Equivalent Transformations of Recursive Programs with Finite Unfolding *

Andrei Sabelfeld
Chalmers University of Technology
Göteborg, Sweden
andrei@cs.chalmers.se

Viktor Sabelfeld
Karlsruhe University
Karlsruhe, Germany
sabelfel@ira.uka.de

Abstract

We study equivalence relations for so-called finite recursive programs (programs with Finite Unfolding). The paper describes the universal interpretation domain and constructs complete systems for the tree and functional equivalence of finite recursive programs. We obtain the upper bound $C^{2^{O(n)}}$ for the size of the determinant of a finite program of size n and show that this bound is also the lower bound.

1 Introduction

The problems of analysis, synthesis, optimisation, verification and debugging of functional programs need a theory of transformations that would allow us to formally describe correct transformations of such programs. Back in the 1960s J. McCarthy started the study of equivalent transformations of functional programs, his work was concerned with the language **LISP**. We aim at constructing a theory that would not be too dependent on the semantic constructions of a particular programming language.

In this paper, the universal model of functional programs, namely, recursive programs, is considered. Two sorts of equivalence are investigated: tree (the canonical forms of unfoldings, determinants, coincide) and functional (the programs result equally for all admissible interpretations).

We consider a wider class of recursive programs than that investigated in the literature, beginning with the classical work of Garland and Luckham

*This paper appeared in *Programming and Computer Software*, 23 (2):70–77, 1997.

[5]. The conditions of conditional terms allow for not only atomic formulas, as was usually assumed in earlier investigations, but also terms of the general form. Besides this, a program limits the set of possible interpretations of any basic symbol by assigning to it a collection of strict parameters: an interpretation should give "undefined" if any strict set contains at least one undefined argument.

We consider the subclass of finite recursive programs, i.e., programs with a finite unfolding. In the paper, we prove the determinant size of such programs to be bounded by $C^{2^{O(n)}}$, where C is the maximum arity of basic symbols involved in the program and n is the program size. We construct complete transformation systems for the tree and functional equivalence relations of finite programs. For tree equivalence, the system consists of three rule schemes: deletion (insertion) of unused definitions, folding (unfolding) and replacement of hopeless terms. For functional equivalence, three more rule schemes are added: context-sensitive replacement, reduction of conditional terms and deletion (insertion) of an unessential test. The algorithms for recognising the functional and tree equivalence of finite recursive programs are based on the application of the formulated transformation rules.

Note, that the problem of tree equivalence is in the general case equivalent to the well-known open problem of equivalence of the deterministic push-down automata, and the problem of functional equivalence is undecidable. One can prove that for finite programs the latter problem is NP-hard.

2 Recursive Programs

Let $\mathcal{X} = \{x, y, z, \dots\}$ be a set of variables, $\mathcal{F}_b = \{\omega, f, g, h, \dots\}$ a set of *basic symbols*, $\mathcal{F}_d = \{F, F_1, F_2, \dots\}$ a set of *defined symbols*, and $\mathcal{F} = \mathcal{F}_b \bigcup \mathcal{F}_d$. Let $r(s)$ denote the *arity* (the number of argument positions) of a symbol $s \in \mathcal{F}$ and $r(\omega) = 0$. By $\mathcal{T}(\mathcal{P}, X)$ we denote the set terms over the functional symbols from the set $\mathcal{P}$ and variables from $X, \mathcal{T} = \mathcal{T}(\mathcal{F}_b, \mathcal{X})$ is the set of basic terms. Basic symbols of arity zero are also called constants. Each basic symbol of a nonzero arity is associated with a nonempty subset $Strict(f)$ of the set $2^{\{1,\dots,r(f)\}}$, which is called a collection of *strict sets of parameters* [3] of the symbol f and limits the set of its possible interpretations. That is, an interpretation of a term $f(d_1, \dots, d_{r(f)})$ is undefined if for each $\Delta \in Strict(f)$ there exists i from Δ such that the value d_i is undefined. The notation $\alpha \prec \beta$ stands for the assertion "term α is a subterm of β".

A *recursive program* is a pair $S = \langle e; DEF \rangle$, where e is a term called an *entry* of the program and DEF is a finite set of definitions of symbols from $\mathcal{F}_d$. A *definition* of a symbol $F \in \mathcal{F}_d$ has the following form.

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

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

Example:

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

The special constant ω will be interpreted only by value "undefined" and, thus will also be called "undefined". Below we use the terminology and denotations introduced in [3].

3 Universal domain and equivalence

A *domain* is a triple $\langle D, \leq, \perp \rangle$, where D is a set, $\perp \in D$ (the "bottom") and $\leq$ is a partial order on D that satisfies the following two conditions:

- $\forall\, d \in D \ \ \perp \leq d.$
- *Completeness condition:* for each chain C in D the *least upper bound* $lub(C)$ belongs to D. ($d \leq lub(C)$ 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.)

