Natural Language Understanding and Logic Programming, III
C. Brown and G. Koch (Editors)
© Elsevier Science Publishers B.V. (North-Holland), 1991

Principle-Based Parsing and Type Inference

Sandiway Fong[1]

Artificial Intelligence Laboratory, MIT, 545 Technology Sq. NE43-810
Cambridge MA 02139 USA. E-mail: sandiway@ai.mit.edu

Abstract

In principle-based parsing, the two basic operations are phrase structure construction and the application of well-formedness conditions that implement linguistic principles. In the interleaved model of execution, principles are applied to partially constructed phrase structures as they are built. By introducing the notion of the type of a principle, we show how type inference rules defined over logic programs can be used as a basis for a practical model of interleaving. We describe an off-line procedure for the automatic interleaving of principles defined by logic programs. An attractive feature of the implemented procedure is that it is automatic in the sense that it is completely transparent to principle definitions.

1 INTRODUCTION

1.1 THE INTERLEAVED MODEL OF EXECUTION

Recently, there has been some interest in parsing strategies for principle-based parsers, e.g. Crocker [3], Fong [4], Johnson [6], Macías [9] and Stabler [11] [12]. In principles-and-parameters theories of syntax, knowledge of grammar is mostly encoded using a set of linguistic 'principles': i.e. well-formedness conditions that phrase structures must satisfy. Phrase structures are, in turn, generated by a small set of simple context-free (CF) grammar rules in accordance with the restrictive principles of $\overline{\text{X}}$-theory.

In general, we can use a wide variety of different parsing strategies for these theories. For example, perhaps the most obvious strategy is to simply enumerate the possible phrase structures for a given sentence, and then 'weed-out' those that are ill-formed by applying each well-formedness condition, or principle, in turn. Then, only those phrase structures that satisfy all well-formedness conditions can be said to be parses of the given sentence. Note that in such a scheme, there is a strict separation between phrase structure construction and principle application. That is, phrase structure for a complete

[1]Support for this work has been provided in part by a research assistant award and in part by NSF Grant No. DCR-85552543 under a Presidential Young Investigator Award to Professor Robert C. Berwick.

sentence will be built before it is tested for well-formedness. An alternative strategy may be based on the observation that we need not wait until phrase structure construction is complete before applying certain principles. That is, many principles exhibit *locality of reference* in the sense that may apply only to certain sub-structures within a given phrase structure. (We discuss such an example immediately below.) Hence, we can take advantage of this by 'mixing', or *interleaving*, principle application with phrase structure construction as individual phrases are built. The motivating principle here is that if some sub-structure is found to be ill-formed, then any complete structure built from that sub-structure will also be ill-formed. For example, the sentence *John saw he* is ill-formed — cf. the well-formed sentence *John saw him.* In fact, the verb phrase corresponding to *saw he* is ill-formed no matter what elements precede or follow it. For example, *Mary believes John saw he*, *John saw he win*, and *I thought that Mary believed John saw he* are all ill-formed sentences. The problem here is that Case assignment, a principle of Case theory, states that verbs such as *see* assign accusative Case to noun phrase elements in object position such as *he.* Therefore, the Case conflict can be discovered by applying the principle of Case assignment as soon as the object of *see* has been identified. Hence, it would seem to make sense, computationally speaking, to favour the interleaved over the non-interleaved approach described earlier.[2] In this paper, we will consider the efficient implementation of the interleaved model.

1.2 OVERVIEW OF THE APPROACH

1.2.1 The Naïve Polling Model

Perhaps the obvious (and general) method for interleaved execution is to simply try to apply each principle every time some structure is built. We will call this the *naïve polling model.* Unfortunately, this is an impractical model because of the relatively high overhead of 'polling' constituents. The reason why polling is expensive comes from the general observation that many linguistic principles are fairly restrictive about the range of configurations that they apply to. In other words, many principles have the following structure: "for all constituents X such that X satisfies some property P, then some property Q *must* hold or else some addition (or modification) R *must* be made to constituent structure." For example, let us consider the principle of Case assignment that was introduced in the previous section. The principle basically states that: "for all configurations of Case assignment X, assign Case to an appropriate noun phrase in X." However, the basic problem here is that it may take a considerable amount of computation to identify and retrieve the relevant components of a Case configuration. For example, we will need to find an element to assign Case, we will also need to find a corresponding element to receive Case, and finally, we will need to check that these two elements bear the appropriate structural relation to each other. But since these requirements are fairly restrictive, most constituents will fail at one of three points mentioned. However, this repeated evaluation of the pre-conditions will result in a high overhead above the amount of computation required to process those configurations that turn out to be Case configurations. (A

[2] We do not address the performance tradeoffs between the two approaches here. Actually, for reasons beyond the scope of this paper, it is not clear that the interleaved approach is always better. See Fong [5] for an explanation.

more concrete example of a principle along these lines may be found in the definition of θ-role-assignment to be discussed in section 2.3.)

Note however that, unless we exhaustively test each structure, we run the risk of either failing to rule out an ill-formed structure or causing a well-formed structure to be ruled out (by failing to assign Case.) In other words, non-exhaustive testing would constitute an *unsound* implementation of the (relativized) universal quantifier. Of course, if certain classes of structures that never satisfy the pre-conditions of a given principle can be identified and eliminated from consideration (on an *off-line* basis), this can lead to a substantial reduction in the number of (unsuccessful) tests necessary at *parse-time*. In the next section, we will outline an improved model that will only need to selectively poll partially-constructed phrase structures.

