DYNAMICS, DEPENDENCY GRAMMAR AND INCREMENTAL INTERPRETATION*

DAVID MILWARD
Centre for Cognitive Science, University of Edinburgh
2, Buccleuch Place, Edinburgh, EH8 9LW, Scotland
davidm@cogsci.ed.ac.uk

Abstract

The paper describes two equivalent grammatical formalisms. The first is a lexicalised version of dependency grammar, and this can be used to provide tree-structured analyses of sentences (though somewhat flatter than those usually provided by phrase structure grammars). The second is a new formalism, 'Dynamic Dependency Grammar', which uses axioms and deduction rules to provide analyses of sentences in terms of transitions between states.

A reformulation of dependency grammar using state transitions is of interest on several grounds. Firstly, it can be used to show that incremental interpretation is possible without requiring notions of overlapping, or flexible constituency (as in some versions of categorial grammar), and without destroying a transparent link between syntax and semantics. Secondly, the reformulation provides a level of description which can act as an intermediate stage between the original grammar and a parsing algorithm. Thirdly, it is possible to extend the reformulated grammars with further axioms and deduction rules to provide coverage of syntactic constructions such as coordination which are difficult to encode lexically.

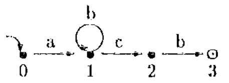
1 Dynamics

Dynamics can roughly be described as the study of systems which consist of a set of states (cognitive, physical etc.) and a family of binary *transition relationships* between states, corresponding to actions which can be performed to change from one state to another (van Benthem, 1990).

This paper introduces a notion of *dynamic grammar*, where each word in a sentence is treated as an action which has the potential to produce a change in state, and each state encodes (in some form) the syntactic or semantic dependencies of the words which have been absorbed so far. There is no requirement for the number of states to be finite. (In fact, since dependency grammar allows centre embedding of arbitrary depth, the corresponding dynamic grammar provides an unlimited number of states).

Dynamic grammars are specified using very simple logics, and a sentence is accepted as grammatical if and only if there is some proof that it performs a transition between some suitable initial and final states. It is worth noting at this early stage that dynamic grammars are not lexicalised rehashes of Augmented Transition Networks (Woods, 1973). ATNs use a finite number of states combined with a recursion mechanism, and act essentially in the same way as a top down parser. They are not particularly suited to incremental interpretation.

To get an idea of how logics (instead of the more usual algebras) can be used to specify dynamic systems in general, it is worth considering a reformulation of the following finite state machine (FSM):



This accepts as grammatical any string which maps from the initial state, 0, to the final state, 3 (i.e. strings of the form: **ab*cb**). The FSM can be reformulated using a logic where the notation,

State0 Str State1

is used to state that the string, **Str**, performs a transition from **State0** to **State1**. The axioms (or atomic proofs) in the logic are provided by the transitions performed by the individual letters. Thus the following are assumed,[1]

0 "a" 1 1 "b" 1 2 "b" 3 1 "c" 2

The transitions given by the single letter strings are put together using a deduction rule, *Sequencing*,[2]

$$\frac{S_0\ \text{String}_a\ S_1 \qquad S_1\ \text{String}_b\ S_2}{S_0\ \text{String}_a\bullet\text{String}_b\ S_2}$$

which states that, provided there is a proof that $\textbf{String}_a$ takes us from some state, $\textbf{S}_0$, to a state $\textbf{S}_1$ and a proof that $\textbf{String}_b$ takes us from $\textbf{S}_1$ to $\textbf{S}_2$, then there is a proof that the concatenation of the strings takes us from S_0 to S_2. The rule puts together strings of letters if the final state reached by the first string matches an initial state for the second string. For example, the rule may be instantiated as:

$$\frac{1\ \text{"bb"}\ 1 \qquad 1\ \text{"c"}\ 2}{1\ \text{"bbc"}\ 2}$$

A string is grammatical according to the logic 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" performs the transitions,

*This research was supported by an SERC research fellowship.

