

Sentence Processing as Dependency Formation
Martin Pickering
Department of Psychology, University of Glasgow


This paper describes a model of incremental interpretation which is based on the assumption that the fundamental operation of sentence processing is to establish dependencies between words and to give interpretations to the resulting combinations of words. A set of words which are linked by such dependencies is known as a dependency constituent; the grammar assigns interpretations to all and only dependency constituents. The processor constructs dependency constituents in an incremental manner, attempting to fit each new word into the current dependency constituent in order to give the whole fragment a unified interpretation. If it is unable to do this, it makes use of a stack mechanism and holds more than one dependency constituent in memory at the same time. The interpretations of dependency constituents are then outputted by the language processor, and are then integrated with general knowledge. In this model, dependency constituents are the fundamental units of language comprehension. The consequence is that processing is highly incremental, but there are some limits on incrementality. In general, the model is serial, but makes some use of underspecification and incorporates a delay component. I shall conclude by explaining how this model captures a range of psycholinguistic data.

Dependency, Constituency and Dependency Categorical Grammar

This section outlines the grammatical theory of dependency categorical grammar, upon which the account of incremental processing is based. It is more fully described in Pickering and Barry (to appear; see also Pickering (1993) and Barry and Pickering (1990)). First, I describe the notion of dependencies between words employed in this theory. I then derive the notion of a dependency constituent. Finally, I sketch dependency CG, and show how it is able to generate dependency constituents and assign them syntactic types and semantic interpretations.

Dependency. In dependency grammar, the primitive syntactic links hold between words rather than phrases. These links are called *dependencies*, and connect the *ruler* (otherwise known as the head or controller) with the *dependent* (or controlled): for each dependency, an arrow is drawn leading from the ruler to the dependent. Mathematically, the analysis for a sentence is a directed graph with the words at the nodes and the dependencies as edges. For example, *John spoke to Mary* can be given the analysis below:

(1) John spoke to Mary



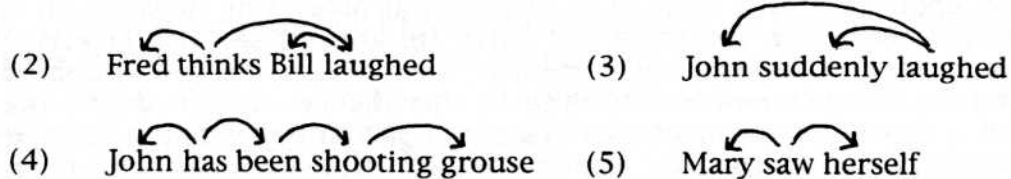
This carries the information that *spoke* has the two dependents, *John* and *to*, and that *to* has the dependent *Mary*. We also say that one word *X* is subordinate to another word *Y* if *X* is either a dependent of *Y* or a dependent of another word which is itself subordinate to *Y*. This means that *John*, *to* and *Mary* are all subordinate to *spoke*. This representation (including word order information) is known as a *dependency diagram*.

The idea of basing grammatical description on the relations between words has a long history, found for instance in the tradition of analysis in terms of government and agreement. Dependency grammars (DGs) of the form considered here can be traced back to Tesnière (1959), with generative grammars being proposed by Gaifman (1965). More recent references include Matthews (1981:78ff) and Hudson (1990). In this paper,

we are concerned with a particular form of DG known as *classical DG* (as described by Gaifman). Classical DG is defined by the following properties. First, it must be possible to trace a path between any two words in the sentence, ignoring the directions on the arrows (this is known as *connectedness*). Second, there must be exactly one word (known as the *root of the sentence*) which has no ruler; in (1), this word is *spoke*. Finally, every other word must have exactly one ruler (the *single-ruler requirement*). Mathematically, classical DG defines directed trees together with a representation of linear order. In addition, every structure in classical dependency grammar obeys adjacency, which is the requirement that no word may intervene between the two words in a dependency unless it is subordinate to one of them.

