Schema Revisited

Colin Stirling

Division of Informatics
University of Edinburgh
Edinburgh EH9 3JZ, UK
cps@dcs.ed.ac.uk

1 Introduction

Two schema problems from the 1970s are examined, monadic recursion schemes and first-order recursion schemas. Research on these problems halted because they were shown to be equivalent to the problem of decidability of language equivalence between DPDA (deterministic pushdown automata). Recently a decidability proof for equivalence of DPDA was given by Sénizergues [10, 11], which therefore also solves the schema problems. However Sénizergues proof is quite formidable. A simplification of the proof was presented by the author [13] using ideas from concurrency theory (for showing decidability of bismilarity [9, 12]) and crucial insights from Sénizergues's intricate proof.

In this abstract we concentrate on first-order schemes and we outline a solution, based on the DPDA equivalence proof, which is reasonably close to its original formulation. We make use of Courcelle's work [1, 2], which shows how to reduce this schema problem to decidability of language equivalence between strict deterministic grammars. And the proof in [13] of decidability of DPDA equivalence proceeds via (a small extension of) these grammars.

2 Monadic recursion schemes

A monadic recursion scheme, following Garland and Luckham [6], is defined relative to a set of basis monadic functions $\mathsf{F} = \{f_1, \ldots, f_k\}$ and a set of predicates $\mathsf{P} = \{P_1, \ldots, P_l\}$ as a finite family

$$\begin{array}{l} F_1 x \stackrel{\text{def}}{=} \text{if } P_1 x \text{ then } \alpha_1 x \text{ else } \beta_1 x \\ \vdots \qquad\qquad \vdots \\ F_n x \stackrel{\text{def}}{=} \text{if } P_n x \text{ then } \alpha_n x \text{ else } \beta_n x \end{array}$$

where each F_i is distinct and each α_i and β_i is a string of defined and basis functions, a member of $(\mathsf{F} \cup \mathsf{DF})^*$ when $\mathsf{DF} = \{F_1, \ldots, F_n\}$, and each $P_i \in \mathsf{P}$. The scheme is usually identified by the initial defined function F_1.

Example 1 A simple example is $Fx \stackrel{\text{def}}{=} \text{if } Px \text{ then } fx \text{ else } FFfx$. □

A computation of a scheme is defined with respect to an interpretation I which fixes the meanings of the basis functions, predicates and variable. An interpretation I over a non-empty value space D is a mapping such that $I(x) \in D$, $I(f_i) \in D \to D$ and $I(P_i) \in D \to \{\mathtt{tt}, \mathtt{ff}\}$. A computation relative to I is then defined using the following transition rules

$$\begin{array}{ll} F_1 x \to F_1 d \quad \text{if} \quad I(x) = d & \delta f_i d \to \delta d' \quad \text{if} \quad I(f_i)(d) = d' \\ \delta F_i d \to \delta \alpha_i d \text{ if} \quad I(P_i)(d) = \mathtt{tt} & \delta F_i d \to \delta \beta_i d \text{ if} \quad I(P_i)(d) = \mathtt{ff} \end{array}$$

The value of a scheme Fx relative to I, written $\mathrm{Val}_I(Fx)$, is a member of $D_\perp$. If $Fx \longrightarrow^* d$ then $\mathrm{Val}_I(Fx) = d$ and if the computation never ends $\mathrm{Val}_I(Fx) = \perp$.

Two schemes F and G are strongly equivalent, written $F \sim G$, if for all interpretations I, $\mathrm{Val}_I(Fx) = \mathrm{Val}_I(Gx)$. The classical equivalence problem for monadic recursion schemes is to show whether or not there is a decision procedure for $F \sim G$. For background and significance of the problem, see Garland and Luckham [6] and references cited therein.

An interpretation I is free if the domain $D = \mathsf{F}^*\{x\}$ and $I(f) = f$ and $I(x) = x$. The result of a computation relative to a free I is a word (or $\perp$).

Example 2 Let I be the free interpretation for example 1 where $I(P)(f^n x) = \mathtt{tt}$ iff $n > 0$. Therefore $Fx \longrightarrow FFfx \longrightarrow Fffx \longrightarrow fffx$, and $\mathrm{Val}_I(Fx) = fffx$. In contrast, if $I(P)(f^n x) = \mathtt{tt}$ iff n is odd then $\mathrm{Val}_I(Fx) = \perp$. □

Relationships with language theory were underpinned by the following result in Garland and Luckham.

Fact 1 *$F \sim G$ iff for all free interpretations I, $\mathrm{Val}_I(Fx) = \mathrm{Val}_I(Gx)$.*

Garland and Luckham showed that the decision problem for schemes reduces to the problem of decidability of DPDA, and Friedman showed that the converse also holds using jump DPDA [5]. We shall now provide a cleaner variant reduction to the DPDA problem, inspired by these authors.

First we assume a "Greibach" normal form for a scheme. In $F_i \stackrel{\text{def}}{=}$ if $P_i x$ then $\alpha_i x$ else $\beta_i x$ each α_i and β_i has one of the forms ϵ or f_j or F_j or $F_j f_k$ or $F_j F_k$. It is straightforward to transform a scheme into normal form by adding auxilary definitions. For instance, example 1 becomes $Fx \stackrel{\text{def}}{=}$ if Px then fx else Gfx and $Gx \stackrel{\text{def}}{=}$ if Px then FFx else FFx.

