On the Semantics of Unification Grammars

Shuly Wintner (shuly@cs.haifa.ac.il)

June 4, 2003

Abstract. An essential prerequisite for modular grammar design is a clear, mathematically well-founded definition for the semantics of grammar formalisms, facilitating reasoning about grammars and their computational properties. This paper shows that existing definitions for the semantics of unification grammars, both operational and denotational, are not compositional with respect to a simple and natural grammar combination operator. Adapting results from the semantics of logic programming languages, we suggest a denotational semantics that we show to be both compositional and fully-abstract. This semantics induces an equivalence relation by which two grammars are equivalent if and only if they can be interchanged in any context. This provides a clear, mathematically sound way for defining grammar modularity.

Keywords: Unification grammars, Modularity, Semantics of programming languages

1. Introduction

In a recent paper, Wintner (2002) proposed a new definition for the semantics of context-free grammars, indicating that the traditional definitions are not compositional with respect to a simple grammar-combination operator. This short note extends the results of Wintner (2002) from the simpler case of context-free grammars to unification formalisms and provides a mathematically clean semantics for unification grammars which supports a limited notion of modularity. Adapting results from the semantics of logic programming languages, we suggest a denotational semantics that is both compositional and fully-abstract. This semantics induces an equivalence relation by which two grammars are equivalent if and only if they can be interchanged in any context. This provides a clear, mathematically sound way for defining grammar modularity.

The only major improvement of this work over Wintner (2002) is the extension of the domain of the discussion from context-free grammars to the more complex and linguistically motivated unification grammars. The line of reasoning, and in particular the mathematical proofs, remain mostly unchanged. We simply list the necessary changes here. For the motivation, the methodology and a discussion of related work, refer to Wintner (2002).

2. Basic definitions

Viewing grammars as formal entities that share many features with computer programs, it is natural to consider the notion of *semantics* of unification-based formalisms. Analogously to logic programming languages, the denotation of unification based grammars can be defined using various techniques. We review in this section the operational definition of Shieber et al. (1995) and the denotational definition of, e.g., Pereira and Shieber (1984) or Carpenter (1992), pp. 204-206. Recalling similar results from Wintner (2002), we find that these definitions are equivalent and that none of them supports compositionality.

2.1. Unification grammars

We assume familiarity with theories of feature structure based unification grammars, as formulated by, e.g., Carpenter (1992) or Shieber (1992). Grammars are defined over *typed feature structures* (TFSs) which can be viewed as generalizations of first-order terms (Carpenter, 1991). TFSs are partially ordered by subsumption, with $\bot$ the least (or most general) TFS. A *multi-rooted structure* (MRS, see Sikkel (1997) or Wintner and Francez (1999)) is a sequence of TFSs, with possible reentrancies among different elements in the sequence. Meta-variables A, B range over TFSs and σ, ρ over MRSs. MRSs are partially ordered by subsumption, denoted '$\sqsubseteq$', with a least upper bound operation of *unification*, denoted '$\sqcup$', and a greatest lowest bound denoted '$\sqcap$'. We assume the existence of a fixed, finite set Words of words. A *lexicon* associates with every word a set of TFSs, its **category**. Meta-variable a ranges over Words and w over strings of words (elements of Words*). Grammars are defined over a *signature* of types and features, assumed to be fixed below.

DEFINITION 1 (Grammars). *A* **rule** *is an MRS of length greater than or equal to 1 with a designated (first) element, the* **head** *of the rule. The rest of the elements form the rule's* **body** *(which may be empty, in which case the rule is depicted as a TFS). A* **lexicon** *is a total function from* Words *to finite, possibly empty sets of TFSs. A* **grammar** $G = \langle \mathcal{R}, \mathcal{L}, A^s \rangle$ *is a finite set of rules* $\mathcal{R}$*, a lexicon* $\mathcal{L}$ *and a* **start symbol** A^s *that is a TFS.*

