The Bakery Algorithm: Yet Another Specification and Verification

EGON BÖRGER*   YURI GUREVICH†

DEAN ROSENZWEIG‡

Abstract

In a meeting at Schloss Dagstuhl in June 1993, Uri Abraham and Menachem Magidor have challenged the thesis that an evolving algebra can be tailored to any algorithm at its own abstraction level. As example they gave an instructive proof which uses lower and higher views to show correctness of Lamport's bakery algorithm. We construct two evolving algebras capturing lower and higher view respectively, enabling a simple and concise proof of correctness for the bakery algorithm.§

Introduction

Uri Abraham [Abraham93] has devised an instructive correctness proof for various variants of Lamport's bakery algorithm relying on a distinction between a lower view and a higher view of the algorithms. Actions at the higher level represents complex lower level computations. He formulates abstract conditions on higher level actions which are then shown to suffice for correctness and fairness (in form of a 'first-come-first-served' property and deadlock–freedom) and to be satisfied by the corresponding lower level computations.

At a seminar in Schloss Dagstuhl in June 1993 Uri Abraham and Menachem Magidor have expressed doubts that such a proof could be naturally carried out in the evolving algebra framework of [Gurevich91], since the latter uses a notion of atomic instantaneous action.

We construct, in Section 1, two evolving algebras, reflecting the lower and higher views of Lamport's improved version of the bakery algorithm (see [Lamport79]).

---

*Dipartimento di Informatica, Universita di Pisa, Corso Italia 40, I–56100 Pisa, boerger@di.unipi.it. Partially supported by MURST 91.

†EECS, University of Michigan, Ann Arbor MI 48109–2122, gurevich@umich.edu. Partially supported by NSF Grant CCR 92-04742 and ONR grant N00014-91-J-11861.

‡FSB, University of Zagreb, Salajeva 5, HR–41000 Zagreb, dean@math.hr. Partially supported by CNR/Gnasaga grant 2.94.

‡To appear in: E. Börger (Ed.), *Specification and Validation Methods*, Oxford University Press, 1995.

In Section 2 we display abstract conditions on higher level actions, in terms of atomic–action semantics, enabling a simple and concise proof of the first–come–first–served property (FCFS) and deadlock–freedom. The conditions are easily seen to be satisfied by corresponding lower level computations. Since actions of an evolving algebra are assumed there to be atomic, that proof treats the case of atomic reads and writes to shared registers.

In Section 3 we explain the semantics of evolving algebras assuming *durative actions*, actions taking time, and allowing overlapping of reads and writes to shared registers. Refining the abstract conditions for the case of *regular* reads (see [Lamport86]), we show that the proof of the previous section goes through with only slight modifications. For the more general case of *safe* registers correctness of the algorithm from [Lamport74] is then easily proved by a slight adaptation of the present argument—the improved algorithm from [Lamport79] is not correct for safe registers, as shown by a simple counterexample.

Thus the two interpretations of evolving algebra dynamics reflect two disciplines for accessing shared registers—by atomic and non–overlapping reads and writes, or by durative and possibly overlapping ones. What really changes is the notion of state: for atomic actions we have global states, whereas for durative actions we have instead local states of agents (see the concept of external and internal locations in section 3.1). The correctness proof however remains essentially the same.

In order to make the paper technically self contained, except for basic notions about evolving algebras of [Gurevich91], we start Section 1 with a review of Lamport's 1979 algorithm and give full details of proofs also in those places where we borrow from [Abraham93].

1 The algorithms

This section presents Lamport's algorithm (taken in a form which is adapted from [Abraham93]), the corresponding 'lower level' evolving algebra, and the more abstract evolving algebra reflecting the 'higher level view'.

1.1 Lamport's algorithm

For arbitrary but fixed N let $P_1, \ldots, P_N$ be processes that may want from time to time to access a 'critical section' CS of code. Any mutual exclusion protocol—which each P_i is supposed to execute in order to enter the critical section—has to prevent two processes from being in the critical section simultaneously. The Bakery Algorithm provides each P_i with a (shared) register R_i and a (private) array $n[1], \ldots, n[N]$ holding natural numbers. Only P_i is allowed to write to R_i but every process can read the register. We assume each register to be initialized with value 0.