Let B be the set of boolean arrays of size l. If $b \in \mathsf{B}$ then b_i is the ith entry of b: the idea is that $b_i = \mathtt{tt}$ means that P_i is true. A stack symbol is an element of DF, a state is a boolean b and the alphabet consists of elements bfb' where $f \in \mathsf{F}$. A configuration of the DPDA has the form $b\delta$ where $\delta \in \mathsf{DF}^*$ is a sequence of stack symbols. Transitions have the form $b\delta \stackrel{bfb'}{\longrightarrow} b'\delta'$ or $b\delta \stackrel{\epsilon}{\longrightarrow} b'\delta'$. Let δ^R be the reverse of δ. Assume a scheme with definition $Fx \stackrel{\text{def}}{=}$ if $P_i x$ then αx else βx. Basic transitions for bF are determined from the scheme as follows.

$$
\begin{array}{rl}
bF \xrightarrow{bfb'} b'\alpha' \text{ if either } & b_i = \mathtt{tt} \text{ and } \alpha^R = f\alpha' \\
\text{or} & b_i = \mathtt{ff} \text{ and } \beta^R = f\alpha' \\
bF \xrightarrow{\epsilon} b\alpha' \text{ if either } & b_i = \mathtt{tt} \text{ and } \alpha^R = \alpha' \text{ and } \alpha \in \mathsf{DF}^* \\
\text{or} & b_i = \mathtt{ff} \text{ and } \beta^R = \alpha' \text{ and } \beta \in \mathsf{DF}^*
\end{array}
$$

There is also the prefix rule, if $bF \xrightarrow{a} b'\alpha'$ then $bF\delta \xrightarrow{a} b'\alpha'\delta$. Basic transitions obey the following two deterministic properties:

$$
\begin{array}{l}
\text{if } bF \xrightarrow{\epsilon} b\delta \text{ then not}(bF \xrightarrow{bfb'} b'\delta') \text{ for any } b'\delta' \\
\text{if } bF \xrightarrow{a} b'\delta' \text{ and } bF \xrightarrow{a} b''\delta'' \text{ then } b' = b'' \text{ and } \delta' = \delta''
\end{array}
$$

The result is therefore a DPDA.

The language of a configuration $b\delta$, written $L(b\delta)$, is the set of words $w \in (\mathsf{B} \times \mathsf{F} \times \mathsf{B})^*$ recognised by $b\delta$ using these transition rules, where ϵ-transitions are swallowed in the usual way, and assuming empty stack acceptance, $L(b\delta) = \{w : b\delta \xrightarrow{w} b'\epsilon \text{ for some } b'\}$. The following is a consequence of Fact 1 and the construction.

Proposition 1 *$F \sim G$ iff for all b, $L(bF) = L(bG)$.*

There is a routine transformation of a DPDA into a context-free grammar (which is strict deterministic [7, 8], more on this later). First one transforms the DPDA into normal form where ϵ-transitions only pop the stack, by examining what happens under repeated basic ϵ-transitions. Next one transforms the normalised DPDA into a context free grammar whose nonterminals are triples of the form bFb' and whose alphabet is the same as that of the DPDA. The idea is that, for instance, a basic transition of the DPDA of the form $bF \xrightarrow{a} b''GH$ becomes in the grammar the family of transitions for each b''', $bFb' \xrightarrow{a} b''Gb'''b'''Hb'$. Hence the language accepted by the nonterminal bFb', $L(bFb')$, is the set of words w such that $bF \xrightarrow{w} b'\epsilon$. Hence $F \sim G$ iff for all b and b', $L(bFb') = L(bGb')$. Decidability of monadic recursion schemes follows from the following theorem.

Theorem 1 *It is decidable whether $L(bFb') = L(bGb')$.*

In this abstract there is only an intimatation of the procedure in section 4, because we shall concentrate on the second schema problem.

3 Recursive program schemes

A recursive program scheme, following Courcelle [1, 2], is defined relative to a set of basis functions $\mathsf{F} = \{f_1, \ldots, f_k\}$ and a set of basis variables $\mathsf{V} = \{x_1, \ldots, x_l\}$. Each basis function f has an associated arity $\rho(f) > 0$, and therefore need not be monadic. A scheme is a finite family of the form

$$\begin{array}{l} F_1(x_1,\ldots,x_{m_1}) \stackrel{\text{def}}{=} t_1 \\ \vdots \qquad\qquad\qquad\qquad \vdots \\ F_n(x_1,\ldots,x_{m_n}) \stackrel{\text{def}}{=} t_n \end{array}$$

where each F_i is distinct, and has an associated arity $\rho(F_i) = m_i$, and where terms t_i are built from the basis and defined functions and variables, and therefore have the form x_j or $f_j(t_1,\ldots,t_{\rho(f_j)})$ or $F_j(t_1,\ldots,t_{\rho(F_j)})$. A scheme is again usually identified with its head function F_1. We let DF be the set of defined functions.

Example 1 A simple example is $F(x) \stackrel{\text{def}}{=} f(F(gx), g(x))$.

