The tree equivalence of linear recursion schemes

V. Sabelfeld
Karlsruhe University, P.O. Box 6980
76128 Karlsruhe, Germany
E-mail: *sabelfel@ira.uka.de*

Abstract

In the paper, a complete system of transformation rules preserving the tree equivalence and a polynomial-time algorithm deciding the tree equivalence of linear polyadic recursion schemes are proposed. The algorithm is formulated as a sequential transformation process which brings together the schemes in question. In the last step, the tree equivalence problem for the given schemes is reduced to a global flow analysis problem which is solved by an efficient marking algorithm.

Keywords: recursion schemes, tree equivalence, transformation rules.

1 Introduction

A recursion scheme is a system of recursive function definitions. This model of recursive programs was first introduced and investigated in [1, 7]. The tree equivalence of recursion schemes was introduced by B. Rosen [9], who pointed out some subclasses of recursion schemes for which the tree equivalence is decidable. Two schemes are *tree equivalent* if their determinants are equal. Informally, the *determinant* $det(S)$ of a scheme S can be obtained from S in two steps. First, we "build" the (possible infinite) unfolding tree of the scheme by sequential unfolding of function calls using the "parallel outermost computation rule". The unfolding tree of the example scheme $S = \langle F(h)\ ; F(x) \Leftarrow f(x, F(gx))\rangle$ is the limit of the term sequence $\omega, f(h,\omega), f(h, f(gh,\omega)), \ldots$ where ω denotes the distinguished constant with the value "undefined". Secondly, the determinant is obtained from the unfolding tree by replacing all subtrees having an undefined value in all interpretations by the constant ω.

The result of this step depends on what we mean by "all interpretations" of basic symbols. If we restrict the interpretations $I(f)$ of the symbol f in our example scheme S by the requirement $\forall d.I(f)(d,\perp) = \perp$, where $\perp$ means "undefined", then $det(S) = \omega$. If interpretations with $\exists d.I(f)(d,\perp) \neq \perp$ are allowed then $det(S)$ coincides with the unfolding tree.

We introduce and investigate a class of recursion schemes which is larger than the class of recursion schemes used in the original definition [7] where basic symbols are interpreted only by total functions. In the recursion schemes considered by B. Rosen [9] all interpretations of a basic symbol f must satisfy the condition $I(f)(\bot, \bot, \ldots, \bot) = \bot$ (i.e. the result is undefined if all arguments are undefined). The restrictions on interpretations of basic symbols in schemes studied in [11] can be described as $I(\mathbf{if})(a, b, c) = \bot$ if $a = \bot \vee b = \bot \wedge c = \bot$ (the result of a test is undefined if the condition or both alternatives are undefined), and $I(f)(d_1, \ldots, d_n) = \bot$ if $\exists i. d_i = \bot$ (the result is undefined if at least one argument is undefined) for all basic symbols f different from **if**. For our recursion schemes much finer restrictions on interpretations of a basic symbol f can be formulated by fixing an arbitrary set $Strict(f)$ of *strict parameter collections*. All interpretations $I(f)$ of a symbol f have to satisfy the following *strictness* condition:

$$\forall d_1, \ldots, d_n. (\forall \Delta \in Strict(f).\ \exists i \in \Delta.\ d_i = \bot) \Rightarrow I(f)(d_1, \ldots, d_n) = \bot.$$

In addition, we cancel all syntactical restrictions to conditions in tests.

We define a class of *linear* recursion schemes, characterised by the property that actual parameters in calls of such schemes contain neither calls nor the constant ω as subterms. The decidability of the tree equivalence for linear schemes follows from the results of [13, 2] which present an algorithm with an upper time bound 2^{2^n} for deciding the equivalence of finite-turn DPDA's. Since the tree equivalence problem for linear recursion schemes can be polynomially reduced to the equivalence problem for one-turn DPDA's [3, 4, 6], the tree equivalence for linear recursion schemes is decidable with the same triple exponential time upper bound.

We describe a direct algorithm that decides the tree equivalence of linear recursion schemes in polynomial time $O(n^6)$, where n is the maximum of the initial scheme sizes. The main ideas of the algorithm are as follows: Firstly, the algorithm is formulated as a sequential transformation process which brings together the schemes in question. The algorithm passes through several control points, in which some conditions are checked (similarity test, key condition checking); if one of the tests fails, the transformation process will terminate with the answer "no" to the question of the equivalence. Secondly, after a number of scheme reducing transformations, we construct the product schemes $S_1 \times S_2$ and $S_2 \times S_1$ with an adjusted structure by means of rule applications. Then, again by means of rule applications, one of the scheme products is transformed into a scheme which represents the computations of both schemes. Finally, the tree equivalence problem for the given schemes is reduced to a global flow analysis problem which is solved by an efficient marking algorithm.

In addition to the algorithm deciding the tree equivalence of linear recursion schemes we also construct a complete transformation system Σ_{lin} for the tree equivalence in this class of schemes. Finally, we extend the results to quasi-

linear recursion schemes, where the tree equivalence remains decidable but the algorithm becomes more complicated, and not polynomial.

A partial solution to the stated problems was achieved in [10], but for an essentially smaller class of recursion schemes, where basic symbols are interpreted only by total functions and all tests are atomic.

2 Recursion scheme definition

Let $\mathcal{X} = \{x, y, z, \ldots\}$ be a set of *variables*, $\mathcal{F}_b = \{\omega, f, g, h, \ldots\}$ be a set of *basic symbols*, $\mathcal{F}_d = \{F, F_1, F_2, \ldots\}$ be a set of *defined symbols*, $\mathcal{F} = \mathcal{F}_b \bigcup \mathcal{F}_d$, and let r denote the *rank* function. Every symbol $s \in \mathcal{F}$ of rank $r(s) = n$ is said to have arity n. The symbol ω stands for "undefined", the least informative term; $r(\omega) = 0$. Basic symbols of arity zero are also called *constants*. Let each basic symbol f be associated with some subset $Strict(f)$ of the set $2^{\{1,\ldots,r(f)\}}$. The set $Strict(f)$ is called the set of *strict parameter collections* of the symbol f and restricts the set of all possible interpretations of the symbol f in the following way: in every interpretation, the term $f(d_1, \ldots, d_{r(f)})$ is undefined if for all $\Delta \in Strict(f)$ there exists i in Δ such that the value d_i is undefined.

For a set X of variables, $X \subset \mathcal{X}$, let $\mathcal{T}(X)$ be the minimal set of *terms* closed under the following conditions:

- $X \subset \mathcal{T}(X)$
- $s \in \mathcal{F},\ t_1, \ldots, t_n \in \mathcal{T}(X), n = r(s) \Rightarrow s(t_1, \ldots, t_n) \in \mathcal{T}(X)$.

The first symbol s of a term $s(t_1, \ldots, t_n)$ is called the *main symbol* of this term. A *recursion scheme* is a pair

$$S = \langle e; DEF \rangle,$$

where e is a term called the *scheme entry* and DEF is a finite set of definitions of symbols in $\mathcal{F}_d$. A *definition* of a symbol $F \in \mathcal{F}_d$ has the form

$$F(x_1, \ldots, x_n) \Leftarrow t,$$

where $n = r(F),\ x_1, \ldots, x_n \in \mathcal{X}$ are different *formal parameters* of F and $t \in \mathcal{T}(x_1, \ldots, x_n)$ is the *body* of the symbol F.

A symbol $F \in \mathcal{F}_d$ is *internal* to the scheme S if S contains the (unique) definition of F; F is *external* to the scheme S if F occurs in S but S does not contain any definition of F. Denote the sets of all internal and external symbols of a scheme S by $Inner(S)$ and $Outer(S)$, respectively. Note that every internal symbol has a *unique* definition since DEF is a set.

The notions of internal and external symbols will be used only in Section 4, where the notion of a fragment will be introduced. A fragment can be viewed as a generalised scheme. Here we require $Outer(S) = \emptyset$ for any scheme S, i.e.

schemes have no external symbols. However, all the definitions for schemes take the case $Outer(S) \neq \emptyset$ into account since they will be also used for fragments.

A term $F(t_1, \ldots, t_n)$, where $n = r(F)$ and $F \in \mathcal{F}_d$ is said to be a *call* to the symbol F, and its subterms $t_1, \ldots, t_n$ are *actual parameters* of this call. A call is an *internal* call if $F \in Inner(S)$, or an *external* one if $F \in Outer(S)$.

Example:

$$S_0 = \left\langle F_1(h,h) \; ; \; \begin{array}{ll} F_1(x,y) & \Leftarrow \mathbf{if}(px, F_2(fx,fy), x) \\ F_2(x,y) & \Leftarrow \mathbf{if}(py, F_1(fy,fx), y) \\ F_3(x) & \Leftarrow \mathbf{if}(px, F_3(fx), x) \\ Strict(f) = Strict(p) = \{\{1\}\}, & \\ Strict(\mathbf{if}) = \{\{1,2\},\{1,3\}\}, Strict(h) = \{\emptyset\} & \end{array} \right\rangle$$

We assume that the set $\mathcal{F}_b$ of basic symbols contains the constant ω, for which the condition $Strict(\omega) = \emptyset$ is satisfied, and for all other symbols $f \in \mathcal{F}_b, f \neq \omega$, we require $Strict(f) \neq \emptyset$. This means that we have a unique designator for a constant which can only be interpreted as "undefined". Note that $Strict(h)$ is not empty in the example scheme S_0 since it contains the empty parameter collection $\emptyset$.

Any occurrence of a subterm in a term or in a scheme can be uniquely identified by its *address*, which represents the path from the root of the term-tree (or from the scheme entry) to the place of the subterm occurrence. The address of the tree root (or the scheme entry) is the empty word Λ (if the tree is unique in the context under consideration), or the entry number in square brackets. The address of the body of a symbol F is the word $[F]$. If an occurrence of a term $f(t_1, \ldots, t_n)$ has an address a, then its subterm occurrences $t_1, \ldots, t_n$ have the addresses $a.1, \ldots, a.n$, respectively. For example, the first occurrence of the term fx in the scheme S_0 has the address $[F_1].2.1$, and the first occurrence of the constant h has the address 1. Two addresses are *independent* iff neither is a prefix of the other.

A *substitution* is an arbitrary map $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{X})$ satisfying the condition $\sigma x \neq x$ for only a finite number of variables x from $\mathcal{X}$. The substitution σ mapping the variable x_i on a term t_i for $i = 1, \ldots, n$ is denoted by $[t_1/x_1, \ldots, t_n/x_n]$. The notion of a substitution can be extended in a natural way to arbitrary terms:

$$\sigma s(t_1, \ldots, t_n) = s(\sigma t_1, \ldots, \sigma t_n)$$

for $s \in \mathcal{F}$ and $n = r(s)$. For example, if $\sigma = [g(y)/x, h(x)/y]$, then $\sigma f(x,y) = f(g(y), h(x))$. We denote by $t[a \leftarrow \tau]$ the term obtained from the term t by replacing the term occurrence at the address a in t by the term τ. If N is a set of mutually independent addresses of subterm occurrences of a term t then $t[N \leftarrow \tau]$ denotes the term obtained from the term t by replacing all subterm occurrences at addresses from N by the term τ.

