Attribute and unification grammar: A review and analysis of formalisms

Nelson Correa
Department of Electrical Engineering, Universidad de los Andes, Bogotá, D.E., Colombia

Abstract

Attribute and unification grammar are syntax-directed grammatical formalisms that bear an important resemblance to each other and, especially in the case of unification grammar, enjoy certain currency as tools for linguistic description. A systematic study and comparison of the two formalisms, however, has not been reported to date. This paper reviews and compares the two formalisms, both from the point of view of their notations and expressive capabilities, and from that of their underlying algebraic semantics. The key to the semantic comparison is an algebraic formulation of the semantics of unification and the algebraic semantics of attribute grammar developed by Chirica and Martin [6]. The main result is that, from the point of view of their definitions and semantics, attribute grammar is more general than unification grammar, since it rests on a richer semantic algebra. This greater generality has important implications regarding the expressive power of the formalism and the possibility of efficient computational implementations for it. Also regarding their semantics, it is revealed that attribute grammar, unlike unification grammar, does not restrict attribute values to their basic term interpretation. This is a double-edged sword which, on the one hand, makes attribute grammar expressive and computationally efficient and, on the other, makes it computationally difficult to implement, because of the so-called "attribute evaluation problem". For unification grammar, the noted restriction on term interpretations is at the heart of the difficulties encountered in implementing such linguistically motivated extensions to the basic formalism as negation and disjunction. The conclusion of this study is that attribute grammar is a better suited and more highly developed grammatical formalism for the description of natural and artificial languages, due to its greater generality, expressive power and, also importantly, more efficient computational implementations.

1. Introduction

Attribute grammar [29] is an elegant formalization of the augmented context-free grammars found in most current natural language systems. It bears an important resemblance to the attribute-value of "complex-feature-based" approaches to language which rely on unification in their underlying interpretation [28,33]. While in attribute grammar (AG) attribute values and conditions are defined declaratively by a set of attribution rules and conditions associated with each production in the grammar, and attribute evaluation is heavily dependent on the modes of the attributes and the data dependencies introduced by the rules used to define their values, in unification-based formalisms (UG), unification of label-value structures serves the dual role of defining attribute values and conditions, and attribute evaluation amounts to the solution of a system of equations on the attribute-value structures, imposed by the particular derivation of the input string. Both formalisms (AG and UG), however, have in common that attribute values and conditions are defined in a syntax-directed manner, on a production-by-production basis.

This paper reviews and compares attribute and unification grammar, taking as a point of departure a study of their underlying algebraic semantics, and then moving on to an analysis and comparison of their expressive capabilities, computational complexity, and implementability issues. This study is of interest since attribute grammar has been shown to be a viable formalism for the computational study of natural language [12,13] and, on the other hand, unification grammar is, with various extensions, the formalism underlying other current theories of language, including Lexical-Functional Grammar LFG [2], Generalized Phrase Structure Grammar GPSG [17], and Head-Driven Phrase Structure Grammar HPSG [32]. The key to the semantic comparison is the now standard algebraic approach to unification theory [34] and the formulation, within the framework of initial algebra semantics, of attribute grammar developed by Chirica and Martin [6]. The main result is that, from the point of view of their definitions and semantics, attribute grammar is more general than unification grammar since it rests on a richer semantic algebra. This greater generality has important implications regarding the expressive power of the formalism and the possibility of efficient computational implementations for it.

From the point of view of their notations and expressive capabilities, we will see that attribute grammar, unlike unification grammar, does not restrict the interpretation of attribute values to their basic term interpretation. This is a double-edged sword which, on the one hand makes AG expressive and computationally efficient, and on the other makes it computationally difficult to implement, because of the so-called "attribute evaluation problem". For unification grammar, the basic term interpretation imposed on feature structures makes the formalism simple to implement, since it is a virtue of unification that it operates on partially instantiated structures, and hence the data dependencies imposed by rules on attribute values may be ignored. However, this simplicity of implementation cannot be maintained without incurring a high computational cost once we consider adding to the basic formalism such linguistically motivated extensions as negation and disjunction, or such important facilities in any computer language for the representation of knowledge and information as computable expressions. Thus, from a practical point of view we show that AG is more computationally efficient and expressive than UG, since it does not rely on unification as the sole operation for attribute evaluation. The key to this higher efficiency is the more "low-level" nature of AG, in which attribute dependencies are observed and revealed explicitly in the grammar, and in which such dependencies are taken into account by the implementation for attribute evaluation.

While we will not directly address it in this paper, we will look at the problem of attribute evaluation in attribute and unification grammar and show that the simplicity of unification approaches to the problem is not without a high cost. Attribute evaluation is one of the most difficult problems of AG. While the problem has been solved for certain subclasses of AG with syntactically unambiguous grammars [25], a general solution for ambiguous grammars (e.g. for natural languages) has only recently begun to be worked out. Current work by the author [14] and by Yellin [42] addresses the problem of ambiguous underlying syntax by an extension of Earley's algorithm or generalized attributed parsing. Tomita [39] also presents an efficient parsing algorithm for augmented context-free grammars, based on a generalization of LR parsing techniques. The extension of Correa [14] does on-line attribute evaluation, allows for ambiguous and even cyclic bases, and is sufficient for S- and L-attributed grammars. Yellin's [42] technique requires a cycle-free base and allows any (well-defined) attributions, but it handles the attribute evaluation off-line, in up to two phases after syntactic analysis. Tomita's [39] generalized LR parser also requires a cycle-free base and handles grammars with synthesized attributes only. In the algorithm proposed by Correa, unification could be used to generalize the algorithm beyond the class of L-attributed grammars and may play an important role in the late stages of attribute evaluation.

In contrast, unification-based formalisms rely on unification as the sole operation for attribute evaluation, ignoring the question of data dependencies between attribute values. Their reliance on unification alone represents one of the major drawbacks of systems for UG, due to the high computational cost of unification and the difficulty of extending the basic framework, as noted before. Techniques already developed for efficient attribute evaluation in attribute grammars, such as an analysis of data dependencies and the use of pre-specified tree traversal or attribute evaluation methods, can be applied to the design and construction of more efficient and general UG systems.

In addition to their formal properties, below, in sections 4 and 5, we will also look at the use of attribute and unification grammar as tools for linguistic description. We will see how the use of unification grammar in current unification-based theories of language shows a clear trend to reduce the role of syntax in language definitions, understood here as the role of phrase structure rules in the characterization of the language defined, and a serious redundancy between syntactic and semantic structure in the representations defined. The balance of this trend is unfortunate since, on the one hand, the first renders virtually useless the various techniques developed for the efficient syntactic analysis of context-free languages and, on the other, the second compounds the problem by making each step in an attributed derivation more computationally expensive, due to the increased size of the attributed representations involved. In contrast, attribute grammar definitions of language generally show a more balanced approach to the division between syntax and semantics and little redundancy between them, thereby lending themselves to more modular and efficient computational implementations.

The paper is organized as follows: Section 2 introduces the formalisms of attribute and unification grammar from the point of view of their notations and informal interpretations, as well as the method of initial algebra semantics. Then, in section 3, we develop a formal algebraic semantics for the two formalisms, based on the work of Pereira and Shieber [31] for unification grammar, and Chirica and Martin [6] for attribute grammar. Section 4 is devoted to an analysis of the expressive capabilities in the formalisms, including their term interpretations, partiality, and the limitations inherent to both formalisms as syntax-directed definitional tools. Section 5 considers questions of computational complexity and implementability of the formalisms. Finally, a conclusion is given regarding the superiority of attribute grammar for the statement of computationally oriented theories of language, but at the same time remarking on the need of new formalisms for principle-based description.

2. Notations and preliminaries

In this section, we introduce the basic ideas of attribute grammar, unification grammar, and the method of initial algebra semantics for programming languages. In what follows, a *language* is a set of strings over a finite set T of symbols and a *grammar* is a formal device for specifying which strings are in the set and for providing a denotation over a certain domain D of "semantic" objects for each string in the language.

The two grammar formalisms we are concerned with are based on augmentations of context-free grammars. A *context-free grammar* is a quadruple (N, T, P, S), where N is a finite set of string categories; T a finite set of terminal symbols; P a finite set of rewriting rules or *productions* of the form $X \rightarrow \sigma$, $X \in N$, $\sigma \in (N \cup T)^*$; and S a distinguished symbol of N. A binary relation $\Rightarrow$ of *derivation* between strings over the vocabulary $N \cup T$ of the grammar is defined such that $\alpha X \beta \Rightarrow \alpha\sigma\beta$ iff $X \rightarrow \sigma$ is a production of P: now, $\Rightarrow^*$ may be defined as the reflexive and transition closure of $\Rightarrow$. The *language generated* by the grammar, noted L(G), is the set of strings $\omega \in T^*$, such that $S \Rightarrow^* \omega$.

2.1. ATTRIBUTE GRAMMAR

An *attribute grammar* is defined upon a context-free grammar G = (N, T, P, S) by associating with each symbol $X \in N \cup T$ a finite set A(X) of *attributes*, and a type or domain D_a for each attribute a thus defined [29]. Each attribute a of symbol X, noted $X.a$, takes values over its domain and represents a specific, possibly context-sensitive property of the symbol. For each derivation D according to the base grammar G, we define a corresponding *attributed derivation* D′ by associating with each occurrence η of a symbol $X \in N \cup T$ in the derivation the attribute set

A(X) of X. Thus, the symbols of interest in attributed derivations are *attributed symbols* of the form $X[X.a_1, \ldots, X.a_n]$, where X is the syntactic category of the symbol and $X.a_i$, for $0 \leq i \leq n$, an attribute of X. We use $\eta.a$ to denote the *attribute occurrences* associated with η, for each attribute $a \in A(X)$.

Attribute values in a given derivation are defined by *attribution rules* of the form $X_i.a \leftarrow f(X_j.b, \ldots, X_k.c)$, associated with each production $p = X_0 \rightarrow X_1 \ldots X_n$ in the grammar, $0 \leq i,j,k \leq n$, and applied at each step in the derivation where the production is used. Here, *f* is an applicative expression (function) whose value depends on the values of attribute occurrences associated with symbols in the production. Each time *p* applies in the derivation, the attribution rule defines the value of the attribute occurrence X.a as a function of the occurrences $X_j.b, \ldots, X_k.c$, associated with other symbols in *p*. We let R(p) denote the packet of attribution rules associated with *p*. The grammar may also define *attribute conditions* of the form $b(X_i.a, \ldots, X_k.b)$, $0 \leq i, k \leq n$, where *b* is a Boolean predicate on the values of attribute occurrences of symbols in *p*. This condition must be satisfied in any derivation requiring the application of *p*, and thus contributes to the notion of grammaticality in the language generated by the grammar. We let B(p) denote the packet of attribute conditions associated with *p*.

