Information Processing Letters 18 (1984) 275–283
North-Holland

18 June 1984

SOME EQUIVALENT TRANSFORMATIONS OF RECURSIVE PROGRAMS BASED ON THEIR SCHEMATIC PROPERTIES

M.B. TRAKHTENBROT *

Computing Center, Siberian Branch of the USSR Academy of Sciences, Novosibirsk 630090, U.S.S.R.

Communicated by A.P. Ershov
Received 30 September 1983
Revised 26 December 1983

Keywords: Recursive scheme, schematic properties, equivalent transformations

1. Introduction

A great interest in recursive program transformations during the last decade is mainly due to the transformational approach to reliable and effective program construction [1,2]. According to this approach, construction of a program starts with its simplest and wittingly correct, but as a rule extremely inefficient version. Then a series of correctness preserving transformations is applied, which results in a correct and efficient program. A language level which provides simplicity and clarity of initial formulations is the level of recursively defined functions and relations. An advanced technique of manipulation with recursive programs is necessary just for more complete use of these advantages of a high level language.

Note that the transformational power of a transformation system for recursive programs (as well as for other models of programs, of course) essentially depends on the completeness of revealing and using different semantic properties of program fragments. There are two basic kinds of such properties: properties of specific operations used in a program which are usually presented in the form of rewriting rules, and properties which are independent of the used operations interpretation and are defined on a program scheme level.

Recursive program transformations based on properties of the first kind (such as associativity, left commutativity, etc.) were studied in [2] and other related works. It was shown that these properties together with folding–unfolding techniques allow one to implement a wide spectrum of recursive program improvements. But it is important that for each class of programs the reached effect essentially depends on one's ability to construct such a system of rewriting rules which quite completely characterizes properties of operations used in that class (see, e.g., [4]) and also to make necessary generalizations during the course of transformations. In any case, the essential shortcoming of this approach is that no properties are used except those initially expressed by rewriting rules.

Tranformations based on schematic properties (such as uselessness of some computations, see [6]) automatically revealed by global flow analysis of program schemes, form an alternative to the previous approach on one hand and supplement it on the other hand. In particular, the following strategy of transformational program construction seems to be the best: on each language level apply first schematic transformations and then transformations taking into account properties of concrete interpretation.

Systematic study of recursive program transformations based on schematic properties was started in [5] where a quite extensive catalogue of these and a global analysis method for revealing necessary properties

* Present address: Tagor street 30/616, Ramat-Aviv, Tel-Aviv 69203, Israel.

are presented (see also [10]). This paper adjoins [5] and presents some new schematic properties (Section 3) and equivalent transformations of recursive program schemes (Section 4). Some possibilities of these transformations are demonstrated, in particular, for simplification of recursive definitions structure by means of some recursive calls elimination.

2. Definitions and notations

All notions necessary in this paper, are well known in the literature [3,7,8]; so we recall them here only briefly and not too formally.

A *recursive scheme* is a system of equations of the form $F_i(\bar{x}_i) \Leftarrow \tau_i$ $(i = 1,\ldots,n)$, where F_i are pairwise distinct *defined* functions symbols, and *terms* τ_i are built of variables $\bar{x}_i$, interpreted *conditional function* **if–then–else** and symbols of *basic* (noninterpreted) functions and predicates on a *domain* D. D is also noninterpreted; it is known only that it contains an element ω (*undefined value*) and that its elements are *partially ordered* by $\sqsubseteq$ relation: ω is the least element, while elements in $D \setminus \{\omega\}$ are pairwise incomparable. The Ω symbol is reserved for a function defined by $\Omega(\bar{x}) \Leftarrow \Omega(\bar{x})$. The right-hand side of the *ith* equation is the *body* of F_i, and a term of the form $F_i(\sigma_{k_1},\ldots,\sigma_{k_l})$ is a *call* to F_i. A term without calls is a *basic term* and is denoted by t to distinct it from arbitrary terms denoted by π, τ, σ. The following is an example of a recursive scheme:

$$F(x, y) \Leftarrow \textbf{if } p(y) \textbf{ then } x \textbf{ else } F(a(x), \textbf{if } q(x) \textbf{ then } y \textbf{ else } F(x, b(y))).$$

Here and further F, G,... denote defined functions, while a, b,... and p, q,... denote basic functions and predicates, respectively.