Figure 1 depicts an example grammar,[1] suppressing the type hierarchy on which it is based.[2]

$$A^s = (cat : s)$$

$$\mathcal{R} = \left\{ \begin{array}{lcll} (cat:s) & \rightarrow & (cat:n) & (cat:vp) \\ (cat:vp) & \rightarrow & (cat:v) & (cat:n) \\ (cat:vp) & \rightarrow & (cat:v) & \end{array} \right\}$$

$$\mathcal{L}(\text{John}) = \mathcal{L}(\text{Mary}) = \{(cat:n)\}$$
$$\mathcal{L}(\text{sleeps}) = \mathcal{L}(\text{sleep}) = \mathcal{L}(\text{loves}) = \{(cat:v)\}$$

Figure 1. An example grammar, G

The definition of unification is lifted to MRSs: let σ, ρ be two MRSs of the same length; the *unification* of σ and ρ, denoted $\sigma \sqcup \rho$, is the most general MRS that is subsumed by both σ and ρ, if such an MRS exists. Otherwise, the unification *fails*.

DEFINITION 2 (Reduction). *An MRS σ **reduces to** a TFS A with respect to a grammar G (denoted $\langle A_1, \ldots, A_k \rangle \Rightarrow_G A$) iff there exists a rule $\rho \in \mathcal{R}$ such that $\sigma' \sqcup \rho = \tau$, where σ' is obtained by concatenating $\bot$ with σ, and A is the first element of τ. When G is understood from the context it is omitted.*

Reduction is an essential notion; it can be viewed as the bottom-up counterpart of derivation. When an MRS σ is taken as representing a (sentential) *form*, it can reduce to a TFS if there is a grammar rule ρ that licenses the reduction. When the body of ρ is unified with σ, the (possibly affected) head of ρ, B, is the reduced TFS.

For two functions f, g, over the same (set) domain, $f + g$ is defined as $\lambda I.f(I) \cup g(I)$. Let $\mathbb{N}_0$ denote the set $\{0, 1, 2, 3, \ldots\}$. Let ITEMS be the set $\{[w, i, A, j] \mid w \in \text{WORDS}^*, A \text{ is a TFS and } i, j \in \mathbb{N}_0\}$. Let $\mathcal{I} = 2^{\text{ITEMS}}$. Meta-variables x, y range over items and I – over sets of items. When $\mathcal{I}$ is ordered by set inclusion it forms a complete lattice with set union as a least upper bound (lub) operation. A function $T : \mathcal{I} \rightarrow \mathcal{I}$ is monotone if whenever $I_1 \subseteq I_2$, also $T(I_1) \subseteq T(I_2)$. It is continuous if for every chain $I_1 \subseteq I_2 \subseteq \cdots$, $T(\bigcup_{j<\omega} I_j) = \bigcup_{j<\omega} T(I_j)$. If a function T is monotone it has a least fixpoint (Tarski-Knaster theorem); if T is also continuous, the fixpoint can be obtained by iterative application of T to

[1] Grammars are displayed using a simple description language, where ':' denotes feature values and ',' denotes conjunction.

[2] Assume that in all the example grammars, the types s, n, v and vp are maximal and pairwise inconsistent.

the empty set (Kleene theorem): $lfp(T) = T \uparrow \omega$, where $T \uparrow 0 = \emptyset$ and $T \uparrow n = T(T \uparrow (n-1))$ when n is a successor ordinal and $\bigcup_{k<n}(T \uparrow n)$ when n is a limit ordinal.

2.2. Compositionality

Gaifman and Shapiro (1989) define compositional semantics as follows: let $\mathcal{P}$ be a class of programs (or program parts); let Ob be a function associating a set of objects, the observables, with a program $P \in \mathcal{P}$. Let Com be a class of composition functions over $\mathcal{P}$ such that for every n-ary function $f \in Com$ and every set of n programs $P_1, \ldots, P_n \in \mathcal{P}$, $f(P_1, \ldots, P_n) \in \mathcal{P}$. A *compositional* equivalence (with respect to Ob, Com) is an equivalence relation '$\equiv$' that preserves observables (i.e., whenever $P \equiv Q$, $Ob(P) = Ob(Q)$), and for every $f \in Com$, if for all $1 \leq i \leq n$, $P_i \equiv Q_i$ then $f(P_1, \ldots, P_n) \equiv f(Q_1, \ldots, Q_n)$. A semantic equivalence relation '$\equiv$' is *fully-abstract* if $P \equiv Q$ iff for all R, $Ob(P \cup R) = Ob(Q \cup R)$.