1.2.2 The Revised Model

Basically, the idea will be to cut the overhead by using information about the range of possible phrases that the pre-conditions of a principle may apply to. For instance, if we can determine that a certain pre-condition only applied to, say, noun, verb and adjectival phrases; then we can safely eliminate polling, say, of all adverbial phrases. Of course, the problem is how to extract this categorial information simply by inspecting the logic definition of a principle. Roughly speaking, a *type* will be a set that represents the range of category labels associated with a configuration. A type inference mechanism that operates over logic definitions will used to compute type values. Before proceeding with the revised model, we will first briefly sketch some of the design goals of the larger parsing system that will provide an idea of the limitations and tradeoffs necessary in the type computation mechanism. (For the complete details, the reader is referred to Fong [5]).

The question of how to represent linguistic principles is one of the most important issues for principle-based parsing. That is, how should we formally represent the natural language, or semi-formal, definitions that are commonly found in the linguistics literature? Of course, it will be generally desirable to use a representation such as logic with a structure and a vocabulary that is 'close' to the notation used by linguists for reasons of perspicuity, ease of theory maintenance, and to ensure correctness of representation.[3] At the same time, for practical reasons, it will also be desirable to have efficiently executable definitions, or at least, definitions that can be transformed into efficient definitions.[4] In the current system, an attempt has been made to abstract principle definitions away from control issues such as the question of non-interleaved versus interleaved execution. The basic idea here being that it should be possible for the grammar writer to specify principles in a manner that is both neutral and transparent with respect to such issues. In the current system, the fact that types are computed when principles are to be interleaved is completely transparent to the grammar writer. (However, the grammar writer is free to explore different control options independently of the representation.) To achieve this, a type inference mechanism must have the following properties:

[3]For example, see Stabler's [11] account of a full 1st-order axiomatization of Chomsky's *Barriers* [1] framework.

[4]For examples of other logic-based proposals along these lines, see Johnson [6] and Stabler [12].

1. The type inference mechanism must be *automatic* in the sense that it should compute types solely from a principle definition without having to solicit extra information from, or be volunteered by, the user, e.g. in the form of type specifications as program annotations. This also implies that the mechanism must be capable of *degrading gracefully* in cases where there is insufficient information to determine a particular type. In the current system, no annotations are allowed.

2. The *off-line* interleaving process must preserve *soundness.* For example, as discussed earlier, type computation must never cause the parser to (erroneously) fail to impose a well-formedness condition on an ill-formed structure.

3. Finally, of course, the inference mechanism must always halt. As will be discussed in a subsequent section, this is a potential problem with the type relations to be defined, given the 'multiple-pass' nature of the implementation.

Although, type inference must be sound, it need not be *complete* (in a sense to be made clear below), in order to retain transparency. Suppose T is the smallest possible sound type for a configuration X, i.e. for every category c in T, there exists a constituent of category c that satisfies the conditions of X. Now, suppose a type T' computed for X by an inference algorithm is *strictly larger* than T in the sense that $T \subset T'$. Then, we say that the algorithm is incomplete (but still sound) since it has failed to determine the (minimal) type of X. To make the point clearer, we can say that the original naïve polling algorithm is equivalent to the revised system with a type inference algorithm that always returns the set of all possible category labels of the (linguistic) theory; that is, the algorithm is effectively failing to compute any meaningful type. (We will denote the set of all possible category labels, i.e. the *universal type*, by U.) The algorithm that we will describe in this paper will automatically substitute U for any part of the principle definition that a type inference fails to apply. As the section on inference rules will describe, this is usually not 'fatal'. In fact, this default substitution allows the inference algorithm to degrade gracefully by computing a strictly larger type but remaining sound.

1.3 ROADMAP

The remainder of this paper consists of three sections. To begin, we will review a typical theory in the principles-and-parameters framework. We will also take the opportunity to provide an example of how linguistic principles may be written using clauses in logic. Next, we will introduce the notion of a type of a principle, and define type inference rules that operate over primitive linguistic relations and logic programs. We will apply these rules to the example principle mentioned above to illustrate how the type computation is actually carried out. Finally, we will discuss the limitations and tradeoffs found in the implemented procedure.

2 THE LINGUISTICS FRAMEWORK

In this section, we review the components of a typical theory in principles-and-parameters framework. The goals of this section will be twofold: first, to introduce the

$$\begin{tikzcd}
 & \text{D-structure: } \bar{X}\text{-rules} \arrow[d, "\text{Move-}\alpha"] & \\
 & \text{S-structure} \arrow[dl, "\text{Phonological Rules}"'] \arrow[dr, "\text{LF Movement}"] & \\
\text{Phonetic Form} & & \text{Logical Form}
\end{tikzcd}$$

Figure 1 A standard model of phrase structure.

$\bar{\bar{X}} \rightarrow$ Spec(X), $\bar{X}$

e.g. X=*N*, Spec(*N*)=*NP* $[_{\bar{\bar{N}}}$[*NP the Vandals'*]$[_{\bar{N}}$ *destruction of the city*]]

X=*N*, Spec(*N*)=*Det* $[_{\bar{\bar{N}}}$[*Det the*]$[_{\bar{N}}$ *destruction of the city*]]

$\bar{X} \rightarrow$ X, Compl(X)

e.g. X=*N*, Compl(*N*)=*NP* $[_{\bar{N}}$ *destruction* [*NP of the city*]]

X=*V*, Compl(*V*)=*NP* $[_{\bar{V}}$ *destroyed* [*NP the city*]]

X=*A*, Compl(*A*)=*NP* $[_{\bar{A}}$ *proud* [*NP of the city*]]

X=*P*, Compl(*P*)=*NP* $[_{\bar{P}}$ *to* [*NP the city*]]

Figure 2 Examples of $\bar{X}$-rules.

notion of a linguistic principle, and secondly, to provide a non-trivial and linguistically-relevant example that will be used to illustrate type computation. (Readers familiar with the framework may wish to skip directly to the formalization of θ-role-assignment in section 2.3.)