Let us construct the *universal domain* $\mathcal{U}$ of all finite and infinite regularly constructed terms. Let T_f be the set of all terms under the signature $\langle \mathcal{X}, \mathcal{F}_b \rangle$ that contains all constants from $\mathcal{F}_b$ and all variables from $\mathcal{X}$ and is closed under application of the rule

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

By $t[N \leftarrow \tau]$ we denote the term obtained from a term t by substituting the term τ for all occurrences of the subterms with addresses from the set N. We introduce a partial order $\sqsubseteq$ on T_f by setting $t_1 \sqsubseteq t_2$ iff there exists a set N of mutually independent addresses in t_2 such that $t_1 = t_2[N \leftarrow \omega]$.

An arbitrary term t that does not contain calls to defined symbols can be *reduced* by repeatedly applying to it the rules from the set

$$R = \{f(t_1, \ldots, t_n) \to \omega \mid f \in \mathcal{F}_b \& \ \forall\ \Delta \in Strict(f)\ \exists\ i \in \Delta\ t_i = \omega\}$$

while applicable. The result of reduction of a term t is denoted by $Red(t)$. Obviously, the reduced term belongs to the set T_f. Consider the set $\tilde{T}_f$ of infinite chains of the form $t_0 \sqsubseteq t_1 \sqsubseteq \ldots \sqsubseteq t_i \sqsubseteq \ldots$ with elements $t_i \in T_f$. Each chain $s \in \tilde{T}_f$ corresponds to a unique, possibly infinite, tree-term $\bar{s}$ which can be defined by using special normal representations of the chains in $\tilde{T}_f$.

Define the *cutting* operator $Cut(t, k)$ of a term t to a depth $k \geq 0$ by

$$Cut(t, k) = t[N_t^k \leftarrow \omega],$$

where N_t^k is the set of addresses of all subterms of the term t at the depth k that are neither variables nor constants. Given a chain $s \in \tilde{T}_f$, we have $s = t_0 \sqsubseteq t_1 \sqsubseteq \ldots \sqsubseteq t_i \sqsubseteq \ldots$. Since s is monotone and the number of tree nodes at the depth i is finite, conclude

$$\forall i\ \exists k_i\ \forall j \geq k_i\ Cut(t_j, i) = Cut(t_{k_i}, i).$$

Using this property, define the *cutting a chains* to a depth i, $\overline{Cut}(s, i)$, as term $\overline{Cut}(s, i) = Cut(t_{k_i}, i)$. A sequence of terms $\bar{s}_i = Red(\overline{Cut}(s, i))$ for $i = 0, \ldots$ is called *normal representation* of the chain s. Since $\forall\ i\ \overline{Cut}(s, i) \sqsubseteq \overline{Cut}(s, i+1)$ for any chain $s \in \tilde{T}_f$ and the reduction operation preserves the order $\sqsubseteq$, we obtain $\forall\ i\ Red(\overline{Cut}(s, i)) \sqsubseteq Red(\overline{Cut}(s, i+1))$, i.e. the normal representation of any chain is also a chain from $\tilde{T}_f$. Let T_U be the set of normal representations of chains from $\tilde{T}_f$. Suppose that $\ll$ is an order relation defined elementwise by

$$\bar{s}^1 \ll \bar{s}^2 \stackrel{dfn}{\equiv} \forall\ i\ \bar{s}_i^1 \sqsubseteq \bar{s}_i^2.$$

Lemma 1 *$\langle T_U, \ll \rangle$ is a partially ordered set.*

Proof. The reflexivity, transitivity and antisymmetry of $\ll$ follow from the corresponding properties of the relation $\sqsubseteq$. □

Theorem 1 *The triple $\mathcal{U} = \langle T_U, \ll, \bar{\omega} \rangle$, where $\bar{\omega}$ is the infinite chain consisting of ω elements, is a domain.*

Proof. By the lemma, $\langle T_U, \ll \rangle$ is a partially ordered set. The $\bar{\omega}$ element is the bottom of T_U, since $\forall\ i\ \omega \sqsubseteq \bar{s}_i$ for any chain $\bar{s} = \{\bar{s}_i\}_{i=1}^{\infty}$. Let us now check the completeness condition. For a chain $\overline{s^0} \ll \overline{s^1} \ll \ldots \ll \overline{s^k} \ll \ldots$ of elements $\overline{s^j} = s_0^j \sqsubseteq \ldots \sqsubseteq s_i^j \sqsubseteq \ldots$ from T_U let us construct the least upper bound $\overline{q} = q_0 \sqsubseteq \ldots \sqsubseteq q_i \sqsubseteq \ldots \times T_U$. For this purpose, consider the sequences $r_i = \{s_i^j\}_{j=0}^{\infty}$ for $i = 0, \ldots$. Since $\overline{s^j} \ll \overline{s^{j+1}}$, each sequence r_i is a chain. Since all elements of the chain r_i are terms of depth no more than i, there exists an element in this chain $q_i = s_i^k$ such that $s_i^l \sqsubseteq s_i^k$ for all $l \geq 0$. Obviously, q_i is the least upper bound of the chain r_i with respect to the order $\sqsubseteq$. It is also clear that the constructed chain $\overline{q} = q_0 \sqsubseteq \ldots \sqsubseteq q_i \sqsubseteq \ldots$ is a normal representation of itself. The fact that $\overline{q}$ is the least upper bound of the chain $\overline{s^0} \ll \ldots \ll \overline{s^k} \ll \ldots$ wrt $\ll$ follows from the fact that q_i is the least upper bound of the chain r_i wrt $\sqsubseteq$. $\Box$

