Modular Context-Free Grammars

Shuly Wintner (`shuly@cs.haifa.ac.il`)
Department of Computer Science, University of Haifa

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. This gives a clear, mathematically sound way for composing parts of grammars.

Keywords: context-free grammars, modularity, programming language semantics

1. Introduction

The tasks of developing large scale grammars for natural languages become more and more complicated: it is not unusual for a single grammar to be developed by a team including a number of linguists, computational linguists and computer scientists. Computational grammars with broad coverage are complex entities, sometimes made up by tens of thousands of lines of code (Oepen et al., 2000; Wahlster, 1997). In such a setup, the problems that grammar engineers face when they design a broad-coverage grammar for some natural language are very reminiscent of the problems tackled by software engineering (Erbach and Uszkoreit, 1990). Considerable effort was invested in making parsers and other grammar manipulating systems more efficient (Wintner and Francez, 1999; Carpenter and Penn, 1999; Callmeier, 2000; Malouf et al., 2000; Sarkar et al., 2000). However, while software developers can benefit from two decades of research of software engineering, grammar engineering is in its infancy. In most large-scale systems (e.g., XTAG (The XTAG Research Group, 1998), LKB (Copestake, 1999), ALE (Carpenter and Penn, 1999), etc.) the grammar consists of a single conceptual entity (even when it is distributed over more than one file), and very few provisions for modular development of grammars exist.

Our goal in this paper is to provide a good definition for modules in context-free grammars (CFGs) and define module composition in a way that supports a transparent implementation. We focus on context-free grammars, probably the simplest linguistic formalism that was ever suggested as appropriate for describing the structure of natural languages. It is widely believed today that CFGs are not expressive enough for this task. Still, the present work can serve as the departure point for exploration with more powerful formalisms, notably unification-based grammars. We believe that the results reported here can be adequately extended to more expressive frameworks.

1.1. Motivation

The motivation for modular grammar development systems is straightforward. Constructing large-scale grammars becomes much simpler when the task can be cleanly distributed among different developers, provided that well-defined interfaces govern the interaction among modules. In an Internet era, modularity facilitates the development of several domain-specific sub-grammars that can reside on different sites and be integrated with server-side modules on demand.

Back in 1990, Erbach and Uszkoreit (1990) observed that

> The development of large grammars is extremely slow. Existing large grammars have usually been developed by a single person... In computer programming, modularization has proved to be a useful concept for the distributed development of large programs. [However,] No methods exist for the modularization of grammars... A modularization concept would not only further efficient development, it would also boost reusability and grammar evaluation.

The needs for modularity are also acknowledged by, e.g., Woszczyna et al. (1998), who note that "Many of the advantages of modularity and shared libraries equally apply to the design of a ... grammar for a large domain," and by Lehmann et al. (1995), noting that "grammars need to be *modular*, and we need a useful working definition of modularity". The exact form of modularity can be debated: in a recent work, Copestake and Flickinger (2000) claim that "Any grammar development environment should provide explicit support for collaborative development", but state further:

> It is often suggested that the problem with grammar engineering is that there is a lack of modularity, but it is not clear to us that this is correct.

Later, they agree that "there are other notions of modularity", but conclude that "despite the many aids to grammar engineering that have been developed, we think that to some extent it just has to be accepted that it really is inherently difficult." We hope to show here a first step towards a systematic solution of an inherently difficult problem.

1.2. Related work

While the needs are clear, solutions are few and far between; very few attempts were made to address modularity directly. In what follows we survey some of them.

While not directly useful for natural languages, *attribute grammars* are a formalism that extends CFGs by augmenting non-terminal symbols with a flat form of feature structures. Attribute grammars are used primarily for specifying the syntax and semantics of programming languages; motivated by the need for more abstraction in the specification, Dueck and Cormack (1990) describe a mechanism for generating attribute grammars from rules which are grouped together as modular attribute grammars. The formalism can be viewed as a way to group the scalar rules together; but it does not provide for interactions among different rules, and it does not define the notion of a grammar module.

Kasper and Krieger (1996) describe a technique for dividing a unification-based grammar into two components, roughly along the syntax/semantics axis. Their motivation is efficiency: observing that syntax usually imposes more constraints on permissible structures, and semantics usually mostly adds structure, they propose to parse with the syntactic constraints first, and apply the semantics later. However, this proposal requires that a single grammar be given, from which the two components can be derived. Also, they observe that the intersection of the languages generated by the two grammars does not yield the language of the original grammar. Another attempt to introduce modularity, as well as other grammar engineering techniques, to a grammatical formalism, was done in the context of ALEP (Theofilidis et al., 1997; Bredenkamp et al., 1997). Here, too, the major consideration is efficiency; there is no discussion detailing how a grammar can be developed in modules, or how grammar fragments are integrated. Still in the context of unification-based grammars, Lehmann et al. (1995) define modules as sets of types in a system that is based on typed feature structures. This proposal is not fully worked out, but we disagree with one of its major assumptions, namely that "in order to reuse a module from a grammar, its dependencies to other modules must be reduced to a minimum." On the contrary, we seek a definition for modules that will explicate the interface among modules and their interactions. In a different work, Keselj (2001) builds on ideas introduced by Wintner (1999c) to provide a definition of modular HPSG. The definition is rather ad-hoc, and no evaluation of its suitability or usefulness is given.