A typical theory such as that of Chomsky [2] may be divided into two components. First, the phrase structure (PS) component consists of several levels of PS representation related by rules of movement as shown in figure 1. The levels of Phonetic Form and Logical Form form the interface of syntax to external systems that produce speech and interpret language, respectively. However, for the purpose of this paper, we will be primarily concerned with principles that apply to the other two (theory-internal) levels of PS representation, namely, D- and S-structure.

2.1 PHRASE STRUCTURE

The level of D-structure may be generated by simple CF rules including those shown in figure 2.[5] The two-bar system shown partitions phrases into three levels: a 'double-bar' level, or *maximal projection*, denoted by $\bar{\bar{X}}$(=XP), that corresponds to the traditional

[5]We have omitted the rules that are necessary to handle empty categories and adjunction structures at D-structure. For example, *NP* $\rightarrow \lambda$ is required to handle passive and parasitic gap constructions, as in [[*NP-e*][*VP was arrested John*]] and *you file which report without* [*NP-e*] *reading* [*NP-e*]. Also, non-$\bar{X}$-rules such as *VP* $\rightarrow$ *VP Adv* and *NP* $\rightarrow$ *NP CP* are required to handle *wh*-adverb questions, e.g. *John* [*VP*[*VP leave*][*Adv why*]], and relative clause constructions, e.g. [*NP*[*NP the man*][*CP who I saw*]], respectively.

notion of a complete phrase; a 'single-bar' level, or *intermediate projection*; and, finally, a 'zeroth-bar' level, or *head projection*, that corresponds to the level of individual lexical items. As the examples show, the $\overline{X}$-rule schema expresses both the uniformity of PS across the four lexical categories N,V,A and P, and the fact that heads, i.e. zero-bar-level constituents, largely determine the internal structure of a phrase. The *specifier* and *complement* positions, represented by Spec(X) and Compl(X), respectively, are the positions normally occupied by subjects and objects.[6] The bar-level system also extends to the two non-lexical categories, inflection (I) and complementizer (C). For example, IP represents the traditional clause (S), as in:

(1) [*IP*[*NP John*][$\overline{I}$ *saw Mary*]]

Here, the subject of the clause, *John*, is assumed to occupy the specifier position of IP. Similarly, CP represents the traditional full clause ($\overline{S}$), as in:

(2) [*CP*[*NP who*][$\overline{C}$ *did*[*IP John see*]]]

This last clause is actually an example of PS at the level of S-structure, not D-structure. Now, S-structure is derived from D-structure by the iterated, possibly null, application of the operation of Move-α that allows syntactic constituents to move freely. The idea here being that principles will rule out all cases of improper movement, leaving only those that are well-formed. (We will return to an example of such a principle presently.) For example, let us consider the sentence *who did John see* again. Here, we understand the object of the verb *see* to be somehow associated with the NP *who*. In the standard analysis, (2) is assumed to be derived from the D-structure representation:

(3) [*IP John*[*VP*[*V see*][*NP who*]]]

This establishes the connection between *who* and the object of *see*, i.e. "the person that John sees." The object *who*, then moves to the clause-initial position found in the S-structure (2). The object position vacated by *who* is filled at S-structure by a *trace* of movement. By convention, the trace and its antecedent will be *coindexed*, as shown below:

(4) [*CP*[*NP who*]$_i$[$\overline{C}$ *did*[*IP John see* [*NP-t*]$_i$]]]

2.2 LINGUISTIC PRINCIPLES

The other component of the theory is a set of linguistics principles that are conveniently grouped into *modules of grammar* such as Case, θ-, Control, Trace, and $\overline{X}$- theories, each of which deals with a different aspect of syntax. Let us now briefly review some of these modules. We have already seen the effects of some of these in the previous sections. The requirement that traces and antecedents be automatically coindexed falls under the province of Trace theory. Similarly, $\overline{X}$-theory governs the structure of the PS rule system

[6]For convenience, the order of the right side elements of the $\overline{X}$-rules will match the constituent order of the examples shown. Note however, that the actual constituent order will be irrelevant to the definitions used in the paper. That is, we will make use of the hierarchical, but not the precedence, information present in the PS representation.

outlined in the previous section. The principles of Case theory deal with the assignment and conditions on the proper distribution of *abstract* Case. (Case is abstract in the sense that it need not be morphologically realized.) We have already encountered an example *John saw *he/him*, in which assigned Case is morphologically realized. In general, Case is only assigned in certain configurations. For example, subjects normally only receive Case in tensed, but not untensed, clauses. For example, *John* receives Case in *John is sad*, but not in **John to be sad*. The ill-formedness of the latter example is explained by the Case filter, a principle of Case theory that requires all overt noun phrases to bear Case.

Case theory will also interact with Move-α. For example, consider the sentence *John seems to be sad*, an example of a 'raising' construction. Here, *John* is understood to be associated with the subject of *sad* in the same sense as *who* was associated with the object of *see* in (4) above. Hence, the underlying D-structure for the raising construction will be *[IP[NP-e][VP seems[IP[NP John][VP be sad]]]]*. Since the matrix clause is tensed, but not the clausal complement of *seems*, *John* must move to the matrix subject position presently occupied by [*NP*-e], an empty NP, in order not to violate the Case filter. (Note that the case where no movement takes place, i.e. **it seems John to be sad*, is ill-formed.) A similar situation occurs with 'passive' constructions such as *[NP John]$_i$ was arrested t_i*. Here, the noun phrase *John* will be assigned Case by virtue of occupying the subject position of a tensed clause. (The corresponding non-movement case **it was arrested John* is ill-formed because, unlike the corresponding active case, the object of a passive predicate does not receive Case.)

