Multiple Interpreters in a Principle-Based Model of Sentence Processing

Matthew W. Crocker
e-mail: mwc@aipna.ed.ac.uk

Department of Artificial Intelligence
University of Edinburgh
80 South Bridge
Edinburgh, Scotland, EH1 1HN

and

Human Communication Research Centre
University of Edinburgh
2 Buccleuch Place
Edinburgh, Scotland, EH8 9LW

Abstract

This paper describes a computational model of human sentence processing based on the *principles and parameters* paradigm of current linguistic theory. The syntactic processing model posits four modules, recovering phrase structure, long-distance dependencies, coreference, and thematic structure. These four modules are implemented as meta-interpreters over their relevant components of the grammar, permitting variation in the deductive strategies employed by each module. These four interpreters are also 'coroutined' via the *freeze* directive of constraint logic programming to achieve incremental interpretation across the modules.

1 Introduction

A central aim of computational psycholinguistics is the development of models of human sentence processing which account not only for empirical performance phenomena, but which also provide some insight into the nature of between parser and grammar relationship. In concurrent research, we are developing a model of sentence processing which has its roots in the *principles and parameters* paradigm of syntactic theory [1], [2], which holds that a number of representations are involved in determining a well-formed analysis of an utterance. This, in conjunction with Fodor's *Modularity Hypothesis* [6], has led us to postulate a model which consists of four informationally encapsulated modules for recovering (1) phrase structure, (2) chains (3) coreference, and (4) thematic structure.

In this paper we will briefly review a model of sentence processing which has been previously proposed in [5] and [3]. We will illustrate how this model can be naturally implemented within the logic programming paradigm. In particular, we sketch a subset of GB theory which defines principles in terms of their representational units, or *schemas*. We then discuss how the individual processors may be implemented as specialised, informationally encapsulated interpreters, and discuss how the 'freeze' directive of constraint logic programming can be used to effectively coroutine the interpreters, to achieve incremental interpretation and concurrency.

2 The Processing Model

In the proposed model, we assume that the sentence processor strives to optimise local comprehension and integration of incoming material, into the current context. That is, decisions about the current syntactic analysis are made incrementally (for each input item) on the basis of principles which are intended maximise the overall interpretation. We have dubbed this the *Principle of Incremental Comprehension* (PIC), stated roughly as follows:

(1) **Principle of Incremental Comprehension:** The sentence processor operates in such a way as to maximise comprehension of the sentence at each stage of processing.

The PIC demands that the language comprehension system (LCS), and any sub-system contained within it (such as the syntactic and semantic processors), apply maximally to any input, thereby constructing a maximal, partial interpretation for a given partial input signal. This entails that each module within the LCS apply concurrently.

The performance model is taken to be directly compatible with the modular, language universal, principle-based theory of current transformational grammar [3]. We further suggest a highly modular organisation of the sentence processor wherein modules are determined by the syntactic representations they recover. This is motivated more generally by Fodor's Modularity Hypothesis [6] which argues that the various perceptual/input systems consist of fast, dumb informationally encapsulated modules. Specifically, we posit four modules *within* the syntactic processor, each affiliated with a "representational" or "informational" aspect of the grammar. These are outlined below in conjunction with the grammatical subsystems to which they are related[1]:

[1] This grouping of grammatical principles with representations is both partial and provisional, and is intended only to give the reader a feel for the "natural classes" exploited by the model.

(2)

Modules & Principles:	
Phrase structure (PS):	$\overline{X}$-theory, Move-α
Chains (ChS):	Bounding, ECP
Theta structure (TS):	θ-theory
Coreference (CiS):	Binding, Control

In Figure 1, we illustrate one possible instance of the organisation within the Syntactic Processor. We assume that the phrase structure module drives processing based on lexical input, and that the thematic structure co-ordinates the relevant aspects of each processor for output to the semantic processor.