Let D and D' be domains. A function $\varphi : D \to D'$ is called *monotone* if $\forall\, d, d' \in D\ d \leq d' \Rightarrow \varphi(d) \leq \varphi(d')$. A monotone function is called *continuous* if it preserves the least upper bounds of chains, i.e. $\varphi(lub(L)) = lub(\varphi(L))$.

An *interpretation* I fixes a domain D associates

- each variable $x \in \mathcal{X}$ with an element $I(x) \in D$ and
- each symbol f from $\mathcal{F}_b$ with a continuous function $I(f) : D^{r(f)} \to D$ satisfying the following *strictness* condition.

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

By structural induction, extend the definition of interpretation I to arbitrary terms:

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

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

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

An important example of interpretation is the *universal interpretation* J with the domain $\mathcal{U}$ which is defined as follows. Let $J(x) = x$ for the 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

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

Finally, the *determinant* of a program S is defined as the limit of the sequence of *determinant approximations* $J(\pi^n e)$ by

$$det(S) = lub(\{J(\pi^n e) : n \geq 0\}),$$

where e is the entry and J is the universal interpretation. The mapping π corresponds to the rule of "parallel outermost computation rule". It performs one step unfolding of all outermost calls in a term. It is defined by $\pi^0 t = t$, $\pi^n = \pi^{n-1}\pi$ for $n > 0$.

For the example program

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

the first elements of the sequence of determinant approximations are

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

Two recursive programs S_1 and S_2 are *tree equivalent* (written $S_1 \approx S_2$, cf. [6]), if $det(S_1) = det(S_2)$.

An interpretation I is *admissible* if its domain D_I contains at least two distinct maximum elements $\mathbf{F}$ and $\mathbf{T}$, different from $\perp$, and the set of basic symbols $\mathcal{F}_b$ contains two constants **false**, **true** and the special ternary symbol **if**, such that $I(\mathbf{false}) = \mathbf{F}, I(\mathbf{true}) = \mathbf{T}$ and $I(\mathbf{if}) = cond$, where

$$cond(d_1, d_2, d_3) = \begin{cases} d_2, & \text{if } d_1 = \mathbf{T} , \\ d_3, & \text{if } d_1 = \mathbf{F} , \\ \perp & \text{otherwise.} \end{cases}$$

Denote the set of all admissible interpretations by $\mathcal{I}$. Two programs S_1 and S_2 are *functionally equivalent* (written $S_1 \sim S_2$) if

$$\forall\, I \in \mathcal{I}\ val(S_1, I) = val(S_2, I),$$

where $val(S, I) = lub(\{I(\pi^n e) : n \geq 0\})$, and e is the entry of the program S.

4 Transformations for the Tree Equivalence

The notions of *fragment, fragment occurrence, transformation rule and the scheme of a transformation* are borrowed from [3].

4.1 Deletion of Unused Definitions

The first rule scheme $L1$ (*Deletion of Unused Definitions*) contains all rules of the form $G_1 \leftrightarrow G_2$, where G_1 is a fragment without entries and G_2 is an empty fragment.

Example:

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

4.2 Folding and Unfolding

Suppose G is a fragment that contains exactly one definition

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

and a is the address of an occurrence of a call $F(t_1, \dots, t_n)$ in a term outside the body of the symbol F. Let $G' = G[a \leftarrow \sigma t]$, where $\sigma = [t_1/x_1, \dots, t_n/x_n]$ (the term $[t_1/x_1, \dots, t_n/x_n]t$ is obtained from the term t by simultaneously substituting all occurrences of the variable x_i by the term $t_i, i = 1, \dots, n$). Then $G \leftrightarrow G'$ belongs to the scheme $L2$ (*Folding and Unfolding*).

Example:

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

4.3 Replacement of Hopeless Terms

An address a in a fragment G is *hopeless* if it satisfies the *tree-emptiness condition* $G/a \approx \omega$ (G/a stands for the term at the address a in the fragment G). In order to find hopeless addresses, one can formulate a problem of forwards semantic analysis of the fragment G as in [2]. The semilattice of properties has only two elements: $L_{st} = \{\bot, \top\}$ so that $\bot \sqcup \top = \top$, $\bot \sqcup \bot = \bot$, $\top \sqcup \top = \top$. Interpret the $\top$ value as "tree-nonemptiness" and the $\bot$ value as "tree-emptiness".