Movement also interacts with Binding theory, a module that deals with the coreference possibilities for anaphors, pronominals, and referential-expressions such as names. For example, Binding conditions tell us that the pronoun *he* in the sentence *who that John knows does he like* may be interpreted either as being *coreferential* with *John*, i.e. *he* and *John* 'denote' the same individual, or *free* in the sense that *he* is not coreferential with any person named in the sentence. These same conditions can also tell us that the former interpretation is unavailable for the similar sentence *he likes everyone that John knows*. That is, *he* must be free in this case. The critical difference between the two examples is one of movement. Although *John* is part of the object of *like* at D-structure in both cases; in the former case, the object *who that John knows* has been moved to a clause-initial position.

This completes the high-level description of some of the principles that a typical principle-based parser will implement. We now turn to a more detailed review of some of the principles of θ-theory. In particular, we will describe the conditions of θ-role-assignment in some detail to provide a example of how principles in the framework may be formalized in logic. The formal definition obtained here will also be used to illustrate the type construction algorithm in a later section.

2.3 FORMALIZING THE CONDITIONS OF θ-ROLE ASSIGNMENT

The module of θ-theory is concerned with the proper distribution of semantic, or *thematic*, roles. For example, consider the sentence:

(5) *the police arrested John*

Here, *the police* and *John* are considered to bear the thematic roles of 'agent' and 'patient', or 'theme', of the predicate *arrest*, respectively. In syntax, these relations are established by having a lexical head such as the verb *arrest* (which has an 'agent' and a 'theme' θ-role), assign its θ-roles to, i.e. *θ-mark*, the subject and object positions, respectively. Now, the fundamental principle of θ-theory is the θ-criterion, a two-part condition which ensures that θ-roles are properly discharged to *arguments*; roughly speaking, syntactic elements that have a referential function. Here is a typical definition of the θ-criterion (from Lasnik & Uriagereka [7]):

(6) θ-Criterion:
a. Every argument must be assigned a θ-role.
b. Every θ-role must be assigned to an argument.

For example, the θ-criterion rules out the following ill-formed examples:

(7) a. **the police arrested John Bill*
b. **the police arrested*
c. **there arrested John*
d. **there arrested*

In (7a), (6a) is violated since *Bill* is an argument without a θ-role. Here, the theme θ-role is assigned to *John*, and the agent role to *the police*. In (7b), no argument is available to bear the theme θ-role; and in (7c), the agent role is assigned to a non-argument, the pleonastic element *there*. In both cases, (6b) is violated. Finally, (7d) violates both sides of the θ-criterion.

The (Extended) Projection Principle requires that the θ-criterion must hold at all syntactic levels, i.e. at D-structure, S-structure and LF. For example, one consequence is that movement from a $\bar{\theta}$-position (a position to which no θ-role is assigned) to a θ-position is prohibited, as in **John$_i$ mentioned that t_i rains* — cf. *John mentioned that it rains* where no movement of a noun phrase has taken place.

To apply the θ-criterion, we have to first assign θ-roles. In general, *internal* θ-roles will be assigned to complement positions, e.g. the 'theme' θ-role of *arrest*, and *external* θ-roles will be assigned to subject positions, e.g. the agent θ-role of *arrest*. Following Chomsky [2], the possible cases of θ-marking are:

1. Lexical heads θ-mark their complements.

   We have already seen the case when the head is a verb. Nouns also θ-mark their complements. For example, consider figure 2 again. Here, the nominal form *destruction* assigns the same (internal) theme θ-role to *the city* in the configuration [[$\overline{N}$ *destruction*][*NP of the city*]] that *destroy* assigns in [[$\overline{V}$ *destroyed*][*NP the city*]]. Similarly, the adjective *proud* θ-marks the pronoun *he* in [$\overline{A}$ *proud*[*NP of him*]], and the preposition *to* assigns a destination θ-role in [$\overline{P}$ *to*[*NP the city*]]. Note that some heads such as the verbs *persuade* and *ask*, as in [$\overline{V}$ *persuaded*[*NP John*][*CP that he should leave*]] and [$\overline{V}$ *asked*[*NP John*][*CP to leave*]], may assign two or more internal roles. Others such as *sleep* take no objects and therefore have no internal roles to assign.

Let X and Y be constituents, and let F_X be the feature set associated with X.
Let L be an arbitrary list, F an instantiated feature, and c a category label.

Primitive Operations
`cat`(X,c) holds if c is the category label of X.
X `complement_of` Y holds if X occupies a complement position in phrase Y.
X `has_constituent` Y holds if Y is an immediate sub-constituent of X.
X `has_constituent_head` Y holds if Y is an immediate sub-constituent of and heads X.
X `has_constituents` L holds if L represents all the immediate sub-constituents of X.
X `has_feature` F holds if $F \in F_X$.
X `specifier_of` Y holds if X occupies the specifier position in phrase Y.

Universal Quantification Form
```
<Operation-Name> in_all_configurations <Variable>
      where <Pre-conditions-1> then <Conditions-1>
      else <Pre-conditions-2> then <Conditions-2>...
```

Figure 3 Linguistic primitives and a universal quantification form.

