ALGOL 48 AND ALGOL 50—ALGOLIC LANGUAGES IN MATHEMATICS

Abstract

This article describes how to express programs with assignment statements and conditional **go to**s in mathematical logic without any programming constructs used in Fortran and Algol. The names come from imagining mathematicians devising a programming language in 1948 and elaborating it in 1950. There are just two innovations—regarding the values of variables as functions, of a natural number valued time, e.g. $x(t)$ instead of a program variable x, and regarding a program counter $pc(t)$ as just another variable.

This way of writing programs has a number of advantages. One is that statements and proofs of correctness are just ordinary proofs in the domain of variables supplemented by Peano arithmetic.

We introduce the "programming languages" Algol 48 and Algol 50 to illustrate in a simpler setting some ideas to be used in Elephant 2000. These are the explicit use of time in a programming language and the representation of the program by logical sentences. The former permits a direct expression of the operational semantics of the language, and the latter permits proofs of properties of programs without any special theory of programming. The properties are deduced from the program itself together with axioms for the domain.

We use these names, because the languages cover some of the ground of Algol 60 but use only a mathematical formalism— old fashioned recursion equations—that precedes the development of programming languages. They are programming languages I imagine mathematicians might have created in 1950 had they seen the need for something other than machine language. Algol 48 is a preliminary version of Algol 50 just as Algol 58 was a preliminary version of Algol 60.

Consider the Algol 60 fragment.

```
0       start :  p := 0;                          .
1                i := n;
2       loop :   if i = 0 then go to done;
3                p := p + m;
4                i := i − 1;
5                go to loop;
6       done :
```

The program computes the product mn by initializing a partial product p to 0 and then adding m to it n times. The correctness of the Algol 60 program is represented by the statement that if the program is entered at *start* it will reach the label *done*, and when it does, the variable p will have the value mn. Different program verification formalisms represent this assertion in various ways, often not entirely formal.

Its partial correctness is conventionally proved by attaching the invariant assertion $p = m(n - i)$ to the label *loop*. Its termination is proved by noting that the variable i starts out with the value n and counts down to 0. This proof is expressed in various ways in the different formalisms for verifying programs.

0.1 Algol 48

In Algol 48 we write this algorithm as a set of old fashioned recursion equations for three functions of time, namely $p(t)$, $i(t)$ and $pc(t)$, where the first two correspond to the variables in the program, and $pc(t)$ tells how the "program counter" changes. The only ideas that would have been unconventional in 1948 are the explicit use of a program counter and the conditional expressions. We have

$$\begin{aligned} p(t+1) = \quad & \textbf{if } pc(t) = 0 \textbf{ then } 0 \\ & \textbf{else if } pc(t) = 3 \textbf{ then } p(t) + m \\ & \textbf{else } p(t), \end{aligned}$$

$$\begin{aligned} i(t+1) = \quad & \textbf{if } pc(t) = 1 \textbf{ then } n \\ & \textbf{else if } pc(t) = 4 \textbf{ then } i(t) - 1 \\ & \textbf{else } i(t), \end{aligned}$$

and

$$\begin{aligned} pc(t+1) = \quad & \textbf{if } pc(t) = 2 \wedge\ i(t) = 0 \textbf{ then } 6 \\ & \textbf{else if } pc(t) = 5 \textbf{ then } 2 \\ & \textbf{else } pc(t) + 1. \end{aligned}$$

The correctness of the Algol 48 program is represented by the sentence

$$\forall m\ n(n \geq 0 \rightarrow \forall t(pc(t) = 0 \rightarrow \exists t'(t' > t \wedge pc(t') = 6 \wedge p(t') = mn)))$$

This sentence may be proved from the sentences representing the program supplemented by the axioms of arithmetic and the axiom schema of mathematical induction. No special theory of programming is required. The easiest proof uses mathematical induction on n applied to a formula involving $p(t) = m(n - i(t))$.

Algol 48 programs are organized quite differently from Algol 60 programs. Namely, the changes to variables are sorted by variable rather than sequentially by time.

0.2 Algol 50

However, by reifying variables, Algol 50 permits writing programs in a way that permits regarding programs in this fragment of Algol 60 as just sugared versions of Algol 50 programs.

Instead of writing $var(t)$ for some variable var, we write $value(var, \xi(t))$, where ξ is a state vector giving the values of all the variables. In the above program, we'll have $value(p, \xi(t))$, $value(i, \xi(t))$ and $value(pc, \xi(t))$.

