Mixed computation in the class of recursive program schemata

By A. P. ERSHOV

To the memory of Professor László Kalmár

Let some class $\mathfrak{A}$ of algorithms be prescribed by a set $\mathscr{P}$ of programs P, a domain $\mathscr{X}$ of input data X, a domain $\mathscr{Y}$ of results Y and a computation $\mathbf{V}$ being a universal process which is defined for any P and X and is either infinite or resultless (yielding an *abort*) or yields some Y as a function of P and X: $Y=\mathbf{V}(P, X)$. *Mixed computation* [1] in $\mathfrak{A}$ is a universal process $\mathbf{M}$ which is defined for any P, X and a parameter μ (specifically characterizing the process). The process is either infinite or resultless or it generates some *residual program* $\mathbf{M}_G(P, M, \mu)$ and yields *partial results* $\mathbf{M}_C(P, X, \mu)$. A mixed computation is correct if for any P, X and μ the following functional identity holds

$$\mathbf{V}(P, X) = \mathbf{V}(\mathbf{M}_G(P, X, \mu),\ \mathbf{M}_C(P, X, \mu)).$$

It has been shown [2] that mixed computation and such related concepts as partial evaluation [3], computation over incomplete information [4], "progonka" [5] may be a basis for solution of many programming problems where efficiency has to be traded off with universality.

It is natural to seek a correct formalism of mixed computation for the most common abstract models of program. The correctness of mixed computation for ALGOL-like programs has been shown in [6]. In this note a correct procedure of mixed computation in the class of recursive program schemata is presented. This class reflects such properties of algorithmic languages as recursion and proceduring.

We shall introduce some notations. If $\mathscr{M}$ is a set of elements m then M^n is an n-tuple of elements from $\mathscr{M}$. The length of a tuple used as an argument of a functional symbol f is always equal to its arity $\varrho(f)$. $\tau[B]$ is a term τ constructed over a set B of basic symbols, $\tau(A)$ is a term τ for which its arguments (variables or constants) A are shown.

According to [7] a *recursive program schema* is specified as a system of equalities *(function declarations)*

$$f_i(X_i^{\varrho(f_i)}) = \tau_i[X_i, C, \{f_1, \ldots, f_k\}, \Pi, \Phi] \quad (i = 1, \ldots, k),$$

where f_i are *defined functions*. $\mathscr{X}$ and $\mathscr{C}$ are countable sets of variables x and constants c, Π and Φ are finite sets of predicate and functional symbols, respectively, of fixed arities.

Predicate terms π are used to define *conditional* terms $\{\pi|t_1|t_2\}$ where t_1 and t_2 are functional or conditional terms. Terms τ_i are arbitrary terms *(function bodies)* over specified sets of symbols.

Let an interpretation of the basic symbols (constants, functions and predicates) converting a schema into a recursive program be given. A system of functions $\varphi_1, \ldots, \varphi_k$ is called a *fixed point* of a recursive program if, having been combined with the system of basic functions Π and Φ, it makes (after substituting φ_i for each f_i) the function declarations identities.

We say that a function φ_1 *covers* a function φ_2 if the graph of φ_1 contains that of φ_2. Under natural assumptions on basic functions and their regions of definiteness each recursive program has a single so called *lowest fixed point* (LFP) covered by any other fixed point of the program [7].

Let T and C be tuples of terms and constants respectively. A *call* is a term in the form $f(T)$; a *bound call* is a term in the form $f(C)$; a *semi-bound* call is a term in the form $f(C^n, T^m)$ where $n+m=\varrho(f)$; a *transitively* bound call is a call having no variables.

Let one function declaration $f(X)=\tau$ in a program be treated as a *leading declaration* and C be a tuple of $\varrho(f)$ constants. A (sequential) *computation* V over a program P is a step-wise process of constructing a sequence of terms $\tau^0=f(C)$, $\tau^1, \tau^2, \ldots$ which either is developed infinitely or ends by an (resultless) *abort* or (sucessfully) by a constant which is taken as the value $\varphi(C)$ of the function $\varphi(X)$ computed over the given program for its leading declaration.

Each step of the construction of τ^{i+1} from τ^i consists of two parts.

1. *Rewriting.* In τ^i somehow a call $f_j(T)$ is chosen. This call is replaced by a term t. The latter is obtained from the function body τ_j of the declaration $f_j(X_j)=\tau_j$ by replacement of variables from X_j by corresponding components of the tuple T. Let τ' be the rewritten term.