2. Verbs that possess an *external* θ-role will θ-mark their subject.

   For example, the verb *hit* will assign an 'agent' θ-role to *John* in *[IP[NP John][VP hit Bill]]*. Note that in some cases, a verb may combine with an internal predicate to θ-mark the subject. For example, in *John is crazy*, the argument *John* is indirectly θ-marked by the adjective *crazy.*

3. Nouns that possess an external θ-role (optionally) θ-mark the subject position of the noun phrase that they head.

   Consider the two noun phrases shown in (8) (taken from Chomsky [2]):

   (8) a. *Bill's fear of John*
   b. *the fear of John*

   In (8a), *Bill* receives same θ-role 'experiencer' that it receives in *Bill fears John*, but in (8b) the corresponding θ-role is not assigned.

In the following discussion, we will make use of the linguistically-motivated primitives shown in figure 3. Two kinds of primitives are listed here: some such as `complement_of` and `specifier_of` represent basic notions defined by $\overline{\text{X}}$-theory. Other primitives such as `has_constituent` and `has_feature` are general relations on PS representation. We will assume that all constituents have associated feature sets, and that features are inherited according to $\overline{\text{X}}$-projections, so that the features of a head with category X will be available at all other levels, $\overline{\text{X}}$ and $\overline{\overline{\text{X}}}$. Finally, we will also assume the existence of a universal quantification form `in_all_configurations` that may be used to express (arbitrary) conditions on phrase structures. (The semantics of this form will be explained presently.)

```
thRoleAssign in_all_configurations CF where
     thetaConfig(CF,Roles,Els) then assignRoles(Roles,Els).

thetaConfig(CF,[Role|Rs],Complements) :-% Case 1: [Xi X Complements]
     CF has_feature grid(_,[Role|Rs]),
     CF has_constituents L,
     CF has_constituent_head Head,
     pick(Head,L,Complements).              % List op: pick out Head from L
                                            % leaving Complements.
thetaConfig(IP,[Role],[NPSpec]) :-          % Case 2: [IP Spec [I1 I VP]]
     cat(IP,ip),
     VP complement_of IP,
     extRoleToAssign(VP,Role),
     NPSpec specifier_of IP.

thetaConfig(NP,Roles,Args) :-               % Case 3: [NP Spec [N1 N ..]]
     cat(NP,np),
     NP has_constituent NPSpec,
     cat(NPSpec,np),
     NP has_feature grid([Role],_),
     optionalThRole(Role,NPSpec,Roles,Args).

optionalThRole(Role,Arg,[Role],[Arg]).      % Commit to assigning the role.
optionalThRole(Role,Arg,[],[]).             % Omit assigning the role.
```

Figure 4 The definition of θ-role-assignment.

The configurations of θ-marking are summarized using the predicate `thetaConfig` shown in figure 4. (For convenience, we adopt standard PROLOG syntax for this and all subsequent definitions.) Here, `CF` is a θ-configuration with two components: (1) a list of the θ-roles to be assigned, and (2) a list of the constituents to receive the θ-roles. Note that the thematic representation of a head, the *θ-grid*, is encoded using a lexical feature `grid(L1,L2)`, where the lists `L1` and `L2` hold the external and internal θ-roles of the head, respectively. For example, *arrest* would have the feature `grid([agent],[theme])`.

The translation from each θ-configuration case described above to the formal definition in terms of the PS primitives provided is relatively straightforward. Let us briefly review the first `thetaConfig` clause. This clause is designed to extract the two components of a θ-configuration if `CF` is a configuration where a lexical head would θ-mark its complements. In general, `CF` need only be the smallest constituent containing both components.[7] This

[7]It is not strictly necessary to pick the smallest configuration; in fact, the definition could be written to accept any arbitrary element containing both components. But, the reason for preferring the smallest lies in the bottom-up nature of the algorithm that builds structure. Recall that, in the interleaved model, we would like to be able to apply principles as early as possible. By defining the θ-configuration to be

```
% NB: assignRoles/2 will be defined in terms of assignRole/2
assignRole(Role,Argument) :- Argument has_feature theta(Role).
```

Figure 5 Assigning a θ-role to an argument.

occurs when the head and its complements are both immediate constituents of CF. The identification of this condition is performed by the last three conjuncts of the clause. The first conjunct is there to make sure that the head has at least one θ-role to assign. For example, this would rule out [*VP*[*V sleeps*]] as a potential case of head–complement θ-marking. (Note that here we make use of the inheritance of features to examine the θ-grid associated with the head.) The definition of the other cases proceeds in a similar fashion. Finally, the operation of θ-role-assignment itself is formalized by universally quantifying over the configurations of phrase structure. The definition shown is meant to be read as follows:

> "In all configurations named by CF such that CF is a θ-configuration with components Roles and Els; then assign the θ-roles Roles to elements Els."

The actual assignment of θ-roles to constituents is carried out by assignRoles. Constituents will contain θ-slots that will be represented as a complex feature theta(S), where S will hold the actual θ-role. Hence, assignment simply boils down to unifying a θ-slot with a given θ-role, as shown in figure 5. (Here, to keep the definition short, we only illustrate the case for a single θ-role and constituent.[8])

The issue of the high overhead involved in repeated evaluation of the pre-conditions of a principle can be seen clearly in this definition. The cost of determining the θ-configurations is by far the most expensive part of applying the principle. As we have seen, the conditions under which θ-marking takes place are quite restricted. Hence, a large majority of constituents that are constructed should fail to satisfy the predicate thetaConfig. But, given that there are three clauses to test and that several conjuncts of the clauses may succeed in the course of testing, each failure can be quite expensive. By comparison, the amount of additional work required to actually assign the θ-roles, once the appropriate components of the configuration have been determined, is small.

the smallest possible constituent, we can apply θ-role-assignment as soon as the relevant components have been found.