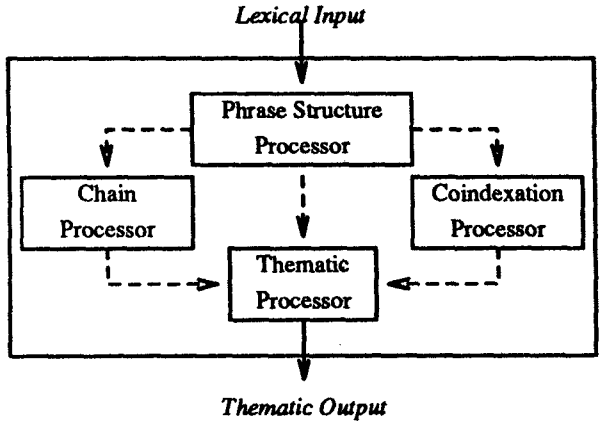

Figure 1: Syntactic Processor Organisation

Just as the PIC applies to the main modules of the LCS as discussed above, it also entails that all modules within the syntactic processor act concurrently so as to apply maximally for a partial input, as illustrated by the operation shown in Figure 2. For the partial input *"What did John put ..."*, we can recover the partial phrase structure, including the *trace* in Infl[2]. In addition, we can recover the chain linking the *did* to its deep structure position in Infl (e-1), and also the θ-grid for the relation 'put' including the saturated agent role 'John'. We might also go one step further and postulate a trace as the direct object of *put*, which could be the θ-position of *What*, but this action might be incorrect if the sentence turned out to be *What did John put the book on?*, for example.

3 Principles and Representations

Before proceeding to a discussion of the model's implementation, we will first examine more closely the representational paradigm which is employed, and discuss some aspects of the principles and parameters theory we have adopted, restricting our attention primarily to phrase structure and Chains. In general, a

[2] We assume here a head movement analysis, where the head of Infl moves to the head of Comp, to account for Subject-Aux inversion.

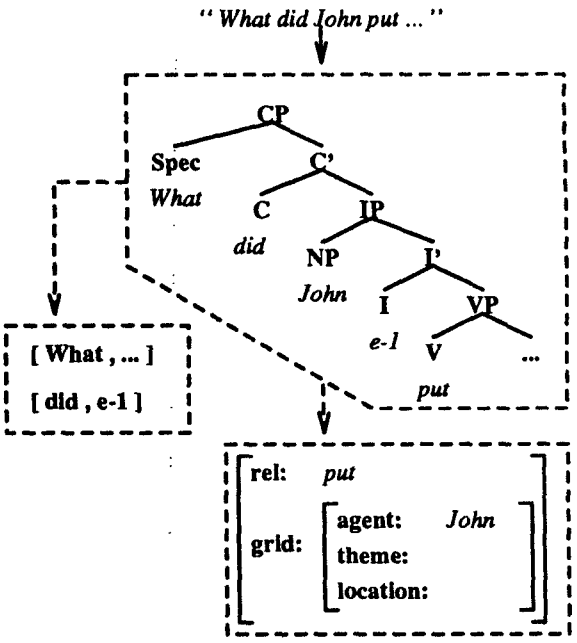

Figure 2: Syntactic Processor Operation

particular representation can be broken down into two fundamental components: 1) *units of information*, i.e. the 'nodes' or feature-bundles which are fundamental to the representation, and 2) *units of structure*, the minimal structural 'schema' for relating 'nodes' with each other. With these two notions defined, the representation can be viewed as some recursive instance of its particular schema over a collection of nodes.

The representation of phrase structure (PS), as determined principally by $\overline{X}$-theory, encodes the local, sister-hood relations and defines constituency. The bar-level notation is used to distinguish the status of satellites:

(3) (a) $\overline{\overline{X}} \rightarrow \textit{Specifier}, \overline{X}$
(b) $\overline{X} \rightarrow \textit{Modifier}, \overline{X}$
(c) $\overline{X} \rightarrow \textit{Complements}, X$
(d) $X \rightarrow \textit{Lexeme}$

The linear precedence of satellites with respect to their sister X-nodes is taken to be parametrised for each category and language. The final rule (d) above, is simply the rule for lexical insertion. In addition to the canonical structures defined by $\overline{X}$-theory, we require additional rules to permit Chomsky-adjunction of phrases to maximal projections, via Move-α, and the rules for inserting traces (or more generally, *empty categories*) — a consequence of the Projection Principle — for moved heads and maximal projections.

