Transformation of annotated programs is a method of program processing which takes into account program application information a priori known and conveyed in annotations. A scheme and languages of annotated programming are described within whose framework many kinds of practical work with programs (e.g., execution, partial evaluation, optimization) can be performed. The problem of global dataflow analysis of annotated programs that covers conventional forward and backward dataflow problems is formulated and solved. A transformation machine concept as an integrated environment for transformations of annotated programs is presented.

Introduction

Transformation techniques are gaining in importance for both theoretical and technological programming. Systems of equivalent transformations have been conventionally used in the optimizing compilers and are currently widely applied to the so-called transformation programming systems where the program development from a specification is a formal, mechanically supported process [1, 2].

Investigations of transformation systems and their applications to various kinds of program manipulations show that when performing transformations it is important to take into account information known about program applications, as well as to employ generalizing and specializing transformations which are nonequivalent. A well-known example of specializing transformation is the so-called partial evaluation (or mixed computation) of programs on partially given inputs [3]. Partial evaluation can be applied to compiling, program generation, including compiler generation and generation of a compiler generator, and metaprogramming without order-of-magnitude loss of efficiency [4].

The main idea of the paper is to consider program processing aimed at improving the program given by a qualitative criterion (memory, time, reliability, etc.) not interfering with the program meaning within the scope of a subfield restricted by formalized comments (annotations) of the program transformed. The class of correct transformations of annotated programs covers various kinds of work with basic programs. It contains both all equivalent transformations and a number of such nonequivalent transformations which specialize or generalize a program to be transformed, in particular partial evaluation. The approach also permits specializing and generalizing transformations of basic programs.
and to employ for their investigation equivalent transformation techniques developed in terms of program schemata theory [6]. Within the approach, transformations can change not only basic programs but their annotations as well. It allows systems of annotated program transformations to be used for solving problems of dataflow analysis and verification. Another advantage of the approach outlined here is the possibility to perform global transformations of basic programs by iterative application of elementary transformations of annotated programs, to construct a system which consists of a relatively small number of elementary transformations of annotated programs and covers a sufficiently broad class of program manipulations including program execution, partial evaluation and optimization.

1. Model for annotated programming

A program model described below is based on large-scale program schemata that covers a broad class of programs and their transformations [5, 6].

Let $S = \{s\}$ be a set of memory states such that for any state $s \in S$ a partition of the set of all variables $V = \{v\}$ into two sets $A(s)$ and $I(s)$ of accessible and inaccessible variables, respectively, is given and for every accessible $v \in A(s)$ its value $s(v)$ is defined. Let $s^1$ and $s^2$ be two memory states. $s^1$ and $s^2$ are equal on a set of the variables $W \subseteq V$ if for any $v \in W$ either $s^1(v) = s^2(v)$ or $v \in I(s^1) \cap I(s^2)$. $s^1$ expands $s^2$ (denoted by $s^2 \preceq s^1$) if $s^1$ and $s^2$ are equal on the set $A(s^2)$.

A program $P$ is a tuple $(g, f, p, r, a, d)$ which consists of

1. a flowgraph $g = (X, U, X_0, Y_0)$, where $x_0 \in X$ is the entry statement having no incoming arcs (i.e. $IN(x_0) = \emptyset$) and only one outgoing arc denoted by $u_0$ (i.e. $OUT(x_0) = \{u_0\}$) and $y_0 \in X$ is the exit statement having no outgoing arcs (i.e. $OUT(y_0) = \emptyset$), and for every arc $u = (x_1, x_2) \in U$ the functions $source(u) = x_1$ and $target(u) = x_2$ are defined;
2. a function of memory transformation $f : X \Rightarrow (S \Rightarrow S)$;
3. a function of control transfer $p : X \Rightarrow (S \Rightarrow U)$;
4. argument and result functions $a, r : X \Rightarrow (S \Rightarrow V)$;
5. applicability predicate $d : X \Rightarrow (S \Rightarrow \{\text{true}, \text{false}\})$,

such that for any $v \in V$, $x \in X$ and $s, s^1, s^2 \in S$ the following properties hold:

1. the memory states $s$ and $f(x)(s)$ are equal on the set $\forall v \in V (x(s) = f(x)(s))$;
2. if $s^1$ and $s^2$ are equal on $a(x)(s^1)$ then $d(x)(s^1) = d(x)(s^2)$, $p(x)(s^1) = p(x)(s^2)$, $a(x)(s^1) = a(x)(s^2)$, $r(x)(s^1) = r(x)(s^2)$ and the memory states $f(x)(s^1)$ and $f(x)(s^2)$ are equal on the set $r(x)(s^1)$;
3. if $I(s) \cap a(x)(s) = \emptyset$, then $d(x)(s)$ is false;
4. $a(x_0)(s) = A(f(x_0)(s))$, $a(y_0)(s) = \emptyset$ and for every $x \in \{x_0, y_0\}$ the memory states $s$ and $f(x)(s)$ are equal on the set $A(f(x)(s))$;