The interpretation of a scheme is either the undefined tree or a completed tree whose depth is finite or infinite and where internal nodes are labelled with elements of F and leaves are labelled with elements of V. The following transition rules generate the tree and they are applied down the depth of the tree starting with $F_1(x_1,\ldots,x_{\rho(F_1)})$

$$\begin{array}{l} x_i \longrightarrow x_i \\ F_i(t'_1,\ldots,t'_{\rho(F_i)}) \longrightarrow t_i\{t'_1/x_1,\ldots,t'_{\rho(F_i)}/x_{\rho(F_i)}\} \\ \text{if } t'_j \longrightarrow t''_j,\ 1 \le j \le \rho(f_i), \text{ then } f_i(t'_1,\ldots,t'_{\rho(f_i)}) \longrightarrow f_i(t''_1,\ldots,t''_{\rho(f_i)}) \end{array}$$

where $\{\cdot/\cdot\}$ is simultaneous substitution. For instance in the case of example 1

$$Fx \longrightarrow f(F(gx),gx) \longrightarrow f(f(F(ggx),ggx),gx) \longrightarrow \ldots$$

The value of a scheme belongs to the family $T^\omega_\perp$ of appropriate trees.

Alternatively one can view $T^\omega_\perp$ as a domain[1] of trees. The meaning of a scheme is the least fixed point with respect to the free tree interpretation, following Damm [3, 4]. In the case of example 1

$$F^0(x) = \perp \qquad\qquad F^{i+1}(x) = f(F^i(gx), gx)$$

So $F^2(x) = f(((f\ \perp), ggx), gx)$. The resulting tree in $T^\omega_\perp$ is $F^\omega(x) = \bigsqcup_{i\ge 0} F^i(x)$ which is the meaning of $Y(\lambda F.\,\lambda x.\, f(F(gx),gx))(x)$ with respect to the free interpretation. Thus schemes are only "first-order". Higher order schemes are considered by Damm [3, 4].

Two schemes F and G with arity n are equivalent, written $F \sim G$, if they produce the same tree, that is if $F^\omega(x_1,\ldots,x_n) = G^\omega(x_1,\ldots,x_n)$. The classical equivalence problem for recursion schemes is to show whether or not there is a decision procedure for $F \sim G$. For background and significance of the problem see [1, 2] and references cited therein.

[1] With ordering $\perp \sqsubseteq T$ and $T_i \sqsubseteq T'_i$ for each i implies $f_j(T_1,\ldots,T_{\rho(f_j)}) \sqsubseteq f_j(T'_1,\ldots,T'_{\rho(f_j)})$.

The equivalence problem for schemes was shown to be interreducible to the DPDA problem by Courcelle [2], via grammars. A key idea is to represent a tree $T \in T^{\omega}_{\perp}$ as the language of its finite branches, $B(T)$. The following finite tree $f(g(x_1, x_2), f(x_1, h(x_3)))$ is given as $\{f^1g^1x_1, f^1g^2x_2, f^2f^1x_1, f^2f^2h^1x_3\}$. Each word is a branch. One splits each basis function f of arity k into terminal symbols $f^1, \ldots, f^k$ reflecting the different directions that can be taken to obtain the branch. In the case of T generated by example 1 above $B(T)$ is the deterministic context-free language $\{f^2g^1x, f^1f^2g^1g^1x, f^1f^1f^2g^1g^1g^1x, \ldots\}$.

If $T = T'$ then $B(T) = B(T')$. But the converse need not hold. Consider the trees generated by the schemes $Fx \stackrel{\text{def}}{=} f(Fx)$ and $Gx \stackrel{\text{def}}{=} g(Gx)$. These trees are not "locally finite" [2]. A tree T is locally finite if whenever u is a prefix of a branch of T then there is a finite word v such that $uv \in B(T)$. Locally finite trees with the same branch language are equal. It is straightforward as Courcelle notes to guarantee local finiteness by increasing the arity of the basis functions by one and adding a new variable. Consider the transformed schemes $Fxy \stackrel{\text{def}}{=} f(Fxy, y)$ and $Gxy \stackrel{\text{def}}{=} g(Gxy, y)$. Two schemes are equivalent iff their transformations are also equivalent. Hence we can restrict attention to schemes that generate locally finite trees, and for these the following holds, as shown by Courcelle [2].

Fact 1 *$F \sim G$ iff $B(F^{\omega}(x_1, \ldots, x_n)) = B(G^{\omega}(x_1, \ldots, x_n))$.*

Following Courcelle, the next step is to transform a scheme into a context-free grammar which generates its branch language. We exclude the case where a scheme generates a single node tree x_i: it is easy to directly check equivalence betweeen such schemes. With this exclusion, we assume that schemes are given in "Greibach" normal form. Each term t_i in the definition of F_i has the form $f(rt_1, \ldots, rt_{\rho(f)})$ where $f \in \mathsf{F}$ and where each rt_j is built from variables and defined function symbols only and where the depth of their embedding is at most two. This normal form is easy to achieve by introducing auxilary defined functions.

An ϵ-free context-free grammar in 3-Greibach normal form consists of a finite set N of nonterminals, a finite alphabet A and a finite family of basic transitions, each of the form $X \stackrel{a}{\longrightarrow} \alpha$ where $X \in \mathsf{N}$, $a \in \mathsf{A}$ and $\alpha \in \mathsf{N}^*$ such that its length, $|\alpha|$ is less than 3. A simple configuration is a sequence of nonterminals whose behaviour is determined by the basic transitions and the prefix rule: if $X \stackrel{a}{\longrightarrow} \alpha$ then $X\beta \stackrel{a}{\longrightarrow} \alpha\beta$ where $\beta \in \mathsf{N}^*$. The language accepted by a simple configuration α, $L(\alpha)$, is the set of words $\{w \in \mathsf{A}^* : \alpha \stackrel{w}{\longrightarrow} \epsilon\}$.