Pickering and Barry (to appear) discusses how to determine the dependency diagram for a given reading of a sentence, and we shall only provide some brief comments here. However, it should be stressed that there is no infallible way of determining this diagram; in this respect, DG fares no better or worse than phrase structure grammar. The first assumption is that the tensed verb in the main clause is the root of the sentence. We then assume that rulers select the (sub)categories of dependent words. In (1), *spoke* is the root of the sentence. It requires a noun and a preposition, so its dependents are *John* and *to*. In addition, *to* requires a noun, so its dependent is *Mary*. The criterion of category selection is supported by the use of government and agreement. For example, *spoke* governs the case of its subject, so that *John* could be replaced with *he* but not with *him*. This suggests that *John* is a dependent of *spoke*, not the other way around.

Let us give a few more dependency diagrams, taken from Pickering and Barry (ms.), by way of illustration:



In (2), *laughed* is the root of the subordinate clause *Bill laughed* (it has no ruler within this clause), and is a dependent of the main verb *thinks*. If the sentence included the complementizer *that*, then *laughed* would be the dependent of *that* and *that* would be a dependent of *thinks*. In (3), we can see that the modifier *suddenly* is treated as dependent on the verb *laughed*. Notice that the root in (4) is the tensed verb *has*, rather than the 'main' verb *shooting*. This makes the point that our criteria for assuming dependencies are essentially syntactic rather than semantic. Finally, (5) shows that not all relations between words are realised as dependencies; the constraints on the form of *herself* can be captured by a part of a fully-specified grammar concerned with binding.

Dependency Constituency. We can derive different accounts of constituency from dependency diagrams. Traditionally, the only theory to have been considered is that of full constituency: a *full constituent* is an expression consisting of any word *W* and every word subordinate to *W*. The full constituents in (1) are *John*, *Mary*, *to Mary* and *John spoke to Mary* (but not *spoke* or *to*).

In contrast, we define a new theory of constituency called dependency constituency. A *dependency constituent* is an expression consisting of words which are all linked by dependencies. In other words, a dependency constituent must be connected. Clearly, all individual words are dependency constituents, as is the whole sentence. In (1), the other dependency constituents are *John spoke*, *spoke to*, *to Mary*, *John spoke to* and *spoke to Mary*. In (2), they are *Fred thinks*, *Bill laughed*, *thinks Bill*

laughed, thinks ... laughed and *Fred thinks ... laughed* (the last two are discontinuous). However, *thinks Bill* and *Fred thinks Bill* are not dependency constituents.

The most important property of dependency constituency is that it is *flexible*, by which we mean that constituents may overlap. For example, *John spoke* and *spoke to* overlap in (1). This is of course not possible in phrase structure grammar, which embodies a *rigid* theory of constituency (not permitting partial overlap). Pickering and Barry (to appear) argue that flexible constituency in general, and dependency constituency in particular, has considerable linguistic advantages over rigid theories of constituency such as versions of context-free phrase structure grammar. They suggest that dependency constituency can capture the considerable freedom that exists in deciding what expressions can serve as conjuncts in coordinations. In this paper, the flexibility inherent in dependency constituency is fundamental to the analysis of incremental interpretation.

Dependency Categorical Grammar. We now outline a means of providing syntactic types (or categories) and semantic interpretations for dependency constituents, using the framework of Dependency Categorical Grammar. In CG, linguistic expressions are classified according to a system of recursively defined types, which are built from a set of primitive types and a set of connectives. The primitive types are phrasal rather than lexical, and normally include such types as S, S', NP and PP, with standard interpretations, as well as N (corresponding to N' in X' syntax). Other expressions receive complex types defined in a "lexicalist" manner, so that the combinatory properties of the lexical items can be mainly determined on the basis of their types. We assume the "bi-directional" system with two connectives / and \: X/Y is the type given to an expression that can combine with a following adjacent expression of type Y to give an expression of type X, and $Y\X$ is the type of expressions that can combine with a preceding adjacent expression of type Y to give an expression of type X.¹

For instance, *John* is given type NP and *John walks* is given type S. The verb *walks* forms a sentence whenever it is preceded by a noun phrase, and hence it receives the type NP\S. Likewise, *loves* forms a sentence when it is preceded by a noun phrase and followed by a noun phrase, so it may receive the type (NP\S)/NP or the type NP/(S/NP). The preposition *to* receives the type PP/NP, since expressions like *to Sue* have the type PP. Therefore *spoke* can receive the type (NP\S)/PP or NP/(S/PP).