The variables of the Algol 60 program correspond to functions of time in the above first Algol 50 version and become distinct constant symbols in the version of Algol 50 with reified variables. Their distinctness is made explicit by the "unique names" axiom

$$i \neq p \wedge i \neq pc \wedge p \neq pc.$$

In expressing the program we use the assignment and contents functions, $a(var, value, \xi)$ and $c(var, \xi)$, of (McCarthy 1963) and (McCarthy and Painter 1967). $a(var, value, \xi)$ is the new state ξ' that results when the variable var is assigned the value $value$ in state ξ. $c(var, \xi)$ is the value of var in state ξ.

As described in those papers the functions a and c satisfy the axioms.

$$c(var, a(var, val, \xi)) = val,$$

$$var1 \neq var2 \rightarrow c(var2, a(var1, val, \xi)) = c(var2, \xi),$$

$$a(var, val2, a(var, val1, \xi)) = a(var, val2, \xi),$$

and

$$var1 \neq var2 \rightarrow a(var2, val2, a(var1, val1, \xi)) = a(var1, val1, a(var2, val2, \xi)).$$

The following function definitions shorten the expression of programs. Note that they are just function definitions and not special constructs.

$$step(\xi) = a(pc, value(pc, \xi) + 1, \xi),$$

$$goto(label, \xi) = a(pc, label, \xi).$$

We make the further abbreviation $loop = start + 2$ specially for this program, and with this notation our program becomes

$$\begin{array}{ll}
\forall t & (\xi(t+1) = \\
& \textbf{if } c(pc, \xi(t)) = start \\
& \quad \textbf{then } step\ a(p, 0, \xi(t)) \\
& \textbf{else if } c(pc, \xi(t)) = start + 1 \\
& \quad \textbf{then } step\ a(i, n, \xi(t)) \\
& \textbf{else if } c(pc, \xi(t)) = loop \\
& \quad \textbf{then } (\textbf{if } c(i, \xi(t)) = 0 \textbf{ then} \\
& \qquad goto(done, \xi(t)) \\
& \qquad \textbf{else } step\ \xi(t)) \\
& \textbf{else if } c(pc, \xi(t)) = loop + 1 \\
& \quad \textbf{then } step\ a(p, c(p, \xi(t)) + m, \xi(t)) \\
& \textbf{else if } c(pc, \xi(t)) = loop + 2 \\
& \quad \textbf{then } step\ a(i, c(i, \xi(t)) - 1, \xi(t)) \\
& \textbf{else if } c(pc, \xi(t)) = loop + 3 \\
& \quad \textbf{then } goto(loop, \xi(t)) \\
& \textbf{else } \xi(t+1))
\end{array}$$

In Algol 50, the consequents of the clauses of the conditional expression are in 1-1 correspondence with the statements of the corresponding Algol 60

program. Therefore, the Algol 60 program can be regarded as an abbreviation of the corresponding Algol 50 program. The (operational) semantics of the Algol 60 program is then given by the sentence expressing the corresponding Algol 50 program together with the axioms describing the data domain, which in this case would be the Peano axioms for natural numbers. The transformation to go from Algol 60 to Algol 50 would be entirely local, i.e. statement by statement, were it not for the need to use statement numbers explicitly in Algol 50.

Program fragments can be combined into larger fragments by taking the conjunction of the sentences representing them, identifying labels where this is wanted to achieve a **go to** from one fragment to another and adding sentences to make sure that the program counter ranges don't overlap.

The correctness of the Algol 50 program for multiplication by addition is expressed by

$$\begin{aligned}\forall t \xi_0 (c(pc, \xi(t)) = start \quad & \wedge \xi(t) = \xi_0 \\ & \rightarrow \exists t' (t' > t \wedge c(p, \xi(t')) = mn \\ & \quad \wedge c(pc, \xi(t')) = done \\ & \quad \wedge \forall var (\neg (var \in \{p, i, pc\}) \rightarrow c(var, \xi(t')) = c(var, \xi_0))))\end{aligned}$$

Note that we quantify over all initial state vectors. The last part of the correctness formula states that the program fragment doesn't alter the state vector other than by altering p, i and pc.

We have not carried the Algol 50 idea far enough to verify that all of Algol 60 is conveniently representable in the same style, but no fundamental difficulties are apparent. In treating recursive procedures, a stack can be introduced, but it would be more elegant to do without it by explicitly saying that the return is to the statement after the corresponding procedure call and variables are restored to their values at the time of the call. This requires the ability to parse the past, needed also for Elephant 2000.

We advocate an extended Algol 50 for expressing the operational semantics of Algol-like programming languages, i.e. for describing the sequence of events that occurs when the program is executed. However, our present application is just to illustrate in a simpler setting some features that Elephant will require. In particular, proper treatment of calling a function procedure with side-effects will require a state that can have a value during the evaluation of an expression.

Nissim Francez and Amir Pnueli (see references) used an explicit time for similar purposes. Unfortunately, they abandoned it for temporal logic.

While some kinds of temporal logic are decidable, temporal logic is too weak to express many important properties of programs.