Given a scheme in normal form we associate a grammar with it as follows. The alphabet A is the set of split functions f^j, $1 \leq j \leq \rho(f)$ for $f \in \mathsf{F}$. The nonterminals N is the set F^j, $1 \leq j \leq \rho(F)$. Assume in the scheme that $F(x_1, \ldots, x_{\rho(F)}) \stackrel{\text{def}}{=} f(rt_1, \ldots, rt_{\rho(f)})$. The basic transitions are defined as follows, for each nonterminal F^i and alphabet symbol f^j.

$$
\begin{array}{ll}
F^i \xrightarrow{f^j} \epsilon & \text{if } rt_j = x_i \\
F^i \xrightarrow{f^j} G^k & \text{if } rt_j = G(rt'_1, \ldots, rt'_{\rho(G)}) \text{ and } rt'_k = x_i \\
F^i \xrightarrow{f^j} G^k H^l & \text{if } rt_j = G(rt'_1, \ldots, rt'_{\rho(G)}) \text{ and} \\
 & \quad rt'_k = H(rt''_1, \ldots, rt''_{\rho(H)}) \text{ and } rt''_l = x_i
\end{array}
$$

The grammar is defined so that the language accepted by a nonterminal F^i is the set of words $w \in \mathsf{A}^+$ such that $wx_i \in B(F^\omega(x_1, \ldots, x_{\rho(F)}))$. For example in the case of the second transition rule because G has x_i in its kth position in the definition of F it follows that $\{f^j w : w \in L(G^k)\} \subseteq L(F^i)$. Hence the following result holds.

Fact 2 *$F \sim G$ iff for each i, $L(F^i) = L(G^i)$.*

4 The decision procedure

The disjoint union of two recursion schemes is a single scheme and therefore we need only consider a single grammar. The equivalence problem is then to show that for each i, $L(F^i) = L(G^i)$ for F, $G \in \mathsf{DF}$. We assume that the grammar is "tidied" as usual by removing redundant nonterminals (those not reachable from any F^i and G^i and those whose language is $\emptyset$). In the following we use X, Y and Z to range over nonterminals and α, β to range over sequences of nonterminals.

The decision procedure consists of two semi-decision procedures. One half is easy, if $L(F^i) \neq L(G^i)$ then there is a smallest word which distinguishes them. The other half is more difficult. We show that $F \sim G$ iff there is a finite tableau proof for this. Tableaux have been used for proving decidability of bisimulation equivalence [9, 12]. They are also implicit in Sénizergues's proof [11] where they appear as strategies.

The tableau proof system is goal directed, and consists of two kinds of rules, simple and conditional. Simple rules have the form

$$
\frac{\text{Goal}}{\text{Subgoal}_1 \ \ldots \ \text{Subgoal}_n}\ \mathcal{C}
$$

where Goal is what currently is to be proved and the subgoals are what it reduces to, provided the side condition $\mathcal{C}$ holds. A conditional rule has the form

$$
\frac{\begin{array}{c}\text{Goal}_1 \\ \vdots \\ \text{Goal}_k \\ \vdots \\ \text{Goal}\end{array}\ \mathcal{C}}{\text{Subgoal}}
$$

where Goal is the current goal to be shown and there is a single subgoal to which it reduces provided that the goals Goal_1,...,Goal_k occur above Goal on the path between it and the root (starting goal) and provided that the side condition $\mathcal{C}$ holds. The use of conditional tableau rules is a new innovation, which is essentially due to Sénizergues.

There is also the important notion of when a current goal counts as final. Final goals are classified as either successful or unsuccessful. A *tableau proof* for a starting Goal is a finite proof tree, whose root is Goal and all of whose leaves are successful final goals, and all of whose inner subgoals are the result of an application of one of the rules.

The first tableau proof rule is the initial simple rule, INIT.

$$\frac{F = G}{F^1 = G^1 \ldots F^n = G^n}$$

The initial goal $F = G$, "are schemes F and G equivalent?" reduces to the subgoals $F^i = G^i$, "is $L(F^i) = L(G^i)$?", for each i.

The main idea of the tableau proof system is to reduce goals to subgoals by following branches down the trees for F and G. F^i represents the subtree for F all of whose branches end in x_i. The configuration $(F^i \cdot w)$ represents the subtree for F given by taking path w down the subtree F^i: it is therefore the subtree whoses branches are $\{v : wvx_i \text{ is a branch of the tree for } F\}$. Clearly $F \sim G$ iff for all w and i, $(F^i \cdot w) \sim (G^i \cdot w)$. We show that the subtree $(F^i \cdot w)$ is naturally described in the grammar.

Basic transitions of the grammar induced by a scheme are "almost" deterministic. If $F^i \xrightarrow{f^j} \epsilon$ and $F^i \xrightarrow{f^j} \alpha$ then $\alpha = \epsilon$ because x_i is in the jth position of f and nothing else is thereby allowed. However if the jth position of f is $G(rt'_1, \ldots, rt'_{\rho(G)})$ then it is possible that $F^i \xrightarrow{f^j} \alpha$ and $F^i \xrightarrow{f^j} \beta$ when $\alpha \neq \beta$. However α and β are "similar": if $\alpha = G^k\alpha'$ then β must have the form $G^l\beta'$ and if $l = k$ then α' and β' must again be similar (both of the form $H^{l'}$). The grammar is in fact strict deterministic [7, 8].

