Asynchronous Multi-Agent ASMs with atomic or durative real-time actions

(Real-Time Bakery Algorithm for Mutual Exlusion)

Egon Börger

Dipartimento di Informatica, Universita di Pisa
http://www.di.unipi.it/~boerger

For details see Chapter 2 (Asynchronous ASMs with Durative Actions) of:

E. Börger, R. Stärk

Abstract State Machines

A Method for High-Level System Design and Analysis

Springer-Verlag 2003

For update info see AsmBook web page:

http://www.di.unipi.it/AsmBook

- Definition of Asynchronous Multi-Agent ASMs
- Asynchronous multi-agent real-time ASM with atomic actions: Lamport's Bakery Algorithm
- An Abstract Bakery Algorithm (for Analysis)
- Analysis of the Bakery ASMs
  - Correctness of the Abstraction
  - Correctness & Liveness of the Abstract Bakery Algorithm
- Refining Atomic to Durative Actions
  - Stable States Resulting from Durative Actions
  - Correctness and Fairness Proofs for Bakery Algorithms
- Exercises and References

Recalling Basic ASMs

- … finite sets of rules, executed by a single agent self, of form
  - If Condition Then Updates
- A computation step is defined as follows:
- In the current state (well-defined global structure) S:
  - determine all the fireable rules in S (s.t. Cond is true in S)
  - compute all expressions t_i,t occurring in updates $f(t_1,\ldots,t_n):=t$
    - this includes reading of the monitored functions, supposed to have well-defined stable values for each execution step
    - execute simultaneously all these function updates
- The updating yields a next state S', which is well-defined by the updates - and by possible changes of the monitored function values, thought of as resulting from implicit actions of the environment
  - environment actions are assumed to be "located" with respect to a machine step in such a way that their result is ready when the machine is going to perform its next step (implicit environment-controller separation principle)

Problems Faced by Asynchronous Multi-Agent Runs

- In runs of asynchronously cooperating independent agents, the possible incomparability of moves - coming with different data, clocks, moments and duration of execution - makes it difficult
  - to uniquely define a global state where moves are executed
  - to locate changes of monitored fcts in the ordering of moves
- The **coherence condition** in the defn of asynchronous multi-agent ASM runs below postulates well-definedness for a relevant portion of state in which an agent is supposed to perform a step, thus providing a notion of "local" stable view of "the" state in which an agent makes a move.
- Changes of monitored functions can be made explicit as resulting from moves of one or more environment agents.
  - Such **monitored moves** - moves of "unkown" agents, as opposed to the **controlled moves** of the explicitly given agents - are often thought of as implicitly given, e.g. by speaking of "time moments" in which changes of monitored functions take place.

Defining Asynchronous Multi-Agent ASMs

- An **Asynchronous Multi-Agent ASM** is a family of pairs (a, ASM(a)) of
  - agents a $\in$ Agent (a possibly dynamic set)
  - basic ASMs ASM (a)
- A **Run** of an asynchronous multi-agent ASM is a partially ordered set (M, $<$) of "moves" m of agents s.t.:
  - finite history: each move has only finitely many predecessors, i.e. {m' | m' $\leq$ m } is finite for each m$\in$ M
  - sequentiality of agents: for each agent a $\in$ Agent, his moves {m | m$\in$ M, a performs m} are linearly ordered by $<$
  - coherence: each (finite) initial segment X of (M, $<$) has an associated state σ (X) – the result of all moves in X with m executed before m' if m<m'– which for every maximal element m$\in$ X is the result of applying move m in state σ(X-{m})

Initial segments of asynchronous multi-agent ASM runs

- **Linearization Lemma.** Let X be a finite initial segment (downward closed subset) of a run of an asynchronous multi-agent ASM. Each linearization of X yields a run with the same final state.
  - Proof: follows from the coherence condition
- Let m be a move of an agent in a run of an asynchronous multi-agent ASM.
- Let Post(m) = {I| I initial segment & m is maximal in I}. Post (m) represents the multiplicity of states, resulting from the last move m and each depending via $\sigma(I) = \sigma(I-\{m\})$ from its "pre"-state.
- Let Pre(m) = {I-{m}| $I \in$ Post(m) } representing the multiplicity of states in which move m can be performed.
- **Pre(m)-Lemma.** Pre(m) has minimal and maximal elements:

  min Pre(m) ={m'|m' < m} max Pre(m) ={m'|not m' ≥ m}

  Pre(m) = {I| I initial segment & min Pre(m) $\subseteq$ I $\subseteq$ max Pre(m) }.