The above remarks are summarized as follows: An *attribute grammar* is a four-tuple AG = (G, A, R, B), where

(i) G = (N, T, P, S) is a context-free grammar;

(ii) $A = \cup_{X \in N \cup T} A(X)$ is a finite set of attributes, with domains D_a, for each $a \in A$;

(iii) $R = \cup_{p \in P} R(p)$ is a finite set of attribution rules, as above; and

(iv) $B = \cup_{p \in P} B(p)$ is a finite set of attribution rules, as above.

Usually in the attribute grammar literature, the process of attribution and attribute evaluation in derivations according to AG is discussed in terms of the derivation trees defined by the grammar. The base grammar G assigns a derivation tree τ to each sentence in L(G). This tree is annotated at each node labelled X with the set A(X) of attributes associated with X; each attribute $a \in A(X)$ defines an attribute occurrence *X.a* at node X. If the grammar is well defined [29], it is possible to evaluate each attribute occurrence on the tree, and we say that τ is *correctly attributed* iff all attribute conditions yield "true". The *language* generated by the attribute grammar L(AG) is now the subset of L(G) whose members have at least one correctly attributed tree.

It is possible to classify the attributes in AG according to the manner in which their values are defined. Let $AF(p) = \{X.a \mid X.a \leftarrow f(\ldots) \in R(p)\}$ be the set of *defining attribute occurrences* associated with symbols in a production $p \in P$. We say an attribute X.a is *synthesized* if $X.a \in AF(p)$, for some production $p = X \rightarrow \sigma$; i.e. if its value depends only on attributes of X or descendents of X. The attribute is *inherited* if the production is of the form $p = A \rightarrow \alpha XB$; i.e. if its value depends on attributes associated with X, its parent, or siblings. We say the grammar is *S-attributed* if it contains only synthesized attributes. A more general and practically important class of grammars is obtained if we allow attributes of both kinds, but such that each inherited attribute depends only on inherited attributes of the parent, or attributes of the sisters to its left; in this case, we say the grammar is *L-attributed* [3].

We are generally interested only in attribute grammars that are well defined in the sense that all attribute occurrences can always be evaluated, in any conceivable derivation tree. An attribute grammar is *complete* if for each attribute occurrence X.a in a tree at least one attribution rule is applicable to compute its value; the grammar is *consistent* if at most one such attribution rule is applicable. In AGs with both inherited and synthesized attributes, it is not always obvious when the semantic rules do not amount to a circular definition. An attribute grammar is said to be *circular* if its attribute dependency graph, which records all the (data) dependency relations that arise from the attribution rules in the grammar, has a cycle, for some derivation tree.[1] Now we are in a position to say an attribute grammar is *well-defined* if it is complete, consistent, and its dependency graph is acyclic, for each derivation tree.

One of the chief practical questions of attribute grammar concerns the manner in which attribute values are evaluated in a given derivation tree. We start with the assumption that the derivation tree τ is available. This tree is decorated, so that each node η labelled by symbol X in the grammar is augmented with the attributes corresponding to it; if $a_i \in A(X)$, for $1 \leq i \leq n$, are the attributes of X, we let $Pos(\eta) = [\eta.a_1, \ldots, \eta.a_n]$ be the attribute frame or *positions* associated with η. We may think of each attribute occurrence $\eta.a_i$ in the attribute frame as a variable of the required type, to be evaluated as indicated by the attribution statements. Now we let $Pos(\tau)$ denote the collection of attribute positions in the whole tree.

The derivation tree, together with the attribution statements in the grammar, determine a collection $DT(\tau)$ of data dependency relations on the collection $Pos(\tau)$ of attribute occurrences in the tree. We say an occurrence $\eta.a$ *directly depends* on another attribute occurrence $\delta.b$, which we note $\delta.b < \eta.a$, if there is an attribution rule "$X.a \leftarrow f(\ldots, Y.b, \ldots)$" that defines the value of $\eta.a$ in terms of the value of $\delta.b$; the first occurrence cannot be evaluated unless the second has already been evaluated. The dependency relation $DT(\tau)$ is now the transitive closure of "$<$", and the *attribute evaluation problem* for attribute grammars can be formulated, rather abstractly, as the problem of finding a total order on $Pos(\tau)$, such that for each position $\eta.a \in Pos(\tau)$, its value depends only on positions lower in the order [15]. Attribute evaluation methods may be formulated such that they proceed from the bottom of the total order. We distinguish *static* attribute evaluators, which determine

[1] Circularity is a decidable property of attribute grammar, although the complexity of the test is exponential in grammar size [21].

the evaluation order at construction time from the grammar, and independently of the individual trees, from *dynamic* attribute evaluators, which determine the evaluation order at runtime, depending on the particular attributed tree. Most practical attribute evaluators are static and may be generated only for certain subclasses of attribute grammar, such as S-attributed, L-attributed, and ordered [25].

Below is a simple attribute grammar which enforces subject-verb agreement and the subcategorization properties of verbs, adapted from [33]. The grammar uses two attributes, *agr* and *subcat*, where *agr* = <person,gender,number> is the agreement feature of nominal and verbal projections, and *subcat* the list-valued subcategorization feature of the verbal projection, including its external argument [41].

$$\begin{array}{lll}
S \rightarrow NP \quad VP & \textit{cond}: & NP.agr = VP.agr \\
 & & VP.subcat = [NP] \\
NP \rightarrow N & \textit{attr}: & NP.agr \leftarrow N.agr \\
VP \rightarrow V & \textit{attr}: & VP.agr \leftarrow V.agr \\
 & & VP.subcat \leftarrow V.subcat \\
VP_0 \rightarrow VP_1 \quad XP, & \text{for} & XP = NP, PP, CP, \text{etc.} \\
 & \textit{attr}: & VP_0.agr \leftarrow VP_1.agr \\
 & & VP_0.subcat \leftarrow rest(VP_1.subcat) \\
 & \textit{cond}: & top(VP_1.subcat) = XP
\end{array}$$

The attribution processes in this grammar are of a rather simple kind; namely, synthesis of the attributes *agr* and *subcat* by the nominal and verbal projections from the corresponding lexical heads. The two attribute conditions, in the first and fourth productions, refer to the values of these attributes and could easily be assimilated to synthesized attributes. See [14] for a typical Earley-style derivation according to this grammar.

2.2. UNIFICATION GRAMMAR

A *unification grammar* is similarly based on a context-free grammar $G = (N, T, P, S)$.[2] It associates with each occurrence of a symbol $X \in N \cup T$ in a derivation a set of attributes or *feature structure* F(X) of *label–value* pairs, where the labels in the pairs are pairwise distinct and taken from a finite set L of labels, and the values are either primitive or complex. Primitive values are taken from a

[2] In what follows, we will only consider unification grammars with a context-free base and organize the definitions and discussion around that base. This allows us to reveal more succinctly the similarities and differences between attribute and unification grammar. See [33] for an extension to grammars with infinite non-terminal domains.

possibly infinite set C of atomic values, while complex values are themselves feature structures of label–value pairs.[3] Notice that the labels in a feature structure must be distinct, but that the values associated with the labels may be equal, where equality may be understood in the sense of type or token identity. In the case of token identity, we say that the values *share* structure. There is a natural interpretation of feature structure descriptions as terms over a language with variables [35], as finite-state automata [24], or as rooted directed acyclic graphs, in which arcs are marked with labels and nodes correspond to values which are themselves directed acyclic graphs [33].

Given a feature structure $F = \{\langle l_1, v_1\rangle, \dots, \langle l_k, v_k\rangle \mid k \geq 0 \text{ and } l_i \neq l_j,\ \text{if } i \neq j\}$ and a label l, we let (F, l) denote the value v if $\langle l, v\rangle \in F$, or *empty* (the empty feature structure) otherwise. Similarly, since values are themselves feature structures, if $l_1, \dots, l_n$ are $n \geq 1$ labels, we let $(F, l_1 \dots l_n)$ denote the value $((F, l_1), \dots, l_n)$. A string $\sigma = l_1 \dots l_n$ of labels is also known as a *path* of length *n*. For completeness, if "ε" is the empty path, we define $\langle F, \varepsilon\rangle = F$.

The feature structure associated with a symbol in a derivation is defined incrementally, in a syntax-directed manner, by a set of equations of the form $\langle i\, \sigma\rangle = V$, associated with each production $p = X_0 \rightarrow X_1 \dots X_n$ in the grammar, and applied at each step in the derivation where the production is used. Here, i is the index of symbol X_i in the production, $0 \leq i \leq n$, $\sigma \in L^*$ a path, and V either a constant $c \in C$ or a pair $\langle j\, \sigma'\rangle$, $0 \leq j \leq n$, $\sigma' \in L^*$. If V is the constant c and $F(X_i)$ contains the path σ, the meaning of such an equation is that $(F(X_i), \sigma) = c$. If $F(X_i)$ does not contain the path σ, it must be added to it as required and then its value made equal to c. If $V = \langle j\, \sigma'\rangle$, the meaning of the equation is that $(F(X_i), \sigma) = (F(X_j), \sigma')$, assuming the paths σ and σ' belong to $F(X_i)$ and $F(X_j)$, respectively. If either path is not in the corresponding feature structure, it must be added to it and then the previous equation applied (cf. [24]). A special case arises with equations of the form $\langle i\, \sigma\rangle = \langle i\, \sigma'\rangle$, where σ' is a subpath of σ, or vice versa. In such cases, the resultant structure is *cyclic* and is excluded from the domain of proper feature structures. The equation $\langle i\, \sigma\rangle = V$ is said to be a *partial descriptor* of $F(X_i)$, since it constrains the feature structure's value on a certain path. Let E(p) be the set of equations associated with production *p* in the grammar.[4]

The above remarks about (context-free based) unification grammar are summarized as follows: A *unification grammar* is a four-tuple UG = (G, L, C, E), where

[3] Shieber [33, p. 7] stipulates finite sets of both labels and atomic values; likewise, the automata theoretic domain of Kasper and Rounds [24] for feature structures requires finite sets.

[4] The usual characterization of unification grammar (e.g. [33]) assumes that the rewriting operation of the grammar is done over feature structures with an explicit attribute (label–value pair) marking the syntactic category of each structure. Our presentation, letting the rewriting be done on the CFG base alone and associating separately the feature structure of each symbol, differs formally but not in substance from this definition of UG.

(i) $G = (N, T, P, S)$ is a context-free grammar;
(ii) L is a finite set of labels or attribute-names;
(iii) C is an infinite set of constants, which correspond to basic feature structures; and
(iv) $E = \cup_{p \in P} E(p)$ is a finite set of equations in feature structures, as above.

