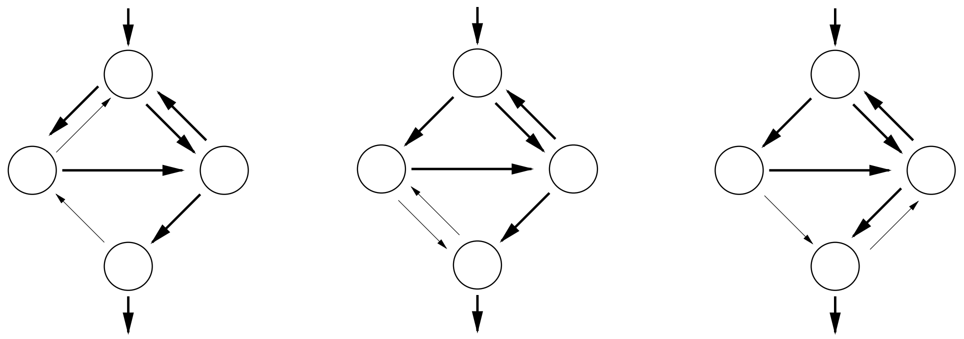
The Power of Programs with Restricted Control Structure

(Preliminary Version)

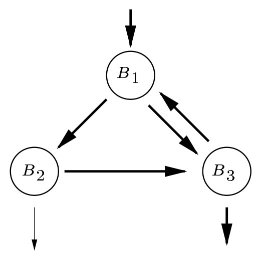
Holger Petersen

Universität Stuttgart, Institut für Informatik
Breitwiesenstraße 20–22, D-70565 Stuttgart, Germany
petersen@informatik.uni-stuttgart.de

In [1] minimal programs violating the paradigms of structured programming are discussed. The control-flow of these programs can most easily be explained graphically, where each circle represents a sequence of statements terminated by a conditional. Control is transferred as indicated by arrows (why some of the arrows are printed in bold-face will be explained below).



The purpose of the present paper is to confirm the remark from [1], that probably very complex programs can be written concisely with the help of these structures. The remark holds in a very strong sense, since universal computations can be carried out by even simpler programs composed of only three segments of code. These sequences will be denoted by B_i, $1 \leq i \leq 3$ (note that only one of the branches is ever taken by B_2).



It will be assumed that the blocks start with a corresponding label B_i and that the underlying programming language provides integer variables and arithmetical operations add, subtract, and multiply. If indirect addressing is available (arrays) we can even eliminate multiplication.

Theorem 1 *Programs with a control structure as given above can compute every partial recursive function in the following sense. If $f(n)$ is a p.r. function then there is such* *a program that, when started with variable A set to 2^n and some auxiliary variables to constant values, eventually stops with $A = 2^{f(n)}$ in case $f(n)$ is defined.*

Proof: We use ideas from [3, 4, 2]. In order to simplify the proof we describe a simulation of a rather elementary device, the two-counter automaton from [3]. For technical reasons we use the following types of instructions for counters A and X:

test n_0, n_1: Transfer control to instruction n_0 under the condition that A equals 0 else to instruction n_1.

$A := A + 1$, $A := A - 1$: Increment resp. decrement A.

$A \leftrightarrow X$: Exchange A and X.

We will briefly discuss why this set of instructions is equivalent to the one from [3]. Counter X can be manipulated independently by temporarily exchanging it with A. It is clear that the combined decrement and test instruction can be split into two instructions. Finally in [3] *every* instruction contains a target for a branch, while we assume by default that control is transferred from instruction i to $i + 1$ unless an instruction of the first type is executed. By inserting unconditional jumps with $n_0 = n_1$ we can simulate these instructions.

Now suppose a sequence $\text{inst}_1, \text{inst}_2, \ldots \text{inst}_k$ of k instructions of the above type is given. Execution starts in inst_1 with $A = 2^n$, $X = 0$ and terminates when control is transferred to inst_k.