Given a program P, let $[\![P]\!]$ be the denotation of P. Let $\cup$ be a composition operator on programs, and $\bullet$ be a composition operator on denotations. A semantics is *commutative* if $\cup$ and $\bullet$ commute, that is, if $[\![P \cup Q]\!] = [\![P]\!] \bullet [\![Q]\!]$. If a semantics is commutative with respect to some operator then it is compositional (with respect to a singleton set containing the same operator).

2.3. Operational semantics

We follow the insight and notation of Shieber et al. (1995) and list a deductive system for parsing unification grammars:

DEFINITION 3 (Deductive parsing). *The deductive parsing system associated with a grammar $G = \langle \mathcal{R}, \mathcal{L}, A^s \rangle$ is defined over* Items *and is characterized by:*

Axioms:

$$\begin{array}{ll} [\epsilon, i, A, i] & \text{if } A \text{ is an } \epsilon\text{-rule in } \mathcal{R} \\ [a, i, A, i+1] & \text{if } A \in \mathcal{L}(a) \end{array}$$

Goals:

$$[w, 0, A^s, |w|]$$

Inference rules:

$$\frac{[w_1, i_1, A_1, j_1], \ldots, [w_k, i_k, A_k, j_k]}{[w_1 \cdots w_k, i, A, j]} \qquad \begin{array}{l} \text{if } j_l = i_{l+1} \text{ for } 1 \leq l < k, \\ i = i_1,\ j = j_k \text{ and} \\ \langle A_1, \ldots, A_k \rangle \Rightarrow_G A \end{array}$$

Notice that the domain of items is infinite, and in particular that the number of axioms is infinite. When an item $[w, i, A, j]$ can be deduced, applying k times the inference rules associated with a grammar G, we write $\vdash_G^k [w, i, A, j]$. When k is irrelevant it is omitted.

DEFINITION 4 (Operational semantics). *The* ***operational denotation*** *of a grammar G is $[\![G]\!]_{op} = \{x \mid \vdash_G x\}$. Consequently, $G_1 \equiv_{op} G_2$ iff $[\![G_1]\!]_{op} = [\![G_2]\!]_{op}$.*

We use the operational semantics to define the *language* generated by a grammar G. The language of a grammar G is $L(G) = \{\langle w, A\rangle \mid [w, 0, A, |w|] \in [\![G]\!]_{op}\}$. Notice that a language is not merely a set of strings; rather, each string is associated with a TFS through the deduction procedure. Note also that the start symbol A^s does not play a role in this definition; this is equivalent to assuming that the start symbol is always the most general TFS, $\bot$.

The most natural observable for a grammar would be its language, either as a set of strings or augmented by TFSs. Thus we take $Ob(G)$ to be $L(G)$ and by definition, the operational semantics '$[\![\cdot]\!]_{op}$' preserves observables.

2.4. Denotational semantics

As an alternative to the operational semantics discussed above, we consider in this section a denotational semantics through a fixpoint of a transformational operator associated with grammars. This is essentially similar to the definition of Pereira and Shieber (1984) and Carpenter (1992), pp. 204-206. We then show that the algebraic semantics is equivalent to the operational one.

Associate with a grammar G an operator T_G that, analogously to the immediate consequence operator of logic programming, can be thought of as a "parsing step" operator in the context of grammatical formalisms. For the following discussion fix a particular grammar $G = \langle \mathcal{R}, \mathcal{L}, A^s\rangle$.