Let $\equiv$ be a partition of the nonterminals N of a context-free grammar (in normal form). The partition $\equiv$ is extended to sequences of nonterminals, $\alpha \equiv \beta$ if $\alpha = \beta$ or there is a δ such that $\alpha = \delta X \alpha_1$ and $\beta = \delta Y \beta_1$ and $X \equiv Y$ and $X \neq Y$. A partition $\equiv$ on N is strict if the basic transitions obey the following two conditions:

$$\text{if } X \xrightarrow{a} \alpha \text{ and } Y \xrightarrow{a} \delta \text{ and } X \equiv Y \text{ then } \alpha \equiv \delta$$
$$\text{if } X \xrightarrow{a} \alpha \text{ and } Y \xrightarrow{a} \alpha \text{ and } X \equiv Y \text{ then } X = Y$$

A context-free grammar is strict deterministic if there exists a strict partition of its nonterminals. The partition on the grammar induced by a scheme is given by $F^i \equiv F^j$ for each F and indices i and j. Clearly the two strictness conditions hold[2]. Hence for α, $\beta \neq \epsilon$ and $\alpha \neq \beta$, $\alpha \equiv \beta$ if $\alpha = \delta F^i \alpha'$ and $\beta = \delta F^j \beta'$ for $i \neq j$.

[2] Similarly the context-free grammar induced by a monadic recursion scheme is strict deterministic when the partition is given by $bFb' \equiv bFb''$

The strictness conditions generalise to words (replacing a with $w \in \mathsf{A}^+$ throughout). It therefore follows that if $X \equiv Y$ then the languages accepted by X and Y are prefix-disjoint and if $X \neq Y$ then they accept disjoint languages. That is, if $w \in L(F^i)$ and $i \neq j$ then no prefix of w including w belongs to $L(F^j)$. This is clear from the tree generated by F: if wx_i is a branch then this excludes vx_j as a branch whenever v is a prefix of w.

A simple configuration of a grammar is a sequence of nonterminals β. A composite configuration is a finite family of simple configurations, $\beta_1 + \ldots + \beta_n$. The language accepted by a composite configuration is the union of the languages acccepted by the components, $L(\beta_1 + \ldots + \beta_n) = \bigcup L(\beta_i)$. For simplicity we also assume that the empty sum, $\emptyset$, is also a configuration. Our main concern is with a subset of such configurations: $\beta_1 + \ldots + \beta_n$ is *admissible* if $\beta_i \equiv \beta_j$ for each pair of components β_i and β_j. Note that the singleton member ϵ is admissible and so is $\emptyset$. Subtrees of (the tree for) F such as $(F^i \cdot w)$ are represented as admissible configurations. Let $(F^i \cdot a)$ be defined as $\sum\{\alpha : F^i \xrightarrow{a} \alpha \text{ is a basic transition}\}$ which is an admissible configuration because the grammar is strict. $L((F^i \cdot a))$ is $\{w : awx_i \text{ is a branch in the tree for } F\}$. If $A = X_1\beta_1 + \ldots + X_n\beta_n$ is admissible then $(A \cdot a)$ is $\sum\{\alpha_{i_1}\beta_1 : X_1 \xrightarrow{a} \alpha_{i_1}\} + \ldots + \sum\{\alpha_{i_n}\beta_n : X_n \xrightarrow{a} \alpha_{i_n}\}$, which is also admissible. The notation is extended to words. $(A \cdot \epsilon) = A$ and $(A \cdot aw) = (A \cdot a) \cdot w$, where $(\emptyset \cdot w) = \emptyset$. It is easy to check that for any w, if A is admissible then $(A \cdot w)$ is also admissible.

We now return to the tableau construction. We let A, B, C and D range over admissible configurations. Goals in the tableau proof system (except for the initial goal $F = G$) have the form $A = B$. The next tableau proof rule is again a simple rule, UNF (unfold). Let $\mathsf{A} = \{a_1, \ldots, a_k\}$.

$$\frac{A = B}{(A \cdot a_1) = (B \cdot a_1) \quad \ldots \quad (A \cdot a_k) = (B \cdot a_k)}$$

UNF allows one to walk down the trees for F^i and G^i. UNF is the strategy T_A in Sénizuergues's proof.

The size of an admissible configuration $A = \beta_1 + \ldots + \beta_n$, written $|A|$, is the length of its largest sequence, $\max\{|\beta_i| : 1 \leq i \leq n\}$. A has many different "shapes", as it can be written in many different ways using obvious equalities (such as $B(C + D) = BC + BD$). A basic shape is a head nonterminal form $X_1A_1 + \ldots + X_kA_k$ where $X_i \neq X_j$, $i \neq j$, and $X_i \equiv X_j$. In this case the X_is are heads and A_is are tails. Another head form is $\alpha_1A_1 + \ldots + \alpha_nA_n + B$ where $\alpha_i \equiv \alpha_j$ and $|\alpha_i| = |\alpha_j|$ and no $A_j = \epsilon$ and $|B| \leq |\alpha_i|$. Instead one may focus on tail forms. If $(X_i \cdot w) = D_i$ (where D_i may be $\emptyset$ and for no prefix v of w is $X_i \cdot v = \epsilon$) then $(X_1A_1 + \ldots + X_kA_k \cdot w) = D_1A_1 + \ldots + D_kA_k$. The shape $D_1A_1 + \ldots + D_kA_k$ highlights the tails A_i. Because the grammar is in 3-Greibach normal form $|D_i| \leq 1 + |w|$ for each i.