As stated above, syntactic combinations are paired with semantic combinations. CG adopts Frege's principle of compositionality, whereby the meanings of expressions of primitive types are treated as basic, and the meanings of other expressions are defined in terms of their relation to these meanings. The meaning of an expression of type X/Y or $Y\X$ is given by a function which maps the meaning of any expression of type Y with which it combines to the meaning of the resultant expression of type X. For the moment, our interest in semantics is limited to those aspects of meaning which reflect function-argument structure. Hence we write the meaning of lexical items as non-logical constants, written as the same lexical item in boldface. Therefore we shall ignore model-theoretic aspects of meaning such as formal aspects of the lexical semantics of individual lexical items, or truth conditions associated with expressions. For example, the meaning of *walks* is **walks**, rather than, say, $\lambda x[\text{walk}(x)]$, and the meaning of *every* is **every** not $\lambda f[\lambda g[\forall x[f(x) \rightarrow g(x)]]]$. Later, we shall consider how these

¹This is Lambek's (1958) notation for types. In the alternative notation used by Steedman (1987), what we write as $Y\X$ is written as $X\Y$.

limited aspects of meaning can be converted into more complete semantic representations.

There are many versions of CG, which differ with respect to the ways in which types are allowed to combine. At one extreme, *classical* CG (Bar-Hillel 1953) allows a restricted range of combinations similar to phrase structure grammar. It is possible to combine *to*, category PP/NP, with *Sue*, category NP, to give the phrase *to Sue*, category PP. This is because the rule $X/Y Y \Rightarrow X$ is valid in classical CG. But it is not possible to combine *spoke*, category (NP\S)/PP, with *to*, category PP/NP, to give the category (NP\S)/NP, even though the expression *spoke to* can be made into a sentence by combining it with an NP to its right and an NP to its left. Hence classical CG has very limited combinatory power. At the other extreme, Lambek CG (Lambek 1958; Moortgat 1988) allows the combination of any two adjacent words, even if these words appear to have no linguistic relationship.

Dependency CG falls between these extremes: it allows the combination of words which are connected by dependencies. This means that it has some rough similarity to Combinatory Categorical Grammar (Steedman 1987). It can be defined by the following recursive schema:

- | | | | | |
|-----|-------|---|--------|---|
| (4) | (i) | $X/Y Y \Rightarrow X$ | (v) | If $X \Rightarrow Y$ then $X/Z \Rightarrow Y/Z$ |
| | (ii) | $Y Y \backslash X \Rightarrow X$ | (vi) | If $X \Rightarrow Y$ then $Z \backslash X \Rightarrow Z \backslash Y$ |
| | (iii) | $(Y \backslash X) / Z \Rightarrow Y \backslash (X / Z)$ | (vii) | If $X Y \Rightarrow Z$ then $X Y / W \Rightarrow Z / W$ |
| | (iv) | $Y \backslash (X / Z) \Rightarrow (Y \backslash X) / Z$ | (viii) | If $X Y \Rightarrow Z$ then $W \backslash X Y \Rightarrow W \backslash Z$ |

A more elegant formulation of dependency CG can be provided in terms of a natural deduction axiomatization of the system (see Barry and Pickering 1990), but we shall ignore this here. For example, these rules allow the combination of any adjacent words in *John spoke to Mary*. On the other hand, in *Fred said Bill laughed*, it is possible to combine *Fred* with *said*, or *Bill* with *laughed*, but not *said* or *Fred said* with *Bill*. This is the desired result, because *said Bill* and *Fred said Bill* are not dependency constituents. More technically, the system allows application, generalised composition and generalised swapping, but no version of type raising.

