WPI-CS-TR-99-03 January 1999

Recursive Adaptable Grammars

by

John N. Shutt

Computer Science
Technical Report
Series

WORCESTER POLYTECHNIC INSTITUTE

Computer Science Department
100 Institute Road, Worcester, Massachusetts 01609-2280

Recursive Adaptable Grammars

John N. Shutt
jshutt@cs.wpi.edu
Computer Science Department
Worcester Polytechnic Institute
Worcester, MA 01609

January 1999

Abstract

This paper presents the Recursive Adaptable Grammar (RAG) formalism. RAGs allow arbitrary Turing-powerful language analysis to be described entirely in terms of a single level of "context-free" derivation. RAGs superficially resemble a limited form of Extended Attribute Grammars (EAGs); but while the EAG derivation step relation may entail arbitrary subsidiary computations, the normal RAG derivation step relation is elementary. The RAG formalism is introduced and defined; basic properties of the formalism are shown; and a well-behavedness property called *strong answer-encapsulation* is developed.

Contents

1 Introduction **1**

2 Mathematical preliminaries **3**

3 RAGs **4**
- 3.1 Overview 5
- 3.2 Algebras 7
- 3.3 Grammars 9
- 3.4 Derivation 9

4 An example **10**

5 Basic results **12**

6 Answer-encapsulation **14**

7 Normal RAGs **17**

8 Conclusions **20**

Acknowledgments **21**

Bibiography **21**

List of Definitions

3.1 Vocabulary . . . 7
3.2 Terminal algebra . . . 7
3.3 Answer algebra . . . 7
3.4 Query algebra . . . 8
3.5 Configuration algebra . . . 8
3.6 Unbound rule . . . 9
3.7 Bound rule . . . 9
3.8 Instance of an unbound rule . . . 9
3.9 Recursive adaptable grammar . . . 9
3.10 Bound rule set of a RAG . . . 9
3.11 RAG derivation . . . 9
4.1 Rule equation . . . 10
4.2 Concatenation of rule equations . . . 11
6.1 Semantic equivalence . . . 14
6.4 Weak answer-encapsulation . . . 15
6.5 Safe operator . . . 15
6.6 Strong answer-encapsulation . . . 15
7.1 Non-circular RAG, left-to-right RAG . . . 17
7.2 Normal RAG . . . 18

List of Theorems

2.1 Existence of Ω-extensions . . . 4
5.1 Termination of RAG derivations . . . 12
5.2 Equivalence of query and pair derivations . . . 13
6.2 $\equiv_G$ is an answer equivalence . . . 14
6.3 (cor) Answer-euqivalence of terminals . . . 15
6.7 (lem) Answer-encapsulation; safe frameworks . . . 15
6.8 (lem) Distribution of concatenation over derivation . . . 16
6.9 Safety of concatenation . . . 16
6.10 Safety of union, mapping, star . . . 17

6.11 Existence of unsafe frameworks 17
7.3 Normal RAGs accept all r.e. languages 18
7.4 Normal RAGs compute all r.e. functions 19

List of Equations

1 Binding of rules based on ρ_G 9
2 Rule equation for terminals 11
3 Rule equation for concatenation 11
4 Rule equation for union 16
5 Rule equation for mapping 17
6 Rule equation for star 17
7 An inherently unsafe rule equation 17

1 Introduction

Historically, formal grammar models have run a three-legged race between technical simplicity, conceptual clarity, and computational power. Technical simplicity makes a model easy to work with mathematically; conceptual clarity makes it easy to work with practically; and computational power makes it applicable to more, and more substantial, languages. Context-free grammars (CFGs) are technically simple and conceptually elegant, but lack power. Unrestricted Chomsky grammars are technically simple and Turing-powerful, but in general rather opaque. Attribute grammars are understandable (up to a point) and Turing-powerful, but technically elaborate, involving arbitrary extra-derivational computations. And so on.

The Recursive Adaptable Grammar (RAG) model is designed to realize all three of these desirable qualities in a single framework. More precisely, it is meant to combine the following four properties. (Technical simplicity and conceptual clarity are distributed across Properties 2–4.)

1. The model is Turing-powerful.

2. The central computational mechanism of the model is the reflexive transitive closure of a derivation step relation (accommodating inductive reasoning).

3. The derivation step relation is *elementary*, in the sense that it is generated in a computationally trivial way (facilitating inductive reasoning).

4. The step relation is generated from a set of production rules with *one-to-many structure* (supporting derivation trees).

Unrestricted Chomsky grammars have all but Property 4; CFGs have all but Property 1. Knuth's Attribute Grammars (AGs; [Knut90]) achieve Turing power by adding a 'semantic' phase of computation separate from the derivation relation; consequently, AGs have all but Property 2 — the derivation relation is no longer central to computation, but merely the less powerful, though more lucid, half of it.

Extended Attribute Grammars (EAGs; [Mads80]) are a reformulation of AGs that restores Property 2 by absorbing the semantic phase of computation into the derivation step relation. A finite description of the grammar is used to generate a (usually) infinite set of production rules, using an infinite nonterminal alphabet constructed by attaching lists of attribute values to atomic nonterminal symbols; derivation then proceeds in the usual CFG fashion. Although the derivation step relation in EAGs is once again central to the computational process, it is no longer *elementary*: The generation of production rules may require use of arbitrarily powerful semantic functions. Hence, EAGs have all but Property 3. Other prominent two-level grammar models, such as van Wijngaarden grammars[1], similarly lack Property 3.

[1]A straightforward technical explanation of van Wijngaarden grammars is [Shut93, §2.3.1]. The original paper is [Wijn65].

Recursive Adaptable Grammars (RAGs) use a technique similar to that of EAGs to generate an infinite set of production rules, but (normally) do not perform any substantial computation on semantic expressions during rule generation. Instead, semantic evaluation takes place within the derivation itself, across an unbounded number of additional derivation steps — unbounded because semantic evaluation "recursively" employs the same facilities as any other RAG derivation, and is thus Turing-powerful.

This paper presents the basic theory of RAGs. The RAG model was first proposed in [Shut93]; definitions here differ slightly from, and supersede, the earlier work.

Familiarity with the basic concepts of initial algebra semantics is assumed; particulars of the approach used here are summarized in §2. Some familiarity with EAGs would be helpful.

RAGs and RAG derivation are defined and explained in §§3–4. §5 proves some basic formal properties of all RAGs; §6 treats an important behavioral restriction called *strong answer-encapsulation*; and §7 develops a RAG normal form. §8 sums up the preceding material, and briefly discusses some possible research applications of the RAG model.

Note: Adaptable grammars

Most programming languages are not context-free, because of context-dependent features such as lexical scope or static typing; but they are what might be termed *locally context-free*, in that the set of permissible expressions within a small region in a particular program can be described by a CFG [Cara63]. Over the years, a number of grammar models have been proposed that attempt to capture this intuitive notion by means of a varying set of production rules [Chri90, Shut93]. Grammars under such models are termed *adaptable grammars*.

More precisely, a grammar model is considered adaptable iff it allows for the explicit manipulation of rule sets from within the grammar. Adaptable grammar models are classified as *imperative* or *declarative*.

An imperative adaptable grammar varies the rule set between sentential forms in a derivation. Each sentential form thus has its own associated rule set. A more concrete way of putting it is that the rule set varies over time during parsing. Unfortunately, the consequences of associating a given rule set with a given sentential form depend on what criterion will be used to decide which rule to apply: In effect, the meaning of the grammar is dependent on the parsing algorithm.

A declarative adaptable grammar varies the rule set between nodes in a derivation tree. The rule set is thus essentially an attribute (as in AGs), with rule set variations following the structure of the derivation tree. The meaning of a declarative adaptable grammar is independent of order of rule application.

In addition to its purely conceptual merits, declarative grammar adaptability offers a way to describe non-context-free language features in terms of context-free structure, increasing the computational power that can be achieved without resorting to more opaque techniques such as arbitrary semantic functions. Therefore, it was adopted as a basic strategy in the design of the RAG model. RAGs are declarative adaptable grammars.

2 Mathematical preliminaries

This section clarifies certain basic mathematical terminology, notation, etc. that will be assumed hereafter, but which may vary between authors. Most of the section concerns *universal algebra* (also called *one-sorted algebra*); note particularly that the definitions of Σ-*morphism* and Σ- and Ω-*extension* are more general than those used by many authors.

Set difference is denoted $A - B = \{x \mid (x \in A) \wedge (x \notin B)\}$. The power set, or set of all subsets, of a set A is $\mathcal{P}(A)$.

A *signature* Σ is a set of operators indexed by arity; arities are nonnegative integers. The set of operators of arity n in Σ is denoted Σ_n. A Σ-*algebra* A consists of a set called the *carrier* of A, and for each operator $\sigma \in \Sigma_n$ a function σ_A of arity n over the carrier of A, called the operation named by σ. (Notation σ_A will be strictly observed only in this section; elsewhere, the operation named by σ may be ambiguously denoted σ.) The carrier of A is ambiguously denoted A, hence $\sigma_A : A^n \to A$. A function of arity 0 is treated as a constant, hence $\sigma \in \Sigma_0$ implies $\sigma_A \in A$. The signature of an algebra A is denoted Σ_A.

