INCORPORATING INHERITANCE AND FEATURE STRUCTURES INTO A LOGIC GRAMMAR FORMALISM

Harry H. Porter, III
Oregon Graduate Center
19600 N.W. Von Neumann Dr.
Beaverton Oregon 97006-1999

ABSTRACT

Hassan Ait-Kaci introduced the ψ-term, an informational structure resembling feature-based functional structures but which also includes taxonomic inheritance (Ait-Kaci, 1984). We describe ψ-terms and how they have been incorporated into the Logic Grammar formalism. The result, which we call Inheritance Grammar, is a proper superset of DCG and includes many features of PATR-II. Its taxonomic reasoning facilitates semantic type-class reasoning during grammatical analysis.

INTRODUCTION

The Inheritance Grammar (IG) formalism is an extension of Hassan Ait-Kaci's work on ψ-terms (Ait-Kaci, 1984; Ait-Kaci and Nasr, 1986). A ψ-term is an informational structure similar to both the feature structure of PATR-II (Shieber, 1985; Shieber, et al, 1986) and the first-order term of logic. ψ-terms are ordered by subsumption and form a lattice in which unification of ψ-terms amounts to *greatest lower bounds* (GLB, $\sqcap$). In Inheritance Grammar, ψ-terms are incorporated into a computational paradigm similar to the Definite Clause Grammar (DCG) formalism (Pereira and Warren, 1980). Unlike feature structures and first-order terms, the atomic symbols of ψ-terms are ordered in an IS-A taxonomy, a distinction that is useful in performing semantic type-class reasoning during grammatical analysis. We begin by discussing this ordering.

THE IS-A RELATION AMONG FEATURE VALUES

Like other grammar formalisms using feature-based functional structures, we will assume a fixed set of *symbols* called the *signature*. These symbols are atomic values used to represent lexical, syntactic and semantic categories and other feature values. In many formalisms (e.g. DCG and PATR-II), equality is the only operation for symbols; in IG symbols are related in an IS-A hierarchy. These relationships are indicated in the grammar using statements such as[1]:

```
boy < masculineObject.
girl < feminineObject.
man < masculineObject.
woman < feminineObject.
{boy, girl} < child.
{man, woman} < adult.
{child, adult} < human.
```

The symbol < can be read as "is a" and the notation $\{a_1, \cdots, a_n\} < b$ is an abbreviation for $a_1 < b, \cdots, a_n < b$. The grammar writer need not distinguish between instances and classes, or between syntactic and semantic categories when the hierarchy is specified. Such distinctions are only determined by how the symbols are used in the grammar. Note that this example ordering exhibits multiple inheritance: `feminineObjects` are not necessarily `humans` and `humans` are not necessarily `feminineObjects`, yet a `girl` is both a `human` and a `feminineObject`.

Computation of LUB ($\sqcup$) and GLB ($\sqcap$) in arbitrary partial orders is problematic. In IG, the grammar writer specifies an arbitrary ordering which the rule execution system automatically embeds in a lattice by the addition of newly created symbols (Maier, 1980).

Symbols may be thought of as standing for conceptual sets or semantic types and the IS-A relationship can be thought of as set

[1] Symbols appearing in the grammar but not in the

inclusion. Finding the GLB–i.e. *unification* of symbols–then amounts to set intersection. For the partial order specified above, two new symbols are automatically added, representing semantic categories implied by the IS-A statements, i.e. human females and human males. The first new category (human females) can be thought of as the intersection of `human` and `feminineObject` or as the union of `girl` and `woman`[2], and similarly for human males. The signature resulting from the IS-A statements is shown in Figure 1.

ψ-TERMS AS FEATURE STRUCTURES

Much work in computational linguistics is focussed around the application of unification to an informational structure that maps attribute names (also called feature names, slot names, or labels) to values (Kay, 1984a; Kay, 1984b; Shieber, 1985; Shieber, et al, 1986). A value is either atomic or (recursively) another such mapping. These mappings are called by various names: feature structures, functional structures, f-structures, and feature matrices. The feature structures of PATR-II are most easily understood by viewing them as directed, acyclic graphs (DAGs) whose arcs are annotated with feature labels and whose leaves are annotated with atomic feature values (Shieber, 1985).

