DRAFT

Program Structure and Computational Complexity

Dennis M. Ritchie

January, 1968

Committee:

Ottinger
Greibach
Putnam
Blum

SYNOPSIS

The major purpose of this thesis is to show that when the language in which computations are described is restricted suitably, there can be an effective relationship between the complexity of a program and that of the computation it describes. We give two examples.

The first example is that of Loop programs. A Loop program is a finite sequence of instructions for manipulating non-negative, unbounded integers stored in registers; the instructions allow incrementing registers by unity, setting registers to zero, and moving the contents of registers. The only control instructions consist of Loops; there is a kind of Loop for each number $n \geq 1$. A Loop with $n = 1$ causes the execution of a portion of the program to be repeated a predetermined number of times equal to the current contents of a register. Loops may be nested, one inside another, to any fixed depth; but Loops with $n > 1$ are defined so as to make a Loop of type $n+1$ equivalent to a variable depth of nesting of Loops of type n.

Each Loop program is assigned an ordinal α, where $0 \leq \alpha < \omega^{\omega}$, which is intended to be the measure of complexity of the program. The ordinal assigned to a program depends effectively on the program, and measures the depth of nesting of the various kinds of Loops.

The idea of Loop programs whose only Loops have $n = 1$, although original with the author, is not unique to him; for example Minsky [17, pp. 212-215] discusses briefly the same idea. Some results of

the theory of such Loop programs have been announced by the author [22] and published by Meyer and the author [15,16]. The generalization with Loop instructions for each $n > 1$ is believed to be entirely new.

For each ordinal α, $0 \leq \alpha < \omega^{\omega}$, we define a function f_{α}. The function is recursive, strictly increasing, and if $\alpha > \beta$, f_{α} majorizes f_{β}. The definition of f_{α} for finite ordinals α is the same as the f_n of [15,16] and in general is a modification of the function W_{α} used by Robbin [25] for much the same purposes. The major results on Loop programs can be stated as follows: for each Loop program $\underset{\sim}{P}$ assigned ordinal α there is a number ~~of~~ p effectively derived from $\underset{\sim}{P}$ such that $\underset{\sim}{P}$ with inputs $x_1,\ldots,x_n$ requires no more than $f_{\alpha}^{(p)}(\max\{x_1,\ldots,x_n\})$ steps to halt (Theorem (3.6)). The notation $f_{\alpha}^{(p)}$ means f_{α} composed with itself p times. There are some programs $\underset{\sim}{P}$ assigned ordinal α which do in fact require $f_{\alpha}^{(p)}(x)$ steps to halt when given input x (Theorem (4.7)). A precise definition of the number of steps used by a program is a by-product of a formalization of Loop programs presented in §2.

Further results on Loop programs, and much of the rest of the thesis, use heavily the notion of computation-time closure. A set of functions is computation-time closed when both of the following are true: if a function f is in the set, a function b is in the set where b bounds the time required to compute f on a Turing machine; if b is in the set and b bounds the time required to compute f on a Turing machine, f is in the set.

If $\mathcal{L}_\alpha$ is the class of functions computable by programs assigned an ordinal less than or equal to α, each $\mathcal{L}_\alpha$ for $\alpha \geq 2$ is computation-time closed. This allows us to show the following: each class $\mathcal{L}_\alpha$ for $\alpha \geq 2$ is closed under limited recursion (Theorem (6.8)); each class $\mathcal{L}$ for $\alpha \geq 2$ can be characterized in arithmetic terms, without reference to Turing machines or Loop programs (Theorem (8.3)); if a program $\underset{\sim}{P}$ assigned ordinal α requires only $f_\alpha^{(p)}$ steps as a function of its inputs where $\beta < \alpha$, then $\underset{\sim}{P}$ can be rewritten effectively to yield a program $\underset{\sim}{P}'$ which is equivalent to $\underset{\sim}{P}$ but is assigned ordinal β. However, it is in general undecidable whether these hypotheses hold for $\underset{\sim}{P}$ (Theorem (12.6)).

