

Arbitration Without Common Modifiable Variables

J.L.W. Kessels

Philips Research Laboratories Eindhoven, The Netherlands

Summary. The binary arbitration problem (or, the problem of mutual exclusion between two competitors) is the problem of preventing two competitors from simultaneously possessing the same token. A solution to this problem is presented together with a formal correctness proof. The solution is specific in that it combines the absence of common modifiable variables with the absence of auxiliary activities. Hence, its implementation does not require an arbiter on a lower level or a degree of concurrency of more than two. The solution is generalized for any arbitrary number of competitors by applying the binary solution in a binary arbitration tree.

1. Introduction

The problem of preventing two or more competitors from simultaneously possessing the same token is one of the basic problems in computer science. The problem is often referred to as the arbitration problem or the problem of mutual exclusion, mostly depending on whether the authors has a hardware- or software-oriented background. If a competitor (exclusively) possesses the token, the competitor is said to be in its critical region and the token is said to be assigned to that competitor; otherwise the token is said to be unassigned.

The problem is to design, from a given set of indivisible (atomic) primitives, a prologue and epilogue to a critical region satisfying the following requirements (if a competitor is in its prologue, critical region or epilogue, the token is said to be engaged; otherwise the token is said to be free):

(1) consistency, i.e. at most one competitor may be in its critical region (in between a prologue and epilogue);

(2) progress, i.e. the state in which a token is engaged and unassigned may not exist for an unbounded period of time;

(3) fairness, i.e. a competitor may not be overtaken in a prologue an unbounded number of times.

One class of solutions can be characterized by the following two properties.

1) The atomic primitives are transient (terminating) operations on global variables.

From this it follows that if a competitor has to wait until some system condition holds (e.g. token is unassigned), it has to do so by busy waiting. This property rules out solutions based on P and V operations (on semaphores) as atomic primitives. One could even dispute whether P and V operations may be conceived as atomic operations.

2) The degree of concurrency is equal to the number of competitors. Hence, if none of the competitors is engaged in a prologue or epilogue, no operations concerning the arbitration are being performed. This property rules out solutions in which either an unassigned token is passed on among auxiliary activities (one for each competitor) or the token is a separate activity.

Depending on the grain size of the atomic operations, several solutions in this class can be designed.

The simplest solutions are based on read-modify primitives, i.e. operations that inspect and modify a variable in one indivisible action. The test-and-set-bit primitive is a well-known example of this sort.

More intricate solutions are based on separate read and modify primitives allowing two or more competitors to modify the same variable. One of the first solutions of this sort was designed by Dekker [1]. Recently, a very elegant solution was presented in [3]; a formal proof of this solution was given in [2].

In both cases, first the binary arbitration problem (i.e. arbitration between two competitors) is solved and then the generalization for N competitors is achieved by applying the binary solution $N - 1$ times.

Note that the implementation of the above mentioned primitives requires arbitration on a lower level. This paper presents a solution of the binary arbitration problem in which each variable is only modified by one competitor. Hence, its implementation does not require an arbiter on a lower level.

The solution is derived from the solution in [3] by distributing the common modifiable variable between the two competitors. In Sect. 3 this solution is generalized for N competitors by applying the binary solution in a binary arbitration tree (requiring $O(\log N)$ binary arbitrations).

2. Binary Arbitration

In this section the following three descriptions are given of the solution presented:

- a neutral description taking the form of a prologue and epilogue procedure (the generalization in Sect. 3 is constructed from these procedures);
- a description in terms of atomic operations allowing the verification of consistency; and
- a description in terms of (larger) transient operations facilitating the verification of the progress and fairness.

Atomic (or indivisible) operations are defined as operations whose execution is not interfered with by other concurrent activities.

Transient operations are defined as (possibly compound) operations whose execution terminates within a bounded period of time.

Consider two competitors, identified by the binary numbers 0 and 1, each provided with two exterior variables: a boolean variable Q and a binary variable R . An exterior variable is defined as a variable that is private with respect to modification (only one activity may modify the variable) and common with respect to inspection (other activities may inspect its value). The four variables taken together are conceived as one compound variable of the type token.

```
type token = record  $Q$ : array [binary] of boolean;  
                $R$ : array [binary] of binary  
           end.
```

Both elements in Q are initially false (token is free), while the initial values of R are immaterial.

The prologue and epilogue procedure both have the following two parameters:

- c , i.e. the identification of the competitor; and
- t , i.e. the token for which the arbitration has to take place.