The Bakery Algorithm is divided into six consecutive phases: *start*, *doorway*, *ticket* assignment, *wait* section, *critical section* and *finale*. A process P_i starts by declaring its interest in accessing the critical section through writing 1 into its register recording the value written also in its corresponding array variable. In the doorway section, P_i copies all the other registers into its array. It then writes a *ticket*, greater than each number in its array, into its register and into $n[i]$. During the subsequent wait section, process P_i keeps reading, into its array, the registers of each other process P_j, until the resulting array value $n[j] = 0$ or $n[j] > n[i]$ or $n[j] = n[i] \wedge j > i$. Then P_i enters the critical section. Upon leaving CS, as finale, P_i sets its register to 0.

Start

```
n[i] := 1
write(R_i,n[i])
```

Doorway

```
for all j≠i, read(R_j,n[j])
```

Ticket

```
n[i] := 1 + max_j n[j]
write(R_i,n[i])
```

Wait

```
for all j≠i, repeat
   read(R_j,n[j]) until
   n[j]=0 or n[j]>n[i] or (n[j]=n[i] and j>i)
```

Critical Section

Finale

```
R_i := 0
```

Note that by ordering pairs of positive integers lexicographically:

$$(i, j) < (k, l) \longleftrightarrow [i < k \text{ or } (i = k \text{ and } j < l)]$$

one can write the until condition as follows: n[j]=0 or (n[j],j)>(n[i],i). The condition assures that, in case two processes get the same 'ticket', the one with smaller identifier gets the priority.

Note also that the for-all commands in the doorway and the wait section may be executed in many ways, in various sequences, all at once, concurrently etc.

1.2 The lower level algebra $\mathcal{B}_1$

As the basis for the subsequent analysis and 'higher level' abstraction, we reformulate here the above Bakery Algorithm as an evolving algebra $\mathcal{B}_1$. It contains, for each process, a *customer–agent*. The customers execute the module with rules Start, Ticket, Entry, Exit, Finale—corresponding to the homonymous Bakery Algorithm phases. In order to preserve the freedom of choosing an ordering of reads, in Doorway and Wait, $\mathcal{B}_1$ contains also *reader–agents* $r(X,Y)$, where X,Y are customers. Each reader–agent $r(X,Y)$ reads, during the doorway and the wait section of X, the register $R(Y)$ of process Y into X's array component $A(X,Y)$, doing the work of the *Doorway* and *Wait* phases. The module of a reader agent has two rules, Read and Check.

Each customer X can execute the rules Start, Ticket, Entry, Exit, Finale only sequentially, in that order; this is assured by the function *mode* which for each X assumes cyclically the values *satisfied, doorway, wait, CS, done, satisfied.* The *mode* function also assures that Ticket and Entry can be executed by X only after all readers have executed their Read and Check rules respectively. Thus the following rules faithfully represent the corresponding phases of the Bakery Algorithm (given that initially all registers $R(X)$ have value 0 and all customers are satisfied).

Customer X

Start

```
if mode(X)=satisfied then
   A(X,X) := 1, R(X) := 1, mode(X) := doorway
```

Ticket

```
if mode(X)=doorway and (∀Y ≠ X) mode(r(X,Y))=wait then
   A(X,X) := 1+max_Y A(X,Y), R(X) := 1+max_Y A(X,Y)
   mode(X) := wait
```

Entry

```
if mode(X) = wait and (∀Y ≠ X) mode(r(X,Y)) = doorway then
   mode(X) := CS
```

Exit

```
if mode(X) = CS then
   mode(X) := done
```

Finale

```
if mode(X) = done then
   R(X) := 0, mode(X) := satisfied
```

Reader $r(X, Y)$

Read

```
if mode(r(X,Y))) = mode(X) then
   A(X,Y) := R(Y)
   if mode(r(X,Y))=doorway then mode(r(X,Y)) := wait
   if mode(r(X,Y))=wait then mode(r(X,Y)) := check
```

Check

```
if mode(r(X,Y)) = check then
   if A(X,Y)=0 or (A(X,Y),id(Y)) > (A(X,X),id(X)) then
      mode(r(X,Y)) := doorway
   else mode(r(X,Y)) := wait
```

The modules of rules are written as templates, i.e. there is a module for each customer X and a module for each reader $r(X, Y)$.