Each scheme S defines a map

$$\pi : \mathcal{T}(\mathcal{X}) \to \mathcal{T}(\mathcal{X})$$

which corresponds to the "parallel outermost computation rule", i.e. it performs one step unfolding of all outermost calls in a term.

$$\pi t = \begin{cases} x, & \text{if } t = x, \text{ where } x \in \mathcal{X}, \\ f(\pi t_1, \ldots, \pi t_n), & \text{if } t = f(t_1, \ldots, t_n), \text{ where } n = r(f) \\ & \text{and } f \in \mathcal{F}_b \cup \ Outer(S), \\ \sigma\tau, & \text{if } t = F(t_1, \ldots, t_n), \text{where } F \in Inner(S), \\ & \sigma = [\ t_1/x_1, \ldots, t_n/x_n\ ], \text{ and} \\ & F(x_1, \ldots, x_n) \Leftarrow \tau \text{ is the definition of } F \text{ in } S. \end{cases}$$

3 Interpretation and the tree equivalence

A *domain* is a triple $\langle D, \leq, \bot \rangle$, where D is an arbitrary set, $\bot$ is the bottom member of D (the "undefined value") and $\leq$ is a partial order on D satisfying the following two conditions:

- $\forall d \in D. \quad \bot \leq d.$
- *Completeness condition:* for every linearly ordered subset (chain) C of D the *least upper bound* $lub(C)$ belongs to D, i.e. $d \leq lub(C)$ for all d in C, and $lub(C) \leq u$ for all u in D such that $d \leq u$ for all d in C.

For example, the *boolean domain* $\mathcal{B}$ consists of three elements

$$\mathcal{B} = \{\bot, \mathbf{false}, \mathbf{true}\}$$

such that $\bot \leq \mathbf{false}$, $\bot \leq \mathbf{true}$, but $\neg(\mathbf{true} \leq \mathbf{false})$ and $\neg(\mathbf{false} \leq \mathbf{true})$.

The *universal term domain* $\mathcal{U} = \langle T_U, \sqsubseteq, \omega \rangle$ is constructed in the following way: we consider the set T_1 of the terms under the signature $\langle \mathcal{X}, \mathcal{F}_b \rangle$ which contains all variables $x \in \mathcal{X}$ and is closed under application of the rule

$$f \in \mathcal{F}_b \wedge t_1, \ldots, t_n \in T_1 \wedge \exists \Delta \in Strict(f). \forall i \in \Delta.\ t_i \neq \omega \Rightarrow f(t_1, \ldots, t_n) \in T_1.$$

The partial order $\sqsubseteq$ on T_1 is introduced by setting $t_1 \sqsubseteq t_2$ iff there exists a set N of mutually independent addresses in t_2, such that $t_1 = t_2[N \leftarrow \omega]$.

Let $\tilde{T}_1$ be the set of all infinite chains of the form $t_1 \sqsubseteq t_2 \sqsubseteq \ldots$ with elements from T_1. There is a unique (infinite) tree $\underline{t}$ corresponding to such a chain $\tilde{t}$. Finally, $T_U = \tilde{T}_1 / \cong_t$ is the factor set of the set $\tilde{T}_1$ under the equivalence relation $\cong_t$ defined by $\tilde{s} \cong_t \tilde{s'} \Leftrightarrow \underline{s} = \underline{s'}$. The elements of the set T_U will be called (infinite) terms (trees). One can prove that $\mathcal{U}$ is a domain with the partial order '$\sqsubseteq$' ($t_1 \sqsubseteq t_2$ iff there is a (possibly infinite) set N of mutually independent addresses in t_2 such that $t_1 = t_2[N \leftarrow \omega]$) and bottom ω.

A function $\varphi : D \to D'$ between two domains is *monotone* iff

$$\forall d, d' \in D.\ d \leq d' \Rightarrow \varphi d \leq \varphi d'.$$

A monotone function is *continuous* if it preserves the least upper bounds of non-empty linearly ordered subsets L of D, i.e. $\varphi(lub(L)) = lub(\varphi(L))$.

An *interpretation* I fixes a domain D and assigns

- a member $I(x) \in D$ to each variable $x \in \mathcal{X}$,
- a continuous function $I(f) : D^{r(f)} \to D$ to each symbol $f \in \mathcal{F}_b$. This function must satisfy the following *strictness* condition:

$$\forall d_1, \ldots, d_n.(\forall\ \Delta \in \mathit{Strict}(f).\ \exists i \in \Delta.\ d_i = \bot) \Rightarrow I(f)(d_1, \ldots, d_n) = \bot.$$

For example, the natural way to bound the interpretations of the ternary symbol **if** is to settle

$$\mathit{Strict}(\mathbf{if}) = \{\{1,2\},\{1,3\}\}.$$

Thereby we restrict all possible interpretations of the symbol **if** to functions *cond* for which the condition

$$\forall\ d, d' \in D.\ (\mathit{cond}(\bot, d, d') = \bot)\ \wedge\ (\mathit{cond}(d, \bot, \bot) = \bot)$$

holds. Another example is the binary symbol '+', $\mathit{Strict}(+) = \{\{1,2\}\}$ with

$$\forall d \in D.\ \forall I.\ (I(+)(\bot, d) = \bot)\ \wedge\ (I(+)(d, \bot) = \bot).$$

Since $\mathit{Strict}(\omega) = \emptyset$, the constant ω can only be interpreted by the value $\bot$.

By an induction definition we can extend the notion of an interpretation I to arbitrary finite terms:

$$I^*(t) = \begin{cases} I(x), & \text{if } t = x, \text{ where } x \in \mathcal{X}; \\ I(f)(I^*(t_1), \ldots, I^*(t_n)), & \text{if } t = f(t_1, \ldots, t_n), \text{ where } n = r(f) \\ & \text{and } f \in \mathcal{F}_b; \\ \bot & \text{otherwise.} \end{cases}$$

For infinite terms $t = lub\{\ t_n : n \geq 0\ \}$ we set

$$I^*(t) = lub\{\ I^*(t_n) : n \geq 0\ \}$$

because $I(f)$ is continuous for each $f \in \mathcal{F}_b$.

An important example of an interpretation is the *universal interpretation* J with the domain $\mathcal{U}$, and $J(x) = x$ for variables $x \in \mathcal{X}$; for $f \in \mathcal{F}_b, n = r(f)$ and $t_1, \ldots, t_n \in T_U$ we set

$$J(f)(t_1, \ldots, t_n) = \begin{cases} f(t_1, \ldots, t_n), & \text{if } \exists\ \Delta \in \mathit{Strict}(f) \\ & \forall\ i \in \Delta\ t_i \neq \omega, \\ \omega & \text{otherwise.} \end{cases}$$

The universal interpretation is extended to calls $F(\bar{t})$ by setting $J(F(\bar{t})) = \omega$, and we define the *approximation sequence* of a term t by

$$\mathit{App}(S, t) = \{J(\pi^n t) \mid n \geq 0\}$$

where $\pi^0 = id$ and $\pi^n = \pi^{n-1}\pi$ for $n > 0$. The *determinant* $det(t)$ *of a term* t is the least upper bound of the approximation sequence $App(S,t)$, and the *determinant* $det(S)$ *of a scheme* S is the determinant of the entry of the scheme S.

For the scheme

$$\langle \begin{array}{l} F(u); F(x) \Leftarrow f(x, F(gx)), \\ Strict(f) = \{\{1\},\{2\}\}, Strict(g) = \{\{1\}\} \end{array} \rangle$$

the first elements of the approximation sequence are

$$\begin{array}{c} J(\pi^0 F(u)) = \omega, \\ J(\pi^1 F(u)) = f(u, \omega), \\ J(\pi^2 F(u)) = f(u, f(gu, \omega)), \\ J(\pi^3 F(u)) = f(u, f(gu, f(ggu, \omega))), \text{ etc.} \end{array}$$

Two schemes S_1 and S_2 are *tree equivalent* (for short: $S_1 \approx S_2$, see also [9]) iff $det(S_1) = det(S_2)$.

4 Fragments and their equivalence

The notion of a *fragment* differs from the notion of a scheme only in that a fragment may have an arbitrary, possibly empty, set of entries, and the restriction $Outer(S) = \emptyset$ is relaxed. Assume the entries of a fragment to be enumerated by non-negative integers. The entry number enclosed in square brackets will be used for its address. We omit this number if the fragment has a single entry. We also omit the angle brackets in the representation of a fragment if it does not contain definitions and has a single entry. In this case the fragment simply is reduced to a term.

Let $Entries(\mathcal{G})$ be the set of numbers for all entries of a fragment $\mathcal{G}$. In order to extend the notion of the tree equivalence to fragments $\mathcal{G}_1$ and $\mathcal{G}_2$ which have equal entry number sets we consider the symbols from the set $\mathcal{O} = Outer(G_1) \cup Outer(G_2)$ in exactly the same way as the symbols from $\mathcal{F}_b$, setting, in particular, $Strict(F) = \{\emptyset\}$ for $F \in \mathcal{O}$.

Two fragments $\mathcal{G}_1$ and $\mathcal{G}_2$ that have equal entry number sets are tree equivalent iff for all $i \in Entries(\mathcal{G}_1)$ the schemes $\mathcal{G}_1^i$ and $\mathcal{G}_2^i$ are tree equivalent which are obtained from $\mathcal{G}_1$ and $\mathcal{G}_2$ by

- deleting all entries except for the entries with the number i, and
- including the symbols from $Outer(\mathcal{G}_1) \cup Outer(\mathcal{G}_2)$ into the basic symbol set,

Note that two fragments with empty entry sets are always tree equivalent.

Let us denote by $\mathcal{G} \downarrow a$ the term at address a in a fragment or term $\mathcal{G}$, by $Var(a)$ — the variable set of the term at address a, and by $\mathcal{G}[a \leftarrow t]$ — the fragment obtained from $\mathcal{G}$ by replacing the term at address a by a term t.

Let $\mathcal{G} = \langle 1 : e_1, \ldots, n : e_n; DEF \rangle$ be a fragment. Two terms α, β are called *tree equivalent in* $\mathcal{G}$ (for short: $\alpha \approx_{\mathcal{G}} \beta$ or simply $\alpha \approx \beta$ if $\mathcal{G}$ is clear from the context), iff $\langle \alpha; DEF \rangle \approx \langle \beta; DEF \rangle$. The term t is called *tree empty* in $\mathcal{G}$ iff $t \approx \omega$.

5 The definition of linear schemes and formal transformations

Let TB be the set of all terms which do not contain calls and the constant ω as subterms. A *linear term* is a term which contains only terms from TB as actual parameters in calls. More precisely, the set TL of linear terms is defined as the minimal set of terms closed under the following conditions:

- $TB \subset TL$.
- $\tau_1, \ldots, \tau_n \in TL,\ f \in \mathcal{F}_b, n = r(f) \Rightarrow f(\tau_1, \ldots, \tau_n) \in TL$.
- $\tau_1, \ldots, \tau_n \in TB,\ F \in \mathcal{F}_d, n = r(F) \Rightarrow F(\tau_1, \ldots, \tau_n) \in TL$.

A symbol $F \in \mathcal{F}_d$ of a scheme S is linear if its body is a linear term. A scheme is *linear* if its entry and all internal symbols are linear.

Example 1: if $Strict(f) = \{\{1\}, \{2\}, \{3\}, \{4\}\}$ then

$$R_1 = \langle F_1(a, a);\ F_1(x, y) \Leftarrow f(x, y, gb, F_1(gx, gy)) \rangle \text{, and}$$

$$R_2 = \left\langle \begin{array}{ll} & F_1(u, v) \Leftarrow f(u, u, F_2(gb), F_3(gu, hv)) \\ F_1(a, c); & F_2(u) \Leftarrow u \\ & F_3(u, v) \Leftarrow f(u, u, F_2(gb), F_1(gu, ggv)) \end{array} \right\rangle$$

are two tree equivalent linear recursive schemes.

