Proof of a Real-Time Mutual-Exclusion Algorithm

Notes on UNITY: 32–92

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

9/10/92

Abstract

Michael Fischer[2] has proposed a mutual exclusion algorithm that ingeniously exploits real time. We prove this algorithm using the time-honored technique of establishing an appropriate invariant.

1 Introduction

Michael Fischer[2] has proposed a mutual exclusion algorithm in which real time is used to speed up certain actions and slow down certain other actions. We prove this algorithm using only the fact that time never runs backwards. Other important facts about time—that eventually time increases beyond any bound—are unnecessary for this proof.

The structure of the proof follows the usual pattern of suggesting an invariant, verifying that the suggested invariant is indeed an invariant and showing that the invariant implies mutual exclusion. The invariant is, as usual, a state predicate. We introduce some auxiliary variables that simplify reasoning about time: For a state-predicate p, let $\overline{p}$ denote the last value of time at which p became *true*. We call $\overline{p}$ the *punch* of p (the time at which p last *punched* the clock). We specify the timing constraints of the algorithm succinctly using such variables.

2 Informal Description of the Algorithm

There are N processes, numbered 1 through N, and a global variable x that assumes an integer value between 0 and N. Figure 1 shows the state transitions of process i, $1 \leq i \leq N$.

The initial state of the process is e. The process transits from e to a to wait for entry to its critical section. The edges of the other transitions are labeled with either an assignment—$x := i$ or $x := 0$—or a test—$x = 0?$ or $x = i?$. An assignment on an edge denotes that the state transition is accompanied by an assignment of the corresponding value to x. A test on an edge denotes that the transition takes place only if the test succeeds.

Process state is d when it is in the critical section. Assume that all tests and assignments are atomic. Initially, all processes are in states e and $x = 0$.

*This material is based in part upon work supported by the Texas Advanced Research Program under Grant No. 003658–219, by the Office of Naval Research Contract N00014-90-J-1640 and by the National Science Foundation Award CCR–9111912.

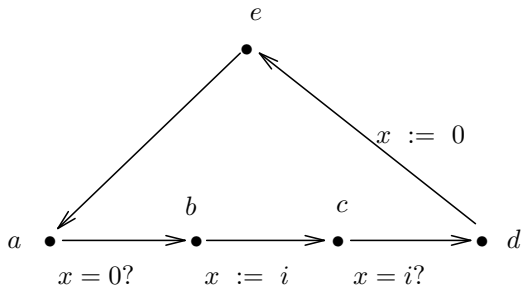

Figure 1: The state transitions of a process i, $1 \leq i \leq N$.

There is no requirement, as yet, that a process transit out of its current state. Thus, a process may stay forever in e (i.e., never attempting to enter its critical section) or in d, thereby preventing all other processes from entering their critical sections forever. It is then easy to construct a scenario where two processes are in their critical sections simultaneously. Timing constraints, given below, guarantee that this possibility is avoided.

(T1) Transition from b to c is completed within a unit of time. Observe that this transition only requires assigning a value to x, and, therefore, the transition is entirely within the control of a process.

(T2) Transition from c to d takes more than one unit of time. This requirement may be implemented by process i waiting for more than a unit of time before testing, $x = i?$. Observe that this transition may never complete.

We will show that mutual exclusion is now guaranteed, i.e., two different processes are never in their d-states simultaneously.

3 Formal Description of the Algorithm

Let s_i denote the state of process i; s_i takes values from $\{a, b, c, d, e\}$. The initial state of the system is

initially $(\forall\ i\ ::\ s_i = e)\ \wedge\ x = 0$

The state transitions of process i are given by

$$
\begin{array}{lll}
\{\alpha_i\}\ s_i\ :=\ a & \text{if} & s_i = e \\
[\!]\ \{\beta_i\}\ s_i\ :=\ b & \text{if} & s_i = a\ \wedge\ x = 0 \\
[\!]\ \{\gamma_i\}\ s_i, x\ :=\ c, i & \text{if} & s_i = b \\
[\!]\ \{\delta_i\}\ s_i\ :=\ d & \text{if} & s_i = c\ \wedge\ x = i \\
[\!]\ \{\epsilon_i\}\ s_i, x\ :=\ e, 0 & \text{if} & s_i = d
\end{array}
$$

There is no fairness requirement on the executions of these statements. Executing a statement in a state where its guard does not hold—such as executing β_i when $x \neq 0$—causes no state change; any execution of a statement when its guard is *true* is called an *effective execution*.