Zajac and Amtrup (2000) present an implementation of a pipeline-like composition operator that enables the grammar designer to break a grammar into sub-grammars that are applied in a sequential manner at run-time. Such an organization is especially useful for dividing the development process into stages that correspond to morphological processing, syntax, semantics, and so on. The notion of composition here is such that sub-grammar G_{i+1} operates on the output of sub-grammar G_i; such an organization might not be suitable for many grammar development frameworks. A similar idea is proposed by Basili et al. (2000): it is an approach to parsing that divides the task into subtasks, whereby a module component P_i takes an input sentence at a given state of analysis S_i and augments this information in S_{i+1} using a knowledge base K_i. Here, too, it is the processing system, not the grammar, which is modularized in a pipeline fashion.

A different approach to designing modular grammars is employed by Woszczyna et al. (1998). Here, grammars encode semantic information rather than syntactic structure, and thus interactions among different modules are kept to a minimum. The sub-domain grammars draw from a shared library of rules in order to maintain consistency in the treatment of common objects such as dates or time expressions; but the parser tags sub-trees according to their sub-domain, which implies that the interaction among modules is minimal.

1.3. Methodology

What is missing from most of the works described above is a systematic methodology for developing a concept of modules. As Erbach and Uszkoreit (1990) point out,

> A prerequisite for achieving the reusability of grammatical resources are mathematical concepts and a representation language for the abstract specification of grammatical knowledge. An abstract declarative specification language with a clean semantics is needed.

In this work we define such a semantics and use it to drive a “good” definition of modules in CFGs. To motivate our approach, we note that a naïve definition of modules would not do. Suppose one defines a module simply as a context-free grammar, with a simple operation that composes modules by unioning their components (non-terminal symbols, terminal symbols and rules). Since rules of two different modules can interact in the composed grammar, the language generated by the composed grammar would not be the union of the languages generated by the two modules. A (formal) solution would have been to rename the non-terminals apart, but this will eliminate any chance of interaction between the two modules, and simply serve to define a grammar for unioning the respective languages. What we seek here is a definition that will facilitate such interaction.

It is customary in computational linguistics to view grammars as computer programs and grammar formalisms as high-level, declarative programming languages (Pereira and Warren, 1983; Pereira and Shieber, 1984; Shieber et al., 1995). Extending this view, it is possible to adapt theoretical results obtained in the area of programming language semantics to the investigation of grammar formalisms. Our departure point in this work 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).

Why is formal semantics important? To use the terminology of Donahue (1976), "the primary goal of formal semantics is to provide more effective communication between the language designer and the various audiences with an interest in the language." This kind of communication can take different forms (Ghezzi and Jazayeri, 1987, Chapter 9): first and foremost, grammar semantics provides a rigorous and unambiguous definition of the grammatical formalism. It provides a basis for comparing different formalisms and within a given formalism, comparing different grammars. It is independent of a particular implementation, and thus constructs that are based on clear semantics are potentially more likely to be reusable across platforms. It forms the basis for proving the correctness of an implementation: it can be mathematically shown that a certain grammar implementation (such as a parser) is faithful to its specification. And it forms the basis for grammar correctness proofs: with a well-defined semantics a grammar designer can prove mathematically that a certain grammar, or a grammar fragment, has a particular effect. As we shall see below, understanding the semantics of grammar formalisms better can lead to natural definitions for grammar composition operators: the semantics can drive the definition of such operators. Another benefit of research on grammar semantics is that it can be used to implement grammar manipulation programs such as parsers or generators, in a similar way to the use of programming language semantics for the construction of compilers and interpreters.

There are several ways to define the semantics of a programming language (or a grammar formalism). Informally, a semantics assigns a *meaning*, or a *denotation*, to every grammar. It also provides a way to abstract away from the structure of the grammar itself: it induces a natural equivalence relation on grammars, whereby two grammars are equivalent if and only if they have the same denotations. As we are motivated by modularity considerations, we seek a semantics that will equate exactly these grammars which "behave" the same in every context. In other words, we search for a definition that will guarantee that two grammars are equivalent if and only if they can be freely substituted for each other, no matter what other grammars they are composed with.

To formalize this notion we introduce two concepts, well established in the theory of programming language semantics (Tennet, 1991, Chapter 1): compositionality (Scott and Strachey, 1971) and full-abstraction (Milner, 1975). Compositionality ensures that the denotation of a grammar is expressed as a function of the denotations of its components. An important benefit of compositionality is that replacing any component of a grammar by a semantically equivalent component yields a grammar that is semantically equivalent to the original. Another benefit is that it allows properties of grammars to be proven by induction on the structure of the composite grammar.

Compositionality can always be achieved in a trivial way by setting the denotation of a grammar to be the grammar itself. However, such a trivial semantics makes too many distinctions between grammars that might otherwise be considered equivalent. A "good" semantics should map grammars that are abstractly equivalent into the same equivalence class; a semantics that makes unnecessary distinctions is "too concrete". An ideal semantics is fully-abstract: it does not distinguish grammar fragments semantically unless there exist contexts for these fragments that allow the differences to be observed.