1.3 The higher level algebra $\mathcal{B}_2$

In this subsection we define an evolving algebra expressing a 'higher level' view of the Bakery Algorithm. The relevant datum to be described abstractly is the *ticket* assigned to a customer X (and written into its register $R(X)$) when X leaves the doorway and enters the wait section. We introduce for this purpose an external function T whose values are determined dynamically by the outside world, cf. [Gurevich91].

The relevant moment to be analyzed is the moment at which a process which has received a ticket is allowed to enter the critical section. This 'permission to go' will also be represented by an external function, *Go*.

In subsequent sections we will impose conditions upon T and *Go* which will be shown to guarantee the correctness of the higher level Bakery Algebra.

The higher level algebra has only one module, parametric in a customer X, which has five rules. Again, we assume that initially all registers have value 0 and all customers are satisfied.

Start

```
if mode(X) = satisfied then
   R(X) := 1, mode(X) := doorway
```

Ticket

```
if mode(X) = doorway then
   R(X) := T(X), mode(X) := wait
```

Entry

```
if mode(X) = wait and Go(X) then
   mode(X) := CS
```

Exit

```
if mode(X) = CS then
   mode(X) := done
```

Finale

```
if mode(X) = done then
   mode(X) := satisfied, R(X) := 0
```

2 Atomic actions interpretation

2.1 Semantics of $\mathcal{B}_1$

We rely on the notion of run of [Gurevich94], specialized to *real time*.

This means that we shall speak of a move (rule execution) taking place at moment a. Since we consider atomic actions here, we assume moves to take zero time. Each move is performed by an agent (a customer or a reader) and, since agents are sequential, two moves by the same agent cannot take place at the same moment. For any moment a the set of all moves taking place not later than a is finite (let us call this property 'cofiniteness'). The state (static algebra) $\mathcal{S}_b$ at time b is the one resulting from all moves taking place before b. We shall denote the value a term t takes (in the state) at time b by t_b.

If a move is executed at time b, $\mathcal{S}_b$ is the state in which the move is executed; for some sufficiently small ϵ, $\mathcal{S}_{b+\epsilon}$ is the state resulting from the move. We do not allow (in this section) to read from and write to the same location at the same time. We assume that no module stalls forever; eventually it makes a move (provided a move is enabled all the time). There is one exception: customers are allowed to remain in mode *satisfied* forever.

We now define intervals of (real) time characterized by the moments of successive executions, by a process X, of its rules Start, Ticket, Entry, Exit.

Definition 2.1. Suppose X executes Start and Ticket rules at moments a and b and does not execute anything in between. Then the open interval $x = (a, b)$ is a *doorway* of X. If b is the last execution of X then the *wait interval* $W(x) = (b, \infty)$ and the *CS interval* $CS(x)$ is undefined. Suppose that the execution of Ticket rule at b is followed by executions of Entry rule at c and Exit rule at d. Then $W(x) = (b, c)$ and $CS(x) = (c, d)$).

By the assumption that no module stalls forever, every doorway is finite. This is in accordance with the fact that in the low–level Bakery Algebra, $T(x)$ is always defined when interpreted as $1 + \max_Y A(X, Y)$.

2.2 Semantics of $\mathcal{B}_2$

The semantics of $\mathcal{B}_2$ is similar to that of $\mathcal{B}_1$. There are no readers around. The definition of doorways and related periods applies also to $\mathcal{B}_2$.

Contrary to $\mathcal{B}_1$, $\mathcal{B}_2$ has external functions, namely T and Go. We are going to impose some constraints on them. To avoid repetitive case distinctions for processes which (being satisfied) have register 0, and of processes which happen to receive the same ticket, we introduce the following notation. If f is a function ¿from the original processes to natural numbers, let

$$f'(X) = \begin{cases} N \cdot f(X) + \mathrm{id}(X), & \text{if } f(X) > 0; \\ \infty, & \text{otherwise.} \end{cases}$$

We assume that the identifiers of the N processes are natural numbers $< N$.

