Compositional Semantics for Linguistic Formalisms

Shuly Wintner*
Institute for Research in Cognitive Science
University of Pennsylvania
3401 Walnut St., Suite 400A
Philadelphia, PA 19018
shuly@linc.cis.upenn.edu

Abstract

In what sense is a grammar the union of its rules? This paper adapts the notion of *composition*, well developed in the context of programming languages, to the domain of linguistic formalisms. We study alternative definitions for the semantics of such formalisms, suggesting a denotational semantics that we show to be compositional and fully-abstract. This facilitates a clear, mathematically sound way for defining grammar modularity.

1 Introduction

Developing large scale grammars for natural languages is a complicated task, and the problems grammar engineers face when designing broad-coverage grammars are reminiscent of those tackled by software engineering (Erbach and Uszkoreit, 1990). Viewing contemporary linguistic formalisms as very high level declarative programming languages, a *grammar* for a natural language can be viewed as a *program*. It is therefore possible to adapt methods and techniques of software engineering to the domain of natural language formalisms. We believe that any advances in grammar engineering must be preceded by a more theoretical work, concentrating on the semantics of grammars. This view reflects the situation in logic programming, where developments in alternative definitions for predicate logic semantics led to implementations of various program composition operators (Bugliesi et al., 1994).

This paper suggests a denotational semantics for unification-based linguistic formalisms and shows that it is compositional and fully-abstract. This facilitates a clear, mathematically sound way for defining grammar modularity. While most of the results we report on are probably not surprising, we believe that it is important to derive them directly for linguistic formalisms for two reasons. First, practitioners of linguistic formalisms usually do not view them as instances of a general logic programming framework, but rather as first-class programming environments which deserve independent study. Second, there are some crucial differences between contemporary linguistic formalisms and, say, Prolog: the basic elements — typed feature-structures — are more general than first-order terms, the notion of unification is different, and computations amount to parsing, rather than SLD-resolution. The fact that we can derive similar results in this new domain is encouraging, and should not be considered trivial.

Analogously to logic programming languages, the denotation of grammars can be defined using various techniques. We review alternative approaches, operational and denotational, to the semantics of linguistic formalisms in section 2 and show that they are "too crude" to support grammar composition. Section 3 presents an alternative semantics, shown to be compositional (with respect to grammar union, a simple syntactic combination operation on grammars). However, this definition is "too fine": in section 4 we present an adequate, compositional and fully-abstract semantics for linguistic formalisms. For lack of space, some proofs are omitted; an extended version is available as a technical report (Wintner, 1999).

2 Grammar semantics

Viewing grammars as formal entities that share many features with computer programs, it is

*I am grateful to Nissim Francez for commenting on an earlier version of this paper. This work was supported by an IRCS Fellowship and NSF grant SBR 8920230.

natural to consider the notion of *semantics* of unification-based formalisms. 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). We show that these definitions are equivalent and that none of them supports compositionality.

2.1 Basic notions

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. *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 underlying type hierarchy.[2]

The definition of unification is lifted to MRSs: let σ, ρ be two MRSs of the same length; the

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

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

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

Figure 1: An example grammar, G

unification of σ and ρ, denoted $\sigma \sqcup \rho$, is the most general MRS that is subsumed by both σ and ρ, if it exists. Otherwise, the unification *fails*.

Definition 2. *An MRS $\langle A_1, \ldots, A_k \rangle$ **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 $\langle B, B_1, \ldots, B_k \rangle = \rho \sqcup \langle \bot, A_1, \ldots, A_k \rangle$ and $B \sqsubseteq A$. When G is understood from the context it is omitted. Reduction can be viewed as the bottom-up counterpart of derivation.*

If f, g, are functions over the same (set) domain, $f + g$ is $\lambda I.f(I) \cup g(I)$. Let ITEMS $= \{[w, i, A, j] \mid w \in \text{WORDS}^*, A$ is a TFS and $i, j \in \{0, 1, 2, 3, \ldots\}\}$. 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 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.