For algebras A and B and signature $\Sigma \subseteq \Sigma_A \cap \Sigma_B$, a Σ-*morphism* f from A to B is a function from the carrier of A to the carrier of B, such that f distributes over all the operations named by Σ; that is, $\sigma \in \Sigma_n$ implies $f(\sigma_A(a_1, \dots, a_n)) = \sigma_B(f(a_1), \dots, f(a_n))$.[2] All Σ-algebras together with all Σ-morphisms between them form a category $\mathbf{Alg}_\Sigma$.

An expression formed from operators in signature Σ, conforming to the arities of the operators, is called a Σ-term. The *term algebra* of Σ, denoted T_Σ, has as carrier the set of all Σ-terms, and for each operator $\sigma \in \Sigma_n$ the obvious construction operation, $\sigma_{T_\Sigma}(\tau_1, \dots, \tau_n) = \sigma(\tau_1, \dots, \tau_n)$. T_Σ is initial on $\mathbf{Alg}_\Sigma$. For any algebra A, the unique Σ_A-morphism from T_{Σ_A} to A is denoted eval_A. An algebra A is *minimal* iff eval_A is surjective.

A *variable set* V over signature Σ is an ordered set of symbols disjoint from Σ. For variable set Y over Σ, $\Sigma(Y)$ denotes the signature constructed by adding the variables of Y to Σ_0. Terms over $\Sigma(Y)$ are called *polynomials* in variables Y over Σ. The term algebra of polynomials in Y over (the signature of) algebra A is denoted $A(Y) = T_{\Sigma_A(Y)}$. The number of distinct variables in a polynomial π is its *arity*, denoted $\mathrm{ar}(\pi)$. If $\pi_0, \dots, \pi_n \in A(Y)$, $\mathrm{ar}(\pi_0) = n$, and $x_1, \dots, x_n$ are the distinct

[2]Some authors require the domain and codomain of a Σ-morphism (here, A and B) to be Σ-algebras. The definition here is more permissive.

variables of π_0 in the order imposed by Y, then the construct $\pi_0(\pi_1, \ldots, \pi_n)$ denotes the polynomial obtained from π_0 by replacing each instance of x_k in π_0 with π_k.

By convention, variable sets are implicitly assumed to be disjoint from all signatures under consideration, except when explicitly included by construction, as in a polynomial signature $\Sigma(Y)$. One therefor speaks simply of a *variable set*, without saying what signature it is over.

For given algebra A, variables Y, and assignment $\theta : Y \to A$, there is exactly one Σ_A-morphism $\overline{\theta} : A(Y) \to A$ such that $\forall y \in Y,\ \overline{\theta}(y) = \theta(y)$. A Σ-equation is a pair of polynomials over signature Σ. A Σ-algebra A *satisfies* Σ-equation $e = \langle \pi_1, \pi_2 \rangle$ iff for every possible assignment $\theta : Y \to A$ of the variables Y in e, $\overline{\theta}(\pi_1) = \overline{\theta}(\pi_2)$. A Σ-equation $\langle \pi_1, \pi_2 \rangle$ is conventionally written $\pi_1 = \pi_2$.

A *specification* is a pair $\Omega = \langle \Sigma, \mathcal{E} \rangle$ of a signature Σ and a set $\mathcal{E}$ of Σ-equations. An Ω-algebra is then a Σ-algebra that satisfies all the equations in $\mathcal{E}$. $\mathbf{Alg}_\Omega$ is the category of all Ω-algebras together with all Σ-morphisms between them. For any possible specification Ω, there exists an initial algebra on $\mathbf{Alg}_\Omega$; further, this initial algebra is minimal.

The *string specification* is $\Omega_{\text{str}} = \langle \Sigma_{\text{str}}, \mathcal{E}_{\text{str}} \rangle$, where Σ_{str} consists of the empty string and binary concatenation, denoted respectively by λ and juxtaposition; and $\mathcal{E}_{\text{str}}$ consists of equations $x\lambda = x$, $\lambda x = x$, and $x(yz) = (xy)z$, in variables x, y, z. For finite alphabet Z, treated as a signature whose operators are all of arity zero, the *string algebra* over Z, denoted Z^*, is the initial algebra over $\Omega_{\text{str}} \cup Z = \langle \Sigma_{\text{str}} \cup Z, \mathcal{E}_{\text{str}} \rangle$.

An algebra B is an *extension* of an algebra A iff there is an injective Σ_A-morphism from A to B. If A is minimal and B is an extension of A, it is conventionally assumed that the carrier of A is a subset of the carrier of B, and the Σ_A-morphism between them is the inclusion mapping $(a \mapsto a)$; one then writes $A \subseteq B$. For signature Σ, algebra B is a Σ-*extension* of algebra A iff B is an extension of A and $\Sigma_B - \Sigma_A \subseteq \Sigma \subseteq \Sigma_B$. For specification $\Omega = \langle \Sigma, \mathcal{E} \rangle$, algebra B is an Ω-*extension* of algebra A iff B is a Σ-extension of A and B satisfies all the equations in $\mathcal{E}$.

The following result will be needed for certain basic constructions in §3.

Theorem 2.1 Suppose A is a minimal algebra, $\Omega = \langle \Sigma, \mathcal{E} \rangle$ a specification, and $\mathbf{C}$ the category of all Ω-extensions of A together with all $(\Sigma \cup \Sigma_A)$-morphisms between them. If $\mathbf{C}$ is nonempty, then there exists an initial algebra on $\mathbf{C}$, and further, this initial algebra is minimal.

A proof of this result constructs specification $\Omega' = \langle \Sigma \cup \Sigma_A, \mathcal{E} \cup \{\tau = \tau' \mid \text{eval}_A(\tau) = \text{eval}_A(\tau')\} \rangle$, and shows that the initial algebra on $\mathbf{Alg}_{\Omega'}$ is initial on $\mathbf{C}$ if $\mathbf{C}$ is nonempty. Details occur in [Shut93, §5.2.4].

3 RAGs

The first subsection below introduces basic ideas underlying RAG derivation. Its purpose is to provide a conceptual framework, into which the elements of the formal model can be placed. No attempt is made to provide a complete picture of the formalism; that is left for §§3.2–3.4, which define the formal elements of the model in stages, motivated in terms of the conceptual framework.

3.1 Overview

A RAG derivation tree is decorated with values from an algebraic domain called an *answer algebra*; such values are called *answers*. The answer algebra of a RAG G is denoted A_G. Each leaf node is labeled by a single answer, called the *syntax* at that node. Each parent node is labeled by an ordered pair of answers, called respectively the *metasyntax* and *semantics* at that node.

- Syntax plays the same role in RAG derivation as terminal symbols, and terminal strings, do in CFG derivation.
- Semantics is, conceptually, a synthesized attribute. The semantic value at a parent node p is usually thought of as denoting the "meaning" of the syntax on (the fringe of) the branch descending from p. The semantics at the root node of the tree is thus understood as the meaning of the entire sentence derived by the tree. CFGs have no counterpart to the semantic role.
- Metasyntax is conceptually an inherited attribute. It plays essentially the same role in RAG derivation as nonterminal symbols do in CFG derivation: It denotes the set of possible branches that can descend from the parent node p on which it occurs, including the syntax at the fringe, the semantics at p, and everything between. The metasyntax of the root node entirely determines what language is being generated; in the canonical case, the metasyntax at the root is a designated element of A_G called the *start symbol* of G.

Figure 1 shows a RAG derivation tree. The grammar used is one that will be explored in detail in §4, following the formal definition of the model in §§3.2–3.4. The terminal alphabet of the grammar is $Z = \{a, b, c\}$, and the language accepted is $L(G) = \{www \mid w \in Z^*\}$. The tree shown derives the sentence *ababab*.

The basic strategy of the grammar is that metasyntactic value star(letter) accepts any string $w \in Z^*$, and synthesizes w as its semantic value. The remaining two branches of the tree then use this synthesized value as metasyntax. In any RAG, when a terminal string is used as metasyntax, it generates itself as syntax and synthesizes itself as semantics; so these two branches generate the same string w as the first did. String w is also used as the semantics at the root node.

Superficially, this tree is generated by syntax production rules

$$\langle s, ab\rangle \to \langle \text{star(letter)}, ab\rangle\langle ab, ab\rangle\langle ab, ab\rangle$$

and so on. These are called *bound rules*, and actually occur only toward the *end* of the process that generates the derivation step relation.