Two other points about the grammar should be made. First, the treatment of modifiers in dependency CG is different from most other versions of CG in that the modifier is treated as the argument rather than the functor. This is in keeping with the assumptions of DG, where the dependency leads from a word in the modified phrase to the root of the modifier. The types for modifiers are simple, such as PoN for postnominal modifiers, rather than N\N. This means that the interpretations given to expressions containing modifiers often appears rather strange: *John walks slowly* is given the interpretation (**walks slowly**) **John**, rather than, say, **slowly** (**walks John**). Hence, a considerable amount of work is needed to convert these interpretations into complete interpretations that can be employed in model-theoretic semantics, which we shall not discuss here. Second, dependency CG can deal with unbounded dependencies by employing complex categories for relative pronouns and similar elements, as in other versions of categorial grammar (e.g. Ades and Steedman 1982). For example, *whom* is given the type PoN/(S/NP), which means that it can be combined with an expression of type S/NP, such as *John likes*, to give a postnominal modifier of type PoN. It is important to note that it is impossible to represent this with the dependency grammar notation. This means that the sentence processing account presented in this paper has to be treated in terms of dependency CG; only the component without unbounded dependencies can be illustrated in terms of dependency diagrams.

Incremental Processing

I shall now describe a model of incremental sentence processing based on the interpretations of dependency constituents, using a shift-reduce parser with a default reduce-first strategy (see Ades and Steedman 1982). Some of this framework is described in Pickering (1993). The sentence processor tries to construct a dependency constituent and a unified interpretation for each fragment that is encountered; if this is impossible, it makes use of a stack. (for now, we shall ignore cases of local ambiguity). In processing (2), *Fred thinks* is interpreted as a unit immediately it is encountered, since it is a dependency constituent. But *Fred thinks Bill* cannot be so interpreted, and therefore *Fred thinks* and *Bill* are given separate syntactic types and interpretations. This demonstrates how the model incorporates a delay component.

For example, here are the way in which (1) and (2) are processed:

Word	Category	Reductions	Stack		
John	NP		NP		
spoke	(NP\S)/PP	1	S/PP		
to	PP/NP	1	S/NP		
Mary	NP	1	S		

Word	Category	Reductions	Stack		
Fred	NP		NP		
thinks	(NP\S)/S	1	S/S		
Bill	NP		S/S	NP	
laughed	NP\S	2	S		

The bottom of the stack is represented on the left. After *Bill*, there are two elements on the stack. This corresponds to the fact that there are two dependency constituents at this point in the parse.

Pickering (1993) discusses how this account can explain the differential complexity of various kinds of recursive constructions. On the one hand, sentences with "right branching" phrase structural representations can be modelled with a shallow stack in which the categories never become complex:

Word	Category	No. of Reductions	Stack		
I	NP		NP		
saw	(NP\S)/NP	1	S/NP		
the	NP/N	1	S/N		
farmer	N/PoN	1	S/PoN		
who	PoN/ (NP\S)	1	S/(NP\S)		
owned	(NP\S)/NP	1	S/NP		
the	NP/N	1	S/N		
dog	N/PoN	1	S/PoN		
which	PoN/ (NP\S)	1	S/(NP\S)		
chased	(NP\S)/NP	1	S/NP		
the	NP/N	1	S/N		
cat	N	1	S		

There is one reduction as every word is added after the first, and the stack never increases beyond a stack depth of one. This corresponds to the fact that each substring of the sentence which includes the first word is a dependency constituent. Notice that the same pattern repeats itself; for instance, the stack after *saw*, *owned* and *chased* is identical.

On the other hand, "nested" constructions require the use of a deep stack (or, on some occasions, a shallow stack with categories of great complexity):

Word	Category	No. of Reductions	Stack		
The	NP/N		NP/N		
cat	N/PoN	1	NP/PoN		
which	PoN/(S/NP)	1	NP/(S/NP)		
the	NP/N		NP/(S/NP)	NP/N	
dog	N/PoN	1	NP/(S/NP)	NP/PoN	
which	PoN/(S/NP)	1	NP/(S/NP)	NP/(S/NP)	
the	NP/N		NP/(S/NP)	NP/(S/NP)	NP/N
farmer	N	1	NP/(S/NP)	NP/(S/NP)	NP
owned	(NP\S)/NP	2	NP/(S/NP)	NP	
chased	(NP\S)/NP	2	NP		
fled	NP\S	1	S		

As each new relative clause is added, the stack depth increases by one, in contrast to the right branching example. This leads to the correct prediction that nested constructions become more difficult to process as each relative clause is added, unlike right branching examples. In the nested example, after *farmer* has been processed, the dependency constituents *the cat which*, *the dog which* and *the farmer* have to be remembered separately, and have to be given independent interpretations. Notice that the complexity of processing reduces when the verbs *owned* and *chased* are encountered. It is possible that the complexity is due to this process of reducing the depth of the stack, rather than remembering a range of elements.