An *interpretation* I provides a concrete domain D_I and, for basic functions and predicate symbols, total functions and predicates on D_I (yielding the result ω iff one of the arguments equals ω). Given an I, a scheme converts into a program defining an n-tuple of *monotonic* (with respect to $\sqsubseteq$) functions $((F_1)_I,\ldots,(F_n)_I)$ which is the least solution of the considered system of equations [7]. The *value* $(S)_I(\bar{x})$ *of a scheme* S under given I and given values of variables $\bar{x}$ (determined as elements of D_I) equals $(F_1)_I(\bar{x})$. Different *computation rules* are known [7] which provide operational definition of $(S_I)(\bar{x})$ (and, moreover, of the value $\tau_I(\bar{x})$ of an arbitrary τ) equivalent to the above denotational definition. In what follows, speaking about computation of a term value, one may mean for definiteness one of these rules, e.g., 'call by name'.

Two schemes S_1 and S_2 are (*functionally*) *equivalent* iff

$$\forall I\ \forall \bar{x} \in D_I^n\ \left[(S_1)_I(\bar{x}) = (S_2)_I(\bar{x})\right].$$

Finally, $\tau[\tau_1/x_1,\ldots,\tau_n/x_n]$ denotes the result of simultaneous substitution of the term τ_i for all occurrences of the variable x_i in τ $(i = 1,\ldots,n)$.

3. Schematic properties

Let S be any arbitrary recursive scheme; let t and τ be arbitrary terms in S. The global flow analysis method described in [5] ('marking method') allows one to construct a set PRED(τ) of variables occurring in τ and sets IDEN(t), ITER(t), OM(τ) of terms occurring in S which possess the following properties:

(1) $\forall x\ \{\exists I\ \exists p[$(predicate p is tested during the course of computation of τ_i) & (x affects the test's result)$] \rightarrow x \in \text{PRED}(\tau)\}$.

(2) $\forall\sigma\,\{\sigma\in \textsc{Iden}(t)\to\forall I\,\forall\bar{x}\in D_I^n\,[\sigma_I(\bar{x})\neq\omega\to\sigma_I(\bar{x})=t_I(\bar{x})]\}$,

that is, if a computation of σ under I terminates, then its result coincides with that of a basic term t, or, in other words,

$$\forall\sigma\,[\sigma\in \textsc{Iden}(t)\to\sigma\sqsubseteq t].$$

(3) $\forall\sigma\,\{\sigma\in \textsc{Iter}(t)\to\forall I\,\forall\bar{x}\in D_I^n\,\exists k\geqslant 0\,[\sigma_I(\bar{x})\neq\omega\to\sigma_I(\bar{x})=t_I^k(\bar{x})]\}$,

this time a basic term t being constructed of unary functions only (i.e., it has the form $t=f_1(f_2,\ldots,f_n(x),\ldots)$), while t^k is defined in the usual way:

$$t^0=x \quad\text{and}\quad t^{k+1}=t[t^k/x] \quad\text{for } k\geqslant 0.$$

(4) $\forall\sigma\,\{\sigma\in \textsc{Om}(\tau)\to\forall I\,\forall\bar{x}\in D_I^n\,[\tau_I(\bar{x})=\omega\to\sigma_I(\bar{x})=\omega]\}$,

i.e., if computation of τ under I does not terminate, then computation of σ does not terminate as well.

Note that the use of such properties seems to be useful not only for application of transformations (as it is shown in the next section) but also for proving various assertions about programs—e.g., about program termination.

Now turn to formal definition of the above enumerated sets.

3.1

$\textsc{Pred}(\tau)$ is defined jointly with one more set, $\textsc{Val}(\tau)$, as follows. For any variable x,

(1) if $\tau=x$, then $\textsc{Val}(\tau)=\{x\}$, $\textsc{Pred}(\tau)=\emptyset$;

(2) let $\tau=f(\tau_1,\ldots,\tau_n)$; if $\exists i\,[x\in \textsc{Val}(\tau_i)]$, then $x\in \textsc{Val}(\tau)$, and if $\exists j\,[x\in \textsc{Pred}(\tau_j)]$, then $x\in \textsc{Pred}(\tau)$;

(3) let $\tau=p(\tau_1,\ldots,\tau_n)$; then $(\tau)=\emptyset$, and if $\exists i\,[x\in \textsc{Val}(\tau_i)\cup \textsc{Pred}(\tau_i)]$, then $x\in \textsc{Pred}(\tau)$;