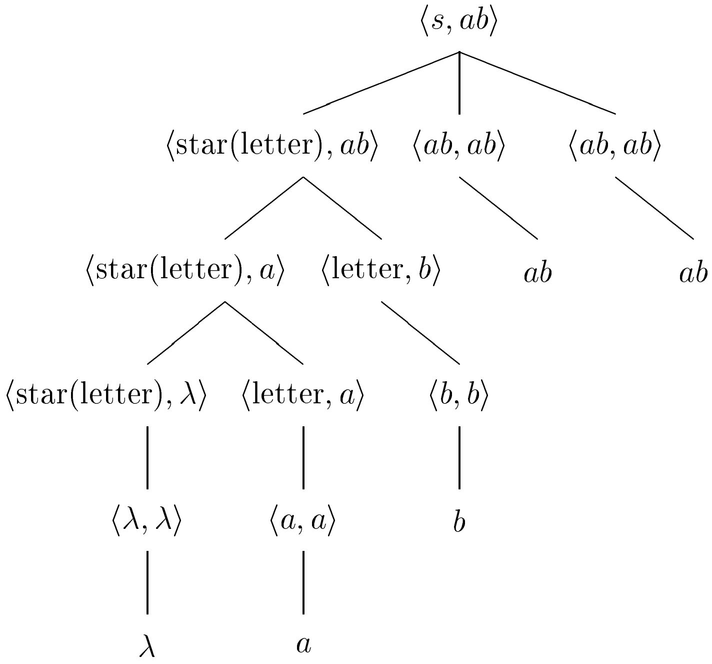

Figure 1: RAG derivation tree

The step relation generation process begins with a function called the *rule function of* G, conventionally denoted ρ_G or simply ρ, which maps each possible metasyntactic value $x \in A_G$ into a set $\rho(x)$ of *unbound rules*, which have the same general structure as bound rules, but with unbound variables in positions ordinarily provided from elsewhere in the tree (inherited metasyntax on the left side of the rule, synthesized semantics on the right); and polynomials in those positions presumably computed by the rule (metasyntax on the right side, semantics on the left). For example, the aforementioned bound rule is instantiated from the unbound rule

$$\langle v_0, v_1\rangle \quad\rightarrow\quad \langle \text{star(letter)}, v_1\rangle\langle v_1, v_2\rangle\langle v_1, v_3\rangle$$

with bindings $v_0 = s$ and $v_1 = v_2 = v_3 = ab$.

The metasyntactic variable v_0 on the left side of an unbound rule r must be bound to a metasyntactic value $x \in A_G$ such that $r \in \rho(x)$ — that is, an answer x that "owns" r. v_0 is thus analogous to `self` in Smalltalk, or `this` in Java or C++. The remaining variables in r can be bound to any answers at all. However, only certain choices of values will lead to successful derivations; for example, in the above unbound rule, no complete derivation can result unless $v_1 = v_2 = v_3$.

3.2 Algebras

The theory of a CFG is founded on two finite alphabets, one nonterminal and one terminal; call them N and Z. Two infinite domains are constructed from these alphabets and play essential roles in the theory: the set $(N \cup Z)^*$ of strings over the combined alphabets, and the set Z^* of terminal strings. The derivation step is a binary relation on $(N \cup Z)^*$, while the language generated is a subset of Z^*.

The theory of a RAG is founded on a nonterminal *signature* Σ and terminal alphabet Z. A hierarchy of four algebras is constructed from these elements — from smallest to largest, the *terminal*[3], *answer*, *query*, and *configuration* algebras. The RAG derivation step will be defined (in §3.4) as a binary relation on the configuration algebra; the language generated is a subset of the answer algebra (and, usually but not necessarily, of the terminal algebra).

The following three operators are reserved for special uses in the RAG model: The *query operator*, shorthand name *qry*, of arity 2, with infix notation $x : y$; the *pairing operator*, shorthand *pair*, arity 2, notation $\langle x, y \rangle$; and the *inverse operator*, shorthand *inv*, arity 1, notation $\overline{x}$. It will be convenient to associate with each of these reserved operators σ a signature Σ_σ consisting of only that operator; hence, Σ_{qry}, Σ_{pair}, and Σ_{inv}.

Definition 3.1 A *vocabulary* is a pair $V = \langle \Sigma, Z \rangle$, where Σ is a signature; Z is a finite alphabet; Σ is disjoint from the string signature over Z; and Σ does not contain the query, pairing, or inverse operator. Z is called the *terminal alphabet*, and Σ the *nonterminal signature*.

In the example from §3.1 above, the terminal alphabet is $\{a, b, c\}$, and there are three nonterminal operators: nonterminal constants (operators of arity zero) s and *letter*, and nonterminal unary operator *star*.

Definition 3.2 Given vocabulary $V = \langle \Sigma, Z \rangle$, the string algebra Z^* is called the *terminal algebra* over V, and denoted $T_V = Z^*$. Elements of T_V are called *terminals*.

Definition 3.3 Given vocabulary $V = \langle \Sigma, Z \rangle$, the initial $(\Sigma \cup \Omega_{\text{str}})$-extension of T_V is called the *answer algebra* over V, and denoted A_V. Elements of A_V are called *answers*. Elements of $A_V - T_V$ are called *nonterminals*.

The explicit inclusion of the string specification Ω_{str} in the construction of A_V causes the usual laws of concatenation (associativity, left/right identity) to apply to nonterminal strings. Had A_V been defined simply as the initial Σ-extension of T_V, the

[3]The name *terminal algebra* here clashes with category-theory usage by some authors, for whom *terminal algebra* is the dual of *initial algebra*. Mitigating this difficulty, the term *final algebra* is also commonly used for the categorical concept.

usual laws would not apply to $A_V - T_V$, because of the way algebraic extensions were defined in §2.

As noted in §3.1 above, the values that decorate derivation trees belong to the answer algebra. There is no formal prohibition against nonterminal syntax, although in practice, syntax is usually terminal because the subjective purpose of constructing a RAG is usually to define a language of terminal strings. From a strictly formal perspective, terminals will play no special role until the introduction of *answer-encapsulation* in §6.

Definition 3.4 Given vocabulary $V = \langle \Sigma, Z \rangle$, the *query algebra* over V, denoted Q_V, is the initial Σ_{qry}-extension of A_V. Elements of Q_V are *queries*.

The existence of Q_V for every vocabulary V is guaranteed by Theorem 2.1.

Recall from §1 that in the RAG model, "semantic evaluation takes place within the derivation itself, across an unbounded number of additional derivation steps". The query operator is the vehicle through which this evaluation occurs. For answers x and y, query $x : y$ designates the semantic value synthesized when metasyntax x recognizes syntax y. In the example from §3.1 above, $s : ababab$ would evaluate, across multiple derivation steps, to value ab.

Definition 3.5 Given vocabulary $V = \langle \Sigma, Z \rangle$, the *configuration algebra* over vocabulary V, denoted C_V, is an extension of A_V as follows. Suppose v is a variable. Let Ω, Ω' be the specifications

$$\begin{aligned} \Omega &= \langle \Sigma_{\mathrm{inv}}, \{v = \overline{v}\} \rangle \\ \Omega' &= \langle (\Sigma_{\mathrm{qry}} \cup \Sigma_{\mathrm{pair}} \cup \Sigma_{\mathrm{inv}}), \{v = \overline{\overline{v}}\} \rangle \end{aligned}$$

Then C_V is the initial Ω'-extension of the initial Ω-extension of A_V. Elements of C_V are *configurations*.

The existence of C_V for every vocabulary V is guaranteed by Theorem 2.1. By construction, $T_V \subseteq A_V \subseteq Q_V \subseteq C_V$.

By the two-stage construction of C_V from A_V, equation $v = \overline{\overline{v}}$ holds for all configurations v, but $v = \overline{v}$ holds only when v is an answer. This is because, under the definitions in §2, if E is an Ω-extension of D, then the elements of the difference set $E - D$ are constrained only by the equations in Ω, not by any other equations that may hold in D.

The pairing operator is used in bound and unbound rules, as illustrated above in §3.1, to group together a metasyntactic value with its associated semantic value. The inverse operator, on the other hand, is a purely technical contrivance. It never occurs in bound or unbound rules; it only arises in intermediate configurations in a derivation, where it is used to regulate the process of query evaluation.

3.3 Grammars

Definition 3.6 An *unbound rule* over vocabulary V is a structure of the form

$$\langle v_0, e_0 \rangle \quad \rightarrow \quad t_0 \langle e_1, v_1 \rangle t_1 \langle e_2, v_2 \rangle t_2 \cdots \langle e_n, v_n \rangle t_n$$

where $n \geq 0$, the t_k are answers over V, the v_k are variables, and the e_k are polynomials in $Q_V(v_0, \ldots, v_n)$. The domain of all unbound rules over V is denoted $\mathcal{R}_V$.

Definition 3.7 A *bound rule* over vocabulary V is a structure of the form

$$\langle a_0, q_0 \rangle \quad \rightarrow \quad t_0 \langle q_1, a_1 \rangle t_1 \langle q_2, a_2 \rangle t_2 \cdots \langle q_n, a_n \rangle t_n$$

where $n \geq 0$, the t_k and a_k are answers over V, and the q_k are queries over V.

Definition 3.8 For unbound rule r over V, an *instance* of r is a bound rule obtained by assigning answers over V to the variables in r, and evaluating both sides of r in C_V under that assignment. The set of all instances of unbound rule r over V is denoted $\beta_V(r)$.

Note that the variables in an unbound rule r are assigned values from A_V, but the left and right sides of r are polynomials over C_V, not A_V. A bound rule is a pair of configurations.