For real intervals I, J we define $I < J$ to mean that $a < b$ for all $a \in I$, $b \in J$. This ordering will help us to formalize the idea that tickets increase together with doorways (see $C1$ below). This should also apply in a way to overlapping doorways; these are ordered by the following relation $\prec$, borrowed from [Abraham93].

Let $X \neq Y$, x ranges over doorways of X, y ranges over doorways of Y.

Definition 2.2. *$x \lhd y$ if $x \cap y \neq \emptyset$ and $T'(x) < T'(y)$. Further, $x \prec y$ if $x \lhd y$ or $x < y$.*

Lemma 2.3. *$x \prec y$ or $y \prec x$.*

Proof Note that $T'(y) \neq T'(x)$ for $X \neq Y$.

Constraints on T and Go

C0 $T(x)$ is a positive integer > 1.

C1 If $y < x$ then either $\mathrm{CS}(y) < \sup(x)$ or $T'(y) < T'(x)$.

C2 If $\mathrm{Go}(X)$ holds at moment $t > \sup(x)$ then, for every $Y \neq X$, there exists a moment $b \in W(x)$ such that $T'(x) < R'_b(Y)$.

C3 If $W(y)$ is finite for all $y \prec x$, then $W(x)$ is finite.

Intuitively, C1 says that tickets respect the temporal precedence of doorways with overlapping wait periods, C3 is an induction principle, and C2 expresses that permission to go is obtained by checking the ticket against competitors' registers.

2.3 $\mathcal{B}_1$ implements $\mathcal{B}_2$ correctly

We check that the constraints are satisfied in the first algebra, where $T(X) = 1 + \max_Y A(X,Y)$, and $Go_t(X)$ means that the condition of the rule Entry is satisfied at moment t.

C0 is satisfied since the maximum in the rule *Ticket* is taken over each Y, including X which at that moment has register value $R(X) = 1$.

C1. Let t be the time of the Read move by $r(X,Y)$ during x. If there exists a Finale move by Y during $(\sup(y), t)$, then $\mathrm{CS}(y) < \sup(x)$. Otherwise $R_t(Y) = T(y)$ and therefore $T(x) \geq 1 + R_t(Y) > T(y) > 0$. Hence $T'(x) > T'(y)$.

C2. $Go(X)$ becomes true in $\mathcal{B}_1$ when all readers $r(X, Y)$ finish their wait-section readings. Fix a $Y \neq X$ and consider the last Read move by $r(X, Y)$ during $W(x)$. In view of the corresponding Check move, the time of that Read move is the desired b.

C3. By contradiction, suppose that the premise is satisfied but the conclusion is false, i.e. $W(x)$ is infinite.

Claim: There is a moment $b \in W(x)$ so late that the following two properties hold for each y:

(i) if $y \prec x$ then $b > sup(CS(y))$, (ii) if $x \prec y$ then $b > sup(y)$.

Given the claim, it suffices to prove that any $r(X, Y)$ finishes its reading during $W(x)$ (in contradiction to the assumption that $W(x)$ is infinite). If $r(X, Y)$ finishes its reading before b, we are done since $b \in W(x)$. Otherwise, by definition of b, no $Y \neq X$ can be in mode doorway at or after b. Thus, at or after b, $r(X, Y)$ can read either 0 or $T(y)$ for some $y \succ x$. In the first case the next Check of $r(X, Y)$ will succeed; in the second case it will also succeed, since $T'(y) > T'(x)$ (by C1 if $x < y$, and by definition of $\lhd$ if $x \lhd y$). Thus, the very first reading at or after b will be the last reading of $r(X, Y)$.

To prove the claim, note that, by the cofiniteness condition of runs, there are only finitely many doorways y coming earlier than or overlapping with x. Note that, for $y \prec x$, $sup(CS(y)) < \infty$ by the assumption that $W(y)$ is finite and that no module stalls forever. It suffices to prove that, for each Y, there is at most one $y > x$. Suppose $x < y$. Then, by C1, $T'(x) < T'(y)$ (since $W(x)$ is infinite), and Y remains waiting forever, i.e. $r(Y, X)$ keeps forever executing waiting section Reads.