Given the rules above, we can see that possible phrase structures are limited to some combination of binary (non-terminal) and unary (terminal) branches. As discussed above, we can characterise the representational framework in terms of *nodes* and *schemas*:

Phrase Structure Schema	
Node:	N-Node: {Cat,Level,ID,Ftrs} T-Node: {Cat,Phon,ID,Ftrs}
Schema:	Branch: N-Node/[N-Node,N-Node] Branch: N-Node/T-Node
Structure:	Tree: N-Node/[Tree_L,Tree_R] Tree: N-Node/T-Node

We allow two types of nodes: 1) non-terminals (N-Nodes), which are the nodes projected by $\overline{X}$-theory, consisting of category, bar level, a unique ID, and the features projected from the head, and 2) terminals (T-Nodes), which are either lexical items or empty categories, which lack bar level, but posses phonological features (although these may be 'nil' for some empty categories). The schema, defines the unit of structure, using the '/' to represent immediate dominance, and square brackets '[...]' to encode sister-hood and linear precedence. Using this notation we define the two possible types of branches, binary and unary, where the latter is applicable just in case the daughter is a terminal node. The full PS representation (or Tree) is defined by allowing non-terminal daughters to dominate a recursive instance of the schema. It is interesting to note that, for phrase structure at least, the relevant principles of grammar can be stated purely as conditions on branches, rather that trees. More generally, we will assume the schema of a particular representation provides a formal characterisation of locality.

Just as phrase structure is defined in terms of branches, we can define Chains as a sequence of links. More specifically, each position contained by the chain is a *node*, which represents its category and level (a phrase or a head), the status of that position (either A or $\overline{A}$), its ID (or location), and relevant features (such as L-marking, Case, and θ). If we adhere to the representational paradigm used above, we can define Chains in the following manner:

Chain Schema	
Node:	C-Node: {Cat,Level,Pos,ID,Ftrs}
Schema:	Link: <C-Node$_i$ ∞ C-Node$_j$>
Structure:	Chain: [C-Node | Chain] (where, <C-Node ∞ head(Chain)>) Chain: [ ]

If we let '∞' denote the linking of two *C-Nodes*, then we can define a *Chain* to be an ordered list of C-Nodes, such that successive C-Nodes satisfy the link relation. In the above definition we have used the '|' operator and list notation in the standard Prolog sense. The 'head' function returns the first C-Node in a (sub) Chain (possibly [ ]), for purposes of satisfying the link relation. Furthermore, <C-Node ∞ [ ]> is a well-formed link denoting the tail, Deep-Structure position, of a Chain. Indeed, if this is the only link in the Chain we refer to it as a 'unit' Chain, representing an unmoved element.

We noted above that each representation's *schema* provides a natural locality constraint. That is, we should be able to state relevant principles and constraints locally, in terms of the schematic unit. This clearly holds for Subjacency, a well-formedness condition which holds between two nodes of a link:

(4) <C-Node$_i$ ∞ C-Node$_j$> $\rightarrow$ subjacent(C-Node$_i$,C-Node$_j$)

Other Chain conditions include the Case filter and θ-Criterion. The former stipulates that each NP Chain receive Case at the 'highest' A-position, while the latter entails that each argument Chain receive exactly one θ-role, assigned to the uniquely identifiable < C-Node$_\theta$ ∞ [ ] > link in a Chain. It is therefore possible to enforce both of these conditions on locally identifiable links of a Chain:

(5) In an argument (NP) Chain,
i) <C-Node$_{\overline{A}}$ ∞ C-Node$_A$> $\rightarrow$ case-mark(C-Node$_A$) or,
ii) C-Node$_A$ = head(Chain) $\rightarrow$ case-mark(C-Node$_A$)

