A THEORETICAL STUDY ON THE TIME ANALYSIS OF PROGRAMS

Akeo Adachi
Department of Academic and Scientific Programs, IBM Japan
Roppongi, Minato-ku, Tokyo, 106 Japan

Takumi Kasai
IBM Thomas J. Watson Research Center, P.O. Box 218
Yorktown Heights, N.Y. 10598, U.S.A.

Etsuro Moriya
Department of Mathematics, Tokyo Woman's Christian University
Zenpukuji, Suginami-ku, Tokyo, 167 Japan

0. Introduction

Meyer and Ritchie [1][2] introduced the notion of loop programs and classified the primitive recursive functions syntactically with the help of loop programs into the hierarchy $L_0 < L_1 < L_2 < \ldots$ by restricting the depth of loop nesting, i.e., $L_n$ is the class of functions computed by loop programs whose depth of loop nesting is not greater than $n$. They also showed that each class $L_n$ can be characterized by computational complexity, measured by the amount of time required on loop programs to compute the functions. In particular, a function is elementary in the sense of Kalmár [3], i.e., belongs to $L_2$, if and only if it can be computed by a loop program whose computing time is bounded by a $k$-fold exponential function of its inputs for some $k$, while there is general agreement that those computations which are exponentially difficult in time are practically intractable. In this respect, many people feel that even the class $L_2$ is still too inclusive in the sense of "practical computation".

In the earlier paper [4] the authors attempted an investigation to obtain a substantial subclass of $L_2$ reflecting practical computation, where the notion of loop programs is extended so as to include additional types of primitive statements such as $x \leftarrow x^2$ and IF-THEN-ELSE, and the use of arrays is allowed as well. It is proved that if such an extended loop program satisfies a certain syntactical restriction called "simplesness", then the computing time of the program is bounded by a polynomial of its inputs whose degree can be effectively determined only by the depth of loop nesting. This is worthy of notice, since it says that we can know syntactically a practical estimation of the time required to execute a
given "simple loop program" before execution.

Here in this paper, on the basis of our earlier work [4], we make a rather precise analysis of time complexity of simple loop programs. We present an algorithm which gives an accurate upper bound of the computing time of any given simple loop program; a slight modification of the algorithm gives a lower bound.

1. Simple Loop Programs

First we review a basic result from Kasai & Adachi [4], being the starting point of our investigation.

Definition. Let $C$, $S$ and $A$ be fixed mutually disjoint countable sets of symbols. An element of $C$, $S$ or $A$ is called a control variable, simple variable or array name, respectively. Let $\text{Var}$ denote the set of all variables, that is,

$$\text{Var} = C \cup S \cup \left\{ a[i] \mid a \in A, i \in N \right\},$$

where $N = \{0, 1, 2, \ldots\}$. A loop program is a statement over $\text{Var}$ defined recursively as follows, where $\overline{A} = \{ a[i] \mid a \in A, i \in \text{OUS} \}$,

\begin{align*}
<\text{atomic statement}> & := u \rightarrow u+1 \mid v \rightarrow v+1 \mid v \rightarrow u \mid u \rightarrow c \\
<\text{loop statement}> & := \text{LOOP } x \text{ DO } \langle \text{statement} \rangle \text{ END} \\
<\text{condition}> & := w_1 = w_2 \mid w_1 \neq w_2 \\
<\text{statement}> & := <\text{atomic statement}> \mid \langle \text{loop statement} \rangle \mid \langle \text{condition} \rangle \langle \text{statement} \rangle \langle \text{statement} \rangle \\
\end{align*}

Definition. A function $d: \text{Var} \rightarrow \mathbb{N}$ is called a memory. We denote the set of memories by $D$. Let $P$ be a loop program, then $P$ realizes the partial function $\hat{P}: D \rightarrow \mathbb{N}$. The time complexity of $P$ is the function $\text{time}_P: D \rightarrow \mathbb{N}$ such that $\text{time}_P(d)$ is the number of atomic statements executed by $P$ under an initial memory $d$. The definition of $\hat{P}$ and $\text{time}_P$ is straightforward so that we omit the details.

Definition. For each loop program $P$, we define the relation $>_P$ on $C_P$ as follows, where $C_P$ denotes the set of control variables appearing in $P$.

We write $x >_P y$ if and only if the program $P$ includes a statement of the form $\text{LOOP } x \text{ DO } Q \text{ END}$, and $Q$ includes $y \rightarrow y+1$.

We say that $P$ is simple if there is no sequence of control variables $x_1, x_2, \ldots, x_k$, $k > 1$, such that