[1] "a" is a string consisting of the single letter, a.

[2] Notation: capital letters will be used to denote variables throughout this paper. '•' will be used to denote concatenation. For example, if String_a = "kl" and String_b = "mn", then $\text{String}_a\bullet\text{String}_b$ = "klmn".

$$0 \xrightarrow{a} 1 \xrightarrow{b} 1 \xrightarrow{b} 1 \xrightarrow{c} 2 \xrightarrow{b} 3$$

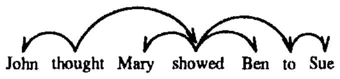
and has the following proof, amongst others:

$$\dfrac{\dfrac{0\text{ ``a'' }1 \qquad \dfrac{\dfrac{1\text{ ``b'' }1 \quad 1\text{ ``b'' }1}{1\text{ ``bb'' }1} \qquad 1\text{ ``c'' }2}{1\text{``bbc'' }2}}{0\text{ ``abbc'' }2} \qquad 2\text{ ``b'' }3}{0\text{ ``abbcb'' }3}$$

Each leaf of the tree is an axiom, and the subproofs are put together using instantiations of the Sequencing Rule.

2 Lexicalised Dependency Grammar

Traditional dependency grammar is not concerned with constituent structure, but with links between individual words. For example, an analysis of the sentence *John thought Mary showed Ben to Sue* might be represented as follows:



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*. A dependency graph is said to respect *adjacency* if no word is separated from its head except by its own dependents, or by another dependent of the same head and its dependents (i.e. there are no crossed links). Adjacency is a reasonably standard restriction, and has been proposed as a universal principle e.g. by Hudson (1988).

Given adjacency, it is possible to extract bracketed strings (and hence a notion of constituent structure) by grouping together each head with its dependents (and the dependents of its dependents). For example, the sentence above gets the bracketing:

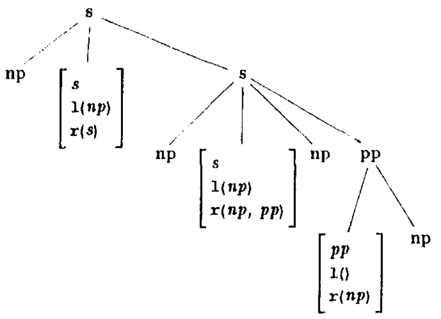
[John thought [Mary showed Ben [to Sue]]]

A noun phrase can be thought of as a noun plus all its dependents, a sentence as a verb plus all its dependents.

In this paper we will assume adjacency, and, for simplicity, that dependents are fixed in their order relative to the head and to each other. Dependency grammars adopting these assumptions were formalised by Gaifman (Hays, 1964). Lexicalisation is relatively trivial given this formalisation, and the work on embedded dependency grammar within categorial grammar (Barry and Pickering, 1990).

Lexicalised Dependency Grammar (LDG) treats each head as a function. For example, the head *thought* is treated as a function with two arguments, a noun phrase and a sentence. Constituents are formed by combining functions with all their arguments. The example above gets the following bracketing and tree structure:

[John thought [Mary showed Ben [to Sue]]]



The tree structure is particularly flat, with all arguments of a function appearing at the same level (this contrasts with standard phrase structure analyses where the subject of *showed* would appear at a different level from its objects).

Lexical categories are feature structures with three main features, a base type (the type of the constituent formed by combining the lexical item with its arguments), a list of arguments which must appear to the left, and a list of arguments which must appear to the right. The arguments at the top of the lists must appear closest to the functor. For example, *showed* has the lexical entry,

$$\text{showed} : \begin{bmatrix} s \\ \mathtt{l}(np) \\ \mathtt{r}(np,\ pp) \end{bmatrix}$$

and can combine with an np on its left and with an np and then a pp on its right to form a sentence.

