} 1 \overset{\text{“b”}}{\rightarrow} 1 \overset{\text{“c”}}{\rightarrow} 2 \overset{\text{“b”}}{\rightarrow} 3$$

Section 9 describes how the use of transition sequences in sentence processing enables semantic representation to be built up word by word from left to right (corresponding to the construction of a left branching proof), and also allows extraction of the transitions performed by any final substring which might act as a conjunct (corresponding to the use of a right branching proof).

A simple example of a system requiring an infinite number of possible states is provided by shift reduce parsing, or, even more simply, by shift reduce recognition. Consider

a shift reduce recogniser for the grammar in Figure 5.

s $\rightarrow$ np vp	John : np
vp $\rightarrow$ v np	likes: v

Fig. 5. PSG(1)

Each state can be represented by a list of categories (the stack). An example transition sequence is the following:

$$\langle\rangle \overset{\text{“John”}}{\rightarrow} \langle \text{np} \rangle \overset{\text{“likes”}}{\rightarrow} \langle \text{v,np} \rangle \overset{\text{“John”}}{\rightarrow} \langle \text{np,v,np} \rangle \overset{\text{“”}}{\rightarrow} \langle \text{vp,np} \rangle \overset{\text{“”}}{\rightarrow} \langle \text{s} \rangle$$

The initial stack is empty. A noun phrase is then shifted onto the stack, then a verb, then another noun phrase. The top element on the stack (the head of the list) is the leftmost element, so categories on the stack appear in the reverse order to the order in which they were shifted. Finally, two reductions take place.

To deal with arbitrarily deep right recursion or centre embedding, the shift reduce recogniser performs an arbitrarily large number of shifts (each increasing the stack length by one), before doing any reductions. Hence, the size of the stack may be arbitrarily large, requiring an infinite number of possible states.

The dynamics for the shift reduce recogniser for this grammar is given by Figure 6.

1. A set of states, {L | L a list over {np,v,s}}
2. A set of axioms,

   L “John” $\langle \text{np} \rangle \bullet$L
   L “likes” $\langle \text{v} \rangle \bullet$L
   $\langle \text{vp,np} \rangle \bullet \mathbf{L}$ “” $\langle \text{s} \rangle \bullet \mathbf{L}$
   $\langle \text{np, v} \rangle \bullet \mathbf{L}$ “” $\langle \text{vp} \rangle \bullet \mathbf{L}$

3. A deduction rule, Sequencing,

$$\frac{S_i \ \text{String}_a \ S_j \qquad S_j \ \text{String}_b \ S_k}{S_i \ \text{String}_a \bullet \text{String}_b \ S_k}$$

4. A set of pairs of states: { ($\langle\rangle$,$\langle$s$\rangle$) }

Fig. 6. Dynamic Specification of Shift Reduce Recognition for PSG(1)

The first two axioms correspond to the shift steps of the recogniser. Shifting is possible whatever the state of the stack, so the axioms (or, to be more precise, axiom *schemata*) contain a variable, L, standing for any list. An example instantiation of the second axiom schema is the following, in which **L** is set to the list $\langle \mathbf{np} \rangle$:

$$\langle \text{np} \rangle \text{ “likes” } \langle \text{v,np} \rangle$$

The next two axiom schemata correspond to the reduce steps of the recogniser. Reduction

occurs without any further input, so the axiom schemata are statements concerning the empty string (they characterise *empty transitions*[5]). Since there can be an arbitrarily long series of reduce steps (for example, after an arbitrarily long series of shifts) the empty transitions are not eliminable (unlike empty transitions in FSMs).

Now consider specifying the dynamics of shift reduce recognition for an arbitrary phrase structure grammar, **G**. Each lexical entry, **W:X** provides potential shift steps, which can be captured by the axiom schemata:

$$\text{L “W” } \langle \text{X} \rangle \bullet \text{L}$$

Each grammar rule, $\mathbf{C}_0 \rightarrow \mathbf{C}_1 \ \ldots \ \mathbf{C}_n$ provides potential reduce steps, which can be captured by the schemata:

$$\langle \text{C}_n \ \ldots \ \text{C}_1 \rangle \bullet \mathbf{L} \text{ “” } \langle \text{C}_0 \rangle \bullet \mathbf{L}$$

The full dynamic specification is given in Figure 7.

1. A set of states, {L | L a list of categories in G}

2. A set of axiom schemata,
   L “W” $\langle$X$\rangle\bullet$L where W: X in G
   $\langle \text{C}_n \ \ldots \ \text{C}_1 \rangle \bullet \mathbf{L}$ “” $\langle \text{C}_0 \rangle \bullet \mathbf{L}$ where $\text{C}_0 \rightarrow \text{C}_1 \ \ldots \ \text{C}_n$ in G

3. A deduction rule, Sequencing,

$$\frac{\text{S}_i \ \text{String}_a \ \text{S}_j \qquad \text{S}_j \ \text{String}_b \ \text{S}_k}{\text{S}_i \ \text{String}_a \bullet \text{String}_b \ \text{S}_k}$$

4. A set of pairs of states: { ($\langle$,$\langle$s$\rangle$) }

Fig. 7. Dynamic Specification of Shift Reduce Recognition

Finally, it is worth briefly remarking that empty transitions are not required for the Dynamic Dependency Grammar described in later sections, however they may turn out to be of use for extensions to the grammar (e.g. for the treatment of wh-movement (Milward 1991)).

3. SENTENCE PROCESSING

The simplest dynamic model of sentence processing takes each word to be an ‘action’, mapping between information states containing both syntactic and semantic information. For example, processing a fragment *saw Ben today* involves mapping from some information state $\mathbf{S}_1$ to an information state $\mathbf{S}_4$:

[5]Otherwise known as *silent transitions*.

$$S_1 \xrightarrow{\text{“saw”}} S_2 \xrightarrow{\text{“Ben”}} S_3 \xrightarrow{\text{“today”}} S_4$$

The model can be extended to allow non-determinism, i.e. more than one possible transition at each node. For example, the string **“yesterday”** might relate some state $\mathbf{S}_4$ to both state $\mathbf{S}_5$ and $\mathbf{S}_{5'}$ (the two states being distinguished, for example, by having different semantic values corresponding to **yesterday** modifying the main sentence, or the embedded sentence). Two possible transition sequences are as follows:

$$S_0 \xrightarrow{\text{“John”}} S_1 \xrightarrow{\text{“believed”}} S_2 \xrightarrow{\text{“Mary”}} S_3 \xrightarrow{\text{“slept”}} S_4 \xrightarrow{\text{“yesterday”}} S_5$$

$$S_0 \xrightarrow{\text{“John”}} S_1 \xrightarrow{\text{“believed”}} S_2 \xrightarrow{\text{“Mary”}} S_3 \xrightarrow{\text{slept}} S_4 \xrightarrow{\text{“yesterday”}} S_{5'}$$

These can be represented more compactly as follows:

$$S_0 \xrightarrow{\text{“John”}} S_1 \xrightarrow{\text{“believed”}} S_2 \xrightarrow{\text{“Mary”}} S_3 \xrightarrow{\text{“slept”}} S_4 \begin{array}{l} \xrightarrow{\text{“yesterday”}} S_5 \\ \xrightarrow{\text{“yesterday”}} S_{5'} \end{array}$$

State S_4 is a choice point on input **“yesterday”**. Choice points can be used as the basis for modelling garden paths (where, in serial models, the wrong choice is made).

The kind of information state which might be expected in a dynamic model of sentence processing can be restricted by making various assumptions. Here four assumptions will be considered: that the number of possible states in any model is infinite, that the number of branches at any choice point is finite, that there are a finite number of transitions on finite input, and that interpretation is incremental.

It is well known that a system with a finite number of states cannot recognise a context free grammar unless there is a limit on centre embedding. Moreover, a system with a finite number of states cannot parse a context free grammar unless there is also a limit on both left and right recursion (Langendoen 1975; see e.g. Pulman 1986 for discussion). Intuitively, each iteration during a left or right recursion adds to a syntax tree and hence changes the information which needs to be contained within the state. Hence the dynamic model will be assumed to have an infinite number of possible states. Limitations on processing of recursion will be assumed to be caused by memory allocation running out during a particular computation on a particular finite machine, instead of an arbitrary decision being made when specifying the model.

The second assumption is that, on performing a transition from a particular state, on a particular input, there are only a finite number of possible next states. The third

assumption is that the number of transitions performed is finite provided the input string is finite (this assumption is trivially satisfied if there are no empty transitions, since then the number of transitions equals the number of words). The second and third assumptions together ensure that complete exploration of the search space is possible, and hence that processing terminates on finite input.

Even with the addition of these first three assumptions, the dynamic model is still a very general one, including, for example, the dynamic specification of shift reduce recognition which was given earlier. A more restrictive notion of what constitutes a dynamic model of sentence processing is obtained by adopting a fourth assumption, that interpretation is *incremental*. Psycholinguistic evidence (e.g. Marslen-Wilson 1973, Tanenhaus, Garnsey and Boland 1990) suggests that interpretation occurs before constituent boundaries, possibly word by word. The stronger position of word by word interpretation is advocated by Just & Carpenter (1980), though limited to content words:

> ... a reader tries to interpret each content word of a text as it is encountered, even at the expense of making guesses that sometimes turn out to be wrong. Interpretation refers to processing at several levels such as encoding the word, choosing one meaning of it, assigning it to its referent, and determining its status in the sentence and in the discourse.

The assumption adopted here is that a non-trivial semantic representation is built word by word[6]. The construction of a representation enables, but does not force, the processor to perform immediate evaluation (of e.g. definite descriptions), or to immediately use the representation in inference or in judging plausibility (Milward and Cooper 1994).

It is worth noting that the assumption that semantic representations are built word by word is stronger than that adopted by some recent work in the categorial grammar tradition (e.g. Pickering 1991, van der Linden 1993). Here, the fragments formed by the grammar and during processing can have missing dependents, such as the arguments of a verb, but not missing heads, such as the verb itself. At present there is no psycholinguistic evidence either for or against the 'head first' proposal, and there seems to be some intuitive evidence against. Unless elaborated, the proposal predicts that there should be

[6] What constitutes a non-trivial representation is debatable. The position taken here is that it must use all the information given so far. Thus, an acceptable representation for the sentence fragment **John likes** would be λ**x. likes'(john',x)** but not a semantic product such as **john'**$*\lambda$**y.**λ**x.likes'(x,y)**. In the product, **john'** is paired with λ**y.**λ**x.likes'(x,y)** but there is no linking of **john'** with the first argument role (the agent role) of **likes'**. Product operators were introduced by Lambek (1958).

no on-line plausibility contrast between the following pairs of examples:

1) a John ate the food which the cat ...
   b John ate the food which the neighbour ...
   c The politician failed to get in because he was advised by a brilliant and courageous ...
   d The politician failed to get in because he was advised by a lazy and disorganised ...
   e Sue believes that a horse with green hair ...
   f Sue can't believe that a horse with green hair ...

In the first two examples, the head of the relative clause (the verb) is yet to appear, so *the cat* cannot be integrated with the rest of the fragment. In the third and fourth examples, *lazy and disorganised* and *cunning and courageous* cannot be integrated since the noun is yet to appear (both Pickering and van der Linden treat adjectives as dependents of nouns). In the fifth and sixth examples[7], the head of the complement sentence is yet to appear, so the noun phrase *a horse with green hair* would be interpreted outside the context set up by the first part of the sentence.

Traditional layered models of sentence processing first build a full syntax tree, and then extract a semantic representation from this. Adapting this to an incremental perspective suggests a model in which each new word creates a new fragment of syntax tree, and from this a semantic representation is extracted i.e.

$$\begin{array}{ccccc} \ldots \rightarrow \text{Tree fragment}_i & & \rightarrow \ldots \rightarrow \text{Completed tree} \\ \downarrow & & \downarrow \\ \text{Semantic representation}_i & & \text{Propositional representation} \end{array}$$

Such a model can be contrasted with e.g. a shift reduce parser, where, at intermediate states in the parse, there is often a collection of separate tree fragments, from which no integrated single semantic representation can be formed.