IS-A statements are taken to be unrelated.

[2] Or anything in between. One is the most liberal interpretation, the other the most conservative. The signature could be extended by adding both classes, and any number in between.

IGs use ψ-terms, an informational structure that is best described as a rooted, possibly cyclic, directed graph. Each node (both leaf and interior) is annotated with a symbol from the signature. Each arc of the graph is labelled with a *feature label* (an *attribute*). The set of feature labels is unordered and is distinct from the signature. The formal definition of ψ-terms, given in set theoretic terms, is complicated in several ways beyond the scope of this presentation–see the definition of well-formed types in (Ait-Kaci, 1984). We give several examples to give the flavor of ψ-terms.

Feature structures are often represented using a bracketed matrix notation, in addition to the DAG notation. ψ-terms, on the other hand, are represented using a textual notation similar to that of first-order terms. The syntax of the textual representation is given by the following extended BNF grammar[3].

```
term         ::=    symbol [ featureList ]
                 |  featureList
featureList  ::=    ( feature , feature ,
                       ... , feature )
feature      ::=    label ⇒ term
                 |  label ⇒ variable [ : term ]
```

Our first example contains the symbols `np`, `singular`, and `third`. The label of

[3] The vertical bar separates alternate constituents, brackets enclose optional constituents, and ellipses are used (loosely) to indicate repetition. The characters ( ) ⇒ , and : are terminals.

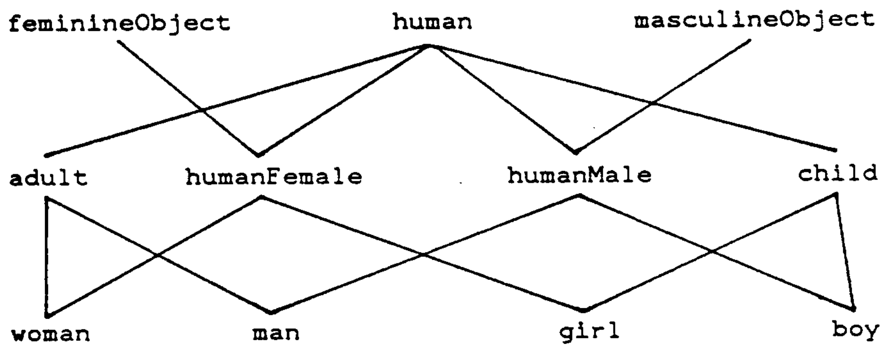

Figure 1. A signature.

the root node, np, is called the *head* symbol. This ψ-term contains two features, labelled by number and person.

```
np ( number ⇒ singular,
     person ⇒ third)
```

The next example includes a subterm at agreement⇒:

```
(cat       ⇒ np,
 agreement ⇒ (number ⇒ singular,
              person ⇒ third))
```

In this ψ-term the head symbol is missing, as is the head symbol of the subterm. When a symbol is missing, the most general symbol of the signature ($\top$) is implied.

In traditional first-order terms, a variable serves two purposes. First, as a wild card, it serves as a place holder which will match any term. Second, as a tag, one variable can constrain several positions in the term to be filled by the same structure. In ψ-terms, the wild card function is filled by the maximal symbol of the signature ($\top$) which will match any ψ-term during unification. Variables are used exclusively for the tagging function to indicate ψ-term *coreference*. By convention, variables always begin with an uppercase letter while symbols and labels begin with lowercase letters and digits.

In the following ψ-term, representing *The man wants to dance with Mary,* X is a variable used to identify the subject of *wants* with the subject of *dance.*

```
sentence (
     subject   ⇒ X: man,
     predicate ⇒    wants,
     verbComp  ⇒    clause (
          subject   ⇒ X,
          predicate ⇒ dance,
          object    ⇒ mary ))
```

If a variable X appears in a term tagging a subterm t, then all subterms tagged by other occurrences of X must be consistent with (i.e. unify with) t[4]. If a variable appears without a subterm following it, the term consisting of simply the *top* symbol ($\top$) is assumed. The constraint implied by variable coreference is not just equality of structure but equality of reference. Further unifications that add information to one sub-structure will necessarily add it to the other. Thus, in this example, X constrains the terms appearing at the paths subject⇒ and verbComp⇒subject⇒ to be the same term.

