PROGRAM EQUIVALENCE AND CONTEXT-FREE GRAMMARS

Barry K. Rosen

IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

ABSTRACT: This note defines a new equivalence relation among systems of recursion equations and a method for assigning context-free grammars to these systems, such that

(1) The new equivalence implies strong equivalence;

(2) Systems are equivalent in the new sense iff their grammars generate the same language;

(3) There is a nontrivial decidable class of systems whose grammars have the decidability properties of LL(k) grammars.

Thus, the decidability of generative equivalence for LL(k) grammars mitigates the general undecidability of strong equivalence between arbitrary systems of recursion equations.

1. Recursion Schemes

The formalism for recursive computation used here is based on [7, §§7-8], which extends and formalizes the well-known recursive calculus of McCarthy [5]. Some departures from [7] are necessary because here we do not use a fixed data space $\underset{\sim}{D}$ and a fixed supply of given partial functions on $\underset{\sim}{D}$.

The operator-operand structures in recursive definitions will be expressed by trees whose nodes are labelled by symbols from an infinite vocabulary V that is the union of a family of disjoint sets: a set F_k of new function symbols for each positive integer k, a set G_k of given function symbols for each nonnegative integer k, a set H of input symbols, and a set W of formal parameters. Each $\alpha \in V$ has a nonnegative integer rank $\rho(\alpha)$ which is k if $\alpha \in F_k \cup G_k$ and 0 otherwise. A ranked tree is a finite tree with nodes labelled in V, such that any node labelled α has exactly $\rho(\alpha)$ sons. The set of all ranked trees is $V_{\#}$. Trees are denoted by "terms" with round brackets and commas as in [6], [7], [10]. We sometimes use overlines to emphasize the distinction between trees and values of functions:

$$\overline{\mathrm{sum}}(\overline{5},\overline{5}) \neq \overline{\mathrm{sum}}(\overline{6},\overline{4}) \quad \text{but} \quad \mathrm{sum}(5,5) = \mathrm{sum}(6,4) = 10.$$

In order to assign values to trees we first interpret the given function symbols and the input symbols as partial functions on a domain $\underset{\sim}{D}$. We reserve a given function symbol C of rank 3 for the $\underset{\sim\sim}{\text{if}}$ ... $\underset{\sim\sim\sim\sim}{\text{then}}$ ... $\underset{\sim\sim\sim\sim}{\text{else}}$ construction. Since conditional expressions may have values despite "undefined" subexpressions, we introduce a new "infinite" element $\infty \notin \underset{\sim}{D}$ as a possible value. Members of $\underset{\sim}{D}$ are "finite" values. An ordinary given function symbol $g \neq C$ may now be interpreted as a total function mapping $\rho(g)$-tuples of members of $\underset{\sim}{D} \cup \{\infty\}$ into $\underset{\sim}{D} \cup \{\infty\}$.

(1.1) Definition. Let $\underset{\sim}{D}$ be any set disjoint from V with at least two members. Let $\infty \notin \underset{\sim}{D}$ and let $\mathrm{yes}, \mathrm{no} \in \underset{\sim}{D}$ with $\mathrm{yes} \neq \mathrm{no}$. Let $C \in G_3$. A base interpretation of V on $\underset{\sim}{D}$ with infinity ∞, truth

yes, falsity no, and conditional operator C is any map I assigning to each $\alpha \in G \cup H$ a total function

$$I_\alpha: \ (\underset{\sim}{D} \cup \{\infty\})^{\rho(\alpha)} \to (\underset{\sim}{D} \cup \{\infty\})$$

such that

(1) $(\forall \theta, \xi, \eta \in \underset{\sim}{D} \cup \{\infty\})[I_C(yes,\xi,\eta) = \xi \ \& \ I_C(no,\xi,\eta) = \eta \ \& \ (\theta \notin \{yes,no\} \text{ implies } I_C(\theta,\xi,\eta) = \infty)]$

(2) $(\forall g \in G - \{C\})(\forall \theta \in (\underset{\sim}{D} \cup \{\infty\})^{\rho(g)})[(\exists i < \rho(g))(\theta_i = \infty) \text{ implies } I_g\theta = \infty]$.

From now on we will say "let I be a base interpretation of V on $\underset{\sim}{D}$" and use the special symbols ∞, yes, no, and C as above. Given such an I we can execute calls on given functions and fetches of input values. Calls on new function symbols will be executed by macroexpansion.

Suppose $f \in F_k$ and $u_0,\ldots,u_{k-1}$ are distinct parameters. A pair of trees

$$\overline{f}(\overline{u}_0,\ldots,\overline{u}_{-1}) \to S$$

is a rule-schema for f iff S is in $(F \cup G \cup \{u_0,\ldots,u_{-1}\})_\#$ and the label S() at the root of S is from $G \cup W$. Input symbols and parameters other than $u_0,\ldots,u_{-1}$ may not occur in S. A finite set of rule-schemata containing at most one schema for each new function symbol is a recursive definition. (This is a special case of Definition 7.2 in [7].) A recursive definition L is complete iff, whenever some $f \in F$ occurs in S for some $R \to S \in L$, then some $R' \to S' \in L$ is a rule-schema for f.