Example 2: Let $n \geq 1, Strict(f) = Strict(g) = Strict(h) = \{\{1\}, \{2\}\}$;

$$L_n = \left\langle \begin{array}{l} F_1(a); \\ F_1(x) \Leftarrow h(F_2(x), F_{n+3}(x)) \\ F_i(x) \Leftarrow F_{i+1}(f(x, x)),\ i = 2, \ldots, n+1 \\ F_{n+2}(x) \Leftarrow x \\ F_i(x) \Leftarrow g(F_{i+1}(x), F_{i+1}(x)),\ i = n+3, \ldots, 2n+1 \\ F_{2n+2}(x) \Leftarrow g(F_{n+2}(x), F_{n+2}(x)) \end{array} \right\rangle$$

and

$$U_n = \left\langle \begin{array}{l} D_1(a); \\ D_1(y) \Leftarrow h(D_{n+3}(y), D_2(y)) \\ D_j(y) \Leftarrow D_{j+1}(g(y,y)),\ j = 2, \ldots, n+1 \\ D_{n+2}(y) \Leftarrow y \\ D_j(y) \Leftarrow f(D_{j+1}(y), D_{j+1}(y)),\ j = n+3, \ldots, 2n+1 \\ D_{2n+2}(y) \Leftarrow f(D_{n+2}(y), D_{n+2}(y)) \end{array} \right\rangle$$

are also two tree equivalent linear recursive schemes. These two examples will be used to illustrate the described algorithm for deciding the tree equivalence of linear schemes. The second example is more representative and demonstrates the main difficulty of the problem: there are two essentially different ways for computation of long terms in equivalent linear schemes — the "top down" way and the "bottom up" way.

An *occurrence* of a fragment $\mathcal{G}_1$ in a fragment $\mathcal{G}_2$ is a part of $\mathcal{G}_2$ such that

- $\mathcal{G}_1$ is a fragment itself, and
- it contains all calls to F in $\mathcal{G}_2$ if $F \in Inner(\mathcal{G}_1)$.

A *transformation rule* is a pair $\mathcal{G}_1 \leftrightarrow \mathcal{G}_2$ of fragments $\mathcal{G}_1, \mathcal{G}_2$ that satisfies the conditions

$$Entries(\mathcal{G}_1) = Entries(\mathcal{G}_2) \text{ and } (i \neq j \Rightarrow Inner(\mathcal{G}_i) \cap Outer(\mathcal{G}_j) = \emptyset).$$

An *application* of a transformation rule $\mathcal{G}_1 \leftrightarrow \mathcal{G}_2$ consists of replacing an occurrence of $\mathcal{G}_1$ by $\mathcal{G}_2$ or replacing an occurrence of $\mathcal{G}_2$ by $\mathcal{G}_1$ where each entry is replaced by the entry with the same number. A *rule scheme* is a description of an arbitrary decidable set of transformation rules. A *rule scheme application* consists in the application of some rule from this set.

Example: Applying the rule

$$\left\langle \begin{array}{l} F_1(u,u), F_1(v,v); \\ F_1(x,y) \Leftarrow \mathbf{if}(x, y, F_1(fx, fy)) \end{array} \right\rangle \leftrightarrow \left\langle \begin{array}{l} F_2(u), F_2(v); \\ F_2(x) \Leftarrow \mathbf{if}(x, x, F_2(fx)) \end{array} \right\rangle$$

to the fragment

$$\langle g(F_1(u,u), F_1(v,v)); F_1(x,y) \Leftarrow \mathbf{if}(x, y, F_1(fx, fy)) \rangle$$

we may get

$$\langle g(F_2(u), F_2(v)); F_2(x) \Leftarrow \mathbf{if}(x, x, F_2(fx)) \rangle.$$

6 The transformation system Σ_{lin}

In Sections 6.1 – 6.7, we describe seven rule schemes for equivalent transformation of linear schemes.

The *relevant address set* of a fragment $\mathcal{G}$ is the minimal set of addresses closed under the following conditions:

- The entry addresses are relevant.
- If $\mathcal{G} \downarrow a = f(t_1, \ldots, t_n)$ for a relevant address a where $f \in \mathcal{F}_b \cup Outer(S)$ then $a.i$ is a relevant address for each $i = 1, \ldots, n$.
- If $\mathcal{G} \downarrow a = F(t_1, \ldots, t_n)$ for a relevant address a where $F \in Inner(\mathcal{G})$, then $[F]$ is a relevant address. In addition, if some address of an occurrence of the i-th formal parameter of F in the body of F is relevant, then $a.i$ is a relevant address.

All other addresses of the fragment $\mathcal{G}$ are irrelevant.

6.1 Deletion/introducing of useless definitions

We call a definition of a symbol F *useless* in a fragment $\mathcal{G}$ if no call to F has a relevant address. The rule scheme *delete/introduce useless definitions* contains all rules $\mathcal{G}_1 \leftrightarrow \mathcal{G}_2$, where the fragment $\mathcal{G}_2$ is obtained from $\mathcal{G}_1$ by deleting a useless definition of a symbol F and by replacing all remaining calls to F by (an arbitrary) constant $C \in \mathcal{F}_b$.

6.2 Replacement of irrelevant term occurrences

The second rule scheme, *replace irrelevant term occurrences*, contains all rules $\mathcal{G} \leftrightarrow \mathcal{G}[a \leftarrow t]$ where a is an irrelevant address of the fragment $\mathcal{G}$, and t is an arbitrary term, $t \in \mathcal{T}(Var(a))$.

Example: In the fragment

$$\langle F(u,v);\, F(x,y) \Leftarrow \mathbf{if}(px, x, F(fx, fy)) \rangle$$

the addresses 2, $[F].3.2$ and $[F].3.2.1$ are irrelevant, hence the transformation

$$\left\langle \begin{array}{l} F(u,v); \\ F(x,y) \Leftarrow \mathbf{if}(px, x, F(fx, fy)) \end{array} \right\rangle \leftrightarrow \left\langle \begin{array}{l} F(u,v); \\ F(x,y) \Leftarrow \mathbf{if}(px, x, F(fx, b)) \end{array} \right\rangle$$

is an application of the *replace irrelevant term occurrences* rule.

6.3 Deletion/adding of redundant parameters

A formal parameter of a symbol $F \in Inner(\mathcal{G})$ is called *redundant* if the body of F does not contain any occurrence of this parameter. Let $\mathcal{G}$ be a fragment containing exactly two definitions

$$F(x_1, \ldots, x_i, \ldots, x_n) \Leftarrow t, \text{ and}$$
$$F'(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \Leftarrow t$$

where x_i is a redundant parameter of F. The rule scheme *delete/add redundant parameters* contains all rules $\mathcal{G} \leftrightarrow \mathcal{G}'$, where $\mathcal{G}'$ is obtained from $\mathcal{G}$ by replacing some calls $F(t_1, \ldots, t_i, \ldots, t_n)$ in $\mathcal{G}$ (e.g., in the bodies of F and F') by corresponding calls $F'(t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n)$.

Example:

$$\left\langle \begin{array}{l} h(x, F(u, x)) \ ; \\ F(x, y) \Leftarrow \mathbf{if}(px, x, F(fx, gx)) \\ F'(x) \Leftarrow \mathbf{if}(px, x, F(fx, gx)) \end{array} \right\rangle \leftrightarrow \left\langle \begin{array}{l} h(x, F'(u)) \ ; \\ F(x, y) \Leftarrow \mathbf{if}(px, x, F'(fx)) \\ F'(x) \Leftarrow \mathbf{if}(px, x, F'(fx)) \end{array} \right\rangle$$

6.4 Simple folding and unfolding

Let $\mathcal{G}$ be a fragment containing exactly one definition $F(x_1, \ldots, x_n) \Leftarrow t$, and let a be an address of an occurrence of a call $F(t_1, \ldots, t_n)$ not in the body of F. The rule scheme *simple fold/unfold* contains all rules $\mathcal{G} \leftrightarrow \mathcal{G}[a \leftarrow \sigma t]$, where $\sigma = [t_1/x_1, \ldots, t_n/x_n]$.

6.5 Copying and identifying

Let $\mathcal{K} = \mathcal{K}_1 \cup \mathcal{K}_2 \cup \ldots \cup \mathcal{K}_m$ be a partition of the set of all internal symbols of the fragment $\mathcal{G}$ in non-empty and disjoint classes. We call the symbols belonging to the same class of the partition $\mathcal{K}$-*similar*. Assume that any two $\mathcal{K}$-*similar* internal symbols F, F' in the fragment $\mathcal{G}$ have definitions $F(x_1, \ldots, x_n) \Leftarrow t$ and $F'(y_1, \ldots, y_n) \Leftarrow t'$, respectively, such that the term t' can be obtained from the term $[y_1/x_1, \ldots, y_n/x_n]\, t$ by replacing some occurrences of internal symbols by $\mathcal{K}$-similar ones. Then the rule scheme *copy/identify* contains all rules $\mathcal{G} \leftrightarrow \mathcal{G}'$ where the fragment $\mathcal{G}'$ is obtained from the fragment $\mathcal{G}$ by replacing some occurrences of internal symbols by $\mathcal{K}$-similar ones.

Example: $\mathcal{K}_1 = \{F_1, F_2\}$,

$$\left\langle \begin{array}{l} F_1(a); \\ F_1(x) \Leftarrow f(x, F_2(gx)) \\ F_2(x) \Leftarrow f(x, F_1(gx)) \end{array} \right\rangle \leftrightarrow \left\langle \begin{array}{l} F_1(a); \\ F_1(x) \Leftarrow f(x, F_1(gx)) \\ F_2(x) \Leftarrow f(x, F_1(gx)) \end{array} \right\rangle$$

The definition of the symbol F_2 is uselesss in the second fragment and can be deleted.

6.6 Replacements for hopeless terms

The *exit set* of a fragment $\mathcal{G}$ of a linear scheme is defined as the minimal set of addresses closed under the following "marking rules":

1. The addresses of variable occurrences are exits.
2. If $\mathcal{G} \downarrow a = f(t_1, \ldots, t_n)$ where $f \in \mathcal{F}_b$, and $\exists\ \Delta \in \mathit{Strict}(f). \forall\ i \in \Delta$ the addresses $a.i$ are exits then the address a is an exit.

3. If $F \in \mathcal{F}_d$ and the address $[F]$ is an exit then all addresses of calls to the symbol F are exits.

Each address which is not an exit is called *hopeless*. The rule scheme *replace a hopeless term* contains all rules $\mathcal{G} \leftrightarrow \mathcal{G}[a \leftarrow \omega]$, where a is an arbitrary hopeless address in $\mathcal{G}$.

Example: Let $Strict(f) = \{\{2\}\}$. Then

$$\left\langle \begin{array}{l} F_1(u); \\ F_1(x) \Leftarrow f(x, F_2(x)) \\ F_2(x) \Leftarrow f(x, F_1(x)) \end{array} \right\rangle \leftrightarrow \left\langle \begin{array}{l} \omega; \\ F_1(x) \Leftarrow f(x, F_2(x)) \\ F_2(x) \Leftarrow f(x, F_1(x)) \end{array} \right\rangle$$

is a replace hopeless term rule.

6.7 Context replacement

If the value of an actual parameter in each call to a symbol F coincides with the value of a term t, then any occurrence of the corresponding formal parameter in the body of F can be replaced by the term t. For example, in the scheme

$$\langle F(a,a); F(x,y) \Leftarrow f(x, F(gx, gy)) \rangle$$

the formal parameters x, y are equal in all calls to the symbol F, so we could replace y by x in the body of F and obtain a tree equivalent scheme:

$$\langle F(a,a); F(x,y) \Leftarrow f(x, F(gx, gx)) \rangle.$$

Such *functional dependencies* of formal parameters (like $x \equiv y$ in example above) can be detected for linear schemes algorithmically. In this Section, we will describe an efficient algorithm for detecting functional parameter dependencies in linear schemes and use it in the *context replacement* transformation rule.