When the semantics of programming languages are concerned, a notion of *observables* is called for: *Ob* is a function associating a set of objects, the observables, with every program. The choice of semantics induces a natural equivalence operator on grammars: given a semantics '$[\![\cdot]\!]$', $G_1 \equiv G_2$ iff $[\![G_1]\!] = [\![G_2]\!]$. An essential requirement of any semantic equivalence is that it

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

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

be *correct* (observables-preserving): if $G_1 \equiv G_2$, then $Ob(G_1) = Ob(G_2)$.

Let '$\cup$' be a composition operation on grammars and '$\bullet$' a combination operator on denotations. A (correct) semantics '$[\![\cdot]\!]$' is *compositional* (Gaifman and Shapiro, 1989) if whenever $[\![G_1]\!] = [\![G_2]\!]$ and $[\![G_3]\!] = [\![G_4]\!]$, also $[\![G_1 \cup G_3]\!] = [\![G_2 \cup G_4]\!]$. A semantics is *commutative* (Brogi et al., 1992) if $[\![G_1 \cup G_2]\!] = [\![G_1]\!] \bullet [\![G_2]\!]$. This is a stronger notion than compositionality: if a semantics is commutative with respect to some operator then it is compositional.

2.2 An operational semantics

As Van Emden and Kowalski (1976) note, "to define an operational semantics for a programming language is to define an implementational independent interpreter for it. For predicate logic the proof procedure behaves as such an interpreter." Shieber et al. (1995) view parsing as a deductive process that proves claims about the grammatical status of strings from assumptions derived from the grammar. We follow their insight and notation and list a deductive system for parsing unification-based grammars.

Definition 3. *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: $[a, i, A, i+1]$ *if* $B \in \mathcal{L}(a)$ *and* $B \sqsubseteq A$;
$[\epsilon, i, A, i]$ *if* B *is an* ϵ*-rule in* $\mathcal{R}$ *and* $B \sqsubseteq A$

Goals: $[w, 0, A, |w|]$ *where* $A \sqsupseteq A^s$

Inference rules:

$$\frac{[w_1, i_1, A_1, j_1], \ldots, [w_k, i_k, A_k, j_k]}{[w_1 \cdots w_k, i, A, j]}$$

if $j_l = i_{l+1}$ *for* $1 \leq l < k$ *and* $i = i_1$ *and* $j = j_k$ *and* $\langle A_1, \ldots, A_k \rangle \Rightarrow_G A$

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 the number of inference steps is irrelevant it is omitted. Notice that the domain of items is infinite, and in particular that the number of axioms is infinite. Also, notice that the goal is to deduce a TFS which is subsumed by the start symbol, and when TFSs can be cyclic, there can be infinitely many such TFSs (and, hence, goals) – see Wintner and Francez (1999).

Definition 4. *The* ***operational denotation*** *of a grammar G is $[\![G]\!]_{op} = \{x \mid \vdash_G x\}$. $G_1 \equiv_{op} G_2$ iff $[\![G_1]\!]_{op} = [\![G_2]\!]_{op}$.*

We use the operational semantics to define the *language* generated by a grammar G: $L(G) = \{\langle w, A \rangle \mid [w, 0, A, |w|] \in [\![G]\!]_{op}\}$. 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.

2.3 Denotational semantics

In this section we consider 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 denotational 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 $i_1 = 1$ 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 B is an ϵ-rule in G and $B \sqsubseteq A$ and $w = \epsilon$; or*
- *$i + 1 = j$ and $|w| = 1$ and $B \in \mathcal{L}(w)$ and $B \sqsubseteq A$.*

For every grammar G, T_G is monotone and continuous, and hence its least fixpoint exists and $lfp(T_G) = T_G \uparrow \omega$. Following the paradigm

of logic programming languages, define a fixpoint semantics for unification-based grammars by taking the least fixpoint of the parsing step operator as the denotation of a grammar.

Definition 6. *The* ***fixpoint denotation*** *of a grammar G is $[\![G]\!]_{fp} = lfp(T_G)$. $G_1 \equiv_{fp} G_2$ 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 $\vdash_G x$.*

The proof is that $[w, i, A, j] \in T_G \uparrow n$ iff $\vdash_G^n [w, i, A, j]$, by induction on n.

Corollary 2. *The relation '$\equiv_{fp}$' is correct: whenever $G_1 \equiv_{fp} G_2$, also $Ob(G_1) = Ob(G_2)$.*

2.4 Compositionality

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 grammars, namely union of the sets of rules (and the lexicons) in the composed grammars.