In an argument Chain,
<C-Node$_\theta$ ∞ [ ]> $\rightarrow$ theta-mark(C-Node$_\theta$)

In describing the representation of a Chain, we have drawn upon Prolog's list notation. To carry this further, we can consider the link operator '∞' to be equivalent to the ',' separator in a list, such that for all [...C-Node$_i$,C-Node$_j$ ...], C-Node$_i$ ∞ C-Node$_j$ holds. In this way, we ensure that each node is well-formed with respect to adjacent nodes (i.e. in accordance with principles such as those identified in (4) & (5)).

4 The Computational Model

In the same manner that linguistic theory makes the distinction between competence (the grammar) and performance (the parser), logic programming distinguishes the declarative specification of a program from its execution. A program specification consists of a set of axioms from which solution(s) can be proved as derived theorems. Within this paradigm, the nature of computation is determined by the inferencing strategy employed by the theorem prover. This aspect of logic programming has often been exploited for parsing; the so called *Parsing as Deduction* hypothesis. In particular it has been shown that meta-interpreters or program transformations can be used to affect the manner in which a logic grammar is parsed [10] [11].

Recently, there has been an attempt to extend the PAD hypothesis beyond its application to simple logic grammars [14], [13] and [8]. In particular, Johnson has developed a prototype parser for a fragment of a GB grammar [9]. The system consists of a declarative specification of the GB model, which incorporates

the various principles of grammar and multiple levels of representation. Johnson then illustrates how the fold/unfold transformation and goal *freezing*, when applied to various components of the grammar, can be used to render more or less efficient implementations. Unsurprisingly, this deductive approach to parsing inherits a number of problems with automated deduction in general. Real (implemented) theorem provers are, at least in the general case, incomplete. Indeed, we can imagine that a true, deductive implementation of GB would present a problem. Unlike traditional, homogeneous phrase structure grammars, GB makes use of abstract, modular principles, each of which may be relevant to only a particular type or level of representation. This modular, heterogeneous organisation therefore makes the task of deriving some single, specialised interpreter with adequate coverage and efficiency, a very difficult one.

4.1 Deduction in a Modular System

In contrast with the single processor model employed by Johnson, the system we propose consists of a number of processors over subsets of the grammar. Central to the model is a declarative specification of the principles of grammar, defined in terms of the representations listed in (2), as described in §3. If we take this specification of the grammar to be the "competence component", then the "performance component" can be stated as a parse relation which maps the input string to a well-formed "State", where State = { PS,TS,ChS,CiS }, the 4-tuple constituting all aspects of syntactic analysis. The highest level of the parser specifies how each module may communicate with the others. Specifically, the PS processor acts as input to the other processors which construct their representations based on the PS representations and their own "representation specific" knowledge. In a weaker model, it may be possible for processors to inspect the current State (i.e. the other representations) but crucially, no processor ever actually "constructs" another processor's representation. The communication between modules is made explicit by the Prolog specification shown below:

(6)
```
parse(LexInput,State) : –
        State = {PS,TS,ChS,CiS},
        ts_module(PS,TS),
        chs_module(PS,ChS),
        cis_module(PS,CiS),
        ps_module(LexInput,PS).
```

The `parse` relation defines the organisation of the processors as shown in Figure 1. The Prolog specification above appears to suffer from the traditional depth-first, left-to-right computation strategy used by Prolog. That is, `parse` seems to imply the sequential execution of each processor. As Stabler has illustrated, however, it is possible to alter the computation rule used [12], so as to permit incremental interpretation by each module: effectively coroutining the various modules. Specifically, Prolog's *freeze* directive allows processing of a predicate to be delayed temporarily until a particular argument variable is (partially) instantiated. In accord with the input dependencies shown in (7), each module is frozen (or 'waits') on its input:

(7)

Input dependencies	
`ts_module`	PS
`chs_module`	PS
`cis_module`	PS
`ps_module`	LexInput