(4) let $\tau=$ **if** π **then** τ_1 **else** τ_2; if $\exists i\,[x\in \textsc{Val}(\tau_i)]$, then $x\in \textsc{Val}(\tau)$, and if $\exists j\,[x\in \textsc{Pred}(\pi)\cup \textsc{Pred}(\tau_j)]$, then $x\in \textsc{Pred}(\tau)$;

(5) let $\tau=F(\tau_1,\ldots,\tau_n)$ where

$$\textsc{Val}(F(x_1,\ldots,x_n))=\{x_{k_1},\ldots,x_{k_s}\} \quad\text{and}\quad \textsc{Pred}(F(x_1,\ldots,x_n))=\{x_{i_1},\ldots,x_{i_r}\};$$

if $\exists j\,[x\in \textsc{Val}(\tau_{k_j})]$, then $x\in \textsc{Val}(\tau)$, and if

$$\exists j\,[x\in \textsc{Val}(\tau_{i_j})\cup \textsc{Pred}(\tau_{i_j})]\vee\exists\ell\,[x\in \textsc{Pred}(\tau_{k_\ell})],$$

then $x\in \textsc{Pred}(\tau)$.

Example. For scheme

$$F(x,y,z)\Leftarrow \textbf{if } p(x) \textbf{ then } a(y,z) \textbf{ else } F(z,b(y),x)$$

we have

$$\textsc{Pred}(F(x,y,z))=\{x,z\} \quad\text{and}\quad \textsc{Val}(F(x,y,z))=\{x,y,z\}.$$

3.2

The precise definition of $\textsc{Iden}(t)$ for arbitrary basic t is too bulky. So we give it only for $t=f_1f_2\ldots f_k(x)$, where all f_i's are unary and a part of parentheses is omitted for the sake of brevity. The complete definition

can be obtained from that below by straightforward generalization. (In (2) and (4) one merely should consider 'initial pieces' of the tree which represents t and their 'continuations').

We first define a set $\overline{\text{IDEN}}(t)$, the complement of IDEN(t):

(1) if τ does not contain x, then $\tau \in \overline{\text{IDEN}}(f_1,\ldots,f_k(x))$;

(2) let $\tau = f(\tau_1,\ldots,\tau_n)$; if

$$\forall i \leqslant k\ \forall \tau' \left[\tau \neq f_1 \ldots f_i(\tau') \vee \tau' \in \overline{\text{IDEN}}(f_{i+1}\ldots f_k(x))\right],$$

then $\tau \in \overline{\text{IDEN}}(f_1 \ldots f_k(x))$;

(3) let $\tau =$ **if** π **then** τ_1 **else** τ_2; if $\exists i\,[\tau_i \in \overline{\text{IDEN}}(f_1 \ldots f_k(x))]$, then $\tau \in \overline{\text{IDEN}}(f_1 \ldots f_k(x))$;

(4) let $\tau = F(\tau_1,\ldots,\tau_n)$ where F is defined by $F(x_1,\ldots,x_n) \Leftarrow \sigma$; if

$$\forall r \leqslant n\ \forall i \leqslant k \left[\sigma \in \overline{\text{IDEN}}(f_1 \ldots f_i(x_r)) \vee \tau_r \in \overline{\text{IDEN}}(f_{i+1} \ldots f_k(x))\right]$$
$$\&\{(\sigma = \textbf{if } \pi \textbf{ then } \sigma_1 \textbf{ else } \sigma_2) \rightarrow$$
$$\rightarrow \exists j\ \forall r \leqslant n\ \forall i \leqslant k \left[\sigma_j \in \overline{\text{IDEN}}(f_1 \ldots f_i(x_r)) \vee \tau_r \in \overline{\text{IDEN}}(f_{i+1} \ldots f_k(x))\right]\},$$

then $\tau \in \overline{\text{IDEN}}(f_1 \ldots f_k(x))$.

All other terms (of the considered scheme) belong to $\text{IDEN}(f_1 \ldots f_k(x))$.

Example. For scheme

$$F(x, y) \Leftarrow \textbf{if } p(x) \textbf{ then } y \textbf{ else } F(a(x), y)$$
$$G(u, v) \Leftarrow \textbf{if } q(v) \textbf{ then } f_1(F(b(v), f_2(u))) \textbf{ else } G(u, c(v))$$

we have $F(x, y) \in \text{IDEN}(y)$ and $G(u, v) \in \text{IDEN}(f_1 f_2(u))$.