(1.2) Definition. A recursion scheme is any pair $\mathfrak{R} = (L, f(x_0,\ldots,x_{\rho(f)-1}))$ such that $f \in F$, the x_i for all $i < \rho(f)$ are distinct input symbols, and L is a complete recursive definition containing a rule-schema for f.

Recursive computation proceeds by macroexpansion of function calls. To each parameter u we will assign a domain D_u of trees that may be substituted for it. The instances of a rule schema are the pairs of trees that can be formed by systematic substitution: replace each parameter u by a specific member of D_u. (For a fastidious discussion, see [7, §§5,6].) The set of all instances of rule-schemata in a recursive definition L is $\underset{\sim}{R}_L$.

Now suppose that $\mathfrak{R} = (L, f(x_0,\ldots,x_{-1}))$ is a recursion scheme and I is a base interpretation of V on a set $\underset{\sim}{D}$. We assign rank 0 to members of $\underset{\sim}{D}$, so that trees in $(V \cup \underset{\sim}{D})_\#$ represent partial computations. To complete our definition of the set $\underset{\sim}{R}_L$ of instances of members of L, we stipulate that

(1.3.1) $(\forall u \in W)(D_u = (V \cup \underset{\sim}{D})_\#)$

so that a call on a new function may be macroexpanded before anything is done to the actual parameters. In order to execute calls on given functions or read the values of inputs, we use the set $\underset{\sim}{R}_{giv}[I]$ of all rules

(1.3.2) $\overline{\alpha}(\overline{\xi}_0,\ldots,\overline{\xi}_{-1}) \to \overline{\eta}$.

for $\alpha \in G \cup H$; $\xi_0,\ldots,\xi_{-1},\eta \in \underset{\sim}{D}$; $I_\alpha(\xi_0,\ldots,\xi_{-1}) = \eta$. In order to evaluate conditional expressions we need the additional set $\underset{\sim}{R}_C[I]$ of all rules

(1.3.3) $\overline{C}(\overline{yes},R,S) \to R$

(1.3.4) $\overline{C}(\overline{no},R,S) \to S$

for $R,S \in (V \cup D)_{\#}$.

Writing $R \Rightarrow S$ for "R can become S as a result of applying a rule at a node in R", we note that $\Rightarrow$ has the Church-Rosser property

(1.3.5) $(\forall R,S,S' \in (V \cup D)_{\#})[(R \overset{*}{\Rightarrow} S \ \& \ R \overset{*}{\Rightarrow} S') \text{ implies } (\exists T \in (V \cup D)_{\#})(S \overset{*}{\Rightarrow} T \ \& \ S' \overset{*}{\Rightarrow} T)]$

by [7, Lemma 7.4]. The "Evaluation Theorem" [7, Thm. 7.5] establishes the existence of a unique total function

(1.3.6) $\text{Eval}_L[I]: \ (V \cup D)_{\#} \to D \cup \{\infty\}$

such that

(1.3.7) $(\forall R \in (V \cup D)_{\#})(\forall \xi \in D)(\text{Eval}_L[I]R = \xi \text{ iff } R \overset{*}{\Rightarrow} \overline{\xi})$.

This function is used to define the value of $\mathfrak{R}$ under I. Note that the value is always defined but may be infinite.

(1.3.8) $\text{Val}[I]\mathfrak{R} = \text{Eval}_L[I]\overline{f}(\overline{x}_0,\ldots,\overline{x}_{-1})$

Recursion schemes that deliver the same value under all base interpretations are strongly equivalent. Since recursion schemes include flowchart schemes as a special case, strong equivalence is undecidable [4, Thm. 4.1]. In §2 we will define "tree equivalence" and show that it implies strong equivalence. In §3 we will characterize tree equivalence in terms of generative equivalence of certain context-free grammars. In §4 we will show that tree equivalence is decidable for an important class of recursion schemes by LL(k) techniques. The decidability of tree equivalence in general remains an open problem. The relations between this note and other work on equivalence of programs are discussed in §5.

2. Tree Equivalence

Simply by omitting the word "finite" from the formal definition [7, Def. 4.2], we can define possibly infinite trees. If $\mathfrak{R} = (L,f(x_0,\ldots,x_{-1}))$ is a recursion scheme then we can summarize all possible computations by generating a possibly infinite tree from $f(x_0,\ldots,x_{-1})$ with top-down applications of rules from R_L. For example, suppose that L is the familiar recursive definition of factorials:

$f(u) \to C(EQ(u,0),1,MULT(u,f(SUB(u,1))))$.

The tree R_∞ is shown in Figure 1.

In general we begin with $f(x_0,\ldots,x_{-1})$ and build up a sequence $R = (R_0,R_1,\ldots)$, where $R_j \Rightarrow R_{j+1}$ by application of a rule from R_L at a node M_j in R_j. The top-down idea (with no other sequencing restrictions) is expressed by saying that each node in R_j labelled by a new function symbol must be at least as long as M_j when expressed as a "Dewey decimal" or "branch numbers" string of nonnegative integers.