However, this particular incremental model is not possible given the possibility of left recursion, and given the assumption made earlier that choice points have finite branching. Consider the sentence fragment:

Mary thinks John

The tree structure of the fragment can be represented as in Figure 8. (A broken line is used since there may be intermediate nodes between the **np** and the **s**, i.e. the **s** dominates the **np**, but does not necessarily immediately dominate it).

[7]Thanks are due to Matt Crocker for pointing out examples of this kind.

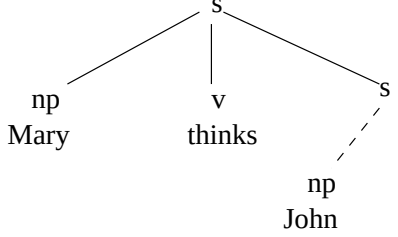


Fig. 8

For languages such as English, where left recursion is possible, the completed sentence may have an arbitrarily large number of intermediate nodes between the **s** and the **np**. For example, **John** could be embedded within a gerund e.g. *Mary thinks John leaving here was a mistake*, and this in turn could be embedded e.g. *Mary thinks John leaving here being a mistake is surprising.* **John** could also be embedded within a sentence which has a sentence modifier requiring its own **s** node e.g. *Mary thinks John will go home probably*[8], and this can be further embedded e.g. *Mary thinks John will go home probably because he is tired.*

If the completed sentence can have an arbitrarily large number of intermediate nodes, it is not possible to provide a finite set of tree fragments after processing *Mary thinks John* which will be compatible with all continuations. To retain completeness, the simple incremental model requires infinite branching, which violates the second assumption.

Note that this problem is not due to the use of a dynamic model of sentence processing. Rather, it is an inherent problem in processing word by word, which is shown up when the dynamics is considered. It helps to explain why other approaches to incremental interpretation have either been word by word and incomplete (e.g. Stabler's top down parser (Stabler 1991) or Moortgat's M-System (Moortgat 1988)[9]), complete, but not fully word by word (e.g. Pulman 1985, 1986), or have abandoned traditional grammatical structure (e.g. Hausser 1989)[10], with attendant difficulties.

Once the problem is isolated, it is not so difficult to resolve. What is required is a way of encoding up an infinite number of tree fragments. There are several possibilities. The first is to use a language describing trees which can express the fact that **John** is dominated by the **s** node, but does not have to specify what it is immediately dominated by (e.g. D-Theory, Marcus et al. 1983, or Structure Unification Grammar, Henderson

[8]The treatment of **probably** as a modifier of a sentence is perhaps controversial. However, treatment of it as a verb phrase modifier would merely shift the potential left recursion to the verb phrase node.

[9]In the Lambek Calculus, the problem of an infinite number of possible tree fragments is replaced by a corresponding problem of initial fragments having an infinite number of possible types.

[10]Hausser's Left Associative Grammars are fully word by word, with words mapping between states. Each state consists of a flat list of categories, paired with a set of rules (the rules which can be applied next).

1990). A semantic representation can then be formed word by word by extracting a particular, 'default', syntax tree from a description of a set of trees (Marcus et al. take the default tree to be the tree formed by strengthening dominance to immediate dominance wherever possible).

A second possibility is to pack together trees which share the same recursive structure. Thompson et al. (1991), show how this can be done for a phrase structure grammar, thereby allowing a generalised version of left-corner parsing, in which left corners may be arbitrarily deeply embedded. The grammar is first converted into an equivalent Tree Adjoining Grammar (Joshi 1987), where recursion is factored out by the introduction of an adjunction operation. During processing, although a large number of TAG tree fragments may be constructed, the number is still finite (each TAG tree represents a possibly infinite number of standard trees which can be obtained from it using adjunction). Default semantic values could be extracted incrementally by assuming no adjunctions are to be performed. A somewhat similar system has recently been proposed by Shieber and Johnson (1993).

A third possibility, the one which will be explored in detail here, provides representations of sets of partial trees in which all the trees in the set have the same semantics. There is no need for default semantic values. To see this is a possible approach, reconsider the fragment *Mary thinks John*. Although there are any number of tree fragments, these can be given a finite number of semantic representations in a semantic theory which includes higher order abstraction. For example, using simplified Montague Semantics (see e.g. Dowty et al. 1981), the tree fragments discussed earlier for *Mary thinks John* can be assigned the following lambda expression, with type $\langle\langle \mathbf{e},\mathbf{t}\rangle,\mathbf{t}\rangle$:

$$\lambda \text{P. thinks'(mary',P(john'))}$$

This lambda expression can be thought of as a way of encoding an infinite set of fragments of semantic (tree) structure: the eventual semantic structure may embed **john'** at any depth e.g.

$$\text{thinks'(mary',sleeps'(john'))}$$
$$\text{thinks'(mary',probably'(sleeps'(john')))}$$
$$\text{etc.}$$

One possible representation for the sets of tree structures is some syntactic correlate of lambda expressions. In practice, however, if all that is required is a mapping between a

string of words and a semantic representation (and no explicit syntax trees need to be constructed), it is possible to use the types of the 'syntactic lambda expressions', rather than the expressions themselves. In the dynamic model described in Section 5, there is simultaneous construction of syntactic types and semantic values (lambda expressions), but no construction of syntactic values (syntactic lambda expressions, or sets of tree fragments). Thus individual transitions are of the form:

$$\begin{array}{c} \text{Syntactic type}_i \\ \text{Semantic rep}_i \end{array} \rightarrow \begin{array}{c} \text{Syntactic type}_{i+1} \\ \text{Semantic rep}_{i+1} \end{array}$$

4. LEXICALISED DEPENDENCY GRAMMAR (LDG)

This section briefly presents a 'core' lexicalised grammar, *Lexicalised Dependency Grammar* (LDG), for which a processing model will be provided in Section 5. LDG is a very simple feature based grammar, where structure is entirely given by subcategorisation. Each word is assigned one or more types which encode how the word can combine to form a phrase. For example, a verb such as **showed** is given a type which encodes the information that it can combine with a noun phrase on the left and a noun phrase and a prepositional phrase on the right to form a sentence. The lexical entry is as follows:

$$\text{showed} : \left[\begin{array}{l} s \\ \mathtt{l}\langle np \rangle \\ \mathtt{r}\langle np,\ pp \rangle \end{array}\right]$$

The feature **l** contains the list of arguments to be found on the left, and **r** the list to be found on the right.[11] The sentence *John thought Mary showed Ben to Sue* is given the tree structure in Figure 9. The tree structure is particularly flat, with all arguments of a verb appearing at the same level. This contrasts with standard phrase structure analyses, in which the subject of **showed** would appear at a different level from its objects. It also contrasts with the analysis given by Head Driven Phrase Structure Grammar (Pollard & Sag 1993). Although Borsley's variation of HPSG (Pollard & Sag 1993: Chapter 9) has a similar division of argument lists (with the subject of a verb on one list, the objects on the other), there are separate combination rules (or rather constraints) for the two lists, ensuring a verb phrase node is created before the subject is attached.

[11] A finite number of features are assumed. In this paper the only ones considered are the category feature (this appears as the top line without a feature name), the **l** feature and the **r** feature.

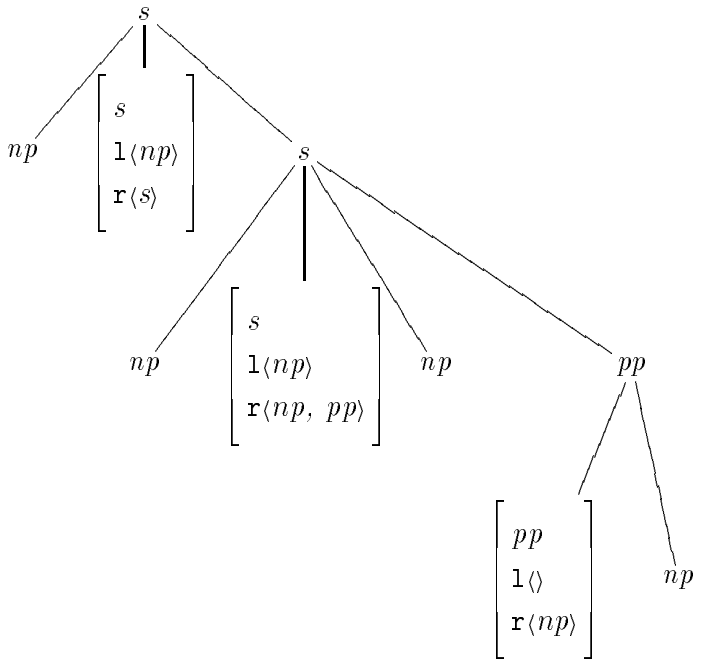


Fig. 9. LDG Analysis of *John thought Mary showed Ben to Sue*

Since structure is determined entirely by subcategorisation, LDGs can be classified as dependency grammars. In fact, they are a straightforward lexicalisation of the formal dependency grammars of Gaifman (see Hays 1964). Instead of a tree structure, a dependency graph can be drawn as in Figure 10.

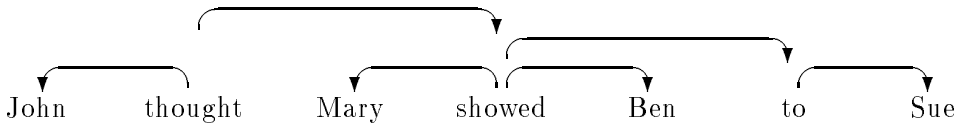


Fig. 10. Dependency Graph

The word **thought** is the head of the whole sentence, and it has two dependents, **John** and **showed**. **showed** is the head of the embedded sentence, with three dependents, **Mary**, **Ben** and **to**. The order of the dependents is fixed, with arguments at the top of a subcategorisation list appearing closest to their *head* (e.g. the **np**, **Ben**, appears closer to **showed** than the preposition, **to**).

By being simultaneously a constituent based formalism and a dependency based formalism, LDGs lose out on much of the descriptive apparatus which normally keeps the two formalisms apart. For example, the possibility of intermediate constituent structure (such as verb phrases) in constituent grammars, or the possibility of free word order,

lack of adjacency[12], or multiple headedness in dependency grammars (e.g. Hudson 1990, Mel'čuk 1988). These descriptive possibilities are only partly compensated for by the incorporation of a feature mechanism. Although LDGs might be regarded as overly simple, they do provide a full range of recursive structures (left and right recursion and center embedding), and, from their equivalence to formal dependency grammar, are weakly context free (Gaifman 1965). They thus provide a reasonable test for the use of dynamics in specifying incremental processing.

It is perhaps worth briefly outlining the relationship between LDGs and categorial grammars. The separation of left and right argument lists is actually a feature of the earliest categorial grammars to incorporate a notion of word order (Bar-Hillel 1953). The tidied-up notation (Bar-Hillel 1964, Chapter 14) presents each category as a fraction, with arguments to the left on one side, and to the right on the other. The only difference between Bar-Hillel's Categorial Grammar and Lexicalised Dependency Grammar is Bar-Hillel's use of recursive types, allowing arguments to have their own arguments (i.e. allowing functions of functions). A recognition of this is implicit in Bar-Hillel's comments on dependency grammar (Bar-Hillel 1964, Chapter 14). Later categorial grammars have adopted a Curried notation, with functional categories applying to one argument at a time.[13]

A specification of Lexicalised Dependency Grammar is given in Figure 11. The distinguished type, $\mathbf{T}_0$ is normally the sentence category, **s**.

Due to the functional nature of the categories used by LDGs, it is trivial to associate them with a simple functional semantics. An assumption is made that any function application (formation of a dependency) is matched by a function application in the semantics. This requires each semantic type to have an equal number of arguments, or a greater number of arguments than its associated syntactic type. Some example mappings are given in Figure 12.[14]

[12] A dependency graph is said to respect *adjacency* if each head is grouped with its dependents i.e. no word is separated from its head except by its own dependents, or by other dependents of the same head and their dependents.