In the ψ-term representation of the sentence *The man with the toupee sneezed,* shown below, the np filling the subject role, X, has two attributes. One is a qualifier filled by a relativeClause whose subject is X itself.

```
sentence (
   subject   ⇒ X: np (
      head      ⇒ man,
      qualifier ⇒ relativeClause (
         subject   ⇒ X,
         predicate ⇒ wear,
         object    ⇒ toupee)),
   predicate ⇒ sneezed)
```

As the graphical representation (in Figure 2) of this term clearly shows, this ψ-term is cyclic.

UNIFICATION OF ψ-TERMS

The unification of two ψ-terms is similar to the unification of two feature structures in PATR-II or two first-order terms in logic. Unification of two terms t_1 and t_2 proceeds as follows. First, the head symbols of t_1 and t_2 are unified. That is, the GLB of the two symbols in the signature lattice becomes the head symbol of the result. Second, the subterms of t_1 and t_2 are unified. When t_1 and t_2 both contain the feature f, the corresponding subterms are unified and added as feature f of the result. If one term, say t_1, contains feature f and the other term does not, then the result will contain feature f with the value from t_1. This is the same result that would obtain if t_2 contained feature f with value $\top$. Finally, the subterm

[4] Normally, the subterm at X will be written following the first occurrence of X and all other occurrences of X will not include subterms.

coreference constraints implied by the variables in t_1 and t_2 are respected. That is, the result is the least constrained ψ-term such that if two paths (addresses) in t_1 (or t_2) are tagged by the same variable (i.e. they *corefer*) then they will corefer in the result.

For example, when the ψ-term

```
(agreement ⇒ X:(number⇒singular),
 subject   ⇒    (agreement⇒X))
```

is unified with

```
(subject⇒
     (agreement⇒
          (person⇒third)))
```

the result is

```
(agreement ⇒ X:(number⇒singular,
                person⇒third),
 subject   ⇒ (agreement⇒X))
```

INHERITANCE GRAMMARS

An IG consists of several IS-A statements and several *grammar rules*. A grammar rule is a definite clause which uses ψ-terms in place of the first-order literals used in first-order logic programming[5]. Much of the notation of Prolog and DCGs is used. In particular, the `:-` symbol separates a rule head from the ψ-terms comprising the rule body. Analogously to Prolog, list notation (using `[`, `|`, and `]`) can be used as a shorthand for ψ-terms representing lists and containing `head` and `tail` features. When the `-->` symbol is used instead of `:-`, the rule is treated as a context-free grammar rule and the interpreter automatically appends two additional arguments (`start` and `end`) to facilitate parsing. The final syntactic sugar allows feature labels to be elided; sequentially numbered numeric labels are automatically supplied.

Our first simple Inheritance Grammar consists of the rules:

```
sent --> noun(Num),verb(Num).
noun(plural) --> [cats].
verb(plural) --> [meow].
```