In section 2 we survey two existing 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 sections 3 and 4, eventually getting at one that is both compositional and fully-abstract. We then discuss grammar modularity in section 5, defining grammar modules and showing that the chosen semantics is indeed fully-abstract for modular CFGs as well.

2. Semantics of CFGs

When formal languages are concerned, one usually thinks of grammars as formal entities which denote languages. In other words, one usually considers two grammars to be equivalent if (and only if) they generate the same language. When grammars are used for natural languages, it is sometimes required that two equivalent grammars also assign isomorphic (or even identical) trees to the sentences they generate. A general way to look at these possibilities is to view grammars as syntactic notions which denote some other entities. The denotation of a grammar G can be its language, $L(G)$; or a set of pairs $\langle w, \tau \rangle$ such that $w \in L(G)$ and τ is some tree assigned to it by G. In other words, a semantics is a function that associates some mathematical entity (such as a set) with each grammar. In this section and the next two we experiment with various definitions for the semantics of CFGs.

Of course, the choice of semantics cannot be arbitrary: one would like a semantics to reflect in some way "what the grammar is doing". To formalize this intuitive notion we define an *observables* function, associating a set $Ob(G)$ with each grammar G. In the domain of programming languages, the observables function reflects the input–output relation. The most natural observables 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. This provides more detailed information about the grammar, including all the "phrases" it defines, paired with their respective syntactic category. In this work we take $Ob(G)$ to be $\{\langle w, A \rangle \mid w \in L_A(G)\}$ ($w \in L_A(G)$ if w can be derived from the category A in G, see below), although all the results still hold when $Ob(G)$ is taken to be $L(G)$. This notion of observables will be refined in section 4 to reflect the view of the lexicon as input to the grammar.

To relate the semantics to the observables function, thereby constraining its arbitrariness, one usually requires that the semantics be *correct*. A semantics is correct if whenever two grammars have the same denotation, they also have the same observables. Of course, if one defines the denotation of a grammar to be its observables, correctness follows trivially; but it is possible to define a correct semantics that is different from the observables function. We investigate such possibilities below.

2.1. Basic definitions

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

[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 $V = P = \bigcup_{a \in \text{WORDS}} \Sigma(a) = \emptyset$).

range over categories and σ, ρ — over *forms*, (possibly empty) sequences thereof. A rule $(A, (A_1, \ldots, A_n))$ is written $A \rightarrow A_1, \ldots, A_n$, and A is its *head*. If $n = 0$ the rule is an ϵ-*rule* (sometimes denoted simply A). $I\!N$ denotes the set $\{0, 1, 2, \ldots\}$. For two functions f, g, over the same (set) domain and range, $f + g$ is defined as $\lambda I. f(I) \cup g(I)$; $f \leq g$ iff for all I, $f(I) \subseteq g(I)$; and $f \circ g$ denotes function composition.

Figure 1 depicts an example grammar. V is the smallest set containing all the categories that occur in P and Σ.

$$
\begin{array}{ll}
A^s: & \{S\} \\
 & \\
P: & S \rightarrow NP\ VP \\
 & VP \rightarrow V \\
 & VP \rightarrow V\ NP \\
 & NP \rightarrow D\ N \\
 & NP \rightarrow PN
\end{array}
\qquad
\begin{array}{ll}
\Sigma: & \text{sleeps} \mapsto \{V\} \\
 & \text{loves} \mapsto \{V\} \\
 & \text{Mary} \mapsto \{PN\} \\
 & \text{John} \mapsto \{PN\} \\
 & \text{the} \mapsto \{D\} \\
 & \text{man} \mapsto \{N\}
\end{array}
$$

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 $A \rightarrow A_1 \cdots A_n \in P$. The reflexive transitive closure of '$\Rightarrow_G$' is '$\overset{*}{\Rightarrow}_G$'. The *language* of a category A in grammar G, $L_A(G)$, is $\{a_1 \cdots a_n \mid A \overset{*}{\Rightarrow}_G A_1, \ldots, A_n$ 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 $i, j \in I\!N\}$. 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): $\mathit{lfp}(T) = T \uparrow \omega$, 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:

$$\frac{[w_i, i_1, A_1, j_1], \ldots, [w_k, i_k, A_k, j_k]}{[w_1 \cdots w_k, i, A, j]} \quad \begin{array}{l} \textit{if } j_l = i_{l+1} \textit{ for } 1 \leq l < k \textit{ and} \\ i = i_1 \textit{ and } j = j_k \textit{ and} \\ A \rightarrow A_1, \ldots, A_k \in P \end{array}$$

When an item $[w, i, A, j]$ can be deduced, applying k times the inference rules associated with a grammar G, we write $G{\vdash^k}[w, i, A, j]$ (where $k = 1$ means using the axioms only for the deduction). 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* $T_G : \mathcal{I} \rightarrow \mathcal{I}$ *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* $A \rightarrow A_1, \ldots, A_k \in P$ *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.