The rule scheme $L3$ (*Replacement of Hopeless Terms*) contains all rules of the form $G \leftrightarrow G[a \leftarrow \omega]$, where a is a hopeless address of the fragment G. It includes, in particular, the rules

$$\{f(t_1, \dots, t_n) \leftrightarrow \omega |\ f \in \mathcal{F}_b, \forall\ \Delta \in Strict(f)\ \exists\ i \in \Delta\ t_i = \omega\}.$$

Example: If $Strict(f) = Strict(g) = Strict(h) = \{\{1\}\}$, then

$$\left\langle \begin{array}{l} F(u,\omega); \\ F(x,y) \Leftarrow g(hy, F(fx,hy)) \end{array} \right\rangle \leftrightarrow \left\langle \begin{array}{l} \omega; \\ F(x,y) \Leftarrow g(hy, F(fx,hy)) \end{array} \right\rangle$$

If $Strict(g) = \{\{1,2\}\}$, then $g(\omega,t) \leftrightarrow \omega$; $\mathbf{if}(\omega,s,t) \leftrightarrow \omega$, $\mathbf{if}(t,\omega,\omega) \leftrightarrow \omega$.

Consider the system of transformations Σ_t consisting of all rules generated by thr schemes $L1$–$L3$. It is easy to prove that the transformations from Σ_t preserve tree-equivalence, i.e. we have

Lemma 2 $S_1 \overset{\Sigma_t}{\leftrightarrow} S_2 \ \Rightarrow \ S_1 \approx S_2$.

5 The Class of Finite Recursive Programs

A recursive program S is *finite* if its determinant $det(S)$ is finite.

A recursive program is *reduced* if

- it does not contain any definitions and
- its entry does not contain tree-empty subterms different from ω.

Lemma 3 *The property of finiteness of a program S is decidable. There exists an algorithm that recognises finiteness of a program and, if it is finite, transforms the program into a reduced one by transformations from Σ_t.*

Proof. The labelling algorithm that finds all hopeless addresses of a program is described in [1]. Thus, by applying the Replacement of Hopeless Terms rule, one can replace all terms of a program at hopeless addresses by the term ω. Further, from the obtained program S' construct its *graph of calls* $G_{S'}$ whose nodes are the entry and all definitions of the program S. An edge connects the entry e or a definition $F(\dots) \Leftarrow e$ with a definition B iff the term e contains the symbol defined in B.

Obviously, $S \approx S'$ and the program S' is finite iff the graph $G_{S'}$ contains no cycles accessible from the node corresponding to the entry.

If S' is finite, eliminate all calls by applying the unfolding transformation and delete all unused definitions. The acyclicity of $G_{S'}$ guarantees the termination of this process. As a result, the initial program S is transformed into a reduced program $S^\star$ such that $S \approx S^\star$ in accordance with Lemma 2.
□

Example: Let

$$S = \langle F(a,\omega,b); F(x,y,z) \Leftarrow f(F(y,z,x),z)\rangle.$$

If $Strict(f) = \{\{1,2\}\}$, then the substitution of ω for all occurrences of terms at hopeless addresses gives the program $S' = \langle \omega; F(x,y,z) \Leftarrow \omega$, whose graph of calls has no edges. In case $Strict(f) = \{\{1\},\{2\}\}$, the initial program has no hopeless address. Thus, $S' = S$ and the graph of calls has a cycle going through the definition of the symbol F and accessible from the entry. This means that in the latter case the program S is not finite.

5.1 The expansion theorem

Theorem 2 *The size of the determinant of a finite program of size n does not exceed C^{2^n+1}, where $C = \max(2,c)$ and c is the maximum arity of all basic symbols occurring in the program. This bound is achievable in the sense that for any n there exists a program of size n whose determinant has the size $2^{2^{\frac{n-2}{5}}} - 1$.*

Proof. We can assume all terms of program S at hopeless addresses are ω, since the corresponding transformation does not increase the program's size. By induction on the number of defined symbols l of S, let us establish that for the depth $h(det(S))$ of the determinant of S, it holds that $h(det(S)) \leq 2^n (n = |S|)$.

For a program without definitions $h(det(S)) \leq |det(S)| = |S| = n \leq 2^n$.

Suppose the statement holds for all programs with no more than l definitions. Assume S has $l+1 > 0$ definitions. Since S is finite, it has at least one definition $F(x_1, \ldots, x_n) \Leftarrow \tau$ of some symbol F in whose body there are no calls. Consider the program S' obtained from S by deleting this definition and declaring the symbol F as a basic one. Let $|\tau| = m < n$, then $|S'| \leq n - m$. By the induction hypothesis, we have $h(det(S')) \leq 2^{n-m}$. The determinant of S can be obtained from $det(S')$ by unfolding all calls to F and reducing the result of unfolding. Thus, $h(det(S)) \leq h(det(S')) \cdot m \leq 2^{n-m} m \leq 2^{n-m} n \leq 2^n$. Since the depth of the tree $det(S)$ does not exceed 2^n and each node has no more than c sons, conclude $|det(S)| \leq C^{2^n+1}$.