When left and right argument lists are empty, categories are said to be *saturated*, and may be written as their base type i.e. $\begin{bmatrix} X \\ \mathtt{l}() \\ \mathtt{r}() \end{bmatrix}$ is identical to X.

A requirement inherited from dependency grammar is for arguments to be saturated categories.[3] LDGs will be specified more formally in Section 4.

It is worth outlining the differences between the categories in LDG and those in a directed categorial grammar. Firstly, in LDG there is no specification of whether arguments to the right or to the left of the functor should be combined with first. Thus, the category, $\begin{bmatrix} X \\ \mathtt{l}(Y) \\ \mathtt{r}(Z) \end{bmatrix}$, maps to both $\mathbf{(X\backslash Y)/Z}$ and $\mathbf{(X/Z)\backslash Y}$.[4] Secondly, arguments in LDG must be saturated, so there can be no functions of functions.[5]

[3] In dependency grammar it is not possible to specify that a head requires a dependent with only some, but not all of its dependents.

[4] So called 'Steedman' notation.

[5] An extended form of LDG which allows unsaturated argu-

3 Dynamic Dependency Grammar

Lexicalised Dependency Grammar can be reformulated as the dynamic grammar, Dynamic Dependency Grammar (DDG). Each state in DDG encodes the syntactic context, and is labelled by the type of the string absorbed so far. For example, a possible set of transitions for the string of words "Sue saw Ben" is as follows:

$$\begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle s\rangle \end{bmatrix} \xrightarrow{Sue} \begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle \begin{bmatrix} s \\ \mathtt{l}\langle np\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix} \rangle \end{bmatrix} \xrightarrow{saw} \begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle np\rangle \end{bmatrix} \xrightarrow{Ben} s$$

The state after absorbing "Sue saw" is of type sentence missing a noun phrase, and that after absorbing "Sue saw Ben" is of type sentence.

States are labelled by complex categories which are similar to the lexical categories of LDG, but without the restriction that arguments must be saturated (for example, the state after absorbing "Sue" has an unsaturated argument on its right list). A string of words, **Str**, is grammatical provided the following statement can be proven:

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

The initial state is labelled with an identity type i.e. a sentence missing a sentence. This can be thought of as a context in which a sentence is expected, or as a suitable type for a string of words of length zero. The final state is just of type sentence.

DDG is specified using a logic consisting of a set of axioms and a deduction rule. The logic is similar, but more general, than that used in Axiomatic Grammar (Milward, 1990).[6] The deduction rule is again called Sequencing. The rule is identical in form to the Sequencing Rule used in the reformulation of the FSM, though here it puts together strings of words rather than strings of letters. The rule is as follows,[7]

$$\frac{\mathrm{C}_0\ \mathrm{String}_a\ \mathrm{C}_1,\ \mathrm{C}_1\ \mathrm{String}_b\ \mathrm{C}_2}{\mathrm{C}_0\ \mathrm{String}_a\bullet\mathrm{String}_b\ \mathrm{C}_2}$$

and is restricted to non-empty strings.[8]

ments has been developed, and this also can be reformulated as a dynamic grammar (Milward, 1992).

[6]Axiomatic Grammar is a particular dynamic grammar designed for English, which takes relationships between states as a primary phenomenon, to be justified solely by linguistic data (rather than by an existing formalism such as dependency grammar).

[7]Here '•' concatenates strings of words e.g. "John"•"sleeps" = "John sleeps".

[8]This restriction is not actually necessary as far as the equivalence between LDGs and DDGs is concerned. However its inclusion makes it trivial to show certain formal properties of DDGs, such as termination of proofs.

