Modularized Context-Free Grammars

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

Given two context-free grammars (CFGs), G_1 and G_2, the language generated by the union of the grammars is *not* the union of the languages generated by each grammar: $L(G_1 \cup G_2) \neq L(G_1) \cup L(G_2)$. In order to account for modularity of grammars, another way of defining the meaning of grammars is needed. This paper adapts results from the semantics of logic programming languages to CFGs. We discuss alternative approaches for defining the denotation of a grammar, culminating in one which we show to be both compositional and fully-abstract. We then show how grammar modules can be defined such that their semantics retains these desirable properties.

1 Introduction

This paper provides the mathematical infrastructure required for defining *modules* in context-free grammars (CFGs). Our departure point is the belief 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 the semantics of predicate logic led to implementations of various program composition operators (Bugliesi et al., 1994). Viewing linguistic formalisms as declarative programming languages, it is possible to adapt methods and techniques of software engineering to the domain of natural language formalisms.

In section 2 we survey two approaches, operational and denotational, to the semantics of grammars. We show that they are equivalent and that both are not compositional. We experiment with alternative semantics in section 3, eventually getting at one that is both compositional and fully-abstract. We then discuss grammar modularity in section 4, defining grammar modules and showing that the chosen semantics is compositional for modular CFGs as well.

2 Semantics of CFGs

2.1 Basics

Grammars are defined over a fixed, non-empty, finite set WORDS of *words*. Meta-variables a and w range over WORDS and strings of words (elements of WORDS*), respectively. A *context-free grammar* is a quadruple $\langle V, \Sigma, P, A^s \rangle$, where V is a finite set of non-terminal symbols, or *categories*; $\Sigma : \text{WORDS} \to 2^V$ is a total function, a *lexicon*, associating a finite, possibly empty set of categories with each word; P is a finite set of *production rules*, each an element of $V \times V^*$; and $A^s \subseteq V$ is a set of categories, the *start symbols*.[1] Meta-variables A, S range over categories and σ, ρ — over *forms*, (possibly empty) sequences thereof. A rule $(A, (A_1, \ldots, A_n))$ is written $A \to A_1, \ldots, A_n$, and A is its *head*. If $n = 0$ the rule is an *ϵ-rule* (sometimes denoted simply A). $\mathbb{N}_0$ denotes the set $\{0, 1, 2, \ldots\}$. For two functions f, g, over the same (set) domain, $f + g$ is defined as $\lambda I. f(I) \cup g(I)$.

Figure 1 depicts an example grammar. V is the smallest set containing all the categories

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

[1] It is easy to see how this definition corresponds to the standard one, where rules can incorporate terminals directly and only one start symbol is present. We deviate from the standard definition to facilitate the definition of modules. Note that the definition allows *empty* grammars (where $V = P = \bigcup_{a \in \text{WORDS}} \Sigma(a) = \emptyset$).

that occur in P and Σ.

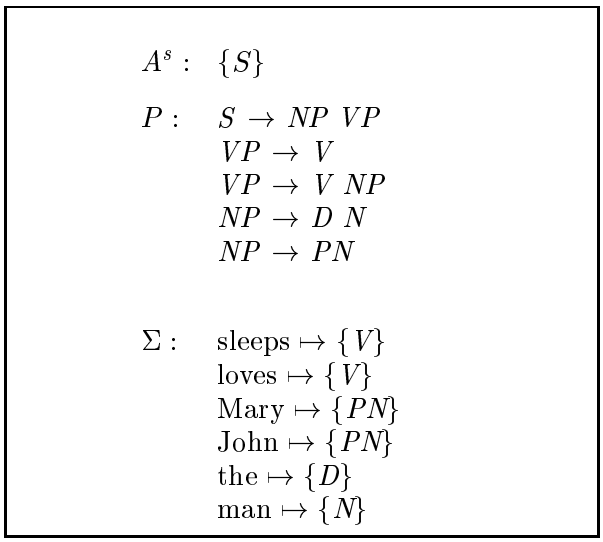

Figure 1: An example grammar

A form σ *derives* a form σ' in a grammar $G = \langle V, \Sigma, P, A^s \rangle$ (denoted $\sigma \Rightarrow_G \sigma'$) iff $\sigma = \sigma_l A \sigma_r$ and $\sigma' = \sigma_l A_1 \cdots A_n \sigma_r$ and $A \to A_1 \cdots A_n \in P$. The reflexive transitive closure of '$\Rightarrow_G$' is '$\stackrel{*}{\Rightarrow}_G$'. The *language* of a category A in grammar G, $L_A(G)$, is $\{a_1 \cdots a_n \mid A \stackrel{*}{\Rightarrow}_G A_1, \ldots, A_n$ and for every $1 \leq i \leq n$, $A_i \in \Sigma(a_i)\}$. The *language* of a grammar G, $L(G)$, is $\bigcup_{S \in A^s} L_S(G)$.

Fix a grammar $G = \langle V, \Sigma, P, A^s \rangle$. Let ITEMS be the set $\{[w, i, A, j] \mid w \in \text{WORDS}^*, A \in V$ 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 $T : \mathcal{I} \to \mathcal{I}$ 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 $\bigcup_{k<n}(T \uparrow k)$ when n is a limit ordinal.

2.2 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. Following this insight, a deductive system for parsing CFGs can serve as a method for defining their operational semantics.