The solution to each equation associated with a production in the grammar corresponds to *feature unification*, an operation that, given two feature structures F_1 and F_2, computes a feature structure F_2 that contains the information in F_1 and F_2 and denotes the intersection of the denotations of F_1 and F_2. Hence the motivation for calling the formalism "unification-based".

An issue not captured in the previous definition of UG is the fact that each symbol in a grammatical rule has its own feature structure associated with it, and that the descriptors associated with the rule, in general, state equivalences between values in these *different* feature structures. Hence, the entire grammatical rule is sometimes understood as having associated with it a feature structure embedding those of all the symbols in the rule, such that unification is always carried on different paths of a single structure [33].

To illustrate UG definitions, the previous grammar for subject-verb agreement and complex subcategorization is now:

$S \rightarrow NP\ VP$	<1 agr> = <2 agr> <2 subcat top> = "NP"
$NP \rightarrow N$	<0 agr> = <1 agr>
$VP \rightarrow V$	<0 agr> = <1 agr> <0 subcat> = <1 subcat>
$VP_0 \rightarrow VP_1\ XP$,	for XP = NP, PP, CP, etc. <0 arg> = <1 agr> <0 subcat> = <1 subcat rest> <1 subcat top> = "XP"

We should note that, unlike attribute grammar, where the set A(X) of attributes associated with a symbol X is fixed and defined in advance by the grammar, the top-level labels of the feature structure F(X) associated with each occurrence of X in a derivation are defined dynamically by the derivation, and may in fact turn out to be different for each occurrence of X in it.[5] In this regard, see section 4.2.

The next two sections, 2.3 and 2.4, present the basic definitions and results of the method of initial algebra semantics for context-free languages that will be

[5] Formally, the feature structure associated with a symbol is either an atomic value or a partial function $f: L \rightarrow F$ from labels into feature structures [31].

used to provide a semantics for attribute and unification grammar. The material is rather technical and may be skipped in a first reading by those readers not interested in the technical details of our presentation.

2.3. INITIAL ALGEBRA SEMANTICS

In this section, we review the necessary notations and results of the initial algebra approach to semantics of Goguen et al. [18]. Initial algebra has been used in the study of programming language semantics and in algebraic specification of abstract data types [19].

Let S be a set of type symbols or *sorts* and $A = \{A_s\}_{s \in S}$ an S-indexed family of sets. Then S^* is the set of all strings of type symbols and for each $w \in S^*$, we let $A^w = A_{s_1} \times A_{s_2} \times \ldots \times A_{s_n}$ if $w = s_1 s_2 \ldots s_n$, and $A^w = \{\ \}$ if w is the empty string. A function $f: A^w \to A_s$ is said to be of arity $|w|$, target type s, and type $\langle w, s \rangle$. If $|w| = 0$, then $f: \to A_s$ is regarded as a constant of type s.

A *signature* or S-sorted operator domain Σ is a family $\Sigma = \{\Sigma_{w,s}\}_{w \in S^*, s \in S}$ of sets of symbols, where each element $f \in \Sigma_{w,s}$ is an operator of type $\langle w, s \rangle$. An *S-indexed set of variables* is a family $V = \{V_S\}_{s \in S}$ of countable sets of variables, where V_s is the set of variables of type s.

DEFINITION

Let Σ be an S-sorted signature. A *many-sorted* (or *S-sorted*) *Σ-algebra* consists of an S-indexed family of sets $A = \{A_s\}_{s \in S}$ and a function $f_A: A^w \to A_s$ for each operator symbol $f \in \Sigma_{w,s}$, $w \in S^*$, and $s \in S$. The set $A = \cup_{s \in S} A_s$ is called the *carrier* of the algebra.

Many-sorted algebra is also known as heterogeneous algebra and is a generalization of one-sorted algebra, in which there is only one sort symbol. It admits a further generalization to order-sorted algebra [19], in which there is a provision for hierarchically defining subsorts of sorts. In the following, we use *A* ambiguously to refer to the algebra and its carrier.

Given an S-sorted signature Σ and S-indexed set V of variables, we let T_Σ denote the *term algebra* over the symbols in Σ (Herbrand universe of terms), and $T_{\Sigma(V)}$ the algebra of terms with variables in V. These algebras are of special interest, as we shall see below.

Given a Σ-algebra A and S-indexed set V of variables, an *assignment* of values in A to variables in V is a family of functions $\theta = \{\theta_s: V_s \to A_s\}_{s \in S}$.

DEFINITION

Let A and B be S-sorted Σ-algebras. A *Σ-homomorphism* $h: A \to B$ is a family of functions $h_s: A_s \to B_s$ that preserves the operations of the algebras. That is, for all $f \in \Sigma_{w,s}$, $\langle w, s\rangle \in S^* \times S$ and $a \in A^w$, $h_s(f_A(a)) = f_B(h^w(a))$, where $h^w: A^w \to A^w$ is the function that maps the i-th component of its argument $a \in A^w$ according to h_i, $0 \leq i \leq |w|$.

PROPOSITION

Given a Σ-algebra A, set V of variables, and assignment $\theta : V \to A$, there is a unique homomorphism $\Theta : T_{\Sigma(V)} \to A$ that extends θ, in the sense that $\Theta(x) = \theta(x)$, for all $x \in V$.

Given a signature Σ, a collection C of Σ-algebras together with all Σ-homomorphisms between the algebras is a category of Σ-algebras. Within this category, certain algebras are special in that they represent the category. The following is the key definition of many-sorted algebra.

DEFINITION

A Σ-algebra A is *initial* in a category C of Σ-algebras and Σ-homomorphisms if and only if there exists a unique Σ-homomorphism from A to any other Σ-algebra B in C.

The following proposition reveals the importance of the initial algebra concept.

PROPOSITION

If A and B are initial Σ-algebras in a category C of Σ-algebras, then they are isomorphic. Furthermore, for each signature Σ, the term T_Σ is initial in the category of all Σ-algebras and Σ-homomorphisms.

The category A_Σ of all Σ-algebras is sometimes called the anarchic category since it has no laws. In general, we are interested in categories that satisfy certain laws on their operations, expressed by means of equations. A *Σ-equation* is an expression of the form $s = t$, where s and t are terms with variables in $T_{\Sigma(V)}$, for some set V of variables.

In order to apply the initial algebra approach to the semantics of context-free languages, we define a class of many-sorted algebras T_G associated with each context-free grammar G. The operator domain of the algebra is sorted by the vocabulary symbols of the grammar and is denoted also by G. This algebra is initial in the category of all G-algebras, and its elements represent the derivation trees according to G. For simplicity, we will rely on our intuitive notion of "derivation tree" and refer the interested reader to [6] for a precise account.

2.4. PARTIALLY ORDERED SETS AND DENOTATIONAL SEMANTICS

The following definitions about complete partially ordered sets (CPOs) and elements of Scott's denotational approach to semantics [38] will be necessary in the next section to give a semantics to attribute and unification grammars.

A partially ordered set (or *poset*) is a pair $\langle P, \leq \rangle$, where P is a non-empty set and "$\leq$" a reflexive, antisymetric, and transitive relation on P. Often, we use P to denote both the set and the poset. A *chain* of a poset P is a subset X of P totally ordered by "$\leq$"; we let $\mathit{lub}_P(X)$ be the *least upper bound* of X in P, if it exists. If ϕ denotes the empty chain, we let $\mathbf{lub}_P(\phi) = \perp_P$ denote the least element of P, if it exists. Where no confusion may result, we drop the subscript P from $\mathbf{lub}_P$ and $\perp_P$. The poset P is a *complete partially ordered* set (CPO) if every chain X of it, including the empty chain, has a least upper bound. Notice that any set S may be made into a flat CPO $S_\perp = S \cup \{\perp\}$ by defining the order to be $a \leq b$, if $a = \perp$ or $a = b$, for all $a, b \in S$.

DEFINITION

Let P and Q be CPOs. A function $f : P \rightarrow Q$ is said to be *continuous* if, for each non-empty chain $X \subseteq P$, $\mathbf{lub}_Q(f(X))$ exists and, furthermore, $f(\mathbf{lub}_P(X)) = \mathbf{lub}_Q(f(X))$. The function is said to be *strict* if $f(\perp_P) = \perp_Q$.

Given arbitrary sets A and B, a partial function $f : A \rightarrow B$ can be made into a strict continuous function $f_\perp : A_\perp \rightarrow B_\perp$ (its natural extension) by letting, for each $a \in A$,

$$f_\perp(a) = \begin{cases} f(a) & \text{if } f(a) \text{ is defined,} \\ \perp_B & \text{if not,} \end{cases}$$

and

$$f_\perp(\perp_A) = \perp_B.$$

The set of all continuous functions $f : P \rightarrow Q$ can be made into a CPO, noted $[P \rightarrow Q]$. Also, for each CPO P, there exists a continuous function $\mathbf{Fix}_P$: $[P \rightarrow P] \rightarrow P$, defined by

$$\mathbf{Fix}_P(f) = \mathbf{lub}_{i \geq 0}(f^i(\perp_P)).$$

$\mathbf{Fix}_P$ is called the *least fixpoint operator* of P.

3. Algebraic semantics of attribute and unification grammar

In this section, we turn to the question of providing a semantics or denotation for a given attribute grammar or unification grammar. Our approach is motivated by the initial algebra approach of [6] for attribute grammars, in which the denotation of an attribute grammar is a set of pairings between derivation trees and a certain collection of functions between the inherited and synthesized attributes at the root of a tree. The approach is within the framework of many-sorted algebra of the last section, and could benefit from recasting in order-sorted algebras [36].

The semantics is given in terms of derivation trees rather than input strings directly since, in general, context-free grammar, the formalism on which both formalisms are based, does not provide a unique mapping from each input string into derivation trees, as fig. 1 shows. Thus, in order to obtain the usual denotation of a grammar,

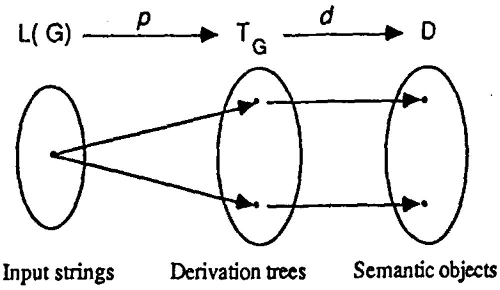

Fig. 1.

as a mapping from input strings into their "meanings", we would simply compose the mapping *p* from input strings to derivation trees with the map *d* from trees into their denotations. The first map is of course the parse relation between strings and trees determined by the context-free grammar, which we take for granted here.