We formulate a data flow analysis problem for a graph $Graph(S)$ obtained from a scheme S in the following way: The nodes of this graph will be the entry address and the addresses of bodies of the internal symbols of the scheme S. The edges will be the call addresses. We draw an edge a from an address b to an address $[F]$, if a is an address of a call to F, occurring in the term at address b.

We analise the functional dependencies of formal parameters of the definitions and use formal grammars to represent such functional dependencies. Let X be a finite set of variables, $X \subset \mathcal{X}$. Below we use the semi-lattice $\mathcal{L}(X)$ of very simple context-free grammars G describing finite languages $L(G)$ of term equalities. These grammars G have terminal sets $\Sigma = \mathcal{F}_b \cup X \cup \{\equiv, (,), ,\}$ and

- exactly two rules $S \to x \equiv A$ and $A \to x$ for each $x \in X$, where S, A are nonterminals, $S \neq A$, and S is the initial nonterminal of the grammar;

- at most one rule of the form $A \to f(A_1, \ldots, A_n)$ for each nonterminal A, with nonterminals $A, A_1, \ldots, A_n$ different from S;
- finite languages $L(G)$.

Note that any two terms $t, t' \in L(G)$ derivable in the grammar G from a nonterminal $A(\neq S)$ are unifiable, i.e. there exists a substitution $\sigma = [t_1/x_1, \ldots, t_n/x_n]$ that satisfies $\sigma t = \sigma t'$. Such a grammar G can be reduced in linear time to a *reduced* grammar

$$Red(G) = \langle N, \Sigma, S, P \rangle,$$

satisfying the following conditions

- $L(G) = L(Red(G))$ is a finite language,
- $\forall A, B \in N.\ L(A) \cap L(B) = \emptyset$,
- $\forall A\ \in N.\ L(A) \neq \emptyset$, and
- $\forall A \in N.\ \exists \alpha, \beta \in \{N \cup\ \Sigma\}^*.\ S \stackrel{*}{\to} \alpha A \beta$.

The *meet* operation $\sqcap$ on reduced grammars corresponds to the language intersection and can be defined in the following way. Let $G_i = \langle N_i, \Sigma, S_i, P_i \rangle$ for $i = 1, 2$ be two reduced grammars. Let $G = \langle N, \Sigma, S, P \rangle$ where $N = N_1 \times N_2, S = \langle S_1, S_2 \rangle$,

$$P = \{S \to x \equiv \langle A_1, A_2 \rangle | (S_i \to x \equiv A_i) \in P_i, i = 1, 2, x \in X\} \cup$$

$$\{\langle A_1, A_2 \rangle \to f(\langle B_1^1, B_1^2 \rangle, \ldots, \langle B_n^1, B_n^2 \rangle) | (A_i \to f(B_1^i, \ldots, B_n^i)) \in P_i, i = 1, 2\}$$
$$\cup\ \{\langle A_1, A_2 \rangle \to x | (A_i \to x) \in P_i, i = 1, 2, x \in X\}.$$

Finally, we define $G_1 \sqcap G_2 \stackrel{def}{=} Red(G)$.

Lemma 1. $L(G_1 \sqcap G_2) = L(G_1) \cap L(G_2)$.

For reduced grammars G_1, G_2, the grammar $G_1 \sqcap G_2$ can be constructed in time $O(kn)$, where $n = max(|G_1|, |G_2|)$ and $k = |X|$.

The partial order $\sqsubseteq$ on reduced grammars is introduced by the definition $G_1 \sqsubseteq G_2 \stackrel{def}{=} L(G_1) \subseteq L(G_2)$. The grammar $\{S \to x \equiv A_x, A_x \to x \mid x \in X\}$ will be denoted by $\mathbf{O}$, $L(\mathbf{O}) = \{x \equiv x \mid x \in X\}$, thus $\forall\ G.\mathbf{O} \sqsubseteq G$. We denote by $\mathcal{L}(X)$ the set of all reduced grammars augmented by a new distinguished element $\mathbb{1}$ that satisfies $\mathbb{1} \sqcap G = G \sqcap \mathbb{1} = G$. The equality relation on $\mathcal{L}(X)$ can be defined by $G_1 = G_2 \stackrel{def}{=} G_1 \sqsubseteq G_2 \wedge G_2 \sqsubseteq G_1$. A nonterminal A of a grammar is said to *know* a term t if $A \stackrel{*}{\to} t$. For a grammar G and a term $t \in TB$, a grammar $G + t$ having a unique nonterminal A that knows t can be built in the following way:

If the grammar G already has a nonterminal that knows t or $G = \mathbb{1}$ then set $G + t \stackrel{def}{=} G$. Otherwise if t is a variable x then add a new nonterminal

A and a rule $A \to x$ to the grammar G; if $t = f(t_1, \ldots, t_n)$, we build the grammar $G' = (\ldots(G + t_1)\ldots + t_n)$, add a new nonterminal A and a rule $A \to f(A_1, \ldots, A_n)$ to the grammar G', where A_i is the nonterminal in G', which knows the term t_i, for $i = 1, \ldots, n$. This construction of the grammar $G + t$ can be done in time proportional to the sum of the sizes of the grammar G and the term t.

For an address a of a call $F(t_1, \ldots, t_n)$ in a scheme we define the grammar transformer

$$[\![call(a)]\!] : \mathcal{L}(Var(a) \cup\ Var(\Lambda)) \to \mathcal{L}(Var([F]) \cup\ Var(\Lambda)).$$

Set $[\![call(a)]\!]\mathbb{1} \stackrel{def}{=} \mathbf{1}$. For $G \neq \mathbb{1}$, let $G' = (\ldots(G + t_1)\ldots + t_n)$. Let A_i be the nonterminal of G' that knows the term t_i, and let B_i be the nonterminal of G' that knows the i-th formal parameter x_i of the symbol $F (i = 1, \ldots, n)$. For all $i = 1, \ldots, n$, if $A_i \neq B_i$, then we delete the rules $S \to x_i \equiv B_i$ and $B_i \to x_i$ from the grammar G' and add the new rules $S \to x_i \equiv A_i$ and $A_i \to x_i$. Finally, define $[\![call(a)]\!]G \stackrel{def}{=} Red(G')$.

Lemma 2. *For all addresses a of a scheme $[\![call(a)]\!]$ is a distributive grammar transformer, i.e.*

$$[\![call(a)]\!](G_1 \sqcap G_2) = [\![call(a)]\!]G_1 \sqcap [\![call(a)]\!]G_2.$$

Proof. Define three elementary grammar transformers:

- $[\![x := t]\!]$ for a term t, new variable x not occurring in t and in the grammar G to which this transformer is applied. Set $[\![x := t]\!]\mathbb{1} \stackrel{def}{=} \mathbb{1}$. For $G \neq \mathbb{1}$, let $G' = (G + t)$ and let A be the nonterminal of G' that knows the term t. Now we add the new rules $S \to x \equiv A$ and $A \to x$ to the grammar G' and define $[\![x := t]\!]G \stackrel{def}{=} G'$.

- $[\![forget(x)]\!]$ for a variable x. The grammar $[\![forget(x)]\!]G$ is obtained from the grammar G by deleting all the rules that contain the variable x and reducing the result.

- $[\![ren(x, y)]\!]$ for variables x, y, where variable y does not occur in the grammar G, to which this transformer is applied. The grammar $[\![ren(x, y]\!]G$ is obtained from the grammar G by renaming all occurrences of the variable x in the rules of the grammar G by the variable y.

To prove the distributivity of the transformer $\mathbf{E} = [\![x := t]\!]$, i.e.

$$\mathbf{E}(G_1 \sqcap G_2) = \mathbf{E}G_1 \sqcap \mathbf{E}G_2,$$

we assume $(y \equiv \tau) \in L(\mathbf{E}(G_1 \sqcap G_2))$ and consider the three cases.

1. $x \neq y$ and x does not occur in τ. Then
   $(y \equiv \tau) \in L(\mathbf{E}(G_1 \sqcap G_2)) \leftrightarrow (y \equiv \tau) \in L(G_1 \sqcap G_2) \leftrightarrow$
   $(y \equiv \tau) \in L(G_1) \cap L(G_2) \leftrightarrow (y \equiv \tau) \in L(\mathbf{E}G_1) \wedge (y \equiv \tau) \in L(\mathbf{E}G_2)$.

2. $x \neq y$ and x occurs in τ. Let $\sigma = \tau[t/x]$, then
   $(y \equiv \tau) \in L(\mathbf{E}(G_1 \sqcap G_2)) \leftrightarrow (y \equiv \sigma) \in L(G_1 \sqcap G_2) \leftrightarrow$
   $(y \equiv \sigma) \in L(G_1) \cap L(G_2) \leftrightarrow (y \equiv \tau) \in L(\mathbf{E}G_1) \wedge (y \equiv \tau) \in L(\mathbf{E}G_2)$.

3. $x = y$. Then x does not occur in t and $(x \equiv t) \in L(\mathbf{E}G)$ for
   $G = G_1, G_2, G_1 \sqcap G_2$, hence $(x \equiv \tau) \in L(\mathbf{E}(G_1 \sqcap G_2)) \leftrightarrow$
   $\exists \sigma, \sigma = [\tau_1/x_1, \ldots, \tau_m/x_m]\sigma t = \sigma\tau \wedge \forall i\ (x_i \equiv \tau_i) \in L(\mathbf{E}(G_1 \sqcap G_2)) \leftrightarrow$
   $\forall i\ (x_i \equiv \tau_i) \in L(G_1 \sqcap G_2) \leftrightarrow \forall i\ (x_i \equiv \tau_i) \in L(G_1) \cap L(G_2) \leftrightarrow$
   $\exists \sigma, \sigma = [\tau_1/x_1, \ldots, \tau_m/x_m]\ \sigma t = \sigma\tau \wedge \forall i\ (x_i \equiv \tau_i) \in L(\mathbf{E}G_1) \cap L(\mathbf{E}G_2) \leftrightarrow$
   $(x \equiv \tau) \in L(\mathbf{E}G_1) \cap L(\mathbf{E}G_2)$.

The distributivity of the transformers $[\![forget(x)]\!]$ and $[\![ren(x,y)]\!]$ is obvious. Now, for an address a of a call $F(t_1, \ldots, t_n)$, the transformer $[\![call(a)]\!]$ can be represented as the composition of the elementary transformers

$$[\![z_i := t_i]\!], [\![forget(x_j)]\!], \text{ and } [\![ren(z_i, y_i)]\!]$$

where $y_1, \ldots, y_n$ are the formal parameters of the symbol F, $z_1, \ldots, z_n$ are auxiliary new variables, and $x_j \in Var(a)$. Finally, the composition of distributive transformers is distributive. □

Now we state a data flow analysis problem for $Graph(S)$ by fixing

- an initial marking μ_0 that associates the grammar $\mathbf{O}$ to the entry node Λ and the grammar $\mathbb{1}$ to all other nodes of $Graph(S)$,
- a semantic function which associates a distributive grammar transformer $[\![call(a)]\!]$ to any edge a of $Graph(S)$.

Let w be a path in $Graph(S)$, i.e. a sequence of adjacent edges, and let

$$\Phi_w(G) = \begin{cases} G, & \text{if } w \text{ does not contain any edge,} \\ [\![call(a)]\!]\Phi_{w'}(G), & \text{if } w = w'a, \text{ where } a \text{ is an address of a call.} \end{cases}$$

Our data flow analysis problem consists of finding the "meet over all paths solution"

$$mop([F]) = \bigsqcap_{w \in W} \Phi_w(\mathbf{O})$$

for all nodes $[F]$ in $Graph(S)$, where W is the set of all paths in $Graph(S)$ from the entry node to a node $[F]$.