2. *Simplification.* Inductively, all such subterms in τ' are evaluated which contain only constants and basic functions and predicates. The evaluated functional terms are replaced by their value, conditional terms are replaced by their *if-* or *else*-part depending on the value of the predicate. If the simplification yields either an abort or a constant c then the process in terminated yielding either the abort or c as a successful result. Otherwise, the simplified term is taken as τ^{i+1}.

Similarly, a *partial* computation is defined which allows τ^0 to be an arbitrary term with variables. Partial computation is terminated when the simplified term contains no available transitively bound calls.

A variety of computations is determined by the method of selection of subterms subjected to rewriting. In the general case a computation provides with a function covered by the LFP of a given recursive program. A computation which guarantees LFP is called *safe*. An example of safe computation is the execution of the "left outermost" call that corresponds to the "call by name". An unsafe computation is the execution of the "left innermost" call (call by value).

Let the first function declaration $f_1(X_1)=\tau_1$ of a recursive program P be leading and let a partition μ of variables X_1 ($X_1=X'\cup X$) and a semi-bound call

$f_1(B, X)$ be given. Let a computation **V** provide the leading declaration with a function $\varphi(X', X)$. A correct mixed computation **M** of the program P for the given partition μ and tuple of constans B is an arbitrary process of transformation of the program P into a program P_B with a leading declaration $f_0(X)=\tau_0$ such that the function $\varphi_B(X)$ provided by **V** for the program P_B satisfies the identity $\varphi(B, X)=$ $=\varphi_B(X)$.

We shall describe a transformation of P which we call an *execution* of the *semi-bound* call $f_1(B, X)$. Let us take a copy of the term τ_1 and replace in it all occurences of variables from X' by the corresponding constants from B with all subsequent simplifications; we will obtain a term τ_0 as a result. Then we take a new functional symbol f_0 of a defined function $f_0(X)$ and replace, in all terms $\tau_0, \tau_1, \ldots, \tau_k$, all semi-bound calls in the form $f_1(B, T)$ by the calls $f_0(T)$, thus obtaining the terms $\tau_0^*, \tau_1^*, \ldots, \tau_k^*$. Let us denote by P^* the program which is obtained from P by attaching to it the equality $f_0(X)=\tau_0^*$ as leading declaration and by replacing the bodies $\tau_1, \ldots, \tau_k$ by the terms $\tau_1^*, \ldots, \tau_k^*$.

Lemma 1. Let $\varphi_1(X', X), \varphi_2, \ldots, \varphi_k$ and $\psi_0, \psi_1, \ldots, \psi_k$ be LFP of the programs P and P^*, respectively. Then $\psi_i=\varphi_i$ $(i=1, \ldots, k)$ and $\psi_0(X)=\varphi_1(B, X)$.

The proof is based on Kleene's theorem on recursion [8]: it can be shown that subsequent approximations of P and P^* to their LFPs satisfy the lemma at each step.

Let us introduce a *reachability* relation over the defined functions $f_1, \ldots, f_k$ of a recursive program: f_j is reachable from f_i if the body of f_i contains calls for f_j. We will also consider the transitive closure of the reachability.

We shall formulate two obvious lemmas.

Lemma 2. Deleting from a program P the declaration of a function which is transitively unreachable from the function of the leading declaration preserves the 1st component of the LFP of P.

Lemma 3. Replacing in P a call $f(T)$ for the function with a declaration $f(X)=\tau(X)$ by the term $\tau(T)$ preserves the 1st component of the LFP of P.

Now we can describe a correct mixed computation with respect to some computation **V**.

Initial step. A semi-bound call $f_1(B, X)$ is given. It is declared to be the *start* of the first cyclic step.

Cyclic step (transformation of P into P'). Let a start $f(B, X)$ be given. The corresponding declaration in P is considered as the leading one. P is transformed into P^* with the leading declaration $f_0(X)=\tau_0$ according to the rules of execution of a semi-bound call. A partial computation **V** with τ_0 as the initial term and τ_0^* as the result (if any) is undertaken. P^* is then transformed into P' by replacing τ_0 by the term τ_0^*, in the declaration $f_0(X)=\tau_0$.

After each cyclic step we look at τ_0^* whether it contains a semi-bound call $f(C, T)$. If so then the term $f(C, Y)$, where Y are variables from the declaration of f which correspond the terms T, is taken as a start for the next cyclic step. Otherwise the mixed computation is terminated yielding the program after the last step with the leading declaration from the first cyclic step as the residual program. Afterwards, the residual program may be simplified according to lemmas 2 and 3.