2.4 Correctness and fairness of $\mathcal{B}_2$

Lemma 2.4. (FCFS) *If $y \prec x$ and $W(x)$ is finite, then $W(y)$ is finite and $CS(y) < CS(x)$.*

Proof Assume the premise is satisfied and the conclusion is false. Take b as given by C2.

Claim 1 : $T'(y) < T'(x)$.

Claim 2 : $\sup(y) < b$.

Given the claims, we have $T'(y) < T'(x) < R'_b(Y)$ and thus Y must be writing to $R(Y)$ sometime in $(\sup(y), b)$. But the first such write after $\sup(y)$ must be a Finale move, which contradicts the assumption that the conclusion of the lemma is false.

Claim 1 follows immediately from definition of $\prec$ in case of overlap, and from C1 otherwise.

To prove Claim 2, we first note that $b > \inf(y)$, in view of $y \prec x$. It is impossible that $\inf(y) < b \leq \sup(y)$, since then $R_b(Y) = 1$.

Lemma 2.5. *$\prec$ is transitive.*

Proof by contradiction. Suppose $x \prec y \prec z \prec x$. Count the number n of $<$'s in the above sequence of $\prec$ signs. In case $n = 0$ the statement follows ¿from the fact that the order of integers (tickets) is transitive, and in cases $n = 2, 3$ the statement follows from the fact that the order $<$ of real intervals is transitive. In case $n = 1$, without loss of generality, we have $x \lhd y \lhd z < x$ and therefore $T'(x) < T'(y) < T'(z)$. By Lemma 2.4, the assumption $x \prec y \prec z \prec x$ implies that $W(x), W(y), W(z)$ are all infinite. Thus we can apply C1 to obtain also $T'(z) < T'(x)$, which is impossible.

Lemma 2.6. (Deadlock freedom) *Every $W(x)$ is finite.*

Proof By cofiniteness condition on runs, $\prec$ is well-founded. Then C3 is precisely the induction principle required to establish the claim.

This section is summarized in the following

Theorem 2.7. *Doorways are linearly ordered by $\prec$. All waiting sections are finite, and $x \prec y$ implies $CS(x) < CS(y)$.*

3 Durative actions interpretation

3.1 Semantics of $\mathcal{B}_1$

Let S be an initial state where all customers are in mode *satisfied*, all readers are in mode *doorway*, and all registers $R(X)$ have value 0—the values of A don't matter. We consider runs from S.

A run of $\mathcal{B}_1$ consists of the following:

- A collection M of elements, called *moves*.
- A function $\mathcal{A}$ from M to the set of agents. $\mathcal{A}(\mu)$ is the agent that makes the move μ.
- A function P that associates a nonempty finite open time interval with each move. $P(\mu)$ is the execution period of μ. No move can last forever.

However, not every triple $(M, \mathcal{A}, P)$ is a run. The following conditions 1–6 should be satisfied. The first condition reflects the fact that our agents are sequential:

1 For each agent X, $\{P(\mu) : \mathcal{A}(\mu) = X\}$ is linearly ordered by $<$. Moreover, this ordered set is isomorphic to an initial segment of positive integers, and if it is infinite then $\sup_\mu P(\mu) = \infty$.

We say that an agent Z is passive at moment t (resp. in interval I) if t does not belong to (resp. I does not intersect) the period $P(\mu)$ of any move of Z. We would like to insure that X has a well defined state $\mathcal{S}_t(X)$ at every passive moment t of X.

2 If $[a, b]$ is a passive interval of an agent X then $\mathcal{S}_b(X) = \mathcal{S}_a(X)$.

To insure that condition 2 is satisfied, we stipulate the following.

A customer X. Locations of dynamic functions internal to X: mode(X), $A(X, X)$ and $R(X)$. Locations of dynamic functions external to X: mode($r(X, Y)$) and $A(X, Y)$ where $Y \neq X$.

A reader $r(X, Y)$. Internal locations: mode($r(X, Y)$) and $A(X, Y)$. External locations: mode(X), $A(X, X)$ and $R(Y)$.