[13] In Curried notation, there are three possible categories corresponding to the LDG category for **showed**: **((np\s)/pp)/np**, **np\((s/pp)/np)**, and **(np\(s/pp))/np**. All three can be proven equivalent given a rule of Associativity (Lambek 1958).

[14] Arguments which appear on the left argument list are taken by convention to be the innermost arguments of the semantic function.

1. A finite set of base types, { $\mathbf{T}_0$, ..., $\mathbf{T}_n$ }
   (such as **s**, **np**, and **pp**)

2. An infinite set of lexical categories of the form,[a]

$$\begin{bmatrix} X \\ \mathtt{l}L \\ \mathtt{r}R \end{bmatrix}$$

3. A lexicon, **L**, consisting of a finite set of lexical entries of form,

   Word: Lexical Category

4. A distinguished base type, $\mathbf{T}_0$. A string is grammatical iff it has the category, $\mathbf{T}_0$.

5. A combination rule stating that,

   String$_1$, .. ,**String**$_i$, **"W"**, **String**$_{i+1}$, .. ,**String**$_{i+j}$: **X**

   if **W**: $\begin{bmatrix} X \\ \mathtt{l}\langle T_i,..,T_1\rangle \\ \mathtt{r}\langle T_{i+1},...,T_{i+j}\rangle \end{bmatrix}$ and **String**$_K$: $\mathbf{T}_K$

Fig. 11. LDGs

[a] **X** is a base type, **L** and **R** are lists of base types of arbitrary length.

s	maps to	t	$\begin{bmatrix} s \\ \mathtt{l}\langle np\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix}$	maps to	$\langle$e,t$\rangle$
np	maps to	e			
n	maps to	$\langle$e,t$\rangle$			

Fig. 12

Lexical entries now consist of a syntactic type and a semantic value. For example, the entry for **likes** is:

likes: $\begin{bmatrix} s \\ \mathtt{l}\langle np\rangle \\ \mathtt{r}\langle np\rangle \end{bmatrix}$
λy.(λx.likes'(x,y))

The grammar can be defined simultaneously in its syntactic and semantic components. The conditions (2) and (5) in Figure 11 need replacing by those in Figure 13.

2. An infinite set of lexical categories of the form,[a]

$$\begin{bmatrix} X \\ \mathtt{l}L \\ \mathtt{r}R \end{bmatrix}$$
$$\lambda x_1(..(\lambda x_n.P)..)$$

5. **String**$_1$, .. ,**String**$_i$, **"W"**, **String**$_{i+1}$, .. ,**String**$_{i+j}$: $\begin{matrix} X \\ ((((((S_x\ S_{i+1})..)S_{i+j})S_i)..)S_1) \end{matrix}$

if W: $\begin{bmatrix} X \\ \mathtt{l}\langle T_i,..,T_1\rangle \\ \mathtt{r}\langle T_{i+1},...,T_{i+j}\rangle \end{bmatrix}_{S_x}$ and String_K : $\begin{matrix} T_K \\ S_K \end{matrix}$

Fig. 13. LDGs with Semantics

[a] **X** is a base type, **L** and **R** are lists of base types. The length of **L** added to that of **R** equals **n**.

5. DYNAMIC SPECIFICATION OF PROCESSING OF LDGS

5.1. *Introduction*

To get an idea of how the dynamic model might look, consider associating each state with a semantic value corresponding to the semantics of what has been processed so far. In processing the sentence *Sue thinks John likes Mary* an appropriate semantic value for the state after processing *Sue thinks* is λ**Q.think'(s',Q)**, after processing *Sue thinks John* is λ**P.think'(s',P(j'))**. Thus, appropriate transitions for the fragment remaining after *Sue thinks* are as follows (with obvious abbreviations):

$$\begin{matrix} \lambda Q.tk'(s',Q) & \overset{\text{"John"}}{\rightarrow} & \lambda P.tk'(s',P(j')) & \overset{\text{"likes"}}{\rightarrow} & \lambda Y.tk'(s',lk'(j',Y)) & \overset{\text{"Mary"}}{\rightarrow} & tk'(s',lk'(j',m')) \\ \langle t,t\rangle & & \langle\langle e,t\rangle,t\rangle & & \langle e,t\rangle & & t \end{matrix}$$

In the transitions above, the semantic values have been specified along with their semantic types. Thus, each state is associated with a semantic type. Given the reverse mapping between syntactic categories and semantic types which was given in Figure 12, this suggests the possibility of associating a syntactic type with each state. For example, suitable syntactic categories during processing of the fragment might be as follows:

$$\begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle s\rangle \end{bmatrix} \quad \text{“John”} \xrightarrow{} \quad \begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle \begin{bmatrix} s \\ \mathtt{l}\langle np\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix}\rangle \end{bmatrix} \quad \text{“likes”} \xrightarrow{} \quad \begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle np\rangle \end{bmatrix} \quad \text{“Mary”} \xrightarrow{} \quad s$$

These *state categories* are similar to the lexical categories of LDG defined earlier, except for allowing the possibility of functions as arguments (for example, the state after absorbing **“John”** has a functional argument on its right list i.e. a sentence missing a noun-phrase). State categories are thus notational variants of Bar-Hillel's categories. Each type encodes the combinatory possibilities for the string of words absorbed so far, and can be regarded as an encoding of the present syntactic context. One type corresponds to a possibly infinite set of syntax tree fragments.

5.2. *Specification of Dynamics for LDGs*

This section specifies the dynamics needed to provide transition sequences such as the one given above for *John likes Mary*. The axiom schemata are difficult to read in their most general formulation, so this section should be read in conjunction with the next section, where various instantiations are presented.

The syntactic component of each state is provided by a *state* category. State categories are defined recursively in Figure 14.

1. If **X** is a base type, then $\begin{bmatrix} X \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix}$ is a state category.

2. If **X** is a base type, and **L** and **R** are lists of state categories, then $\begin{bmatrix} X \\ \mathtt{l}L \\ \mathtt{r}R \end{bmatrix}$ is a state category.

Fig. 14. Definition of State Categories

In the example in Section 5.1, the embedded sentence *John likes Mary* was treated as a transition between a category missing a sentence, and one without anything missing. Embedded sentences and non-embedded sentences will be treated alike, so the initial

state category is as follows:[15]

$$\begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle s\rangle \end{bmatrix}$$

An appropriate semantics for the initial state is just the identity function, $\lambda\mathbf{P}.\mathbf{P}$. Processing of a sentence is concluded when nothing more is expected, i.e. the state characterised by the category:

$$\begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix}$$

Where there is unlikely to be confusion this state is abbreviated to just s (a sentence missing nothing is just a sentence). The semantics associated with this state is the semantics of the sentence as a whole.

The crucial part of the formalisation is the axioms which say how to get from one state to another. It turns out that just two axiom schemata are required, and these provide transitions between an infinite number of states. The first schema in Figure 15 is *Application*. Application applies whenever the category of the word matches the category of what is expected (the top item on the right list of the state category), and is similar to both the Application and Generalised Composition rules of categorial grammar (e.g. Ades and Steedman 1982). The axiom will be explained by going through the examples in Section 5.3, but it may also be helpful to consider the depiction of Application in Figure 16 as an operation on 'partial' tree structures which puts together a partial tree of type $\mathbf{Y}$ with a partial tree of type $\mathbf{X}$ to obtain a new partial tree which is missing elements of type $\mathbf{R}_1$ to $\mathbf{R}_j$ and $\mathbf{R'}_1$ to $\mathbf{R'}_h$. $\langle\mathbf{L}_1 \ldots \mathbf{L}_k\rangle$ corresponds to the argument list $\mathbf{L}$ in Figure 15, $\langle\mathbf{R}_1 \ldots \mathbf{R}_j\rangle$ to $\mathbf{R}$, $\langle\mathbf{R'}_1 \ldots \mathbf{R'}_h\rangle$ to $\mathbf{R}'$:

[15] Using unification, it is not actually necessary to start processing with the initial category fully specified. For example, it is possible to start with an expectation of some category, $\mathbf{X}$, and end up with a proof that the string is a sentence.

1. A set of states, $\{ \begin{bmatrix} \mathrm{T} \\ \mathrm{S} \end{bmatrix} \mid \mathrm{T}$ a state category, S a lambda expression $\}$

2. Two axiom schemata,

Application:[a]

$$\begin{bmatrix} Y \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle \begin{bmatrix} X \\ \mathtt{l}L \\ \mathtt{r}\langle\rangle \end{bmatrix} \rangle \bullet R' \end{bmatrix}_{\mathbf{F}} \quad \text{“W”} \quad \begin{bmatrix} Y \\ \mathtt{l}\langle\rangle \\ \mathtt{r}R \bullet R' \end{bmatrix} \quad \text{where} \quad \text{W:} \begin{bmatrix} X \\ \mathtt{l}L \\ \mathtt{r}R \end{bmatrix}_{\mathbf{G}}$$

$$\lambda \mathbf{r}_1.(\lambda \mathbf{r}_2 ... (\lambda \mathbf{r}_j.\ \mathbf{F}(((\mathbf{G}\ \mathbf{r}_1)\mathbf{r}_2)...\mathbf{r}_j)))$$

Prediction:[b]

$$\begin{bmatrix} Y \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle \begin{bmatrix} X \\ \mathtt{l}L \bullet L' \\ \mathtt{r}\langle\rangle \end{bmatrix} \rangle \bullet R' \end{bmatrix}_{\mathbf{F}} \quad \text{“W”}$$

$$\begin{bmatrix} Y \\ \mathtt{l}\langle\rangle \\ \mathtt{r}R \bullet \langle \begin{bmatrix} X \\ \mathtt{l}\langle Z\rangle \bullet L' \\ \mathtt{r}\langle\rangle \end{bmatrix} \rangle \bullet R' \end{bmatrix} \quad \text{where} \quad \text{W:} \begin{bmatrix} Z \\ \mathtt{l}L \\ \mathtt{r}R \end{bmatrix}_{\mathbf{G}}$$

$$\lambda \mathbf{r}_1.(\lambda \mathbf{r}_2 ... (\lambda \mathbf{r}_j.(\lambda \mathbf{h}.\ \mathbf{F}(\lambda \mathbf{l}_k ... \mathbf{l}_1(\mathbf{h}((\mathbf{G}\ \mathbf{r}_1 ... \mathbf{r}j)\mathbf{l}_k ... \mathbf{l}_1))))))$$

3. A deduction rule, Sequencing,

$$\frac{\mathrm{S}_i\ \mathrm{String}_a\ \mathrm{S}_j \qquad \mathrm{S}_j\ \mathrm{String}_b\ \mathrm{S}_k}{\mathrm{S}_i\ \mathrm{String}_a \bullet \mathrm{String}_b\ \mathrm{S}_k}$$

4. A set of pairs of states,

$$\{ \ (\ \underset{\lambda \mathrm{Q.Q}}{\begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle s\rangle \end{bmatrix}} \ , \ \underset{\mathrm{P}}{\begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix}} \ ) \ \}$$

Fig. 15. Specification of Dynamics for LDGs

[a] $\mathbf{L}$, $\mathbf{L}'$, $\mathbf{R}$ and $\mathbf{R}'$ are lists of categories. The length of $\mathbf{R}$ is $\mathbf{j}$.

[b] The length of $\mathbf{L}$ is $\mathbf{k}$, of $\mathbf{R}$ is $\mathbf{j}$.