The second example of a restricted program language is that describing the multiple recursive functions [19,21]. Each multiple recursive function can be defined by a formal system of equations which can effectively be assigned an ordinal $\alpha < \omega^\omega$. If $\mathcal{R}_\alpha$ is the class of functions defined by systems of equations assigned ordinal α, then $\bigcup_{\alpha<\omega^n} R_\alpha$ is the class of n-recursive functions; Péter shows [21] that the 1-recursive functions are the same as the primitive recursive functions. Much the same theorems are proved for $\mathcal{R}_\alpha$ as for $\mathcal{L}_\alpha$. In particular, $\mathcal{R}_\alpha$ is computation-time closed for $\alpha \geq 2$ (Theorem (9.3)); if $f \in \mathcal{R}_\alpha$, $f(x_1,\ldots,x_n)$ can be computed by a Turing machine in $f_{1+\alpha}^{(p)}(\max\{x_1,\ldots,x_n\})$ steps for some p which is effectively found from the recursion equations defining f (Theorem (9.1)); $f_{1+\alpha} \in \mathcal{R}_\alpha$ (Theorem (8.3)). These facts alone show: for $\alpha \geq 2$, $\mathcal{L}_{1+\alpha} = \mathcal{R}_\alpha$ (Theorem (10.1)).

The same kind of techniques are applied to the hierarchies of Axt [2], Grzegorczyk [9] and Robbin [25]. All of these hierarchies are shown to be identical to a portion of the $\mathcal{L}_\alpha$ and $\mathcal{R}_\alpha$ hierarchies, and thus to each other. Specifically, if $\mathcal{P}_\alpha$, $\alpha < \omega$, are the Axt classes, $\mathcal{L}_\alpha = \mathcal{P}_\alpha$ for $\alpha \geq 4$ (Theorem (10.4)); if $\mathcal{E}^G_\alpha$, $\alpha < \omega$, are the Grzegorczyk classes, $\mathcal{L}_\alpha = \mathcal{E}^G_{\alpha+1}$ for $2 \leq \alpha < \omega$ (Theorem (10.9)); if $\mathcal{E}_\alpha$, $\alpha < \omega^\omega$, are a trivial modification of the Robbin classes, $\mathcal{L}_\alpha = \mathcal{E}_\alpha$ for $\alpha \geq 2$ (Theorem (10.6)). All of these results are straightforward using computation-time closure. Not all are new, however. According to a personal communication, Axt showed $\mathcal{P}_\alpha = \mathcal{E}^G_{\alpha+1}$ for $\alpha \geq \alpha_o$, $\alpha_o \approx 7$ but used a different method. Meyer showed the same thing independently [14], using a method like ours. Robbin [25] showed that $\bigcup_{\alpha<\omega^n} \mathcal{E}_\alpha$ is the same as the class of n-recursive functions; however, he did not subdivide the latter class after the manner of our $\mathcal{R}_\alpha$. It should be mentioned as well that Robbin established the identity of the n-recursive functions and those functions defined by ordinal recursion over certain "standard" well-orderings of type ω^{ω^n}, and also the classes of functions occurring in a restricted version of the Kleene subrecursive hierarchy [13]. It seems likely that by closer study equality of these classes could be established at each ordinal.

Chapters II, III, and IV stud~~ied~~y Loop programs and multiple recursive functions; Chapter V contains three applications of the tools developed in the earlier chapters. The most important of these, as we have indicated, is the idea of computation-time closure. An early appearance of this idea, without an explicit name, was in R. W. Ritchie [23], who used it to characterizing classes which form a hierarchy of elementary functions. Cobham [6] pointed out how each Grzegorczyk [9] class could be characterized in terms of the property, after the manner of our Theorem (6.2), which states $\mathcal{L}_\alpha$ is precisely the class of functions computable by a Turing machine in a time bounded by $f_\alpha^{(p)}$ for some p. As we mentioned, Meyer [14] and also Robbin [25] used the idea as well.

Chapter V, §13, discusses unnested and bounded n-recursion [20, 21] and their relation to the $\mathcal{L}_\alpha$ classes thus strengthening some theorems of Péter [20,21]. §14 examines the properties of computation-time closed classes of functions in general; its major results are that each $\mathcal{L}_\alpha$ includes a sequence of classes, all computation-time closed and closed under limited recursion and substitution, which is densely ordered under set inclusion (Theorem (14.14)); also, $\mathcal{L}_\alpha$ includes an infinite sequence of classes with the same closure properties but pairwise incomparable under set inclusion (Theorem (14.15)). These two results were obtained in collaboration with Albert R. Meyer. §15 applies Lemma (14.13) to obtain a strengthened version of the Super Speed-up theorem of Blum [4]. Among the consequences of our Theorem (15.3) is that there are functions lying very low in the $\mathcal{L}_\alpha$ hierarchy whose computation can be sped up, in Blum's sense, very considerably.