States of an agent reflect only the values of internal locations. Notice that every location of any function is internal to some agent.

Call a move μ of an agent X *atomic* with respect to an external location ℓ if ℓ is not updated during $P(\mu)$. A move μ is *atomic* if it is atomic with respect to all its external locations. An agent is *atomic* if all its moves are so.

3 If an agent X makes an atomic move μ and $P(\mu) = (a, b)$ then $\mathcal{S}_b(X)$ is the result of executing one step of X at $\mathcal{S}_a(X)$. (See [Gurevich94] for the definition of the result of a one-step execution of a sequential evolving algebra at a given state.)

4 All customers are atomic. All Check moves of readers are atomic. All Read moves of any $r(X, Y)$ are atomic with respect to mode(X).

Read moves of a reader $R(X, Y)$ may be non atomic with respect to $R(Y)$. We adopt Lamport's notion of regular reads (with a different but equivalent definition):

5 Suppose that (a, b) is the period of a Read move μ by a reader $Q = r(X, Y)$. The value of $A(X, Y)$ in state $\mathcal{S}_b(Q)$, at passive moment b of Q, is nondeterministically chosen among the values of $R(Y)$ at moments t satisfying at least one of the following conditions:

- t is Y's last passive moment $\leq a$,
- t is one of Y's passive moments in (a, b),
- t is Y's first passive moment $\geq b$.

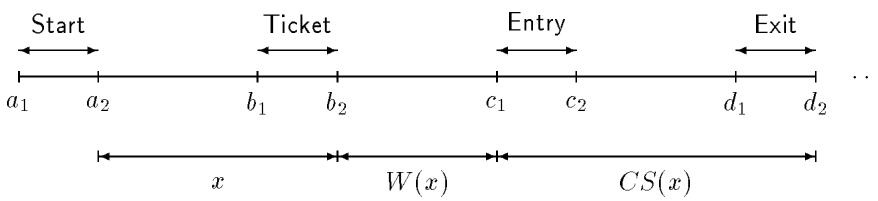
Let $\xi(\mu)$ be the chosen moment t.

6 If an agent Z has an infinite passive interval during which it is enabled in its final state then Z is an original agent and its mode is *satisfied*.

In other words, we assume again that no agent stalls forever except if it is an original agent in mode satisfied. We will use the following refined definition of doorway, wait and CS sections.

Definition 3.1. • *Suppose X executes* Start *during (a_1, a_2) and then executes* Ticket *during (b_1, b_2), so that the interval $[a_2, b_1]$ is passive for X. Then $x = (a_2, b_2)$ is a doorway of X.*

- *Suppose that X executes* Ticket *during (b_1, b_2). If the execution of* Ticket *is not followed by an execution of* Entry *then the wait period $W(x)$ is (b_2, ∞). Suppose that the execution of* Ticket *is followed by an execution of* Entry *during some period (c_1, c_2), so that the interval $[b_2, c_1]$ is passive for X. Then $W(x) = (b_2, c_1)$.*
- *Suppose that X executes* Entry *during (c_1, c_2) and then executes* Exit *during (d_1, d_2), so that the interval $[c_2, d_1]$ is passive for X. Then the critical section period $CS(x)$ is (c_1, d_2).*



3.2 Semantics of $\mathcal{B}_2$

The semantics of $\mathcal{B}_2$ is similar to that of $\mathcal{B}_1$. The constraints C0, C1 and C3 of the previous section remain the same, while C2 is refined for regular registers as follows.

C2 If Go(X) holds at moment $t > \sup(x)$ then, for every $Y \neq X$, there exists a passive moments b for Y such that $T'(x) < R'_b(Y)$ and one of the following holds:
either $b \in W(x)$;
or b is the last passive moment of Y which is $\leq \inf(W(x))$;
or b is the first passive moment of Y which is $\geq \sup(W(x))$.

3.3 $\mathcal{B}_1$ implements $\mathcal{B}_2$ correctly

The proofs that C0 and C3 hold of $\mathcal{B}_1$ remain the same; the proofs for C1 and C2 are modified as follows.