– 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 $1 \leq l < k$ and $i = i_1$ and $j = j_k$ and $A \rightarrow A_1, \ldots, A_n \in P$. Furthermore, for every $1 \leq l \leq k$, 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 $1 \leq l \leq k$, $[w_l, i_l, A_l, j_l] \in T_G \uparrow (n-1)$. 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 $1 \leq l \leq k$ and $i_{l+1} = j_l$ for $1 \leq l < k$ and $i_1 = 1$ and $j_k = j$ and $A \rightarrow A_1, \ldots, A_n \in P$ and $w = w_1 \cdots w_k$. By the induction hypothesis, for every $1 \leq l \leq k$, $G\vdash^{n-1}[w_l, i_l, A_l, j_l]$, and the inference rule is applicable, so by an additional step of deduction we obtain $G\vdash^n[w, i, A, j]$.

2.4. Compositionality

The choice of semantics induces a natural equivalence relation on grammars: given a semantics '$[\![\cdot]\!]$', $G_1 \equiv G_2$ iff $[\![G_1]\!] = [\![G_2]\!]$. Recall that a semantics is *correct* if whenever $G_1 \equiv G_2$, also $Ob(G_1) = Ob(G_2)$. The way we defined it, the operational semantics of a grammar, '$[\![\cdot]\!]_{op}$', is correct (by theorem 1). By theorem 2, this holds for the denotational semantics '$[\![\cdot]\!]_{fp}$', too.

But correctness is only a sufficient condition for a good semantics. We are concerned in this paper with the composition of grammar fragments, or modules, through some composition operator defined over CFGs. In order for a semantics to be useful in this context, it must also be *compositional.* Intuitively, if a semantics is compositional, then whenever two grammars are equivalent they can be interchanged in any context without effecting the observables. This property is important for a variety of reasons: for example, when constructing an optimizing compiler for grammars, one can define a denotation-preserving optimization and use the optimized code instead of the original one without obstructing the observed behavior of the grammar. This is only possible when the semantics is known to be compositional.

DEFINITION 4. *Let '$\oplus$' be a (commutative) composition operation on grammars. A (correct) semantics '$[\![\cdot]\!]$' is* ***compositional*** *(*Gaifman and Shapiro, 1989*) with respect to '$\oplus$' if whenever $[\![G_1]\!] = [\![G_2]\!]$ and $[\![G_3]\!] = [\![G_4]\!]$, also $[\![G_1 \oplus G_3]\!] = [\![G_2 \oplus G_4]\!]$. It is* ***commuting*** *(*Brogi et al., 1992*) with respect to '$\oplus$' if there exists a combination operator on denotations, '$\bullet$', such that $[\![G_1 \oplus G_2]\!] = [\![G_1]\!] \bullet [\![G_2]\!]$.*

While we are only interested in compositionality here, commutativity is a stronger notion: if a semantics is commuting with respect to some operator then it is compositional (Wintner, 1999b). It will sometimes be easier to prove commutativity rather than compositionality.

We now define a simple composition operator, called *grammar union*, on CFGs. This is probably the most natural, even naïve, composition operator: it simply unions the components of two grammars. For real grammar development applications, this operator might turn out to be insufficient, but it seems to be a necessary one. In the rest of this work, we limit the discussion to this single grammar composition operator.

DEFINITION 5. *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 $G_1 \uplus G_2$, is a new grammar $G = \langle V_1 \cup V_2, \Sigma_1 + \Sigma_2, P_1 \cup P_2, A_1^s \cup A_2^s \rangle$.*

	G_1 :	G_2 :	$G_1 \uplus G_2$:
A^s :	$\{S\}$	$\{\}$	$\{S\}$
P :	$S \rightarrow NP\ VP$		$S \rightarrow NP\ VP$
	$VP \rightarrow V$		$VP \rightarrow V$
	$VP \rightarrow V\ NP$		$VP \rightarrow V\ NP$
		$NP \rightarrow D\ N$	$NP \rightarrow D\ N$
		$NP \rightarrow PN$	$NP \rightarrow PN$
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 2. Grammar union

Figure 2 exemplifies grammar union. Observe that for every two grammars G_1 and G_2, $G_1 \uplus G_2 = G_2 \uplus G_1$.

THEOREM 3. *'$[\![\cdot]\!]_{Op}$' is not compositional with respect to grammar union.*

Proof. Consider again the grammars of figure 2. Let G'_1 be the same as G_1, with the only difference being that $P'_1 = P_1 \setminus \{VP \rightarrow V\ NP\}$. Now $[\![G_1]\!]_{Op} = [\![G'_1]\!]_{Op} = \{[\text{loves}, i, V, i{+}1]\}_{i\geq 0} \cup \{[\text{loves}, i, VP, i{+}1]\}_{i\geq 0} \cup \{[\text{sleeps}, i, V, i+1]\}_{i\geq 0} \cup \{[\text{sleeps}, i, VP, i+1]\}_{i\geq 0}$. But when composed with G_2, both grammars yield different denotations. Since G'_1 lacks the rule $VP \rightarrow V\ NP$, no sentences with transitive verbs are generated. For example, for $x = [\text{John loves Mary}, 0, S, 3]$, $x \in [\![G_1 \uplus G_2]\!]_{Op}$ but $x \notin [\![G'_1 \uplus G_2]\!]_{Op}$. Thus, $[\![G_1]\!]_{Op} = [\![G'_1]\!]_{Op}$ but $[\![G_1 \uplus G_2]\!]_{Op} \neq [\![G'_1 \uplus G_2]\!]_{Op}$ and hence '$[\![\cdot]\!]_{Op}$' is not compositional.