It is well known [8] how a meet over all paths problem can be solved. We use a marking algorithm which effectively builds a stationary marking μ of $Graph(S)$ such that $\mu[F] = mop([F])$ for all nodes $[F]$.

A reachable marking μ' is either the initial marking or a marking which is obtained from a reachable marking μ by an application of the following marking rule to an edge a leading from a node u to a node v in $Graph(S)$:

$$\mu' v = \mu v \sqcap ([\![call(a)]\!]\mu u)$$

for the node v and $\mu' v' = \mu v'$ for all other nodes v' in $Graph(S)$.

A reachable marking is called stationary, if it is not changed by any application of the marking rule.

The semi-lattice of reduced grammars satisfies the descending chain condition, and all grammar transformers $[\![call(a)]\!]$ are distributive. So it follows from the results proved in [8] that there exist a unique stationary marking μ that satisfies $\mu[F] = mop([F])$ for all nodes $[F]$ of the graph $Graph(S)$.

Since the stationary marking μ is a solution to a meet over all paths problem, the condition $(x \equiv t) \in L(\mu[F])$ is true for a node $[F]$ and for a formal parameter x of F iff $S \approx S[a \leftarrow t]$ holds for all addresses a of the occurrences of x in the body of the symbol F.

Our final transformation rule scheme *context replacement* contains all rules $S \leftrightarrow S[a \leftarrow t]$ where a is an address of an occurrence of a formal parameter x in the body of a symbol F, and the condition $(x \equiv t) \in L(\mu[F])$ is satisfied for the stationary marking μ of $Graph(S)$.

Example:

$$\left\langle \begin{array}{l} F(h,h); \\ F(x,y) \Leftarrow \mathbf{if}(px, F(fx,fy), gx) \end{array} \right\rangle \leftrightarrow \left\langle \begin{array}{l} F(h,h); \\ F(x,y) \Leftarrow \mathbf{if}(px, F(fx,fx), gx) \end{array} \right\rangle$$

Here $(x \equiv y) \in L(\mu[F])$ holds for the stationary marking μ of the former scheme.

The transformation system Σ_{lin} contains all the rules generated by the rule schemes described in Sections 6.1 – 6.7.

6.8 Correctness of the transformation system Σ_{lin}

Theorem 1. *If $\mathcal{G} \leftrightarrow \mathcal{G}'$ is a rule of the transformation system Σ_{lin} then $\mathcal{G} \approx \mathcal{G}'$.*

Proof. Useless definitions, terms at irrelevant addresses, redundant parameters and applications of the *copying and identifying* rule do not affect the process of the approximation sequence construction for a term.

Simple fold/unfold affects only the "speed" of the approximation sequence construction, but not the determinant. If $\mathcal{G} \leftrightarrow \mathcal{G}'$ is a *simple fold/unfold* rule, and A, A' are the approximation sequences of the corresponding entries of the fragments $\mathcal{G}$ and $\mathcal{G}'$, respectively, then $\forall n \exists m A_n \sqsubseteq A'_m \wedge \forall n \exists m A'_n \sqsubseteq A_m$. Thus, the least upper bounds of these sequences coincide. Note that each application of our *simple fold/unfold* to a fragment that contains the definition $F(\bar{x}) \Leftarrow \tau$ cannot destroy this definition, i.e.

$$\left\langle \begin{array}{l} F(h); \\ F(x) \Leftarrow a \end{array} \right\rangle \leftrightarrow \left\langle \begin{array}{l} F(h); \\ F(x) \Leftarrow F(x) \end{array} \right\rangle$$

is not a simple fold/unfold rule.

To prove $\mathcal{G} \downarrow a \approx \omega$ for a hopeless address a in a fragment $\mathcal{G}$ it suffices to show that if $t \not\approx \omega$ for $t = \mathcal{G} \downarrow a$ then the address a is an exit. We can prove this statement first for terms t without calls by induction on the height $h(t)$ of the term t. If $h(t) = 0$ then t is a variable or a constant different from ω. Hence the address a will be declared an exit by the application of marking rule 1 (variable) or 2 (constant) from the definition of an exit. Suppose the statement is true for all terms (without calls) of height n, and consider a term t of height $n+1$. Then $t = f(t_1, \ldots, t_m)$ and from $t \not\approx \omega$ it follows that $\exists \Delta \in Strict(f).\ \forall i \in \Delta.\ t_i \not\approx \omega$. By the induction hypothesis we conclude that there exists a collection $\Delta \in Strict(f)$ such that for all $i \in \Delta$ the address $a.i$ is an exit. Therefore, the address a will be declared an exit of t by application of the rule 2 from the definition of an exit. Suppose now t contains a call. Then $\exists n.\ J(\pi^n) \neq \omega$ and therefore $J(t') \neq \omega$ for $t' = (\pi^n t)[N \leftarrow \omega]$ where N is the set of all call addresses in $\pi^n t$. The term t' does not contain calls, so the process of determination of exits in t' by means of rules 1-2 applications will declare the root address of t' an exit. Adding the applications of the rule 3 after declaring a body of a defined symbol in $\mathcal{G}$ to be an exit we obtain from this process the process for determining the exits of the fragment $\mathcal{G}$. Since the root address of t' was an exit, so is the address a.

Context replacement preserves the tree equivalence because it uses the meet over all path solution of the parameter dependence analysis problem described in this Section. □

7 Linear schemes reduction

A linear scheme S is called *reduced* if

1. there are no useless definitions in S,
2. there are no hopeless addresses in S other than addresses of occurrences of the term ω,
3. the entry of the scheme S is either a call or the term ω,
4. the body of each internal symbol is either a variable, or a call, or a term of the form $f(t_1, \ldots, t_n)$, where $n = r(f) \geq 0, f \in \mathcal{F}_b$, and each term t_i for $i = 1, \ldots, n$ is either a call or the term ω.

Thus, only three kinds of definitions may occur in reduced linear schemes: *projective definitions*, where the body is a variable, *chain definitions*, where the body is a call, and *basic definitions*, where the body is a term of the form $f(t_1, \ldots, t_n), n = r(f) \geq 0, f \in \mathcal{F}_b$, and each term t_i for $i = 1, \ldots, n$ is either a call or the term ω.

Note that there is a unique reduced linear scheme which is tree equivalent to ω, namely the scheme ω itself. The two tree equivalent but different schemes L_n and U_n from Example 2 show that reduced linear schemes are not unique up to renaming of formal parameters and defined symbols.

Theorem 2. *By application of rules from Σ_{lin} any linear scheme S can be transformed into a reduced linear scheme S' such that $S \approx S'$.*

Proof. The first reducibility condition is achieved by application of the *delete useless definitions* rule. The bodies of such definitions have irrelevant addresses which can be detected in linear time.

Let us bound the time needed for detection of all exits of a linear scheme. We assume that the scheme contains a description of the strict parameter collections for each basic symbol of the scheme. Using a marking algorithm for detection of the scheme exits, we have to apply the marking rules at most once to any address of a term from TB or of a call. To any address of a term $t = f(t_1, \ldots, t_m), f \in \mathcal{F}_b, t \notin TB$, the marking rule is applied no more than $m+1$ times. The proper application of the marking rule to the term $f(t_1, \ldots, t_m)$ requires time proportional to the length of the writing of the strict parameter collection of the symbol f. Hence we can take the cube of the scheme size as a rough upper bound for the time needed for detection of all exits of a linear scheme. After replacing all terms at hopeless addresses by ω we obtain a scheme that satisfies the second reducibility condition.

If the entry t of the scheme S is neither a call nor the term ω, we introduce a definition $F(x_1, \ldots, x_n) \Leftarrow t$ of a new useless symbol F (i.e. not appearing in S) into the scheme S, where $x_1, \ldots, x_n$ are all variables of the term t. By *simple fold* rule application replace the entry of the scheme by the call $F(x_1, \ldots, x_n)$. As a result, the third reducibility condition for S also becomes true.

If the body of a symbol F in S has the form $f(t_1, \ldots, t_m)$, where $f \in \mathcal{F}_b$, then each subterm $t_i, t_i \neq \omega, i = 1, \ldots, m$, of the body can be transformed into a call in exactly the same way as it has been done above for the scheme entry. As a result, some new definitions may occur, and we have to reduce their bodies. It is clear, however, that this process terminates, since the dephts of the bodies in the new definitions decrease. When the process terminates, each definition in the scheme becomes either projective or chain or basic. The size of the obtained reduced scheme is $O(k \cdot n)$, where $n = |S|$ is the size of the given scheme and k is the maximal number of formal parameters of the symbols defined in S. $\square$

The reduction of the schemes R_1 and R_2 from Example 1 results in the reduced schemes

$$R_1' = \left\langle \begin{array}{ll} F_1(a,a); & \\ F_1(x,y) & \Leftarrow f(F_2(x), F_2(y), F_3, F_1(gx, gy)) \\ F_2(x) & \Leftarrow x \\ F_3 & \Leftarrow gF_4 \\ F_4 & \Leftarrow b \end{array} \right\rangle$$

and

$$R_2' = \left\langle \begin{array}{ll} F_1(a,c); & \\ F_1(u,v) & \Leftarrow f(F_2(u),F_2(u),F_2(gb),F_3(gu,hv)) \\ F_2(u) & \Leftarrow u \\ F_3(u,v) & \Leftarrow f(F_2(u),F_2(u),F_2(gb),F_1(gu,ggv)) \end{array} \right\rangle$$

respectively. The schemes from Example 2 are already reduced.

8 The similarity of reduced schemes

In the following, we suppose the schemes under consideration to be not tree empty, since after reducing a tree empty scheme to the scheme ω, the tree equivalence and transformation problems become trivial for such schemes. We will introduce a decidable similarity relation for reduced linear schemes. The similarity of reduced linear schemes will be shown to be a necessary condition for their tree equivalence.

We define a preliminary (in general, not symmetric) binary *compatibility* relation on defined symbols of reduced schemes S_1 and S_2. First of all, if F_i is the main symbol of the entry of the scheme S_i, $i = 1, 2$, then F_1 is compatible with F_2. Further, if a defined in S_1 symbol F_1 is compatible with a defined in S_2 symbol F_2, then

- if $S_i \downarrow [F_i] = f(t_i^1, \ldots, t_i^k)$ for $i = 1, 2$, and t_i^j is a call to F_i^j then F_1^j is compatible with F_2^j;
- if the body of the symbol F_i is a call to some symbol $F_i'(i = 1, 2)$ then F_1' is compatible with F_2';
- if the definition of the symbol F_2 is projective any symbol called in the body of F_1 is compatible with F_2;
- if the definition of the symbol F_1 is basic, and the body of the symbol F_2 is a call to some symbol F', then the symbol F_1 is compatible with F';
- if the body of the symbol F_1 is a call to some symbol F' and the definition of the symbol F_2 is basic, then the symbol F' is compatible with F_2.

All other symbols are incompatible.

A defined symbol F of a scheme S is *fruitless*, if $det(F(x_1, \ldots, x_{r(F)}))$ is a finite term. Otherwise, the symbol F is said to be *fruitful*.

Call two reduced linear schemes S_1 and S_2 *similar* (for short: $S_1 \bowtie S_2$), if the compatibility relation on defined symbols of these schemes satisfies the following conditions:

- fruitful symbols are incompatible with fruitless ones, and vice versa; and

- if $F(\bar{x}) \Leftarrow f(t_1, \ldots, t_n)$ and $F'(\bar{y}) \Leftarrow g(t'_1, \ldots, t'_m)$ are basic definitions of two compatible symbols, then $f = g$ and for all $i = 1, \ldots, n$ either both terms t_i and t'_i are equal to ω, or both differ from ω.