Consider the program

$$S_m = \left\langle \begin{array}{l} F_m a; \\ F_i x \Leftarrow F_{i-1} F_{i-1} x, 1 \leq i \leq m \\ F_0 x \Leftarrow f(x,x). \end{array} \right\rangle$$

Let us denote by $\varphi_i(x)$ the determinant of the program obtained from S_m by replacing the entry by the term $F_i x, 0 \leq i \leq m$. Obviously, the terms $\varphi_i(x)$ satisfy the equalities $\varphi_0(x,x) = f(x,x)$ and $\varphi_i(x) = \varphi_{i-1}(\varphi_{i-1}(x))$ for $i = 1, \ldots, m$. Let us show by induction on i that the term $\varphi_i(x,x)$ is a complete binary tree under the signature $\langle\{x\},\{f\}\rangle$ with 2^{2^i} leaves x. Indeed, $\varphi_0(x,x) = f(x,x)$. If the term $\varphi_{i-1}(x,x)$ has exactly $2^{2^{i-1}}$ occurrences of x, then after replacing them by $\varphi_{i-1}(x,x)$ we will have in the term $\varphi_i(x,x) = \varphi_{i-1}(x,x)[\varphi_{i-1}(x,x)/x]$ exactly $2^{2^{i-1}} \cdot 2^{2^{i-1}} = 2^{2^i}$ leaves. Hence, the term $det(S_m) = \varphi_m(a)$ has exactly 2^{2^m} leaves and $2^{2^m+1} - 1$ nodes. Since $|S_m| = n = 5m + 7$, we have $|det(S_m| = 2^{2^{\frac{n-7}{5}+1}} - 1 = 2^{2^{\frac{n-2}{5}}} - 1.\square$

5.2 Completeness of the System Σ_t

The algorithm for recognising the tree equivalence of finite programs S_1 and S_2 consists of converting the initial programs by applying transformations from Σ_t into reduced programs (terms) e_1, e_2. It only remains to check whether the reduced programs coincide.

$$S_1 \approx S_2 \;\Leftrightarrow\; S_1^\star \approx S_2^\star \;\Leftrightarrow\; e_1 \approx e_2 \;\Leftrightarrow\; e_1 = e_2.$$

Theorem 3 (Completeness Theorem) $S_1 \overset{\Sigma_t}{\leftrightarrow} S_2 \;\Leftrightarrow\; S_1 \approx S_2$ *for finite* S_1, S_2.

Proof. Lemma 2 gives the forward implication. It remains to show to show how to transform arbitrary tree equivalent programs one into another. The algorithm for recognising tree equivalence was formulated in such a way that when applied to an arbitrary pair of programs, it yields the answer **Yes** along with the sequence of transformations from Σ_t converting one program into the other. $\square$

6 Transformations Preserving Functional Equivalence

6.1 Context Replacement

The rule scheme $F1$ (*Context Replacement*) allows one to eliminate redundant tests in some conditional terms. For example,

$$\mathbf{if}(px, \mathbf{if}(px, \alpha, \beta), \gamma) \sim \mathbf{if}(px, \alpha, \gamma).$$

Here, checking the condition $p(x)$ in the then-alternative of the conditional in the left-hand side is redundant. The transformation formulation also takes into account the fact that if a term σ is "more defined" than the term α, i.e. $\alpha \sqsubseteq \sigma$, and $I(\alpha) \in \{\mathbf{true}, \mathbf{false}\}$, then $I(\sigma) = I(\alpha)$.

The rule scheme *Context Replacement* contains all rules of the form

$$\{\mathbf{if}(\alpha, \beta, \gamma) \leftrightarrow \mathbf{if}(\alpha, \beta[a \leftarrow \mathbf{true}], \gamma) \mid \alpha \sqsubseteq \beta/a\}, \text{ and}$$

$$\{\mathbf{if}(\alpha, \beta, \gamma) \leftrightarrow \mathbf{if}(\alpha, \beta, \gamma[b \leftarrow \mathbf{false}]) \mid \alpha \sqsubseteq \gamma/b\},$$

where a and b are addresses of some subterms in β and γ, respectively.
Examples:

$$\mathbf{if}(\alpha, \mathbf{if}(\alpha, \beta, \gamma), \delta) \leftrightarrow \mathbf{if}(\alpha, \mathbf{if}(\mathbf{true}, \beta, \gamma), \delta),$$

if $Strict(g) = \{\{1\}, \{2\}\}$, then

$$\mathbf{if}(g(\omega, x), g(y, x), z) \leftrightarrow \mathbf{if}(g(\omega, x), \mathbf{true}, z).$$

6.2 Reduction of Conditional Terms

The rule scheme $F2$ (*Reduction of Conditional Terms*) contains all rules of the form

$$\mathbf{if}(\mathbf{true}, \beta, \gamma) \leftrightarrow \beta, \quad \mathbf{if}(\mathbf{false}, \beta, \gamma) \leftrightarrow \gamma,$$

where β and γ are arbitrary terms.

