own pubs

IFIP CONGRESS 65

Symposium: Special Purpose Programming Languages

Title : A programming Language for Processor Construction

Author : C.A.R.Hoare
48, Sebright Road,
Barnet,
Herts, ENGLAND

Language : English

Precis : A language has been designed to enable complex multi-pass processors to be specified in the same straightforward manner as single-pass processors. The main innovation of the language is that it permits the reference to and use of the value of a variable in advance of the first assignment to that variable. This reversal of one of the fundamental laws of programming is shown to lead to perfectly well defined algorithms which are susceptible of mechanical implementation.

as a result of Negative comments of Naur* at the congress, I never wrote this up for publication (note added 1998)

he said writing a nine-pass compiler was easy

Summary: The design of algorithms for the translation of advanced programming languages into efficient programs expressed in the machine code of existing computers is an undertaking which requires a profound understanding of the general nature of the source and object languages, as well as a detailed analysis of the numerous individual cases which are capable of special treatment. These two requirements are quite independent of the language in which the translation algorithm is written. However a good language can relieve the programmer of a number of tedious administrative tasks, such as storage allocation and subroutine entry, which tend to consume a disproportionate amount of effort, and to distract him from his more essential tasks. Some of the most serious of the administrative problems associated with the design of processors arise from the use of multiple pass methods, since the algorithm has to be split into several independent segments, and intermediate languages must be designed to establish communication between each segment and its successor. Even when the processor has been designed and implemented, the complexity of the multiple pass structure makes it difficult to trace and correct an error, and virtually impossible to make improvements in the light of experience.

The main logical limitation on a single-pass compiler is that the compilation of any section of the object program can be based only on information gleaned from scanning earlier parts of the source program, and cannot depend on any knowledge of the characteristics of the program as a whole. This is a result of the fact that calculation can only be made in terms of variables whose values have already been assigned to them, so that information can be carried in one direction only. If, however, this apparently fundamental rule of programming is relaxed, and the compilation of a section of the object program is allowed to depend on the value of a variable which will in fact be assigned only when some later part of the source program has been read in, then the logical limitations of single-pass compilers can be removed, and there never arises the need for specifying multiple-pass processors. The great advantage of this is that single-pass processors are relatively easy to specify, especially with the assistance of syntax-directed techniques.

The second reason why many complex processors are split into multiple passes is a shortage of space in the immediate access store of the computer. This, however, is a practical problem rather than a conceptual one, and can be fairly readily solved in the implementation of the language in which the processor is written.

The recommended form of a single-pass syntax-directed processor is the same as that described by Brooker and Morris in the Computer Journal Vol 3 No.3 (1960). A full set of syntactic definitions of the source language is given; and with some of them is associated a translation routine which processes those portions of the source text which satisfy the syntactic definition. It is recommended that the language of the translation routines be greatly simplified; it should be confined to assignment statements and calls of subsidiary translation routines, which will process subsections of the text being processed by the given routine. In addition, it is essential to introduce ALGOL-type conditional expressions, and a block structure together with declarations of integer and string variables. It is also helpful to add a few standard functions for dictionary handling. In spite of the extreme simplicity of the language, the addition of the forward-looking facility makes it powerful enough for the purpose for which it was designed.

The first stage of the application of the processor to a particular source program can follow very closely the method of Irons as described in Annual Review in Automatic Programming Vol 3 (1963) The result of the first stage is not, however, the object program itself, but rather a program written in the language of the translation routines; and it is only when this program is obeyed that the object program is actually compiled. But this program is impossible to obey in the normal way as it stands, since it may refer to the value of a variable before that value has been assigned; and therefore a preliminary transformation has to be made. Firstly, the program is transformed into an identically equivalent program from which all inner blocks have been removed, and in which no identifier occurs more than once on the left hand side of an assignment statement. The satisfaction of these conditions is made possible by the absence of go to statements, conditional statements, and for statements from the language of the translation routines. The next stage is to rearrange sequence of the statements of the program in such a way that every variable occurs on the left hand side of an assignment before its first occurrence on the right hand side. This can be done by the same topological sorting algorithms that is widely used in PERT programs. The occurrence of an inconsistent cycle which makes the sort impossible would be a symptom of a grave programming error; in fact, of an attempt to define the value of a variable in terms of its own, as yet unknown, value. After the sort is completed, the resulting program can be obeyed in the normal way to produce as output the value of the object program.