  Therefore Pre(m) is closed under union and intersection.

Characterizing Concurrent Moves in Asynchronous Multi-Agent ASM Runs

- **Concurrent-Moves Lemma.** In runs of an asynchronous multi-agent ASM, the concurrency (incomparability by the run order) of moves m,m' of agents is equivalent to any of the following:
- minPre(m) $\cup$minPre(m') $\in$ Pre(m) $\cap$Pre(m') both moves can come after all the moves which come before any of the two
- Pre(m) $\cap$Pre(m') $\neq \emptyset$
- There exist I,J $\in$ Pre(m) with m' $\in$ I-J
  before one of the moves, the other may be executed or not
- Proof. If m,m' are concurrent, then I = minPre(m) $\cup$minPre(m') is an initial segment and minPre(m) $\subseteq$I $\subseteq$ maxPre(m), so that I $\in$ Pre(m). By symmetry then I $\in$ Pre(m').
- Let J $\in$ Pre(m) $\cap$Pre(m') and set I = J $\cup${m'}. Then I $\in$ Pre(m).
- Suppose I,J $\in$ Pre(m) with m' $\in$ I-J. m'<m would imply m' $\in$ minPre(m) $\subseteq$ J and thus m'$\in$J. m<m' would imply m'$\notin$ maxPre(m) $\supseteq$I and thus m'$\notin$ I. Therefore m, m' are incomparable, i.e. concurrent.

Stability and change of term values in initial segments

- Extending term evaluation from states to state sets:
  - $Val_{Pre(m)}(t) = c$ if all initial segments in Pre(m) have the same value c for t: $\forall I \in Pre(m): Val_{\sigma(I)}(t) = c$

    We then say that $Val_{Pre(m)}(t)$ is indisputable, writing also that $t_{Pre(m)}$ (or its value) is indisputable
  - $Val_{Pre(m)}(t) =$ undef otherwise
- Defn: move m may change the value of t iff m changes the value of t in some linearization of the given run, formally:

$$\text{for some } I \in Pre(m): Val_{\sigma(I)}(t) \neq Val_{\sigma(I \cup \{m\})}(t)$$

- Defn: m must change the value of t iff m changes the value of t in every linearization of the given run, formally:

$$\text{for every } I \in Pre(m): Val_{\sigma(I)}(t) \neq Val_{\sigma(I \cup \{m\})}(t)$$

Conditions for indisputability of terms

- **Sufficient indisputability condition.** If $t_{Pre(m)}$ is not indisputable, then there is a move m' concurrent with m which may change t.
  - Proof by contradiction. By the Pre(m)-lemma, for every $I \in Pre(m)$ one can reach $\sigma(I)$ from $\sigma(minPre(m))$ by applying moves concurrent to m. By assumption none of them may change t, so that $Val_{\sigma(I)}(t) = Val_{\sigma(minPre(m))}(t)$. That would mean that $t_{Pre(m)}$ is indisputable.
- **Necessary indisputability condition.** If there is a move m' concurrent with m which must change t, then $t_{Pre(m)}$ is not indisputable.
  - Proof . By the concurrent-moves lemma, there is a common initial segment $I \in Pre(m) \cap Pre(m')$. Then also $I \cup \{m'\} \in Pre(m)$, and since m' must change t, I and $I \cup \{m'\}$ have different values for t.