6.3 Deletion of an Unessential Test

Consider the following relation on terms.

$$\beta < \alpha \overset{def}{=} \forall\, I \in \mathcal{I}\ (I(\beta) \neq \perp \Rightarrow\ I(\alpha) = \mathbf{T} \vee I(\alpha) = \mathbf{F}).$$

It is a decidable relation, since it can be checked, for example, by verifying the implication for a finite set of finite interpretations.

The rule scheme $F3$ (*Deletion of an Unessential Test*) contains all rules of the form $\mathbf{if}(\alpha, \beta, \beta) \leftrightarrow \beta$, where $\beta < \alpha$.

Example:

$$\mathbf{if}(p, x, y) \ \leftrightarrow\ \mathbf{if}(p, \mathbf{if}(p, x, y), \mathbf{if}(p, x, y)).$$

Let Σ_f be the transformation system containing all rules generated by the rule schemes $L1$ — $L3$ and $F1$ — $F3$.

7 Completeness of the System Σ_f

It is easy to prove that applying rules from Σ_f preserves functional equivalence, i.e. we have

Lemma 4 $S_1 \overset{\Sigma_f}{\leftrightarrow} S_2 \;\Rightarrow\; S_1 \sim S_2$.

A reduced program (term) t is *normal*, if

- each of its subterms $f(t_1, \dots, t_n)$ with an outer symbol $f \neq \mathbf{if}$ does not contain subterms σ such that $f(t_1, \dots, t_n) < \sigma$;
- each of its subterms $\mathbf{if}(\alpha, \beta, \gamma)$ does not contain subterms σ of the term α different from α such that $\mathbf{if}(\alpha, \beta, \gamma) < \sigma$; and
- the conditions of all conditional subterms of the term t are different from the constants **true** and **false**.

Lemma 5 *Any finite program can be converted into a functionally equivalent normal term by applying the rules from* Σ_f.

Proof. Let us first reduce the initial program and eliminate constant conditions of conditional terms by applying $F3$. If after doing so there is an occurrence of a term t in the program such that $t = f(t_1, \dots, t_n), f \neq \mathbf{if}, \sigma \prec t$ or $t = \mathbf{if}(\alpha, \beta, \gamma), \sigma \prec \alpha, \sigma \neq \alpha$ for some subterm σ of t, then by applying $F3$ replace this occurrence of t by the term $\mathbf{if}(\sigma, t, t)$ to which we repeatedly apply the following two transformations (in any order, while possible).

- $\mathbf{if}(\sigma, t, t) \xrightarrow{F1} \mathbf{if}(\sigma, t[N_\sigma \leftarrow \mathbf{true}], t[N_\sigma \leftarrow \mathbf{false}])$, where N_σ is the set of addresses of all occurrences of the term σ in the term t.
- By applying $\mathbf{if}(\mathbf{true}, t_1, t_2) \longrightarrow t_1$ and $\mathbf{if}(\mathbf{false}, t_1, t_2) \longrightarrow t_2$ eliminate constant conditions of conditional terms in each of the alternatives of the obtained term.

It is clear that this process will terminate, since the term size monotonely decreases with each transformation step. □

Example:

For the term $t = \mathbf{if}(\mathbf{if}(\sigma, \beta, \gamma), \mu, \nu)$ it holds that $t < \sigma$. Therefore, its normalisation yields

$$t \xrightarrow{F3} \mathbf{if}(\sigma, \mathbf{if}(\mathbf{if}(\sigma, \beta, \gamma), \mu, \nu), \mathbf{if}(\mathbf{if}(\sigma, \beta, \gamma), \mu, \nu)) \quad \xrightarrow{F1}$$
$$\mathbf{if}(\sigma, \mathbf{if}(\mathbf{if}(\mathbf{true}, \beta, \gamma), \mu, \nu), \mathbf{if}(\mathbf{if}(\mathbf{false}, \beta, \gamma), \mu, \nu))$$
$$\xrightarrow{F1} \mathbf{if}(\sigma, \mathbf{if}(\beta, \mu, \nu), \mathbf{if}(\gamma, \mu, \nu)).$$

Example:

If $Strict(g) = \{\{1\}, \{2\}\}$, then for the term $t = \mathbf{if}(g(\mathbf{if}(g(x, y), x, x), y), u, v)$ it holds that $t < g(x, y)$, and thus

$$t \xrightarrow{F3} \mathbf{if}(g(x, y), \mathbf{if}(g(\mathbf{if}(g(x, y), x, x), y), u, v), \mathbf{if}(g(\mathbf{if}(g(x, y), x, x), y), u, v))$$
$$\xrightarrow{F1} \mathbf{if}(g(x, y), \mathbf{if}(g(\mathbf{if}(\mathbf{true}, x, x), y), u, v), \mathbf{if}(g(\mathbf{if}(\mathbf{false}, x, x), y), u, v))$$
$$\xrightarrow{F3} \mathbf{if}(g(x, y), \mathbf{if}(g(x, y), u, v), \mathbf{if}(g(x, y), u, v))$$
$$\xrightarrow{F1} \mathbf{if}(g(x, y), \mathbf{if}(\mathbf{true}, u, v), \mathbf{if}(\mathbf{false}, u, v))$$
$$\xrightarrow{F3} \mathbf{if}(g(x, y), u, v).$$

Informally, in normal terms, the condition of any conditionals subterm is minimum in the sense that it does not include subterms that are checked during any execution of this conditional.

Let us describe the algorithm below for recognising functional equivalence of finite programs S_1 and S_2. We will be applying it to a pair of normal terms s and t obtained from the initial programs by reducing and consequent normalising of the entries. The algorithm is recursive: in order to determine whether $s \sim t$, it calls itself with the queries $s' \sim t'$ for some term pairs s' and t' such that either $h(s') < h(s)$ or $h(s') = h(s)$ and $h(t') < h(t)$. The notation $[\alpha \triangleleft \beta] : \mathcal{T} \to \mathcal{T}$ in the algorithm description stands for the following transformation of terms: the term $[\alpha \triangleleft \beta]\gamma$ is obtained from the term γ by repeatedly applying the following three transformations while at least one of them is applicable.

- All subterms τ such that there exists a term σ such that $\sigma \sim \alpha$ and $\sigma \sqsubseteq \tau$ are replaced by the term β.
- By applying the rules $\mathbf{if}(\mathbf{true}, t_1, t_2) \longrightarrow t_1$ and $\mathbf{if}(\mathbf{false}, t_1, t_2) \longrightarrow t_2$, conditionals with constant conditions are reduced.
- By applying the ruled $L3$, tree-empty subterms are replaced by ω.

Here, α is an arbitrary term and $\beta \in \{\mathbf{true}, \mathbf{false}\}$.

The recognising algorithm.

$s \sim t \;=\; (s = t)$, if s and t are constants or variables.

$$\begin{array}{rl|l} \omega \sim \mathbf{if}(\alpha,\beta,\gamma) = & \omega \sim \alpha \vee & \mathbf{if}(\alpha,\beta,\gamma) \xleftrightarrow{L3} \omega \\ & \omega \sim \beta \;\&\; \omega \sim \gamma \vee & \mathbf{if}(\alpha,\omega,\omega) \xleftrightarrow{L3} \omega \\ & \mathbf{true} \sim \alpha \;\&\; \omega \sim \beta \vee & \mathbf{if}(\mathbf{true},\omega,\beta) \xleftrightarrow{F2} \omega \\ & \mathbf{false} \sim \alpha \;\&\; \omega \sim \gamma & \mathbf{if}(\mathbf{false},\alpha,\omega) \xleftrightarrow{F2} \omega \end{array}$$

$\omega \sim f(t_1,\ldots,t_n) = \forall\, \Delta \in Strict(f)\; \exists\, i \in \Delta\; \omega \sim t_i \quad | \quad f(t'_1,\ldots,t'_n) \xleftrightarrow{L3} \omega$, where $t'_i = \omega$, if $\omega \sim t_i$ and $t'_i = t_i$ otherwise.

If s is a constant ($s \neq \omega$) or a variable, and t is an arbitrary normal term, then $s \sim t = \; (s = t)$.

If $f \neq \mathbf{if}$ and $g \neq \mathbf{if}$, then

$$\begin{array}{rl} f(t_1,\ldots,t_n) \sim g(t'_1,\ldots,t'_m) = & \omega \sim f(t_1,\ldots,t_n) \;\&\; \omega \sim g(t'_1,\ldots,t'_m) \vee \\ & f = g \;\&\; \forall\, i\ (1 \le i \le n)\ t_i \sim t'_i \end{array}$$

If $f \neq \mathbf{if}$ and $t = f(t_1,\ldots,t_n)$, then $t \sim \mathbf{if}(\alpha,\beta,\gamma)$ does not hold, because assuming $t \sim \mathbf{if}(\alpha,\beta,\gamma)$ implies $t < \alpha$, which contradicts the fact that t normal.