Definition 3.9 A *recursive adaptable grammar (RAG)* is a four-tuple $G = \langle \Sigma, Z, \rho, s \rangle$, where $V = \langle \Sigma, Z \rangle$ is a vocabulary, $\rho : A_V \rightarrow \mathcal{P}(\mathcal{R}_V)$, and $s \in A_V$. ρ is the *rule function*, and s the *start symbol*, of G. Subscript G may be used in place of subscript V wherever the latter could occur; thus answer algebra A_G, etc. The rule function of an arbitrary grammar G is denoted ρ_G, and the start symbol s_G.

3.4 Derivation

Definition 3.10 For RAG G and answer $a \in A_G$, $\beta_G(a)$ denotes the following set of bound rules.

$$\beta_G(a) \quad = \quad \{\langle a, q \rangle \rightarrow c \mid (\langle a, q \rangle \rightarrow c) \in \beta_G(r) \text{ for some } r \in \rho_G(a)\} \tag{1}$$

The union of $\beta_G(a)$ for all answers $a \in A_G$ is denoted $\beta(G)$.

Set $\beta_G(a)$ is analogous to the set of all rules in a CFG with a particular nonterminal a on their left hand sides. The constraint built into Equation 1 is that, when binding an unbound rule $(\langle v_0, e_0 \rangle \rightarrow \pi) \in \rho(a)$, the leftmost variable v_0 must be bound to a. Informally, §3.1 noted the resemblance of v_0 to `self` in OOP. Formally, in derivations this constraint will prevent a pair $\langle a, c \rangle$ from being expanded via an instance of an unbound rule r unless $r \in \rho(a)$. The left side of the pair may, consequently, be thought of as an inherited attribute that determines how a derivation tree can grow downward from a given node; cf. the note on adaptable grammars at the end of §1.

Definition 3.11 The *derivation step relation* for a RAG G is the minimal binary relation $\underset{G}{\Rightarrow}$ over C_G satisfying the following axioms. Throughout the axioms, a, q, and c, without or without subscripts or primes, are assumed to be universally quantified over A_G, Q_G, and C_G, respectively.

> **Axiom 3.11a (rule application)** If $(\langle a, q\rangle \to c) \in \beta(G)$ then $\langle a, \overline{q}\rangle \underset{G}{\Rightarrow} c$.
>
> **Axiom 3.11b (substitution)** Suppose $\sigma \neq \mathrm{inv}$ is an operator of arity n over C_G, and for some $1 \leq k \leq n$, $c_k \underset{G}{\Rightarrow} c'_k$. Let $c'_j = c_j$ for all $j \neq k$. Then $\sigma(c_1, \ldots, c_n) \underset{G}{\Rightarrow} \sigma(c'_1, \ldots, c'_n)$.
>
> **Axiom 3.11c (inversion)** If $c_1 \underset{G}{\Rightarrow} c_2$ then $\overline{c_2} \underset{G}{\Rightarrow} \overline{c_1}$.
>
> **Axiom 3.11d (query reduction)** $a_1 : \overline{\langle a_1, a_2\rangle} \underset{G}{\Rightarrow} a_2$.

The *derivation relation* for G is the reflexive transitive closure of $\underset{G}{\Rightarrow}$, denoted $\overset{*}{\underset{G}{\Rightarrow}}$. The transitive closure of $\underset{G}{\Rightarrow}$ is denoted $\overset{+}{\underset{G}{\Rightarrow}}$. The *language accepted* by an answer $a \in A_G$ is $L_G(a) = \{x \in A_G \mid \langle a, y\rangle \overset{*}{\underset{G}{\Rightarrow}} x$ for some $y \in A_G\}$. The language accepted by G is $L(G) = L_G(s_G)$. The (in general, multivalued) partial function *computed* by an answer $a \in A_G$ is the function f_a such that $\forall x, y \in A_G,\ f_a(x) = y$ iff $a : x \overset{*}{\underset{G}{\Rightarrow}} y$. The partial function computed by G is $f_G = f_{s_G}$.

Axioms 3.11a and 3.11b closely parallel the definition of a Chomsky derivation step relation. Axiom 3.11a corresponds to the Chomsky proposition that $x \to y$ implies $x \Rightarrow y$ (modulo the Axiom's telltale use of the inverse operator). Axiom 3.11b yields as a corollary the proposition that $x \underset{G}{\Rightarrow} y$ implies $pxq \underset{G}{\Rightarrow} pyq$.

Axioms 3.11c and 3.11d only come into play if the query operator is used. Their net effect (in concert with the use of the inverse operator in Axiom 3.11a) is the following result, which will be proven later as Theorem 5.2.

$$\forall x, y, z \in A_G, \qquad x : y \overset{*}{\underset{G}{\Rightarrow}} z \quad \text{iff} \quad \langle x, z\rangle \overset{*}{\underset{G}{\Rightarrow}} y$$

As discussed in §3.1 above, the elements x, y, z in this ternary relation are the *meta-syntax* x, *syntax* y, and *semantics* z. In a derivation tree based on the ternary relation, the root node would be labeled by $\langle x, z\rangle$, while the fringe of the tree would be y. The ordinary derivation $\langle x, z\rangle \overset{*}{\underset{G}{\Rightarrow}} y$ may be thought of as constructing this derivation tree from the top down. It will emerge in §§4–5 that, in effect, the query derivation $x : y \overset{*}{\underset{G}{\Rightarrow}} z$ constructs the same derivation tree from the bottom up (achieving this reversal of direction by means of the inverse operator).

4 An example

Just as a CFG is often specified by simply enumerating its production rules, leaving the nonterminal and terminal alphabets implicit, a RAG may be specified by a series of equations that define the behavior of its rule function.

Definition 4.1 Suppose G is a RAG, and Σ a subset of the signature of A_G. A *rule equation over* Σ is an equation of the form

$$\rho(\sigma(x_1,\ldots,x_n)) \quad = \quad f(x_1,\ldots,x_n,\rho(x_1),\ldots,\rho(x_n))$$

where $\sigma \in \Sigma_n$, the x_k are variables, $f : A_G^n \times \mathcal{P}^n(\mathcal{R}_G) \to \mathcal{P}(\mathcal{R}_G)$, and for all inputs ω to f, every answer operator (i.e., terminal or nonterminal operator) that occurs in $f(\omega)$ also occurs in either Σ or ω. The rule equation *specifies* σ. G *satisfies* the rule equation iff the equation holds for $\rho = \rho_G$ when the x_k are universally quantified over A_G.

Conventionally, rule equations are provided for each of the nonterminal operators, and the start symbol is assumed to be nonterminal constant s. The rule equations for terminal operators are not specified. It is assumed that

$$\forall t \in T_G, \qquad \rho_G(t) \quad = \quad \{\langle v_0, t\rangle \to t\} \tag{2}$$

Hence $t\!:\!t \overset{*}{\underset{G}{\Rrightarrow}} t$. The assumed rule equation for binary concatenation, although conceptually quite simple, requires some technical care to avoid variable-name collisions. It uses the auxiliary concept of concatenation of unbound rules.

Definition 4.2 Suppose r and r' are unbound rules. The *concatenation* rr' is constructed as follows. Let $r = (\langle v_0, e_0\rangle \to \pi)$ and $r' = (\langle v_0, e_0'\rangle \to \pi')$. Assume without loss of generality that v_0 is the only variable-name shared by r and r'. Then $rr' = (\langle v_0, e_0 e_0'\rangle \to \pi\pi')$.

The conventional rule equation is then

$$\rho(ab) \quad = \quad \{r_a r_b \mid (r_a \in \rho(a)) \wedge (r_b \in \rho(b))\} \tag{3}$$

The grammar in the following example is the same one used earlier in §3.1.

Example 4.3 Let Z be the alphabet $Z = \{a, b, c\}$, and let G be the following RAG.

$$\begin{array}{rcl}
\rho(\text{letter}) & = & \left\{\ \langle v_0, v_1\rangle \to \langle z, v_1\rangle \quad \middle| \quad z \in Z\ \right\} \\
\rho(\text{star}(x)) & = & \left\{ \begin{array}{ccc} \langle v_0, v_1\rangle & \to & \langle \lambda, v_1\rangle \\ \langle v_0, v_1 v_2\rangle & \to & \langle \text{star}(x), v_1\rangle\langle x, v_2\rangle \end{array} \right\} \\
\rho(s) & = & \left\{\ \langle v_0, v_1\rangle \quad \to \quad \langle \text{star(letter)}, v_1\rangle\langle v_1, v_2\rangle\langle v_1, v_3\rangle\ \right\}
\end{array}$$

Nonterminal constant *letter* accepts a single letter $z \in Z$, and synthesizes semantic value z. To be precise, $(\langle \text{letter}, z\rangle \to \langle z, z\rangle) \in \beta_G(\text{letter})$, therefore by Axiom 3.11a, $\langle \text{letter}, z\rangle \underset{G}{\Rrightarrow} \langle z, z\rangle$; and by Equation 2, $(\langle z, z\rangle \to z) \in \beta_G(z)$, therefore by Axiom 3.11a, $\langle z, z\rangle \underset{G}{\Rrightarrow} z$. Putting these together,

$$\langle \text{letter}, z\rangle \quad \underset{G}{\Rrightarrow} \quad \langle z, z\rangle \quad \underset{G}{\Rrightarrow} \quad z$$

or, in brief, $\langle \text{letter}, z\rangle \underset{G}{\overset{*}{\Rightarrow}} z$. The language accepted by *letter* is $L_G(\text{letter}) = Z$.