DEFINITION 5. *Let $T_G : \mathcal{I} \to \mathcal{I}$ be a transformation on sets of items, where for every $I \subseteq$ Items, $[w, i, A, j] \in T_G(I)$ iff either*

- *there exist $y_1, \ldots, y_k \in I$ such that $y_l = [w_l, i_l, A_l, j_l]$ for $1 \leq l \leq k$ and $i_{l+1} = j_l$ for $1 \leq l < k$ and $i_1 = i$ and $j_k = j$ and $\langle A_1, \ldots, A_k\rangle \Rightarrow A$ and $w = w_1 \cdots w_k$; or*
- *$i = j$ and A is an ϵ-rule in G and $w = \epsilon$; or*
- *$i + 1 = j$ and $|w| = 1$ and $A \in \mathcal{L}(w)$.*

Since for every grammar G, T_G is monotone and continuous, the least fixpoint of T_G always exists and $lfp(T_G) = T_G \uparrow \omega$. Following the paradigm of logic programming languages, define a fixpoint semantics for unification grammars by taking the least fixpoint of the parsing step operator as the denotation of a grammar.

DEFINITION 6 (Fixpoint semantics). *The fixpoint denotation of a grammar G is $[\![G]\!]_{fp} = lfp(T_G)$. Consequently, $G_1 \equiv_{fp} G_2$ iff $lfp(T_{G_1}) = lfp(T_{G_2})$.*

The denotational definition is equivalent to the operational one:

THEOREM 1. *For $x \in$ ITEMS, $x \in lfp(T_G)$ iff $\vdash_G x$.*
Proof. See Wintner (2002), Theorem 2.

2.5. GRAMMAR UNION

While the operational and the denotational semantics defined above are standard for complete grammars, they are too coarse to serve as a model when the composition of grammars is concerned. When the denotation of a grammar is taken to be $[\![G]\!]_{op}$, important characteristics of the internal structure of the grammar are lost. To demonstrate the problem, we introduce a natural composition operator on unification grammars, namely union of the sets of rules (and the lexicons) in the composed grammars.

DEFINITION 7 (Grammar union). *If $G_1 = \langle \mathcal{R}_1, \mathcal{L}_1, A_1^s \rangle$ and $G_2 = \langle \mathcal{R}_2, \mathcal{L}_2, A_2^s \rangle$ are two grammars over the same signature, then the* **union** *of the two grammars, denoted $G_1 \cup G_2$, is a new grammar $G = \langle \mathcal{R}, \mathcal{L}, A^s \rangle$ such that $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$, $\mathcal{L} = \mathcal{L}_1 + \mathcal{L}_2$ and $A^s = A_1^s \sqcap A_2^s$.*

Figure 2 exemplifies the union operation on grammars. Observe that $G_1 \cup G_2 = G_2 \cup G_1$.

PROPOSITION 2. *The equivalence relation '$\equiv_{op}$' is not compositional with respect to $Ob, \{\cup\}$.*
Proof. See Wintner (2002), Theorem 3.

The implication of the above proposition is that while grammar union might be a natural, well defined syntactic operation on grammars, the standard semantics of grammars is too coarse to support it. Intuitively, this is because when a grammar G_1 includes a particular rule ρ that is inapplicable for reduction, this rule contributes nothing

$$
\begin{array}{ll}
G_1: & A^s = (cat:s) \\
 & (cat:s) \;\; \rightarrow \;\; (cat:n) \;\; (cat:vp) \\
 & \mathcal{L}(\text{John}) = \{(cat:n)\} \\
G_2: & A^s = (\bot) \\
 & (cat:vp) \;\; \rightarrow \;\; (cat:v) \\
 & (cat:vp) \;\; \rightarrow \;\; (cat:v) \;\; (cat:n) \\
 & \mathcal{L}(\text{sleeps}) = \mathcal{L}(\text{loves}) = \{(cat:v)\} \\
G_1 \cup G_2: & A^s = (cat:s) \\
 & (cat:s) \;\; \rightarrow \;\; (cat:n) \;\; (cat:vp) \\
 & (cat:vp) \;\; \rightarrow \;\; (cat:v) \\
 & (cat:vp) \;\; \rightarrow \;\; (cat:v) \;\; (cat:n) \\
 & \mathcal{L}(\text{John}) = \{(cat:n)\} \\
 & \mathcal{L}(\text{sleeps}) = \mathcal{L}(\text{loves}) = \{(cat:v)\}
\end{array}
$$

Figure 2. Grammar union

to the denotation of the grammar. But when G_1 is combined with some other grammar, G_2, ρ might be used for reduction in $G_1 \cup G_2$, where it can interact with the rules of G_2. The question to ask is, then, in what sense is a grammar a union of its rules? We suggest an alternative semantics for unification grammars that naturally supports compositionality.