3.3

Let $t = f_1,\ldots,f_k(x)$ be a basic term. To define ITER(t) we first describe $\overline{\text{ITER}}(t)$, which is the complement of ITER(t):

(1) if τ does not contain x, then $\tau \in \overline{\text{ITER}}(t)$;

(2) let $\tau = f(\tau_1,\ldots,\tau_n)$; if

$$\forall i \left(\tau \in \overline{\text{IDEN}}(t^i)\right) \,\&\, \forall \tau' \left(\tau \neq t[\tau'/x] \vee \tau' \in \overline{\text{ITER}}(t)\right),$$

then $\tau \in \overline{\text{ITER}}(t)$;

(3) let $\tau =$ **if** π **then** τ_1 **else** τ_2; if $\exists i\,[\tau_i \in \overline{\text{ITER}}(t)]$, then $\tau \in \overline{\text{ITER}}(t)$;

(4) let $\tau = F(\tau_1,\ldots,\tau_n)$ where F is defined by $F(x_1,\ldots,x_n) \Leftarrow \sigma$; if

$$\forall i \left[\sigma \in \overline{\text{ITER}}(t[x_i/x]) \vee \tau_i \in \overline{\text{ITER}}(x)\right]$$
$$\&\ \left\{\sigma = \textbf{if } \pi \textbf{ then } \sigma_1 \textbf{ else } \sigma_2 \rightarrow \exists j\ \forall i \left[\sigma_j \in \overline{\text{ITER}}(t[x_i/x]) \vee \tau_i \in \overline{\text{ITER}}(x)\right]\right\},$$

then $\tau \in \overline{\text{ITER}}(t)$.

All other terms (of the considered scheme) belong to ITER(t).

Note that such a definition is not constructive because of unboundness of the quantifier on i in (2). Nevertheless, this difficulty can be overcome, since it can be shown that if a scheme's size (defined as a number of symbols in the text of the scheme) equals N, then the mentioned quantifier can be replaced by $\forall i \leqslant N$.

Also note that this definition does not generalize to the case of arbitrary basic t (as it took place for IDEN(t)) so as to keep the property of ITER(t) formulated at the beginning of this section.

Example. For scheme

$$F(x, y) \Leftarrow \textbf{if } p(x) \textbf{ then } y \textbf{ else } F(a(x), y)$$
$$G(u, v) \Leftarrow \textbf{if } q(u) \textbf{ then } f_1(F(b(u), f_2(v))) \textbf{ else } G(c(u), f_1(f_2(v)))$$

we have $G(u, v) \in \text{ITER}(f_1 f_2(v))$.

3.4

To define $\text{OM}(\tau)$ we need the notion of a strict tuple of a term's arguments introduced in [9]. Informally, if a term σ contains variables $x_1, \ldots, x_n$ and $\langle x_{i_1}, \ldots, x_{i_k} \rangle$ is its strict tuple, then the value of $\sigma[\Omega/x_{i_1}, \ldots, \Omega/x_{i_k}]$ equals ω under any I and any values of its variables. $\text{OM}(\tau)$ is defined as follows:

(1) $\Omega \in \text{OM}(\tau)$;

(2) $\tau \in \text{OM}(\tau)$;

(3) if $\sigma \in \text{OM}(\tau)$, then replacement of any occurrence of any subterm in σ by a term from $\text{OM}(\tau)$ yields a term from $\text{OM}(\tau)$;

(4) if $\sigma = \varphi(\sigma_1, \ldots, \sigma_n)$ where φ is any function (**if–then–else**, basic or defined) and a strict tuple $\langle x_{i_1}, \ldots, x_{i_s} \rangle$ of $\varphi(\bar{x})$ exists such that $\forall j\, [\sigma_{i_j} \in \text{OM}(\tau)]$, then $\sigma \in \text{OM}(\tau)$;

(5) if $\tau = \pi[\sigma_1/x_1, \ldots, \sigma_n/x_n]$ and $\sigma = F(\sigma_1, \ldots, \sigma_n)$ where F is defined by $F(x_1, \ldots, x_n) \Leftarrow \pi$, then $\sigma \in \text{OM}(\tau)$;

(6) if τ is a basic term, then $\forall \sigma\, [\sigma \in \text{OM}(\tau)]$;

(7) let $\tau = f(\tau_1, \ldots, \tau_n)$ where f is either **if–then–else** or a basic function; if

$$\forall i \left[\sigma \in \text{OM}(\tau_i)\right] \vee \left\{\sigma = f(\sigma_1, \ldots, \sigma_n) \;\&\; \forall j \left[\sigma_j \in \text{OM}(\tau_j)\right]\right\},$$

then $\sigma \in \text{OM}(\tau)$;

