Auxiliary Variables

Notes on UNITY: 15-90

Jayadev Misra[*]
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712
(512) 471-9547
misra@cs.utexas.edu

7/10/90

1 Introduction

Auxiliary variables are usually employed to record the history of a computation, and, thereby, allow reasoning over the entire computation history. They are typically defined by augmenting the program text; if, for instance, x is a program variable and y is a variable that counts the number of times x has changed value then y may be defined by:

- initially setting y to 0 and,

- transforming a statement of the form
  $x \ := \ e$
  to
  $x \ := \ e \ \| \ y \ := \ y+1 \quad \text{if} \quad x \neq e$

In this note we show (1) how auxiliary variables can be defined directly without resorting to program text, and (2) how they can be employed to state program properties that are not directly expressible in the UNITY-logic.

2 Defining Auxiliary Variables

The auxiliary variable y, as described above, may be defined as follows (m, n are free variables):

initially $y = 0$
$x = m \ \wedge \ y = n \ \ \textit{unless} \ \ x \neq m \ \wedge \ y = n + 1$

We give a few more examples. In the following, m, n, r, S are free variables.

[*]This material is based in part upon work supported by the Texas Advanced Research Program under Grant No. 003658–065 and by the Office of Naval Research Contract N00014-90-J-1640

- Let x, y be integer valued program variables. Let the auxiliary variable c count the number of assignments to x, y that results in $x > y$.

  initially $c = 0$
  $(x, y) = (m, n) \wedge c = r$ *unless* $(x, y) \neq (m, n) \wedge [(x > y \wedge c = r{+}1) \vee (x \leq y \wedge c = r)]$

- Let x be a program variable. Let the auxiliary variable s be the sequence of distinct values assumed by x.

  initially $s = \langle\langle x \rangle\rangle$ $\{\langle\langle x \rangle\rangle$ is the singleton sequence consisting of the value of $x\}$
  $x = m \ \wedge \ s = S$ *unless* $x \neq m \ \wedge \ s = S; x$ $\{;$ is concatenation$\}$

- Let t be a semaphore. Let the auxiliary variables u, d for (*u*p, *d*own) be the number of times t has been increased and decreased, respectively.

  initially $u, d = 0, 0$
  $t = m \ \wedge \ u = n$ *unless* $(t < m \ \wedge \ u = n) \ \vee \ (t = m + 1 \ \wedge \ u = n + 1)$
  $t = m \ \wedge \ d = r$ *unless* $(t > m \ \wedge \ d = r) \ \vee \ (t = m - 1 \ \wedge \ d = r + 1)$

  (The fact that t remains nonnegative—i.e., invariant $t \geq 0$—may be stated separately.)

  We use the definitions of u, d to prove that $t - u + d$ is constant. (Thus, $t - u + d$ is always equal to the initial value of t, since u, d are initially 0.)

  $t = m \ \wedge \ u = n \ \wedge \ d = r$ *unless* $(t = m - 1 \ \wedge \ u = n \ \wedge \ d = r + 1) \ \vee$
  $(t = m + 1 \ \wedge \ u = n + 1 \ \wedge \ d = r)$
  , conjunction of the two *unless*es
  $(t, u, d) = (m, n, r)$ *unless* $(t, u, d) \neq (m, n, r) \ \wedge \ t - u + d = m - n + r$
  , consequence weakening
  $t - u + d$ constant , constant introduction (see [1])

3 Specifications Using Auxiliary Variables

A number of program properties that cannot be directly specified using UNITY-logic may be specified if we assume the existence of appropriate auxiliary variables. Some examples follow.

- Predicate p is eventually stable (i.e., eventually p remains true forever; in linear temporal logic this property is written as $\Diamond\Box\ p$):
  We postulate the existence of an auxiliary boolean variable b satisfying

  true $\mapsto \ p \ \wedge \ b$,
  $p \ \wedge \ b$ stable

- Once variable x exceeds 10 it remains positive: (Note that by observing the value of x in a given state—say $x = 5$—it is impossible to predict if x will be positive in the subsequent states.)
  We postulate an auxiliary boolean variable b satisfying

  $x > 10 \ \Rightarrow \ b$,
  $b \ \wedge \ x > 0$ stable

- Once predicate p holds either it holds forever or q becomes *true* eventually:
  We postulate an auxiliary boolean variable b that becomes *true* whenever p is falsified.