The set of axioms is infinite since we need to consider transitions between an arbitrary number of categories.[9] The set can be described using just two axiom schemata, Prediction and Application. Prediction is given below, but is best understood by considering various instantiations.[10]

$$\begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle \begin{bmatrix} Y \\ \mathtt{l}L\bullet L' \\ \mathtt{r}\langle\rangle \end{bmatrix} \rangle\bullet R' \end{bmatrix} \text{"W"} \begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}R\bullet\langle \begin{bmatrix} Y \\ \mathtt{l}\langle X\rangle\bullet L' \\ \mathtt{r}\langle\rangle \end{bmatrix} \rangle\bullet R' \end{bmatrix}$$

$$\text{where}\quad \mathrm{W}\!: \begin{bmatrix} X \\ \mathtt{l}L \\ \mathtt{r}R \end{bmatrix}$$

Prediction is usually used when the category of the word encountered does not match the category expected by the current state. Consider the following instantiation:

$$\begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle s\rangle \end{bmatrix} \text{"Sue"} \begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle \begin{bmatrix} s \\ \mathtt{l}\langle np\rangle \\ \mathtt{r}\langle\rangle \end{bmatrix} \rangle \end{bmatrix} \quad \text{where}\quad \mathrm{Sue}\!:\!np$$

The current state expects a sentence and encounters a noun phrase with lexical entry **Sue:np**. The resulting state expects a sentence missing a noun phrase on its left e.g. a verb phrase.

Application gets its name from its similarity to function application (though it actually plays the role of both application and composition). The schema is as follows:

$$\begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle \begin{bmatrix} X \\ \mathtt{l}L \\ \mathtt{r}\langle\rangle \end{bmatrix} \rangle\bullet R' \end{bmatrix} \text{"W"} \begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}R\bullet R' \end{bmatrix} \quad \text{where}\quad \mathrm{W}\!: \begin{bmatrix} X \\ \mathtt{l}L \\ \mathtt{r}R \end{bmatrix}$$

An example instantiation is when a noun phrase is both expected and encountered e.g.

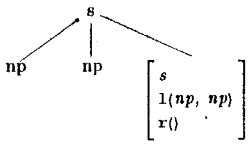
$$\begin{bmatrix} s \\ \mathtt{l}\langle\rangle \\ \mathtt{r}\langle np\rangle \end{bmatrix} \text{"Ben"}\ s \qquad \text{where}\quad \mathrm{Ben}\!:\!np$$

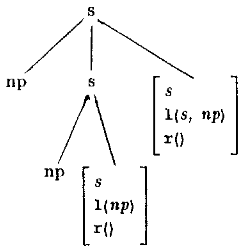
Given a word and a particular current state, the only non-determinism in forming a resulting state is due to lexical ambiguity or from a choice between using Prediction or Application (Prediction is possible whenever Application is). Non-determinism is gen-

[9]An infinite number of distinguishable states is required to deal with centre embedding.

[10]L, L', R and R' are lists of categories. '•' concatenates lists e.g. $\langle k,l\rangle\bullet\langle m,n\rangle = \langle k,l,m,n\rangle$.

erally kept low due to states being labelled by types as opposed to explicit tree structures. This is easiest to illustrate using a verb final language. Consider a pseudo English where the strings, "Ben Sue saw" and "Ben Sue sleeps believes" are acceptable sentences, and have the following LDG analyses:





Despite the differences in the LDG tree structures, the initial fragment of each sentence, "Ben Sue", can be treated identically by the corresponding DDG. The proof of the transition performed by the string "Ben Sue" involves two applications of Prediction put together using the Sequencing Rule. The transitions are as follows:

$$\begin{bmatrix} s \\ \mathrm{l}() \\ \mathrm{r}(s) \end{bmatrix} \xrightarrow{Ben} \begin{bmatrix} s \\ \mathrm{l}() \\ \mathrm{r}(\begin{bmatrix} s \\ \mathrm{l}(np) \\ \mathrm{r}() \end{bmatrix}) \end{bmatrix} \xrightarrow{Sue} \begin{bmatrix} s \\ \mathrm{l}() \\ \mathrm{r}(\begin{bmatrix} s \\ \mathrm{l}(np,\ np) \\ \mathrm{r}() \end{bmatrix}) \end{bmatrix}$$

The transitions for the two sentences diverge when we consider the words *saw* and *sleeps*. In the former case, Application is used, in the latter, Prediction then Application.