Nonterminal operator *star* takes an argument x, accepts the Kleene closure of $L_G(x)$, and synthesizes the concatenation of the semantic values assigned by x to the substrings. In this case, the argument is *letter*, and

$$\rho(\text{star}(\text{letter})) \quad = \quad \left\{ \begin{array}{rcl} \langle v_0, v_1\rangle & \rightarrow & \langle \lambda, v_1\rangle \\ \langle v_0, v_1 v_2\rangle & \rightarrow & \langle \text{star}(\text{letter}), v_1\rangle\langle \text{letter}, v_2\rangle \end{array} \right\}$$

Technically, $\beta_G(\text{star}(\text{letter}))$ contains bound rules with all possible bindings for v_1 and v_2; but the only such bound rules that can be useful in deriving an answer are those of the forms

$$\begin{array}{lrcl} & \langle \text{star}(\text{letter}), \lambda\rangle & \rightarrow & \langle \lambda, \lambda\rangle \\ \forall w \in Z^*, x \in Z, & \langle \text{star}(\text{letter}), wx\rangle & \rightarrow & \langle \text{star}(\text{letter}), w\rangle\langle \text{letter}, x\rangle \end{array}$$

Axiom 3.11a converts these bound rules to derivation steps. Axiom 3.11b allows the steps to be chained together so that, by induction, for any string of symbols $w = x_1 \cdots x_n \in Z^*$,

$$\langle \text{star}(\text{letter}), w\rangle \quad \underset{G}{\overset{*}{\Rightarrow}} \quad \langle \lambda, \lambda\rangle\langle \text{letter}, x_1\rangle \cdots \langle \text{letter}, x_n\rangle$$

Combining this with the earlier results for *letter*, $\langle \text{star}(\text{letter}), w\rangle \underset{G}{\overset{*}{\Rightarrow}} w$. The language accepted by *star*(*letter*) is $L_G(\text{star}(\text{letter})) = Z^*$.

Knowing this, the only useful bound rules in $\beta_G(s)$ are

$$\forall w \in Z^*, \qquad \langle s, w\rangle \quad \rightarrow \quad \langle \text{star}(\text{letter}), w\rangle\langle w, w\rangle\langle w, w\rangle$$

Combining the results on star(letter) with Equation 2 and Axioms 3.11a and 3.11b, $\langle s, w\rangle \underset{G}{\overset{*}{\Rightarrow}} www$ for all $w \in Z^*$. The language accepted by s is $L_G(s) = L(G) = \{www \mid w \in Z^*\}$.

Since $\langle s, w\rangle \underset{G}{\overset{*}{\Rightarrow}} www$, by Axiom 3.11c, $www \underset{G}{\overset{*}{\Rightarrow}} \overline{\langle s, w\rangle}$. By Axiom 3.11b, $s\!:\!www \underset{G}{\overset{*}{\Rightarrow}} s\!:\!\overline{\langle s, w\rangle}$. By Axiom 3.11d, $s\!:\!\overline{\langle s, w\rangle} \underset{G}{\Rightarrow} w$. Therefore, $s\!:\!www \underset{G}{\overset{*}{\Rightarrow}} w$. s computes the function $f(www) = w$ on $L_G(s)$; hence G computes f.

5 Basic results

In most grammar models, there is no way for a terminal string to occur as the left side of a derivation step; therefore, once a terminal string has been derived, computation necessarily halts. The corresponding result for the RAG model *would* be that an answer cannot occur as the left side of a derivation step; but no such result holds because, by Axiom 3.11c, $c \underset{G}{\overset{*}{\Rightarrow}} a$ for $a \in A_G$ implies $a \underset{G}{\overset{*}{\Rightarrow}} \overline{c}$. Thus, in general, derivation can continue indefinitely after an answer has already been derived. However, the following theorem shows that once an answer has been derived, it is pointless to carry the derivation further because it cannot possibly converge to an answer a second time.

Theorem 5.1 If G is a RAG, $a \in A_G$, and $a \overset{+}{\underset{G}{\Rightarrow}} c$, then $c \notin A_G$.

Proof. Suppose G is a RAG. We will construct two sets of configurations, L and R (short for *left* and *right*), such that $L \cup R = C_G - A_G$; if $c \underset{G}{\Rightarrow} c'$ and $c' \in A_G \cup L$ then $c \in L$; and if $c \underset{G}{\Rightarrow} c'$ and $c \in A_G \cup R$ then $c' \in R$. Given such sets L and R, if $a \in A_G$ and $a \overset{+}{\underset{G}{\Rightarrow}} c$ then $c \in R$, and therefore $c \notin A_G$.

Let Π be the set of all polynomials over C_G of the form $\pi(\pi_1, \ldots, \pi_n)$ such that π is a polynomial of arity $n \geq 1$ over A_G, one of the π_k is just a variable, and the rest of the π_k are terms over C_G. (That is, Π is the set of all polynomials of arity 1 over C_G such that no instance of the variable occurs within an argument of the query, pairing, or inverse operator.) Let K_k for $k \geq 0$ be the following hierarchy of sets.

$$
\begin{aligned}
& K_0 = \{\pi(\sigma(c_1, c_2)) \mid \pi \in \Pi,\ \sigma \in \{\mathrm{qry}, \mathrm{pair}\},\ \text{and } c_1, c_2 \in C_G\} \\
\forall n \geq 1, \quad & K_n = \{\pi(\overline{c}) \mid \pi \in \Pi \text{ and } c \in K_{n-1}\} \\
& L = \bigcup_{n \geq 0} K_{2n} \\
& R = \bigcup_{n \geq 0} K_{2n+1}
\end{aligned}
$$

By construction, any configuration that involves the query or pairing operator is in some K_k. Because $a = \overline{a}$ for all $a \in A_G$, any configuration that doesn't involve the query or pairing operator is in A_G. And by construction the K_k are disjoint from A_G; therefore, $L \cup R = C_G - A_G$.

Suppose $x \underset{G}{\Rightarrow} y$. It will be shown that if $y \in A_G \cup L$ then $x \in L$, and if $x \in A_G \cup R$ then $y \in R$. Because $\underset{G}{\Rightarrow}$ is the minimal relation satisfying its axioms, $x \underset{G}{\Rightarrow} y$ must follow from one of Axioms 3.11a, 3.11d by a finite number of deductive steps via Axioms 3.11b, 3.11c. Proceed by induction on the number of deductive steps.

Base case. If $x \underset{G}{\Rightarrow} y$ is implied by Axiom 3.11a, then $x \in L - R$ and $y \in A_G \cup L - R$. If $x \underset{G}{\Rightarrow} y$ is implied by Axiom 3.11d, then $x \in L - R$ and $y \in A_G$.

Inductive step. Suppose the propositions hold for $c \underset{G}{\Rightarrow} c'$, and $x \underset{G}{\Rightarrow} y$ follows from $c \underset{G}{\Rightarrow} c'$ by a single deductive step using Axiom 3.11b or 3.11c. Let $x = \sigma(x_1, \ldots, x_n)$ and $y = \sigma(y_1, \ldots, y_n)$, where σ is an operator of arity n on C_G.

Suppose σ is the inverse operator. Then $x = \overline{c'}$ and $y = \overline{c}$. If $y \in A_G \cup L$, then $c \in A_G \cup R$, by inductive hypothesis $c' \in R$, and $x = \overline{c'} \in L$. Similarly, if $x \in A_G \cup R$ then $y \in R$.

Suppose σ is the query or pairing operator. Then x and y are both in $L - R$.

Suppose σ isn't the query, pairing, or inverse operator. Then for some $1 \leq k \leq n$, $x_k = c \underset{G}{\Rightarrow} c' = y_k$, and for all $j \neq k$, $x_j = y_j$. If $x \in A_G$, then $x_k \in A_G$, by inductive hypothesis $y_k \in R$, and $y \in R$. Similarly, if $y \in A_G$ then $x \in L$. On the other hand, suppose $x \in R$; then one of the x_i must be in R. If $i \neq k$, then $y_i = x_i \in R$; if $i = k$, then $y_i \in R$ by inductive hypothesis. Either way, $y_i \in R$, so $y \in R$. Similarly, if $y \in L$ then $x \in L$. $\square$

Theorem 5.2 If G is a RAG and $x, y, z \in A_G$, then

$$x : y \overset{*}{\underset{G}{\Rightarrow}} z \quad \text{iff} \quad \langle x, z\rangle \overset{*}{\underset{G}{\Rightarrow}} y$$

Proof. Suppose $\langle x, z\rangle \overset{*}{\underset{G}{\Rightarrow}} y$. By Axiom 3.11c, $y \overset{*}{\underset{G}{\Rightarrow}} \overline{\langle x, z\rangle}$. By Axiom 3.11b, $x : y \overset{*}{\underset{G}{\Rightarrow}} x : \overline{\langle x, z\rangle}$. By Axiom 3.11d, $x : \overline{\langle x, z\rangle} \underset{G}{\Rightarrow} z$. Therefore, $x : y \overset{*}{\underset{G}{\Rightarrow}} z$.

