Chapter 6. Array Dataflow Analysis

Paul Feautrier
Laboratoire PRiSM, Université de Versailles Saint-Quentin
78035 VERSAILLES CEDEX, FRANCE
Paul.Feautrier@ prism.uvsq.fr

Summary. While mathematical reasoning is about fixed values, programs are written in term of memory cells, whose contents are changeable values. To reason about programs, the first step is always to abstract from the memory cells to the values they contains at a given point in the execution of the program. This step, which is known as Dataflow Analysis, may use different techniques according to the required accuracy and the type of programs to be analyzed.

This paper gives a review of the ad hoc techniques which have been designed for the analysis of Array Programs. An exact solution is possible for the tightly constrained static control programs. The method can be extended to more general programs, but the results are then approximation to the real dataflow. Extensions to complex statements and to the interprocedural case are also presented.

The results of Array Dataflow Analysis may be of use for program checking, program optimization and parallelization.

Acknowledgement. The work which is reported here has taken many years of research by many peoples to evolve from the rough sketch in [Fea88a] to the present state of affairs. I would like to acknowledge contributions by Denis Barthou and Jean-François Collard [BCF97], by Vincent Lefebvre [LF97] and by Arnauld Leservot [Les96].

1. Introduction

There are many situations in which one needs to thoroughly understand the behavior of a program. The most obvious one is at program checking time. If we could extract a description of a program as, e.g., a set of mathematical equations and compare it to a specification, also given in the same medium, debugging would become a science instead of an art. Reverse engineering is another case in point. But the most important application of such analyses is to optimization. Each optimization has to be proved valid in the sense that it does not modifies the program ultimate results. To achieve this, we have to know, in a more or less precise way, what these results are intended to be. Since the most aggressive type of optimization a program can be subjected to is parallelization, understanding a program before attempting to parallelize it is a very important step.

Now, since the time of Von Neuman, programs are written in term of “variables” which are in fact symbolic names for memory cells. Values are never given, or even named, but always alluded to as “the present content

1 except in the case of constants.

 Springer-Verlag Berlin Heidelberg 2001
of cell \(x\). On the other hand, in mathematics, the subject of discourse is always a value which never change, albeit it can be unknown or arbitrary. The value in a given memory cell can be modeled as a function of time (that function may be constant).

Obviously, “time” here is not physical time. Besides the fact that exhibiting such a function would be nearly impossible, it would have the added inconvenience of not being portable among different computers. We will use a logical time, to be defined later. The only requirement is that there must be a “time arrow”: time must belong to an ordered set. Since the state of a computer memory does not change except at each execution of an assignment, logical time is not continuous but discrete. Each time step is an operation of the computer, which corresponds, from the point of view of the programmer, to the execution of an instruction. For program analysis purposes, there is some leeway in the definition of an operation. It may be the execution of a machine instruction, as in the case of Instruction Level Parallelization, or the execution of an assignment statement, as in most of this paper, or the execution of a complex statement, as in Sect. 4.

If we stipulate that the meaning of a program is given by expressing the value of variables as a function of (logical) time, then dataflow analysis is the process of extracting properties of these functions from the program text. These properties may be of widely varying precision. In some cases, one may exhibit a closed formula for the function. In other cases, one may only knows that it has positive values. In the most frequent cases, one has to be content with relations between values taken either at the same time (Floyd’s assertions [Flo67]) or at different times. As before, these relations may be more or less precise. We will show that, for a simple but useful category of programs, the result of Array Dataflow Analysis is a system of equations relating the values of variables at distinct time points.

Dataflow analysis is based on the observation that the value one may retrieve from a memory cell is the one which was written last. In the scalar case, this allows one to write dataflow equations, which may be solved either by iterative methods or by direct methods. In the case of array cells, the problem is more difficult because there is no simple method for deciding if two references to the same array are references to the same cell or not: two occurrences of \(a[i]\) are references to the same cell iff \(i\) has not been modified in between. Conversely, it may happen that \(a[i]\) and \(a[j]\) refer to the same cell if the values \(i\) and \(j\) are equal.

There is a general method for devising dataflow analyses [CC77]. One starts from a semantical description of the source language, and then one abstracts the features of interest by constructing a nonstandard semantics. The

\[2\] We will adhere to the following convention: identifiers will always be written in a Teletype font. Their values at a given time will always be denoted by the same letter in an italic font. If necessary, the time will be indicated by various devices (accents, subscripts, arguments).