Definition 1. *The **deductive parsing** system associated with a grammar $G = \langle V, \Sigma, P, A^s \rangle$ is defined over* ITEMS *and is characterized by:*

Axioms: $[\epsilon, i, A, i]$ *if A is an ϵ-rule in P;*
$[a, i, A, i+1]$ *if $A \in \Sigma(a)$*

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

Inference rules:

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

if $j_l = i_{l+1}$ for $1 \leq l < k$ and $i = i_1$ and $j = j_k$ and $A \to A_1, \ldots, A_k \in P$

When an item $[w, i, A, j]$ can be deduced, applying k times the inference rules associated with a grammar G, we write $G \vdash^k [w, i, A, j]$. When the number of inference steps is irrelevant we omit it. The operational denotation of a grammar G, then, is $[\![G]\!]_{op} = \{x \mid G \vdash x\}$. It is easy to show how the operational semantics corresponds to the language of a grammar:

Theorem 1. *$w \in L_A(G)$ iff $G \vdash [w, 0, A, |w|]$.*

2.3 Denotational semantics

As an alternative to the operational semantics described above, Pereira and Shieber (1984) define denotational semantics through a fixpoint of a transformational operator associated with grammars. 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 CFGs.

Definition 2. *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* $A \to A_1, \ldots, A_k \in P$ *and* $w = w_1 \cdots w_k$*; or*
- $i = j$ *and* A *is an* ϵ*-rule in* P *and* $w = \epsilon$*; or*
- $i + 1 = j$ *and* $|w| = 1$ *and* $A \in \Sigma(w)$.

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

Definition 3. *The* ***fixpoint denotation*** *of a grammar* G *is* $[\![G]\!]_{fp} = lfp(T_G)$.

Theorem 2. *The operational and the denotational semantics coincide:* $x \in lfp(T_G)$ *iff* $G \vdash x$.

Proof. See figure 2 □

2.4 Compositionality

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 most natural observable for a grammar would be its language, but an alternative choice would be to take all the derivable strings, paired with the category that derives them. Thus we might take $Ob_1(G)$ to be $L(G)$, and $Ob_2(G)$ to be $\{\langle w, A\rangle \mid w \in L_A(G)\}$.

The choice of semantics induces a natural equivalence relation 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 be *correct* (observables-preserving): if $G_1 \equiv G_2$, then $Ob(G_1) = Ob(G_2)$. The way we defined it, the operational semantics of a grammar, '$[\![\cdot]\!]_{op}$', is correct with respect to both Ob_1 and Ob_2 (by theorem 1). By theorem 2, this holds for the denotational semantics '$[\![\cdot]\!]_{fp}$', too. Thus, we take $Ob(G)$ to be $\{\langle w, A\rangle \mid w \in L_A(G)\}$.

Let $\cup$ be a (commutative) 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 $[\![G_1 \cup G_3]\!] = [\![G_2 \cup G_4]\!]$. It is *commuting* (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 commuting with respect to some operator then it is compositional.

Definition 4. *The* ***union*** *of two CFGs,* $G_1 = \langle V_1, \Sigma_1, P_1, A_1^s\rangle$ *and* $G_2 = \langle V_2, \Sigma_2, P_2, A_2^s\rangle$*, denoted* $G_1 \cup G_2$*, is a new grammar* $G = \langle V_1 \cup V_2, \Sigma_1 + \Sigma_2, P_1 \cup P_2, A_1^s \cup A_2^s\rangle$.

Figure 3 exemplifies grammar union. Observe that it is commutative: $G_1 \cup G_2 = G_2 \cup G_1$.

Theorem 3. *'$[\![\cdot]\!]_{op}$' is not compositional.*

Proof. Consider again the grammars of figure 3. Let G_1' be the same as G_1, with the only difference being that $P_1' = P_1 \setminus \{VP \to V\ NP\}$. Now $[\![G_1]\!]_{op} = [\![G_1']\!]_{op} = \{[\text{loves}, i, V, i+1]\}_{i\geq 0} \cup \{[\text{sleeps}, i, V, i+1]\}_{i\geq 0}$. But when combined with G_2, both grammars yield different denotations. Since G_1' lacks the rule $VP \to V\ NP$, no sentences with transitive verbs are generated. For example, for $x = [\text{John loves Mary}, 0, S, 3]$, $x \in [\![G_1 \cup G_2]\!]_{op}$ but $x \notin [\![G_1' \cup G_2]\!]_{op}$. Thus, $[\![G_1]\!]_{op} = [\![G_1']\!]_{op}$ but $[\![G_1 \cup G_2]\!]_{op} \neq [\![G_1' \cup G_2]\!]_{op}$ and hence '$[\![\cdot]\!]_{op}$' is not compositional. □

The rule $VP \to V\ NP$ contributes nothing to the denotation of G_1, thereby allowing $[\![G_1]\!]_{op}$ to be equal to $[\![G_1']\!]_{op}$. However, when G_1 is combined with G_2, this rule can be used in a derivation, yielding $[\![G_1 \cup G_2]\!]_{op} \neq [\![G_1' \cup G_2]\!]_{op}$. The denotational semantics '$[\![\cdot]\!]_{fp}$' will not be any different, of course, but it provides a better starting point for experimenting with alternative definitions.

3 A compositional semantics

To overcome the problems delineated above, we follow Mancarella and Pedreschi (1988) in moving one step further, considering the grammar transformation operator itself (rather than its fixpoint) as the denotation of a grammar.