Observe that the rule $VP \rightarrow V\ NP$ 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 composed with G_2, this rule can be used in a derivation, yielding $[\![G_1 \uplus G_2]\!]_{Op} \neq [\![G'_1 \uplus G_2]\!]_{Op}$. 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. For the following discussion, we fix the observables function 'Ob' and the grammar union operator '$\uplus$'.

DEFINITION 6. *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, when a composition operation on denotations is defined as:

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

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

Proof. We show that for every set of items I, $T_{G_1 \uplus G_2}(I) = T_{G_1}(I) \cup T_{G_2}(I)$.

- 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, $x \in T_{G_1 \uplus G_2}(I)$ in any case.

- if $x \in T_{G_1 \uplus G_2}(I)$ 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 $G_1 \uplus G_2$ 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 $x \in T_{G_1}(I) \cup T_{G_2}(I)$.

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 7. *A semantics '$[\![\cdot]\!]$' is* ***fully abstract*** *iff*

$$[\![G_1]\!] = [\![G_2]\!] \textit{ iff for all } G,\ Ob(G_1 \uplus G) = Ob(G_2 \uplus G)$$

Intuitively, if a semantics is compositional and fully-abstract, then two grammars are equivalent iff they can be interchanged in every context.

PROPOSITION 5. *The algebraic semantics '$[\![\cdot]\!]_{al}$' is not fully abstract.*

Proof. 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, $Ob(G \uplus G_1) = Ob(G \uplus G_2)$. The only difference between $G \uplus G_1$ and $G \uplus G_2$ 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 $G \uplus G_1$ is also derivable with $G \uplus G_2$ and hence $Ob(G \uplus G_1) = Ob(G \uplus G_2)$.

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 $[\![G_1]\!]_{id} \neq [\![G_2]\!]_{id}$, observe that $T_{G_1}(\{[w,i,A_1,j]\}) = \{[w,i,A_1,j],[w,i,A_3,j]\}$; however, $T_{G_2}(\{[w,i,A_1,j]\}) = \{[w,i,A_1,j],[w,i,A_2,j]\}$. To see that $Ob(G \uplus G_1) = Ob(G \uplus G_2)$ 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 $G \uplus G_1$ is derivable in $G \uplus G_2$ and vice versa.

4. A compositional and fully-abstract semantics

The choice of $lfp(T_G)$ as the denotation of a grammar is "too crude" (it identifies grammars that behave differently when composed 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 Maher (1984) and Maher (1988) to CFGs.

The main idea is to view the grammar as two separate units: the rules and the lexicon. The rules encode the operation of the grammar, while the lexicon (which consists of the lexical entries and also the ϵ-rules, viewed as lexical entries of the empty word) is input to this operation. This separation is motivated by both linguistic and computational considerations. Contemporary linguistic theories clearly distinguish between the grammar and the lexicon; and computational implementations such as parsers also tend to manipulate the lexicon "on demand": rather than compile the entire lexicon, only the lexical entries of words in a given input are compiled at run-time. In this section we show that viewing the lexicon as input to the grammar supports a compositional and fully-abstract definition for grammar semantics.

First, we associate the following operator with a grammar:

DEFINITION 8. *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 $i_1 = i$ 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

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

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. 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

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

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

Separating the lexicon from the grammar requires that the definition of observables be amended, to reflect the fact that the lexicon is only available at run-time. In other words, the observables of a given grammar must be defined *with respect to a given input.* Thus, let

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

A semantics is correct iff $G_1 \equiv G_2$ implies that for all I, $Ob(G_1, I) = Ob(G_2, I)$. 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, $\langle w, A\rangle \in Ob(G, I_G)$ iff $w \in L_A(G)$.*

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

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

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:

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

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

$$\begin{array}{l} [\![G_1]\!]_{fn}(\{x\}) = \{x\} \\ [\![G_2]\!]_{fn}(\{x\}) = \{x, [w, i, VP, j]\} \\ [\![G_1 \uplus G_2]\!]_{fn}(\{x\}) = \{x, [w, i, VP, j], [w, i, S, j]\} \end{array}$$

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

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 '$\uplus$'.

LEMMA 8. *For every two grammars G_1, G_2, $([\![G_1]\!]_{fn} + [\![G_2]\!]_{fn}) \leq ([\![G_1]\!]_{fn} \circ [\![G_2]\!]_{fn})$.*

Proof. Recall that if f, g are two functions over the same domain and range, $f \leq g$ iff for all I, $f(I) \subseteq g(I)$; and $f \circ g$ denotes function composition. Clearly, for every grammar G and set I,

$$(*) \quad (R_G + Id)^{\omega}(I) \supseteq I \ \text{ since } (R_G + Id)(I) \supseteq I$$

Now $([\![G_1]\!]_{fn} \circ [\![G_2]\!]_{fn})(I) = [\![G_1]\!]_{fn}([\![G_2]\!]_{fn}(I))$. $[\![G_2]\!]_{fn}(I) \supseteq I$, hence $[\![G_1]\!]_{fn}([\![G_2]\!]_{fn}(I)) \supseteq [\![G_2]\!]_{fn}(I)$. From (*) and monotonicity, also $[\![G_1]\!]_{fn}([\![G_2]\!]_{fn}(I)) \supseteq [\![G_1]\!]_{fn}(I)$. Hence $([\![G_1]\!]_{fn} \circ [\![G_2]\!]_{fn})(I) \supseteq ([\![G_1]\!]_{fn} + [\![G_2]\!]_{fn})(I)$ and $([\![G_1]\!]_{fn} + [\![G_2]\!]_{fn}) \leq ([\![G_1]\!]_{fn} \circ [\![G_2]\!]_{fn})$.