(2.1) Definition. An expansion of a recursion scheme $\mathfrak{R} = (L,f(x_0,\ldots,x_{-1}))$ is any pair (R,M) where R is a possibly infinite sequence of trees and M is a possibly infinite sequence of nodes such that

(1) $R_0 = f(x_0,\ldots,x_{-1})$

(2) $(\forall j < |M|)(j+1 < |R| \;\&\; M_j \in \mathrm{Dom}\, R_j \;\&\; R_j \Rightarrow R_{j+1}$ by applying a rule in $R_{\sim L}$ at $M_j)$

(3) $(\forall j < |M|)(\forall n \in \mathrm{Dom}\, R_j)(R_j n \in F$ implies $|M_j| \leq |n|)$.

A complete expansion is any expansion (R,M) such that M is infinite or $R_{|M|} \in (G \cup H)_{\#}$.

Any recursion scheme has a complete expansion (R,M). By the condition $S() \in G \cup W$ that we imposed on the rule-schemata $R \to S$ in §1, the sequence $(|M_0|, |M_1|,\ldots)$ is eventually above any particular number when $|M|$ is infinite. (The case where $|M|$ is finite is both rare and trivial.)

Let (R,M) be a complete expansion and let $j < |R|$. Suppose n is a node of R_j with all ancestors (including itself) labelled by given function symbols or input symbols. Then n and its label $R_j n$ persist in $R_{j+1}, R_{j+2},\ldots$ and should be considered part of the infinite tree R_∞, as in Figure 1. Using techniques from [7], it is not difficult to prove the following lemma.

(2.2) Lemma. Let $\mathfrak{A}$ be a recursion scheme and let (R,M) be a complete expansion of $\mathfrak{A}$. For each $j < |R|$ set

(1) $\Gamma_j = \{n \in \mathrm{Dom}\, R_j \mid (\forall m \text{ anc } n)(R_j n \in G \cup H)\}$

and let R_∞ be the set of ordered pairs

(2) $R_\infty = \bigcup_{j < |R|} R_j \restriction \Gamma_j$

where $\restriction$ denotes restriction of functions. Then R_∞ is a possibly infinite tree in $(G \cup H)_{\#}$ that does not depend on the choice of (R,M). □

Now we define an equivalence relation among possibly infinite trees which reduces to equality among finite trees.

(2.3) Definition. A node in a possibly infinite tree is hopeful if it is an ancestor of a leaf and is hopeless otherwise. Possibly infinite trees R and S are equivalent ($R \cong S$) iff they have the same sets of hopeful nodes and $Rn = Sn$ for each hopeful node n.

Assigning possibly infinite trees to recursion schemes by means of Lemma 2.2, we define a new equivalence relation among recursion schemes.

(2.4) Definition. Let $\mathfrak{A}$ and $\mathfrak{S}$ be recursion schemes with possibly infinite trees R_∞ and S_∞. Then $\mathfrak{A}$ and $\mathfrak{S}$ are tree equivalent iff $R_\infty \cong S_\infty$.

If I is a base interpretation of V on $\underset{\sim}{D}$ then there is a function $\mathrm{Eval}_\phi[I]$ (as in (1.3.6)) assigning values in $\underset{\sim}{D} \cup \{\infty\}$ to trees in $(V \cup \underset{\sim}{D})_{\#}$ with no macroexpansion of new function calls. The following lemma shows that two trees will evaluate the same way if they differ only at nodes that are deleted by conditional branching.

(2.5) Lemma. Let I be a base interpretation of V on $\underset{\sim}{D}$ and let $R,S \in (V \cup \underset{\sim}{D})_{\#}$. Let P be

a nonempty subset of Dom R ∩ Dom S such that all fathers of members of P are in P. Let Q be the set of all sons of members of P that are not themselves in P. Suppose that

(1) $R \upharpoonright P = S \upharpoonright P$

and that every $n \in Q$ has an ancestor m such that $Rm = C$ and either

(2) $Eval_{\emptyset}[I](R/m\cdot(0)) \neq no \;\&\; m\cdot(2)$ anc n

or

(3) $Eval_{\emptyset}[I](R/m\cdot(0)) \neq yes \;\&\; m\cdot(1)$ anc n.

Then $Eval_{\emptyset}[I]R = Eval_{\emptyset}[I]S$.

Proof: Beginning with the leftmost leaf of R, we form a list $(p_0, p_1, \ldots)$ of nodes in R together with a list $(\xi_0, \xi_1, \ldots)$ of values in $\underset{\sim}{D} \cup \{\infty\}$, where ξ_j is the value of the subtree of R at p_j. We proceed in Polish suffix order, except that a node n labelled C is added to the list as soon as the relevant sons have been added. The irrelevant sons and their descendents are never added at all. Eventually we have $p_J = ()$ and $\xi_J = Eval_{\emptyset}[I]R$.

Now consider similar lists $(p'_0, p'_1, \ldots)$ and $(\xi'_0, \xi'_1, \ldots)$ for S. By complete induction on j, $p_j = p'_j \in P$ and $\xi_j = \xi'_j$ for all j. For j=J we have $Eval_{\emptyset}[I]R = Eval_{\emptyset}[I]S$. □

(2.6) Theorem. Tree equivalence implies strong equivalence.