Using this feature we may effectively "coroutine" the four modules, by freezing the PS processor on Input, and freezing the remaining processors on PS. The result is that each representation is constructed incrementally, at each stage of processing. To illustrate this, consider once again the partial input string *"What did John put ..."*. The result of the call `parse([what,did,john,put|_],State)` would yield the following instantiation of State (with the representations simplified for readability):

(8)
```
State = {  PS = cp/[np/what,
                    c1/[c/did,
                        ip/[np/John
                            i1/[i/trace-1,
                                vp/[v/put, _]]]],

           TS = [rel:put,grid:[agent:john | _]],

           ChS = [ [what, _], [did,trace-1] ],

           CiS = _  }
```

The PS representation has been constructed as much as possible, including the trace of the moved head of Infl. The ChS represents a partial chain for *what* and the entire chain for *did*, which moved from its deep structure position to the head of CP, and TS contains a partial θ-grid for the relation 'put', in which the Agent role as been saturated.

This is reminiscent of Johnson's approach [9], but differs crucially in a number of respects. Firstly, we posit several processors which logically exploit the grammar, and it is these processors which are coroutined, not the principles of grammar themselves. Each interpreter is responsible for recovering only one, homogeneous representation, with respect to one input representation. This makes reasoning about the computational behaviour of individual processors much easier. At each stage of processing, the individual processors increment their representations if and only if, for the current input, there is a "theorem" provable from the grammar, which permits the new structure to be added. This meta-level "parsing as deduction" approach permits more finely tuned control of the parser as a whole, and allows us to specify distinct inferencing strategies for each interpreter, tailoring it to the particular representation.

4.2 The PS-Module Specification

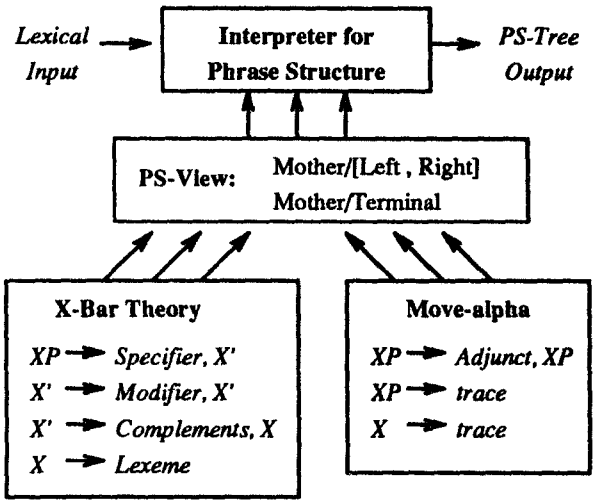

Figure 3: The Phrase Structure Module

We have illustrated in §3 that the various representations and grammatical principles may be defined in terms of their respective schematic units. Given this, the task of recovering representations (roughly parsing) is simply a matter of proving well-formed representations, as recursive instances of 'schematic axioms', i.e. those instantiations of a schema which are considered well-formed by the grammar. The form of the PS-Module can be depicted as in Figure 3. The PS interpreter incorporates lexical input into the phrase structure tree based on possible structures allowed by the grammar. Possible structures are determined by the `ps_view` relation, which returns those possible instantiations of the PS schema (as described in §3) which are well-formed with respect to the relevant principles of grammar. In general, `ps_view` will return any possible branch structure licensed by the grammar, but is usually constrained by partial instantiation of the query. In cases where multiple analyses are possible, the `ps_view` predicate may use some selection rule to choose between them [3]. The following is a specification of the PS interpreter:

```
(9)  ps_module(X-X0,Node/[L/LD,R/RD]) : –
         non_lexical(Node),
         ps_view(Node/[L,R]),
         ps_module(X-X1,L/LD),
         ps_module(X1-X0,R/RD).
     ps_module(X-X0,Node/Daughters) : –
         ps_ec_eval(X-X0,Node/Daughters).
     ps_module(X-X0,Node/Daughters) : –
         ps_lex_eval(X-X0,Node/Daughters).
```

As we have discussed above, the `ps_module` is frozen on lexical input represented here as as difference-list.