Lemma 3. *Tree equivalent reduced linear schemes are similar:* $S_1 \approx S_2 \Rightarrow S_1 \bowtie S_2$.

Proof. It follows from the definition of the compatibility relation that if $S_1 \approx S_2$ and the symbols F, F' are compatible, then there exist some calls t_F and $t_{F'}$ to these symbols such that $det(t_F) = det(t_{F'})$. If $\neg(S_1 \bowtie S_2)$, then there exists a pair of compatible symbols, for which at least one of the similarity conditions is violated. Then any two calls to these symbols have different determinants. □

The algorithm deciding the similarity for reduced linear schemes follows directly from the definition of the compatibility relation for defined symbols of the schemes. The detection of all fruitful symbols of a scheme needs time linear in the scheme size. Then we build the compatible symbol pairs and check the two conditions from the similarity definition for all compatible pairs. The upper bound for the complexity of this algorithm is the square of the maximum size of the schemes.

9 The product of reduced schemes

Suppose we are given a pair of (not tree empty) similar reduced linear schemes:

$$S_1 = \langle t;\ \{F_i(x_1, \ldots, x_{k_i}) \Leftarrow \tau_i,\ i = 1, \ldots, n_1\}\rangle,$$

$$S_2 = \langle t';\ \{F'_j(y_1, \ldots, y_{l_j}) \Leftarrow \tau'_j,\ j = 1, \ldots, n_2\}\rangle.$$

We can assume all defined symbols of these schemes to be pairwise disjoint. If a symbol H is defined in both schemes replace all occurrences of H in one of the schemes (for example, in the first one) by a new symbol H' which does not occur in either scheme. Let us show how this transformation can be carried out by means of application of rules from Σ_{lin}. Let $H(\bar{x}) \Leftarrow \tau$ be the definition of H in the first scheme, and let τ' be the term obtained from τ by replacement of all occurrences of the symbol H by the symbol H'. Introduce a new useless definition $H'(\bar{x}) \Leftarrow \tau'$ in the first scheme, and by application of the *copy* rule replace the main symbol of each call to H in the first scheme by H'. As the result, the definition of H in the first scheme becomes useless and can be deleted.

Let us fix a map $\mathcal{P} : \mathcal{F}_d \times \mathcal{F}_d \to \mathcal{F}_d$ which maps different pairs of defined symbols to different defined symbols. Denote

$$H_{i,j} = \mathcal{P}(F_i, F'_j) \text{ and } H'_{j,i} = \mathcal{P}(F'_j, F_i).$$

A *product* $S_1 \times S_2$ of the schemes S_1 and S_2 is the reduced linear scheme

$$S_1 \times S_2 = \langle t''; \{H_{i,j}(x_1, \ldots, x_{k_i}) \Leftarrow \sigma_{i,j} |\ F_i \text{ is compatible with } F'_j\}\rangle$$

where $t'' = H_{p,q}(t_1, \ldots, t_{k_p})$ for some p, q such that $F_p(t_1, \ldots, t_{k_p})$ is the entry of the scheme S_1, and F'_q is the main symbol of the entry of the scheme S_2. If the definition of the symbol F_i is projective then $\sigma_{i,j} = \tau_i$ for all j such that F_i is compatible with F'_j; if F_i has a chain definition, and $\tau_i = F_p(t_1, \ldots, t_{k_p})$ for some p then

$$\sigma_{i,j} = \begin{cases} H_{p,q}(t_1, ..., t_{k_p}), & \text{if } \tau'_j \text{ is a call to } F'_q, \\ H_{p,j}(t_1, ..., t_{k_p}), & \text{if } \tau'_j \text{ is not a call.} \end{cases}$$

Finally, if F_i has a basic definition, $\tau_i = f(\ldots, F_p(t_1, \ldots, t_{k_p}), \ldots)$ for some p, and $f \in \mathcal{F}_b$, then

$$\sigma_{i,j} = \begin{cases} H_{i,q}(x_1, ..., x_{k_i}), & \text{if } \tau'_j \text{ is a call to } F'_q, \\ f(...H_{p,q}(t_1, ..., t_{k_p})...), & \text{if } \tau'_j = f(...F'_q(t'_1..., t'_{l_q}), ...), \\ f(..., H_{p,j}(t_1, ..., t_{k_p}), ...), & \text{if } \tau'_j \text{ is a variable.} \end{cases}$$

Lemma 4. *The product scheme $S_1 \times S_2$ can be obtained from the scheme S_1 by application of transformation rules from Σ_{lin}, thus $S_1 \times S_2 \approx S_1$.*

Proof. To get a compact description of our transformation process we call a pair (i, j) *singular* if the symbol F_i with a basic definition is compatible with the symbol F'_j with a chain definition.

The first step in constructing the product scheme consists of introducing useless definitions of new symbols $H_{i,j}, r(H_{i,j}) = r(F_i)$, with the bodies τ_i, for all pairs (i, j) such that F_i is compatible with F'_j. After this step, for the partition

$$\mathcal{K}_i = \{F_i\} \cup \ \{H_{i,j} \mid F_i \text{ is compatible with } F'_j\}, \ i = 1, \ldots, n_1$$

the premise of the copy rule is satisfied, and by applying this rule we replace the main symbols of the entry and of the bodies of symbols $H_{i,j}$ for non-singular pairs (i, j) by the elements from class $\mathcal{K}_i$, which have been defined in the description of the product scheme. Let $(i, j_1), \ldots, (i, j_q)$ be a sequence of singular pairs such that $q \geq 1$, the term τ'_{j_l} is a call to $F'_{j_{l+1}}$ for $l = 1, \ldots, q$, and the pair (i, j_{q+1}) is non-singular. Such a sequence is finite since the scheme S_2 does not contain hopeless addresses different from occurrences of the term ω. By applying the copy rule for "final singular" pairs (i, j_q) replace the symbols of the class $\mathcal{K}_i$ occurring in the body of H_{i,j_q} by elements of this class such that after replacement this body will coincide with the body of the symbol $H_{i,j_{q+1}}$. By applying the simple fold rule replace the body of the symbol H_{i,j_l} by the call $H_{i,j_{l+1}}(x_1, \ldots, x_{k_{j_{l+1}}})$, for $l = 1, \ldots, q$. After this transformation all definitions of the symbols F_i for $i = 1, \ldots, n_1$ become useless, so we can delete these definitions and get the scheme $S_1 \times S_2$. $\square$

While constructing the scheme $S_2 \times S_1$ we will use symbols $H'_{j,i}$ instead of $H_{i,j}$. Then the defined symbol sets of the schemes $S_1 \times S_2$ and $S_2 \times S_1$ will be disjoint.

Lemma 5. *The mapping $\Upsilon(H_{i,j}) = H'_{j,i}$ of the defined symbols of the scheme $S'_1 = S_1 \times S_2$ into defined symbols of the scheme $S'_2 = S_2 \times S_1$ satisfies the following conditions*

- *if the entry of the scheme S'_1 is a call to a symbol F, then the entry of the scheme S'_2 is a call to the symbol $\Upsilon(F)$;*
- *the symbol F is fruitful iff the symbol $\Upsilon(F)$ is fruitful;*
- *if both definitions for F and $\Upsilon(F)$ are not projective, then*
  - *either both definitions for F and $\Upsilon(F)$ are basic,*

    $$S'_1 \downarrow [F] = f(\lambda_1, \ldots, \lambda_m),\ S'_2 \downarrow [\Upsilon(F)] = f(\lambda'_1, \ldots, \lambda'_m),$$

    and for all $i = 1, \ldots, m$ either $\lambda_i = \lambda'_i = \omega$ or $H'_i = \Upsilon(H_i)$ holds for the main symbols H_i and H'_i of the terms λ_i and λ'_i, respectively;
  - *or definitions for F and $\Upsilon(F)$ are both chain definitions, and $H' = \Upsilon(H)$ holds for the main symbols H and H' of their bodies.*

Proof. The following table shows how the body $\sigma_{i,j}$ of the symbol $H_{i,j}$ depends on the kinds of definitions of the symbols F_i and F'_j:

		$F'_j\bar{y} \Leftarrow f(\ldots F'_q(\bar{t}')\ldots)$ 1	$F'_j\bar{y} \Leftarrow F'_q(\bar{t}')$ 2	$F'_j\bar{y} \Leftarrow y_l$ 3
$F_i\bar{x} \Leftarrow f(\ldots F_p(\bar{t})\ldots)$	1	$f(\ldots H_{p,q}(\bar{t})\ldots)$	$H_{i,q}(\bar{x})$	$f(\ldots H_{p,j}(\bar{t})\ldots)$
$F_i\bar{x} \Leftarrow F_p(\bar{t})$	2	$H_{p,j}(\bar{t})$	$H_{p,q}(\bar{t})$	$H_{p,j}(\bar{t})$
$F_i\bar{x} \Leftarrow x_k$	3	x_k	x_k	x_k

If one of the symbols F and $\Upsilon(F)$ is fruitful but the other one is fruitless then we immediately get a contradiction to the similarity of the given schemes. The other properties of the map Υ follow from the description of the schemes S'_1 and S'_2 and can be easily proved by case analysis of various combinations of kinds of definitions of the given schemes S_1 and S_2:

line, row	$S'_1 : H_{i,j}$	$S'_2 : \Upsilon(H_{i,j}) = H'_{j,i}$
$1,1:$	$H_{i,j}(\bar{x}) \Leftarrow f(\ldots, H_{p,q}(\bar{t}), \ldots)$	$H'_{j,i}(\bar{y}) \Leftarrow f(\ldots, H'_{q,p}(\bar{t}'), \ldots)$
$2,1:$	$H_{i,j}(\bar{x}) \Leftarrow H_{p,j}(\bar{t})$	$H'_{j,i}(\bar{y}) \Leftarrow H'_{j,p}(\bar{y})$
$3,1:$	$H_{i,j}(\bar{x}) \Leftarrow x_k$	$H'_{j,i}(\bar{y}) \Leftarrow f(\ldots, H'_{q,i}(\bar{t}'), \ldots)$
$1,2:$	$H_{i,j}(\bar{x}) \Leftarrow H_{i,q}(\bar{x})$	$H'_{j,i}(\bar{y}) \Leftarrow H'_{q,i}(\bar{t}')$
$2,2:$	$H_{i,j}(\bar{x}) \Leftarrow H_{p,q}(\bar{t})$	$H'_{j,i}(\bar{y}) \Leftarrow H'_{q,p}(\bar{t}')$
$3,2:$	$H_{i,j}(\bar{x}) \Leftarrow x_k$	$H'_{j,i}(\bar{y}) \Leftarrow H'_{q,i}(\bar{t}')$
$1,3:$	$H_{i,j}(\bar{x}) \Leftarrow f(\ldots, H_{p,j}(\bar{t}), \ldots)$	$H'_{j,i}(\bar{y}) \Leftarrow y_l$
$2,3:$	$H_{i,j}(\bar{x}) \Leftarrow H_{p,j}(\bar{t})$	$H'_{j,i}(\bar{y}) \Leftarrow y_l$
$3,3:$	$H_{i,j}(\bar{x}) \Leftarrow x_k$	$H'_{j,i}(\bar{y}) \Leftarrow y_l$

□

For schemes R_1' and R_2' from Example 1 we can build the product scheme

$$\left\langle \begin{array}{l} H_{1,1}(a,a); \\ H_{1,1}(x,y) \Leftarrow f(H_{2,2}(x), H_{2,2}(y), H_{3,2}, H_{1,3}(gx,gy)) \\ H_{1,3}(x,y) \Leftarrow f(H_{2,2}(x), H_{2,2}(y), H_{3,2}, H_{1,1}(gx,gy)) \\ H_{2,2}(x) \Leftarrow x \\ H_{3,2} \Leftarrow gH_{4,2} \\ H_{4,2} \Leftarrow b \end{array} \right\rangle$$

10 Completeness of the system Σ_{lin}

This Section is entirely concerned with the proof of the completeness theorem for the system Σ_{lin}.