Associated with any nonterminal F^i is a smallest word $w(F^i)$ such that $w(F^i) \in L(F^i)$, and so $(F^i \cdot w(F^i)) = \epsilon$. Note that if $(F^i \cdot v) = \epsilon$ and $j \neq i$ then $(F^j \cdot v) = \emptyset$. An important measure is M which is $\max\{|w(X)| : X \text{ is a nonterminal}\}$.

UNF allows one to proceed down the trees for F and G. Any subgoal $A = B$ can be thought of as $(F^i \cdot w) = (G^i \cdot w)$ where w is a prefix of a branch. The next step is to permit tree surgery and transplantation to "balance" the subtree expressions. We give the tableau rule BAL(L). This is a conditional tableau rule. In Sénizuergues's proof this is the strategy T_B.

$$\frac{\begin{array}{c} X_1A_1 + \ldots + X_kA_k = B \\ \vdots \qquad\qquad \mathcal{C} \\ D_1A_1 + \ldots + D_kA_k = B' \end{array}}{D_1(B \cdot w(X_1)) + \ldots + D_k(B \cdot w(X_k)) = B'}$$

where $\mathcal{C}$ is the condition: there are exactly M applications of UNF between the top goal and bottom goal and no other rule is applied, and each $D_i \neq \epsilon$. To understand the rule assume that $D_1A_1 + \ldots + D_kA_k = B'$ is the current goal. This reduces to the subgoal beneath it provided that the top goal appears above it in the proof tree and condition $\mathcal{C}$ holds. There is also the symmetric rule BAL(R) where the premises are $B = \ldots$ and $B' = \ldots$, and the conclusion is $B' = \ldots$.

Consider the top goal of BAL(L), $A = B$. Let B have shape $\beta_1B_1 + \ldots + \beta_nB_n + C$ where $|\beta_i| = \mathrm{M}+1$. Because $(X_i \cdot w(X_i)) = \epsilon$ it follows that $(A \cdot w(X_i)) = A_i$. Therefore if the top goal is true then $L(A_i) = L(B \cdot w(X_i))$. It is this substitution of $(B \cdot w(X_i))$ for A_i for each i in the bottom goal which the rule sanctions. Moreover $(B \cdot w(X_i))$ is $(\beta_1 \cdot w(X_i))B_1 + \ldots + (\beta_n \cdot w(X_i))B_n + (C \cdot w(X_i))$ because $|w(X_i)| < |\beta_j|$. Also B' has the shape $B'_1B_1 + \ldots + B'_nB_n + C'$ (where $|C'|, |B'_i| \leq 2\mathrm{M} + 1$). Putting all this together it means that the subgoal has the following form, where some of the A'_i and B'_j may be $\emptyset$ and $B_{n+1} = \epsilon$.

$$A'_1B_1 + \ldots + A'_nB_n + C''B_{n+1} = B'_1B_1 + \ldots + B'_nB_n + C'B_{n+1}$$

We think of this subgoal as "balanced" because they have this common tail form, and all their heads have bounded size.

Introducing balanced subgoals is not sufficient for showing decidability. For the sizes of the common tails may keep growing. There is one more tableau rule, CUT, which allows one to cut the common tails. The exact formulation relies on families of auxiliary nonterminals ranged over by V, each of which has an associated definition $V \stackrel{\mathrm{def}}{=} B$. We say that $(V_1, \ldots, V_n)$ is a family of recursive nonterminals if for each i either $V_i \stackrel{\mathrm{def}}{=} A_{i1}V_1 + \ldots + A_{in}V_n$ where $A_{i1} + \ldots + A_{in}$ is admissible and does not contain auxiliary nonterminals, or $V_i \stackrel{\mathrm{def}}{=} V_j$ and $j \leq i$ and $V_j \stackrel{\mathrm{def}}{=} V_j$. An auxiliary nonterminal can only appear as a final element in a sequence of nonterminals. Admissibility is extended to such families of sequences. A configuration which is a singleton V is admissible and $\beta_1V'_1 + \ldots + \beta_kV'_k$ is admissible if the head $\beta_1 + \ldots + \beta_k$ is admissible and each β_i is distinct, and there is a family of recursive nonterminals $(V_1, \ldots, V_n)$ such that each V'_i is one of the V_js. An admissible configuration can therefore be presented in tail form

$A = A_1V_1 + \ldots + A_nV_n$. The definition of $(A \cdot w)$ is refined. If $(A_i \cdot w) = \epsilon$ and $V_i \stackrel{\text{def}}{=} B$ then $(A \cdot w) = B$. The language accepted by A is the set of words w such that $(A \cdot w) = V_i$ where $V_i \stackrel{\text{def}}{=} V_i$. Two configurations containing auxiliary nonterminals are equivalent if they accept the same words and agree on their terminating nonterminals.

The idea of CUT is that a balanced goal

$$(1) \quad A_1B_1 + \ldots + A_nB_n = C_1B_1 + \ldots + C_nB_n$$

where the A_is and C_is do not contain recursive nonterminals, can be reduced to a subgoal of the form

$$(2) \quad A_1V_1 + \ldots + A_nV_n = C_1V_1 + \ldots + C_nV_n$$

where $(V_1, \ldots, V_n)$ is a family of recursive nonterminals. The mechanism for reducing goal (1) to goal (2) involves constructing the recursive family $(V_1, \ldots, V_n)$ from a subsidary family of goals, $A_1^iB_1 + \ldots + A_n^iB_n = C_1^iB_1 + \ldots + C_n^iB_n$ where $i \geq 1$, with the same tails as (1).