3.1. SEMANTICS OF ATTRIBUTE GRAMMAR

Our first concern is the manner in which the attribution rules in an attribute grammar are stated. In our informal presentation of attribute grammar in section 2.1, the attribution rules associated with a production $p = X_0 \to X_1 \ldots X_n$ in the base grammar are of the form $X_i.a \leftarrow f(X_j.b, \ldots, X_k.c)$, where the expression on the right represents a semantic function $f_{pia} : D_b \times \ldots \times D_c \to D_a$, which defines the value of the attribute *a* of the *i*th symbol X_i in *p*, in terms of the values of attributes of other symbols in *p*.

If we let S be a set of sort symbols denoting the domains D_a of the attributes, the language in which the semantic functions are expressed is an S-sorted language of terms with variables, constructed from an S-sorted set V of *varible symbols* and an S-sorted signature Σ of *primitive function symbols* (including constants). The variables are written as $X_i.a$, where X is a syntactic category, *i* an optional index to indicate the instance of the category in the production, and $a \in A(X)$ an attribute of X. The primitive function symbols have types in $S^* \times S$ and are interpreted as functions over the domains D_a of the attributes. Using this language, each attribution rule is an equation of the form $\upsilon = t$, where υ is a variable symbol and t a term. The domains D_a of the attributes and the primitive function symbols in Σ define a Σ-algebra. The semantic function, expressed by the term t, is therefore a derived operation of the (Σ)-algebra.[6] The equation is well-typed if υ and t are terms of the same sort $s \in S$.

According to the definitions of section 2.1, an attribute grammar is well defined if it allows us to evaluate all attribute occurrences in any conceivable derivation tree. The grammar is complete and consistent if and only if for each production $p = X_0 \rightarrow X_1 \ldots X_n$ it defines (i) all and only the inherited attributes of X_i, for $1 \leq i \leq n$, and (ii) all and only the synthesized attributes of X_0. Thus, if we let $X.\eta$ and $X.\sigma$ denote, respectively, the vectors of inherited and synthesized attributes of X (in some fixed chosen order), the equations associated with the pth production may be written as:

$$X_0.\sigma = G_p(X_0.\eta, X_0.\sigma, X_1.\eta, X_1.\sigma, \ldots, X_n.\eta, X_n.\sigma),$$

$$X_i.\eta = F_{pi}(X_0.\eta, X_0.\sigma, X_1.\eta, X_1.\sigma, \ldots, X_n.\eta, X_n.\sigma), \qquad \text{for } 1 \leq i \leq n,$$

where G_p is the vector of terms consisting of the semantic functions f_{p0a}, for all (synthesized) attributes $a \in AS(X_0)$, in the order chosen in $X_0.\sigma$; and F_{pi} is similarly the vector of terms consisting of the semantics functions f_{pia}, for $1 \leq i \leq n$ and all (inherited) attributes $a \in AI(X_i)$, in the order chosen in $X_i.\eta$.

One problem to deal with in connection with the use of terms to define the semantic functions is that the function defined may be partial or have different values, according to the interpretation given to it. This happens, for example, with the term "if p then x else y", with a conditional primitive operation. This expression might be evaluable when p is "true" even if y is "undefined" or depends circularly on the value of the expression. The interpretation of the terms thus bears on important issues of attribute grammar, such as the circularity problem, their classification, and the formulation of attribute evaluation methods.

Taking Scott's denotational approach [38], the solution to this problem is to extend each attribute domain D_a into a CPO, adding a new value "$\perp$" to the domain, and to require all semantic functions f_{pia} to be continuous on the extended domain $D_a \cup \{\perp\}$. The new value $\perp$ can now be used to formalize the idea of a variable

[6] Strictly speaking, a term in a language does not define a derived operation, since we do not know the intended domain of the definition. However, following standard mathematical practice, the type information is inferred on syntactic grounds, so that if t is the term, $x_1, \ldots, x_n$ the variables in t in some fixed chosen order, and $\mathbf{f}$ its outermost operator symbol (in prefix notation), then the type of the derived operation is $s_1 \times \ldots \times s_n \rightarrow s_f$, where s_i is the type of x_i and s_f the target type of $\mathbf{f}$.

over the domain being "undefined". The above primitive semantic function "λpxy . if p then x else y" can now be extended in various continuous ways, making it strict in some or all of its arguments.

The above discussion regarding the algebraic definition of attribute grammar is summarized in the following definition of *Knuthian semantic systems* [6]:

DEFINITION

A *Knuthian semantic system* (K-system) is a quadruple $K = (G, D, ^-, _-, F)$, where

(i) $G = (N, T, P, S)$ is a context-free grammar;

(ii) D is a Σ-algebra, called the *primitive semantic algebra*, with a given set S of sorts and S-sorted operator domain Σ. The carriers D_s, for $s \in S$, are called the primitive semantic domains and the operations f_D of D, for each primitive function symbol $f \in \Sigma_{w,s}$ for $w \in S^*$, and $s \in S$, the primitive semantic functions;

(iii) $^- : N \cup T \rightarrow S^*$ and $_- : N \cup T \rightarrow S^*$ are two attribute *type-assignment functions*, which associate with each terminal and non-terminal symbol A the type strings A^- and A_- of its inherited and synthesized attributes of A, respectively.

(iv) $F = \cup_{p \in P}\{G_p, F_{pi} \mid p = X_0 \rightarrow X_1 \ldots X_n, 1 \leq i \leq n\}$ is the set of semantic functions associated with the productions in the grammar, given as terms with variables in the term algebra T_D. These functions must be derived semantic operations of the algebra D and, if $p = X_0 \rightarrow X_1 \ldots X_n$, of the types

$$\begin{aligned} &G_p: X_0^- \; X_{0-} \; X_1^- \; X_{1-} \ldots X_n^- \; X_{n-} \rightarrow X_{0-} \\ & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{for } 1 \leq i \leq n. \\ &F_{pi}: X_0^- \; X_{0-} \; X_1^- \; X_{1-} \ldots X_n^- \; X_{n-} \rightarrow X_i^-, \end{aligned}$$

We say a K-system is *continuous* if its underlying primitive algebra D is continuous. In this case, the semantic functions G_p and F_{pi} are continuous.

Having obtained a precise algebraic definition of attribute grammar (as a K-system), the question of providing a denotation for it can be directly answered. Given a derivation tree τ, assume a fixed ordering $n_0, n_1, \ldots, n_n, \ldots, n_m$ of its nodes, $n \geq 0$ and $m \geq n$, such that n_0 is the root and $n_1, \ldots, n_n$ the immediate descendents of n_0. Now, for $1 \leq i \leq m$, let C_i be the category of node n_i and let η_i and σ_i be distinct variable strings ranging over the domains $C^{C_i^-}$ and $D^{C_{i-}}$ of inhertited and synthesized attributes of C_i, respectively. We let w_τ be the vector of output and internal positions of the tree, defined recursively as $w_\tau = \sigma_0$, if $m = 0$ (i.e. τ consists of just its root node), and $w_\tau = \langle \sigma_0, \eta_1, \ldots, \eta_n, w_{\tau_1}, \ldots, w_{\tau_n} \rangle$ if the root is n_0 and $\tau_1, \ldots, \tau_n$ are the subtrees of τ with roots $n_1, \ldots, n_n$.

Since η_0 represents the input positions of the tree τ, the *attribute evaluation problem* now amounts to solving the recursively defined system of equations $w_\tau = H_\tau(\eta_0, w_\tau)$, given by

$$\begin{aligned}
\sigma_0 &= G_p(\eta_0, \sigma_0, \eta_1, \sigma_1, \ldots, \eta_n, \sigma_n), \\
\eta_i &= F_{pi}(\eta_0, \sigma_0, \eta_1, \sigma_1, \ldots, \eta_n, \sigma_n), \quad \text{for } 1 \le i \le n, \\
w_{\tau_i} &= H_{\tau_i}(\eta_i, w_{\tau_i}), \qquad\qquad\qquad\quad \text{for } 1 \le i \le n.
\end{aligned}$$

The bottom of this recursion occurs when $n = 0$; i.e. when the tree is only the root node, in which case the system of equations reduces to $\sigma_0 = G_p(\eta_0, \sigma_0)$. The solutions to the system are the fixpoints of H_τ on the second argument, of which we choose the least fixpoint, which always exists if the K-system is continuous. If AG is a continuous K-system, C_0 the category at the root of the tree τ, and D^τ denotes the domain of the output and internal positions of the tree, the *least fixpoint semantics* of τ assigned by the K-system is the function $S_\tau : D^{C_0^-} \to D^\tau$, defined by

$$S_\tau = \lambda\, \eta \cdot \mathrm{Fix}_{D^\tau}(H_\tau(\eta, w_\tau))$$

from the input attributes of the tree (η_0 is the vector of inherited attributes at the root C_0 of τ) into the vector of output and internal positions of τ ($w_\tau \in D^\tau$). If t has no inherited attributes at its root, the semantic function reduces to $S_\tau = \mathrm{Fix}_{D^\tau}(H_\tau(w_\tau))$, which is the fixpoint attribute valuation at all positions w_τ of the tree. If, as is usually the case, we are interested only in the synthesized attributes at the root of τ, we may take the semantic function associated with τ to be $h_\tau = \Pi_\sigma \circ S_\tau$, where Π_σ is the projection function $\Pi_\sigma : D^\tau \to D^{C_0-}$, which maps the output and internal positions $w_\tau = \langle \sigma_0, \eta_1, \ldots, \eta_n, w_{\tau_1}, \ldots, w_{\tau_n} \rangle$ of τ into the vector σ_0 of synthesized attributes at the root C_0 of the tree. Chirica and Martin [6] show that the family $h = \{h_\tau \mid \tau \text{ is a derivation tree in } T_G\}$ of functions is the unique homomorphism $h : T_G \to [D^{C_0^-} \to D^{C_0-}]$ from the algebra T_G of trees into the semantic algebra $[D^{C_0^-} \to D^{C_0-}]$ of evaluation functions.

The algebraic formulation of the attribute evaluation problem, which by itself is a rather complex process, frees it from the combinatorial aspects of attribute dependency graphs and from any recourse to tree traversal algorithms. The algebraic definition is also more general than the standard one, since by appropriate continuous extension of the primitive semantic functions it may handle even circular definitions. Lastly, the algebraic formulation suggests a method for converting an arbitrary continuous K-system, with inherited and synthesized attributes, into a purely synthesized one with the same base context-free grammar. For this, we eliminate all attributes of the converted system and associate instead a single synthesized attribute with each non-terminal symbol, over the domain $[D^{C_0^-} \to D^{C_0-}]$ of evaluation functions in the original system. A more modest application of this technique results in the "hoisting" transformation of Waite and Goos [40], whereby an inherited attribute *a* of a symbol is eliminated, and all uses of the attribute in the grammar are replaced by functions from the domain of *a* into the domain of the attribute being defined.