Definition 7. *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 grammar union. Observe that for every G, G', $G \cup G' = G' \cup G$.

Proposition 3. *The equivalence relation '$\equiv_{op}$' is not compositional with respect to $Ob, \{\cup\}$.*

Proof. Consider the grammars in figure 2. $[\![G_3]\!]_{op} = [\![G_4]\!]_{op} = \{[\text{"loves"}, i, (cat : v), i+1] \mid i \geq 0\}$ but for $I = \{[\text{"John loves John"}, i, (cat : s), i+3 \mid i \geq 0\}$, $I \subseteq [\![G_1 \cup G_4]\!]_{op}$ whereas $I \not\subseteq [\![G_1 \cup G_3]\!]_{op}$. Thus $G_3 \equiv_{op} G_4$ but $(G_1 \cup G_3) \not\equiv_{op} (G_1 \cup G_4)$, hence '$\equiv_{op}$' is not compositional with respect to $Ob, \{\cup\}$. □

G_1: $A^s = (cat : s)$
$(cat : s) \rightarrow (cat : n) \quad (cat : vp)$
$\mathcal{L}(\text{John}) = \{(cat : n)\}$

G_2: $A^s = (\bot)$
$(cat : vp) \rightarrow (cat : v)$
$(cat : vp) \rightarrow (cat : v) \quad (cat : n)$
$\mathcal{L}(\text{sleeps}) = \mathcal{L}(\text{loves}) = \{(cat : v)\}$

G_3: $A^s = (\bot)$
$\mathcal{L}(\text{loves}) = \{(cat : v)\}$

G_4: $A^s = (\bot)$
$(cat : vp) \rightarrow (cat : v) \quad (cat : n)$
$\mathcal{L}(\text{loves}) = \{(cat : v)\}$

$G_1 \cup G_2$: $A^s = (cat : s)$
$(cat : s) \rightarrow (cat : n) \quad (cat : vp)$
$(cat : vp) \rightarrow (cat : v)$
$(cat : vp) \rightarrow (cat : v) \quad (cat : n)$
$\mathcal{L}(\text{John}) = \{(cat : n)\}$
$\mathcal{L}(\text{sleeps}) = \mathcal{L}(\text{loves}) = \{(cat : v)\}$

$G_1 \cup G_3$: $A^s = (cat : s)$
$(cat : s) \rightarrow (cat : n) \quad (cat : vp)$
$\mathcal{L}(\text{John}) = \{(cat : n)\}$
$\mathcal{L}(\text{loves}) = \{(cat : v)\}$

$G_1 \cup G_4$: $A^s = (cat : s)$
$(cat : s) \rightarrow (cat : n) \quad (cat : vp)$
$(cat : vp) \rightarrow (cat : v) \quad (cat : n)$
$\mathcal{L}(\text{John}) = \{(cat : n)\}$
$\mathcal{L}(\text{loves}) = \{(cat : v)\}$

Figure 2: Grammar union

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 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. We suggest an alternative, fixpoint based semantics for unification based grammars that naturally supports compositionality.

3 A compositional semantics

To overcome the problems delineated above, we follow Mancarella and Pedreschi (1988) in considering the grammar transformation operator itself (rather than its fixpoint) as the denota-

tion of a grammar.

Definition 8. *The **algebraic denotation** of G is $[\![G]\!]_{al} = T_G$. $G_1 \equiv_{al} G_2$ iff $T_{G_1} = T_{G_2}$.*

Not only is the algebraic semantics compositional, it is also commutative with respect to grammar union. To show that, a composition operation on denotations has to be defined, and we follow Mancarella and Pedreschi (1988) in its definition:

$$T_{G_1} \bullet T_{G_2} = \lambda I.T_{G_1}(I) \cup T_{G_2}(I)$$

Theorem 4. *The semantics '$\equiv_{al}$' is commutative with respect to grammar union and '$\bullet$': for every two grammars G_1, G_2, $[\![G_1]\!]_{al} \bullet [\![G_2]\!]_{al} = [\![G_1 \cup G_2]\!]_{al}$.*

Proof. It has to be shown that for every set of items I, $T_{G_1 \cup G_2}(I) = T_{G_1}(I) \cup T_{G_2}(I)$.