Definition 5. *The* ***algebraic denotation*** *of a grammar* G *is* $[\![G]\!]_{al} = T_G$. $G_1 \equiv_{al} G_2$ *iff* $T_{G_1} = T_{G_2}$.

Not only is this semantics compositional, it is also commuting with respect to grammar union,

- If $G\vdash^n x$ then $x \in T_G \uparrow n$. By induction on n: if $n = 1$ then x is an axiom and therefore either $x = [a, i, A, i+1]$ and $A \in \Sigma(a)$ or $x = [\epsilon, i, A, i]$ and A is an ϵ-rule. By the definition of T_G, $T_G(\emptyset) = \{[\epsilon, i, A, j] \mid A \text{ is an } \epsilon\text{-rule}\} \cup \{[w, i, A, i+1] \mid |w| = 1 \text{ and } A \in \Sigma(w)\}$, so $x \in T_G \uparrow 1$.
  Assume that the hypothesis holds for $n-1$; assume that $G\vdash^n [w, i, A, j]$ for $n > 1$. Then the inference rule must be applied at least once, i.e., there exist items $[w_1, i_1, A_1, j_1], \ldots, [w_k, i_k, A_k, j_k]$ such that $j_l = i_{l+1}$ for $1 \le l < k$ and $i = i_1$ and $j = j_k$ and $A \to A_1, \ldots, A_n \in P$. Furthermore, for every $1 \le l \le k$, the item $[w_l, i_l, A_l, j_l]$ can be deduced in $n-1$ steps: $G\vdash^{n-1}[w_l, i_l, A_l, j_l]$. By the induction hypothesis, for every $1 \le l \le k$, $[w_l, i_l, A_l, j_l] \in T_G \uparrow (n-1)$. By the definition of T_G, applying the first clause of the definition, $[w, i, A, j] \in T_G(T_G \uparrow (n-1)) = T_G \uparrow n$.
- If $x \in T_G \uparrow n$ then $G\vdash^n w$. By induction on n: if $n = 1$, that is, $[w, i, A, j] \in T_G \uparrow 1$, then either $i = j$, $w = \epsilon$ and A is an ϵ-rule in G, or $i + 1 = j$ and $A \in \Sigma(w)$. In the first case, $G\vdash^1 [w, i, A, j]$ by the first axiom of the deductive procedure; in the other case, $G\vdash^1 [w, i, A, j]$ by the second axiom.
  Assume that the hypothesis holds for $n-1$ and that $[w, i, A, j] \in T_G \uparrow n = T_G(T_G \uparrow (n-1))$. Then there exist items $y_1, \ldots, y_k \in T_G \uparrow (n-1)$ such that $y_l = [w_l, i_l, A_l, j_l]$ for $1 \le l \le k$ and $i_{l+1} = j_l$ for $1 \le l < k$ and $i_1 = 1$ and $j_k = j$ and $A \to A_1, \ldots, A_n \in P$ and $w = w_1 \cdots w_k$. By the induction hypothesis, for every $1 \le l \le k$, $G\vdash^{n-1}[w_l, i_l, A_l, j_l]$, and the inference rule is applicable, so by an additional step of deduction we obtain $G\vdash^n [w, i, A, j]$.

Figure 2: Proof of theorem 2

	G_1:	G_2:	$G_1 \cup G_2$:
A^s:	$\{S\}$	$\{\}$	$\{S\}$
P:	$S \to NP\ VP$		$S \to NP\ VP$
	$VP \to V$		$VP \to V$
	$VP \to V\ NP$		$VP \to V\ NP$
		$NP \to D\ N$	$NP \to D\ N$
		$NP \to PN$	$NP \to PN$
Σ:	sleeps $\mapsto \{V\}$		sleeps $\mapsto \{V\}$
	loves $\mapsto \{V\}$		loves $\mapsto \{V\}$
		Mary $\mapsto \{PN\}$	Mary $\mapsto \{PN\}$
		John $\mapsto \{PN\}$	John $\mapsto \{PN\}$
		the $\mapsto \{D\}$	the $\mapsto \{D\}$
		man $\mapsto \{N\}$	man $\mapsto \{N\}$

Figure 3: Grammar union

when a composition operation on denotations is defined as:

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

Theorem 4. *The semantics '$\equiv_{al}$' is commuting 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. See figure 4 □

Intuitively, since T_G captures only one step of the computation, it cannot capture interactions

We show 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 $\rho \in P_1 \cup P_2$ that licenses x. Then either $\rho \in P_1$ or $\rho \in P_2$, but in any case ρ would have licensed the same item, 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 $x = [w, i, A, i{+}1]$ and $|w| = 1$ and $A \in (\Sigma_1 + \Sigma_2)(w)$. Then either $A \in \Sigma_1(w)$ or $A \in \Sigma_2(w)$, and therefore $x \in T_{G_1}(I) \cup T_{G_2}(I)$.

Figure 4: Proof of theorem 4

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 6. *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. See figure 5 □

A simple solution to the problem 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. Evidently, for G_1 and G_2 of the above proof, $[\![G_1]\!]_{id} = [\![G_2]\!]_{id}$, so they no longer constitute a counter example. Also, it is easy to see that the proof of theorem 4 requires only a slight modification for it to hold in this case, so $T_G + Id$ is commuting (and hence compositional).