Notation: We will use the following abbreviations

$$a_i\ \equiv\ s_i = a \qquad b_i\ \equiv\ s_i = b \qquad , \ \ldots\ , \qquad e_i\ \equiv\ s_i = e$$

Observe that these predicates are mutually exclusive, i.e.,

$$a_i\ \wedge\ b_i\ \equiv\ \mathit{false},\ \text{etc.},$$

and $a_i\ \vee\ b_i\ \vee\ c_i\ \vee\ d_i\ \vee\ e_i\ \equiv\ \mathit{true}$ □

Formalization of Time

In order to state the timing constraints, we introduce a variable *now*[1]. Informally, the value of *now* at any point during the computation is the current time. The value of *now* is changed by some mechanism outside the given program; the mutual exclusion program can read the current time and assign it to a variable t by executing

$$t \ := \ now$$

The mechanism (or process) that changes *now* could operate synchronously or asynchronously with the actions of the given program. Thus, the value of *now* before and after the execution of

$$t \ := \ now$$

may be different (denoting that execution of this statement consumes some time). The value assigned to t in this case is the value of *now* just before the execution of this statement is started. This interpretation supports the axiom of assignment: Predicate $p(t)$ holds after this assignment if $p(now)$ holds before.

For this paper, we require only that (1) *now* assumes non-negative real values and (2) *now* is monotone nondecreasing. For a formal basis for the introduction of time, including the requirement about the eventual increase of *now*, see [1] and [3].

It is convenient to introduce the following auxiliary variables for study of real time systems. For a state predicate p, let $\overline{p}$ be the value of *now* when p last became *true* (more precisely, $\overline{p}$ is the value of *now* just prior to the execution of the action that last truthified p); initially $\overline{p}$ equals *now* if p holds, else $\overline{p} < 0$. This definition of $\overline{p}$ can be expressed directly as a property of the program (see Misra[3]), or $\overline{p}$ can be defined by augmenting a program text, as shown below. We introduce $\overline{b}_i, \overline{c}_i$ by augmenting β_i, γ_i.

$$\begin{array}{lll} \{\beta_i\}\ s_i, \overline{b}_i \ := \ b, now & \text{if} & s_i = a \ \wedge \ x = 0 \\ \{\gamma_i\}\ s_i, x, \overline{c}_i \ := \ c, i, now & \text{if} & s_i = b \end{array}$$

Initially, $\overline{e}_i = now$ and $\overline{a}_i, \overline{b}_i, \overline{c}_i, \overline{d}_i$ are negative. From the fact that *now* is non-negative and monotone nondecreasing, we can derive, for any p,

(Observation 1) $\overline{p} \leq now$.

Remark: The auxiliary variables $\overline{p}$ can be used to state the most common kinds of real-time constraints:

Once p becomes *true* it remains *true* for at least Δ units,

can be written as

$$\neg p \ \wedge \ \overline{p} \geq 0 \ \Rightarrow \ now > \overline{p} + \Delta$$

and, p is falsified within τ units of being *true*, is expressed by

$$p \ \Rightarrow \ now \leq \overline{p} + \tau$$ □

Timing Constraints

We can now state (T1,T2) formally. For all i, $1 \leq i \leq N$,

$$\begin{array}{lrcl} \text{(T1)} & (c_i \ \vee \ d_i) & \Rightarrow & \overline{c}_i \leq 1 + \overline{b}_i \\ \text{(T2)} & d_i & \Rightarrow & 1 + \overline{c}_i < \overline{d}_i \end{array}$$

The antecedent of (T1), $c_i \ \vee \ d_i$, guarantees that in the current state both $\overline{b}_i$ and $\overline{c}_i$ are defined; similar remarks apply for the antecedent of (T2).

4 Proof of Mutual Exclusion

We establish the following two predicates as invariants. In the following, j, k satisfy $1 \leq j \leq N$ and $1 \leq k \leq N$.

(I1) $(\forall\ j, k \quad :: \quad x = k \ \Rightarrow\ \overline{b}_j \leq \overline{c}_k)$
(I2) $(\forall\ k \quad :: \quad d_k \ \Rightarrow\ x = k)$

Mutual exclusion is immediate from I2:

$$d_i \ \wedge\ d_j \ \Rightarrow\ x = i \ \wedge\ x = j \ \Rightarrow\ i = j$$

Next, we prove that for the program of Section 3 augmented with the timing constraints, the predicates (I1,I2) are invariants.