On the other hand, suppose $x : y \overset{*}{\underset{G}{\Rightarrow}} z$. Of the four derivation axioms, only 3.11b and 3.11d can imply a derivation step with a left-hand side of the form $c : d$. If such a step is implied by Axiom 3.11b, its right-hand side will have the form $c' : d'$, where $c \overset{*}{\underset{G}{\Rightarrow}} c'$ and $d \overset{*}{\underset{G}{\Rightarrow}} d'$. If the step is implied by Axiom 3.11d, it will have the form $x' : \overline{\langle x', z'\rangle} \underset{G}{\Rightarrow} z'$, where $x', z' \in A_G$. Therefore, $x : y \overset{*}{\underset{G}{\Rightarrow}} z$ implies

$$x : y \quad \overset{*}{\underset{G}{\Rightarrow}} \quad x' : \overline{\langle x', z'\rangle} \quad \underset{G}{\Rightarrow} \quad z' \quad \overset{*}{\underset{G}{\Rightarrow}} \quad z$$

where $x \overset{*}{\underset{G}{\Rightarrow}} x'$, $y \overset{*}{\underset{G}{\Rightarrow}} \overline{\langle x', z'\rangle}$, and $x', z' \in A_G$. By Axiom 3.11c, $\langle x', z'\rangle \overset{*}{\underset{G}{\Rightarrow}} y$. By Theorem 5.1, $x' = x$ and $z' = z$. $\square$

6 Answer-encapsulation

A technical similarity has already been noted, in §3.1 and again in §3.4, between the binding constraints on the metasyntactic variable of an unbound rule (Equation 1) and the meaning of `self` in OOP. By implication, RAG answers are compared to OOP objects. This high-level analogy between answers and objects forms the basis for an important RAG well-behavedness criterion called *answer-encapsulation.*

The idea behind answer-encapsulation is that when two answers have the same "interface", they cannot be distinguished in any way. The notion of "interface" is formalized by the following definition of *semantic equivalence.* In essence, answers x, x' are semantically equivalent if (1) they accept the same syntax $L_G(x) = L_G(x')$, and (2) given any syntax y, they synthesize semantically equivalent answers z, z'.

Definition 6.1 Suppose G is a RAG. A *semantic equivalence* for G is an equivalence relation $\equiv$ on A_G such that, for all $x, x', y, z \in A_G$, if $x \equiv x'$ and $x : y \overset{*}{\underset{G}{\Rightarrow}} z$ then $\exists z' \equiv z$ such that $x' : y \overset{*}{\underset{G}{\Rightarrow}} z'$. The union of all semantic equivalences for G is a relation denoted $\equiv_G$. Answers a, b are *semantically equivalent* iff $a \equiv_G b$.

Theorem 6.2 Suppose G is a RAG. Then $\equiv_G$ is a semantic equivalence for G.

Proof. By construction, $\equiv_G$ is reflexive and symmetric, and $\forall x, x', y, z \in A_G$, if $x \equiv_G x'$ and $x : y \overset{*}{\underset{G}{\Rightarrow}} z$ then $\exists z' \equiv_G z$ such that $x' : y \overset{*}{\underset{G}{\Rightarrow}} z'$. It therefore suffices to show that $\equiv_G$ is transitive.

Let $\equiv$ be the transitive closure of $\equiv_G$. $\equiv_G \subseteq \equiv$. By construction, $\equiv$ is an equivalence relation. Suppose $x, x', y, z \in A_G$, $x \equiv x'$, and $x : y \overset{*}{\underset{G}{\Rightarrow}} z$. Then there exists a finite sequence of answers $x_0, \ldots, x_n$ such that $x = x_0$, $x' = x_n$, and for $1 \leq k \leq n$, $x_{k-1} \equiv_G x_k$. Therefore, there exist answers $z_0, \ldots, z_n$ such that $z = z_0$, and for $1 \leq k \leq n$, $z_{k-1} \equiv_G z_k$ and $x_k : y \overset{*}{\underset{G}{\Rightarrow}} z_k$. In particular, $z_n \equiv z$ and $x' : y \overset{*}{\underset{G}{\Rightarrow}} z_n$, therefore $\equiv$ is a semantic equivalence, and by construction of $\equiv_G$, $\equiv \subseteq \equiv_G$. □

Corollary 6.3 Suppose G is a RAG, and ρ_G obeys Equation 2. Then for all $x, y \in T_G$, $x \equiv_G y$ iff $x = y$.

Proof. Suppose G is a RAG, ρ_G obeys Equation 2, and $x, y \in T_G$. If $x = y$, then $x \equiv_G y$ because $\equiv_G$ is reflexive. If $x \neq y$, then $L_G(x) \neq L_G(y)$, so $x \not\equiv_G y$. □

Definition 6.4 Suppose G is a RAG. G is *weakly answer-encapsulated* iff $\equiv_G$ is a congruence on A_G.

That is, if σ has arity n on A_G, and $a_k \equiv_G b_k$ for all $1 \leq k \leq n$, then $\sigma(a_1, \ldots, a_n) \equiv_G \sigma(b_1, \ldots, b_n)$. Weak answer-encapsulation superficially delivers the intended property, that semantically equivalent answers "cannot be distinguished" (by substituting one for another as arguments to σ).

However, weak answer-encapsulation is a holistic property of ρ_G, a net consequence of how all the rule equations of G interact with each other. When constructing RAGs in practice, it is convenient to be able to design the rule equations separately, yet guarantee that when combined they will produce an answer-encapsulated RAG. The following definitions provide a formal basis for such a guarantee.

Definition 6.5 Suppose G is a RAG, σ an operator of arity n on A_G. σ is *G-safe* iff, given any answers $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ with $a_i \equiv_G b_i$ for all $1 \leq i \leq n$, $\sigma(a_1, \ldots, a_n) \equiv_G \sigma(b_1, \ldots, b_n)$.

Thus, G is weakly answer-encapsulated iff all the operators on A_G are G-safe.

Definition 6.6 A RAG *framework* is a tuple $\mathcal{F} = \langle \Sigma, \mathcal{E} \rangle$, where Σ is a signature and $\mathcal{E}$ is a set of rule equations over Σ. A RAG G *satisfies* $\mathcal{F}$ iff $\Sigma \subseteq \Sigma_{A_G}$ and G satisfies all $e \in \mathcal{E}$. The class of all RAGs satisfying $\mathcal{F}$ is denoted $\mathcal{G}_{\mathcal{F}}$. $\mathcal{F}$ is *satisfiable* iff $\mathcal{G}_{\mathcal{F}}$ is nonempty. $\mathcal{F}$ is *safe* iff, for every operator σ specified by any $e \in \mathcal{E}$, and for every $G \in \mathcal{G}_{\mathcal{F}}$, σ is G-safe.

A RAG G is *strongly answer-encapsulated* iff there exists a safe framework $\mathcal{F} = \langle \Sigma, \mathcal{E} \rangle$ such that $\mathcal{E}$ specifies all the operators on A_G, and G satisfies $\mathcal{F}$.

Lemma 6.7 Suppose G is a RAG. If G is strongly answer-encapsulated, then G is weakly answer-encapsulated.

Suppose $\mathcal{F}_1 = \langle \Sigma_1, \mathcal{E}_1 \rangle$ and $\mathcal{F}_2 = \langle \Sigma_2, \mathcal{E}_2 \rangle$ are safe frameworks, and let $\mathcal{F}_1 \cup \mathcal{F}_2 = \langle \Sigma_1 \cup \Sigma_2, \mathcal{E}_1 \cup \mathcal{E}_2 \rangle$. Then $\mathcal{F}_1 \cup \mathcal{F}_2$ is also a safe framework.

Suppose $\mathcal{F}$ is a framework, and all the operators specified by $\mathcal{F}$ have arity 0. Then $\mathcal{F}$ is safe.

Proof. The first two results follow trivially from Definition 6.6. By Definition 6.5, if G is a RAG and σ a constant operator on A_G, then σ is G-safe; it follows that a framework that only specifies constant operators must be safe. $\square$

The remainder of this section is concerned with proving the safeness — or in one case, unsafeness — of particular frameworks. Beyond the simple results of Lemma 6.7, no general framework safeness theorems will be offered here; there is one such general theorem in [Shut93, §6.3.1], but it employs extensive formal machinery for an elaborate result of rather limited utility.

Lemma 6.8 Suppose G is a RAG that satisfies Equation 3, $x, y, z \in A_G$, $x : y \overset{*}{\underset{G}{\Rightarrow}} z$, and $x = x_1 \cdots x_n$. Then $y = y_1 \cdots y_n$ and $z = z_1 \cdots z_n$ such that $x_k : y_k \overset{*}{\underset{G}{\Rightarrow}} z_k$ for $1 \leq k \leq n$.