(8) let $\tau = F(\tau_1, \ldots, \tau_n)$ where

$$\text{PRED}(F(x_1, \ldots, x_n)) = \{x_{i_1}, \ldots, x_{i_s}\} \quad \text{and} \quad \text{VAL}(F(x_1, \ldots, x_n)) = \{x_{j_1}, \ldots, x_{j_r}\};$$

if

$$\sigma = F(\sigma_1, \ldots, \sigma_n) \;\&\; \forall k \left[\sigma_{i_k} = \tau_{i_k} \vee \left(\tau_{i_k} \text{ is basic } \&\; \sigma_{i_k} \in \text{IDEN}(\tau_{i_k})\right)\right] \;\&\; \forall m \left[\sigma_{j_m} \in \text{OM}(\tau_{j_m})\right],$$

then $\sigma \in \text{OM}(\tau)$.

Example. For scheme

$$F(x, y) \Leftarrow \textbf{if } p(x) \textbf{ then } a(x, y) \textbf{ else } F(b(x), c(y))$$
$$G(u, v) \Leftarrow \textbf{if } q(u) \textbf{ then } F(b(u), v) \textbf{ else } F(b(u), F(c(u), v))$$

and terms $\tau = F(b(u), v)$ and $\sigma = F(b(u), F(c(u), v))$ we have $\sigma \in \text{OM}(\tau)$ (though $\sigma \not\sqsubseteq \tau$) and $\tau \notin \text{OM}(\sigma)$.

Finally note that all sets defined above present some trustworthy but, generally speaking, noncomplete information. For example, each term in $\text{OM}(\tau)$ has the property mentioned at the beginning of the section, but there can exist terms with such property which do not belong to $\text{OM}(\tau)$. Such incompleteness may be caused by nonfreeness of the analyzed scheme (a scheme S is nonfree if there exists an I such that some predicate is tested with the same term more than once during the computation of S_I), as can be seen from the following example:

$$F(x, y) \Leftarrow \textbf{if } p(x) \textbf{ then } a(x) \textbf{ else } (\textbf{if } p(x) \textbf{ then } b(y) \textbf{ else } F(x, c(y))).$$

Here $F(x, y) \notin \text{IDEN}(a(x))$, though $F(x, y) \sqsubseteq a(x)$.

4. Transformations of recursive program schemes

In this section we consider a series of recursive schemes transformations. Being based on results of global analysis (revealing properties described above), they are applied locally to occurrences of terms in a scheme. If S, S′ are two schemes, then a transformation is written down as $S \leftrightarrow S'$ which, in particular, indicates its reversibility.

T1 [Replacement of a term by its 'analytical representation']. Let S be a scheme, τ an occurrence of a term in S, and σ an occurrence of a subterm in τ. Let, further, $\exists t\,[\sigma \in \text{IDEN}(t)]$ and τ' is obtained from τ by replacement of σ by t. If $\tau' \in \text{OM}(\sigma)$, then $S \leftrightarrow S'$ where S′ is obtained from S by replacement of τ by τ'.

Example

$$
\begin{aligned}
&F(x, y) \Leftarrow \textbf{if } p(x, y) \textbf{ then } x \textbf{ else } F(G(x, y), G(x, y))\\
&G(u, v) \Leftarrow \textbf{if } q(v) \textbf{ then } h(u) \textbf{ else } G(u, f(v))\\
&\quad\updownarrow\\
&F(x, y) \Leftarrow \textbf{if } p(x, y) \textbf{ then } x \textbf{ else } F(h(x), G(x, y))\\
&G(u, v) \Leftarrow \textbf{if } q(v) \textbf{ then } h(u) \textbf{ else } G(u, f(v)).
\end{aligned}
$$

Here $\sigma = G(x, y) \in \text{IDEN}(h(x))$ and $\tau' = F(h(x), G(x, y)) \in \text{OM}(\sigma)$ (according to point (4) of the definition of $\text{OM}(\sigma)$, since $\langle y \rangle$ is a strict tuple of $F(x, y)$). Similarly, the second occurrence of σ could be replaced by h(x), but not both since $F(h(x), h(x)) \notin \text{OM}(\sigma)$.

Thus, due to T1, in certain situations computations of recursively defined functions can be replaced by computations over explicit 'formulas'. In other words, T1 realizes a sort of partial computation in a recursive scheme. In this sense there is some analogy with reduction of a term in an interpreted scheme, when a term with values of all its variables being known is replaced by explicit computed result.