Proof: Let $\mathfrak{R}$ and $\mathfrak{S}$ be tree equivalent recursion schemes with complete expansions (R,M) and (S,N) and possibly infinite trees R_∞ and S_∞. Let I be a base interpretation of V on $\underset{\sim}{D}$. We must show that $Val[I]\mathfrak{R} = Val[I]\mathfrak{S}$. By symmetry, it will suffice to show this under the assumption $Val[I]\mathfrak{S} \neq \infty$.

Suppose some $\xi \in \underset{\sim}{D}$ has $Val[I]\mathfrak{S} = \xi$. We will show that some $k < |S|$ has

(1) $Eval_{\emptyset}[I]S_k = \xi$

and then that some $j < |R|$ has

(2) $Eval_{\emptyset}[I]R_j = Eval_{\emptyset}[I]S_k$.

By these and the definition (1.3.8), we will then have $Val[I]\mathfrak{R} = Val[I]\mathfrak{S}$, as wanted.

Let $T \overset{=}{1}> T'$ iff T can become T' by an application of one of $\mathfrak{S}$'s rule-schemata. Let $T \overset{=}{2}> T'$ iff T can become T' by an application of a rule in $\underset{\sim}{R}_{giv}[I] \cup \underset{\sim}{R}_C[I]$. Applications of rules in this set cannot create calls on new function symbols and are not necessary before macroexpansion of calls already present, so

(3) $(\overset{=}{2}> \overset{=}{1}>) \subseteq (\overset{*}{1}=> \overset{*}{2}=>)$.

(Readers familiar with [7] will have no difficulty in constructing a formal proof.)

By (3) and induction on the number of macroexpansions needed to derive $\overline{\xi}$ from S_0, some $T \in (V \cup D)_{\#}$ has

(4) $S_0 \overset{*}{\overline{I}}> T$ & $\mathrm{Eval}_{\emptyset}[I]T = \xi$.

Let P be the set of nodes in Dom T whose ancestors are all labelled in $G \cup H$. We will use Lemma 2.5 with T in the role of R and one of the S_k in the role of S in Lemma 2.5.

By the Church-Rosser property (as in (1.3.5)) and the eventually increasing character of $(|N_0|, |N_1|, \ldots)$, some $k < |S|$ has $P \subseteq \Delta_k$ and $S_k \wedge P = T \wedge P$, where Δ_k is the contribution from S_k to S_∞ (as in (2.2.1)). We verify the hypothesis of Lemma 2.5 with the aid of (4) and $\xi \neq \infty$. The lemma implies that $\mathrm{Eval}_{\emptyset}[I]T = \mathrm{Eval}_{\emptyset}[I]S_k$. This proves (1).

Now that k has been chosen, let H be the set of all hopeful nodes of S_∞ that are in Δ_k. By tree equivalence, there is a $j < |R|$ such that $H \subseteq \Gamma_j$ and $R_j \wedge H = S_k \wedge H$. We apply Lemma 2.5 again, but now S_k plays the role of R and R_j plays the role of S in Lemma 2.5. This proves (2). □

There are counterexamples to the converse, even for monadic recursion schemes which are "anarchic" in the sense of [10, Def. 3.13]. Indeed, let $\mathfrak{R} = (L, f(x))$, where L consists of the rule-schemata

$f(u) \to C(P(u), g(A(u)), B(u))$

$g(u) \to C(Q(u), g(D(u)), H(f(E(u))))$.

Let $\mathfrak{S} = (M, f(x))$, where M consists of the rule-schemata

$f(u) \to C(P(u), H(g(A(u))), B(u))$

$g(u) \to C(Q(u), g(D(u)), f(E(u)))$.

Then $\mathfrak{R}$ and $\mathfrak{S}$ are strongly equivalent but not tree equivalent. An open problem is the construction of an effective map J such that, for some interesting class of recusion schemes, $\mathfrak{R}$ is strongly equivalent to $\mathfrak{S}$ iff $J\mathfrak{R}$ is tree equivalent to $J\mathfrak{S}$.

3. Grammatical Characterization of Tree Equivalence

Rounds [9, p. 115] proved that the infiniteness problem for indexed languages is solvable by assigning a context-free grammar to each system in a class of tree-manipulating systems closely related to recursive definitions. By adapting his construction to our situation, we assign a grammar to each recursion scheme in such a way that schemes are tree equivalent iff their grammars generate the same context-free language.

For the sake of brevity we suppose that a recursion scheme $\mathfrak{R} = (L, f(x_0, \ldots, x_{-1}))$ is given, so that we can omit "let $\mathfrak{R}$ be a recursion scheme" and the like from our assertions. We will construct a grammar $\mathrm{CFG}(\mathfrak{R})$ with a set V_T of terminals, a set V_N of nonterminals, an initial nonterminal $\mathfrak{X}$, and a set Π of productions. Let $[\alpha, i]$ be a new symbol for each $\alpha \in V$ with positive rank and each $i < \rho(\alpha)$. Let $\mathfrak{X}$ be another new symbol. Using new symbols and some of the symbols from V itself, we define the terminal and nonterminal vocabularies of $\mathrm{CFG}(\mathfrak{R})$.