Unfortunately, this does not solve the problem. Let G_1 be the grammar

$$A_1^s = \emptyset, \Sigma_1 = \emptyset,$$
$$P_1 = \{A_1 \to A_2,\ A_2 \to A_3,\ A_3 \to A_1\}$$

and G_2 be the grammar

$$A_1^s = \emptyset, \Sigma_1 = \emptyset,$$
$$P_2 = \{A_1 \to A_3,\ A_2 \to A_1,\ A_3 \to A_2\}$$

To see that $[\![G_1]\!]_{id} \neq [\![G_2]\!]_{id}$, note that $T_{G_1}(\{[w, i, A_1, j]\}) = \{[w, i, A_1, j], [w, i, A_3, j]\}$; $T_{G_2}(\{[w, i, A_1, j]\}) = \{[w, i, A_1, j], [w, i, A_2, j]\}$. To see that $Ob(G \cup G_1) = Ob(G \cup G_2)$ for every G, observe that every rule application in either G_1 or G_2 can be modeled by two rule applications in the other grammar, and hence every category derivable in $G \cup G_1$ is derivable in $G \cup G_2$ and vice versa.

The choice of $lfp(T_G)$ as the denotation of a grammar is "too crude" (it identifies grammars that behave differently when combined with other grammars); $T_G + Id$ is "too fine" (it distinguishes between grammars that can be interchanged in any context); the "right" semantics, therefore, lies somewhere in between. Lassez and Maher (1984) suggest the following semantics for logic programs: rather than consider the operator associated with the entire program, they only look at the rules (excluding the facts), and take the meaning of a program to be the function that is obtained by infinite applications of the operator associated with the rules. We adapt below the results of Lassez and

Let G_1 be the grammar $A_1^s = \{S\}, \Sigma_1 = \emptyset, P_1 = \{S \to NP\ VP,\ NP \to NP\}$ and G_2 be the grammar $A_2^s = \{S\}, \Sigma_2 = \emptyset, P_2 = \{S \to NP\ VP\}$.

- $G_1 \not\equiv_{al} G_2$: because $T_{G_1}(\{[w, i, NP, j]\}) = \{[w, i, NP, j]\}$ but $T_{G_2}(\{[w, i, NP, j]\}) = \emptyset$
- 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 $NP \to NP$ in the former. This rule can contribute nothing to a derivation, since any item it licenses must already be derivable. Therefore, any item derivable with $G \cup G_1$ is also derivable with $G \cup G_2$ and hence $Ob(G \cup G_1) = Ob(G \cup G_2)$.

Figure 5: Proof of proposition 5

Maher (1984) and Maher (1988) to CFGs. First, this would require associating the following operator with a grammar:

Definition 7. *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 $A \to A_1, \ldots, A_k \in P$ 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 this is a function (from sets to sets), not a set.*

R_G is defined similarly to T_G (definition 2), ignoring the items added (by T_G) due to ϵ-rules and lexical items. Let

$$\begin{aligned} I_G \quad = \quad & \{[\epsilon, i, A, i] \mid A \text{ is an } \epsilon\text{-rule in } G\} \\ & \cup \{[a, i, A, i+1] \mid A \in \Sigma(a)\} \end{aligned}$$

Then for every grammar G and every set of items I, $T_G(I) = R_G(I) \cup I_G$. Furthermore, the functional semantics is naturally related to the fixpoint semantics: when the former is applied to I_G, it yields the latter.

Theorem 6. *For every grammar G, $(R_G + Id)^\omega(I_G) = lfp(T_G)$.*

Proof. See figure 6 □

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 Ob_2, as before, we could in general have $[\![G_1]\!]_{fn} = [\![G_2]\!]_{fn}$ but $Ob(G_1) \neq Ob(G_2)$, 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 that the semantics is correct, we define the observables of a grammar G *with respect to a given input*. Thus, let

$$Ob_I(G) = \{\langle w, A\rangle \mid [w, 0, A, |w|] \in [\![G]\!]_{fn}(I)\}$$

A semantics is correct iff $G_1 \equiv G_2$ implies that for all I, $Ob_I(G_1) = Ob_I(G_2)$. Clearly, for every given set of items I, if $[\![G_1]\!]_{fn} = [\![G_2]\!]_{fn}$ then $[\![G_1]\!]_{fn}(I) = [\![G_2]\!]_{fn}(I)$, and hence '$[\![\cdot]\!]_{fn}$' is correct. When the input is taken to be I_G, the expected observables are obtained:

Theorem 7. *For every grammar G, $\langle w, A\rangle \in Ob_{I_G}(G)$ iff $w \in L_A(G)$.*

Proof. See figure 7. □

To show that '$[\![\cdot]\!]_{fn}$' is compositional we must define an operator for combining denotations. Unfortunately, the simplest operator, '+', would not do. To see that, consider the following grammars:

$$\begin{aligned} G_1: \quad & A_1^s = \emptyset, \Sigma_1 = \emptyset, P_1 = \{S \to VP\} \\ G_2: \quad & A_2^s = \emptyset, \Sigma_2 = \emptyset, P_2 = \{VP \to V\} \end{aligned}$$

Observe that, for $x = [w, i, V, j]$,

$$\begin{aligned} & [\![G_1]\!]_{fn}(\{x\}) = \{x\} \\ & [\![G_2]\!]_{fn}(\{x\}) = \{x, [w, i, VP, j]\} \\ & [\![G_1 \cup G_2]\!]_{fn}(\{x\}) = \{x, [w, i, VP, j], [w, i, S, j]\} \end{aligned}$$