$\mathbf{if}(\alpha,\beta,\gamma) \sim \mathbf{if}(\sigma,\mu,\nu) =$

$$\begin{array}{c} h(\mathbf{if}(\sigma,\mu,\nu)) < h(\mathbf{if}(\alpha,\beta,\gamma)) \;\&\; \mathbf{if}(\sigma,\mu,\nu) \sim \mathbf{if}(\alpha,\beta,\gamma) \;\vee \\ \omega \sim \mathbf{if}(\alpha,\beta,\gamma) \;\&\; \omega \sim \mathbf{if}(\sigma,\mu,\nu) \;\vee \\ \alpha \sim \sigma \;\&\; [\rho \lhd \mathbf{true}]\beta \sim [\rho \lhd \mathbf{true}]\mu \;\&\; [\rho \lhd \mathbf{false}]\gamma \sim [\rho \lhd \mathbf{false}]\nu \;\vee \\ \mathbf{if}(\alpha,\beta,\gamma) < \sigma \;\&\; [\sigma \lhd \mathbf{true}]\mathbf{if}(\alpha,\beta,\gamma) \sim [\sigma \lhd \mathbf{true}]\mu \;\& \\ [\sigma \lhd \mathbf{false}]\mathbf{if}(\alpha,\beta,\gamma) \sim [\sigma \lhd \mathbf{false}]\nu. \end{array}$$

Here, ρ is the term with minimum depth among the terms α and σ. If the last disjunctive component is true (and all previous ones are false), the transformation is performed n the following way.

$$\begin{array}{c} t = \mathbf{if}(\alpha,\beta,\gamma) \xleftrightarrow{F3} \mathbf{if}(\sigma,t,t) \xleftrightarrow{recursion,F1,F2,L3} \mathbf{if}(\sigma,[\sigma \lhd \mathbf{true}]t,[\sigma \lhd \mathbf{false}]t) \\ \xleftrightarrow{recursion} \mathbf{if}(\sigma,[\sigma \lhd \mathbf{true}]\mu,[\sigma \lhd \mathbf{false}]\nu) \xleftrightarrow{recursion} \mathbf{if}(\sigma,\mu,\nu). \end{array}$$

Theorem 4 *Σ_f is a complete system of transformations for functional equivalence. i.e. $S_1 \overset{\Sigma_f}{\leftrightarrow} S_2 \;\Leftrightarrow\; S_1 \sim S_2$ for finite programs S_1, S_2.*

Proof. Lemma 4 gives the forward implication. It remains to show to show how to transform arbitrary functional equivalent programs one into another. However, in case the above algorithm for recognising functional equivalence finds a pair of programs to be equivalent, it also points to the sequence of rules from Σ_f converting one program into the other. □

Example:

For the terms $s = \mathbf{if}(p, \mathbf{if}(q, x, y), \mathbf{if}(q, u, v))$ and $t = \mathbf{if}(q, \mathbf{if}(p, x, u), \mathbf{if}(p, y, v))$ it holds that $s < q$, hence

$$s \xrightarrow{F3} \mathbf{if}(q, s, s) = \mathbf{if}(q, \mathbf{if}(p, \mathbf{if}(q, x, y), \mathbf{if}(q, u, v)), \mathbf{if}(p, \mathbf{if}(q, x, y), \mathbf{if}(q, u, v)))$$
$$\xrightarrow{F1} \mathbf{if}(q, \mathbf{if}(p, \mathbf{if}(\mathbf{true}, x, y), \mathbf{if}(\mathbf{true}, u, v)), \mathbf{if}(p, \mathbf{if}(\mathbf{false}, x, y), \mathbf{if}(\mathbf{false}, u, v)))$$
$$\xrightarrow{F2} \mathbf{if}(q, \mathbf{if}(p, x, u), \mathbf{if}(p, y, v)) = t.$$

8 Conclusion

In view of the estimate of the determinant size for finite programs proved in Section 5.1, it is unlikely that there exists a polynomial algorithm for recognising functional equivalence of such programs. There is even less hope for constructing a polynomial algorithm for recognising functional equivalence of finite programs, because the problem is NP-hard.

Among the rules of the transformation system Σ_f, the most complex rule is the deletion of an unessential test, because its application requires checking the condition $t < s$ for terms t and s. Note, if we restrict ourselves to the class of programs where the conditions of conditional terms are basic terms without conditionals, then $t < s$ can be checked in linear time.

References

[1] V.E. Kotov, V.K. Sabelfeld. The Theory of Program Schemes (In Russian), Moscow, Nauka, 1991.

[2] V.K. Sabelfeld. Transformations for Recursive Program Schemes, in Mixed Computations and Program Transformations. Novosibirsk, 1986, pp. 99–111.

[3] V.K. Sabelfeld. Tree Equivalence of Linear Recursive Schemes, Programmirovanie, 1994, pp. 5–26.

[4] Courcelle B., Vuillemin J. Semantics and axiomatics of a simple recursive language. Report No. 60, February 1974, Institute of Informatics and Automatics, Rocquencourt, France.

[5] 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

[6] Rosen B.K. Program equivalence and context-free grammars//JCSS. — 1975. Vol. 11, pp. 358–374.

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