3. A compositional, fully-abstract semantics

Associate the following operator with a unification grammar G:

DEFINITION 8. *Let $R_G : \mathcal{I} \rightarrow \mathcal{I}$ be a transformation on sets of items, where for every $I \subseteq$ ITEMS, $[w, i, A, j] \in R_G(I)$ iff there exist $y_1, \ldots, y_k \in I$ such that all the following hold:*

- $y_l = [w_l, i_l, A_l, j_l]$ *for* $1 \leq l \leq k$
- $i_{l+1} = j_l$ *for* $1 \leq l < k$, $i_1 = i$ *and* $j_k = j$
- $\langle A_1, \ldots, A_k \rangle \Rightarrow A$
- $w = w_1 \cdots w_k$.

The functional denotation of a grammar G is $[\![G]\!]_{fn} = (R_G + Id)^\omega = \Sigma_{n=0}^{\infty}(R_G + Id)^n$. Notice that R_G^ω is not $R_G \uparrow \omega$: the former is a function from sets of items to set of items; the latter is a set of items.

Observe that R_G is defined similarly to T_G (definition 5), ignoring the items added (by T_G) due to ϵ-rules and lexical items. If we define the set of items $Init_G$ to be those items that are added by T_G independently of the argument it operates on, then for every grammar G and every set of items I, $T_G(I) = R_G(I) \cup Init_G$. Relating the functional semantics to the fixpoint one, we follow Lassez and Maher (1984) in proving that the fixpoint of the grammar transformation operator can be computed by applying the functional semantics to the set $Init_G$.

DEFINITION 9. *For every grammar $G = \langle \mathcal{R}, \mathcal{L}, A^s \rangle$, let*

$$Init_G = \{[\epsilon, i, A, i] \mid A \text{ is an } \epsilon\text{-rule in } G\} \cup \{[a, i, A, i+1] \mid A \in \mathcal{L}(a)\}.$$

THEOREM 3. *For every grammar G, $(R_G + Id)^\omega(Init_G) = lfp(T_G)$.*
Proof. See Wintner (2002), Theorem 6.

The choice of '$[\![\cdot]\!]_{fn}$' as the semantics calls for a different notion of observables:

DEFINITION 10 (Observables). *The **observables** of a grammar $G = \langle \mathcal{R}, \mathcal{L}, A^s \rangle$ with respect to an input set of items I are $Ob_I(G) = \{\langle w, A \rangle \mid [w, 0, A, |w|] \in [\![G]\!]_{fn}(I)\}$.*

COROLLARY 4. *The semantics '$[\![\cdot]\!]_{fn}$' is Ob_I-preserving: if $G_1 \equiv_{fn} G_2$ then for every I, $Ob_I(G_1) = Ob_I(G_2)$.*

The above definition corresponds to the previous one in a natural way: when the input is taken to be $Init_G$, the observables of a grammar are its language.

THEOREM 5. *For every grammar G, $L(G) = Ob_{Init_G}(G)$.*
Proof. See Wintner (2002), Theorem 7.

To show that '$[\![\cdot]\!]_{fn}$' is compositional we must define an operator for combining denotations. Define $[\![G_1]\!]_{fn} \bullet [\![G_2]\!]_{fn}$ to be $([\![G_1]\!]_{fn} + [\![G_2]\!]_{fn})^\omega$. Then '$[\![\cdot]\!]_{fn}$' is commutative with respect to '$\bullet$' and '$\cup$'.

THEOREM 6. $[\![G_1 \cup G_2]\!]_{fn} = [\![G_1]\!]_{fn} \bullet [\![G_2]\!]_{fn}$.
Proof. See Wintner (2002), Theorem 10.