Efficient parsing algorithms can be based upon DDGs due to this relative lack of non-determinism in choosing between states.[11] The simplest algorithm is merely to non-deterministically apply Prediction and Application to the initial category. Derivations of algorithms from more complex logics, and the use of normalised proofs and equivalence classes of proofs are described in Milward (1991).

4 LDGs → DDGs

An LDG can be specified more formally as follows:

1. A finite set of base types T_0, .. T_n (such as s, np, and pp)
2. An infinite set of lexical categories of the form, $\begin{bmatrix} X \\ \mathrm{l}L \\ \mathrm{r}R \end{bmatrix}$ where **X** is a base type, and **L** and **R** are lists of base types. When **L** and **R** are empty, a category is identical to its base type, X
3. A finite lexicon, L, which assigns lexical categories to words
4. A distinguished base type, T_0. A string is grammatical iff it has the category, T_0
5. A combination rule stating that,
   if W has category, $\begin{bmatrix} X \\ \mathrm{l}(T_i,..,T_1) \\ \mathrm{r}(T_{i+1},...,T_{i+j}) \end{bmatrix}$
   and String$_1$ has category T_1, String$_2$ has category T_2 etc.
   then the string formed by concatenating
   String$_1$, .. ,String$_i$, "W", String$_{i+1}$, .. ,String$_{i+j}$
   has category X

The corresponding DDG is as follows:

1. A set of categories of the form, $\begin{bmatrix} X \\ \mathrm{l}L \\ \mathrm{r}R \end{bmatrix}$
   where X is a base type, and **L** and **R** are lists of categories
2. Two axiom schemata, Application and Prediction
3. The lexicon, L (as above)
4. One deduction rule, Sequencing
5. A distinguished pair of categories, $\begin{bmatrix} T_0 \\ \mathrm{l}() \\ \mathrm{r}(T_0) \end{bmatrix}$, T_0
   where T_0 is as above. A string, Str, is grammatical iff it is possible to prove:
   $$\begin{bmatrix} T_0 \\ \mathrm{l}() \\ \mathrm{r}(T_0) \end{bmatrix} \quad \mathrm{Str} \quad T_0$$

A proof that any DDG is strongly equivalent to its corresponding LDG is given by Milward (1992). The proof is split into a soundness proof (that a DDG accepts the same strings of words and assigns them corresponding analyses[12]), and a completeness proof (that a DDG accepts whatever strings are accepted by the corresponding LDG).

5 Incremental Interpretation

It is possible to augment each state with a semantic type and a semantic value. Adopting a 'standard' λ-calculus semantics (c.f. Dowty et al, 1981) we obtain the following transitions for the string "Sue saw":

[11]Determinism can also be increased by restricting the axioms according to the properties of a particular lexicon. For example, there is no point predicting categories missing two noun phrases to the left when parsing English.

[12]For this purpose, it is convenient to treat an analysis in a DDG as the transitions performed by each word. Each analysis is a label for an equivalence class of proofs.

$$\begin{array}{lll}
\begin{bmatrix} s \\ \mathrm{l}() \\ \mathrm{r}(s) \end{bmatrix} \xrightarrow{Sue} &
\begin{bmatrix} s \\ \mathrm{l}() \\ \mathrm{r}(\begin{bmatrix} s \\ \mathrm{l}(np) \\ \mathrm{r}() \end{bmatrix}) \end{bmatrix} \xrightarrow{saw} &
\begin{bmatrix} s \\ \mathrm{l}() \\ \mathrm{r}(np) \end{bmatrix} \\
t \rightarrow t & (e \rightarrow t) \rightarrow t & e \rightarrow t \\
\lambda \mathrm{Q.Q} & \lambda \mathrm{P.P(sue')} & \lambda \mathrm{Y.saw'(sue',Y)}
\end{array}$$

The semantic types can generally be extracted from the syntactic types. The base types *s* and *np* map to the semantic types *t* and *e*, standing for truth-value and entity respectively. Categories with arguments map to corresponding functional types.