We now state an important result which underpins the rule CUT.

Lemma 1 *Assume $0 < m \leq n$. If for all $i : 1 \leq i \leq m$, $L(A_1^iB_1 + \ldots + A_n^iB_n) = L(C_1^iB_1 + \ldots + C_n^iB_n)$ then there is a family of recursive nonterminals $(V_1, \ldots, V_n)$ such that*

1. *For each $i : 1 \leq i \leq m$, $L(A_1^iV_1 + \ldots + A_n^iV_n) = L(C_1^iV_1 + \ldots + C_n^iV_n)$,*
2. *If $V_j \stackrel{\text{def}}{=} A_1'V_1 + \ldots + A_n'V_n$ then $L(B_j) = L(A_1'B_1 + \ldots + A_n'B_n)$,*
3. *If $V_i \stackrel{\text{def}}{=} V_j$ then $L(B_i) = L(B_j)$.*

The recursive family $(V_1, \ldots, V_n)$ which issues from the proof of Lemma 1 is said to be "canonical" for the family $A_1^iB_1 + \ldots + A_n^iB_n = C_1^iB_1 + \ldots + C_n^iB_n$ of true goals. The construction of canonical nonterminals is independent of the tails B_i.

Fact 1 *If $(V_1, \ldots, V_n)$ is canonical for $A_1^iB_1 + \ldots + A_n^iB_n = C_1^iB_1 + \ldots + C_n^iB_n$ then it is also canonical for the family $A_1^iD_1 + \ldots + A_n^iD_n = C_1^iD_1 + \ldots + C_n^iD_n$, where $i : 1 \leq i \leq k$.*

The proof of Lemma 1 assembles the canonical family in stages. At stage j, the family $(V_1^{j+1}, \ldots, V_n^{j+1})$ is constructed from $(V_1^j, \ldots, V_n^j)$. If each V_i^{j+1} has the same definition as V_i^j then the construction terminates. In fact it must terminate by stage $j = n - 1$. The building of the V_i^{j+1}s from the V_i^js appeals to a smallest distinguishing word u_{j+1} for $L(A') \neq L(C')$, where A' is $A_1^lV_1^j + \ldots + A_n^lV_n^j$ and C' is $C_1^lV_1^j + \ldots + C_n^lV_n^j$ for some l. The depth of a canonical family is given by the sum over all stages of the distinguishing words, $\sum |u_j|$.

We need to consider how to introduce recursive nonterminals when the family of goals need not all be true. The idea is to approximate canonicity by defining when a recursive family $(V_1, \ldots, V_n)$ is "canonical to depth d" where $d \geq 0$, for a

family of goals $A_1^i B^1 + \ldots + A_n^i B_n = C_1^i B_1 + \ldots + C_n^i B_n$. The construction is the same as for the proof of Lemma 1, except that we stop at the first stage $j \geq 0$ with $(V_1^j, \ldots, V_n^j)$ as the required family of recursive nonterminals if the sum of the distinguishing words $s_j = |u_1| + \ldots + |u_j|$ is no larger than d, and for all w such that $|w| \leq d - s_j$, $w \in L(A_1^i V_1^j + \ldots + A_n^i V_n^j)$ iff is $w \in L(C_1^i V_1^j + \ldots + C_n^i V_n^j)$, for each i.

The rule CUT, where $k \leq n$, is as follows.

$$\frac{\begin{array}{l} A_1^1 B_1 + \ldots + A_n^1 B_n = C_1^1 B_1 + \ldots + C_n^1 B_n \\ \vdots \\ A_1^k B_1 + \ldots + A_n^k B_n = C_1^k B_1 + \ldots + C_n^k B_n \\ \vdots \\ A_1 B_1 + \ldots + A_n B_n = C_1 B_1 + \ldots + C_n B_n \end{array}}{A_1 V_1 + \ldots + A_n V_n = C_1 V_1 + \ldots + C_n V_n} \; \mathcal{C}$$

where $\mathcal{C}$ is the condition that $(V_1, \ldots, V_n)$ is canonical to depth d for the family of goals $A_1^i B_1 + \ldots + A_n^i B_n = C_1^i B_1 + \ldots + C_n^i B_n$, $1 \leq i \leq k$, and there are at least d applications of UNF (as well as possibly applications of BAL) between $A_1^k B_1 + \ldots + A_n^k B_n = C_1^k B_1 + \ldots + C_n^k B_n$ and the final goal in the premises $A_1 B_1 + \ldots + A_n B_n = C_1 B_1 + \ldots + C_n B_n$. CUT is essentially the strategy T_C in S'enizuergues's proof (although he uses regular expressions and not recursive nonterminals).

From Fact 1 it follows that for any other family of goals with different tails D_i but the same heads A_j^i, C_j^i the same recursive nonterminal family is introduced. It is this feature which guarantees that there is a finite tableau proof for $F \sim G$.

We have now seen all the tableau proof rules, INIT, UNF, BAL(L), BAL(R) and CUT. There is also the important notion of when a current goal counts as final. Final goals are classified as either successful or unsuccessful. A *tableau proof* for the starting goal $F = G$ is a finite proof tree, whose root is $F = G$ and all of whose leaves are successful final goals, and all of whose inner subgoals are the result of an application of one of the rules. Successful final goals are as follows:

$$\begin{array}{ll} & A = B \\ & \vdots \quad \text{UNF at least once} \\ A = A & A = B \end{array}$$

An identity and a goal which is repeated count as successful. Unsuccessful final goals are

$$\emptyset = B \text{ and } L(B) \neq \emptyset \qquad A = \emptyset \text{ and } L(A) \neq \emptyset \qquad V_i = V_j \text{ and } i \neq j$$

The tableau rules are sound and complete, which we now explain. First UNF is complete in the sense that if the premise is true then so are the subgoals.