Proof. Suppose G etc. as in the lemma. By Theorem 5.2, $\langle x, z \rangle \overset{*}{\underset{G}{\Rightarrow}} y$. By the derivation axioms (Definition 3.11), there exists an unbound rule $r \in \rho_G(x)$ and a bound rule $(\langle x, q \rangle \to c) \in \beta_G(r)$ such that $c \overset{*}{\underset{G}{\Rightarrow}} y$ and $q \overset{*}{\underset{G}{\Rightarrow}} z$, so that

$$\langle x, z \rangle \quad \overset{*}{\underset{G}{\Rightarrow}} \quad \langle x, \overline{q} \rangle \quad \underset{G}{\Rightarrow} \quad c \quad \overset{*}{\underset{G}{\Rightarrow}} \quad y$$

By Equation 3, there must exist unbound rules $r_k \in \rho_G(x_k)$, and bound rules $(\langle x_k, q_k \rangle \to c_k) \in \beta_G(r_k)$, such that $q = q_1 \cdots q_n$ and $c = c_1 \cdots c_n$. Again by the derivation axioms, since $c \overset{*}{\underset{G}{\Rightarrow}} y$ there must exist y_k such that $y = y_1 \cdots y_n$ and $c_k \overset{*}{\underset{G}{\Rightarrow}} y_k$; and since $q \overset{*}{\underset{G}{\Rightarrow}} z$, z_k such that $z = z_1 \cdots z_n$ and $q_k \overset{*}{\underset{G}{\Rightarrow}} y_k$. Therefore, by Theorem 5.2, $x_k : y_k \overset{*}{\underset{G}{\Rightarrow}} z_k$. $\square$

Theorem 6.9 If $\mathcal{F}$ is a framework containing Equation 3 and no other, then $\mathcal{F}$ is safe.

Proof. Suppose G is a RAG that satisfies Equation 3. Let $\equiv$ be the unique binary relation over A_G such that $a \equiv b$ iff there exist a_1, a_2, b_1, b_2 such that $a = a_1 a_2$, $b = b_1 b_2$, $a_1 \equiv_G b_1$, and $a_2 \equiv_G b_2$. It suffices to show that $\equiv$ is a semantic equivalence, hence concatenation is G-safe. Suppose $x, x', y, z \in A_G$, $x : y \overset{*}{\underset{G}{\Rightarrow}} z$, and $x \equiv x'$. By construction, $x = x_1 x_2$ and $x' = x'_1 x'_2$ such that $x_1 \equiv_G x'_1$ and $x_2 \equiv_G x'_2$. By Lemma 6.8, $y = y_1 y_2$ and $z = z_1 z_2$ such that $x_1 : y_1 \overset{*}{\underset{G}{\Rightarrow}} z_1$ and $x_2 : y_2 \overset{*}{\underset{G}{\Rightarrow}} z_2$. By Theorem 6.2, $\exists z'_1, z'_2 \in A_G$ such that $z_1 \equiv_G z'_1$, $z_2 \equiv_G z'_2$, $x'_1 : y_1 \overset{*}{\underset{G}{\Rightarrow}} z'_1$, and $x'_2 : y_2 \overset{*}{\underset{G}{\Rightarrow}} z'_2$. Since G obeys Equation 3, $b_1 b_2 : y_1 y_2 \overset{*}{\underset{G}{\Rightarrow}} z'_1 z'_2$. By construction, $z'_1 z'_2 \equiv z_1 z_2 = z$. $\square$

Here are some other commonly used rule equations.

$$\rho(a \sqcup b) \quad = \quad \left\{ \begin{array}{lcl} \langle v_0, v_1 \rangle & \to & \langle a, v_1 \rangle \\ \langle v_0, v_1 \rangle & \to & \langle b, v_1 \rangle \end{array} \right\} \tag{4}$$

$$\rho([a,b]) \;=\; \{\ \langle v_0, b\rangle \;\to\; \langle a, v_1\rangle\ \} \tag{5}$$

$$\rho(\mathrm{star}(a)) \;=\; \left\{ \begin{array}{rcl} \langle v_0, \lambda\rangle & \to & \lambda \\ \langle v_0, v_1 v_2\rangle & \to & \langle a, v_1\rangle\langle \mathrm{star}(a), v_2\rangle \end{array} \right\} \tag{6}$$

The binary operator of Equation 4, when supported by that rule equation, is conventionally called the *union* operator; the binary operator of Equation 5, the *mapping* operator. ($[a,b]$ *maps* answers recognized by a into semantic value b.) By convention, RAGs will be assumed to satisfy Equations 4–6 unless otherwise stated.

Theorem 6.10 A framework containing Equation 4 and no other is safe. A framework containing Equation 5 and no other is safe. A framework containing Equations 6 and 3 and no others is safe.

These results can be proved by the same straightforward technique that was used for Theorem 6.9, extending $\equiv_G$ over the specified operator to an equivalence $\equiv$ and showing that $\equiv$ is a semantic equivalence. The only complication is that the *star* operator must be shown G-safe by induction on derivation length, using Equation 3 in the inductive step.

As a contrast to the above examples, consider the following rule equation. (This equation was an early candidate for the mapping operator, and the concept of answer-encapsulation was motivated partly from studying it.)

$$\rho([a,b]) \;=\; \{\ \langle v_1, b\rangle \;\to\; a\ \} \tag{7}$$

Theorem 6.11 If $\mathcal{F}$ is a satisfiable framework that includes Equation 7, then $\mathcal{F}$ is not safe.

Proof. Suppose $\mathcal{F}$ is satisfiable and includes Equation 7. Suppose $G \in \mathcal{G}_{\mathcal{F}}$ has an operator σ of arity zero in A_G with $L_G(\sigma) = \{\sigma\}$ and $\sigma : \sigma \overset{*}{\underset{G}{\Rightarrow}} \sigma$. (One can always construct such a G.) By Equation 7, $L_G([\sigma,\sigma]) = \{\sigma\}$ and $[\sigma,\sigma] : \sigma \overset{*}{\underset{G}{\Rightarrow}} \sigma$, therefore $[\sigma,\sigma] \equiv_G \sigma$. However, $L_G([[\sigma,\sigma],\sigma]) = \{[\sigma,\sigma]\} \neq L_G([\sigma,\sigma])$, so the mapping operator is not G-safe. □

7 Normal RAGs

This section defines the class of *normal RAGs*, which are simply RAGs that have several convenient well-behavedness conditions, and shows that this restricted class of RAGs is Turing-powerful.

One of the conditions for normality is *non-circularity.* This is essentially the same concept as in the theory of attribute grammars; but whereas determining AG non-circularity is a computationally complex problem involving interactions between semantic rules of different syntax productions [Knut90], RAG non-circularity can be determined separately for each unbound rule.

Definition 7.1 Suppose r is an unbound rule

$$\langle v_0, e_0\rangle \rightarrow t_0\langle e_1, v_1\rangle t_1 \cdots \langle e_n, v_n\rangle t_n$$

Then r is *non-circular* iff there exists a permutation $k_1, \ldots, k_n$ of the integers from 1 to n such that for all $1 \leq i \leq n$, e_{k_i} uses only variables v_0 and $\{v_{k_j} \mid j < i\}$. Further, r is *left-to-right* iff the identity permutation $k_i = i$ satisfies this condition. A RAG G is non-circular iff for all $a \in A_G$ and all $r \in \rho_G(a)$, r is non-circular; and left-to-right iff all such r are left-to-right.

Recall that, by design, the RAG derivation step relation was meant to be "generated in a computationally trivial way" (§1, Property 3). There are three distinct computational activities involved in generating the RAG step relation, i.e., determining all possible right-hand sides for a given left-hand side or vice versa: (1) deciding equality of terms over C_G; (2) computing ρ_G of a given term over A_G; and (3) unifying a term over C_G with one side of a step in one of Axioms 3.11a–3.11d (the main top-level activity, which may invoke both of the preceding two, and may recursively invoke itself on a subterm).

Because of the way the hierarchy of algebras over a RAG was constructed in Definitions 3.1–3.5, deciding equality of terms is straightforward. The following definition of *normal RAG* restricts the rule function in a way that keeps it 'elementary' as well.

Definition 7.2 A RAG G is *normal* iff G satisfies Equations 2 and 3; G is strongly answer-encapsulated and non-circular; and all of the nonterminal operators of G are specified by rule equations of the form

$$\rho(\sigma(x_1, \ldots, x_n)) = \left\{ \begin{array}{ccc} \pi_1 & \rightarrow & \pi'_1 \\ & \vdots & \\ \pi_m & \rightarrow & \pi'_m \end{array} \right\}$$

where the π_k and π'_k are polynomials over C_G.

Note that rule equations of this form, besides assuring that the rule function will be easy to compute, also make non-circularity (and even more so, left-to-right-ness) easy to check.

Theorem 7.3 Suppose G is an unrestricted Chomsky grammar. Then there exists a normal RAG H such that $L(H) = L(G)$.