We show that for every n, $(T_G + Id) \uparrow n = (\Sigma_{k=0}^{n-1}(R_G + Id)^k)(I_G)$ 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 2, which are exactly I_G. Also, $(\Sigma_{k=0}^{0}(R_G + Id)^k)(I_G) = (R_G + Id)^0(I_G) = I_G$.
Assume that the proposition holds for $n-1$, that is, $(T_G + Id) \uparrow (n-1) = (\Sigma_{k=0}^{n-2}(R_G + Id)^k)(I_G)$. Then

$$\begin{array}{lll}
(T_G + Id) \uparrow n = & (T_G + Id)((T_G + Id) \uparrow (n-1)) & \text{definition of } \uparrow \\
= & (T_G + Id)((\Sigma_{k=0}^{n-2}(R_G + Id)^k)(Init_G)) & \text{by the induction hypothesis} \\
= & (R_G + Id)((\Sigma_{k=0}^{n-2}(R_G + Id)^k)(Init_G)) \cup Init_G & \text{as } T_G(I) = R_G(I) \cup Init_G \\
= & (R_G + Id)((\Sigma_{k=0}^{n-2}(R_G + Id)^k)(Init_G)) & \\
= & (\Sigma_{k=0}^{n-1}(R_G + Id)^k)(Init_G) &
\end{array}$$

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

Figure 6: Proof of theorem 6

Let $x = [w, 0, A, |w|]$. Then:

$$\begin{array}{lll}
\langle w, A\rangle \in Ob_{I_G}(G) & \text{iff} \quad x \in [\![G]\!]_{fn}(I_G) & \text{definition of } Ob_I \\
& \text{iff} \quad x \in lfp(T_G) & \text{by theorem 6} \\
& \text{iff} \quad G \vdash x & \text{by theorem 2} \\
& \text{iff} \quad w \in L_A(G) & \text{by theorem 1}
\end{array}$$

Figure 7: Proof of theorem 7

That is, $([\![G_1]\!]_{fn} + [\![G_2]\!]_{fn})(\{x\}) = [\![G_1]\!]_{fn}(\{x\}) \cup [\![G_2]\!]_{fn}(\{x\}) \neq [\![G_1 \cup G_2]\!]_{fn}(\{x\})$.

However, define $[\![G_1]\!]_{fn} \bullet [\![G_2]\!]_{fn}$ to be $([\![G_1]\!]_{fn} + [\![G_2]\!]_{fn})^\omega$. Then '$[\![\cdot]\!]_{fn}$' is commuting with respect to '$\bullet$' and '$\cup$'.

Lemma 8. *For every grammar G, $[\![G]\!]_{fn}$ is increasing: $[\![G]\!]_{fn}(I) \supseteq I$ for all I.*

Proof. Since $(R_G + Id)(I) \supseteq I$, also $(R_G + Id)^\omega(I) \supseteq I$. □

Definition 8. *If f, g are two functions over the same domain and range, let $f \leq g$ iff for all I, $f(I) \subseteq g(I)$. Let $f \circ g$ denote function composition.*

Lemma 9. *For every two grammars G_1, G_2, $([\![G_1]\!]_{fn} + [\![G_2]\!]_{fn}) \leq ([\![G_1]\!]_{fn} \circ [\![G_2]\!]_{fn})$.*

Proof. See figure 8 □

Lemma 10. *For every grammar G, $[\![G]\!]_{fn}$ is idempotent: $[\![G]\!]_{fn} \circ [\![G]\!]_{fn} = [\![G]\!]_{fn}$.*

Proof. See figure 9 □

Theorem 11. $[\![G_1 \cup G_2]\!]_{fn} = [\![G_1]\!]_{fn} \bullet [\![G_2]\!]_{fn}$.

Proof. See figure 10. □

Since '$[\![\cdot]\!]_{fn}$' is commuting, it is also compositional. Furthermore, once the lexicon (including ϵ-rules) is viewed as input to the grammar, the semantics is also fully-abstract:

Theorem 12. *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 figure 11 □

4 Grammar modules

We now have a semantics that naturally supports grammar composition. We want to provide an explicit definition of grammar modules with a compositional, fully-abstract semantics. Our motivation in the definition comes

By definition, $([\![G_1]\!]_{fn} \circ [\![G_2]\!]_{fn})(I) = [\![G_1]\!]_{fn}([\![G_2]\!]_{fn}(I))$. From lemma 8, $[\![G_1]\!]_{fn}(I) \supseteq I$, hence $[\![G_1]\!]_{fn}([\![G_2]\!]_{fn}(I)) \supseteq [\![G_2]\!]_{fn}(I)$. From lemma 8 and monotonicity, also $[\![G_1]\!]_{fn}([\![G_2]\!]_{fn}(I)) \supseteq [\![G_1]\!]_{fn}(I)$. Hence $([\![G_1]\!]_{fn} \circ [\![G_2]\!]_{fn})(I) \supseteq ([\![G_1]\!]_{fn} \cup [\![G_2]\!]_{fn})(I)$ and $([\![G_1]\!]_{fn} + [\![G_2]\!]_{fn}) \leq ([\![G_1]\!]_{fn} \circ [\![G_2]\!]_{fn})$.