(3.1.1) $V_T = \{x_0,\ldots,x_{-1}\}\cup\{\alpha\varepsilon G_0 \mid (\exists R\to S\varepsilon L)(S^{-1}(\alpha)\neq\emptyset)\}\cup\{[\alpha,i] \mid i<\rho(\alpha)\,\&\,\alpha\varepsilon G\,\&\,(\exists R\to S\varepsilon L)(S^{-1}(\alpha)\neq\emptyset)\}$

(3.1.2) $V_N = \{\mathfrak{X}\}\cup\{g\varepsilon F \mid (\exists R\to S\varepsilon L)(S^{-1}(g)\neq\emptyset)\}\cup\{[g,i] \mid i<\rho(g)\,\&\,g\varepsilon F\,\&\,(\exists R\to S\varepsilon L)(S^{-1}(g)\neq\emptyset)\}$

Paths in trees can be represented by strings in a manner that preserves information about branching and that uses strings in $(V_T \cup V_N)^*$ when the trees are from a complete expansion of $\mathfrak{R}$. Recall that nodes in trees are strings of integers and that $h{:}n$ is the h-long prefix of n.

(3.2) Definition. Let $S \varepsilon V_\#$ and $n \varepsilon$ Dom S. The preencoding of n in S is the string w of length $|n|$ such that, for all $h < |n|$,

(1) $w_h = [\alpha,n_h]$ where $\alpha = S(h{:}n)$.

The encoding of n in S is the string $w\bullet(Sn)$, where w is the preencoding. The set of all encodings of nodes in $S^{-1}(F\cup G_0\cup H)$ is denoted $P_+[S]$. For each $u \varepsilon W$, the set of all preencodings of nodes in $S^{-1}(u)$ is denoted $P_u[S]$.

We wish to generate encodings of leaves in the tree R_∞ by simulating macroexpansions with context-free productions. First we define a set of productions for the initial symbol.

(3.3.1) $\Pi_{init} = \{\mathfrak{X} \to (f)\} \cup \{\mathfrak{X} \to ([f,i],x_i) \mid i < \rho(f)\}$

Next we define a set of productions that simulate the effects of macroexpansion on paths through calls on new functions.

(3.3.2) $\Pi_{thru} = \{[g,i] \to v \mid (\exists R \to S\varepsilon L)(R() = g \;\&\; i < \rho(g) \;\&\; v \varepsilon P_{R(i)}[S])\}$

Finally, we define a set of productions that simulate the effects of macroexpansion on paths that end at calls on new functions.

(3.3.3) $\Pi_+ = \{g \to v \mid (\exists R \to S \varepsilon L)(R() = g \;\&\; v \varepsilon P_+[S])\}$

The union

(3.3.4) $\Pi = \Pi_{init} \cup \Pi_{thru} \cup \Pi_+$

is the set of productions of CFG($\mathfrak{R}$).

Now let $\overset{*}{\underset{L}{\Rightarrow}}$ be the leftmost derivability relation [8, p. 227] defined by CFG($\mathfrak{R}$). Let (R,M) be a complete expansion of $\mathfrak{R}$ and let $w \varepsilon (V_T \cup V_N)^*$. By straightforward induction on lengths of derivations we can prove two lemmas.

(3.4) Lemma. For any $i < \rho(f)$,

$([f,i],x_i) \overset{*}{\underset{L}{\Rightarrow}} w\bullet(x_i)$ iff $(\exists j<|R|)(\exists n\varepsilon\text{Dom } R_j)(R_j n = x_i \,\&\, w$ preencodes n in $R_j)$. □

(3.5) Lemma.

(f) $\overset{*}{\underset{L}{\Rightarrow}} w$ iff $(\exists j<|R|)(\exists n \varepsilon$ Dom $R_j)(R_j n \varepsilon F\cup G_0$ and w encodes n in $R_j)$. □

Combining these lemmas with the definition of Π_{init} (3.3.1) and the construction for the possibly infinite tree R_∞ in Lemma 2.2, we have

(3.6) Corollary. The language generated by $CFG(\mathfrak{R})$ is exactly the set of encodings of leaves in R_∞. □

Before drawing our conclusions about tree equivalence, we note that the language in question is the disjoint union of $\rho(f) + 1$ languages, one for each rule in Π_{init}. If we replace Π_{init} by $\{\mathcal{X} \to (f)\}$ and then by each $\{\mathcal{X} \to ([f,i],x_i)\}$ for $i < \rho(f)$, we obtain grammars $CFG_+(\mathfrak{R}), CFG_0(\mathfrak{R}), \ldots, CFG_{-1}(\mathfrak{R})$ that are somewhat simpler than $CFG(\mathfrak{R})$.