[8]The implementation of assignRoles is a little more complicated for two reasons: (1) In general, θ-grid roles are unordered. For example, verbs such as *give* have two internal θ-roles that may permute freely in syntax, as in *gave John a book* and *gave a book to John*. (2) Also, θ-roles may have several syntactic realizations. For example, following Chomsky [2], *persuade* select a goal and a proposition as internal θ-roles. A proposition may be realized as either as a noun phrase or a clause, as in *persuaded* [*NP John*][*CP that he should leave*] or *persuaded* [*NP John*][*NP of the importance of leaving*]; but a goal may only be realized as a noun phrase. Note however, that although θ-roles are unordered, principles may apply to exclude ungrammatical permutations. For example, **persuaded* [*CP that he should leave*][*NP John*] is ruled out by the Case adjacency requirement which is present in English. That is, a noun phrase must be adjacent to its Case assigner in order to receive structural Case.

3 TYPE INFERENCE

In this section, we will formally define the notion of a type together with type equations for linguistic primitives and type inference rules over logic programs. Next, we will illustrate the process of type computation on the definition of θ-role-assignment introduced in the previous section. We will also take the opportunity to discuss the features and limitations of the implemented type computation algorithm.

3.1 TYPE DEFINITIONS

We begin by defining the possible category labels:

Definition 1 (Category labels)
Let L be the set of $\overline{\mathrm{X}}$-theory head category labels, i.e. $\{N, V, A, P, I, C\}$. Let L' be the set of non-$\overline{\mathrm{X}}$ category labels, e.g. $\{Adv, Det\}$. Now, let proj denote the $\overline{\mathrm{X}}$ "projects to" relation. That is, let c proj $\bar{c}$ and $\bar{c}$ proj $\bar{\bar{c}}$ hold, for $c \in L$. Then, let proj* be the reflexive transitive closure of proj. Finally, let V denote the (finite) set of all possible category labels, i.e. $\{c' | \ c \text{ proj}^* c', \text{for } c \in L\} \cup L'$.

We can now define the notion of a type in terms of category labels:

Definition 2 (Type)
A *type* is simply a member of 2^V. In particular, let U, the *universal type*, be the largest member of this set, i.e. V.

In a logic program, types will be associated with logical variables that represent constituents. We will use the expression $X : T$ to represent a constituent variable X with associated type T. We will now define some of the type equations associated with linguistically-motivated primitives such as those in figure 3.

Definition 3 (Category labelling)
Given a constituent X and a category label c:
$\texttt{cat}(X\!:T, c)$ holds $\Rightarrow$
  $T = \{c\}$ if c is a category label, or
  $T = \text{typeValue}(c)$ if c is uninstantiated.

For example, consider [Defn. 3] which defines the type of its first argument as follows:[9]

1. When c is instantiated to some label, say NP, as in "cat(X,NP)," we can immediately determine that X can only range over constituents with that label — for our example, we say that the type of X is {NP}.

2. On the other hand, c may be left uninstantiated: for example, we might write "cat(X,C), head(C)" to state that X must be a head. Here, the type of X cannot be determined solely from "cat(X,C)." In general, if 'typeValue(C)' is

[9]Note that these type definition rules are transparent to the user. However, if the grammar writer wishes to incorporate a new primitive relation involving constituents, then a corresponding type definition should be written — although one is not strictly necessary, that is, the inference algorithm will still function, albeit, less effectively.

returned, the inference algorithm will interpret it to mean that should make a second pass through the predicate in which cat(X,C) occurs to find other literals that involve C and compute all possible values of C where feasible.[10] In this case, **head** is a simple primitive of $\overline{\text{X}}$-theory that holds only for categories N, V, A, P, I, and C. Hence, the type of X will become {N,V,A,P,I,C}.

The lexicon can also play a part in type determination. Consider [Defn. 4] which applies to the feature membership primitive:

Definition 4 (Feature membership)
Given a constituent X and an instantiated feature F. Let E and F_E be a lexical entry and its feature set:
$X: T$ **has_feature** F holds $\Rightarrow$
$T = \{c \mid \exists\, E$ of category c' st. $F \in F_E, c'$ proj$^* c\}$
when F is a lexical feature, else
$T = \text{typeValue}(F)$ if F has a given type, otherwise
$T = U$.

For example, if only verbs and adjectives may be exceptional case markers (ECM), and "X **has_feature** ecm" holds, then we can infer (assuming feature inheritance) that X must have type (or be a sub-subset of) {V, $\overline{\text{V}}$, VP, A, $\overline{\text{A}}$, AP}. This is captured by the first type equation. The second equation applies to non-lexical features that are assigned independently of the lexicon, but have type definitions specified. Finally, the last equation introduces the universal type U. This applies as a 'catch-all' for all features not handled by the earlier type equations. (In fact, unless otherwise stated, type U will be the default type assigned whenever no type equation applies.)

Next, to conclude our examples of primitive relations, [Defn. 5] illustrates the notion of a type relation:

Definition 5 (Head relation)
Given two constituents X and Y:
$Y: T_Y$ **head_of** $X: T_X$ holds $\Rightarrow$
$T_X = \{c \mid c'$ proj$^* c$, for each $c' \in T_Y\}$,
$T_Y = \{c \mid \text{head}(c)$, c proj$^* c'$ for each $c' \in T_X\}$

head_of holds if the first constituent is the head of the second. In this case, two type equations, one for each argument, are defined in terms of the type of the other.

We now move on to examples of type composition rules for logic programs. In the following definitions, a *simple formula* F may be an atom. If F_1 and F_2 are simple formulas than so are: (\+F_1), (F_1,F_2), (F_1;F_2). For example, [Defn. 6] defines how types may be composed for conjunction ',':

[10]The conditions under which typeValue(C) causes partial evaluation of a literal are too detailed to cover here. In this case, the predicate **head** is defined by the clauses "**head(a) head(n) head(v)** ...**head(c)**." The algorithm detects that **head** is defined using ground unit clauses, from which it can collect values for C without going into an infinite loop.