Figure 8: Proof of lemma 9

(Lassez and Maher, 1984) For every I, $([\![G]\!]_{fn} \circ [\![G]\!]_{fn})(I) = ((R_G + Id)^\omega \circ (R_G + Id)^\omega)(I) = (R_G + Id)^\omega((R_G + Id)^\omega(I)) = \Sigma_{i=0}^\infty (R_G + Id)^i(\Sigma_{j=0}^\infty (R_G + Id)^j(I)) = \Sigma_{i=0}^\infty \Sigma_{j=0}^\infty (R_G + Id)^{i+j}(I) = \Sigma_{m=0}^\infty (R_G + Id)^m(I) = (R_G + Id)^\omega(I) = [\![G]\!]_{fn}$.

Figure 9: Proof of lemma 10

(Lassez and Maher, 1984)

$$
\begin{array}{rcll}
[\![G_1 \cup G_2]\!]_{fn} & = & (R_{G_1} + Id + R_{G_2} + Id)^\omega & \\
& \leq & ((R_{G_1} + Id)^\omega + (R_{G_2} + Id)^\omega)^\omega & \text{as } R_G + Id \leq (R_G + Id)^\omega \text{ for every } G \\
& \leq & ((R_{G_1} + Id)^\omega \circ (R_{G_2} + Id)^\omega)^\omega & \text{by lemma 9} \\
& \leq & ((R_{G_1 \cup G_2} + Id)^\omega \circ (R_{G_1 \cup G_2} + Id)^\omega)^\omega & \text{as } G_1 \cup G_2 \supseteq G_1 \text{ and } G_1 \cup G_2 \supseteq G_2 \\
& = & ([\![G_1 \cup G_2]\!]_{fn} \circ [\![G_1 \cup G_2]\!]_{fn})^\omega & \text{definition of '}[\![\cdot]\!]_{fn}\text{'} \\
& = & ([\![G_1 \cup G_2]\!]_{fn})^\omega & \text{by lemma 10} \\
& = & [\![G_1 \cup G_2]\!]_{fn} & \text{since } (f^\omega)^\omega = f^\omega
\end{array}
$$

Thus all the inequations are equations, and in particular

$$[\![G_1 \cup G_2]\!]_{fn} = ((R_{G_1} + Id)^\omega + (R_{G_2} + Id)^\omega)^\omega = ([\![G_1]\!]_{fn}^\omega + [\![G_2]\!]_{fn}^\omega)^\omega = [\![G_1]\!]_{fn} \bullet [\![G_2]\!]_{fn}$$

Figure 10: Proof of theorem 11

Assume towards a contradiction that $[\![G_1]\!]_{fn} \neq [\![G_2]\!]_{fn}$ and still for all I and all G, $Ob_I(G_1 \cup G) = Ob_I(G_2 \cup G)$. Without loss of generality, assume that for $x = [w, i, A, j]$, there exists I such that $x \in [\![G_1]\!]_{fn}(I)$ but $x \notin [\![G_2]\!]_{fn}(I)$. Let $G = \langle V, \Sigma, P, A^s \rangle$ be a CFG, where $V = \{S, B_1, B_2\}$ is a set of categories such that $V \cap (V_1 \cup V_2) = \emptyset$, $\Sigma(a) = \emptyset$ for every a, $A^s = \{S\}$ and $\rho = S \rightarrow B_1\ A\ B_2$ is the only rule in P. Let $y = [w, 0, S, |w|]$, $y_1 = [\epsilon, 0, B_1, i]$, $y_2 = [\epsilon, j, B_2, |w|]$ and $J = I \cup \{y_1, y_2\}$.
$\langle w, S \rangle \in Ob_J(G_1 \cup G)$: because ρ can be applied to y_1, x, y_2, yielding exactly y.
$\langle w, S \rangle \notin Ob_J(G_2 \cup G)$: because the only occurrence of S is in ρ, there must be an application of ρ in $G_2 \cup G$. For ρ to apply, x must be derivable in $G_2 \cup G$. Since neither y_1 nor y_2 can contribute to such a derivation (since B_1 and B_2 occur nowhere else in $G_2 \cup G$), x must be derivable in G_2, in contradiction to the assumption.
Hence $Ob_J(G_1 \cup G) \neq Ob_J(G_2 \cup G)$, in contradiction to the assumption. Hence '$[\![\cdot]\!]_{fn}$' is fully abstract.

Figure 11: Proof of theorem 12

from practical problems of grammar engineering: when designing broad-coverage grammars for natural languages, grammar engineers are faced with problems similar to those tackled by software engineers. A good definition for grammar modules must provide for information encapsulation, separate and incremental compilation, module linkage, etc. Viable for these applications is a definition of an *interface* through which a grammar module can interact with its environment. Following Gaifman and Shapiro (1989), we define two channels for such interaction: a set of *import* and a set of *export* categories.

Definition 9. *A **grammar module** is a quadruple $M = \langle G, Im, Ex, Int\rangle$, where $G = \langle V, \Sigma, P, A^s\rangle$ is a CFG and Im, Ex and Int partition V into three disjoint classes. $M_1 = \langle G_1, Im_1, Ex_1, Int_1\rangle$ and $M_2 = \langle G_2, Im_2, Ex_2, Int_2\rangle$ are **composable** iff $Ex_1 \cap Ex_2 = \emptyset$ and $Int_1 \cap V_2 = Int_2 \cap V_1 = \emptyset$. In this case, their **composition**, denoted $M_1 \cup M_2$, is $M = \langle G, Im, Ex, Int\rangle$, where $G = G_1 \cup G_2$, $Im = (Im_1 \cup Im_2) \setminus (Ex_1 \cup Ex_2)$, $Ex = Ex_1 \cup Ex_2$ and $Int = Int_1 \cup Int_2$.*

Figure 12 exemplifies module composition.