Theorem 3. *Σ_{lin} is a complete system of transformations preserving the tree equivalence of linear recursion schemes.*

Proof. We say a pair of recursion schemes S_1, S_2 satisfies the *separation condition* if any formal parameter and any defined symbol does not occur in both schemes S_1 and S_2. In the previous Section, we constructed the schemes $S_1 \times S_2$ and $S_2 \times S_1$ such that no defined symbol occurs in both schemes. Let us show how the separation condition can be achieved by means of rule applications for variables of an arbitrary pair S, S' of recursion schemes. Assume that the variable $x \in \mathcal{X}$ is a formal parameter of both a symbol H from S and a symbol from S'. We replace the definition $H(\ldots, x, \ldots) \Leftarrow \tau$ in S by $H(\ldots, y, \ldots) \Leftarrow [y/x]\,\tau$, where y is a new variable, not occurring yet in either of the schemes S and S'. By application of the rules from Σ_{lin} this transformation can be done in the following way: first of all, introduce a useless definition $F(\ldots, x, y, \ldots) \Leftarrow \tau$. By applying the *adding of redundant parameters* rule replace each call $H(\ldots, t, \ldots)$ by the call $F(\ldots, t, t, \ldots)$. Now the definition of the symbol H becomes useless and is deleted. Since the actual parameters, corresponding to the formal parameters x and y, coincide in all calls to F, we have $(x \equiv y) \in L(\mu[F])$ for the stationary marking μ of $Graph(S)$. Thus, by applying the context replacement rule we can replace all occurrences of x by the variable y. As a result, the formal parameter x of the symbol F becomes redundant. We introduce again a useless definition $H(\ldots, y, \ldots) \Leftarrow [y/x]\,\tau$, by applying the *delete redundant parameters* rule we replace each call $F(\ldots, t, t, \ldots)$ by the call $H(\ldots, t, \ldots)$. Then we delete the useless definition of the symbol F.

We can assume that the given tree equivalent reduced linear schemes are not tree empty. Thus, due to Lemma 4 it is sufficient to transform one of the two tree equivalent reduced linear schemes from the pair $S_1' = S_1 \times S_2, S_2' = S_2 \times S_1$ satisfying the separation condition into another one by applying rules from Σ_{lin}.

In the next step, we add the formal parameters $y_1, \ldots, y_{l_j}$ (of the symbol F_j') to the symbol $H_{i,j}$. To this end, for all pairs (i,j) such that the symbol F_i is compatible with F_j', we introduce a useless definition $K_{i,j}(\bar{z}) \Leftarrow S_1' \downarrow [H_{i,j}]$ of a new symbol $K_{i,j}, r(K_{i,j}) = r(F_i) + r(F_j')$, to the scheme S_1'. By applying

the *adding of redundant parameters* rule we replace the occurrence of each call $H_{i,j}(t_1, \ldots, t_{k_i})$ by the call $K_{i,j}(t_1, \ldots, t_{k_i}, b, b, \ldots, b)$ where b is an arbitrary constant (but not ω!). As a result, the definitions of all symbols $H_{i,j}$ become useless and are deleted. Now, all addresses of the inserted actual parameter occurrences of the constant b are irrelevant. Applyingi the *replace irrelevant term occurrences* rule we replace

- each call $K_{p,j}(t_1, \ldots, t_{k_p}, b, \ldots, b)$ that occurs in the body of $K_{i,j}$ by the call $K_{p,j}(t_1, \ldots, t_{k_p}, y_1, \ldots, y_{l_j})$ if F_i has a chain definition, but F'_j not, or if F_i has a basic definition, but F'_j has a projective one;
- each call $K_{p,q}(t_1, ..., t_{k_p}, b, ..., b)$ that occurs in the body of $K_{i,j}$ by the call $K_{p,q}(t_1, ..., t_{k_p}, t'_1, ..., t'_{l_q})$, if F'_j has a chain definition $F'_j\bar{y} \Leftarrow F'_q(t'_1, ..., t'_{l_q})$;
- each call $K_{p,q}(t_1, \ldots, t_{k_p}, b, \ldots, b)$ at address $[K_{i,j}].m$ (i.e. the m-th subterm of the body) by the call $K_{p,q}(t_1, \ldots, t_{k_p}, t'_1, \ldots, t'_{l_q})$, if both definitions for F_i and F'_j are basic, and $S_2 \downarrow ([F'_j].m) = F'_q(t'_1, ..., t'_{l_q})$.

We denote the obtained scheme by $\hat{S}_1$. Applying the algorithm described in Section 6.7, we construct a stationary marking μ of $Graph(\hat{S}_1)$. This marking is needed in the last step of the transformation of the scheme $\hat{S}_1$ into $S_2 \times S_1$.

We call a defined symbol $K_{i,j}$ of the scheme $\hat{S}_1$ *critical*if either its definition is projective or the scheme $S_2 \times S_1$ contains a projective definition for $H'_{j,i}$. It is clear that each critical symbol is fruitless. A critical symbol $K_{i,j}$ is said to be *minimal*, if there exists a non-critical symbol of the scheme $\hat{S}_1$, whose body contains a call to $K_{i,j}$. The following condition is called the *key condition* for a critical symbol $K_{i,j}$:

There exists a nonterminal of the grammar $\mu[K_{i,j}]$, which knows both terms

$$det(\hat{S}_1 \downarrow [K_{i,j}]) \text{ and } det((S_2 \times S_1) \downarrow [H'_{j,i}])$$

(at least one of these terms is a variable).

We can prove that if $\hat{S}_1 \approx S_2 \times S_1$ then the key condition is satisfied for each minimal critical symbol of the scheme $\hat{S}_1$. Indeed, if the key condition is false for a minimal critical symbol $K_{i,j}$ with a projective definition $K_{i,j}(\hat{z}) \Leftarrow x$ (the symmetric case when the definition of $H'_{i,j}$ is projective can be considered similarly) then there exists a path w that leads from entry node to the node $[K_{i,j}]$ in $Graph(\hat{S}_1)$ such that $(x \equiv det((S_2 \times S_1) \downarrow [H'_{j,i}])) \notin L(\Phi_w(\mathbf{O}))$ holds. But then $det(\hat{S}_1) \downarrow a \neq det(S_2 \times S_1) \downarrow a$ for some address a which contradicts $det(\hat{S}_1) = det(S_2 \times S_1)$.

Now we have to replace the body of each minimal critical symbol $K_{i,j}$ of the scheme $\hat{S}_1$ by the body of the symbol $H'_{j,i}$. This can be done by applying rules from Σ_{lin} in the following way.

If the definition of $K_{i,j}$ is projective then add to the scheme $\hat{S}_1$ the new definitions of all symbols called from the body of $H'_{j,i}$ directly or indirectly.

Furthermore, using the truth of the key condition for $K_{i,j}$ and applying the *context replacement* rule, we replace the body of the symbol $K_{i,j}$ by the term $det((S_2 \times S_1) \downarrow [H'_{j,i}])$, after that it only remains to fold this term into term $(S_2 \times S_1) \downarrow [H'_{j,i}]$.

If the definition of $K_{i,j}$ is not projective we carry out these transformations in the opposite direction: first we apply the *simple fold/unfold* rule and replace the body of the symbol $K_{i,j}$ by its determinant which due to the truth of the key condition can be replaced by the variable that coincides with the body of $H'_{j,i}$. After that it only remains to delete some useless definitions possibly arisen in the last step.

Note that after this transformation step all occurrences of the formal parameters $x_1, \ldots, x_{k_i}$ of $K_{i,j}$ are subterms of actual parameters in some calls. Therefore, all addresses of actual parameters corresponding to these formal ones, become irrelevant. We replace all terms at such irrelevant addresses by a constant b. As a result, all formal parameters $x_1, \ldots, x_{k_i}$ of the symbol $K_{i,j}$ become redundant, and we delete them by applying the *delete redundant parameters* rule. Now the schemes can differ only in designations of non-critical symbols. Their renaming can be achieved by application of the copy rule as it was done in the beginning of this section to reach the separation condition for schemes. □

For our schemes from Example 2 we have

$\hat{L}_n =$

$$\left\langle \begin{array}{l} K_{1,1}(a,a); \\ K_{1,1}(x,y) \Leftarrow h(K_{2,n+3}(x,y), K_{n+3,2}(x,y)) \\ K_{i,n+3}(x,y) \Leftarrow K_{i+1,n+3}(f(x,x),y), i = 2,\ldots,n+1 \\ K_{n+3,j}(x,y) \Leftarrow K_{n+3,j+1}(x, g(y,y)), j = 2,\ldots,n+1 \\ K_{n+2,n+3}(x,y) \Leftarrow x \\ K_{i,n+2}(x,y) \Leftarrow g(K_{i+1,n+2}(x,y), K_{i+1,n+2}(x,y)), i = n+3,...,2n+1 \\ K_{2n+2,n+2}(x,y) \Leftarrow g(K_{n+2,n+2}(x,y), K_{n+2,n+2}(x,y)) \\ K_{n+2,n+2}(x,y) \Leftarrow x, \end{array} \right\rangle$$

$U_n \times L_n =$

$$\left\langle \begin{array}{l} H'_{1,1}(a); \\ H'_{1,1}(y) \Leftarrow h(H'_{n+3,2}(y), H'_{2,n+3}(y)) \\ H'_{n+3,i}(y) \Leftarrow H'_{n+3,i+1}(y), i = 2,\ldots,n+1 \\ H'_{j,n+3}(y) \Leftarrow H'_{j+1,n+3}(g(y,y)), j = 2,\ldots,n+1 \\ H'_{j,n+2}(y) \Leftarrow f(H'_{j+1,n+2}(y), H'_{j+1,n+2}(y)), j = n+3,...,2n+1 \\ H'_{2n+2,n+2}(y) \Leftarrow f(H'_{n+2,n+2}(y), H'_{n+2,n+2}(y)) \\ H'_{n+2,n+2}(y) \Leftarrow y. \end{array} \right\rangle$$

The stationary marking of the scheme $\hat{L}_n$ can be described in the following

short form:

$$
\begin{array}{l}
K_{1,1}(x,y) : \{x \equiv y \equiv a\}, \\
K_{i,n+3}(x,y) : \{x \equiv \hat{f}^{i-2}(y), y \equiv a\}, i = 2, \ldots, n+2, \\
K_{n+3,j}(x,y) : \{y \equiv \hat{g}^{j-2}(x), x \equiv a\}, j = 2, \ldots, n+2, \\
K_{i,n+2}(x,y) : \{y \equiv \hat{g}^{n}(x), x \equiv a\}, i = n+3, \ldots, 2n+2, \\
K_{n+2,n+2}(x,y) : \{y \equiv \hat{g}^{n}(x), x \equiv a\},
\end{array}
$$

where

$$
\hat{f}^n(u) = \begin{cases} u, & \text{if } n = 0, \\ f(\hat{f}^{n-1}(u), \hat{f}^{n-1}(u)), & \text{if } n > 0. \end{cases}
$$

The minimal critical symbols are $K_{n+2,n+3}$ and $K_{n+2,n+2}$, and the key conditions for them are satisfied due to

$$
\begin{array}{c}
det(\hat{L}_n \downarrow [K_{n+2,n+3}]) = x(\equiv \hat{f}^n(y)), \\
det(U_n \times L_n \downarrow [H'_{n+3,n+2}]) = \hat{f}^n(y), \\
det(\hat{L}_n \downarrow [K_{n+2,n+2}]) = \hat{g}^n(x)(\equiv y), \\
det(U_n \times L_n \downarrow [H'_{n+2,n+2}]) = y.
\end{array}
$$

11 A polynomial decision algorithm

Theorem 4. *The tree equivalence of linear recursion schemata is decidable in time $O(n^6)$.*

Proof. We extract the algorithm deciding the tree equivalence of schemes from the description of the transformation process turning a linear scheme into another tree equivalent one.