Moves with indisputable different term values compare

- **Lemma.** If the values of $t_{Pre(m)}$ and of $t_{Pre(m')}$ are different but both indisputable, then m and m' are not concurrent but compare in the run order (i.e. m<m' or m'<m).
- Proof. If $t_{Pre(m)}$ and $t_{Pre(m')}$ are indisputable and m,m' are concurrent, then by the concurrent-moves lemma there is a common initial segment $I \in Pre(m) \cap Pre(m')$ so that $t_{Pre(m)} = Val_{\sigma(I)}(t) = t_{Pre(m')}$

Focused Terms

- Defn: A term t is focused in a given run if any move which may change its value must change it.
- Sufficient Condition for Focus. If the value of a term t may be changed only by one agent a, then it is focused.
- Proof. It suffices to show that $t_{Pre(m)}$ is indisputable
  - because then $t_{Post(m)}$ is also indisputable, and the assumption that only a may change the value of t implies: $t_{Post(m)}$ is different from $t_{Pre(m)}$ iff m changes t

  By the sufficient indisputability condition, $t_{Pre(m)}$ is indisputable because moves concurrent to m must be moves of other agents than a, therefore by assumption none of them may change t.
- Synchronization Lemma. For a focused term t, $t_{Pre(m)}$ is indisputable iff , in the given partial order, m compares with all moves which change the value of t.

  Proof: follows from the indisputability and focus conditions.

- Definition of Asynchronous Multi-Agent ASMs
- Asynchronous Multi-Agent Real-Time ASM with Atomic Actions:

  Lamport's Bakery Algorithm

- An Abstract Bakery Algorithm (for Analysis)
- Analysis of the Bakery ASMs
  - Correctness of the Abstraction
  - Correctness & Liveness of the Abstract Bakery Algorithm
- Refining Atomic to Durative Actions
  - Stable States Resulting from Durative Actions
  - Correctness and Fairness Proofs for Bakery Algorithms
- Exercises and References

Mutual Exclusion Problem

- **The Problem** (Dijkstra) . Define a mutual exclusion program, i.e. a protocol which guarantees the following properties, if it is applied each time a process - among any finite number of processes - wants to enter a "critical code section":
  - mutual exclusion: it never happens that two processes are simultaneously in the critical section
  - deadlock freedom: every process which attempts to enter the critical section will eventually enter it
  - fairness: the first-come-first-served principle holds (for a reasonable notion of coming-first which has to be defined)

Lamport's Bakery Algorithm for Mutual Exclusion

- **Idea** (Lamport 1974) . When customer process $\mathbf{P}_i$ ($1 \leq i \leq n$) wants to enter the critical code section CS, it
  - **StartS** by entering the doorway, showing its interest
  - fetches the next available new **Ticket** when it canGetTicket (after doorwayRead of the current ticket of every process, adding 1 to the maximum ticket encountered), then continues to waitReadCheck the tickets until it canGo because there is no other process with a smaller ticket (NB. No bound on tickets)
  - **EnterS** CS to execute the Critical Section code
  - **ExitS** CS, moving outside, when it has done the CS code
- Tickets of $\mathbf{P}_i$ are written exclusively by $\mathbf{P}_i$
  - in a register $\mathbf{R}_i$. Registers are shared for reading, to copy them
  - into a private array P, recording for the array owner $\mathbf{P}_i$ what it has last got as value read in the register of any process
- doorway/waitReadCheck of $\mathbf{R}_j$ & recording the value in P ($\mathbf{P}_i$, $\mathbf{P}_j$) is done independently for each i,j (by reader agents)

Customer Signature in Lamport Bakery ASM

- CUSTOMER : finite static domain of process agents
- READER finite static domain of reader agents
- Each customer self has the following state signature:
  - controlled fct mode : {doorway, wait, CS, outside}
  - auxiliary monitored fcts (events) interested, done triggering entry into the bakery resp. exit from the critical section
  - register R:Nat where tickets are posted to the other customers. R(self) is controlled (private write) for self, R(Y) for Y≠self is monitored (public read) for self.
  - P:CUSTOMER → Nat. P(self) is controlled for self P(Y) for Y≠self is monitored for self
  - reader : CUSTOMER – {self} → READER static injective fct yielding the readers of self which report ticket values from registers of other customers . Locations mode $_{reader(c)}$ are monitored for self (for inspection of wait/doorway values)