  $p \ \textit{unless} \ b$,
  $b \ \mapsto \ q$

- Either $p \mapsto q$ or $r \mapsto s$ in a given program:
  We postulate an auxiliary boolean variable b where

  b constant,
  $p \ \wedge \ b \ \mapsto \ q$,
  $r \ \wedge \ \neg b \ \mapsto \ s$

- If predicate p holds in the i^{th} step of an execution and predicate r in the k^{th} step of the execution, $k \geq i$, then predicate q holds in the j^{th} execution step, for some j, $i \leq j \leq k$. Note that if p and r hold simultaneously then q also holds at that step.
  We postulate an integer valued auxiliary variable k, satisfying

  $p \ \Rightarrow \ k \leq 0$, $r \ \Rightarrow \ k \geq 0$, $k = 0 \ \equiv \ q$,
  $k < 0 \ \ \textit{unless} \ \ k = 0$

It is not obvious that this definition captures the intent. Therefore, we will show that in any valid execution—i.e., in which there is a q "between" every pair of p, r—a value can be assigned to k at each execution step such that the above properties hold. Conversely, we show that k cannot be assigned value satisfying the above properties in any invalid execution.

First, we show how to assign values to k in a valid execution. In the following p_i, q_i, r_i etc. denote that the corresponding predicate holds after the i^{th} computation step (initially, 0^{th} step has completed). We write k_i to denote k's value after the i^{th} step.

$$k_0 = 0 \quad \text{if} \quad q_0 \quad \sim \quad -1 \quad \text{if} \quad p_0 \ \wedge \ \neg q_0 \quad \sim \quad 1 \quad \text{if} \quad \neg p_0 \ \wedge \ \neg q_0$$

and for $i > 0$,

$$\begin{aligned} k_i = \ & 0 \quad \text{if} \quad q_i \sim \\ & -1 \, \text{if} \quad (p_i \ \wedge \ \neg q_i) \ \vee \ (\neg p_i \ \wedge \ \neg q_i \ \wedge \ \neg r_i \ \wedge \ k_{i-1} < 0) \sim \\ & 1 \quad \text{if} \quad (r_i \ \wedge \ \neg q_i) \ \vee \ (\neg p_i \ \wedge \ \neg q_i \ \wedge \ \neg r_i \ \wedge \ k_{i-1} \geq 0) \end{aligned}$$

Note: The definition for k_i excludes the condition $p_i \ \wedge \ \neg q_i \ \wedge \ r_i$, because this is impossible in a valid execution sequence.

It can be shown, using induction on i, that for all i,

$$p_i \ \Rightarrow k_i \leq 0 \ , \ r_i \ \Rightarrow \ k_i \geq 0, \text{ and } k_i = 0 \equiv q_i$$

The remaining property to prove is: $k < 0$ *unless* $k = 0$, which is equivalent to

$$k_t < 0 \ \Rightarrow \ k_{t+1} \leq 0, \qquad\qquad \text{for all } t, t \geq 0.$$

The proof uses the following three facts that follow from the defining equations for k. For all i,

1. $k_0 < 0 \ \Rightarrow \ p_0$

2. $k_{i-1} \geq 0 \ \wedge \ k_i < 0 \ \Rightarrow \ p_i$
3. $k_{i-1} < 0 \ \wedge \ \neg q_i \ \wedge \ \neg r_i \ \Rightarrow \ k_i < 0$

Given $k_t < 0$, let u be the smallest index such that

$$\langle \forall \ s \ : \ u \leq s \leq t \ :: \ k_s < 0 \rangle$$

Clearly u exists and $u \leq t$, because $k_t < 0$. Now,

if $u = 0$ then $k_0 < 0$ and hence p_0 holds, from (1);
if $u > 0$ then $k_{u-1} \geq 0$ (because u is the smallest index) and hence, from (2), p_u holds.

Thus, in either case, p_u holds. Also, since

$$k_i < 0 \ \Rightarrow \ \neg q_i$$

we have $\langle \forall \ s \ : \ u \leq s \leq t \ :: \ \neg q_s \rangle$. Now consider the following three possibilities for $(t+1)$.

Case 1) q_{t+1}: Then $k_{t+1} = 0$
Case 2) r_{t+1}: Since p_u, $\langle \forall \ s \ : \ u \leq s \leq t \ :: \ \neg q_s \rangle$ and r_{t+1} hold, we have q_{t+1} because the sequence is valid. Hence $k_{t+1} = 0$.
Case 3) $\neg q_{t+1} \ \wedge \ \neg r_{t+1}$: Since $k_t < 0$, from (3), $k_{t+1} < 0$.
Thus, in all cases $k_{t+1} \leq 0$.

It is not hard to show that in an invalid sequence k cannot be assigned values satisfying the given properties.

4 References

1. "Monotonicity, Stability and Constants," J. Misra, *Notes on UNITY: 10-89*, Austin, Texas, December 16, 1989.