(3.7) Theorem. Let $\mathfrak{R} = (L,f(x_0,\ldots,x_{-1}))$ and $\mathfrak{S} = (M,g(y_0,\ldots,y_{-1}))$ be recursion schemes. Then $\mathfrak{R}$ and $\mathfrak{S}$ are tree equivalent iff $CFG(\mathfrak{R})$ and $CFG(\mathfrak{S})$ generate the same language, so that $\mathfrak{R}$ and $\mathfrak{S}$ are tree equivalent iff the following all hold:

(1) $\quad \text{Lang } CFG_+(\mathfrak{R}) = \text{Lang } CFG_+(\mathfrak{S})$

(2) $\quad (\forall i < \rho(f))(\forall j < \rho(g))\ [x_i = y_j \text{ implies } \text{Lang } CFG_i(\mathfrak{R}) = \text{Lang } CFG_j(\mathfrak{S})]$

(3) $\quad (\forall i < \rho(f))\ [(\forall j < \rho(g))(x_i \neq y_j) \text{ implies } \text{Lang } CFG_i(\mathfrak{R}) = \emptyset]$

(4) $\quad (\forall j < \rho(g))\ [(\forall i < \rho(f))(x_i \neq y_j) \text{ implies } \text{Lang } CFG_j(\mathfrak{S}) = \emptyset]$.

Proof: Comparing the definitions (2.3) and (2.4) for tree equivalence with the definition (3.2) of encodings, we see that $\mathfrak{R}$ and $\mathfrak{S}$ are tree equivalent iff the set of encodings of leaves in R_∞ is exactly the set of encodings of leaves in S_∞. By Corollary 3.6 we have the first "iff" assertion. By

$$\text{Lang } CFG(\mathfrak{R}) = \text{Lang } CFG_+(\mathfrak{R}) \cup \bigcup_{i < \rho(f)} \text{Lang } CFG_i(\mathfrak{R}),$$

$$\text{Lang } CFG_+(\mathfrak{R}) \subseteq V_T^* \cdot G_0^1,$$

$$(\forall i < \rho(f))(\text{Lang } CFG_i(\mathfrak{R}) \subseteq V_T^* \cdot (x_i)),$$

and the corresponding relations for $\mathfrak{S}$, we have the second "iff" assertion.

4. Application of LL(k) Techniques

The strongest available decidability result for equivalence of context-free grammars is for LL(k) grammars [8, Thm. 8]. Although $CFG(\mathfrak{R})$ is hardly ever LL(k) for a nontrivial $\mathfrak{R}$, the subgrammars $CFG_+(\mathfrak{R})$ and $CFG_i(\mathfrak{R})$ do have the property in some interesting cases.

(4.1) Definition. For any computable map μ from recursion schemes to positive integers, let $\mathcal{LL}(\mu)$ be the set of all $\mathfrak{R} = (L,f(x_0,\ldots,x_{-1}))$ such that the grammars $CFG_+(\mathfrak{R})$ and $CFG_i(\mathfrak{R})$ for all $i < \rho(f)$ are $LL(\mu(\mathfrak{R}))$.

Tree equivalence is decidable for $\mathcal{LL}(\mu)$ recursion schemes by Theorem 3.7 and the well known decidability of the context-free emptiness problem. There is a systematic procedure for deciding whether a recursion scheme is in $\mathcal{LL}(\mu)$ by [8, Thm. 1], but this is rather cumbersome and indirect. We therefore

consider sufficient conditions for the $\mathcal{LL}(\mu)$ property that can be tested by simple inspection of the recursion scheme before construction of the grammars.

If $R \to S$ is a rule-schema of $\mathfrak{R}$ and n,n' are distinct occurrences of a parameter u_j in S, then the preencodings v,v' of n,n' lead to productions $[g,j] \to v$ and $[g,j] \to v'$ in $CFG_j(\mathfrak{R})$, where $g = R()$. We wish to distinguish v from v' by inspecting terminals that appear to the left of any nonterminals: i.e., by inspecting ancestors of n,n' labelled by given function symbols.

(4.2) Definition. A recursion scheme $\mathfrak{R} = (L,f(x_0,\ldots,x_{-1}))$ is left-separable iff, whenever $R \to S \in L$ and n,n' are distinct occurrences of a single parameter in S, then some common ancestor m of n and n' has

(1) $(\forall h < |m|)(S(h+1 : m) \in G)$

(2) $n_{|m|} \neq n'_{|m|}$.

(4.3) Definition. The depth $\delta(\mathfrak{R})$ of a recursion scheme $\mathfrak{R} = (L,f(x_0,\ldots,x_{-1}))$ is the largest $|n| + 1$ such that $n \in \mathrm{Dom}\ S$ for some $R \to S \in L$.

(4.4) Theorem. If $\delta \leq \mu$ then every left-separable recursion scheme without constants is in $\mathcal{LL}(\mu)$.

Proof: Let $\mathfrak{R} = (L,f(x_0,\ldots,x_{-1}))$ be a left-separable recursion scheme without constants, so that $CFG_+(\mathfrak{R})$ generates $\emptyset$ and is therefore LL(1). We will show that each $CFG_i(\mathfrak{R})$ for $i < \rho(f)$ is $LL(\delta(\mathfrak{R}))$.

The initial symbol has just one production, $\mathfrak{X} \to ([f,i],x_i)$, in $CFG_i(\mathfrak{R})$. The other nonterminals accessible from $\mathfrak{X}$ have the form $[g,j]$ for $g \in F$ and $j < \rho(g)$. Suppose that $[g,j] \to v$ and $[g,j] \to v'$ are distinct productions for $[g,j]$. Let w,w' be the maximal terminal prefixes of v,v'. By $|w| \leq \delta(\mathfrak{R})$ and $|w'| \leq \delta(\mathfrak{R})$, it will suffice to show that $w_h \neq w'_h$ for some $h < \mathrm{Min}(|w|,|w'|)$.