The following transformations, T2–T4, carry out different commutations of basic and defined functions.

T2. Let S be a scheme, $\tau = F(\tau_1, \ldots, \tau_{i-1}, g(\sigma_1, \ldots, \sigma_m), \tau_{i+1}, \ldots, \tau_n)$ is an occurrence of a term in S, where $F(x_1, \ldots, x_n) \in \text{IDEN}(x_i)$ and $x_i \notin \text{PRED}(F(x_1, \ldots, x_n))$. Then $S \leftrightarrow S'$, where S′ is obtained from S by replacement of τ by

$$\tau' = g(F(\tau_1, \ldots, \tau_{i-1}, \sigma_1, \tau_{i+1}, \ldots, \tau_n), \ldots, F(\tau_1, \ldots, \tau_{i-1}, \sigma_m, \tau_{i+1}, \ldots, \tau_n)).$$

Example

$$
\begin{aligned}
&F(x, y) \Leftarrow \textbf{if } p(x) \textbf{ then } y \textbf{ else } F(a(x), y)\\
&G(u, v) \Leftarrow \textbf{if } q(u, v) \textbf{ then } b(v) \textbf{ else } F(c(v), g(v, G(v, u)))\\
&\quad\updownarrow\\
&F(x, y) \Leftarrow \textbf{if } p(x) \textbf{ then } y \textbf{ else } F(a(x), y)\\
&G(u, v) \Leftarrow \textbf{if } q(u, v) \textbf{ then } b(v) \textbf{ else } g(F(c(v), v), F(c(v), G(u, v))).
\end{aligned}
$$

Here $F(x, y) \in \text{IDEN}(y)$ and $y \notin \text{PRED}(F(x, y))$. Note that being suitably directed, T2 allows one to reduce a number of recursive calls in a scheme.

T3. Let S be a scheme, $\tau = F(\tau_1,\ldots,g(\sigma_1,\ldots,\sigma_m),\ldots,g(\sigma_1,\ldots,\sigma_m),\ldots,\tau_n)$ is an occurrence of a term in S, where occurrences of $g(\sigma_1,\ldots,\sigma_m)$ occupy argument positions $i_1,\ldots,i_m$ of the function F. If

$$\exists k\left[F(x_1,\ldots,x_n)\in \textsc{Iden}\left(g^k\left(x_{i_1},\ldots,x_{i_m}\right)\right)\right] \quad \text{and} \quad \forall j\left[x_{i_j}\notin \textsc{Pred}(F(x_1,\ldots,x_n))\right],$$

then $S \leftrightarrow S'$ where S′ is obtained from S by replacement of τ by

$$\tau' = g\left(F(\tau_1,\ldots,\sigma_1,\ldots,\sigma_m,\ldots,\tau_n),\ldots,F(\tau_1,\ldots,\sigma_1,\ldots,\sigma_m,\ldots,\tau_n)\right).$$

Example

$$F(x, y, z) \Leftarrow \textbf{if } p(z) \textbf{ then } g(g(x, y), g(x, y)) \textbf{ else } F(x, y, a(z))$$
$$G(u, v) \Leftarrow \textbf{if } q(u) \textbf{ then } b(v) \textbf{ else } F(g(u, v), g(u, v), c(u))$$
$$\updownarrow$$
$$F(x, y, z) \Leftarrow \textbf{if } p(z) \textbf{ then } g(g(x, y), g(x, y)) \textbf{ else } F(x, y, a(z))$$
$$G(u, v) \Leftarrow \textbf{if } q(u) \textbf{ then } b(v) \textbf{ else } g(F(u, v, c(u)), F(u, v, c(u))).$$

Here $F(x, y, z)\in \textsc{Iden}(g^2(x, y))$ and $x, y \notin \textsc{Pred}(F(x, y, z))$. Note that being applied in a suitable direction, T3 allows one to replace one group of common subexpressions by another, in which subexpressions have more simple structure.

T4. Let S be a scheme, $\tau = F(\tau_1,\ldots,\tau_{i-1}, f(\tau_i), \tau_{i+1},\ldots,\tau_n)$ is an occurrence of a term in S where f is unary. If $F(x_1,\ldots,x_n)\in \textsc{Iter}(f(x_i))$ and $x_i \notin \textsc{Pred}(F(x_1,\ldots,x_n))$, then $S \leftrightarrow S'$ where S′ is obtained from S by replacement of τ by $\tau' = f(F(\tau_1,\ldots,\tau_i,\ldots,\tau_n))$.