Since '$[\![\cdot]\!]_{fn}$' is commutative, it is also compositional. Finally, this semantics is also fully-abstract:

THEOREM 7. *The semantics '$[\![\cdot]\!]_{fn}$' is fully-abstract: for every two grammars G_1 and G_2, if for every grammar G and set of items I, $Ob_I(G_1 \cup G) = Ob_I(G_2 \cup G)$, then $G_1 \equiv_{fn} G_2$.*

Proof. See Wintner (2002), Theorem 11.

4. Conclusions

This short note presents an alternative definition for the semantics of unification-based linguistic formalisms which is both compositional and fully-abstract (with respect to grammar union, a simple syntactic combination operations on grammars). This is a straight-forward extension of the results reported in Wintner (2002) from context-free grammars to unification formalisms.

The functional semantics '$[\![\cdot]\!]_{fn}$' defined here assigns to a grammar a function which reflects the (possibly infinite) successive application of grammar rules, viewing the lexicon as input to the parsing process. We believe that this is a key to modularity in grammar design. A grammar module has to define a set of items that it "exports", and a set of items that can be "imported", in a similar way to the declaration of interfaces in programming languages. We are currently working out the details of such a definition. An immediate application will facilitate the implementation of grammar development systems that support modularity in a clear, mathematically sound way.

The results reported here can be extended in various directions. First, we are only concerned in this work with one composition operator, grammar union. But alternative operators are possible, too. In particular, it would be interesting to define an operator which combines the information encoded in two grammar rules, for example by unifying the rules. Such an operator would facilitate a separate development of grammars along a different axis: one module can define the syntactic component of a grammar while another module would account for the semantics. The composition operator will unify each rule of one module with an associated rule in the other. It remains to be seen whether the grammar semantics we define here is compositional and fully-abstract with respect to such an operator.

A different extension of these results should provide for a distribution of the type hierarchy among several grammar modules. While we assume in this work that all grammars are defined over a given signature, it is more realistic to assume separate, interacting signatures. We hope to be able to explore these directions in the future.

Acknowledgements

This paper is a revised version of Wintner (1999). I am grateful to Nissim Francez for commenting on an earlier version. This work was supported by the Israeli Science Foundation (grant no. 136/01).

References

Carpenter, B.: 1991, 'Typed Feature Structures: A Generalization of First-Order Terms'. In: V. Saraswat and U. Kazunori (eds.): *Logic Programming – Proceedings of the 1991 International Symposium*. Cambridge, MA, pp. 187–201, MIT Press.

Carpenter, B.: 1992, *The Logic of Typed Feature Structures*, Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.

Gaifman, H. and E. Shapiro: 1989, 'Fully abstract compositional semantics for logic programming'. In: *16th Annual ACM Symposium on Principles of Logic Programming*. Austin, Texas, pp. 134–142.

Lassez, J.-L. and M. J. Maher: 1984, 'Closures and fairness in the semantics of programming logic'. *Theoretical computer science* **29**, 167–184.

Pereira, F. C. N. and S. M. Shieber: 1984, 'The semantics of grammar formalisms seen as computer languages'. In: *Proceedings of the 10th international conference on computational linguistics and the 22nd annual meeting of the association for computational linguistics*. Stanford, CA, pp. 123–129.

Shieber, S., Y. Schabes, and F. Pereira: 1995, 'Principles and Implementation of Deductive Parsing'. *Journal of Logic Programming* **24**(1-2), 3–36.

Shieber, S. M.: 1992, *Constraint-Based Grammar Formalisms*. Cambridge, Mass.: MIT Press.

Sikkel, K.: 1997, *Parsing Schemata*, Texts in Theoretical Computer Science – An EATCS Series. Berlin: Springer Verlag.

Wintner, S.: 1999, 'Compositional Semantics for Linguistic Formalisms'. In: *Proceedings of ACL'99, the 37th Annual Meeting of the Association for Computational Linguistics*. pp. 96–103.

Wintner, S.: 2002, 'Modular Context-Free Grammars'. *Grammars* **5**(1), 41–63.

Wintner, S. and N. Francez: 1999, 'Off-line Parsability and the well-foundedness of Subsumption'. *Journal of Logic, Language and Information* **8**(1), 1–16.