Let n,n' be the occurrences of a parameter u_j preencoded by v,v'. By the definition (4.2) of left-separability, n and n' have a common ancestor m such that $|m| < \mathrm{Min}(|w|,|w'|)$ and $n_{|m|} \neq n'_{|m|}$. Setting $h = |m|$ and referring to (3.2.1) in the definition of preencodings, we have

$$w_h = [Sm,n_h] \neq [Sm,n'_h] = w'_h. \square$$

By systematically interchanging pairs of words like "left" and "right" or "prefix" and "suffix", we can convert any concept or result in formal language theory to a mirror image. In particular, we may define RR(k) grammars, which can be parsed from the top down by DPDA's that scan their inputs from the right and that invert rightmost derivation sequences. The following definition is the mirror image of (4.1).

(4.5) Definition. For any computable map μ from recursion schemes to positive integers, let $\mathcal{RR}(\mu)$ be the set of all $\mathfrak{R} = (L,f(x_0,\ldots,x_{-1}))$ such that the grammars $CFG_+(\mathfrak{R})$ and $CFG_i(\mathfrak{R})$ for all $i < \rho(f)$ are $RR(\mu(\mathfrak{R}))$.

Tree equivalence is decidable for $\mathcal{RR}(\mu)$ recursion schemes by Theorem 3.7 and the mirror image of [8, Thm. 8]. Since "left" and "right" in the grammars correspond to "up" and "down" in the trees, the following definition will act as a mirror image of (4.2).

(4.6) Definition. A recursion scheme $\mathfrak{R} = (L,f(x_0,\ldots,x_{-1}))$ is right-separable iff, whenever $R \to S \in L$ and n,n' are distinct occurrences of a single parameter in S, then some ancestors m of n and m' of n' and some nonnegative integer J have

(1) $|n| - |m| = |n'| - |m'| = J$

(2) $(\forall j < J)(S(|m| + j : n) \in G \;\&\; S(|m'| + j : n') \in G)$

(3) $Sm \neq Sm'$ or $(J \neq 0 \;\&\; n_{|m|} \neq n'_{|m'|})$.

By the same reasoning used for Theorem 4.4, we have the following theorem.

(4.7) Theorem. If $\delta \leq \mu$ then every right-separable recursion scheme without constants is in $\mathcal{RR}(\mu)$. □

The restriction to separable recursion schemes is severe but not crippling. Figure 2 displays a free [4, §6] flowchart scheme that corresponds to the recursion scheme $\mathfrak{R} = (L,f(X,Y))$, where L consists of the rule-schemata

$$f(u,v) \to C(P(u),f(A(u,u),v),g(u,B(v)))$$

$$g(u,v) \to C(Q(u),f(B(v),v),v).$$

The grammar $CFG_0(\mathfrak{R})$ is not LL(k) for any k; the grammar $CFG_1(\mathfrak{R})$ is not RR(k) for any k. On the other hand, consider $(M,f(x))$, where M consists of the rule-schema

$$f(u) \to C(P(u),u,H(f(L(u)),f(R(u)))).$$

This is a left-separable and right-separable recursion scheme without constants that cannot be simulated by any flowchart scheme [6, Thm. 1].

5. Discussion

Ashcroft, Manna and Pnueli [1, Thm. 4] proved that strong equivalence itself is decidable for free monadic recursion schemes by adapting the DPDA construction from [8, Thm. 8]. The general idea that a very strong form of program equivalence could be expressed as generative equivalence of context-free grammars was put forward by Zeiger [11, §8] but was only worked out in a situation where the schemes were monadic and the grammars were actually regular [11, §5]. Our example at the end of §4 includes dyadic function letters and involves a nonregular language.

Our recursive calculus differs from the one studied by Paterson and Hewitt [6] in that we allow top-down macroexpansion while they require bottom-up macroexpansion. Suppose that $R \to S$ is a rule-schema

and u is a parameter in R that would be thrown away on some branches of conditionals in S, where the tests in the conditionals do not involve u. Then a recursion scheme using $R \to S$ can give us finite values in some interpretations where all bottom-up computations diverge. This situation cannot arise in Strong's calculus because each "predicate" must take all the formal parameters as arguments [10, Def. 3.1], so his formalism is a special case of ours in which one may proceed upward or downward according to the convenience of the moment.

Studies of recursive computation usually impose the bottom-up restriction, but we have several reasons for preferring to allow top-down computing.

First, top-down macroexpansion gains some finite values but loses none. The only difference in values that can occur is

$$\mathrm{Val}[I]\Phi(\mathrm{BU}) = \infty \;\&\; \mathrm{Val}[I]\Phi(\mathrm{TD}) \neq \infty.$$

Second, the two methods can be implemented with equal ease and efficiency in random access memories, using well known linked data structure manipulation techniques (viz. Knuth [3, Chap. 2]). The lack of side effects in pure recursion allows us to avoid independent evaluations of all 41 copies of an argument expression substituted for a formal parameter that occurs 41 times in the right half of a rule-schema. The familiar difficulties with ALGOL 60 call-by-name simply do not arise.