LEMMA 9. *For every grammar G, $[\![G]\!]_{fn}$ is idempotent: $[\![G]\!]_{fn} \circ [\![G]\!]_{fn} = [\![G]\!]_{fn}$.*

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

THEOREM 10. $[\![G_1 \uplus G_2]\!]_{fn} = [\![G_1]\!]_{fn} \bullet [\![G_2]\!]_{fn}$.

Proof. (Lassez and Maher, 1984)

$$\begin{array}{rcl}
[\![G_1 \uplus G_2]\!]_{fn} & = & (R_{G_1} + Id + R_{G_2} + Id)^\omega \\
& \leq & ((R_{G_1} + Id)^\omega + (R_{G_2} + Id)^\omega)^\omega \\
& & \text{since } R_G + Id \leq (R_G + Id)^\omega \text{ for every } G \\
& \leq & ((R_{G_1} + Id)^\omega \circ (R_{G_2} + Id)^\omega)^\omega \\
& & \text{by lemma 8} \\
& \leq & ((R_{G_1 \uplus G_2} + Id)^\omega \circ (R_{G_1 \uplus G_2} + Id)^\omega)^\omega \\
& & \text{since } G_1 \uplus G_2 \supseteq G_1 \text{ and } G_1 \uplus G_2 \supseteq G_2 \\
& = & ([\![G_1 \uplus G_2]\!]_{fn} \circ [\![G_1 \uplus G_2]\!]_{fn})^\omega \\
& & \text{definition of '}[\![\cdot]\!]_{fn}\text{'} \\
& = & ([\![G_1 \uplus G_2]\!]_{fn})^\omega \\
& & \text{by lemma 9} \\
& = & [\![G_1 \uplus G_2]\!]_{fn} \\
& & \text{since } (f^\omega)^\omega = f^\omega
\end{array}$$

Thus all the inequations are equations, and in particular $[\![G_1 \uplus G_2]\!]_{fn} = ((R_{G_1} + Id)^\omega + (R_{G_2} + Id)^\omega)^\omega = ([\![G_1]\!]^\omega_{fn} + [\![G_2]\!]^\omega_{fn})^\omega = ([\![G_1]\!]_{fn} + [\![G_2]\!]_{fn})^\omega = [\![G_1]\!]_{fn} \bullet [\![G_2]\!]_{fn}$.

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 11. *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, $Ob(G_1 \uplus G, I) = Ob(G_2 \uplus G, I)$, then $G_1 \equiv_{fn} G_2$.*

Proof. Assume towards a contradiction that $[\![G_1]\!]_{fn} \neq [\![G_2]\!]_{fn}$ and still for all I and all G, $Ob(G_1 \uplus G_I) = Ob(G_2 \uplus G_I)$. 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 $G = \langle V \cup \{A\}, \Sigma, P, A^s \rangle$ 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 $J = I \cup \{y_1, y_2\}$.
$\langle w, S \rangle \in Ob(G_1 \uplus G, J)$: because ρ can be applied to y_1, x, y_2, yielding exactly y.
$\langle w, S \rangle \notin Ob(G_2 \uplus G, J)$: because the only occurrence of S is in ρ, there must be an application of ρ in $G_2 \uplus G$. For ρ to apply, x must be derivable in $G_2 \uplus G$. Since neither y_1 nor y_2 can contribute to such a derivation (since B_1 and B_2 occur nowhere else in $G_2 \uplus G$), x must be

derivable in G_2, in contradiction to the assumption.
Hence $Ob(G_1 \uplus G, J) \neq Ob(G_2 \uplus G, J)$, in contradiction to the assumption. Hence '$[\![\cdot]\!]_{fn}$' is fully abstract.

5. Grammar modules

In the previous section we defined a compositional and fully-abstract semantics for context-free grammars (with respect to the grammar union operator). In order to get at the right semantics, we separated the grammar to two components – the rules and the lexicon – and viewed the rules as operating on the lexicon, which is viewed as input. We can now extend this view, and allow the grammar to operate on inputs other than the lexicon. In particular, with the semantic definition constructed in the previous section, we can view a grammar as operating on a set of items generated by some other grammar: the denotation of a grammar is a function from sets of items to sets of items, and the input to some grammar can be *any* set, not just the one constructed for the lexicon. This facilitates a natural definition for the interface between two grammars which we work out in this section.

Following Gaifman and Shapiro (1989), we define two channels for interaction among modules: 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, 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* $G = G_1 \uplus G_2$*,* $Im = (Im_1 \cup Im_2) \setminus (Ex_1 \cup Ex_2)$*,* $Ex = Ex_1 \cup Ex_2$ *and* $Int = Int_1 \cup Int_2$*.*

Figure 3 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: the observed behavior of a module is the observed behavior of its grammar component, filtered such that only exported categories can be observed.