Example A. (Power function x^n)

$$\mathrm{pow}(x, n) = \{n = 0 | 1 | \{n \text{ is even } | \mathrm{pow}^2(x, n/2) | x \times \mathrm{pow}(x, n-1)\}\}.$$

Let $\mathrm{pow}(x, N)=N(x)$. The residual program for $\mathrm{pow}(x, 5)$ before simplification

$5(x) = x\times 4(x);$

$4(x) = (2(x))^2;$

$2(x) = (1(x))^2;$

$1(x) = x\times 0(x);$

$0(x) = 1;$

$\mathrm{pow}(x, n) = \{n = 0|1|\{n \text{ is even}|\mathrm{pow}^2(x, n/2)|x\times \mathrm{pow}(x, n-1)\}\}.$

The residual program after simplification:

$5(x) = x\times((x\times 1)^2)^2.$

Let $\mathrm{pow}(5, n)=\exp(n)$. The residual program pow $(5, n)$ after simplification:

$\exp(n) = \{n = 0|1|\{n \text{ is even}|\exp^2(n/2)|5\times\exp(n-1)\}\}.$

Example B. (Akkerman function)

$A(x, y) = \{x = 0|y+1|\{y = 0|A(x-1, 1)|A(x-1, A(x, y-1))\}\}.$

Let $A(3, y) = \exp(y)$; $A(2, y) = \mathrm{mult}(y)$; $A(1, y) = \mathrm{add}(y)$, $A(m, n) = amn$.

The residual program for $A(3, y)$ after simplification:

$\exp(y) = \{y = 0|a21|\mathrm{mult}(\exp(y-1))\};$

$\mathrm{mult}(y) = \{y = 0|a11|\mathrm{add}(\mathrm{mult}(y-1))\};$

$\mathrm{add}(y) = \{y = 0|a01|\mathrm{add}(y-1)+1\}.$

Let $A(x, N)=aN(x)$. The residual program for $A(x, 3)$ before simplification:

$a3(x) = \{x = 0|4|A(x-1, a2(x))\};$

$a2(x) = \{x = 0|3|A(x-1, a1(x))\};$

$a1(x) = \{x = 0|2|A(x-1, a0(x))\};$

$a0(x) = \{x = 0|1|a1(x-1)\};$

$A(x, y) = \{x = 0|y+1|\{y = 0|a1(x-1)|A(x-1, A(x, y-1))\}\}.$

Notice, that elimination of non-recursive declarations can be made in different ways due to the mutual recursion of $a0$ and $a1$. Eliminating $a0$ and $a2$ we obtain (exploiting the logical dependencies):

$a3(x) = \{x = 0|4|A(x-1, A(x-1, a1(x)))\};$

$a1(x) = \{x = 0|2|A(x-1, a1(x-1))\};$

$A(x, y) = \{x = 0|y+1|\{y = 0|a1(x-1)|A(x-1, A(x, y-1))\}\}.$

COMPUTING CENTRE
SIBERIAN BRANCH OF THE USSR AC. SCI.
NOVOSIBIRSK 630090, USSR

References

[1] Ershov, A. P., Об одном теоретическом принципе системного программирования *Dokl. Akad. Nauk SSSR*, v. 223, No. 2, 1977, pp. 272—275.
[2] Ершов, А. П., О сущности трансляции, *Программирование*, No. 5, 1977, pp. 21—39.
3] Beckman, L., A. Haraldson, Ö. Oskarsson, E. Sandewall., A partial evaluator, and its use as a programming tool, *Artificial Intelligence*, v. 7, No. 4, 1976, pp. 319—357.
[4] Бабич, Г. Х., Л. Ф. Штернберг, Т. И. Юганова, Алгоритмический язык инкол для выполнения вычислений с неполной информацией, *Программирование*, No. 4, 1976, pp. 24—32.
[5] Турчин, В. Ф., Эквивалентные преобразования программ на Рефале, Автоматизированная система управления строительством, *Труды ЦНИПИАСС*, Moscow, No. 6, 1974, p. 36.
6] Ershov, A. P. and V. E. Itkin, Correctness of mixed computation in Algol-like programs, *Lecture Notes in Computer Science*, v. 53, 1977, pp. 59—77.
7] Manna, Z., S. Ness, J. Vuillemin, Inductive methods for proving properties of programs, *Comm. ACM*, v. 16, No. 8, 1973, pp. 491—502.
[8] Kleene, S. C., *Introduction to metamathematics*, Amsterdam—Groningen, 1952.

(Received August 1, 1978)