The sentence to be parsed is supplied as a goal

[5] This is to be contrasted with LOGIN, in which ψ-

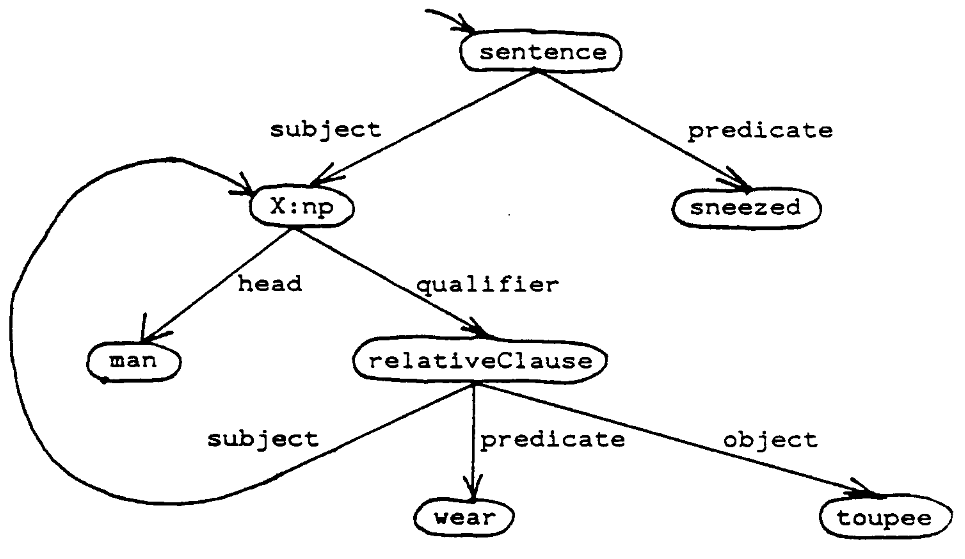

Figure 2. Graphical representation of a ψ-term.

clause, as in:

```
:- sent([cats,meow],[]).
```

The interpreter first translates these clauses into the following equivalent IG clauses, expanding away the notational sugar, before execution begins.

```
sent(start⇒P1,end⇒P3) :-
     noun(1⇒Num,start⇒P1,end⇒P2),
     verb(1⇒Num,start⇒P2,end⇒P3).
noun(1⇒plural,
     start⇒list(head⇒cats,tail⇒L),
     end⇒L).
verb(1⇒plural,
     start⇒list(head⇒meow,tail⇒L),
     end⇒L).
:- sent(start⇒list(
                    head⇒cats,
                    tail⇒list(
                              head⇒meow,
                              tail⇒nil)),
          end⇒nil).
```

As this example indicates, every DCG is an Inheritance Grammar. However, since the arguments may be arbitrary ψ-terms, IG can also accomodate feature structure manipulation.

TYPE-CLASS REASONING IN PARSING

Several logic-based grammars have used semantic categorization of verb arguments to disambiguate word senses and fill case slots (e.g. Dahl, 1979; Dahl, 1981; McCord, 1980). The primary motivation for using ψ-terms for grammatical analysis is to facilitate such semantic type-class reasoning during the parsing stage.

As an example, the DCG presented in (McCord, 1980) uses unification to do taxonomic reasoning. Two types unify iff one is a subtype of the other; the result is the most specific type. For example, if the first-order term `smith:_` representing an untyped individual[6], is unified with the type expression `X:person:student`, representing the student subtype of person, the result is `smith:person:student`.

terms replace first-order terms rather than predications.

[6] Here the colon is used as a right-associative infix operator meaning *subtype*.

While this grammar achieves extensive coverage, we perceive two shortcomings to the approach. (1) The semantic hierarchy is somewhat inflexible because it is distributed throughout the lexicon, rather than being maintained separately. (2) Multiple Inheritance is not accommodated (although see McCord, 1985). In IG, the ψ-term `student` can act as a typed variable and unifies with the ψ-term `smith` (yielding `smith`) assuming the presence of IS-A statements such as:

```
student < person.
{smith, jones, brown} < student.
```

The taxonomy is specified separately—even with the potential of dynamic modification—and multiple inheritance is accommodated naturally.

OTHER GRAMMATICAL APPLICATIONS OF TAXONOMIC REASONING

The taxonomic reasoning mechanism of IG has applications in lexical and syntactic categorization as well as in semantic type-class reasoning. As an illustration which uses ψ-term predications, consider the problem of writing a grammar that accepts a prepositional phrase or a relative clause after a noun phrase but only accepts a prepositional phrase after the verb phrase. So *The flower under the tree wilted, The flower that was under the tree wilted,* and *John ate under the tree* should be accepted but not **John ate that was under the tree. The taxonomy specifies that* `prepositionalPhrase` and `relativeClause` are `npModifiers` but only a `prepositionalPhrase` is a `vpModifier`. The following highly abbreviated IG shows one simple solution:

```
{prepositionalPhrase,
   relativeClause} < npModifier.
prepositionalPhrase < vpModifier.

sent(...) --> np(...),
               vp(...),
               vpModifier(...).
np(...) --> np(...),
            npModifier(...).
np(...) --> ...
vp(...) --> ...
prepositionalPhrase(...) --> ...
```

```
relativeClause(...) --> ...
```

IMPLEMENTATION

We have implemented an IG development environment in Smalltalk on the Tektronix 4406. The IS-A statements are handled by an ordering package which dynamically performs the lattice extension and which allows interactive display of the ordering. Many of the techniques used in standard depth-first Prolog execution have been carried over to IG execution. To speed grammar execution, our system precompiles the grammar rules. To speed grammar development, incremental compilation allows individual rules to be compiled when modified. We are currently developing a large grammar using this environment.