This account can be used to model the data discussed by Pickering and Barry (1991), who provide evidence that the processor does not wait until the proposed "gap" or "empty category" site before forming an unbounded dependency. In a sentence like *On which saucer did you put the cup*, the purported empty category site is sentence final, after *cup*: This is because the "canonical" word order is *you put the cup on which saucer*. Pickering and Barry provide evidence that the unbounded dependency is in fact formed at *put*, and the current model of incremental processing models this: a single dependency constituent is formed at *put*.

So far, the account has described the operations of a modular sentence processor. However, the interpretations of dependency constituents are treated as the outputs of the module which can then be related to "central processes" such as general knowledge. In particular, the assumption is that these interpretations can be employed in reasoning. But there is a problem at this point: the interpretations are in general not propositional, and so it is unclear how they could be used as premises in inference. Hence the next stage in the model has to convert these non-propositional interpretations into propositions (Chater, Pickering and Milward ms.). Roughly, this is done by existentially quantifying over missing arguments, so that *Fred thinks* is interpreted by central processes

as meaning that (it is being expressed that) Fred thinks something. Fragments such as *Fred thinks Bill* are treated by central processes as expressing two independent propositions: roughly, that Fred thinks something, and that Bill exists. It is possible that on-line inferencing will occur on the basis of these two propositions, but no inferencing will be based on any interpretation of *Fred thinks Bill* as a whole. The assumption is that dependency constituents are the linguistic units which can serve as the basis for inference.

Modelling Experimental Data

On this account, sentence processing involves a delay component. Much sentence processing is completely incremental, in that the processor can form a dependency constituent immediately a string of words is encountered. But if no single dependency constituent can be formed, then the fragment is not given a unified analysis.

Nicol and Pickering (in press) report an experiment using cross-modal priming which is incompatible with most standard accounts of sentence processing. However, their findings can be incorporated into an extended version of this account. Sentence (6) below is in fact ambiguous between a complement clause and a relative clause interpretation:

- (6) The receptionist informed the doctor that the journalist had phoned about the events.

However, the relative clause interpretation (where *that the journalist had phoned* modifies *the doctor*) is very hard to obtain. Most models of processing (e.g. Frazier 1979; Crain and Steedman 1985) assume that the choice of reading is made by the time *doctor* or *that* is encountered. However, we found priming of *nurse*, an associate of *doctor*, when it was presented at the offset of *phoned*, compared with a control sentence where *that* is replaced by *why*. The difference between these two sentences is that the relative clause reading is potentially available in the experimental material, and the priming indicates that the relative clause reading is in fact obtained (in accordance with Swinney, Ford, Frauenfelder and Bresnan ms.). In other words, this is evidence that the dispreferred reading has not been discarded at the embedded verb.

The present account can model this data in the following way. The NP *the journalist* cannot be linked into a dependency constituent with *the receptionist informed the doctor*. At this point, there are simply two interpreted but unconnected sentence fragments (as well as the *that*). These can be linked into a single dependency constituent when *had phoned* is encountered. However, this combination can be conducted in two ways, forming either the relative clause or the complement clause analysis. The assumption is that the complement clause reading is in fact chosen, but that the relative clause reading is momentarily active after *had phoned*. This analysis is then rapidly discarded, leaving only the complement clause analysis. On this account, the technique is tapping into is a dispreferred syntactic analysis, analogous to a dispreferred sense of a word (e.g. Swinney 1979). At present, I am running an extended version of this experiment, which I hope to be able to report at the conference. I also intend to make some proposals regarding the processing of other kinds of ambiguity in terms of this general framework.

The choice of analyses at *had phoned* constitutes a reduce-reduce ambiguity in terms of the shift-reduce parser: on either analysis, a single dependency constituent is formed, though the interpretation associated with the two analyses is different. In this case, the model assumes that both readings are computed in parallel and a choice is then made between them,