Definition 6 (Conjunction) Given simple formulas F_1 and F_2:

$$\frac{X: T_1(F_1), \qquad X: T_2(F_2)}{X: T_1 \cap T_2(F_1, F_2)}$$

This inference rule has the following interpretation: "if X has type T_1 in F_1, and X also has type T_2 in F_2, then X has type T_1 intersect T_2 in the conjunction of F_1 and F_2." Although, strictly speaking, Horn clauses do not have disjunctions in the body of the clause, (see Lloyd [8]); it is both simple and convenient to allow the corresponding rule for disjuncts:

Definition 7 (Disjunction) Given goals F_1 and F_2:

$$\frac{X: T_1(F_1), \qquad X: T_2(F_2)}{X: T_1 \cup T_2(F_1 ; F_2)}$$

The type rule for predicate definition is quite similar to that for disjunction, except, of course, for the variable substitutions:

Definition 8 (Predicate Definition)
Let predicate P be defined by n clauses of the form $A_i\text{:-}F_i$, $\quad 1 \leq i \leq n$, where A_i and F_i are atoms and simple formulas, respectively. Let A be an atom and let σ_i be a most general unifying substitution for A_i wrt. A. Then:

$$\frac{\forall i \text{ st. } \exists \sigma_i \text{ st. } A = A_i\sigma_i \quad X: T_i(A_i\text{:-}F_i)\sigma_i}{X: \bigcup_i T_i(A)}$$

Finally, we conclude the series of type definitions with a brief discussion of negation \+ (as failure):

Definition 9 (Negation)

$$\frac{X: T(F)}{X: U(\backslash\!+F)}$$

Note that X has type U after negation, irrespective of its original type. This is a conservative rule which essentially provides no type information. Note however, that exceptions are defined for certain goals. For example,the primitive **has_feature** [Defn. 4] is an example where type information is not deleted across negation:

Definition 10 (Negation and feature membership)
Given a constituent X and an instantiated feature F. Let E and F_E be a lexical entry and its feature set:
+ $X: T$ **has_feature** F holds $\Rightarrow$

$T = \{c \mid \exists E$ of category c' st. $F \notin F_E$, c' proj$^*c\}$ if F is a lexical feature, else
$T = U - T'$ where $T' =$ typeValue(F) if F has given type, otherwise
$T = U$.

3.2 TYPE COMPUTATION

In this section, we will present an example of type computation using θ-role-assignment, as defined in section 2.3. Let us consider again the clauses for `thetaConfig` shown previously in figure 4.

1. The first clause encodes the case of head–complement θ-marking. For this to occur, the head must have a θ-grid with one or more internal θ-roles. This is encoded by the first conjunct "`CF has_feature grid(_, [Role|Rs])`." The type inference algorithm will consult the type rule [Defn. 4]. According to this rule the algorithm must consult the lexicon to find all categories c' that have one (or more) lexical entries with a θ-grid containing one or more internal arguments. We have that such entries exist for categories `N`, `V`, `A` and `P` only: for instance, see the examples shown on page 8. Then, the type of `CF` is determined as the set of labels c such that c is zero or more projections of c'. Hence, `CF` has type constrained by $\{N,\overline{N},NP,V,\overline{V},VP, A,\overline{A},AP,P,\overline{P},PP\}$.

   Not all the primitives have associated type equations. For example, there is no equation for the second conjunct "`CF has_constituents L`." The list operation used in the fourth conjunct `pick` is a generic predicate that has no special linguistic significance. Hence, `CF` is given type U in both cases.

   However, the third conjunct, namely "`CF has_constituent_head Head`," specifies that `CF` immediate dominates its head; in other words, `CF` must be a single-bar category. Hence, `CF` has type upper-bounded by $\{\overline{N}, \overline{V}, \overline{A}, \overline{P}, \overline{I}, \overline{C}\}$.

   Using the type rule for conjuncts, we intersect each type instance to obtain $\{\overline{N}, \overline{V}, \overline{A}, \overline{P}\}$.

2. The second and third clauses encode two cases of external θ-role assignment: a noun assigning its external θ-role to its NP specifier, and the subject of IP being assigned an external θ-role, either directly or indirectly, via the VP predicate. In both cases, by applying the type rule for category labelling [Defn. 3], `CF` will be immediately restricted to a singleton set, $\{NP\}$ in the former and $\{IP\}$ in the latter case. Note that the algorithm will still try to explore the other conjuncts present in both clauses.[11]

Finally, using the type rule for predicate definition [Defn. 8] together with the precondition in `assignThRoles`, we can infer that the type of `thConfig` is the union of the types computed for each of the three clauses: that is, $\{\overline{N}, NP, \overline{V}, \overline{A}, \overline{P}, IP\}$. Note that this type is obviously much smaller than the universal type. Although not all structures that have its category labelled as one of these type components, e.g. $[_{\overline{V}}[_{V}\ \textit{sleep}]]$ is not a θ-configuration as mentioned before, we have safely and automatically eliminated much of the unnecessary overhead incurred by the naïve polling model.