Example

$$F(x, y) \Leftarrow \textbf{if } p(y) \textbf{ then } x \textbf{ else } F(f(x), g(y))$$
$$\updownarrow$$
$$F(x, y) \Leftarrow \textbf{if } p(y) \textbf{ then } x \textbf{ else } f(F(x, g(y))).$$

Here $F(x, y)\in \textsc{Iter}(f(x))$ and $x \notin \textsc{Pred}(F(x, y))$.

In [7] equivalence of these two schemes was proved by computational induction. Note also that transition from one scheme to another could be carried out by the Burstall–Darlington technique [2]. However, in contrast to our transformation T4, in each of these latter cases some 'eureka' is required to form induction hypothesis or to make appropriate generalization. This might become a rather difficult task for less trivial schemes.

Concluding this section we formulate two more transformations.

T5 [Removal of recursion over 'almost' invariant arguments]. Let a scheme S contain definition of a function F and let $\tau = F(\tau_1,\ldots,\tau_n)$ be an occurrence of a term in F's body. If

$$\forall i\left[x_i \in \textsc{Pred}\left(F(x_1,\ldots,x_n)\right) \rightarrow \tau_i \in \textsc{Iden}(x_i)\right],$$

then $S \leftrightarrow S'$, where S′ is obtained from S by replacement of τ by Ω.

Example

$$
\begin{aligned}
&F(x, y) \Leftarrow \textbf{if } p(x, y) \textbf{ then } a(y) \textbf{ else } F(x, G(y, b(x)))\\
&G(u, v) \Leftarrow \textbf{if } q(v) \textbf{ then } u \textbf{ else } G(u, c(v))\\
&\quad\updownarrow\\
&F(x, y) \Leftarrow \textbf{if } p(x, y) \textbf{ then } a(y) \textbf{ else } \Omega\\
&G(u, v) \Leftarrow \textbf{if } q(v) \textbf{ then } u \textbf{ else } G(u, c(v)).
\end{aligned}
$$

Here $x, y \in \text{PRED}(F(x, y))$, $x \in \text{IDEN}(x)$ and $G(y, b(x)) \in \text{IDEN}(y)$.

T6 [Removal of useless test]. Let S be a scheme and $\tau = \textbf{if } \pi \textbf{ then } \sigma \textbf{ else } \sigma$ is an occurrence of a term in S. If $\sigma \in \text{OM}(\pi)$, then $S \leftrightarrow S'$ where S′ is obtained from S by replacement of τ by σ.

Example

$$
\begin{aligned}
&F(x, y) \Leftarrow \textbf{if } p(x) \textbf{ then } a(x, y) \textbf{ else } F(b(x), y)\\
&G(u, v) \Leftarrow \textbf{if } q(F(u, v)) \textbf{ then } F(c(v), F(u, v)) \textbf{ else } F(c(v), F(u, v))\\
&\quad\updownarrow\\
&F(x, y) \Leftarrow \textbf{if } p(x) \textbf{ then } a(x, y) \textbf{ else } F(b(x), y)\\
&G(u, v) \Leftarrow F(c(v), F(u, v)).
\end{aligned}
$$

Here $F(c(v), F(u, v)) \in \text{OM}(F(u, v))$.

Correctness of transformations T1–T6 arises from the following theorem.

Theorem. *Transformations* T1–T6 *keep the functional equivalence relation.*

Proof. We do not consider the full proof of the theorem here but rather give a sketch of it. We assume the fact to be proved that all the sets defined syntactically in Section 3 possess the semantic properties formulated at the beginning of that section.

For T1, let us show that $(\tau)_I = (\tau')_I$ for an arbitrary interpretation I. It follows from $\sigma \in \text{IDEN}(t)$ that $\sigma \sqsubseteq t$ and hence $\tau \sqsubseteq \tau'$, because τ' is obtained from τ by substitution of t for σ and all functions in a recursive scheme (both basic and defined ones) are monotonic. Now, if $(\sigma)_I \neq \omega$, then $(\sigma)_I = (t)_I$ and $(\tau)_I = (\tau')_I$, and if $(\sigma)_I = \omega$, then $(\tau')_I = \omega$ (since $\tau' \in \text{OM})\sigma)$ and $\tau \sqsubseteq \tau'$ implies $(\tau)_I = \omega$, i.e., again $(\tau)_I = (\tau')_I$.