The denotation of a module is based on the denotation of its grammar, taking into account its interface:

Definition 10. *The denotation of a grammar module $M = \langle G, Im, Ex, Int\rangle$ is*

$$[\![M]\!]m = \langle [\![G]\!]_{fn}, Im, Ex\rangle$$

The notion of observables is adapted to modules in the same vein:

Definition 11. *The observables of a module $M = \langle G, Im, Ex, Int\rangle$ are*

$$Ob_I(M)\{\langle w, A\rangle \mid [w, 0, A, |w|] \in [\![G]\!]_{fn}(I) \text{ and } A \in Ex\}$$

Trivially, the module semantics '$[\![\cdot]\!]m$' is correct with respect to the above definition: if two modules are semantically equivalent, they must have the same exported categories; hence restricting the observables to those categories that are exported by a module does not affect correctness.

Finally, a combination operation on module denotations reflects the interface requirements:

Definition 12. *If $M_1 = \langle G_1, Im_1, Ex_1, Int_1\rangle$ and $M_2 = \langle G_2, Im_2, Ex_2, Int_2\rangle$ are grammar modules, then*

$$[\![M_1]\!]m \bullet [\![M_2]\!]m = \langle [\![G_1]\!]_{fn} \bullet [\![G_2]\!]_{fn}, (Im_1 \cup Im_2) \setminus (Ex_1 \cup Ex_2), Ex_1 \cup Ex_2\rangle$$

Since module composition is defined in terms of grammar union, the module semantics we suggest is trivially compositional and fully-abstract:

Theorem 13. *If $M_1 = \langle G_1, Im_1, Ex_1, Int_1\rangle$ and $M_2 = \langle G_2, Im_2, Ex_2, Int_2\rangle$ are composable then $[\![M_1 \cup M_2]\!]m = [\![M_1]\!]m \bullet [\![M_2]\!]m$.*

Proof. See figure 13. □

Theorem 14. *Let $M_1 = \langle G_1, Im_1, Ex_1, Int_1\rangle$ and $M_2 = \langle G_2, Im_2, Ex_2, Int_2\rangle$ be two modules. If every module that is composable with M_1 is also composable with M_2, and for every such module $M = \langle G, Im, Ex, Int\rangle$ and set of items I, $Ob_I(M \cup M_1) = Ob_I(M \cup M_2)$ then $[\![M_1]\!]m = [\![M_2]\!]m$.*

Proof. Since every module that is composable with M_1 is also composable with M_2, we have $Ex_1 = Ex_2$, $Im_1 = Im_2$ and $V_1 = V_2$. The proof of theorem 12 can be extended to modules as follows: assume that $[\![M_1]\!]m \neq [\![M_2]\!]m$ but for every module M and set of items I, $Ob_I(M \cup M_1) = Ob_I(M \cup M_2)$. Let G, x, y, y_1, y_2, I, J be as in the proof of theorem 12. Let M be the module $\langle G, \emptyset, \{S\}, \{B_1, B_2\}\rangle$. Then $Ob_J(M_1 \cup M) \neq Ob_J(M_2 \cup M)$, contradicting the assumption. Hence '$[\![\cdot]\!]m$' is fully-abstract. □

Since module composition is defined in terms of grammar union and set union, some desirable properties of this operation can be easily observed. In the following two propositions, '=' means that both sides are defined and equal, or both are undefined.

Proposition 15. *Module composition is commutative: $M_1 \cup M_2 = M_2 \cup M_1$.*

Proposition 16. *Module composition is associative: $(M_1 \cup M_2) \cup M_3 = M_1 \cup (M_2 \cup M_3)$.*

	M_1 :	M_2 :	$M_1 \cup M_2$:
A^s :	$\{S\}$	$\{\}$	$\{S\}$
P :	$S \to NP\ VP$		$S \to NP\ VP$
	$VP \to V$		$VP \to V$
	$VP \to V\ NP$		$VP \to V\ NP$
		$NP \to D\ N$	$NP \to D\ N$
		$NP \to PN$	$NP \to PN$
Σ :	sleeps $\mapsto \{V\}$		sleeps $\mapsto \{V\}$
	loves $\mapsto \{V\}$		loves $\mapsto \{V\}$
		Mary $\mapsto \{PN\}$	Mary $\mapsto \{PN\}$
		John $\mapsto \{PN\}$	John $\mapsto \{PN\}$
		the $\mapsto \{D\}$	the $\mapsto \{D\}$
		man $\mapsto \{N\}$	man $\mapsto \{N\}$
Ex :	$\{S\}$	$\{NP\}$	$\{S,\ NP\}$
Im :	$\{NP\}$	$\emptyset$	$\emptyset$
Int :	$\{V,\ VP\}$	$\{D,\ N,\ PN\}$	$\{V,\ VP,\ D,\ N,\ PN\}$