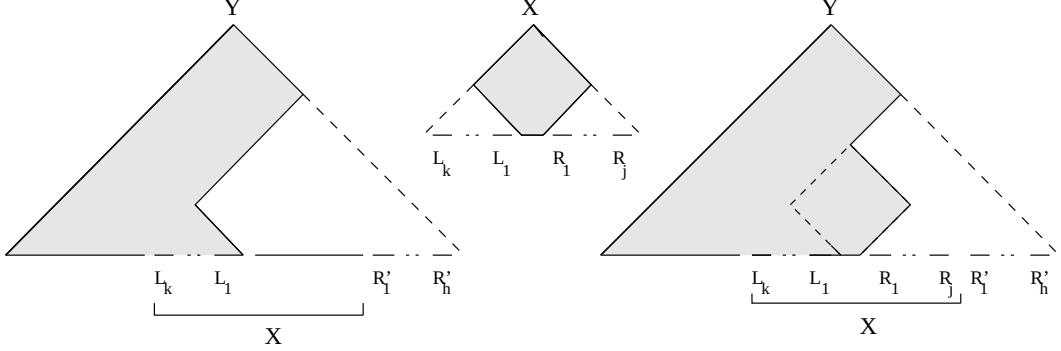


Fig. 16. Application

The second axiom schema is *Prediction*, and this can be applied when one category is expected, but another is found. It is somewhat similar to performing a type raising and then a composition in e.g. Combinatory Categorial Grammar (Steedman 1988). Prediction can be depicted as follows, with $\langle \mathbf{L'}_1 \ldots \mathbf{L'}_i \rangle$ corresponding to $\mathbf{L'}$:

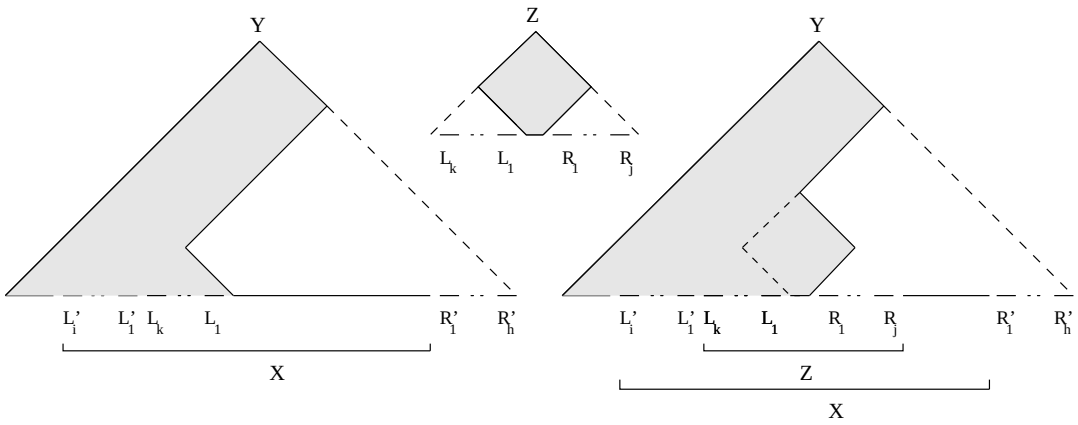


Fig. 17. Prediction

It is worth noting that, given appropriate change of notation, Application and Prediction are both Lambek valid rules of type combination (i.e. the type of the resulting state is a valid combination of the type of the initial state and the type of the word according to the Lambek Calculus (Lambek 1958)). However, unlike in the Lambek Calculus, the resulting type is a state category, not the type of some string. It therefore cannot act as an argument of some other category. This is an important distinction which will be discussed again in Section 10.

Transitions are put together by a Sequencing Rule (similar to that used for the FSM and shift reduce recogniser). Here the Sequencing Rule combines together non-empty strings of words and matches up corresponding final and initial states (comprising a state category and a semantic value).

The dynamics specified here is sound and complete, i.e. there is a proof of the statement,

$$\underset{\lambda Q.Q}{\begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle s\rangle \end{bmatrix}} \quad \text{Str} \quad \underset{P}{\begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix}}$$

if and only if **Str** is a sentence with semantics, **P**. A soundness and completeness proof using structural induction is not particularly difficult, but is too long to include here.

5.3. *Examples*

This section sketches two proofs of acceptability of the sentence, *Sue admitted Ben came yesterday perhaps*. Consider the transition performed by *Sue*. The initial state expects a sentence, and finds a noun phrase. Thus, the only axiom which can apply is Prediction, which is instantiated as follows:

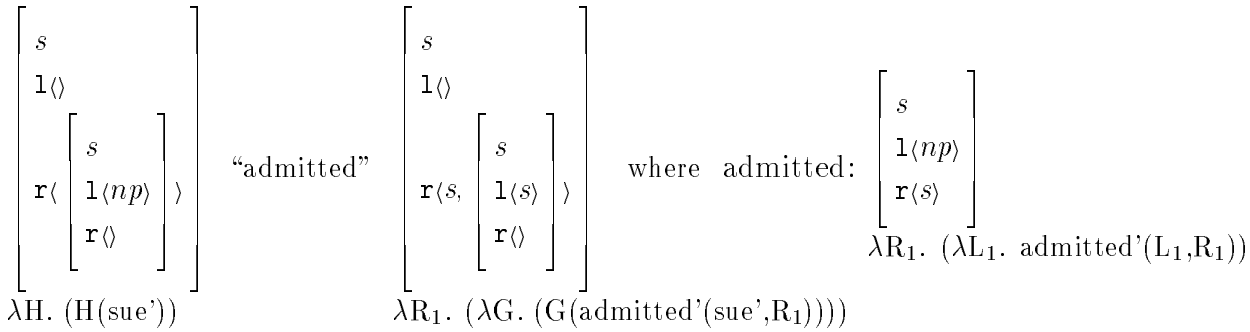
$$\underset{\lambda Q.Q}{\begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle s\rangle \end{bmatrix}} \text{ “Sue” } \underset{\lambda H.\ (\lambda Q.Q)\ (H(\text{sue'}))}{\begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle \begin{bmatrix} s \\ \mathtt{l}\langle np\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix} \end{bmatrix}} \quad \text{where} \quad \text{Sue: } \begin{matrix} np \\ \text{sue'} \end{matrix},$$

The new state expects a sentence which is missing a noun phrase. By Beta-Reduction, the semantics associated with the new state is λ**H.(H(sue'))**.

Now consider the possible transitions for *admitted*. Both Application and Prediction can apply. For example, Application is instantiated as follows (the lambda expressions are in their Beta-Reduced forms):

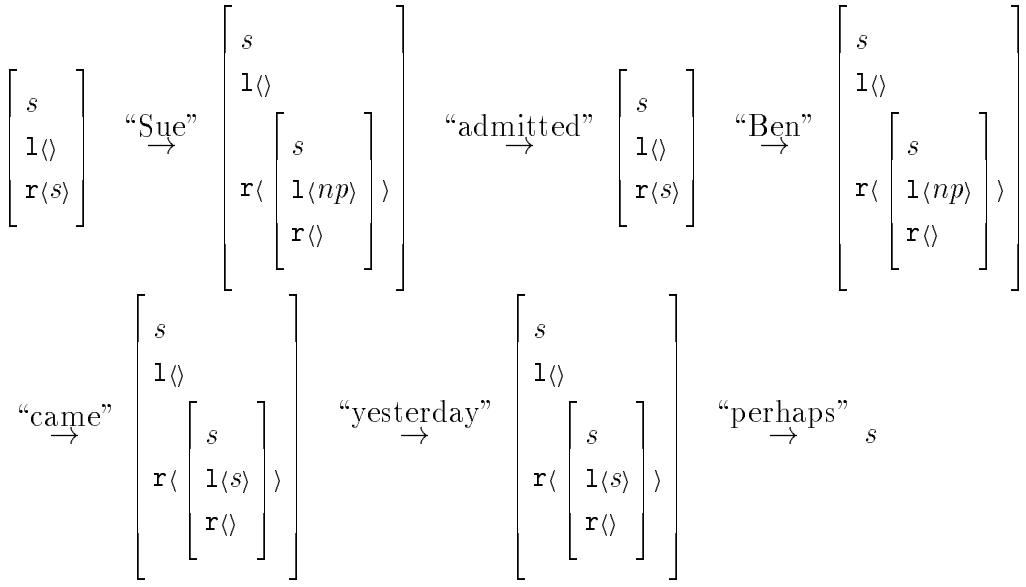
$$\underset{\lambda H.\ (H(\text{sue'}))}{\begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle \begin{bmatrix} s \\ \mathtt{l}\langle np\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix} \rangle \end{bmatrix}} \text{ “admitted” } \underset{\lambda R_1.\ \text{admitted'}(\text{sue'},R_1)}{\begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle s\rangle \end{bmatrix}} \quad \text{where admitted: } \underset{\lambda R_1.\ (\lambda L_1.\ \text{admitted'}(L_1,R_1))}{\begin{bmatrix} s \\ \mathtt{l}\langle np\rangle \\ \mathtt{r}\langle s\rangle \end{bmatrix}}$$

Prediction is also possible, and this corresponds to the possibility that the sentence is embedded, or modified. The instantiation of Prediction is as follows:



The two transitions can be put together using Sequencing to obtain possible transitions for **"Sue admitted"**.

Now consider two possible transition sequences. The first, Example 1, has the following syntactic component:



Example 1 corresponds to the reading where both **yesterday** and **perhaps** modify the embedded sentence **Ben came**[16]. This reading involves both left and right recursion (the two modifiers and the two sentences respectively). This is reflected in the transitions by there being repetitions of the same state category (the first and third, second and fourth, and the fifth and sixth). The modification of **Ben came** by at least one modifier is ensured by the use of Prediction for **came** rather than Application (Prediction can be used whenever Application can, but not vice-versa).

The syntactic component of Example 2 is as follows:

[16] This example chooses to treat adverbs as heads of sentences rather than as dependents of verbs. This choice is standard for categorial grammars, but less common for dependency grammars.

$$\begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle s\rangle \end{bmatrix} \xrightarrow{\text{“Sue”}} \begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle \begin{bmatrix} s \\ \mathtt{l}\langle np\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix}\rangle \end{bmatrix} \xrightarrow{\text{“admitted”}} \begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle s, \begin{bmatrix} s \\ \mathtt{l}\langle s\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix}\rangle \end{bmatrix} \xrightarrow{\text{“Ben”}} \begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle \begin{bmatrix} s \\ \mathtt{l}\langle np\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix}, \begin{bmatrix} s \\ \mathtt{l}\langle s\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix}\rangle \end{bmatrix}$$

$$\xrightarrow{\text{“came”}} \begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle \begin{bmatrix} s \\ \mathtt{l}\langle s\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix}, \begin{bmatrix} s \\ \mathtt{l}\langle s\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix}\rangle \end{bmatrix} \xrightarrow{\text{“yesterday”}} \begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle \begin{bmatrix} s \\ \mathtt{l}\langle s\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix}\rangle \end{bmatrix} \xrightarrow{\text{“perhaps”}} s$$

Example 2 provides the reading where each modifier attaches to a different sentence. This is a case of centre embedding, and is reflected in the transitions by there being duplicate categories within the same state category (there are two sentence modifiers in the fifth category in this case). Prediction is used for both **admitted** and **came**.

6. IMPLEMENTATION

A straightforward implementation of the dynamics is possible by exploring all possible transition sequences. Choice points occur due to lexical ambiguity, or due to Application being available as an alternative to Prediction. Disregarding lexical ambiguity, this gives a maximum branching factor of just two. However, this still means that parsing and recognition take exponential time in the worst case (i.e. in time proportional to $\mathbf{2}^n$ when there are **n** words in the sentence).

To provide polynomial time parsing or recognition, the choices made at one stage must be hidden from subsequent stages until the choice can be resolved. For parsing, this requires ‘packing’ of both state categories and semantic values (the number of readings of an English sentence, for example, is exponentially related to its length in the worst case). This section will concentrate on the simpler problem of packing just the state categories, thereby providing recognition in polynomial time (order $\mathbf{n}^3$).