Completeness for BAL is that if the premise goals (those above the subgoal) are true then so is the subgoal. The statement of completeness for CUT is that there are correct applications of it. If $(V_1, \ldots, V_n)$ is canonical for the first k premises then there is a depth d for which it is canonical. Moreover $(V_1, \ldots, V_n)$ needs to be a recursive family for the true goal $A_1B_1 + \ldots + A_nB_n = C_1B_1 + \ldots + B_nC_n$, in which case the subgoal follows.

For soundness of the tableau rules consider global soundness of the proof system. The overall idea is that if there is a successful tableau whose root is false then there is a path through the tableau within which each subgoal is false. The idea is refined using approximants. If $F^i \not\sim G^i$ then there is smallest distinguishing word w. One can define n-equivalence between F^i and G^i, if for all words w such that $|w| \leq n$, w does not distinguish between F^i and G^i. UNF obeys the simple soundness property that if the goal is not $n+1$-equivalent then a subgoal is not n-equivalent. Therefore if the root is false then there is an offending path (of false goals) through the tableau within which the approximant indices decrease whenever rule UNF has been applied, and hence this would mean that a successful final goal is false (which, as we shall show, is impossible). Soundness of the conditional rules is that if the premises are on an offending path then the subgoal preserves the falsity index of the goal immediately above it. In the case of BAL(R) assume that the offending path passes through the premise goals. There is a least n such that for the initial premise B is n-equivalent to $X_1A_1 + \ldots + X_kA_k$ and B is not $n+1$-equivalent to $X_1A_1 + \ldots + X_kA_k$. As there are exactly M applications of UNF between the initial and final premise it follows that B' is $(n - \mathrm{M})$-equivalent to $D_1A_1 + \ldots + D_kA_k$. However, as this is the offending path B' is not $(n + 1 - \mathrm{M})$-equivalent to $D_1A_1 + \ldots + D_kA_k$. A small argument shows that B' is not $(n + 1 - M)$-equivalent to $D_1(B \cdot w(X_1)) + \ldots + D_k(B \cdot w(X_k))$ (because A_i is $(n - \mathrm{M})$-equivalent to $(B \cdot w(X_i)$. There is a similar soundness argument for CUT. The idea of this style of soundness is essentially due to Sénizuergues (although he uses the different framework of deduction systems).

The main result is as follows, and a similar result holds for monadic resursion schemes.

Theorem 1 *$F \sim G$ iff there is a finite tableau proof for $F = G$.*

5 Conclusion

We have sketched decidability of equivalence for two old schema problems. However there are many open questions for further work. First we do not have a complexity bound for the decision procedures. Secondly we have only shown decidability for first-order recursion schemes. There is a known hierarchy of schema problems at higher order [3, 4]. The branch languages of higher-order schemes are deterministic context-sensitive languages, as illustrated by the following 2nd-order scheme

$$\Phi(G, H)(x) \stackrel{\text{def}}{=} f(\Phi(Gg, Hh)(x), G(Hx))$$

starting from $\Phi(g,h)(x)$. And so little is known about deterministic context-sensitive languages.

References

1. Courcelle, B. (1978). A representation of trees by languages I, *Theoretical Computer Science*, **6**, 255-279.
2. Courcelle, B. (1978). A representation of trees by languages II, *Theoretical Computer Science*, **7**, 25-55.
3. Damm W, (1977). Languages defined by higher type program schemes, *Lecture Notes in Computer Science*, **52**, 164-179.
4. Damm W, (1979). An algebraic extension of the Chomsky-hierarchy. *Lecture Notes in Computer Science*, **74**, 266-276.
5. Friedman, E. (1977). Equivalence problems for deterministic context-free languages and monadic recursion schemes. *Journal of Computer and System Sciences*, **14**, 344-359.
6. Garland, S., and Luckham, D. (1973). Program schemes, recursion schemes, and formal languages. *Journal of Computer and System Sciences*, **7**, 119-160.
7. Harrison, M. (1978). *Introduction to Formal Language Theory*, Addison-Wesley.
8. Harrison, M., and Havel, I. (1973). Strict deterministic grammars. *Journal of Computer and System Sciences*, **7**, 237-277.
9. Hüttel, H., and Stirling, C. (1991). Actions speak louder than words: proving bisimilarity for context free processes. *Proceedings 6th Annual Symposium on Logic in Computer Science*,IEEE Computer Science Press, 376-386.
10. Sénizergues, G. (1997). The equivalence problem for deterministic pushdown automata is decidable. *Lecture Notes in Computer Science*, **1256**, 671-681.
11. Sénizergues, G. (1998). L(A) = L(B)? Tech. Report LaBRI, Université Bordeaux I, pp. 1-166. (To appear in *Theoretical Computer Science*.)
12. Stirling, C. (1998). Decidability of bisimulation equivalence for normed pushdown processes. *Theoretical Computer Science*, **195**, 113-131.
13. Stirling, C. (1999). Decidability of DPDA equivalence. Tech. Report LFCS-99-411, University of Edinburgh, pp. 1-25. (To appear in *Theoretical Computer Science*.)