We define a unary operator *inv* that maps one binary number (competitor identification) onto the other:

$$\text{inv } c: (c + 1) \bmod 2;$$

and a binary operator *plus* for addition in $GF(2)$:

$$a \text{ plus } b: (a + b) \bmod 2$$

```
procedure binary-arbitration-prologue ( $c$ : binary; var  $t$ : token);  
begin with  $t$   
    do begin  $Q[c] := \text{true}$ ;  
             $R[c] := R[\text{inv } c] \text{ plus } c$ ;  
            repeat until not  $Q[\text{inv } c]$  or  $(R[c] \neq (R[\text{inv } c] \text{ plus } c))$   
    end  
end;  
  
procedure binary-arbitration-epilogue ( $c$ : binary; var  $t$ : token);  
begin with  $t$   
    do  $Q[c] := \text{false}$   
end
```

Note that in both procedures, competitor c only modifies array elements with index c .

Verification of the consistency aspect requires a description of the program in terms of atomic operations. For this purpose, each competitor is provided with an additional binary (private) variable T . Moreover, in order to facilitate the correctness proof, each atomic operation is extended by an auxiliary assignment to an integer ghost variable L . In the following description, the prologue and epilogue are not given as separate procedures, but as embedding program parts of a critical region of competitor c . Atomic operations are enclosed in angular brackets, they are labeled from A1 through A6.

```

<A1:  $L[c] := 1$ ;  $Q[c] := \text{true}$ >;
<A2:  $L[c] := 2$ ;  $T[c] := R[\text{inv } c]$ >;
<A3:  $L[c] := 3$ ;  $R[c] := T[c] \text{ plus } c$ >;
  while  $L[c] < 4$ 
  do begin <A4: if not  $Q[\text{inv } c]$  then  $L[c] := 4$ >;
          <A5: if  $T[c] \neq R[\text{inv } c]$  then  $L[c] := 4$ >
  end;
  Critical Region;
<A6:  $L[c] := 0$ ;  $Q[c] := \text{false}$ >

```

The elements in Q , L and T are initialized so as to satisfy the following predicate (the initial values of R are immaterial):

$$(\mathbf{A } c: L[c] = 0 \text{ and not } Q[c] \text{ and } T[c] = R[\text{inv } c]).$$

We consider the following predicates:

$$(\mathbf{A } c: L[c] > 0 \Leftrightarrow Q[c]) \quad (\text{P1})$$

$$(\mathbf{N } c: T[c] \neq R[\text{inv } c]) + (\mathbf{N } c: R[c] \neq (T[c] \text{ plus } c)) = 1 \quad (\text{P2})$$

where “ $\mathbf{N } c: P(c)$ ” stands for “the number of
values of c for which $P(c)$ holds”;

$$(\mathbf{E } c: L[c] < 2) \text{ or } (\mathbf{E } c: L[c] < 4 \text{ and } T[c] = R[\text{inv } c]) \quad (\text{P3})$$

and observe that these predicates hold initially.

P1 is a system invariant

Proof. A1 and A6 are the only statements that are relevant to P1; both statements establish the truth of one factor and preserve the truth of the other.

P2 is a system invariant.

Proof. A2 and A3 are the only statements that are relevant to P2. A2 either leaves both terms invariant or it decreases the first term by one. In the latter case, the second term was zero and hence is increased by one. A3 either leaves both terms invariant or it decreases the second term by one. In the latter case the first term was zero and hence is increased by one.

P1 and P2 and P3 is a system invariant.

Proof. A2, A3, A4, A5 and A6 are the only statements that are relevant to P3.

- A2 establishes the truth of the second term.
- If A3 modifies $R[c]$, the first term in P2 was zero (since the second term was one); from this it follows that the second term in P3 invariantly holds.
- From P1 it follows that the condition in A4 implies the first term in P3; the truth of this term is preserved.
- A5 can only invalidate a first factor in the second term; from the condition it follows that it will only do so if the corresponding second factor was already false; hence, the term is left invariant.
- A6 establishes the truth of the first term.

P3 implies

$$(\mathbf{E} c: L[c] < 4)$$

hence

$$(\mathbf{N} c: L[c] = 4) \leq 1.$$

This is a formal specification of the consistency requirement (mutual exclusion).

Note that the initial values of the elements of T are immaterial, since each element is modified before its value is inspected.

When progress and fairness are to be proved, it is more convenient to conceive (the largest possible) transient operations as the building bricks of the program. The following program of competitor c is a description in these terms.

Transient operations are enclosed in braces.

```
{T1:  $Q[c] := \text{true}; R[c] := R[\text{inv } c]$  plus  $c$ };
    repeat until  $B(c)$ ;
    Critical Region;
{T2:  $Q[c] := \text{false}$ }
```

where $B(c)$ is the progress condition of competitor c

$$B(c): \text{not } Q[\text{inv } c] \text{ or } (R[c] \neq (R[\text{inv } c] \text{ plus } c)).$$

We define a persistent predicate as a predicate whose value can only be modified from false to true. For the program of one competitor the progress condition of the other is a persistent predicate, since T1 establishes the truth of the second term and T2 establishes the truth of the first term.