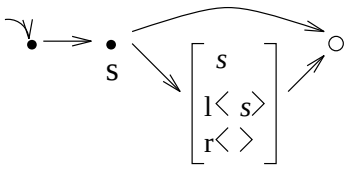
The first step is a straightforward adaptation of the idea of structuring stacks as acyclic directed graphs (Tomita 1985). Reconsider the two transition sequences in Section 5.3. The two state categories after **admitted** are as follows:

$$\begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle s\rangle \end{bmatrix} \qquad \begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \\ \mathtt{r}\langle s, \begin{bmatrix} s \\ \mathtt{l}\langle s\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix}\rangle \end{bmatrix}$$

All the state categories during recognition of a sentence share the same base type, **s**, and have an empty left list. Thus, only the right lists need be represented. The two right lists are as follows:

$$\langle s\rangle \qquad \langle s, \begin{bmatrix} s \\ \mathtt{l}\langle s\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix}\rangle$$

These can be merged into the following graph, where the end node is represented by ○:



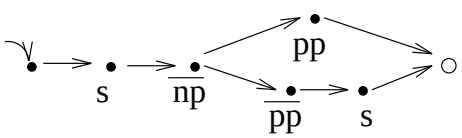
Each list of state categories corresponds to a path within the graph. For example, the top path in the graph corresponds to the list, $\langle s\rangle$.

Prediction and Application can be adapted to work with graphs instead of state categories, and this enables the differences between the two state categories to be hidden until as late as possible in the processing (in this case, until after the modifier **yesterday**).

The second step is to perform further compaction of the graphs by allowing categories to share elements on their left subcategorisation lists. Consider the following pair of right lists:

$$\langle s, \begin{bmatrix} pp \\ \mathtt{l}\langle np\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix}\rangle \qquad \langle s, \begin{bmatrix} s \\ \mathtt{l}\langle np, pp\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix}\rangle$$

The second element in each list can be 'flattened' by distinguishing arguments from base types (by putting lines above arguments), and letting left arguments precede their base type. The two right lists can then be represented by the following graph:



The top path, for example, represents a list consisting of an **s** followed by a category with an **np** on its left subcategorisation list and base category **pp**.

Details of the recognition algorithm are given in the Appendix, along with a proof that recognition is order $\mathbf{n}^3 \cdot \mathbf{G}_a^3$ in the worst case, where $\mathbf{G}_a$ is the maximum number of possible syntactic categories for any one word.

7. DYNAMIC GRAMMARS

This section defines a very general notion of *dynamic grammar*. A more restrictive class of grammars will be defined in Section 11. A dynamic grammar is similar to a dynamic specification. There are two main changes. Firstly, there is explicit statement of what it means for a string to be grammatical. Secondly, more than one deduction rule is allowed, and these may have *side conditions*[17]. The details are given in Figure 18.

1. A set of states[a] (not necessarily finite).
2. A finite set of axiom schemata of the form,

   $$S_a \text{ String } S_b \qquad \text{(Side-Condition)}$$

3. A finite set of deduction rules of the form,

   $$\frac{S_0 \text{ Str}_0 \; S_1 \quad \ldots \; S_{n-1} \text{ Str}_{n-1} \; S_n}{S_a \text{ Str}_a \; S_b} \qquad \text{(Side-Condition)}$$

4. A condition of the form: A string, **Str**, is a sentence if and only if, $\mathbf{S}_i$ **Str** $\mathbf{S}_f$

Fig. 18. Unrestricted Dynamic Grammars

[a] The terminology of *states* is used to keep as close as possible to the dynamic specifications. However, as part of a grammar these can be thought of as merely labels, since there is no constraint which forces them to be representations of the states of some process.

Dynamic Dependency Grammar is specified in Figure 19. Dynamic Dependency Grammars accept exactly the same strings as Lexicalised Dependency Grammars with the same lexicon.

[17] So far the only side conditions have been on axiom schemata, and have been of the form **W: Lexical Category**.

1. A set of state categories.
2. Two axiom schemata, Application and Prediction.
3. A single deduction rule, Sequencing
4. A string , **Str**, is a sentence if and only if,

$$\begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle s\rangle \end{bmatrix} \text{ Str } \begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix}$$

Fig. 19. Dynamic Dependency Grammars

Strong equivalence between Lexicalised Dependency Grammars and Dynamic Dependency Grammars requires a proof of a one-to-one correspondence between transition sequences and parse trees. An alternative is to prove equivalence between an LDG with semantics, and a DDG with semantics i.e. to prove that both accept the same strings and give them the same readings. A DDG with semantics is specified in Figure 20. The proof of equivalence is a trivial adaptation of the proof of soundness and completeness of the dynamic specification in Figure 15.

1. A set of states, comprising a state category paired with a lambda expression.
2. Two axiom schemata, Application and Prediction.
3. A single deduction rule, Sequencing
4. A string , **Str**, is a sentence with semantics **P** iff,

$$\underset{\lambda Q.Q}{\begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle s\rangle \end{bmatrix}} \text{ Str } \underset{\mathrm{P}}{\begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix}}$$

Fig. 20. Dynamic Dependency Grammars with Semantics

The main advantage of DDGs over LDGs as a linguistic formalism is the possibility of embedding DDGs within richer grammars which include extra axioms and deduction rules. Although purely lexicalist grammars such as LDG provide an adequate treatment of 'canonical' combinations between functions and their arguments, there are various syntactic constructions which are not easily lexicalised. Prime candidates for treatment using the mechanisms provided by the dynamic grammars are asymmetrical phenomena, such as extraposition or heavy noun phrase shift (cf. Milward 1991). The next section will also argue that coordination, in particular, non-constituent coordination can be successfully specified using a dynamic grammar.

Finally, it is worth noting that dynamic grammars can be given a more traditional appearance (if required) by rewriting transitions between states as *transition types*. A string, **Str** has transition type $\mathbf{S}_a \rightarrow \mathbf{S}_b$ if it maps between $\mathbf{S}_a$ and $\mathbf{S}_b$. The definition of a dynamic grammar using transition types is provided by Figure 21.

1. A finite set of axiom schemata of the form,

   $\text{String} : \text{S}_a \rightarrow \text{S}_b$ (Side-Condition)

2. A finite set of deduction rules of the form,

   $$\frac{\text{Str}_0\text{: S}_0 \rightarrow \text{S}_1 \quad \ldots \quad \text{Str}_{n-1}\text{: S}_{n-1} \rightarrow \text{S}_n}{\text{Str}_a\text{: S}_a \rightarrow \text{S}_b} \qquad \text{(Side-Condition)}$$

3. A condition of the form: A string, **Str**, is a sentence if and only if, **Str:** $\mathbf{S}_i \rightarrow \mathbf{S}_f$

Fig. 21. Unrestricted Dynamic Grammars with Transition Types

8. ENGLISH NON-CONSTITUENT COORDINATION

Coordination is generally regarded as problematic due to the possibility of conjuncts being non-constituents (according to 'standard' grammars), and having types which do not match (even if the conjuncts are constituents). Both possibilities (and combinations of them) are illustrated by the following:

2) a Sue lent [Joe a book], and [Ben a paper]
   b [Smith loaned] and [his widow later donated] a valuable collection of manuscripts to the library (Abbott 1976)
   c Finding no [cars inside] or [lorries outside] the warehouse, the police gave up the search
   d You can call me [directly], or [between 2pm and 5pm through my secretary]
   e Sue put [a lamp on the table], and [on the ledge a large antique punchbowl]
   f Pat is [a Republican] and [proud of it] (Sag et al. 1985)

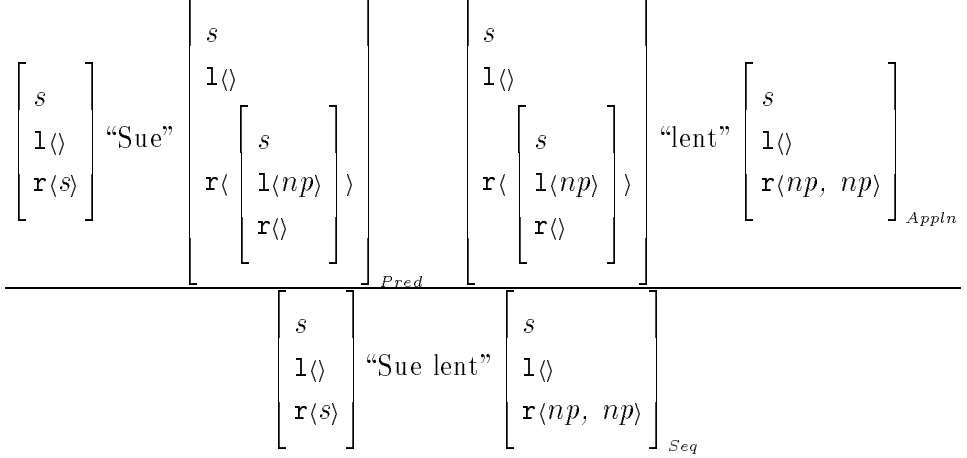
This section briefly sketches an analysis of the syntax of coordination using deduction rules which has been developed by Milward (1991, 1994). The simplest form of Coordination Rule is given in Figure 22.

$$\frac{\text{S}_0\ \text{String}_i\ \text{S}_1 \quad \text{S}_0\ \text{String}_j\ \text{S}_1}{\text{S}_0\ \text{String}_i \bullet \text{“W”} \bullet \text{String}_j\ \text{S}_1} \qquad \text{where} \quad \text{W} \in \{\text{and, or}\}^a$$

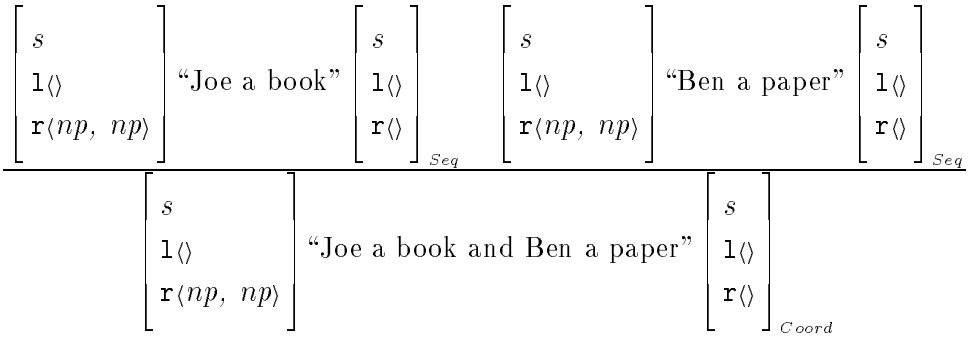
Fig. 22. Coordination Rule

[a] $\mathbf{String}_i$ and $\mathbf{String}_j$ are non-empty strings of words.

This allows strings to coordinate provided they perform the same transitions between state categories. As an example, consider a proof of the acceptability of (2a) in Dynamic Dependency Grammar plus Coordination (DDGC). A subproof for the shared substring **"Sue lent"** is as follows:



The Coordination Rule can be instantiated as follows:



The transitions for **"Joe a book"** and **"Ben a paper"** can be proven by two uses of Application, followed by Sequencing. The proof for the sentence as a whole is obtained by a use of Sequencing to put together **"Sue lent"** and **"Joe a book and Ben a paper"**.

The two conjuncts, **Joe a book** and **Ben a paper** are *parallel* conjuncts (the lexical categories match one to one: **Joe** has the same category as **Ben**, **book** the same as **paper**). Since parallel conjuncts can always perform the same transitions, the coordination rule always allows coordination between them. Moreover, since any substring of a simple sentence (not including a conjunction) can perform a transition (i.e. has a transition type), the account predicts that any substring of a sentence can coordinate with a parallel substring.