	M_1 :	M_2 :	$M_1 \Cup M_2$:
A^s :	$\{S\}$	$\{\}$	$\{S\}$
P :	$S \rightarrow NP\ VP$		$S \rightarrow NP\ VP$
	$VP \rightarrow V$		$VP \rightarrow V$
	$VP \rightarrow V\ NP$		$VP \rightarrow V\ NP$
		$NP \rightarrow D\ N$	$NP \rightarrow D\ N$
		$NP \rightarrow PN$	$NP \rightarrow PN$
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 3. Module composition

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

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

Trivially, the module semantics '$[\![\cdot]\!]_m$' is correct with respect to the above definition. 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 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 composable then* $[\![M_1 \Cup M_2]\!]_m = [\![M_1]\!]_m \bullet [\![M_2]\!]_m$.

Proof.

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

THEOREM 13. *For every two modules* $M_1 = \langle G_1, Im_1, Ex_1, Int_1\rangle$, $M_2 = \langle G_2, Im_2, Ex_2, Int_2\rangle$, *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, $Ob(M \Cup M_1, I) = Ob(M \Cup M_2, I)$ *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 11 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, $Ob(M \Cup M_1, I) = Ob(M \Cup M_2, I)$. Let G, x, y, y_1, y_2, I, J be as in the proof of theorem 11. Let M be the module $\langle G, \emptyset, \{S\}, \{B_1, B_2\}\rangle$. Then $Ob(M_1 \Cup M, J) \neq Ob(M_2 \Cup M, J)$, 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 (where '=' means that both sides are defined and equal, or both are undefined):

PROPOSITION 14. *Module composition is commutative:* $M_1 \Cup M_2 = M_2 \Cup M_1$.

PROPOSITION 15. *Module composition is associative:* $(M_1 \Cup M_2) \Cup M_3 = M_1 \Cup (M_2 \Cup M_3)$.

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 4). 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 *RC*, imported from M_3, and M_3 imports the category S from M_1.

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

Figure 4. Three grammar modules

Observe that M_1 and M_2 are composable, since $Ex_1 \cap Ex_2 = \{S\} \cap \{NP\} = \emptyset$, $Int_1 \cap V_2 = \{V,\ VP\} \cap \{NP,\ D,\ N,\ PN,\ RC\} = \emptyset$ 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 5. This, in turn, is composable with M_3; the result of this composition yields $(M_1 \Cup M_2) \Cup M_3$ (figure 5). 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.

6. Conclusions

This paper defines modules in context-free grammars in a way that is compositional and fully-abstract (with respect to grammar union, a natural grammar composition operator). In contrast to the standard definitions for the semantics of CFGs, our definition is such that two

	$M_1 \Cup M_2$:	$(M_1 \Cup M_2) \Cup M_3$:
A^s :	$\{S\}$	$\{S\}$
P :	$S \to NP\ VP$ $VP \to V$ $VP \to V\ NP$ $NP \to D\ N$ $NP \to PN$ $NP \to D\ N\ RC$	$S \to NP\ VP$ $VP \to V$ $VP \to V\ NP$ $NP \to D\ N$ $NP \to PN$ $NP \to D\ N\ RC$ $RC \to REL\ S$
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 :	$\{S,\ NP\}$	$\{RC,\ NP,\ S\}$
Im :	$\{RC\}$	$\{\}$
Int :	$\{V,\ VP,\ D,\ N,\ PN\}$	$\{REL,\ V,\ VP,\ D,\ N,\ PN\}$

Figure 5. Composed modules

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, 1999a) 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.

Acknowledgements

This research was supported a post-doctoral fellowship from the Institute for Research in Cognitive Science, University of Pennsylvania, and by the Israel Science Foundation (grant number 136/01-1).

References

Basili, R., M. T. Pazienza, and F. M. Zanzotto: 2000, 'Customizable modular lexicalized parsing'. In: *Proceedings of the sixth international workshop on parsing technologies (IWPT 2000).* Trento, Italy, pp. 41–52.

Bredenkamp, A., T. Declerck, F. Fouvry, B. Music, and A. Theofilidis: 1997, 'Linguistic Engineering using ALEP'. In: *Proceedings of RANLP'97.* Tzigov Chark, Bulgaria.

Brogi, A., E. Lamma, and P. Mello: 1992, 'Compositional model-theoretic semantics for logic programs'. *New Generation Computing* **11**, 1–21.

Bugliesi, M., E. Lamma, and P. Mello: 1994, 'Modularity in Logic Programming'. *Journal of Logic Programming* **19,20**, 443–502.

Callmeier, U.: 2000, 'PET – a platform for experimentation with efficient HPSG processing techniques'. *Natural Language Engineering* **6**(1), 99–107.

Carpenter, B. and G. Penn: 1999, 'ALE: The Attribute Logic Engine – User's Guide'. Technical report, Lucent Technologies and Universität Tübingen.

Copestake, A.: 1999, 'The (new) LKB System'. Technical report, Stanford University.

Copestake, A. and D. Flickinger: 2000, 'An open-source grammar development environment and broad-coverage English grammar using HPSG'. In: *Proceedings of the Second conference on Language Resources and Evaluation (LREC-2000).* Athens, Greece.