Hence, the progress condition of a particular competitor can only be invalidated by that competitor (when executing T1).

Both programs leave the following predicate invariant

$$(\mathbf{A} c: \text{not } Q[c]) \text{ or } (\mathbf{E} c: Q[c] \text{ and } B(c)).$$

Proof. T1 either establishes or preserves the truth of the second term, and T2 either establishes the first term or it preserves the second term.

This invariant is a formal specification of the progress requirement; it states: either the token is free or there is a competitor engaged in its prologue, critical region or epilogue for which the progress condition holds. From this it follows that the state in which the token is engaged and unassigned is a transient state, since it will become assigned or free within a finite time.

From the fact that the progress condition of one competitor is a persistent predicate of the other, it follows that infinite overtaking in the prologue is impossible.

Overtaking a competitor in its prologue would imply reclaiming the right to progress from a busy-waiting activity. In my opinion, the (safe) construction of such a capability requires read-modify primitives. Hence, for binary arbitration based on separate read and modify primitives, fairness is implied by the consistency and progress requirements.

Note that while a competitor is waiting for its progress condition the other competitor will modify each of its exterior variables once at most.

3. General Arbitration

Any solution to the binary arbitration problem can easily be generalized by applying this solution in a nested way (nested critical regions). Consider a binary tree, the leaves represent the competitors and all other nodes represent tokens to which binary arbitration may be applied.

To gain the exclusive privilege over all other competitors, a competitor has to traverse the path from its leaf up to an including the root; each edge on this path represents a binary arbitration prologue. Relinquishing the privilege implies a backward traversal of the path; each edge then represents a binary arbitration epilogue. The fact that the nodes are relinquished in the reverse order, agrees with the total arbitration being conceived as nested binary arbitrations; it is even essential since it guarantees that at most two competitors are engaged with each node.

The following numbering scheme for the nodes is convenient if the binary tree is accommodated in an array (this schema is also often used in heap-sort implementations).

```
number (node): if node=root
                then 1
                else if node=leftson (father (node))
                     then  $2 \times \text{number (father (node))}$ 
                     else  $2 \times \text{number (father (node))} + 1$ .
```

This numbering scheme allows a simple derivation of the number of the father from the number of any son node.

Let us take N competitors, where N is a power of two, identified by the numbers $0 \dots N-1$.

In the balanced arbitration tree, the leaf number of competitor c is $N+c$. In order to simplify the reversed traversal in the epilogue, each non-leaf node in the arbitration tree is provided with a binary variable which, if the corresponding token is assigned, will contain the label of the edge, traversed by the competitor to which it is assigned.

We declare the following arrays:

```
t: array [1...N-1] of token;
e: array [1...N-1] of binary.
```

Initially, all tokens are free. The prologue and epilogue of the general solution are given below.

```
procedure general-arbitration-prologue (c: 0...N-1);
var n: 1...2*N-1;
begin n:=N+c;
    while n>1
    do begin binary-arbitration-prologue (n mod 2, t[n div 2]);
        e[n div 2]:=n mod 2;
        n:=n div 2
    end
end;
```

```

procedure general-arbitration-epilogue;
var  $n$ :  $1 \dots 2 * N - 1$ ;
begin  $n := 1$ ;
    while  $n < N$ 
    do begin  $n := 2 * n + e[n]$ ;
        binary-arbitration-epilogue ( $n \bmod 2, t[n \div 2]$ )
    end
end

```

The while-loops in both procedures maintain the following loop-invariant: all tokens on the path from leaf $N + c$ up to and including node n are assigned to competitor c and no other tokens are assigned to that competitor (one may conceive the leaves as also containing a token that is always assigned to the corresponding competitor).

Note that only those variables constituting the tokens one level higher than the leaves may be conceived as exterior variables in the strict sense. All other variables may be modified by more than one competitor, though by one at most at any given time. One could conceive each node as one activity with (possibly) concurrency on lower abstraction levels.

Acknowledgement. Acknowledgement is due to C.S. Scholten for his simplification of the correctness proof.

References

1. Dijkstra, E.W.: Co-operating sequential processes. In: Programming Languages pp. 43–112. (F. Genuysed) Academic Press, New York 1968
2. Dijkstra, E.W.: An assertional proof of a program. G.L. Peterson, EWD 779 (1981)
3. Peterson, G.L.: Myths about the mutual exclusion problem. Information Processing Letters 12, 4, 115–116 (1981)

Received January 27, 1982