The matching up of state categories ensures that each conjunct makes similar connections to the surrounding material. Consider the following examples:[18]

3) a * Sue saw$_i$ the man$_j$ [through the telescope]$_i$ and [with the troublesome kid]$_j$

[18]The indexing is used to indicate different bracketings. In (b), the coindexing with **Mary** corresponds to the bracketing **[a friend of Mary]'s handbag**, and the coindexing with **handbag** corresponds to the bracketing **the manufacturer of [Mary's handbag]**. Example (b) is a variant of an example of Paul Dekker's (Barry and Pickering 1993).

b * I saw [a friend of]$_i$ and [the manufacturer of]$_j$ Mary$_i$'s handbag$_j$

These examples are problematic for accounts which delete under morphological identity (e.g. van Oirsouw 1987). They are also problematic for Lambek Calculus accounts (e.g. Moortgat 1988), which make similar predictions to deletion under identity of morphology plus lexical category (each conjunct can be assigned the same type if there are the same lexical categories for the shared parts of the sentence). Example (3a) is ruled out by DDGC, since the decisions as to whether there is a noun modifier, or a sentence modifier are taken within the shared fragment **"Sue saw the man"** (Prediction must be used for **saw** in one case, and for **man** in the other). The state categories have therefore diverged before the start of the conjunction. Example (b) is similarly ruled out since the choice as to whether or not **Mary** forms part of a noun phrase **Mary's handbag** again occurs within a shared fragment.

Further restrictions on parallel coordination can be imposed by levels of linguistic structure other than syntax. In particular, prosody seems to have a role, at least in determining preferences if not actual grammaticality (note that, to avoid circularity in the account, the prosodic structure must not itself be determined entirely by syntactic structure). To illustrate how prosody interacts with coordination, consider the following examples:

4) a John drove a car to, and Peter drove a van to Amsterdam
   b John drove a car to, and Peter drove a car from Amsterdam
   c John drove a car to, and Peter drove a car from a small suburb of Amsterdam

The examples have identical similar structure, but vary in acceptability. Example (a) is unacceptable according to some speakers[19]. Example (b) is much improved, with the prepositions gaining weight from contrastive stress. Example (c) is further improved by the use of a longer final noun phrase.

Now reconsider the examples in (2). There are two cases of parallel coordination (a) and (c), which are both predicted to be grammatical by the DDGC. There are also two cases of non-parallel coordination, (b) and (d), which are predicted to be grammatical.

However, the DDGC fails to predict the acceptability of examples (e) and (f). One explanation for the failure to accept (e) is that the grammar needs augmenting with a rule of heavy noun phrase shift (cf. Milward 1991). An alternative explanation which applies to both (e) and (f), is that failure is due to choice points within the shared

[19]The example is similar to a 'starred' example in Kempen (1992). Kempen argues for the use of prosody in explaining both coordination and self-repair.

material. In (e), there is a choice of two alternative orders for the arguments of **put**. In (f) there is a choice of two different categories for the argument.

For (e) the choice of word order can be delayed by using lexical entries with 'under-specified' word order (e.g. by using linear precedence constraints). For (f), the choice of argument category can be delayed by using feature bundles (cf. Sag et al. 1985), or disjunctive types (cf. Morrill 1990). Alternatively, choices can be delayed using a more general mechanism for packing state categories (such as the graph structuring discussed in Section 6).[20]

Finally, note that, using the alternative notation of transition categories, the coordination rule can be rewritten as follows:

$$\frac{\mathrm{Str}_a\colon \mathrm{X} \qquad \mathrm{Str}_b\colon \mathrm{X}}{\mathrm{Str}_a \bullet \text{“W”} \mathrm{Str}_b\colon \mathrm{X}} \qquad \text{where X } \epsilon \text{ \{and,or\}}$$

This is just the phrasal coordination rule:

$$\mathrm{X} \rightarrow \mathrm{X}\ \mathrm{Conj}\ \mathrm{X}$$

Thus phrasal coordination has been regained, but at the level of transition types.

9. IMPLEMENTATION OF COORDINATION

This section sketches an implementation of a dynamic grammar including coordination, and compares this to the treatment of coordination in ATNs (Woods 1973). Reconsider example (2a):

Sue lent [Joe a book] and [Ben a paper]

A possible transition sequence for **“Sue lent Joe a book”** is the following:

$$\begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle s\rangle \end{bmatrix} \overset{\text{“Sue”}}{\rightarrow} \begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle \begin{bmatrix} s \\ \mathtt{l}\langle np\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix}\rangle \end{bmatrix} \overset{\text{“lent”}}{\rightarrow} \begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle np,\ np\rangle \end{bmatrix} \overset{\text{“Joe”}}{\rightarrow} \begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle np\rangle \end{bmatrix} \overset{\text{“a”}}{\rightarrow} \begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle n\rangle \end{bmatrix} \overset{\text{“book”}}{\rightarrow} s$$

[20] Use of the exact mechanism suggested in Section 6 would not be appropriate, since it would allow (3a) and disallow (2b).

The transition sequence can be constructed incrementally, one word at a time. On encountering the conjunction, **and**, the parser looks back down the transition history, and non-deterministically chooses one of the earlier state categories, which it then uses to start processing **"Ben a paper"**. Processing of the second conjunct can be concluded if there is a state which matches the state before the conjunction. In this case, the state after processing **"paper"** is the matching state, **s**. A new transition history is now created, including just the initial and final states of the conjunction. The final part of the sequence is as follows:

$$\text{"lent"} \xrightarrow{} \left[\begin{array}{l} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle np,\ np\rangle \end{array}\right] \text{"Joe a book and Ben a paper"} \xrightarrow{} s$$

The procedure deals with iterated and nested coordination, and correctly disallows interleaved coordination e.g.

5) * [John saw each boy and each] or [Fred saw each] girl

A detailed description, and a correctness proof is given in Milward (1991). There is also a semantics provided, however this can be criticised for not being fully incremental: the semantics for a second conjunct is built up prior to integration with the semantics for the rest of the sentence.

The implementation of coordination has much in common with the SYSCONJ system of Woods (1973), which was developed in order to provide a treatment of coordination within Augmented Transition Networks (ATNs). In encountering a conjunction, the SYSCONJ system similarly chooses some earlier state (to be precise, an earlier configuration of the parsing stack) which it uses for the second conjunct. There are however some essential differences. Firstly, on completing a conjunct, SYSCONJ does not immediately merge the two stack configurations, but, instead, separately parses both conjuncts in parallel until a constituent is completed. For example, on parsing the sentence,

6) John gave Mary a book and Peter a paper about subjacency

the SYSCONJ system separately parses *Peter a paper about subjacency* and *Mary a book about subjacency* before conjoining at the level of some enclosing constituent (for example the verb phase). The result is therefore similar to starting with the sentence:

7) John gave Mary a book about subjacency and gave Peter a paper about subjacency

There are two problems with such an approach. The first is similar to that encountered by Conjunction Reduction analyses of coordination, and pointed out by Gazdar (1981). If parts of sentences which are shared by both conjunct (such as the *about subjacency*) are parsed separately, then there are problems in interpretation. For example

8) John gave Mary a book and Peter a paper by the same author

is fine, but

9) John gave Mary a book by the same author and Peter a paper by the same author

is nonsensical. The second problem is due to the possibility of nested coordination. Consider the sentence:

10) John wanted to study medicine when he was eleven, law when he was twelve, and to study nothing at all when he was eighteen

The smallest constituent containing *to study medicine when he was eleven* is the verb phase *wanted to study medicine when he was eleven*. However, if coordination of the first two conjuncts occurs at this level, it is difficult to see how to deal with the final conjunct.

A second difference between the SYSCONJ treatment and a dynamic treatment concerns the use in SYSCONJ of stack based configurations rather than full parsing histories. This means that once something is popped off the stack its internal structure cannot be accessed by the coordination routine, ruling out examples such as the following,

11) John gave some books to Mary and papers to George

in which the **np**, *some books*, is completed prior to the conjunction being reached.

Finally, it is worth noting that both SYSCONJ and the dynamic treatment of coordination hypothesise a dependence of coordination facts upon characterisations of states (state configurations in ATNs, state categories in dynamic grammars). If dynamic grammars are assumed to be directly implemented (without further packing of state categories), then both theories hypothesise a dependence of coordination on actual non-determinism within processing. This suggests that any adequate model of the syntax of coordination must be based on a psycholinguistically plausible processing model. To illustrate this consider a well known example of garden pathing:

12) The horse raced past the barn fell

The choice between the use of **raced** as the main verb, or as part of the reduced relative is usually assumed to be within the fragment **"the horse raced"**. Thus there is a correct prediction of the unacceptability of the following:

13) * The horse raced [past the barn fell] and [beside the hedge]

10. RELATIONSHIP OF DYNAMIC GRAMMARS TO CATEGORIAL GRAMMARS

Categorial grammars have often been claimed to be incremental, due to the possibility of assigning types to initial fragments of sentences. For example, the Lambek Calculus allows every initial fragment of a sentence to be assigned a type (in fact, it assigns an infinite number of types (see e.g. Moortgat 1988)). Combinatory Categorial Grammar (Steedman 1988) allows some, but not all initial fragments to be assigned a type (a fragment such as *John thinks many* would not normally get a type, since type raising applies to full noun phrases rather than to determiners.

Dynamic grammars are distinguished from categorial grammars in their assignment of types (and semantics) to states as opposed to fragments. But is there a real difference? Consider the two axiom schemata in DDG. These are both of the following form:

$$S_0 \quad W \quad S_1 \qquad \text{where} \quad W : C$$

Now consider adopting categories (such as those in Bar-Hillel's Categorial Grammar, Bar-Hillel 1964) which include both the state categories and the lexical categories. The axioms could then be rewritten in the following form, with a restriction that **C** must be the type of a lexical entry:

$$S_1 \quad \rightarrow \quad S_0 \quad C$$

S_1 can then be considered to be the type of an initial substring, as opposed to the state which the substring brings us to. What is obtained is a notational variant of a subset of the Lambek Calculus, since both axioms are Lambek valid (given an appropriate translation of the categories). However, the subset is an unnatural one on several counts. Firstly, to obtain the correct type for a sentence would require the sentence to be fronted with a dummy word with category **s/s**. Secondly, the rewrite rules would have the restriction that the righthand type must be from a lexical entry. Thirdly, the combination rules are not symmetrical (mirror-image rules could be added, but this would result in non-trivial spurious ambiguity).

The unnaturalness of the subset is perhaps to be expected, since the approach to dealing with incremental interpretation and coordination exemplified by DDG/DDGC is fundamentally different from that exemplified by CCG or the Lambek Calculus. DDGs accept exactly the same strings as the corresponding LDGs, and give them the same analyses. Thus the ability to provide incremental interpretation in no way affects the original expressiveness of the grammar. Similarly DDGC adds a treatment of coordination without affecting the analysis of strings which do not have a conjunction.

In contrast, in Steedman and Moortgat's work (e.g. Steedman 1988, Moortgat 1988), incrementality and coordination have been used to justify the introduction of powerful combination rules into the grammar itself. However, there is some evidence that the addition of rules which are powerful enough to deal with coordination will lead to incorrect predictions elsewhere. Consider the following sentences:

14) a The [young inexperienced] and [elderly] applicants have the most difficulty getting a job
b # Some very [young inexperienced] candidates are required

CCG and the Lambek Calculus compose the types for **young** and **inexperienced** to get the type **n/n**, which is also the type for **elderly**. **n/n** is an appropriate type for the argument of the modifier **very**, yet the only reading for example (b) is where **very** modifies **young** as opposed to **young inexperienced**. Since **very** can modify both **inexperienced** and **young**, the incorrect reading cannot easily be excluded by the addition of extra features to the types.

The above example is one illustration of the effect of allowing composition operations within a grammar which includes higher order types. A more vivid example is given by the following pair of sentences:

15) a

Children	who came from far away	reluctantly	arrived
np	np\np	(np\s)/(np\s)	np\s

b

Children	reluctantly	who came from far away	arrived
np	(np\s)/(np\s)	np\np	np\s

This is a case where backwards composition allows the noun phrase modifier to be composed with the verb phrase, thereby predicting that (b) is acceptable.