- if $x \in T_{G_1}(I) \cup T_{G_2}(I)$ then either $x \in T_{G_1}(I)$ or $x \in T_{G_2}(I)$. From the definition of grammar union, $x \in T_{G_1 \cup G_2}(I)$ in any case.
- if $x \in T_{G_1 \cup G_2}(I)$ then x can be added by either of the three clauses in the definition of T_G.
  - if x is added by the first clause then there is a rule $\rho \in \mathcal{R}_1 \cup \mathcal{R}_2$ that licenses the derivation through which x is added. Then either $\rho \in \mathcal{R}_1$ or $\rho \in \mathcal{R}_2$, but in any case ρ would have licensed the same derivation, so either $x \in T_{G_1}(I)$ or $x \in T_{G_2}(I)$.
  - if x is added by the second clause then there is an ϵ-rule in $G_1 \cup G_2$ due to which x is added, and by the same rationale either $x \in T_{G_1}(I)$ or $x \in T_{G_2}(I)$.
  - if x is added by the third clause then there exists a lexical category in $\mathcal{L}_1 \cup \mathcal{L}_2$ due to which x is added, hence this category exists in either $\mathcal{L}_1$ or $\mathcal{L}_2$, and therefore $x \in T_{G_1}(I) \cup T_{G_2}(I)$.

□

Since '$\equiv_{al}$' is commutative, it is also compositional with respect to grammar union. Intuitively, since T_G captures only one step of the computation, it cannot capture interactions among different rules in the (unioned) grammar, and hence taking T_G to be the denotation of G yields a compositional semantics.

The T_G operator reflects the structure of the grammar better than its fixpoint. In other words, the equivalence relation induced by T_G is finer than the relation induced by $lfp(T_G)$. The question is, how fine is the '$\equiv_{al}$' relation? To make sure that a semantics is not *too* fine, one usually checks the reverse direction.

Definition 9. *A **fully-abstract** equivalence relation '$\equiv$' is such that $G_1 \equiv G_2$ iff for all G, $Ob(G_1 \cup G) = Ob(G_2 \cup G)$.*

Proposition 5. *The semantic equivalence relation '$\equiv_{al}$' is not fully abstract.*

Proof. Let G_1 be the grammar

$$\begin{array}{lcl} A_1^s & = & \bot, \\ \mathcal{L}_1 & = & \emptyset, \\ \mathcal{R}_1 & = & \{(cat:s) \rightarrow (cat:np)\ (cat:vp), \\ & & (cat:np) \rightarrow (cat:np)\} \end{array}$$

and G_2 be the grammar

$$\begin{array}{lcl} A_2^s & = & \bot, \\ \mathcal{L}_2 & = & \emptyset, \\ \mathcal{R}_2 & = & \{(cat:s) \rightarrow (cat:np)\ (cat:vp)\} \end{array}$$

- $G_1 \not\equiv_{al} G_2$: because for $I = \{[\text{“John loves Mary”}, 6, (cat : np), 9]\}$, $T_{G_1}(I) = I$ but $T_{G_2}(I) = \emptyset$
- for all G, $Ob(G \cup G_1) = Ob(G \cup G_2)$. The only difference between $G \cup G_1$ and $G \cup G_2$ is the presence of the rule $(cat : np) \rightarrow (cat : np)$ in the former. This rule can contribute nothing to a deduction procedure, since any item it licenses must already be deducible. Therefore, any item deducible with $G \cup G_1$ is also deducible with $G \cup G_2$ and hence $Ob(G \cup G_1) = Ob(G \cup G_2)$.

□

A better attempt would have been to consider, instead of T_G, the following operator as the denotation of G: $[\![G]\!]_{id} = \lambda I.T_G(I) \cup I$. In other words, the semantics is $T_G + Id$, where Id is the identity operator. Unfortunately, this does not solve the problem, as '$[\![\cdot]\!]_{id}$' is still not fully-abstract.

4 A fully abstract semantics

We have shown so far that '$[\![\cdot]\!]_{fp}$' is not compositional, and that '$[\![\cdot]\!]_{id}$' is compositional but not fully abstract. The "right" semantics, therefore, lies somewhere in between: since the choice of semantics induces a natural equivalence on grammars, we seek an equivalence that is cruder than '$[\![\cdot]\!]_{id}$' but finer than '$[\![\cdot]\!]_{fp}$'. In this section we adapt results from Lassez and Maher (1984) and Maher (1988) to the domain of unification-based linguistic formalisms.