For T2, let $\pi_k = F(\tau_1, \ldots, \tau_{i-1}, \sigma_k, \tau_{i+1}, \ldots, \tau_n)$ for $k = 1, \ldots, n$. Since $x_i \notin \text{PRED}(F(\bar{x}))$ and terms τ and π_k differ only in the *ith* argument of F, the 'protocols' of computations of τ and π_k must coincide in the sense that in both cases we have the same sequence of predicates tested on the same arguments. So either under given I both sequences are infinite and $(\tau)_I = (\pi_k)_I = \omega$, and hence $(\tau)_I = (\tau')_I = \omega$, or both are finite and lead to some results. It follows from $F(\bar{x}) \in \text{IDEN}(x_i)$ that these results must be $(g(\sigma_1, \ldots, \sigma_m))_I$ for τ and $(\sigma_k)_I$ for π_k, and hence under such I, $(\tau)_I = (\tau')_I$ as required.

Transformations T3 and T4 are proved in a similar way.

For T5, if the call $\tau = F(\tau_1, \ldots, \tau_n)$ is never macro-expanded under I, then τ is obviously equivalent to Ω for such I. Otherwise we have the fact that to compute $(F(\bar{x}))_I$ for some values of its arguments we need to compute $(\tau)_I$ for the same values of arguments (since τ stands in the body of F). Consider now the sequence of predicates tested starting from the moment of macro-expansion of $F(\bar{x})$ up to the moment of

the first macro-expansion of τ, and recall that $(\tau_i)_I$ equals either x_i or ω for all $x_i \in \text{PRED}(F(\bar{x}))$. If for all such x_i's we have $(\tau_i)_I = x_i$, then clearly this sequence will be repeated infinitely, thus yielding $(\tau)_I = \omega$. In the other case, when some of the mentioned $(\tau_i)_I$'s equal ω, $(\tau)_I$ must be yet less defined (since F is monotonic, as well as all other functions in a scheme), i.e., again $(\tau)_I = \omega$. Hence, τ is always equivalent to Ω.

At last, for T6, if $(\pi)_I$ equals true or false, then clearly $(\tau)_I = (\sigma)_I$. Otherwise, if $(\pi)_I = \omega$, then $(\tau)_I = \omega$ by definition of the **if–then–else** function and $(\sigma)_I = \omega$ since $\sigma \in \text{OM}(\pi)$. □

5. Conclusions

Equivalent transformations presented in this paper allow one essentially to increase the effectiveness of recursive programs, especially if accompanied by other known transformations [2,5,6]. Their application in practice requires, of course, to first carefully analyze the complexity of revealing necessary schematic properties. We do not regard this question here, but intend to do it in future publications. Here we only wish to note that such a complexity essentially depends on the degree of completeness of the information used for program transformation. Remember, for example, the notion of a strict tuple used in the definition of $\text{OM}(\tau)$ in Section 3. One-element strict tuples (or, in other words, strict variables of terms) can be found in time polynomial on scheme's size, while for finding all strict tuples, exponential time is probably needed.

References

[1] F.L. Bauer, P. Pepper and H. Wössner, Notes on the project CIP: Outline of a transformation system, TUM-INFO-7729, Technische Universität München, 1977.

[2] R.M. Burstall and J. Darlington, A transformation system for developing recursive programs, J. ACM 24 (1) (1977) 44–67.

[3] R.L. Constable and D. Gries, On classes of program schemata, SIAM J. Comput. 1 (1) (1972) 66–118.

[4] J. Darlington, A synthesis of several sorting algorithms, Acta Informatica 11 (1) (1978) 1–30.

[5] A.P. Ershov and V.K. Sabelfeld, An essay of schematic theory of recursive programs, in: Translation and Program Models (Novosibirsk, 1980) pp. 23–43 (in Russian).

[6] I. Guessarian, Program transformations and algebraic semantics, Theoret. Comput. Sci. 9 (1) (1979) 39–66.

[7] Z. Manna, S. Ness and J. Vuillemin, Inductive methods for proving properties of programs, Comm. ACM 16 (8) (1973) 491–502.

[8] B.K. Rosen, Program equivalence and context-free grammars, J. Comput. System Sci. 11 (3) (1975) 358–374.

[9] V.K. Sabelfeld, On one algorithm of recursive programs optimization, in: Proc. All-Union Conf. on Translation Methods (Novosibirsk, 1981) pp. 141–143 (in Russian).

[10] V.K. Sabelfeld, Tree equivalence of linear recursive schemata is polynomial-time decidable, Inform. Process. Lett. 13 (4,5) (1981) 147–153.