SHAW77
Shaw, M., Wulf, W. A., and London, R.L.; Abstraction and Verification in Alphard : Defining and Specifying Iteration and Generators; CACM 20, 8 (August 1977).

SYNOPSIS

This paper is one of a series by the authors on the programming language Alphard. It presents Alphard's loop construct which is an amalgamation of the 'for' and 'while' loops of other languages. The major difference in Alphard is that the loop control variable can "operate on abstract entities without explicit dependence on the representation of those entities." The authors then derive proof rules for this construct and show that in most cases they are equivalent to the proof rules given for the verification of programs in other languages.

The authors present the following example: if S is a set of integers and we want the sum of the members of S, we would like to write

```
sum <- 0; for x   S do  sum <- sum + x
```

instead of the more conventional

```
for( sum=0, i=1; i <= sizeofS; i++ ) sum += s[i];
or
sum <- 0; p <- S;
while( p != NULL )
   sum += S.value; p <- S.next;
```

which imply an array or a list representation respectively. The first notation suppresses detail about how the iteration over an abstract data structure is performed. In Alphard this abstraction is accomplished using a 'form'. A form is an extension of a base data type that creates a new data type and defines the possible operations on this new type. This new data type is defined by the operations that it optionally inherits from the base type as well as new operations specified in the form body. A generator is a special type of form with two specific operations '&init' and '&next' which are used to control iteration. For example the Alphard statement

```
for  x : gen(y)  while B(x) do 'statements'
```

has the following semantics

- call the &init function of the generator gen with parameter y , this gives x its first value and creates a hidden boolean variable b
- if b is false terminate the loop
- for each iteration call the &next function which gives x and b their next values. Terminate if b is false.

The authors also introduce a more specialized looping construct the 'first' statement which is meant to handle loops with early exits, for example searching a list for a particular value.

The rest of the paper is devoted to the verification of generators and the loops that they are used in. This is done by including pre and post conditions with the &init and &next functions and an invariant which applies to the whole generator. Correctness proofs for loops using the generator can than be derived. Since in the general case the verification procedure can become very involved, proof rules for some simpler constructs the 'pure' while and the 'pure' for are given. It is also shown that in some commonly occurring cases generators can be proved to terminate (assuming that the loop body does).

COMMENTS

The stated goal of the Alphard project is "to increase the equality and the total lifetime cost of real programs". The generator construct is designed to make programs more understandable through data abstraction and more reliable through verification. However, as the authors note, it has restrictions in comparison with other loop constructs. In particular it is not possible to modify the structure on which the generator is operating within the loop that is controlled by the generator.