3.2. SEMANTICS OF UNIFICATION GRAMMAR

Our approach to the semantics of unification grammar is similar to that in the preceding section for attribute grammars; that is, it is based on the idea of mapping each derivation tree into the vector of feature structures associated with all nodes of the tree. The semantics of a unification grammar will then be a function from derivation trees into the function that maps each given (input) feature structure into a more specific instance of this structure.

Our first concern must be the interpretation of feature structure descriptions and of the descriptor equations used to define equivalence between them in a grammatical rule. We take a *feature structure* to be, as in section 2.2, either an atomic value c, taken from a possibly infinite set C of atomic values, or a collection $F = \{\langle l_i, v_i \rangle\}$ of label-value pairs, where the labels are taken from a finite set L of labels and are pairwise distinct, and the values are again feature structures, either atomic or themselves complex feature structures. We take the collection FS of all finite feature structures thus defined to be the *domain of feature structures*; it may be understood as the set of finite trees with arcs labelled by elements of L and terminal nodes labelled by elements of C.

As Pereira and Shieber [32] point out, it is important to distinguish features structures from the domain of their descriptions. For the latter, we shall take the domain of feature structures extended with variables over the domain of descriptions. That is, *feature structure description* is either an atomic value $c \in C$, a variable v taken from a countable collection V of variables, or a collection $F = \{\langle l_i, v_i \rangle\}$ of label-value pairs, where the labels $l_i \in L$ are pairwise distinct and the values v_i are finite descriptions. We let FD denote the *domain of feature structure descriptions*. If *d* is a feature structure description, we let *var*(*d*) denote the set of variables contained in *d* and say the feature description is *ground* if it contains no variables. Note that the set of ground feature structure descriptions is exactly the domain F of feature structures.

An *assignment* of values in FD to variables in V is a function $\theta : V \rightarrow FD$. Abusing terminology, we say the domain of the assignment is the set $dom(\theta) = \{v \mid v \in V \text{ and } \theta v \neq v\}$. We say the assignment is *ground* if it assigns to each variable in its domain a ground description. A *substitution* is an assignment σ with a finite domain.[7] If *d* is a description and σ a substitution, we let σd denote the description obtained by replacing each variable v in *var*(d) by its image $\sigma(v)$ under σ, with the usual

[7] An assignment, and hence a substitution, may be extended in a unique manner into a homomorphism on the algebra of feature descriptions, as we saw in section 2.3. However, for this extension to be meaningful, we must first reveal the algebraic structure of descriptions (cf. [35]).

care about variable name clashing and renaming; if the substitution is ground, then there is no such problem about renaming. The *denotation* of a feature structure description d may now be defined as the set

$$[d] = \{\sigma d \mid \sigma d \text{ is ground, for all substitutions } \sigma\}$$

of all ground descriptions (i.e. feature structures) that result, under all substitutions σ. There is a natural partial order called *subsumption* that results in feature structure descriptions, based on their denotation: Given descriptions d_1 and d_2, we have $d_1 \leq d_2$ (d_1 subsumes d_2) if the denotation of d_1 contains the denotation of d_2. The bottom of this partial order is $\perp$, which denotes the variables, while the top elements (there is not a unique one) are in the domain of (ground) feature structures. The order may be made into a complete lattice by adding the description $\top$ to FD, which denotes the inconsistent feature structure description and has an empty denotation.

Since a partial order is the basic concept underlying the formal framework of unification [34], the operation can be defined on the domain of feature structure descriptions. If d_1 and d_2 are feature descriptions, their *unification* $d_1 \equiv d_2$ is the least upper bound of d_1 and d_2; it corresponds to the descriptor that contains the information of both d_1 and d_2 and denotes the intersection of the denotations of d_1 and d_2. It can be shown that unification is continuous in the domain FD of feature descriptions.

Now we consider the partial descriptor equations associated with the rules in the grammar. Following Pereira and Shieber [31], the descriptors associated with a production $p = X_0 \rightarrow X_1 \ldots X_n$ are expressions $d_{pk} \in E(p)$, for $1 \leq k \leq |E(p)|$, of one of the following two forms:

$$\langle i\sigma \rangle = c, \qquad 0 \leq i \leq n;\ \sigma \in L^*;\ \text{and } c \in C,$$

$$\langle i\sigma \rangle = \langle j\sigma' \rangle, \qquad 0 \leq i, j \leq n \text{ and } \sigma, \sigma' \in L^*.$$

Each descriptor expression is taken as the specification of an equation on the feature structure descriptions associated with the symbols in the production. If $\sigma = l_1, \ldots, l_h$, where $l_i \in L$, and $F(X_i)$ denotes the description of symbol X_i, the first descriptor means the equation E_{pk}

$$F(X_i) \equiv \langle l_1, \langle \ldots, \langle l_h, c \rangle \ldots \rangle,$$

where the expression on the right-hand side is the feature structure description consisting only of the path σ and the value "c" at the end. If $\sigma = l_1, \ldots, l_h$ and $\sigma' = l'_1, \ldots, l'_m$, the second descriptor means the equation E_{pk}

$$F(X_i) \equiv \langle l_1, \langle \ldots, \langle l_h, v \rangle \ldots \rangle \ \&\ F(X_j) \equiv \langle l'_1, \langle \ldots, \langle l'_m, v \rangle \ldots \rangle,$$

where the variable v does not occur in either description $F(X_i)$ or $F(X_j)$, and is "shared" by the descriptions $F(X_i)$ and $F(X_j)$, at the end of the corresponding paths.

We say a partial description is *cyclic* if it entails an equation of the form $v \equiv d$, where d is a feature structure description that contains the variable v. This formulation of cyclic descriptions allows us to see that their unification has the unification "occurs-check" problem. Notice that since our definition of feature structures allows only finite ones (i.e. those generated by application of the construction rules a finite number of times) cyclic descriptions are inconsistent and have an empty denotation.

The equations E_{pk} corresponding to the set of descriptors E(p) associated with production p in the grammar are obtained individually as above. Care must be taken, however, that the variable in each equation of a descriptor with two paths is unique, not occurring in any other equation or in any of the feature structure descriptions associated with symbols in the production. Now, if the $d_{pk} \in E(p)$, for $1 \leq k \leq |E(p)|$, are the descriptors of p and the E_{pk} are the corresponding equations, the equation associated with the production p is the conjunction $E_p = E_{p1} \& \ldots \& E_{p,|E(p)|}$ of the equations determined by the descriptors in E(p). In general, notice that there may be some symbols in the production whose associated feature structure descriptions are not referred to in the equation of the production; these then are descriptions that do not communicate or share any information with the neighboring elements.

As before, having obtained a precise algebraic definition of unification grammar (as, say, a U-system), the question of providing a denotation for it can be directly answered. Given a derivation tree τ, assume a fixed ordering $n_0, n_1, \ldots, n_m$ of its nodes, $m \geq 0$, such that n_0 is the root. Now, for $0 \leq i \leq m$, let C_i be the category of node n_i, p_i the production associated with n_i, or "*nil*" if n_i is terminal, and $F(C_i)$ be the feature structure of C_i. We let w_τ be the vector $\langle F(C_0), \ldots, F(C_m) \rangle$ of feature structure descriptions of the tree. If $E_0, \ldots, E_m$ denote the equations associated with the productions $p_1, \ldots, p_m$, the equation associated with the tree is $E_\tau = E_0 \& \ldots \& E_m$. The solution of this equation on w_τ yields the feature structure description at each node of the tree; this solution exists since unification is a continuous operation on feature structure descriptions.

The input elements of this equation are the feature structures associated with the terminal nodes of the tree; the individual productions have no feature structures associated with them. Notice that this algebraic formulation of the unification problem for UGs is independent of the evaluation order for the equation associated with the tree. The formulation is also free from the combinatorial aspects of dependency graphs and from any recourse to tree traversal algorithms.

To conclude this section, we note that our somewhat informal treatment of feature structures and unification can be formalized further, as in the framework of feature unification of Smolka and Aït-Kaci [35]. The present approach is inspired by that of Pereira and Shieber [31], where the semantics is in terms of pairings of input strings and feature structures, for each grammatical category, but differs from it in the details.

4. Expressive capabilities

In light of the algebraic formulation of their semantics above, it is of interest to contrast the expressive capabilities of attribute and unification grammar. We consider the domains and interpretation of attribute values, partiality, and the limitations inherent to both formalisms as syntax-directed definitional tools.

4.1. FEATURE STRUCTURE DOMAINS AND TERM INTERPRETATIONS

It is clear from the formalization of sections 3.1 and 3.2 that the underlying semantic algebras of attribute and unification grammar are quite different. While the algebra of AG is heterogeneous, since it assumes a collection of possibly distinct attribute domains D_a, for each attribute $a \in A$, the algebra of feature structure descriptions in current versions of UG is one-sorted and furthermore restricted to a finite initial domain C of constant symbols [31]. Head-Driven Phrase Structure Grammar [32] provides an exception to the one-sorted approach to the domain of feature structures, using an order-sorted collection of domains used to capture regularities in the lexicon and simplify the statement of grammatical rules. Smolka and Aït-Kaci [35] and Siekmann [34] consider order-sorted feature structure descriptions and order-sorted unification on this domain. The advantages of a move to order-sorted domains lie in a remarkable increase in the expressiveness of the logic, and possibly also in the efficiency of unification algorithms.

The restriction of the initial domain C of constants to a finite set in unification grammar is not necessary in its algebraic semantics. It means, however, that one of the most venerable data types in computer science, namely the natural numbers, cannot be modeled directly; instead, it becomes necessary to use constructor labels such as "0" and "s:nat" to obtain a representation. The direct availability of infinite data types as the natural numbers is a necessity not only in general purpose computer languages, but also in grammatical formalisms; most current linguistic theories use indices of various sorts (like binding and agreement), whose domain is the natural numbers. While these domains may be represented indirectly by constructor labels as noted above, the approach is cumbersome and inefficient, since the space required for a representation and the time required for simple operations such as comparison for equality are proportional to the size of the representations.

Another practically important point of difference between AG and UG lies in the semantic interpretation of their operator symbols. In AG, for example, an expression like "3 + 5", where "+" is the addition operation, denotes the integer 8, and thus would be equivalent to "5 + 3". In UG, as in all languages of uninterpreted terms, the same expression "3 + 5", where now "+" is an infix functor interpreted as the symbol for addition, denotes itself, although it would also be interpreted as 8 by its human user. Since "5 + 3" also denotes itself, it is no longer equivalent to "3 + 5" (unless the commutativity of addition were stipulated by an axiom). Notice that this simple example of interpretation of numerical expressions, and the full interpretation of terms in general, is of interest in linguistic descriptions, and particularly in situations where those descriptions are used for natural language processing applications, since it is necessary to evaluate the truth of simple copulative sentences like "3 + 5 is 8" or "John's father is Paul". The problem of autodenotation or *basic term interpretation* [15] of *all* terms in a language imposes serious practical restrictions on any computer language. It must be borne in mind that neither UG nor AG are intended as theories of natural language, but rather as formalisms and computer languages that should provide the facilities and expressive capabilities in which such theories may be stated.[8)] In fact, pure logic programming, although interesting in its own theoretical terms, is not very practical, and one finds in languages like PROLOG operators such as "is", which side-step the logic to evaluate expressions made up of variables, constants, and certain allowed operators – so-called "computable expressions". (A similar case can be made about the relation between pure functional programming and LISP.)