The problems of using composition rely on the existence of words with higher order types i.e. functions of functions. Since these are not allowed in dependency grammar, it may seem that an unfair comparison is being made. However, it is possible to provide strongly equivalent dynamic grammars for extended LDGs, which include function of

functions (the dynamic grammar requires state categories with one extra argument list, and slightly more complicated versions of Prediction and Application). These extended LDGs are similar to categorial grammars with Forward and Backward Application, and Associativity, but no Composition or Type-Raising.

11. THE SPACE OF DYNAMIC GRAMMARS

In Section 3 various assumptions were added to the general notion of a dynamic specification to provide a set of specifications which seemed plausible as models of sentence processing. Similarly, various assumptions can be added to the general notion of a dynamic grammar to obtain a set of grammars which seem plausible as grammars of natural language. Here two assumptions will be considered which parallel the assumptions made for processing.

The first assumption is that a dynamic grammar must be a *decidable* logic, i.e. it must be possible to prove a statement in finite time. This can be achieved either by individual proofs of decidability for individual grammars, or by imposing conditions on deduction rules and side conditions. The simplest set of conditions (and one satisfied by DDGC) is to limit side conditions to lexical entries, and to restrict deduction rules as in Figure 23.

A finite set of deduction rules of the form,

$$\frac{S_0\ Str_0\ S_1\ \ \ldots\ S_{n-1}\ Str_{n-1}\ S_n}{S_a\ Str_a\ S_b}$$

$\mathbf{Str}_0$, ..., $\mathbf{Str}_{n-1}$ are proper substrings of $\mathbf{Str}_a$

Fig. 23. Restricted Deduction Rules

Given these conditions, proofs must terminate since, after the application of a deduction rule, the new subproofs both involve a substring which has length at least one less than the original. For formal proofs of decidability of more complex grammars see Milward (1991).

The second assumption is that the grammar is *incremental* i.e. it provides a state for each initial substring from which a semantic representation can be extracted. The paper has shown that it is possible to form incremental dynamic grammars based on Lexicalised Dependency Grammar. What other possibilities are there? The possibility of generalising left corner parsing (Thompson et al. 1991) suggests that it is possible to provide an incremental dynamic grammar which is strongly equivalent to a PSG. How-

ever, this result has not been proven, and any implementation of the grammar would be highly non-deterministic (there would be far less packing of tree structure than in DDGs). There have been proofs of dynamic grammar equivalents to a variety of lexicalised grammars where structure is given purely by subcategorisation: for example, for an LDG with free word order (but retaining adjacency), and for an LDG with functions of functions (equivalent to a categorial grammar with Associativity, and Application, but not Composition).

12. CONCLUSION

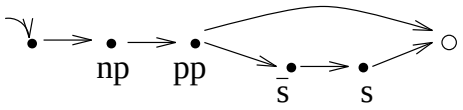
The paper has shown how dynamics provides a useful level of abstraction for the description of sentence processing. A particular processing model has been described which provides fully word by word incremental interpretation, and recognition in time $\mathbf{n}^3$.

The paper has also introduced the notion of a dynamic grammar, and has shown how this can be used for linguistic description, in particular for non-constituent coordination. The approach has been compared with categorial grammars and with ATNs.

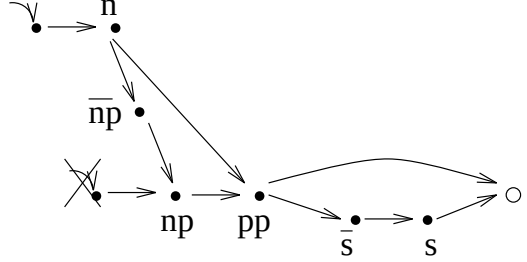
APPENDIX *Recognition in $O(n^3)$*

Background

Section 6 provided motivation for using graphs to pack sets of right lists. This section is concerned with how the graphs are manipulated. A graphical representation is effective if updates to the graph are local, leaving most of the graph intact. Consider the graph appropriate after processing the string **"Mary showed"**:



Now consider processing the determiner **the**, which has category noun phrase missing a noun. The category expected is a noun phrase, so Application would replace this with a noun. Prediction would replace the noun phrase by a noun followed by a noun phrase modifier. Using the graphical representation of right lists, Prediction can share a noun node with Application, and just add a left argument noun phrase. The following graph is created by making a new initial node, and linking it appropriately into the old graph:



Now consider the operations performed by Application and Prediction in more formal terms. In recognition of a sentence, the base type and left lists of the state categories remain constant, so only the right lists need to be represented. Considering just the right lists, Application and Prediction look as follows[21]:

$$\left\langle \begin{bmatrix} X \\ \mathtt{l}\langle L_1 \ \ldots L_k\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix} \right\rangle \bullet R' \quad \text{“W”} \quad \langle R_1 \ \ldots R_j\rangle \bullet R' \qquad \text{where} \quad \text{W:} \begin{bmatrix} X \\ \mathtt{l}\langle L_1 \ \ldots L_k\rangle \\ \mathtt{r}\langle R_1 \ \ldots R_j\rangle \end{bmatrix}$$

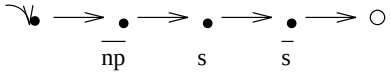
Fig. 24. Application (with only right lists specified)

$$\left\langle \begin{bmatrix} X \\ \mathtt{l}\langle L_1 \ \ldots L_k\rangle \bullet L' \\ \mathtt{r}\langle\rangle \end{bmatrix} \right\rangle \bullet R' \quad \text{“W”} \quad \langle R_1 \ \ldots R_j\rangle \bullet \left\langle \begin{bmatrix} X \\ \mathtt{l}\langle Z\rangle \bullet L' \\ \mathtt{r}\langle\rangle \end{bmatrix} \right\rangle \bullet R'$$

$$\text{where} \quad W\text{:} \begin{bmatrix} Z \\ \mathtt{l}\langle L_1 \ \ldots L_k\rangle \\ \mathtt{r}\langle R_1 \ \ldots R_j\rangle \end{bmatrix}$$

Fig. 25. Prediction (with only right lists specified)

Lexical entries can be represented using acyclic directed graphs[22]. For example, one lexical entry for **admitted** is the graph:



The left arguments precede the base type, which is succeeded by the right arguments. Arguments as opposed to base types are distinguished by having a line above. The Application Axiom can be depicted graphically as follows:

[21] The **L** and **R** lists of Figure 15 have been expanded into $\langle \mathbf{L}_1 \ \ldots \mathbf{L}_k\rangle$ and $\langle \mathbf{R}_1 \ \ldots \mathbf{R}_j\rangle$. **k** and **j** are integers ≥ 0. When **j**,**k**= 0, L,R= $\langle\rangle$.

[22] In this algorithm there is no real purpose, apart from an explanatory one, in using graphs for lexical entries. It is however possible to imagine variants of the algorithm in which all the lexical entries for a given word are compacted into a single graph, giving more efficient storage and processing.

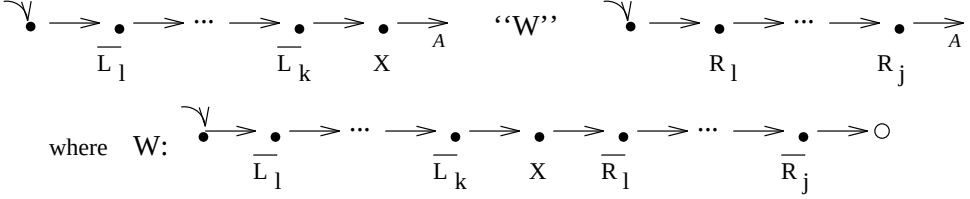


Fig. 26. Application

This should be read as: if there is a path from the initial node to node **A**, with the nodes on the path labelled by $\overline{\mathbf{L}_1} \ldots \overline{\mathbf{L}_k}$ and **X** consecutively, and word **W** is input, then the new graph contains a path from its initial node to node **A**, with the nodes on the path labelled by $\mathbf{R}_1 \ldots \mathbf{R}_j$ consecutively. The preservation of the link to node **A** captures the fact that the final part of the path (labelled by the categories in $\mathbf{R}'$) is unaffected by application of the axiom.

The Prediction Axiom can be depicted similarly, except for the fact that **A** is not allowed to be the end node[23]:

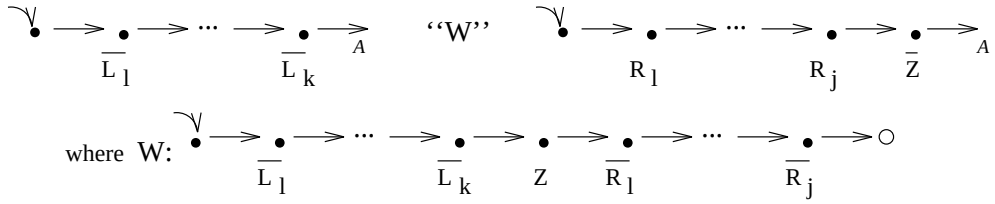


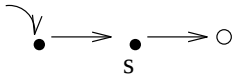
Fig. 27. Prediction

The algorithm described next combines Prediction and Application into a single operation, taking a graph representing a set of state categories and a graph representing a lexical entry, and returning a new graph representing the set of appropriate new state categories.

Algorithm

Initialisation

Form the graph:



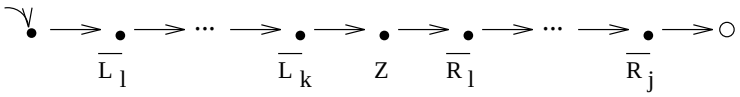
The initial node is labelled **IN**.

[23]The final part of the path is labelled by the categories in $\mathbf{L}'$, by **X** and by $\mathbf{R}'$ consecutively. This labelling is appropriate for any non-empty path in a graph representing right lists.

For each word

Create a new node, **IN′**.

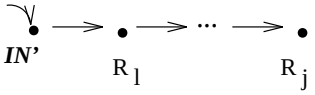
For each lexical entry, represented by a graph of the form:



1. Find the set of non-end nodes, $\mathbf{S}_a$, accessible from the end of a path starting at **IN** and including node labels $\overline{\mathbf{L}_1} \ldots \overline{\mathbf{L}_k}$. If $\mathbf{S}_a$ is empty, fail.

2. Find the set of nodes, $\mathbf{S}_b$, possibly empty, which are accessible from the end of a path starting at a node in $\mathbf{S}_a$, and including node label **Z**.

3. Create a path from **IN′** to a graph labelled by the elements in **R** ($\mathbf{R}_1$ to $\mathbf{R}_j$), i.e.



4. Create a link from the node labelled by $\mathbf{R}_j$ to a new node labelled by $\overline{Z}$. Create links from the node labelled by $\overline{Z}$ to each node in $\mathbf{S}_a$. (This step corresponds to using Prediction).

5. If $\mathbf{S}_b$ is non-empty, create links from the node labelled by $\mathbf{R}_j$ to each node in $\mathbf{S}_b$. (This step corresponds to using Application).

Delete node **IN**. Set **IN** to **IN′**.

Termination

When there are no more words, check that the graph contains an empty path i.e. check that there is a link from the initial node to the end node.

Space Complexity

The run-time space requirement is determined by the size of the graph structured stack. The initial graph has 3 nodes. On the input of a word, each lexical entry adds nodes for its **R** list, and a base category node (the **Z** node above). If the maximum number of lexical entries for any word is $\mathbf{G}_a$, and the maximum right list is $\mathbf{R}_{max}$, this gives a maximum increase of $\mathbf{G}_a \cdot (\mathbf{R}_{max}+\mathbf{1})$. Thus after **m** words, the maximum number of nodes, $\mathbf{K}_m$, is $\mathbf{m} \cdot \mathbf{G}_a \cdot (\mathbf{R}_{max}+\mathbf{1}) + \mathbf{3}$.