The execution of such a sequence of instructions will be simulated by a program equipped with variables A, X, Z, $I_1, \ldots I_k$, A', X', $I'_1, \ldots I'_k$. While A and X represent the two counters of the simulated machine, the additional variables have the following interpretation. Z and $I_1, \ldots I_k$ are flag variables which are either 0 or 1. Z stores the information, whether $A = 0$, while I_i is 1 if and only if the instruction currently executed is inst_i. The primed variables hold intermediate values.

The main part of the simulator is block B_1. It consists of the following assignments.

$$
\begin{array}{rcl}
X' & := & X + \sum\limits_{i:\text{inst}_i=\text{`}A\leftrightarrow X\text{'}} I_i \cdot (A - X); \\
A' & := & A + \sum\limits_{i:\text{inst}_i=\text{`}A\leftrightarrow X\text{'}} I_i \cdot (X - A) + \sum\limits_{i:\text{inst}_i=\text{`}A:=A+1\text{'}} I_i - \sum\limits_{i:\text{inst}_i=\text{`}A:=A-1\text{'}} I_i; \\
I'_1 & := & \sum\limits_{i:\text{inst}_i=\text{`test }1,\ n_1\text{'}} I_i \cdot Z + \sum\limits_{i:\text{inst}_i=\text{`test }n_0,\ 1\text{'}} I_i \cdot (1 - Z); \\
\vdots & \vdots & \vdots \\
I'_m & := & \sum\limits_{i:\text{inst}_i=\text{`test }m,\ n_1\text{'}} I_i \cdot Z + \sum\limits_{i:\text{inst}_i=\text{`test }n_0,\ m\text{'}} I_i \cdot (1 - Z) + \sigma(m - 1); \\
\vdots & \vdots & \vdots \\
I'_k & := & \sum\limits_{i:\text{inst}_i=\text{`test }k,\ n_1\text{'}} I_i \cdot Z + \sum\limits_{i:\text{inst}_i=\text{`test }n_0,\ k\text{'}} I_i \cdot (1 - Z) + \sigma(k - 1);
\end{array}
$$

$$X := X'; \quad A := A'; \quad I_1 := I'_1; \ldots I_k := I'_k; \quad Z := 0$$

where

$$\sigma(m) = \begin{cases} I_m & \text{inst}_m \text{ is not of the form 'test } n_0, n_1\text{'}, \\ 0 & \text{else.} \end{cases}$$

Block B_1 is terminated by a conditional statement 'if $A = 0$ then goto B_2 else goto B_3'.

The blocks B_2 and B_3 are given by '$Z := 1$; goto B_3' and 'if $I_k = 1$ then stop else goto B_1'. Initially $I_1 = 1$ and $I_i = 0$ for $2 \leq i \leq k$, $A = 2^n$, $Z = X = 0$. Then every execution of B_1 simulates one step of the counter-automaton. □

Since we multiplied by 0 or 1 only we can simulate the multiplication in the preceding proof with the help of an array. For if Y is an array with at least 2 entries, the product $D = GH$ can be computed by $Y[0] := 0; Y[1] := G; D := Y[H]$ if H is either 0 or 1.

Corollary 1 *All nontrivial properties of functions computed by programs of the types described above are undecidable.*

The above structure can easily be identified in the programs from [1]. The relevant arrows are drawn in bold-face, the superfluous circles can be filled with dummy statements.

References

[1] V. Claus. Total unstrukturierte Programme. *GMD-Spiegel*, (4):46–50, 1987.

[2] A. R. Meyer and D. M. Ritchie. The complexity of loop programs. In *Proc. 22th National Conference of the ACM*, pages 465–469. Thompson Book Co., 1967.

[3] M. L. Minsky. Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines. *Annals of Mathematics*, 74:437–455, 1961.

[4] M. L. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, Englewood Cliffs, 1967.