Finally, we contrast attribute and unification grammar regarding their expressive capability for negation and disjunction. Both capabilities are linguistically motivated by a wide range of phenomena, as Karttunen [27] has shown, and are used in the formal statement of current linguistic theories. In the domain of syntax, for example, a grammatical constraint of universal grammar is that "all lexical NPs have Case" [9]. If *Case* is an attribute with domain *nom, acc, dat, gen, abl, nil,* where the first five values are Cases from traditional grammar and *nil* is used to mark the absence of Case on a noun phrase, the grammatical constraint may be captured either as the disjunction "(Case = nom $\vee$ Case = acc $\vee \ldots \vee$ Case = abl)", or more elegantly and succinctly as the negation "$\neg$Case = nil", associated with every lexical NP.

Negation and disjunction are commonly used in attribute grammar, in productions with conditions (cf. section 2.1). Such conditions are assimilated to synthesized attributes, which must evaluate to "true" for the application of the production to be valid. Like all expressions in attribute grammar, an attribute condition cannot be evaluated until all attribute values on which it depends are known (value semantics) or sufficient to determine the value of the condition (lazy semantics). The mechanism for the evaluation of conditions is thus directly built into the formalism. Another common use of negation and disjunction in AG is in conditional expressions such as "if p then e_1 else e_2" in attribution rules, where p is a Boolean condition and e_1 and e_2 are expressions; these greatly enhance the expressive power of the formalism and are just syntactic sugar for "p & $e_1 \vee \neg p$ & e_2".

Negation and disjunction are not commonly found in current unification-based formalisms or systems, except in some as that described in [27]. Disjunctive specification of feature values is, however, explicitly assumed in theories like

8) HPSG assumes in its inventory of formal tools, in addition to unification, the notion of *functionally dependent values*. This is a device by which the value of a given attribute in a feature structure may be defined as a function of the values of two or more other attributes in the feature structure. The intent here is that the function symbols are fully interpreted [32].

HPSG [32] and LFG [2]. Disjunction has been shown to be the source of NP-completeness for the consistency problem of feature structure descriptions [24].

4.2. PARTIALITY

There is another aspect in the interpretation of feature structures. As they were defined in sections 2.2 and 3.2, feature structures are finite objects of label-value pairs with no denotation assigned to them. However, one important fact about their interpretation in UG is that feature structures encode, in the label-value pairs, partial linguistic information about fully specified linguistic objects. In this regard, feature structures may be traced back to the symbols used in phonological description or the "complex symbols" of Chomsky's Extended Standard Theory [7]. Feature structures are then partial descriptions of objects in an underlying semantic domain, and their interpretation may be made in terms of sets of objects in the underlying domain or as elements participating in a partial order [31].

More generally, feature structures are important in knowledge representation and other new computer science applications as a means to represent taxonomically organized data. In one approach [35], the data are organized into *feature types*, which in turn are organized by subtyping and whose elements are records. Every feature type defines a set of features corresponding to the fields of its record elements. A *feature structure* (feature term) is used to denote the elements of a feature type, and as such is a partial description of the objects in the type. As we have noted, this order-sorted algebraic approach to data types is used in HPSG [32] to define a hierarchy of lexical types, which permits an efficient organization of lexical information in the theory, and a major simplification and elimination of redundancy in its statement.

The notion of partiality does not exist and unfortunately cannot be captured directly in attribute grammar.[9] The reason has to do with the fact that the attribute set $A(X)$ associated with each symbol $X \in N \cup T$ of the grammar is fixed and defined in advance by the grammar. Furthermore, the attribute grammar is well defined if it is complete and consistent, i.e. if it defines the value of each attribute of X in each occurrence of X in a derivation, and if it additionally defines the value of each such attribute at most once. We may think of each attribute symbol $X[X.a_1, \ldots, X.a_n]$ in a derivation, $a_i \in A(X)$, as a record and the set $A(X)$ as the definition of the fields of this record.

An important empirical question that deserves attention in linguistic applications, however, is whether the description of a given *linguistic* object in a complete linguistic analysis can be partial in the above sense (i.e. be unspecified for some

[8] The notion can always be captured indirectly where required, by means of an attribute over the domain of feature structures of the needed type, and unification as the required operation. Notice that due to its extremely general formulation, unification is always a valid and plausible primitive attribution operation in attribute grammar.

of the features that define the object). If this is not the case, then a formalism that requires all fields of an object to be present and defined in each analysis has some clear advantages from a software engineering point of view.

4.3. PRINCIPLE-BASED AND SYNTAX-DIRECTED DEFINITIONS

This is one respect in which attribute and unification grammar are very much alike. In both cases, attribute values or feature structure descriptions are defined in a syntax-directed manner, by equations associated with the productions in the grammar. Both are additionally *declarative* formalisms, since the attribute valuation for a given derivation is completely independent of the order in which the productions are applied in the derivation. In attribute grammar there are, however, constraints in the order in which the equations may be solved, dictated by the data dependencies between them. These two facts, in spite of the important differences noted in the previous two subsections, make attribute and unification grammar closely related; both are syntax-directed declarative grammatical formalisms.

Since Chomsky's [8] "Remarks on Nominalization", there has been a move in linguistic theory and in the development of new formalisms for linguistic description from intricate rule systems with conditions on the applications of rules, to interacting systems of principles, which state well-formedness conditions on the representations defined by the rules [9].[10] This methodological shift from "rule-based grammars" to "principle-based grammars" [1] may be seen clearly in Government-binding (GB) theory, Lexical-Functional Grammar (LFG), Generalized Phrase Structure Grammar (GPSG), and Head-Driven Phrase Structure Grammar (HPSG), and its theoretical objective is to move away from merely descriptively adequate grammars for particular (natural) languages to explanatorily adequate ones. On the notion of explanatory adequacy, the reader is referred to [7] and [10].

The language in which the principles in a principle-based grammar are couched is usually a language with terms and notation that may be used to refer to the representations defined by the (rules of the) grammar. Thus, we find terms that refer to nodes and relations between nodes of derivation trees, and to attributes and relations between attributes at nodes of trees. Examples of these principles include the Case filter and Binding axioms in GB theory, the Head Feature Convention and Foot Feature Principle in GPSG, their equivalents in HPSG, and various principles in LFG. While some of these principles are merely "syntactic sugar" that state general conditions that may be complied directly into the rules of a syntax-directed formalism like attribute or unification grammar, others are not, since they span nodes in a representation which may be arbitrarily far away. Examples of the former include the Case filter in GB theory, reviewed in section 4.1, and most principles

[10] We shall not enter here into the distinction between "rules" and "principles". For a more extended discussion of this and other matters, see [30] and [2].

in GPSG, LFG, and HPSG. For example, the Case filter may be translated into an attribute condition "if *lexical* then $\neg$*Case* = nil", associated with each NP rule in the grammar. Examples of the latter in GB theory include the Binding axioms and all principles that use the notion of government between nodes (the Empty Category Principle and Chain well-formedness principles, for example). Capturing principles with "unbounded" domains in a syntax-directed formalism undoubtedly requires the introduction of auxiliary attributes or mechanisms that locally capture relations between certain nodes or attributes in a derivation; in this regard, see [12] on chain formation in GB theory, [17] on the use of the "SLASH" feature in GPSG, [23] on functional uncertainty in LFG, and [32] on the use of the "SLASH" feature in HPSG.

To conclude this analysis of expressive capabilities in attribute and unification grammar and put the two grammatical formalisms in perspective, we see that both, as syntax-directed formalisms, have shortcomings as underlying frameworks for principle-based descriptions. The work of Berwick [2], Johnson [22], and Stabler [37] for Government-binding theory, and of Gazdar et al. [17] and Pollard and Sag [32] for unification-based theories in the direction of overcoming this problem is very welcome.

5. Computational complexity and implementability

One of the benefits associated with the algebraic characterization we have seen of attribute and unification grammar is that it now allows a careful study of their computational properties. In this section, we comment on computational complexity and implementability issues for AG and UG, including decidability, parsing, attribute evaluation, and feature unification.

5.1. DECIDABILITY AND COMPLEXITY

Attribute and unification grammar are extensions of context-free grammar with the ability to encode unrestricted amounts of information in their attribution. This makes the decidability problem for both formalisms dependent upon the degree of ambiguity of the underlying context-free grammar.

PROPOSITION

Attribute and unification grammar are decidable if the underlying context-free grammar is cycle-free – i.e. if it is finitely ambiguous.

Proof of this proposition may be made by reduction of a Turing machine to an attribute or a unification grammar. The reduction uses the attribution of the grammar symbols to encode the tape of the machine, and is possible just in the case the base grammar is cyclic. Detailed proofs are found in [13] for AGs and [15] for definite-clause programs (a formalism closely related to UG). The property of cycle-freeness has also been called "offline parsability" by Pereira and Warren [30], and used to show that LFG parsing is decidable under this condition [2].[11)]

Once we restrict our attention to cycle-free grammars to consider the added complexity of attribute evaluation or unification to the parsing process, some further considerations become necessary. First, as Church and Patil [11] have shown, the degree of ambiguity of a cycle-free grammar need not be bounded and may grow exponentially in the length of the input string; ambiguity of this sort is in fact quite common in natural language, as in the problem of prepositional phrase attachment. Since each different derivation of the input may lead to a different attribute evaluation, it is not very revealing to consider the added complexity on this already exponential problem. Hence, we restrict our attention to attribute evaluation on only *one* of the possible derivations. The second consideration to be made, in the case of attribute grammar, is that the complexity added to the parsing process is at least that of the primitive attribution functions used in the grammar. Hence, the complexity added should be given as a function of that in the attribution functions. For unification grammar, given that unification is the sole attribution operation used, the complexity result can directly take this into account.