[3] This is one way in which we might implement attachment principles to account for human preferences, see Crocker [4] for discussion.

The top-level of the PS interpreter is broken down into three possible cases. The first handles non-lexical nodes, i.e. those of category C or I, since phrase structure for these categories is not lexically determined, and can be derived 'top-down', strictly from the grammar. We can, for example, automatically hypothesize a Subject specifier and VP complement for Infl. The second clause deals with the postulation of empty categories, while the third can be considered the 'base' case which simply attaches lexical material. Roughly, `ps_ec_eval` attempts to derive a leaf which is a trace. This action is then verified by the concurrent Chain processor, which determines whether or not the trace is licensed (see the following section). This implements an approximation of the filler-driven strategy for identifying traces, a strategy widely accepted in the psycholinguistic literature[4].

4.3 The Chain-Module Specification

Just as the phrase structure processor is delayed on lexical input, the chain processor is frozen with respect to phrase structure. The organisation of the Chain Module is shown in Figure 4, and is virtually identical to that of the PS Module (in Figure 3). However rather than recovering branches of phrase structure, it recovers links of chains, determining their well-formedness with respect to the relevant grammatical axioms.

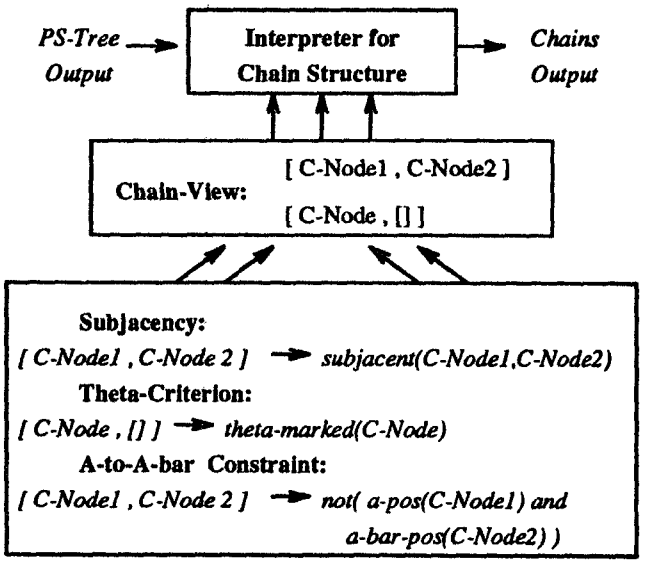

Figure 4: The Chain Module

For this module, incremental processing is implemented by 'freezing' with respect to the input tree representation. The following code illustrates how the top-level of the Chain interpreter can traverse the PS tree, such that it is coroutined with the recovery of the PS representation:

[4] The is roughly the *Active Filler Strategy* [7]. For discussion on implementing such strategies within the present model see [4].

(10)
```
chs_module(X/[L/LD,R/RD],CS) : –
        chain_int(X/[L,R],CS),
        chs_module(L/LD,CS),
        chs_module(R/RD,CS).
chs_module(X/Leaf,CS) : –
        chain_int(X/Leaf,CS).
```

I will assume that **chs_module** is frozen such that it will only execute if the daughter(s) of the current sub-tree is instantiated. Given this, **chs_module** will perform a top-down traversal of the PS tree, delaying when the daughters are uninstantiated, thus coroutined with the PS-module. The **chain_int** predicate then determines if any action is to be taken, for the current branch, by the chain interpreter:

(11)
```
chain_int(X/[Satellite,Right],CS) : –
        visible(X/[Satellite,Right],C-Node),
        chain_member(C-Nodes,CS).
chain_int(X/[Left,Satellite],CS) : –
        visible(X/[Left,Satellite],C-Node),
        chain_member(C-Nodes,CS).
```