Third, none of the known results in schematology have been shown to require the bottom-up restriction. The basic facts about singlevaluedness and minimal fixed points are more difficult to establish for top-down computation, but this has been done [2] [7, §§7,8]. Some of the results derived for bottom-up macroexpansion may really depend upon this restriction, however. Before such questions are investigated, it will be premature to urge that top-down or bottom-up macroexpansion be taken as the only correct way to compute recursively.

Acknowledgment

I am grateful to H. R. Strong for his comments on a preliminary version of this note.

References

[1] E.A. Ashcroft, Z. Manna and A. Pnueli, Decidable Properties of Monadic Functional Schemas, Artificial Intelligence Memo. AIM-148, Stanford University, Stanford, California, July 1971.

[2] J.M. Cadiou and Z. Manna, Recursive Definitions of Partial Functions and Their Computations, Proc. ACM Conf. on Proving Assertions About Programs (1972), 58-65.

[3] D.E. Knuth, "The Art of Computer Programming, Vol. I: Fundamental Algorithms", Addison-Wesley, Reading, Massachusetts, 1968.

[4] D.C. Luckham, D.M.R. Park, and M.S. Paterson, On Formalized Computer Programs, J. Computer and System Science 4 (1970), 220-249.

[5] J. McCarthy, Basis for a Mathematical Theory of Computation. In P. Braffort and D. Hirschberg (Eds.), "Computer Programming and Formal Systems", North-Holland, Amsterdam, 1963, 33-70.

[6] M.S. Paterson and C.E. Hewitt, Comparative Schematology, Proc. ACM Conf. on Concurrent Systems and Parallel Computation (1970), 119-127.

[7] B.K. Rosen, Tree-Manipulating Systems and Church-Rosser Theorems, J.ACM (to appear).

[8] D.J. Rosenkrantz and R.E. Stearns, Properties of Deterministic Top-Down Grammars, Information and Control 17 (1970), 226-256.

[9] W.C. Rounds, Tree-Oriented Proofs of Some Theorems on Context-Free and Indexed Languages, Second Annual ACM Symposium on Theory of Computing (1970), 109-116.

[10] H.R. Strong, Translating Recursion Equations Into Flow Charts, J. Computer and System Science 5 (1971), 254-285.

[11] H.P. Zeiger, Formal Models for Some Features of Programming Languages, ACM Symposium on Theory of Computing (1969), 211-215.

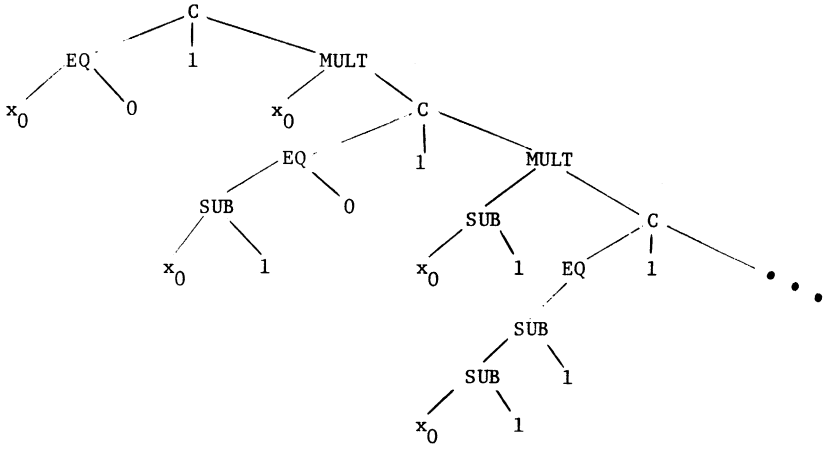

Figure 1. Infinite tree generated by the factorial recursion scheme.

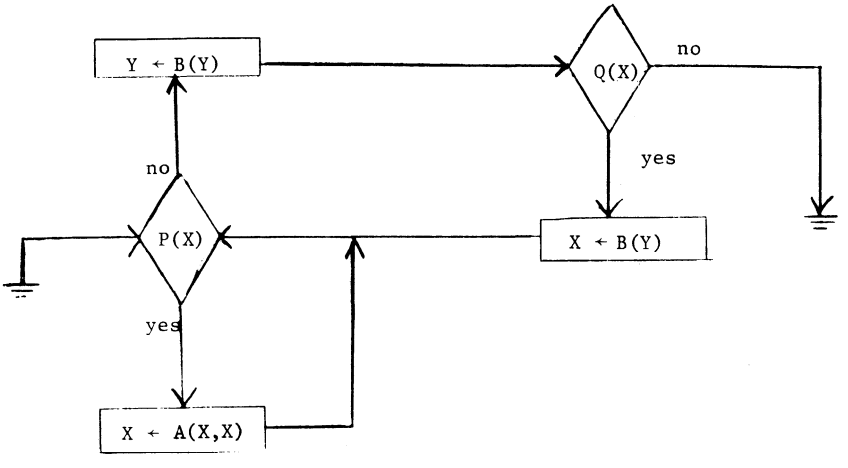

Figure 2. Free Flowchart Scheme With Input Locations X,Y and Output Location Y.