Provided a close mapping between syntactic and semantic types is assumed, the addition of semantic values to the axiom schemata is relatively trivial, as is the addition of semantic values to the lexicon. For example, the semantic value given to the verb *saw* is **λYλX.saw'(X,Y)**, which has type *e→(e→t)*.

It is worth contrasting the approach taken here with two other approaches to incremental interpretation. The first is that of Pulman (1985). Pulman's approach separates syntactic and semantic analysis, driving semantic combinations off the actions of a parser for a phrase structure grammar. The approach was important in showing that hierarchical syntactic analysis and word by word incremental interpretation are not incompatible. The second approach is that of Ades and Steedman (1982) who incorporate composition rules directly into a categorial grammar. This allows a certain amount of incremental interpretation due to the possibility of forming constituents for some initial substrings. However, the incorporation of composition into the grammar itself does have some unwanted side effects when mixed with a use of functions of functions. For example, if the two types, N/N and N/N are composed to give the type N/N, then this can be modified by an adjectival modifier of type (N/N)/(N/N). Thus, the phrase *the very old green car* can get the bracketing, [the [very [old green]] car]. Although the Application schema used in DDGs does compose functions together, DDGs have identical strong generative capacity to the LDGs they are based upon (the coverage of the grammars is identical, and the analyses are in a one-to-one correspondence).[13]

6 Applications

So far, Dynamic Dependency Grammars can be seen solely as a way to provide incremental parsing and interpretation for Lexicalised Dependency Grammars. As such, they are not of particular linguistic significance. However, it is possible to use DDGs as subsets of more expressive dynamic grammars, where extra axioms and deduction rules are used to provide coverage of syntactic phenomena which are difficult to encode lexically (e.g. coordination, topicalisation and extraposition). For example, the following deduction rule (again restricted to non-empty strings),

$$\frac{\mathrm{C_0\ String_a\ C_1,\ C_0\ String_b\ C_1}}{\mathrm{C_0\ String_a \bullet \text{“and”} \bullet String_b\ C_1}}$$

provides an account of the syntax of non-constituent coordination (Milward, 1991). The sentences ***John gave Mary a book and Peter a paper*** and ***Sue sold and Peter thinks Ben bought a painting*** are accepted since "Mary a book" and "Peter a paper" perform the same transitions between syntactic states, as do "Sue sold" and "Peter thinks Ben bought"

The grammars described in this paper have been implemented in Prolog. A dynamic grammar based upon the extended version of LDGs is being developed to provide incremental interpretation for the natural language interface to a graphics package.

References

Ades, A. and Steedman, M. (1972) On the Order of Words. *Linguistics and Philosophy* 4, 517-558.

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

Barry, G. and Pickering, M. (1990) Dependency and Constituency in Categorial Grammar. In Barry, G. and Morrill, G. (eds), *Studies in Categorial Grammar.* Centre for Cognitive Science, University of Edinburgh.

van Benthem, J. (1990) General Dynamics. ITLI report, Amsterdam (to appear in *Theoretical Linguistics*).

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

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

Hudson, R. (1988) Coordination and Grammatical Relations. *Journal of Linguistics* 24, 303-342.

Milward, D. (1990) Coordination in an Axiomatic Grammar. In Coling-90, Helsinki, vol 3, 207-212.

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

Milward, D. (1992) Dynamic Grammars. Technical Report, Centre for Cognitive Science, University of Edinburgh. In preparation.

Pulman, S. (1985) A Parser That Doesn't. In 2nd European ACL, Geneva.

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

[13]This is also true for dynamic reformulations of extended versions of LDG which allow functions of functions.