In estimating the added complexity of attribute evaluation or unification on a derivation, we take as a basis the fact that the length of any derivation in a cycle-free grammar is linearly bounded by *n*, the length of the input string. If we let f_t be a function that bounds the size of each attributed term associated with a symbol in the derivation, as a function of *n*, and f_A a function that bounds the complexity of the attribution functions as a function of the arguments involved, we can see that the size of the attributed symbols in the derivation is at worst $f_t(n)$ and hence that the complexity of evaluating an attribution function on these symbols is at worst $f_A(f_t(n))$. Since the number of attributed symbols in the derivation is proportional to the length of the derivation (i.e. to *n*), we conclude that the complexity of the attributed derivation is $O(n^* f_t(n)))$. In contrast, the complexity of a simple context-free derivation is $O(n)$.

The remarks of the preceding paragraph are valid for attribute and unification grammar. Hence, the relative complexities of the two formalisms depend on the characteristics of their bounding functions f_t and f_A. For unification grammar, the size of the terms is linearly related to the length of the derivation – i.e. $f_t(n) = O(n)$;[12)] furthermore, the complexity of unification is linear in the size of its arguments, for

[11)] HPSG has recently been shown to be undeciable, this time not due to properties of the base grammar, but due to the introduction of lexical rules and their ability to encode unrestricted amounts of information in the attribution of the symbols generated [4].

[12)] Once we consider adding the HPSG notion of functionally dependent values to the basic unification formalism [32], the noted linear relation between the size of attributed terms and the length of derivations need no longer hold, since it becomes easy to write attribution statements that violate that relation.

the best algorithms known [34]. Hence, $f_A(f_t(n))$ is also linear and the complexity of the derivation with unification is $O(n^2)$.[13] It should be noted that the linear unification algorithms tend to be inefficient in practical cases (small terms) due to a large overhead in the manipulation of pointer structures, and that a naive unification algorithm can be exponential in the worst case. Hence, the complexity of practical unification systems can be much higher than the above theoretical result suggests.

For attribute grammar, the situation is not so clear-cut, given that its attribution domains and functions have been left largely unspecified. Thus, in the absence of any information or restrictions on the attribution domains and functions, the functions f_t and f_A can grow arbitrarily quickly. However, it is not reasonable to suppose (or require) that f_t will grow at most linearly with the length of the input string and that f_A will also do the same on the size of its arguments, as we have seen is the case for unification grammars. In fact, with some noteworthy exceptions, it can be seen in current practice and use of attribute grammar the use of rather simple and unstructured attribute domains; examples are enumerated types, the integers or reals, and simple non-recursive record types. These domains have the property that their elements are of a small and constant size, independent of the input length. The exceptions can be counted on one hand and include, for example, list, set, or tree domains in linguistic applications (e.g. for subcategorization lists or sets of binding indices), or the symbol table in a compiler. These have in common that the size of their members grows at most linearly with input length. Thus, in contrast to unification grammar, we shall expect $f_A(f_t(n))$ to be constant for attribute grammar, so that attributed derivations on AGs are $O(n)$ – i.e. linear on input length.[14]

5.2. PARSING

Although the parsing problem has been explored at length for attribute and unification grammars, it has been done so with different applications in mind. Attribute grammar has been oriented since its beginnings primarily towards the semantic definition and translation of programming languages. Thus, a common requirement on the underlying context-free syntax is that it be unambiguous [40]. The grammar is often further restricted to fit a particular style of parsing, such as LL, LR, or LALR, and current compiler-writing systems based on attribute grammar methodology work with this restriction [26].

In contrast, unification-based formalisms and systems have been developed purposely with natural language applications in mind. Given that local and global ambiguity is a fact of natural languages, unification formalisms often do not impose any restriction on their context-free base, and unification systems such as

[13] An identical result is reported by Pereira and Warren [30] for definite clause grammars.

[14] As stated in the main text, this result is valid only for f_t and f_A constant, and for the reasons noted above, this is true in practical attribute grammar descriptions. See also section 5.2.

PATR-II [33] and the Grammar Development Environment (GDE) of Cambridge University [5] employ all-path parsers, such as may be obtained from Earley's [16] algorithm or chart parsing techniques.

The restrictions imposed on the context-free base of attribute grammar are not inherent to the formalism and have been made on current systems only for practical purposes, to make possible the application of the most efficient context-free parsing methods. As we saw in section 3.1, the correctness and well-definedness of an AG is not dependent on the non-ambiguity of its base. Current work by the author and by Yellin [42] addresses the problem of ambiguous underlying syntax in attribute grammar by an extension of Earley's algorithm or generalized attributed parsing. The generalization of LR parsing techniques of Tomita [39] also allows the use of a limited form of attribution in grammar rules. The extension of Earley's algorithm worked out in [14] does on-line attribute evaluation, allows for ambiguous and even cyclic bases, and is sufficient to handle the attribution of S- and L-attributed grammars; these are two important subclasses of attribute grammar, which may be sufficient to cover important aspects of natural language attribution. Considering the generalization of the algorithm to a wider class of attribute grammars will bring us to the next section, on attribute evaluation, but first we consider the question of computational efficiency in the *use* of grammatical formalisms in current unificiation-based theories of language.

Questions of computational efficiency in the use of grammatical formalisms have special relevance for the analysis of computationally oriented linguistic theories and their degree of success. Here, we take the efficiency of an attributed grammatical formalism, as used in a particular linguistic theory, as a function of the syntactic derivations it provides for the strings in the languages defined by the theory and the complexity of the attributed symbols involved in those derivations. The question forces a close scrutiny of the syntactic rules and the makeup of the attribution posited by the theories in the symbols. In this regard, the use of the unification grammar formalism in some current unification-based theories of languages shows a clear trend to reduce the role of syntax in language definitions, understood here as the role of phrase structure rules in the characterization of the languages defined, and a serious redundancy between syntactic and semantic structures in the representations defined.

The trend to reduce the role of syntax is seen most clearly in HPSG [32], where the number of phrase structure rules is reduced to an extremely small number (perhaps four), and the attribution associated with each lexical sign is used to indicate the particular kinds of complement and adjunct constituents that the sign selects, and thus semantically constrain the application of the rules. The redundancy of syntactic and semantic representation in a theory results from the use of attribution to record structurally defined relations. In LFG, for instance, attributes are used to represent the grammatical relations between a sign and its complements. Although the motivation and details of the analysis provided are subtle, such relations are essentially structurally defined and, in fact, in other frameworks such as

Chomsky's [7] Extended Standard Theory, they are so. Similarly, HPSG uses attribution to record the structurally-defined mother–daughter relations between nodes in a derivation tree. In both cases, the size of the resultant attribution in each derivation symbol is increased dramatically, being dependent upon the length of the strings generated. The balance of the trend noted is unfortunate since, on one hand, the first renders virtually useless the various techniques developed for the efficient syntactic analysis of context-free languages and, on the other, the second compounds the problem by making each step in an attributed derivation more computationally expensive, due to the increased size of the attributed representations involved.

In contrast, attribute grammar definitions of language generally show a more balanced approach to the division between syntax and semantics and little redundancy between them, thereby lending themselves to more modular and efficient computational implementations.

5.3. ATTRIBUTE EVALUATION

Attribute evaluation is one of the most difficult problems of attribute grammar. Modulo the problem of syntactic ambiguity in the underlying context-free base noted in the previous section, the attribute evaluation problem has been studied and solved for major subclasses of AG, as we saw in section 2.1 (e.g. [25]).

However, the attribute evaluation problem does not seem to show up in unification-based formalisms. The reason for this lies in the remarks of section 4.1 regarding the Herbrand (or basic term) interpretation of all terms (feature structures) in UG and the use of unification for the statement of equations between them. This virtually eliminates the attribute evaluation problem from the formalism. While in AG a simple attribution rule like "X.a $\leftarrow$ Y.b" or "X.a $\leftarrow$ f(. . . , Y.b, . . .)" cannot be evaluated unless the value of the attribute occurrence *Y.b* in the right-hand side expression is known, the corresponding equation "v $\equiv$ u" or "v $\equiv$ f(. . . , u, . . .)" in a unification-based formalism can be evaluated symbolically, even if the value of the variable *u* on the right-hand side is not known. In the latter case, evaluation of the equation yields the unification of the structure *v* with the term *u* or *f*(. . . , *u*, . . .) on the right, which may only be partially instantiated. Since the evaluation of a conjunction e_1 & e_2 & . . . & e_n of unification equations is order independent, the unifications associated with a UG derivation of a given input string can be performed in any order, including that in which they are found by the particular parsing algorithm used. This is an important advantage of UG, which was in fact used in the attribute evaluator of the Government-binding parser in [13].

The advantage of symbolic attribute evaluation quickly disappears, however, once we consider extending the expressive power of the formalism to include negation or disjunction, or the notion of functionally dependent values, as noted in section 4.1. In particular, for a negation equation "$\neg$b(. . . , d, . . .)" to be evaluated, where *b* is a Boolean predicate that depends on the value of a description *d*, it is necessary to know the relevant facts about *d*. For otherwise it may be necessary to enumerate all instances of *d* over a possibly infinite domain, to verify which satisfy the equation. The problem is less severe for disjunction, since now we have to consider enumeration only over the alternatives in the disjunction, but it is enough to make the evaluation problem for unification with disjunction NP-complete [24].[15] Finally, notice that in a practical computer language, with computable expressions such as arithmetic and string operations over their arguments, it is often required that the arguments input to the expression be evaluated before the expression is evaluated. In this case, we must also postpone evaluation of the computable expression.

The above remarks suggest that extension of the basic unification grammar formalism to include new facilities like negation, disjunction, and computable expressions, which are all linguistically motivated and a necessity in any realistic computer language, requires careful analysis and consideration of the data dependencies introduced in the representations defined by a given grammatical definition, and the development of efficient strategies for the evaluation of the same representations. Likewise, theories of language that seek to address the problem of linguistic performance must heed the observations of the preceding section regarding the computational efficiency of their grammatical descriptions, which must carefully take into account the strengths and limitations of the formalism in which they are couched.

5.4. FEATURE UNIFICATION

From our discussion of feature structure descriptions in section 3.2, it is clear that these descriptions are terms in a language with variables. In this interpretation, we need to supply an implicit functor of variable arity $n \geq 0$, and selectors (the labels in the feature structure description) that serve to identify the argument positions in the term.

Unification of feature structure descriptions under this interpretation is similar to standard (one-sorted) unification in logic programming languages. However, there is an added dimension of complexity to standard unification. Since feature structure descriptions have an implicit functor to which arguments may be added, as selectors are added, we must consider the problem of storing and searching label–value pairs in a given feature structure description. We consider two basic representation strategies for this sort of information.