As in Prolog, top-down evaluation is not *complete*. Earley Deduction (Pereira and Warren, 1980; Porter, 1986), a sound and complete evaluation strategy for Logic programs, frees the writer of DCGs from the worry of infinite left-recursion. Earley Deduction is essentially a generalized form of chart parsing (Kaplan, 1973; Winograd, 1983), applicable to DCGs. We are investigating the application of alternative execution strategies, such as Earley Deduction and Extension Tables (Dietrich and Warren, 1986) to the execution of IGs.

ACKNOWLEDGEMENTS

Valuable interactions with the following people are gratefully acknowledged: Hassan Ait-Kaci, David Maier, David S. Warren, Fernando Pereira, and Lauri Karttunen.

REFERENCES

Ait-Kaci, Hassan. 1984. A Lattice Theoretic Approach to Computation Based on a Calculus of Partially Ordered Type Structures, Ph.D. Dissertation, University of Pennsylvannia, Philadelphia, PA.

Ait-Kaci, Hassan and Nasr, Roger. 1986. LOGIN: A Logic Programming Language with Built-in Inheritance, *Journal of Logic Programming*, 3(3):185-216.

Dahl, Veronica. 1979. Logical Design of Deductive NL Consultable Data Bases, *Proc. 5th Intl. Conf. on Very Large Data Bases*, Rio de Janeiro.

Dahl, Veronica. 1981. Translating Spanish into Logic through Logic, *Am. Journal of Comp. Linguistics*, 7(3):149-164.

Dietrich, Susan Wagner and Warren, David S. 1986. Extension Tables: Memo Relations in Logic Programming, Technical Report 86/18, C.S. Dept., SUNY, Stony Brook, New York.

Kaplan, Ronald. 1973. A General Syntactic Processor, in: Randall Rustin, Ed., *Natural Language Processing*, Algorithmics Press, New York, NY.

Kay, Martin. 1984a. Functional Unification Grammar: A Formalism for Machine Translation, *Proc. 22nd Ann. Meeting of the Assoc. for Computational Linguistics* (COLING), Stanford University, Palo Alto, CA.

Kay, Martin. 1984b. Unification in Grammar, *Natural Lang. Understanding and Logic Programming Conf. Proceedings*, IRISA-INRIA, Rennes, France.

Maier, David. 1980. DAGs as Lattices: Extended Abstract, Unpublished manuscript.

McCord, Michael C. 1980. Using Slots and Modifiers in Logic Grammars for Natural Language, *Artificial Intelligence*, 18(3):327-368.

McCord, Michael C. 1985. Modular Logic Grammars, *Proc. of the 23rd ACL Conference*, Chicago, IL.

Pereira, F.C.N. and Warren, D.H.D. 1980. Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks, *Artificial Intelligence*, 13:231-278.

Pereira, F.C.N. and Warren, D.H.D. 1983. Parsing as Deduction, *21st Annual Meeting of the Assoc. for Computational Linguistics*, Boston, MA.

Porter, Harry H. 1986. Earley Deduction, Technical Report CS/E-86-002, Oregon Graduate Center, Beaverton, OR.

Shieber, Stuart M. 1985. An Introduction to Unification-Based Approaches to Grammar, Tutorial Session Notes, *23rd Annual Meeting of the Assoc. for Computational Linguistics*, Chicago, IL.

Shieber, S.M., Pereira, F.C.N., Karttunen, L. and Kay, M. 1986. *A Compilation of Papers on Unification-Based Grammar Formalisms, Parts I and II,* Center for the Study of Language and Information, Stanford.

Winograd, Terry. 1983. *Language as a Cognitive Process, Vol. 1: Syntax,* Addison-Wesley, Reading, MA.