Note: To be completely formal, we should also show that (I1,I2) cannot be falsified by the process that changes *now*. Since *now* does not appear in either predicate, this demonstration is trivial. □

Proof of the invariance of (I1)

We rewrite (I1) as $(\forall\ j, k \quad :: \quad x \neq k \ \vee\ \overline{b}_j \leq \overline{c}_k)$ to simplify logical manipulations. Initially, $x = 0$. Therefore, initially $x \neq k$, for any k, $1 \leq k \leq N$, and, hence, (I1) holds initially. Next, consider the actions that can falsify the terms in (I1), for arbitrary j, k.

- $x \neq k$ can only be falsified by setting x to k, i.e., by executing γ_k. Effective execution of γ_k assigns

  $$\overline{c}_k \ :=\ now$$

  This action establishes $\overline{b}_j \leq \overline{c}_k$ as a postcondition, because (using the axiom of assignment to replace $\overline{c}_k$ by *now*) $\overline{b}_j \leq now$ is a precondition, from Observation 1. Therefore, γ_k preserves (I1).

- the term $\overline{b}_j \leq \overline{c}_k$ can be affected only by the actions β_j (that may change $\overline{b}_j$) and γ_k (that may change $\overline{c}_k$). We have shown above that γ_k preserves (I1). We show that β_j also preserves (I1). A precondition for the effective execution of β_j is $x = 0$, and β_j preserves $x = 0$. Therefore, $x = 0$, i.e., $x \neq k$ is a postcondition of an effective execution of β_j.

Proof of the invariance of (I2)

Initially $\langle \forall\ k \quad :: \quad e_k \rangle$. Therefore, (I2) holds initially. Next, consider the actions that can falsify $\neg d_k \ \vee\ x = k$, for arbitrary k.

- $\neg d_k$ can be falsified only by setting s_k to d, i.e., by effectively executing δ_k. A precondition for the effective execution of δ_k is $x = k$. The action δ_k does not assign to x, and, hence, preserves $x = k$. Therefore, $\neg d_k \ \vee\ x = k$ holds as a postcondition of δ_k.

  The predicate $x = k$ can be falsified by (1) setting x to 0, i.e., executing ϵ_i, for some i, or (2) setting x to i, $i \neq k$, i.e., executing γ_i, $i \neq k$. We consider these two possibilities, next.

- Executing ϵ_i, for some i: Action ϵ_i has a precondition d_i. From (I2), for $k \neq i$, $\neg d_k$ holds as a precondition; also, $\neg d_k$ is preserved by ϵ_i. Furthermore, $\neg d_i$ is a postcondition of ϵ_i. Therefore, $\neg d_k$, and hence, $\neg d_k \ \vee\ x = k$ holds as a postcondition of ϵ_i, for any i.

- Executing γ_i, $i \neq k$: We show that $\neg d_k$ is a precondition for the execution of γ_i. Since the effective execution of γ_i preserves $\neg d_k$, we have then $\neg d_k \ \vee\ x = k$ as a postcondition.

  We prove that $\neg d_k$ is a precondition by assuming d_k as a precondition and deriving a contradiction.

	d_k	, assumption
	$x = k$	, from (I2) and above
(1)	$\overline{b}_i \leq \overline{c}_k$	, from above and (I1), instantiating j by i
(2)	$1 + \overline{c}_k < \overline{d}_k$	, from (T2) and that d_k holds

Following the execution of γ_i, timing constraint (T1) holds, i.e.,

$$\text{(T1)} \qquad (c_i \ \vee \ d_i) \ \Rightarrow \ \overline{c}_i \leq 1 + \overline{b}_i.$$

The effective execution of γ_i sets c_i to *true* and $\overline{c}_i$ to *now*. Applying the axiom of assignment (to replace c_i by *true* and $\overline{c}_i$ by *now*)

(3) $now \leq 1 + \overline{b}_i$

holds prior to the effective execution of γ_i. We derive a contradiction from (1,2,3) and (Observation 1).

$$1 + \overline{b}_i \ \ \{1\} \leq 1 + \overline{c}_k \ \ \{2\} < \overline{d}_k \ \ \{\text{Observation 1}\} \leq now \ \ \{3\} \leq 1 + \overline{b}_i$$

Acknowledgement: We are grateful to Jacob Kornerup for suggestions about the structure of the proof.

References

[1] Martin Abadi and L. Lamport. An old fashioned recipe for real time. Technical report, DEC Systems Research Center, 1991.

[2] Michael J. Fischer, June 1985. (personal communication with Leslie Lamport).

[3] J. Misra. Safety properties. (manuscript), The University of Texas at Austin, 1992.