One possible representation is representing each feature structure by a vector consisting of label–value pairs, for all possible labels, and assuming a fixed order for the label set. In this manner, a constant access time can be guaranteed to each possible argument of the structure. Since the set *L* of labels in the grammar may be large, and typically each feature structure selects only a small subset of labels,

[15] Kaplan and Zaenen [23] present, within the framework of Lexical Functional Grammar, a treatment of long-distance dependencies in natural language that is stated over functional rather than phrase structure. The formal account is given by a device "functional uncertainty", that allows the (finite) specification of infinite disjunctions.

according to its syntactic category or type, this alternative seems too wasteful of memory space. Furthermore, unification of two structures would always require testing up to $|L|$ label–value pairs. A second representation of feature structures is using lists of label–value pairs. This can be more economical in memory space, but since in general the selectors may be added in any order, the list would be unordered. Hence, it becomes necessary to consider maintaining the list in order and searching on it each time a selector is given. If the list is unordered, searching will require on the average $s/2$ steps, where s is the number of selectors in the feature structure. Notice that this shows up in the complexity of the unification algorithm as a term dependent on grammar size (but not input string length).

Several alternatives for the representation of feature structures are used in practical systems. In the Cambridge University GDE, the user has the choice between feature structure and standard term unification [5]. On the other hand, in the record processing language PLNLP of Heidorn [20], the most used labels in the grammar are stored in a vector associated with each record, while the others are kept in a list. This optimization has a major impact on the efficiency of the record processing system.

6. Conclusion

Unification is attractive for its elegance and conciseness in the description of complex terms and the operations between them, but can easily encourage and lead to computationally inefficient descriptions when interpreted on a machine. By taking similar algebraic characterizations of attribute and unification grammar, the present article has allowed us to examine in detail some of the important properties and issues in the two formalisms, including their descriptive capabilities, complexity, implementability issues and, also importantly, the problems of use of the two formalisms in current state-of-the-art. One suggestion that can be made about unification grammar is that its rather simple domain of feature structures can be significantly improved by turning it into a full order-sorted algebra (cf. [35]), and has in fact already been claimed for HPSG theory. This has the potential of greatly improving the expressiveness of UG and also the computational efficiency of UG systems, although at the expense of greater implementation complexity and more carefully crafted grammatical descriptions.

Attribute grammar and unification-based formalisms are closely related syntax-directed definitional tools for grammatical description. The implications of this close relation and the uses that may be made of it are similar to those obtained from the also close relation that has been shown to exist between attribute grammar and logic programming [15]. Efficient attribute evaluation methods developed for attribute grammar, based on an analysis of the data dependencies determined by the attribution, may apply to replace general unification in the execution of logic programs and unification formalisms by more specialized and efficient forms of unification. In general, however, we have seen that current unification-based theories of languages like Head-driven Phrase Structure Grammar already make demands beyond the capabilities of the basic unification grammar formalism, very close to those provided by attribute grammar. The conclusion of this study is that attribute grammar is a better suited and more highly developed grammatical formalism for the description of natural and artificial languages, due to its greater generality, expressive power and, also importantly, more efficient computational implementations.

To bring to an end and to put the present review and analysis of the two grammatical formalisms into perspective, we note that attribute and unification grammar are more similar than dissimilar grammatical formalisms, and that both fall short of the sort of formalisms needed for "principle-based" descriptions of language, advocated in current linguistic theory [1,10,32]. This mode of description is increasingly more important for theoretical and computational linguistics, as advance is made on the generative (i.e. precise) study of particular languages, and research turns to the Port Royal grammarian's conception that "with respect to its substance, grammar is one and the same for all languages, though it does vary accidentally" (cf. Chomsky, op cit.).

Acknowledgements

The author would like to thank Rodrigo Cardoso for many helpful discussions on algebraic semantics and unification, which have clarified my understanding of them. I am indebted to two anonymous reviewers of this journal for thoughtful criticism that has helped to improve the overall quality and presentation of this paper and, especially to one of them, for drawing my attention to recent work in HPSG and its order-sorted algebraic approach. I also thank Martin Kay and anonymous reviewers for the IWPT-91 Conference, who pointed out in response to [14] the need for a comparison of the notation and semantics of attribute and unification grammar. Any shortcomings and infelicities in this paper are of course my own. The research was made possible by an equipment grant from IBM Research; special thanks to Sanjaya Addanki and Wlodek Zadrozny for their continued support.

References

[1] R. Berwick, Principle-based parsing, MIT A.I. Memo 972 (revised), MIT, Cambridge, MA (1988).

[2] J. Bresnan and R. Kaplan, Lexical functional grammar: A formal system of grammatical representation, in: *The Mental Representation of Grammatical Relations*, ed. J. Bresnan (MIT Press, Cambridge, MA, 1982) pp. 173–281.

[3] G. Bochmann, Semantic evaluation from left to right, Commun. ACM 19(1976)55–62.

[4] B. Carpenter, The generative power of categorial grammars and head-driven phrase structure grammars with lexical rules, Comp. Linguistics 17(1991)301–313.

[5] J. Carroll, B. Boguraev, C. Grover and T. Briscoe, A development environment for large natural language grammars, Report No. TR-127, Computer Laboratory, University of Cambridge (1988).

[6] L. Chirica and D. Martin, An order-algebraic definition of Knuthian semantics, Math. Syst. Theory 13(1979)1–27.

[7] N. Chomsky, *Aspects of the Theory of Syntax* (MIT Press, Cambridge, MA, 1965).
[8] N. Chomsky, Remarks on nominalization, in: *Readings in English Transformational Grammar*, ed. R. Jacobs and P. Rosenbaum (Waltham, MA, 1970).
[9] N. Chomsky, *Lectures on Government and Binding* (Foris, Dordrecht, 1981).
[10] N. Chomsky, *Knowledge of Language* (Praeger, New York, 1986).
[11] K.W. Church and R. Patil, Coping with syntactic ambiguity: Or how to put the block in the box on the table, Amer. J. Comp. Linguistics 8(1982)139–149.
[12] N. Correa, An attribute grammar implementation of government-binding theory, *Proc. 25th Annual Meeting of the ACL*, Stanford University, Stanford, CA (1987).
[13] N. Correa, Syntactic analysis of English with respect to government-binding grammar, Ph.D. Dissertation, Department of Electrical and Computer Engineering, Syracuse University, Syracuse, NY (1988).
[14] N. Correa, An extension of Earley's algorithm for S- and L-attributed grammars, *Proc. Int. Conf. on Current Issues in Computational Linguistics*, Sains Malaysia University, Penang, Malaysia (1991).
[15] P. Deransart and J. Maluszynski, Relating logic programs and attribute grammars, J. Logic. Progr. 2(1985)119–155.
[16] J. Earley, An efficient context-free parsing algorithm, Commun. ACM 13(1970)94–102.
[17] G. Gazdar, E. Klein, G. Pullum and I. Sag, *Generalized Phrase Structure Grammar* (Harvard University Press, Cambridge, MA, 1985).
[18] J.A. Goguen, J.W. Thatcher, E. Wagner and J. Wright, Initial algebra semantics and continuous algebras, J. ACM 24(1977)68–95.
[19] J.A. Goguen, J.W. Thatcher, E. Wagner and J. Wright, Initial algebra approach to the specification, correctness, and implementation of abstract data types, in: *Current Trends in Programming Methodology*, Vol. 4, ed. R. Yeh (Prentice-Hall, Englewood Cliffs, NJ, 1978).
[20] G. Heidorn, Augmented phrase structure grammars, in: *Theoretical Issues in Natural Language Processing*, ed. B.L. Webber and R. Schank, ACL (1975).
[21] M. Jazayeri, W. Odgen and W.C. Rounds, The intrinsically exponential complexity of the circularity problem for attribute grammars, Commun. ACM 18(1975)697–706.
[22] M. Johnson, Use of knowledge of language, *Proc. 26th Annual Meeting of the ACL*, SUNY Buffalo, Buffalo, NY (1988).
[23] R. Kaplan and A. Zaenen, Long-distancs dependencies, constituent structure, and functional uncertainty, in: *Alternative Conceptions of Phrase Structure*, ed. M. Baltin and A. Kroch (Chicago University Press, Chicago, 1987).
[24] R. Kasper and W.C. Rouds, A logical semantics for feature structures, *Proc. 24th Annual Meeting of the ACL*, Columbia University, New York (1986) pp. 257–265.
[25] U. Kastens, Ordered attribute grammars, Acta Inf. 13(1980)229–256.
[26] U. Kastens, B. Hutt and E. Zimmermann, *GAG: A Practical Compiler Generator*, LNCS 141 (Springer, Berlin, 1982).
[27] L. Karttunen, Features and values, *Proc. 10th Int. Conf. on Computational Linguistics*, Stanford University, Stanford, CA (1984).
[28] M. Kay, Parsing in functional unification grammar, in: *Natural Language Parsing*, ed. D. Dowty, L. Kartunen and A. Zwicky (Cambridge University Press, Cambridge, 1985).
[29] D. Knuth, Semantics of context-free languages, Math. Syst. Theory 2(1968)127–145.
[30] F. Pereira and D.H. Warren, Parsing as deduction, *Proc. 21st Annual Meeting of the ACL*, Cambridge, MA (1983) pp. 137–144.
[31] F. Pereira and S. Shieber, The semantics of grammar formalisms seen as computer languages, *Proc. 10th Int. Conf. on Computational Linguistics*, Stanford University, Stanford, CA (1984) pp. 123–129.
[32] C. Pollard and I. Sag, *Information-Based Syntax and Semantics, Vol. 1 – Fundamentals*, CSLI Lecture Notes, Vol. 13 (Chicago University Press, Chicago, 1987).

[33] S. Shieber, *An Introduction to Unification Based Approaches to Grammar*, CSLI Lecture Notes No. 4, Stanford, CA (1986).

[34] J. Siekmann, Unification theory, FB Informatik, Universität Kaiserslautern, Kaiserslautern, Germany (1987).

[35] G. Smolka and H. Aït-Kaci, Inheritance hierarchies: semantics and unification, MCC Technical Report No. AI-057-87, Austin, TX (1987).

[36] G. Smolka, W. Nutt, J. Goguen and J. Meseguer, Order-sorted equational computation, SEKI Report No. SR-87-14, FB Informatik, Universität Kaiserslautern, Kaiserslautern, Germany (1987).

[37] E. Stabler, *The Logical Approach to Syntax: Foundations, Specification, and Implementations of Theories of Government and Binding* (MIT Press, Cambridge, MA, 1991).

[38] J. Stoy, *Denotational Semantics: The Scott–Strachey Approach to Programming Language Theory* (MIT Press, Cambridge, MA, 1977).

[39] M. Tomita, An efficient augmented-context-free parsing algorithm, Comp. Linguistics 13(1987) 31–46.

[40] W. Waite and G. Goos, *Compiler Construction* (Springer, New York, 1984).

[41] E. Williams, Argument structure and morphology, The Linguistic Review, No. 1 (1981).

[42] D. Yellin, Generalized attributed parsing, Manuscript, IBM Research, Yorktown Heights, New York (1988).