1. First, reduce the given linear schemes S_1 and S_2.

2. If both schemes reduce to the scheme ω then stop with the answer "Yes". If one of the schemes reduces to ω, but the other does not, then stop with the answer "No" .

3. Using the algorithm described in the Section 8 check the similarity of the obtained reduced schemes. If they are not similar the algorithm terminates with the answer "No" .

4. If $S_1 \bowtie S_2$ then construct the product schemes $S_1 \times S_2$, $S_2 \times S_1$, the scheme $\hat{S}_1$, and the stationary marking μ of $Graph(\hat{S}_1)$.

5. Check the key condition for each minimal critical symbol of the scheme $\hat{S}_1$. If it is false for at least one minimal critical symbol the algorithm terminates with the answer "No" otherwise it stops with the answer "Yes" .

Let us approximate the complexity of this algorithm as a function of $n = max(|S_1|, |S_2|)$ and the maximum k of the formal parameter numbers of the defined symbols of the schemes, $k \leq n$. The scheme reduction requires no more than $O(n^3)$ elementary steps. The similarity test, constructing the product schemes, and the scheme $\hat{S}_1$ need no more than $O(kn^2)$ steps. The sizes of the schemes $\hat{S}_1$ and $S_2 \times S_1$ do not exceed $O(kn^2)$. The maximal size of the grammars arising in the course of constructing the stationary marking μ of $Graph(\hat{S}_1)$ does not exceed $O(kn^2)$. The length of the maximal chain in $L(X)$ is no greater than $3|X| \leq 6k$. The time needed for execution of a meet operation and of an application of the mark transformer $[\![call(.)]\!]$ to marks of size $O(kn^2)$ is proportional to $O(kn^2)$. Summing up, we get the upper bound $O(n^6)$ for the complexity of the stationary marking μ construction algorithm. Now we want to check the key condition for a minimal critical symbol $K_{i,j}$ without explicitly building the terms $det(\hat{S}_1 \downarrow [K_{i,j}])$ and $det((S_2 \times S_1) \downarrow [H'_{j,i}])$, because it would require too much time. In Example 2, the determinant of the term $H'_{n+3,n+2}(y)$ has the size $O(2^n)$, although the size of the scheme $U_n \times L_n$ is $O(n)$. We wish to reduce the key condition check to testing a more general condition $Test$.

$Test(A, F, G)$: *A nonterminal A of a reduced grammar G knows the determinant of the body of a fruitless symbol F defined in the scheme $S_2 \times S_1$.*

To check $Test(A, F, G)$, we can use the following relations.

- If $(S_2 \times S_1) \downarrow [F]$ is a variable x, then $Test(A, F, G)$ is true, iff A knows x.
- If $(S_2 \times S_1) \downarrow [F]$ is a call to a symbol F', then

$$Test(A, F, G) = Test(A, F', [\![call([F])]\!]G).$$

- If $(S_2 \times S_1) \downarrow [F] = f(t_1, \ldots, t_m)$, and F_i is the main symbol of the term t_i, then $Test(A, F, G)$ is true, iff the grammar G contains a rule $A \to f(A_1, \ldots, A_m)$, and $Test(A_i, F_i, [\![call([F].i)]\!]G)$ is true for all $i = 1, \ldots, m$.
- $Test(A, F, G_1) \ \wedge \ Test(A, F, G_2) \Leftrightarrow Test(A, F, G_1 \sqcap G_2)$.

We describe a procedure $TEST(A, F, G)$ checking $Test(A, F, G)$ for a scheme S, where $S = \hat{S}_1$ or $S = \hat{S}_2$, by application of a tabular technique. To prevent repeated execution of calls to $TEST$, use a table $Gram$ for maintaining in $Gram[A, F]$ the meet of all grammars, for which $TEST(A, F, G)$ has been processed, and a table $Answer$ for maintaining in $Answer[A, F]$ the result of $TEST(A, F, Gra[A, F])$.

$TEST(A, F, G)$: **forall** A, F **do** $Gram[A, F] := \mathbb{1}$ **od**;
if $Gram[A, F] \sqsubseteq G$ **then return** $Answer[A, F]$
else $Gram[A, F] := Gram[A, F] \sqcap G$;
  if $S \downarrow [F] = x$ **then** $Q :=$ (A knows x)
  elsif $S \downarrow [F] = F'(t_1, \ldots, t_m)$ **then** $Q := TEST(A, F', [\![call([F])]\!]G)$

elsif $S \downarrow [F] = f(t_1, \ldots, t_p)$
then $Q := (G$ has a rule $A \to f(A_1, \ldots, A_p)) \wedge$
$(\forall i = 1, \ldots, p\ TEST(A_i, F_i, [\![call([F].i)]\!]G)$,
where F_i is the main symbol of the term t_i
fi
$Answer[A, F] := Q$; **return** Q
fi

We can assume that each of the original schemes contains no more than one projective definition, hence the number of minimal critical symbols of the scheme $\hat{S}_1$ is no greater than $O(n)$. For a critical symbol $K_{i,j}$, there exists a simple (with pairwise disjoint edges) path w to the node $[K_{i,j}]$ in $Graph(\hat{S}_1)$. An application of a transformer $[\![call(a)]\!]$ can increase the grammar's size by no more than the sum of actual parameter size of the call at address a, thus $|\Phi_w(\mathbf{O})| \leq O(n)$ holds. The applications of the meet operation can increase this size by a factor of k, so a reduced grammar $\mu K_{i,j}$ in the stationary marking μ has a size bound $O(kn)$. Let us approximate the time needed for execution of a call $TEST(A, F, G)$. This call leads to no more than $O(n^3)$ further calls of $TEST$. Using the algorithm described in [5] for the acyclic congruence closure we can perform the operations $[\![call(.)]\!]$, $\sqcap$, and $\sqsubseteq$ for grammars of size $O(kn)$ in $O(kn)$ time. The total amount of time for executing $TEST(A, F, G)$ is $O(n^5)$. Summing up the time bounds for all algorithm steps we get the upper time bound $O(n^6)$ for the complexity of the algorithm deciding the tree equivalence of linear recursion schemes. □

12 Quasi-linear schemes

Consider a class of *quasi-linear* recursion schemes, which differ from the linear ones only in that actual parameters may contain the constant ω. We can transform each quasi-linear scheme S into a linear one by

- replacing each subterm t of an actual parameter by ω if $t \approx \omega$, and
- for each definition $F(x_1, \ldots, x_n) \Leftarrow \tau$ of the scheme S, and for every partition of the set $\{1, \ldots, n\} = \Delta \cup \Delta'$ into two non-intersecting subsets $\Delta \neq \emptyset$ and Δ', we introduce a useless definition

$$F_\Delta(x_{j_1}, \ldots, x_{j_{n-k}}) \Leftarrow [\omega/x_{i_1}, \ldots, \omega/x_{i_k}]\,\tau$$

of a new symbol F_Δ where $\Delta = \{i_1, \ldots, i_k\}, \Delta' = \{j_1, \ldots, j_{n-k}\}$, and

- repeatedly applying the *simple fold/unfold* rule to replace each call

$$F(t_1, \ldots, t_n) \text{ by the call } F_\Delta(t_{j_1}, \ldots, t_{j_{n-k}})$$

where $\Delta = \{i | t_i = \omega\}$;

- finally, deleting all useless definitions of the scheme obtained.

Thus, the transformation system Σ_{lin} will be complete also for the tree equivalence of quasi-linear schemes, and we obtain an algorithm deciding the tree equivalence of quasi-linear schemes. However, from the described reduction we cannot derive a polynomial bound for the complexity of this algorithm because of the exponential size growth in the transformation of quasi-linear schemes into linear ones.

13 Conclusion

In [9], B. Rosen described a technique that reduces the tree equivalence problem in subclasses of recursion schemes to the equivalence problem in some subclasses of context-free grammars. In this reduction, the scheme determinant is encoded by the words of the modelled grammar.

Our method to decide the tree equivalence problem reminds the *Parallel Stacking* and *Alternate Stacking* techniques described by L. G. Valiant [12] and used for solving the equivalence problem for subclasses of (non-singular) DPDAs. In these techniques, the two DPDA's are simulated simultaneously using one stack. Alternate Stacking involves simulating two DPDA's A_1 and B_2 with one non-deterministic PDA C whose stack contents $a_1 b_1 \dots a_n b_n$ are encodings of the stack contents $a_1 \dots a_n$ and $b_1 \dots b_n$ for A and B, respectively. The non-deterministic stack machine C accepts an input iff A and B are inequivalent. Since the emptyness of C is decidable, it follows that the equivalence of A and B is decidable. This technique is only successful if the top stack segments can be kept uniformly bounded.

The tree equivalence problem for the whole class of recursion schemes is inter-reducible to the equivalence problem for deterministic pushdown automata [3, 4, 6].

Acknowledgements

Many thanks go to Andrei Sabelfeld for his supportive critic and help. I would also like to thank Michaela Huhn for the proof reading and the anonymous referee for the constructive comments.

References

[1] deBakker J., Scott Dana. A theory of programs, August 1969, unpublished report.

[2] Beeri C. An improvement on Valiant's decision procedure for equivalence of deterministic finite-turn pushdown machines. – Theoretical Comput. Sci. – 1976. – No 3. – P. 305–320.

[3] Courcelle B. On jump deterministic pushdown automata.– Math. Systems Theory, 1977. – Vol. 11. – P. 87–109.

[4] Courcelle B. A representation of trees by languages.– Theoretical Comput. Sci. 1978, – Vol. 6. – P. 225–279; Vol. 7. – P. 25–55.

[5] Downey P.J., Sethi R., Tarjan R. J. Variations on the common subexpression problem.–J.ACM, 1980. – Vol. 27. – No 4. – P. 758–771.

[6] Gallier J.H. DPDA's in 'Atomic normal form' and applications to equivalence problems.–Theoretical Comput. Sci.– 1981. – Vol. 14. – No 2. – P. 155–186.

[7] Garland S.J., Luckham D.C. Program schemes, recursion schemes and formal languages.– Report UCLA-ENG-7154, June 1971, School of Engineering and Applied Science, University of California, Los Angeles.

[8] Kam J.B., Ullman J.D. Monotone data flow analysis frameworks.–Acta Informatica. – 1977. – Vol. 7. – No 3. – P. 305–318.

[9] Rosen B.K. Program equivalence and context-free grammars.–JCSS. – 1975. Vol. 11. – P. 358–374.

[10] Sabelfeld V.K. Tree equivalence of linear recursive schemata is polynomial-time decidable.–Information Processing Letters. – 1981. – Vol. 13. – No 4. – P. 147–153.

[11] Sabelfeld V.K. Equivalent Transformations of Recursion Schemes and Admissible Rewriting Rules.–Novosibirsk, 1993. – 20p. – (Prepr. /ISI SD RAS; No 20).

[12] Valiant L.G. Decision procedures for families of deterministic pushdown automata, – Ph. D. Thesis, University of Warwick, U.K., 1973.

[13] Valiant L.G. The Equivalence problem for deterministic finite-turn pushdown machines.–Information and Control. – 1975. – No 5. – P. 123–133.