Consider the following semantics for logic programs: rather than taking the operator associated with the entire program, look only at the rules (excluding the facts), and take the meaning of a program to be the function that is obtained by an infinite applications of the operator associated with the rules. In our framework, this would amount to associating the following operator with a grammar:

Definition 10. *Let $R_G : \mathcal{I} \to \mathcal{I}$ 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 $y_l = [w_l, i_l, A_l, j_l]$ for $1 \leq l \leq k$ and $i_{l+1} = j_l$ for $1 \leq l < k$ and $i_1 = 1$ and $j_k = j$ and $\langle A_1, \ldots, A_k \rangle \Rightarrow A$ and $w = w_1 \cdots w_k$.*

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 11. *For $G = \langle \mathcal{R}, \mathcal{L}, A^s \rangle$, $Init_G = \{[\epsilon, i, A, i] \mid B$ is an ϵ-rule in G and $B \sqsubseteq A\} \cup \{[a, i, A, i+1] \mid B \in \mathcal{L}(a)$ for $B \sqsubseteq A\}$*

Theorem 6. *For every grammar G,*

$$(R_G + Id)^\omega(Init_G) = lfp(T_G)$$

Proof. We show that for every n, $(T_G + Id) \uparrow n = (\Sigma_{k=0}^{n-1}(R_G + Id)^k)(Init_G)$ by induction on n.
For $n = 1$, $(T_G + Id) \uparrow 1 = (T_G + Id)((T_G + Id) \uparrow 0) = (T_G + Id)(\emptyset)$. Clearly, the only items added by T_G are due to the second and third clauses of definition 5, which are exactly $Init_G$. Also, $(\Sigma_{k=0}^{0}(R_G + Id)^k)(Init_G) = (R_G + Id)^0(Init_G) = Init_G$.
Assume that the proposition holds for $n-1$, that is, $(T_G + Id) \uparrow (n-1) = (\Sigma_{k=0}^{n-2}(R_G + Id)^k)(Init_G)$. Then

$(T_G + Id) \uparrow n =$
  definition of $\uparrow$
$(T_G + Id)((T_G + Id) \uparrow (n-1)) =$
  by the induction hypothesis
$(T_G + Id)((\Sigma_{k=0}^{n-2}(R_G + Id)^k)(Init_G)) =$
  since $T_G(I) = R_G(I) \cup Init_G$
$(R_G + Id)((\Sigma_{k=0}^{n-2}(R_G + Id)^k)(Init_G)) \cup Init_G =$
$(R_G + Id)((\Sigma_{k=0}^{n-2}(R_G + Id)^k)(Init_G)) =$
$(\Sigma_{k=0}^{n-1}(R_G + Id)^k)(Init_G)$

Hence $(R_G + Id)^\omega(Init_G) = (T_G + Id) \uparrow \omega = lfp(T_G)$. □

The choice of '$[\![\cdot]\!]_{fn}$' as the semantics calls for a different notion of observables. The denotation of a grammar is now a function which reflects an infinite number of applications of the grammar's rules, but completely ignores the ϵ-rules and the lexical entries. If we took the observables of a grammar G to be $L(G)$ we could in general have $[\![G_1]\!]_{fn} = [\![G_2]\!]_{fn}$ but $Ob(G_1) \neq Ob(G_2)$ (due to different lexicons), that is, the semantics would not be correct. However, when the lexical entries in a grammar (including the ϵ-rules, which can be viewed as empty categories, or the lexical entries of traces) are taken as *input*, a natural notion of observables preservation is obtained. To guarantee correctness, we define the observables of a grammar G *with respect to a given input*.

Definition 12. *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 7. *The semantics '$[\![\cdot]\!]_{fn}$' is correct: 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 8. *For all G, $L(G) = Ob_{Init_G}(G)$.*

Proof.