C1. Let μ be the Read move by $r(X, Y)$ during x and $t = \xi(\mu)$. If there exists a Finale move ν by Y such that $P(\nu)$ intersects $P(\mu)$, then CS(y) $<$ sup(x). Otherwise $R_t(Y) = T(y)$ and therefore $T(x) \geq 1 + R_t(Y) > T(y) > 0$. Hence $T'(x) > T'(y)$.

C2. $Go(X)$ becomes true in $\mathcal{B}_1$ when all readers $r(X, Y)$ finish their wait-section readings. Fix a $Y \neq X$ and consider the last Read move ν by $r(X, Y)$ during $W(x)$. The desired b is $\xi(\nu)$.

3.4 Correctness and fairness of $\mathcal{B}_2$

All proofs of the previous section remain, except for the proof of Lemma 2.4, which is modified as follows.

Proof Assume premise is satisfied and conclusion is false. Take b as given by C2.

Claim 1 : $T'(y) < T'(x)$.

Claim 2 : $\sup(y) < b$.

Given the claims, we have $R'_{\sup(y)}(Y) = T'(y) < T'(x) < R'_b(Y)$ and thus Y must be writing to $R(Y)$ somewhere in $(\sup(y), b)$ so that this write starts before $\sup(W(x))$. But the first such write after $\sup(y)$ must be a Finale move, which contradicts the assumption that the conclusion of the lemma is false.

Claim 1 follows immediately from definition of $\prec$ in case of overlap, and from C1 otherwise.

To prove Claim 2, we first establish that $b \geq \inf(y)$. Since $\inf(y)$ is a passive moment of Y such that $\inf(y) < \sup(x) = \inf(W(x))$ (in view of $y \prec x$), so $b < \inf(y)$ could not be the last passive moment of Y which is $\leq inf(W(x))$. Neither can we have $\inf(y) \leq b < \sup(y)$, since then $R_b(Y) = 1$. Finally $b \neq \sup(y)$, since otherwise we would have $R_b(Y) = T(y)$, contradicting Claim 1.

3.5 Counterexample for safe registers

The following example shows that the algorithm of [Lamport79] is not correct for the more general case of *safe* registers (see [Lamport86])—where a read overlapping with a write may get any admissible value whatsoever.

There are two customers X and Y which act at the indicated times as follows:

12.00–12.05:	X and Y both write 1 into their registers and the array
12.05–12.10:	Y reads 1 from $R[X]$
12.10–12.40:	Y writes ticket 2 into $R[Y]$ and the array
12.15–12.20:	X reads from $R[Y]$ getting (by overlap) 17
12.25–12.30:	X writes ticket 18 to $R[X]$ and the array
12.30–12.35:	X reads $R[Y]$ getting 117 (by overlap)
12.45–12.50:	Y reads 18 from $R[X]$
13.00:	X and Y both go to CS

It is however easy to adapt the present proof to show correctness of the algorithm of [Lamport74] for safe registers, rephrased as an appropriate evolving algebra, using the same abstract conditions C0–C3.

Acknowledgement. We thank Uri Abraham for useful discussions.

Bibliography

[Abraham93] Abraham, U. *Bakery Algorithms*. Manuscript. 1993, pp.35

[Gurevich91] *Evolving Algebras. A Tutorial Introduction*, EATCS Bulletin 43, February 1991, pp. 264–284. A slightly revised version appeared in "Current Trends in Theoretical Computer Science", Eds. G. Rozenberg and A. Salomaa, World Scientific, 1993, 266–292.

[Gurevich94] Gurevich, Y., *Evolving Algebra 1993: Lipari Guide*, this volume

[Lamport74] L.Lamport, *A new solution of Dijkstra concurrent programming problem.* In: *Comm. ACM*, vol.17,8, 453-455.

[Lamport79] L.Lamport, *A New Approach to Proving the Correctness of Multiprocess Programs.* In: *ACM Transaction on Programming Languages and Systems*, vol.1.1, July 1979, 84-97.

[Lamport86] L.Lamport, *On Interprocess Communication.* In: *Distributed Computing*, vol.1, 77-101.