Figure 12: Module composition

$$
\begin{array}{lcll}
[\![M_1 \cup M_2]\!]m & = & \langle [\![G_1 \cup G_2]\!]_{fn}, (Im_1 \cup Im_2) \setminus (Ex_1 \cup Ex_2), Ex_1 \cup Ex_2 \rangle & \text{definitions 9, 10} \\
& = & \langle [\![G_1]\!]_{fn} \bullet [\![G_2]\!]_{fn}, (Im_1 \cup Im_2) \setminus (Ex_1 \cup Ex_2), Ex_1 \cup Ex_2 \rangle & \text{by theorem 11} \\
& = & [\![M_1]\!]m \bullet [\![M_2]\!]m & \text{by definition 12}
\end{array}
$$

Figure 13: Proof of theorem 13

We conclude with a comprehensive example of three composable modules: M_1, defining *sentences*, M_2, which deals with *noun phrases*, and M_3, whose purpose is to generate *relative clauses* (figure 14). A closer look at the *internal* categories of each module reveals the kind of information encapsulation that is obtained by this approach: for example, the categories *D, N* and *PN* are internal to the *NP* module (M_2). They cannot be accessed, or used, by any of the other two modules. This is as if the internal structure of noun phrases is not available to other modules – although M_1 does make use of the category *NP*, which it imports. The denotation of M_1 is thus dependent on noun phrases, imported from M_2. In the same way, M_2 depends on *RELCL*, imported from M_3, and M_3 imports the category S from M_1.

Observe that M_1 and M_2 are composable, since $Ex_1 \cap Ex_2 = \{S\} \cap \{NP\} = \emptyset$, $Int_1 \cap V_2 = \{V,\ VP\} \cap \{NP,\ D,\ N,\ PN,\ RELCL\} = \emptyset$ and $Int_2 \cap V_1 = \{D,\ N,\ PN\} \cap \{S,\ V,\ VP,\ NP\} = \emptyset$. Their composition yields $M_1 \cup M_2$, depicted in figure 15. This, in turn, is composable with M_3; the result of this composition yields $(M_1 \cup M_2) \cup M_3$ (figure 15). Notice that the same result would have been obtained by computing $M_1 \cup (M_2 \cup M_3)$ (or, for that matter, $(M_1 \cup M_3) \cup M_2$). Notice that $(M_1 \cup M_2) \cup M_3$ has an empty set of imported categories. It is therefore complete, and does not depend on any additional, external information.

	M_1 :	M_2 :	M_3 :
A^s :	$\{S\}$	$\emptyset$	$\emptyset$
P :	$S \to NP\ VP$ $VP \to V$ $VP \to V\ NP$	$NP \to D\ N$ $NP \to PN$ $NP \to D\ N\ RELCL$	$RELCL \to REL\ S$
Σ :	sleeps, loves $\mapsto \{V\}$	John, Mary $\mapsto \{PN\}$ the $\mapsto \{D\}$ man $\mapsto \{N\}$	that, who, $\epsilon \mapsto \{REL\}$
Ex :	$\{S\}$	$\{NP\}$	$\{RELCL\}$
Im :	$\{NP\}$	$\{RELCL\}$	$\{S\}$
Int :	$\{V,\ VP\}$	$\{D,\ N,\ PN\}$	$\{REL\}$

Figure 14: Three grammar modules

	$M_1 \cup M_2$:	$(M_1 \cup M_2) \cup M_3$:
A^s :	$\{S\}$	$\{S\}$
P :	$S \to NP\ VP$ $VP \to V$ $VP \to V\ NP$ $NP \to D\ N$ $NP \to PN$ $NP \to D\ N\ RELCL$	$S \to NP\ VP$ $VP \to V$ $VP \to V\ NP$ $NP \to D\ N$ $NP \to PN$ $NP \to D\ N\ RELCL$ $RELCL \to REL\ S$
Σ :	sleeps, loves $\mapsto \{V\}$ John, Mary $\mapsto \{PN\}$ the $\mapsto \{D\}$ man $\mapsto \{N\}$	sleeps, loves $\mapsto \{V\}$ John, Mary $\mapsto \{PN\}$ the $\mapsto \{D\}$ man $\mapsto \{N\}$ that, who, $\epsilon \mapsto \{REL\}$
Ex :	$\{S,\ NP\}$	$\{RELCL,\ NP,\ S\}$
Im :	$\{RELCL\}$	$\{\}$
Int :	$\{V,\ VP,\ D,\ N,\ PN\}$	$\{REL,\ V,\ VP,\ D,\ N,\ PN\}$

Figure 15: Composed modules

5 Conclusions

This paper defines modules in context-free grammars in a way that is compositional and fully-abstract. In contrast to the standard definitions for the semantics of CFGs, our definition is such that two modules are semantically equivalent if and only if they can be interchanged in every context. This gives a clear, mathematically sound way for composing parts of grammars.

CFGs are usually not considered a suitable model for natural languages; rather, most linguistic theories use more powerful formalisms such as (some variant of) unification grammars. We believe that the results reported on in this paper can be extended to the more expressive domain. We have shown elsewhere (Wintner, 1999) that a functional semantics, similar to the one defined here, is compositional and fully-abstract for unification grammars, and we are currently developing a model for modularity in this framework.

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.

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.

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. 1999. Compositional semantics for linguistic formalisms. In *Proceedings of ACL'99, the 37th Annual Meeting of the Association for Computational Linguistics*, June.