Graphs can be represented using ordered adjacency lists, where each node is paired with an ordered list of the nodes it points to (see e.g. Aho et al. 1983). Each node cannot point to more than $\mathbf{K}_m - 1$ other nodes (it cannot point to itself), so the space requirement is bounded by $\mathbf{K}_n^2$ for an $\mathbf{n}$ word sentence. Hence recognition has worst case space complexity of $(\mathbf{n}\cdot\mathbf{G}_a\cdot(\mathbf{R}_{max}+\mathbf{1}) + \mathbf{3})^2$.

Proof of Recognition in $O(n^3)$

Initialisation: constant time.

Step 1: This is the crucial step in the algorithm. A naive depth first search will give complexity related to the length of the maximum left list. After processing $\mathbf{m}$ words, there are a maximum of $\mathbf{K}_m$ nodes to be searched from **IN**, and from each of these there are a maximum of $\mathbf{K}_m$ to be searched. This gives worst case complexity of $\mathbf{K}_m^{L_{max}+1}$ where $\mathbf{L}_{max}$ is the longest left list (an acceptable result for English perhaps, where $\mathbf{L}_{max}$ might be taken to be 1). An algorithm with generally better complexity is achieved by merging together lists of nodes at each stage. From **IN** there are a maximum of $\mathbf{K}_m$ nodes, and from each of these there are also a maximum of $\mathbf{K}_m$ nodes. However, since there are only $\mathbf{K}_m$ nodes altogether, a merge of the $\mathbf{K}_m$ lists of max. $\mathbf{K}_m$ long must result in a new list which itself is only of maximum length $\mathbf{K}_m$. The merging of the $\mathbf{K}_m$ lists of length $\mathbf{K}_m$ is $O(\mathbf{K}_m^2)$[24], so the total complexity is $\mathbf{L}_{max}\cdot\mathbf{K}_m^2$.

Step 2: This requires looking through a maximum of $\mathbf{K}_m$ nodes, and merging the node lists of the nodes labelled by $\mathbf{X}$. The step is therefore proportional to $\mathbf{K}_m^2$ in the worst case.

Step 3: Time proportional to $\mathbf{R}_{max}$

Step 4: Time proportional to $\mathbf{K}_m$

Step 5: Time proportional to $\mathbf{K}_m$

Termination: constant time.

Steps 1 to 5 are repeated for each lexical entry. Given a grammar ambiguity factor of $\mathbf{G}_a$ i.e. the maximum number of syntactic categories for any one word, and assuming

[24]Merging of the first two ordered lists of length $\mathbf{K}_m$ takes time proportional to $\mathbf{2K}_m$ (see e.g. Aho, Hopcroft and Ullman 1983). The result has length bounded by $\mathbf{K}_m$, so can be merged with the third list, also in time proportional to $\mathbf{2K}_m$. Thus total time is bounded by $\mathbf{2K}_m\cdot\mathbf{K}_{m-1}$.

lexical lookup in constant time, then, for each word, time for steps 1 to 5 is proportional to $\mathbf{G}_a \cdot (\mathbf{L}_{max}+\mathbf{1}) \cdot \mathbf{K}_m^2$, where $\mathbf{K}_m = \mathbf{n} \cdot \mathbf{G}_a \cdot (\mathbf{R}_{max}+\mathbf{1}) + \mathbf{3}$.

For an $\mathbf{n}$ word sentence, $\mathbf{m}$ ranges from $\mathbf{1}$ to $\mathbf{n}$, giving a total time complexity for steps 1 to 5 of $\mathbf{G}_a \cdot (\mathbf{L}_{max}+\mathbf{1}) \cdot (\mathbf{G}_a \cdot (\mathbf{R}_{max}+\mathbf{1}) + \mathbf{3})^2 \cdot (\mathbf{1}^2 + \mathbf{2}^2 + \ldots + \mathbf{n}^2)$. Worst case recognition is therefore of order $\mathbf{n}^3 \cdot \mathbf{G}_a^3$.

Appendix Summary

With respect to sentence length, the worst case space and time requirements, $O(\mathrm{n}^2)$ and $O(\mathrm{n}^3)$ respectively, are identical to those for recognition using Kipps' variant of Tomita's algorithm (Kipps 1991), or Earley's algorithm (Earley 1970). With respect to grammar size, there is a difference. Earley's algorithm has a factor of $|\mathrm{G}|^2$, where $|\mathrm{G}|$ is the size of the grammar. Tomita's algorithm is exponential in grammar size in the worst case (Johnson 1991). For recognition of LDGs using the algorithm described here the factor is G_a^3, where G_a is the maximum number of syntactic categories for any one word.

REFERENCES

Abbott, B.: 1976, 'RNR as a Test for Constituenthood', *Linguistic Inquiry* 7, 639-642.

Ades, A. and Steedman, M.: 1982, 'On the Order of Words', *Linguistics and Philosophy* 4, 517-558.

Aho, A.V., Hopcroft, J.E. and Ullman, J.D.: 1983, *Data Structures and Algorithms*, Addison-Wesley.

Aho, A.V. and Ullman, J.D.: 1972, *The Theory of Parsing, Translation and Compiling, Volume 1:Parsing*, Prentice-Hall Inc, New Jersey.

Arbib, M.A.: 1969, *Theories of Abstract Automata*, Prentice-Hall Inc, New Jersey.

Bar-Hillel, Y.: 1953, 'A Quasi-Arithmetical Notation for Syntactic Description', *Language* 29, 47-58.

Bar-Hillel, Y.: 1964, *Language and Information: Selected Essays on Their Theory and Application*, Addison-Wesley.

Barry, G. and Pickering, M.:1993, 'Dependency Categorial Grammar and Coordination', *Linguistics* 31(5), 855-902, Mouton de Gruyter.

van Benthem, J.: 1991, *Language in Action: Categories, Lambdas and Dynamic Logic*, North-Holland, Amsterdam.

Dowty, D.R., Wall, R.F. and Peters, S.: 1981, *Introduction to Montague Semantics*. D.Reidel, Dordrecht.

Earley, J.: 1970, 'An Efficient Context-free Parsing Algorithm', *ACM Communications* 13(2), 94-102.

Gaifman, H.: 1965, 'Dependency Systems and Phrase Structure Systems' *Information and Control* 8: 304-337.

Gazdar, G.: 1981, 'Unbounded Dependencies and Coordinate Structure', *Linguistic Inquiry* 12, 155-184.

Hays, D.G.: 1964, 'Dependency Theory: A Formalism and Some Observations', *Language* 40, 511-525.

Hausser, R.: 1989, *Computation of Language: An Essay on Syntax, Semantics and Pragmatics in Natural Man-Machine Communication*, Springer-Verlag.

Henderson, J.: 1990, *Structure Unification Grammar: A Unifying Framework for Investigating Natural Language*, Technical Report MS-CIS-90-94, University of Pennsylvania.

Hoare, C.A.R.: 1969, 'An Axiomatic Basis for Computer Programming', *Communications of the ACM* 12, 576-583.

Hudson, R.A.: 1990, *English Word Grammar*, Blackwell, Oxford.

Joshi, A.K.: 1987, 'An Introduction to Tree Adjoining Grammars', in Manaster-Ramer (ed.), *Mathematics of Language*, John Benjamins, Amsterdam.

Johnson, M.: 1991, 'The Computational Complexity of GLR Parsing', in Tomita, M. (ed.), *Generalized LR Parsing*, Kluwer.

Just, M. and Carpenter, P.: 1980, 'A Theory of Reading, from Eye Fixations to Comprehension', *Psychological Review* 87, 329-354.

Kipps, J.R.: 1991, 'GLR Parsing in Time $O(n^3)$', in Tomita, M. (ed.), *Generalized LR Parsing*, Kluwer.

Kempen, G.: 1992, 'The Syntax of Coordination, Conjunction Reduction, and Gapping: A Psycholinguistic Approach', unpublished abstract, University of Leiden.

Lambek, J.: 1958, 'The Mathematics of Sentence Structure', *American Mathematical Monthly* 65, 154-169

Langendoen, D.T.: 1975, 'Finite State Parsing of Phrase Structure Languages and the Status of Readjustment Rules in Grammar', *Linguistic Inquiry* 6, 533-554.

van der Linden, E.: 1993, *A Categorial, Computational Theory of Idioms*, PhD Thesis, Utrecht University.

Marcus, M., Hindle, D., and Fleck, M.: 1983, 'D-Theory: Talking about Talking about Trees', in *Proceedings of the 21st ACL*, Cambridge, Mass. 129-136.

Marslen-Wilson, W.: 1973, 'Linguistic Structure and Speech Shadowing at Very Short Latencies', *Nature* 244, 522-523.

Mel'čuk, I.A.: 1988, *Dependency Syntax: Theory and Practice*, State University of New York Press, Albany.

Milward, D.: 1991, *Axiomatic Grammar, Non-Constituent Coordination, and Incremental Interpretation*, PhD thesis, University of Cambridge.

Milward, D.: 1992, 'Dynamics, Dependency Grammar and Incremental Interpretation', in *Proceedings of COLING 92*, Nantes, vol 4, 1095-1099.

Milward, D.: 1994, 'Non-Constituent Coordination: Theory and Practice', in *Proceedings of COLING 94*, Kyoto, Japan, 935-941.

Milward, D. and Cooper, R.: 1994, 'Incremental Interpretation: Applications, Theory and Relationship to Dynamic Semantics', in *Proceedings of COLING 94*, Kyoto, Japan, 748-754.

Moortgat, M.: 1988, *Categorial Investigations: Logical and Linguistic Aspects of the Lambek Calculus*, Dordrecht: Foris.

Morrill, G.: 1990, 'Grammar and Logical Types', in Barry, G. and Morrill, G. (eds.), *Studies in Categorial Grammar*, Edinburgh Working Papers in Cognitive Science, 5.

van Oirsouw, R.R.: 1987, *The Syntax of Coordination*. Croom-Helm.

Pickering, M.: 1991, *Processing Dependencies*, PhD Thesis, University of Edinburgh.

Pollard, C. and Sag, I.A.: 1993, *Head-Driven Phrase Structure Grammar*, University of Chicago Press and CSLI Publications, Chicago.

Pulman, S.G.: 1985, 'A Parser that Doesn't', in *Proceedings of the 2nd European ACL*, Geneva, 128-135.

Pulman, S.G.: 1986, 'Grammars, Parsers, and Memory Limitations', *Language and Cognitive Processes* 1(3), 197-225.

Sag, I.A., Gazdar, G., Wasow, T., and Weisler, S.: 1985, 'Coordination and How to Distinguish Categories', *Natural Language and Linguistic Theory* 3, 117-171.

Shieber, S.M., and Johnson, M.: 1993, 'Variations on Incremental Interpretation', *Journal of Psycholinguistic Research* 22(2), 287-318.

Stabler, E.P.: 1991, 'Avoid the Pedestrian's Paradox', in Berwick, R.C. et al. (eds.), *Principle-Based Parsing: Computation and Psycholinguistics*, 199-237, Kluwer, Netherlands.

Steedman, M.: 1985, 'Dependency and Coordination in the Grammar of Dutch and English', *Language* 61(3), 523-568.

Steedman, M.: 1988, 'Combinators and Grammars', in Oehrle et al. (eds.), *Categorial Grammars and Natural Language Structures*, 417-442.

Tanenhaus, M.K., Garnsey,S. and Boland, J.: 1990, 'Combinatory Lexical Information and Language Comprehension', in Altmann, G.T.M. (ed.) *Cognitive Models of Speech Processing: Psycholinguistic and Computational Perspectives*, MIT Press, Cambridge, Mass.

Thompson, H., Dixon, M., and Lamping, J.: 1991, 'Compose-Reduce Parsing', in *Proceedings of the 29th ACL*, 87-97.

Tomita, M.: 1985, *Efficient Parsing for Natural Language*, Kluwer Academic Publishers, Boston, Ma.

Woods, W.: 1973, 'An Experimental Parsing System for Transition Network Grammars,' in Rustin, R. (ed.), *Natural Language Processing*, Algorithmics Press, New York.

Centre for Cognitive Science

University of Edinburgh

2 Buccleuch Place, Edinburgh EH8 9LW Scotland