$L(G) =$
  definition of $L(G)$
$\{\langle w, A\rangle \mid [w, 0, A, |w|] \in [\![G]\!]_{op}\} =$
  definition 4
$\{\langle w, A\rangle \mid \vdash_G [w, 0, A, |w|]\} =$
  by theorem 1
$\{\langle w, A\rangle \mid [w, 0, A, |w|] \in lfp(T_G)\} =$
  by theorem 6
$\{\langle w, A\rangle \mid [w, 0, A, |w|] \in [\![G]\!]_{fn}(Init_G)\} =$
  by definition 12
$Ob_{Init_G}(G)$

□

To show that the semantics '$[\![\cdot]\!]_{fn}$' is compositional we must define an operator for combining denotations. Unfortunately, the simplest operator, '+', would not do. However, a different operator does the job. Define $[\![G_1]\!]_{fn} \bullet [\![G_2]\!]_{fn}$ to be $([\![G_1]\!]_{fn} + [\![G_2]\!]_{fn})^\omega$. Then '$[\![\cdot]\!]_{fn}$' is commutative (and hence compositional) with respect to '$\bullet$' and '$\cup$'.

Theorem 9. $[\![G_1 \cup G_2]\!]_{fn} = [\![G_1]\!]_{fn} \bullet [\![G_2]\!]_{fn}$.

The proof is basically similar to the case of logic programming (Lassez and Maher, 1984) and is detailed in Wintner (1999).

Theorem 10. *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 $G_1 \equiv_{fn} G_2$.*

The proof is constructive: assuming that $G_1 \not\equiv_{fn} G_2$, we show a grammar G (which depends on G_1 and G_2) such that $Ob_I(G_1 \cup G) \neq Ob_I(G_2 \cup G)$. For the details, see Wintner (1999).

5 Conclusions

This paper discusses alternative definitions for the semantics of unification-based linguistic formalisms, culminating in one that is both compositional and fully-abstract (with respect to grammar union, a simple syntactic combination operations on grammars). This is mostly an adaptation of well-known results from logic programming to the framework of unification-based linguistic formalisms, and it is encouraging to see that the same choice of semantics which is compositional and fully-abstract for Prolog turned out to have the same desirable properties in our domain.

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.

References

Antonio Brogi, Evelina Lamma, and Paola Mello. 1992. Compositional model-theoretic semantics for logic programs. *New Generation Computing*, 11:1–21.

Michele Bugliesi, Evelina Lamma, and Paola Mello. 1994. Modularity in logic programming. *Journal of Logic Programming*, 19,20:443–502.

Bob Carpenter. 1991. Typed feature structures: A generalization of first-order terms. In Vijai Saraswat and Ueda Kazunori, editors, *Logic Programming – Proceedings of the 1991 International Symposium*, pages 187–201, Cambridge, MA. MIT Press.

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

Gregor Erbach and Hans Uszkoreit. 1990. Grammar engineering: Problems and prospects. CLAUS report 1, University of the Saarland and German research center for Artificial Intelligence, July.

Haim Gaifman and Ehud Shapiro. 1989. Fully abstract compositional semantics for logic programming. In *16th Annual ACM Symposium on Principles of Logic Programming*, pages 134–142, Austin, Texas, January.

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

M. J. Maher. 1988. Equivalences of logic programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 16, pages 627–658. Morgan Kaufmann Publishers, Los Altos, CA.

Paolo Mancarella and Dino Pedreschi. 1988. An algebra of logic programs. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming: Proceedings of the Fifth international conference and symposium*, pages 1006–1023, Cambridge, Mass. MIT Press.

Fernando C. N. Pereira and Stuart 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*, pages 123–129, Stanford, CA, July.

Stuart Shieber, Yves Schabes, and Fernando Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1-2):3–36, July/August.

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

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

M. H. Van Emden and Robert A. Kowalski. 1976. The semantics of predicate logic as a programming language. *Journal of the Association for Computing Machinery*, 23(4):733–742, October.

Shuly Wintner and Nissim Francez. 1999. Off-line parsability and the well-foundedness of subsumption. *Journal of Logic, Language and Information*, 8(1):1–16, January.

Shuly Wintner. 1999. Compositional semantics for linguistic formalisms. IRCS Report 99-05, Institute for Research in Cognitive Science, University of Pennsylvania, 3401 Walnut St., Suite 400A, Philadelphia, PA 19018.