The **chain_int** predicate decides whether or not the satellite of the current branch is relevant, or 'visible' to the Chain interpreter, and if so returns an appropriate C-Node for that element. The two visible entities are antecedents, i.e. arguments (if we assume that all arguments form chains, possibly of unit length) or elements in an $\overline{A}$ positions (such as [Spec,CP] or a Chomsky-adjoined position) and traces. If a trace or an antecedent is identified, then it must be a member of a well-formed chain. The **chain_member** predicate postulates new chains for lexical antecedents, and attempts to append traces to existing chains. This operation must in turn satisfy the **chain_view** relation, to ensure the added link obeys the relevant grammatical constraints.

5 Summary and Discussion

In constructing a computational model of the proposed theory of processing, we have employed the logic programming paradigm which permits the transparent distinction between competence and performance. At the highest level, we have a simple specification of the models organisation, in addition we have employed a 'freeze' control strategy which permits us to coroutine the individual processors, permitting maximal incremental interpretation. The individual processors consist of specialised interpreters which are in turn designed to perform incrementally. The interpreters construct their representations, by incrementally adding units of structure which are locally well-formed with respect to the principles of the module. The implementation is intended to allow some flexibility in specifying the grammatical principles, and varying the control strategies of various modules. It is possible that some instantiations of the syntactic theory will prove more or less compatible with the processing model. It is hoped that such results may point the way to a more unified theory of grammar and processing or will at least shed greater light on the nature of their relationship.

Acknowledgements

I would like to thank Elisabet Engdahl and Robin Cooper for their comments on various aspects of this work. This research was conducted under the support of an ORS award, an Edinburgh University Studentship and the Human Communication Research Centre.

References

[1] N. Chomsky. *Barriers. Linguistic Inquiry Monograph Thirteen*, The MIT Press, Cambridge, Massachusetts, 1986.

[2] N. Chomsky. *Knowledge of Language: Its Nature, Origin and Use. Convergence Series*, Praeger, New York, 1986.

[3] M. Crocker. Incrementality and Modularity in a Principle-Based Model of Sentence Processing. Presented at *The Workshop on GB-Parsing*, Geneva, Switzerland, 1990.

[4] M. Crocker. Multiple Meta-Interpreters in a Logical Model of Sentence Processing. In Brown and Koch, editors, *Natural Language Understanding and Logic Programming, III*, Elsevier Science Publishers (North-Holland), (to appear).

[5] M. Crocker. *Principle-Based Sentence Processing: A Cross-Linguistic Account.* HCRC/RP 1, Human Communication Research Centre, University of Edinburgh, U.K., March 1990.

[6] J. Fodor. *Modularity of Mind.* MIT Press, Cambridge, Massachusetts, 1983.

[7] L. Frazier and C. Clifton. Successive Cyclicity in the Grammar and Parser. *Language and Cognitive Processes*, 4(2):93–126, 1989.

[8] M. Johnson. Program Transformation Techniques for Deductive Parsing. In Brown and Koch, editors, *Natural Language Understanding and Logic Programming, III*, Elsevier Science Publishers (North-Holland), (to appear).

[9] M. Johnson. Use of Knowledge of Language. *Journal of Psycholinguistic Research*, 18(1), 1989.

[10] F. Pereira and D. Warren. Parsing as Deduction. In *Proceedings of Twenty-First Conference of the ACL*, Cambridge, Massachusetts, 1983.

[11] F. Pereira and S. Shieber. *Prolog and Natural-Language Analysis. CSLI Lecture Notes*, Center for the Study of Language and Information, Stanford, California, 1987.

[12] E. Stabler. Avoid the pedestrian's paradox. unpublished ms., Dept. of Linguistics, UCLA, 1989.

[13] E. Stabler. Relaxation Techniques for Principle-Based Parsing. Presented at *The Workshop on GB Parsing*, Geneva, Switzerland, 1990.

[14] E. Stabler. *The Logical Approach to Syntax.* The MIT Press, Cambridge, Massachusetts, (forthcoming).