Proof. Suppose $G = \langle Z, T, R, s\rangle$ is an unrestricted Chomsky grammar; here, Z is the finite alphabet of symbols, $T \subset Z$ the terminal alphabet, $R \subset Z \times Z$ the finite set of production rules, and $s \in Z - T$ the start symbol. Let H be a RAG with terminal alphabet Z, start symbol *sentence*, and the following rule equations.

$$\begin{array}{rcl}
\text{string} & = & \text{star}\left(\bigsqcup_{z\in Z} z\right) \\
\text{terminal} & = & \text{star}\left(\bigsqcup_{t\in T} t\right) \\
\text{rule} & = & \bigsqcup_{(\beta\to\alpha)\in R} [\alpha, \beta] \\
\rho(\text{step}) & = & \left\{ \ \langle v_0, v_1v_2v_3\rangle \ \to \ \langle\text{string}, v_1\rangle\langle\text{rule}, v_2\rangle\langle\text{string}, v_3\rangle \ \right\} \\
\rho(\text{derive}) & = & \left\{ \begin{array}{rcl} \langle v_0, \lambda\rangle & \to & s \\ \langle v_0, \text{derive}\!: v_1\rangle & \to & \langle\text{step}, v_1\rangle \end{array} \right\} \\
\rho(\text{sentence}) & = & \left\{ \ \langle v_0, \text{derive}\!: v_1\rangle \ \to \ \langle\text{terminal}, v_1\rangle \ \right\}
\end{array}$$

H is left-to-right. The above rule equations are for constants, hence safe by Lemma 6.7; and all other rule equations of H are conventional; hence H is strongly answer-encapsulated. So H is normal.

Nonterminal *string* recognizes any string $w \in Z^*$, and synthesizes semantic value w; i.e., $\text{string}\!: w \overset{*}{\underset{H}{\Rightarrow}} w$. Nonterminal *terminal* does the same except that it recognizes only terminal strings $w \in T^* \subset Z^*$. Nonterminal *rule* recognizes the right side of any production rule of G, and synthesizes the left side of that production rule; thus, if $(\alpha \to \beta) \in R$, then $\langle\text{rule}, \alpha\rangle \overset{*}{\underset{H}{\Rightarrow}} \beta$, or equivalently, $\text{rule}\!: \beta \overset{*}{\underset{H}{\Rightarrow}} \alpha$. Nonterminal *step* recognizes the right side of any derivation step of G, and synthesizes the left side of that step; thus, if $x \underset{G}{\Rightarrow} y$, then $\langle\text{step}, x\rangle \overset{*}{\underset{H}{\Rightarrow}} y$, or equivalently, $\text{step}\!: y \overset{*}{\underset{H}{\Rightarrow}} x$.

Nonterminal *derive* recognizes s, the start symbol of G; and also recognizes the right side of any derivation step of G whose *left* side is recognized by *derive*. So, by induction, *derive* recognizes exactly the sentential forms of G — that is, $L_H(\text{derive}) = \{w \in Z^* \mid s \overset{*}{\underset{G}{\Rightarrow}} w\}$.

Therefore, nonterminal *sentence* recognizes just those terminal strings that are also sentential forms of G; that is, $L_G(\text{sentence}) = L(G)$. □

Theorem 7.4 Suppose M is a deterministic Turing machine with input alphabet T and tape alphabet Z, and $f_M : T^* \to Z^*$ is the partial function computed by M. Then f_M is computed by some normal RAG G.

Proof. Suppose M etc. as in the theorem. Let $\star$ be a symbol not in Z, and let H be an unrestricted Chomsky grammar that accepts the language $L(H) = \{x \star y \mid f_M(x) = y\}$. Following the proof of Theorem 7.3, construct RAG G with nonterminal *sentence* that recognizes exactly $L(H)$; and add a new start symbol *start*, where

$$\begin{array}{rcl} \text{out} & = & \text{star}\left(\bigsqcup_{z\in Z}[\lambda, z]\right) \\ \rho(\text{start}) & = & \{\ \langle v_0, v_2\rangle \ \to \ \langle \text{terminal}, v_1\rangle\langle \text{out}, v_2\rangle\langle \text{sentence} :(v_1 \star v_2), v_3\rangle\ \} \end{array}$$

Nonterminal *terminal* is part of the construction from the earlier proof; it recognizes any string over the terminal alphabet of H. Nonterminal *out* always recognizes λ, and synthesizes ("at random") any string over alphabet Z. Nonterminal *sentence*, also from the earlier proof, recognizes any string in $L(H)$, and synthesizes λ.

Of the three pairs on the right side of the unbound rule for *start*, the first recognizes any string over $Z \cup \{\star\}$; the second recognizes λ; and the third recognizes λ — if it recognizes anything. However, the left component of the third pair cannot be reduced to an answer unless the values x, y synthesized by the first two pairs satisfy $f_M(x) = y$. The value synthesized by *start* is then y. Hence, $\text{start} : x \overset{*}{\underset{G}{\Rightarrow}} f_M(x)$. $\Box$

8 Conclusions

The stated design goal of the RAG model was to combine Turing power with 'technical simplicity' and 'conceptual clarity' in a single grammar formalism.

Conceptual clarity was approached (and, one hopes, achieved) by paralleling the broad structure of CFGs. RAGs have four components analogous to those of a CFG; the finite CFG nonterminal alphabet is generalized to an arbitrary nonterminal signature, and the finite CFG rule set is replaced by a rule function. Also, RAG rules have the one-to-many structure of CFG rules, supporting use of derivation trees.

At a more technical level, RAGs resemble EAGs. RAG unbound rules are analogous to EAG rule forms, with compound structures (RAG pairs, EAG attributed nonterminal forms) where a CFG rule would have nonterminal symbols. The analogy extends fairly well to the binding of variables to produce bound rules, and moderately well to the use of bound rules in inducing the derivation step. The elementarity of the derivation step is reconciled with Turing power by introducing the *query operator* for all nontrivial computation within a pair, and distributing query evaluation over an unbounded number of derivation steps through the purely technical device of the *inverse operator*. (The inverse operator is purely technical in the sense that, for purposes of language definition or parsing, the meaning of a RAG can be understood entirely without it, provided Theorems 5.1 and 5.2 are accepted without proof.)

What emerges, rising above these technical devices back to the purely conceptual level, is a characteristically RAG view of derivation as a *ternary* relation between metasyntax, syntax, and semantics. At the heart of this notion is the equivalence in Theorem 5.2, which (as just noted) is readily understood without resort to the technical details that support it. This equivalence gives rise, in turn, to the conceptually "recursive" nature of the model.

In a RAG derivation tree, each parent node is labeled with its metasyntax and semantics, while its syntax is the fringe of its branch. At both conceptual and, to some extent, technical levels, metasyntax is an *inherited* value in the tree: The model fixes its value at the root node, and Equation 1 uses the metasyntax at a parent node to fix the set of rules that can be used to expand *downward* from that node.[4] Semantics is then — especially in a non-circular RAG — conceptually synthesized.

Behind the stated goals of the RAG design, there is a broader agenda. Programming language design is replete with traditionally desirable language properties that are notoriously subjective, such as 'simplicity', or 'abstraction'. M. Felleisen took a first step toward formalizing such notions in [Fell90], which proposed a definition for *expressiveness*. That work was fairly successful in capturing the essence of the concept of "syntactic sugar", but its definitions do not seem able to encompass the ability to express abstractions [Shut99]. The RAG design was intended to provide an ideal tool for formally analyzing otherwise subjective language properties: Conceptual clarity provides an extensive interface with subjective concepts, maximizing opportunities for the grammar model to be applied to the problem; technical simplicity makes formal analysis easier; and Turing power maximizes the generality of the analysis.

Acknowledgments

The author would like to thank Roy Rubinstein and Mike Gennert for their thoughtful comments on the content and presentation of this paper.

References

[Cara63] Alfonso Caracciolo di Forino, "Some Remarks on the Syntax of Symbolic Programming Languages", *Communications of the ACM* 6 no. 8 (August 1963), pp. 456–460.

[Chri90] Henning Christiansen, "A Survey of Adaptable Grammars", *SIGPLAN Notices* 25 no. 11 (November 1990), pp. 35–44.

[Fell90] Matthias Felleisen, "On the Expressive Power of Programming Languages", in Neil D. Jones, editor, *ESOP '90: 3rd European Symposium on Programming* [Copenhagen, Denmark, May 15–18, 1990, *Proceedings*] [*Lecture Notes in Computer Science* 432], New York: Springer-Verlag, 1990, pp. 134–151.

[4] One could imagine variants on the formalism in which an altered Equation 1 might cause metasyntax to be synthesized.

[Knut90] Donald Ervin Knuth, "The Genesis of Attribute Grammars", in Pierre Deransart and Martin Jourdan, editors, *Attribute Grammars and their Applications* [*International Conference WAGA*] [*Lecture Notes in Computer Science* 461], New York: Springer-Verlag, 1990, pp. 1–12.

[Mads80] Ole Lehrmann Madsen, "On Defining Semantics by Means of Extended Attribute Grammars", in Neil D. Jones, editor, *Semantics-Directed Compiler Generation* [*Proceedings of a Workshop*, Aarhus, Denmark, January, 1980] [*Lecture Notes in Computer Science* 94], New York: Springer-Verlag, 1980, pp. 259–299.

[Shut93] John N. Shutt, "Recursive Adaptable Grammars", Master's Thesis, Worcester Polytechnic Institute, Worcester, Massachusetts, 1993; Emended, June 4, 1998.

[Shut99] John N. Shutt, "S-Expressiveness and the Abstractive Power of Programming Languages", technical report, Worcester Polytechnic Institute, Worcester, Massachusetts, 1999. To appear.

[Wijn65] Aad van Wijngaarden, "Orthogonal design and description of a formal language", Technical Report MR 76, Mathematisch Centrum, Amsterdam, 1965.