Donahue, J. E.: 1976, *Complementary definitions of programming language semantics*, Vol. 42 of *Lecture notes in computer science.* Berlin, Heidelberg and New York: Springer Verlag.

Dueck, G. D. P. and G. V. Cormack: 1990, 'Modular attribute grammars'. *The Computer Journal* **33**(2), 164–172.

Erbach, G. and H. Uszkoreit: 1990, 'Grammar Engineering: Problems and Prospects'. CLAUS report 1, University of the Saarland and German research center for Artificial Intelligence.

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.

Ghezzi, C. and M. Jazayeri: 1987, *Programming language concepts.* New York: John Wiley & Sons, second edition.

Kasper, W. and H.-U. Krieger: 1996, 'Modularizing Codescriptive Grammars for Efficient Parsing'. In: *Proceedings of the 16th Conference on Computational Linguis tics.* Kopenhagen, pp. 628–633. Also available as Verbmobil-Report 140.

Keselj, V.: 2001, 'Modular HPSG'. Technical Report CS-2001-05, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.

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

Lehmann, S., D. Estival, and M. van der Kraan: 1995, 'A Modular Organization for TFS Grammar'. In: *Integrative Ansätze in der Computerlinguistik, DGfS/CL95.* Düsseldorf, pp. 55–60.

Maher, M. J.: 1988, 'Equivalences of Logic Programs'. In: J. Minker (ed.): *Foundations of Deductive Databases and Logic Programming.* Los Altos, CA: Morgan Kaufmann Publishers, Chapt. 16, pp. 627–658.

Malouf, R., J. Carroll, and A. Copestake: 2000, 'Efficient feature structure operations without compilation'. *Natural Language Engineering* **6**(1), 29–46.

Mancarella, P. and D. Pedreschi: 1988, 'An algebra of logic programs'. In: R. A. Kowalski and K. A. Bowen (eds.): *Logic Programming: Proceedings of the Fifth international conference and symposium*. Cambridge, Mass., pp. 1006–1023, MIT Press.

Milner, R.: 1975, 'Processes: a mathematical model of computing agents'. In: H. E. Rose and J. C. Shepherdson (eds.): *Logic Colloquium '73*. Amsterdam, pp. 157–174, North-Holland.

Oepen, S., D. Flickinger, H. Uszkoreit, and J.-I. Tsujii: 2000, 'Introduction to this special issue'. *Natural Language Engineering* **6**(1), 1–14.

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.

Pereira, F. C. N. and D. H. D. Warren: 1983, 'Parsing as Deduction'. In: *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*. pp. 137–144.

Sarkar, A., F. Xia, and A. Joshi: 2000, 'Some Experiments on Indicators of Parsing Complexity for Lexicalized Grammars'. In: *Efficiency in Large-Scale Parsing Systems: Workshop held at COLING 2000*. Luxembourg.

Scott, D. S. and C. Strachey: 1971, 'Towards a mathematical semantics for computer languages'. In: J. Fox (ed.): *Proceedings of the symposium on computers and automata*. New York, pp. 19–46, Polytechnic Institute of Brooklyn Press.

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

Tennet, R. D.: 1991, *Semantics of programming languages*, Prentice Hall International Series in Computer Science. Prentice Hall.

The XTAG Research Group: 1998, 'A Lexicalized Tree Adjoining Grammar for English'. IRCS Report 98–18, Institue for Research in Cognitive Science, University of Pennsylvania, 3401 Walnut St, Suite 400A, Philadelphia, PA 19104.

Theofilidis, A., P. Schmidt, and T. Declerck: 1997, 'Grammar Modularization for Efficient Processing: Language Engineering Devices and Their Instantiations'. In: *Proceedings of the DGFS/CL*. Heidelberg.

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

Wahlster, W.: 1997, 'VERBMOBIL: Erkennung, Analyse, Transfer, Generierung und Synthese von Spontansprache'. Verbmobil-Report 198 198, Deutsches Forschungszentrum für Künstliche Intelligenz, Saarbrücken, Germany.

Wintner, S.: 1999a, '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.: 1999b, 'Compositional Semantics for Unification-based Linguistic Formalisms'. IRCS Report 99-05, Institute for Research in Cognitive Science, University of Pennsylvania, 3401 Walnut St., Suite 400A, Philadelphia, PA 19018.

Wintner, S.: 1999c, 'Modularized Context-Free Grammars'. In: *MOL6 – Sixth Meeting on Mathematics of Language*. Orlando, Florida, pp. 61–72.

Wintner, S. and N. Francez: 1999, 'Efficient Implementation of Unification-Based Grammars'. *Journal of Language and Computation* **1**(1), 53–92.

Woszczyna, M., M. Broadhead, D. Gates, M. Gavaldà, A. Lavie, L. Levin, and A. Waibel: 1998, 'A modular approach to spoken language translation for large domains'. In: *Proceedings of AMTA-98*.

Zajac, R. and J. W. Amtrup: 2000, 'Modular Unification-Based Parsers'. In: *Proceedings of the sixth international workshop on parsing technologies (IWPT 2000)*. Trento, Italy, pp. 278–288.

Address for Offprints: Department of Computer Science, University of Haifa, Mount Carmel, 31905 Haifa, Israel