Let us extend this example one step further to illustrate the capability of the inference procedure to trace a number of different type variables. In general, the evaluation of

[11] Incidentally, if the empty type for a definition is ever returned, then that definition can never succeed. Although, we have argued for the transparency of the type inference procedure, this is perhaps one case in which debugging information can and should be conveyed to the grammar writer.

```
inCaseAssign in_all_configurations CF
   where inCaseConfig(CF,Case,Items)
        then assignInherentCase(Case,Items).

inCaseConfig(CF,Case,Items) :-
      thetaConfig(CF,_,Items),
      Head head_of CF,
      cat(Head,C),
      inherentCaseAssigner(C,Case).

inherentCaseAssigner(n,gen).
inherentCaseAssigner(a,gen).
inherentCaseAssigner(p,obq).
```

Figure 6 A definition of inherent Case assignment.

type relations may require the algorithm to make 'multiple-passes' through principle definitions. We have already seen one example of this, namely "`cat`(X,C)`, head`(C)," in which, the computation of the type of X will require the computation of the possible values for the type value C. In the current implementation, this is done by making a separate pass for each such unknown. A more sophisticated example can be found in the fragment shown in figure 6. This example is taken from the definition of inherent Case assignment, an assignment operation that operates on a proper subset of θ-configurations. The additional restriction here being that the head must be an inherent Case assigner.

By using the type rule for the `head_of` primitive [Defn. 5], the algorithm can infer that the type of `CF` is related to the type of `Head`. Therefore, by switching to computing the type of `Head`, the algorithm may be able impose further constraints on the type of `CF`. By using rule [Defn. 3] on the atom "`cat(Head,C)`," we can infer that the type of `Head` is given by the set of possible values for `C`. Switching once more to compute the values of `C`, we can follow the definition of `inherentCaseAssigner` to obtain `C` = {`N,A,P`}. By plugging these values back into the type equation for `head_of`, `CF` is constrained to be { N,$\overline{\text{N}}$,NP,A,$\overline{\text{A}}$,AP,P,$\overline{\text{P}}$,PP}. Finally, we know that `CF` has type {$\overline{\text{N}}$, NP, $\overline{\text{V}}$, $\overline{\text{A}}$, $\overline{\text{P}}$, IP} in `thetaConfig`, so by type intersection, we have that inherent Case assignment has type {$\overline{\text{N}}$, NP, $\overline{\text{A}}$,$\overline{\text{P}}$}.

Although, type relations allow the algorithm to infer the type of one variable in terms of another, as illustrated by in the preceding example; unfortunately, this also allows *circularity* in a chain of type inferences to occur. Perhaps, the simplest example occurs in the definition of `head_of` where the type of the first argument is defined in terms of the second, and vice-versa. Currently, the adopted solution is to keep track of the names of the variables being traced, and to terminate the search as soon as a previously encountered variable has been detected. Then, type computation will be resumed with the universal type substituted for all outstanding type variables. This system can be extended to cope with looping caused by recursive predicates by treating a recursive call as if the system switched to computing a new type variable. For example, this will prevent the algorithm from looping on `p(...,X,...) :- p(...,X,...)`, where X is the constituent variable being traced. This rudimentary loop detection scheme seems to be sufficient to guarantee termination, but at the cost of generality.[12]

4 CONCLUSIONS

We have described a procedure that takes logical definitions of linguistic principles and extracts categorial information that can be used to efficiently guide the interleaved model of parsing. The procedure relies on type definitions being present for the various linguistically-motivated primitives. An implementation of the procedure has been tested and found to work well on the twenty to thirty principles that are used in the PP-PARSER (described in Fong [5]). Automatic type inference is practical in this context because the principle definitions make considerable use of the linguistic primitives for which type definitions exist. In general, we can probably rely on the grammar writer to make use of the supplied primitives. Hence, the type computation process usually has access to enough type information so that the universal type may be avoided. Moreover, since the 'grammar programming language' in this case is fairly simple, we have not needed to deal with the complexities introduced, say, by data type polymorphism or coercion — features that exist in general programming languages, e.g. see Milner [10]. These two factors conspire to make the type computation problem simple enough to be practical for use in principle-based parsers.

References

[1] Chomsky, N.A. *"Barriers."* M.I.T. Press. 1986.

[2] Chomsky, N.A. *"Knowledge of Language: Its Nature, Origin, and Use."* Prager. 1986.

[3] Crocker, M.W. *"A Principle-Based System for Syntactic Analysis," (m.s.)* 1989.

[4] Fong, S. & R.C. Berwick "The Computational Implementation of Principle-Based Parsers," *International Workshop on Parsing Technologies.* Carnegie Mellon University. 1989.

[5] Fong, S. *"Computational Properties of Principle-Based Grammatical Theories,"* Ph.D thesis. Dept. of Electrical Engineering and Computer Science, M.I.T. *(forthcoming).*

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

[12] It is easy to construct an example to show the limitations of the loop detection scheme. For example, consider the following (artificial) definition of a principle p: `p(0,X) :- cat(X,np). p(1,X) :- p(0,X).` The loop detection algorithm will prematurely abort the type computation process in the case of the goal `p(1,`X`)`.

[7] Lasnik, H. & J. Uriagereka *A Course in GB Syntax: Lectures on Binding and Empty Categories*. 1988. M.I.T. Press.

[8] Lloyd, J.W. *"Foundations of Logic Programming,"* 2nd Ed., Springer Verlag. 1987.

[9] Macías, B. *"Government-Binding Theory and Parsing as Deduction." (m.s.)* 1989.

[10] Milner, R. "A Theory of Type Polymorphism in Programming," *Journal of Computer and System Sciences*. 17, 348–375. 1978.

[11] Stabler, E.P., Jr. *"The Logical Approach to Syntax: Foundations, Specifications and Implementations of Theories of Government and Binding." (m.s.)* M.I.T. Press. 1989.

[12] Stabler, E.P., Jr. *"Avoid the Pedestrian's Paradox," (m.s.)* 1990.