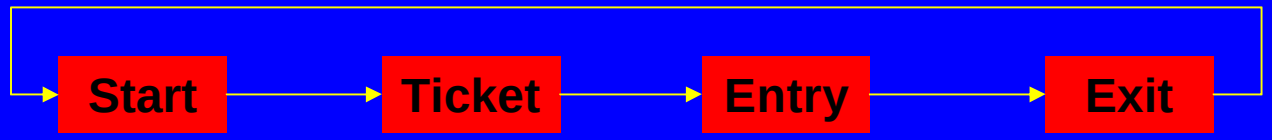
Reader Signature in Lamport Bakery ASM

- Each reader self, one per pair of customers to guarantee independent readings, has a state of the following signature:
  - controlled fct mode : {doorway, wait, check}
  - master, subject : CUSTOMER interpreting each reader as serving its unique master customer c to report the ticket from the register of its unique subject customer y. More precisely the following function is injective in c,y:

$$reader_c(y) = \iota\, r\ (r \in READER,\ master_r = c,\ subject_r = y)$$

  - controlled location $P_{master}(subject)$ (for subject ≠ master), monitored locs $P_{master}(master)$ (for read), $mode_{master}$ (for inspecting values wait and doorway) , R(subject)
- Initially mode = outside, R = P(c) = 0 for each customer c. For each reader mode = doorway.
  - We also write reader(X,Y) instead of $reader_X(Y)$.

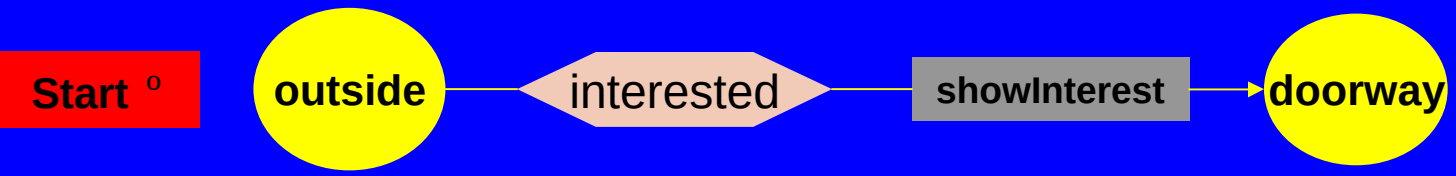
ASM Rules for any Basic Bakery Customer self

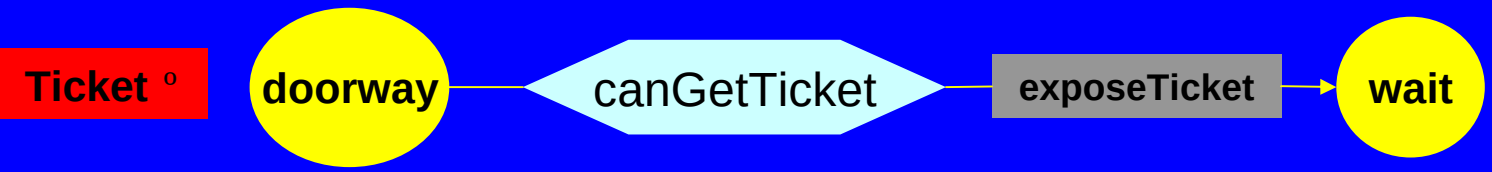


showInterest =
R :=1, P(self) := 1

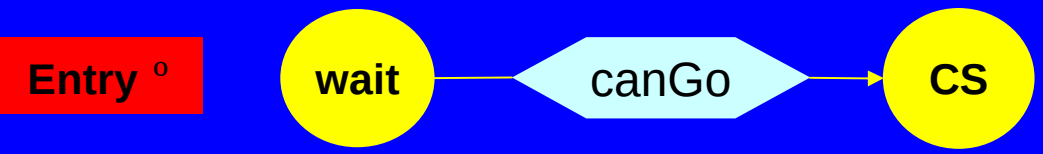
exposeTicket =
R :=nextTicket
P(self) := nextTicket

cancelTicket =
R :=1





interested, done:
monitored fcts



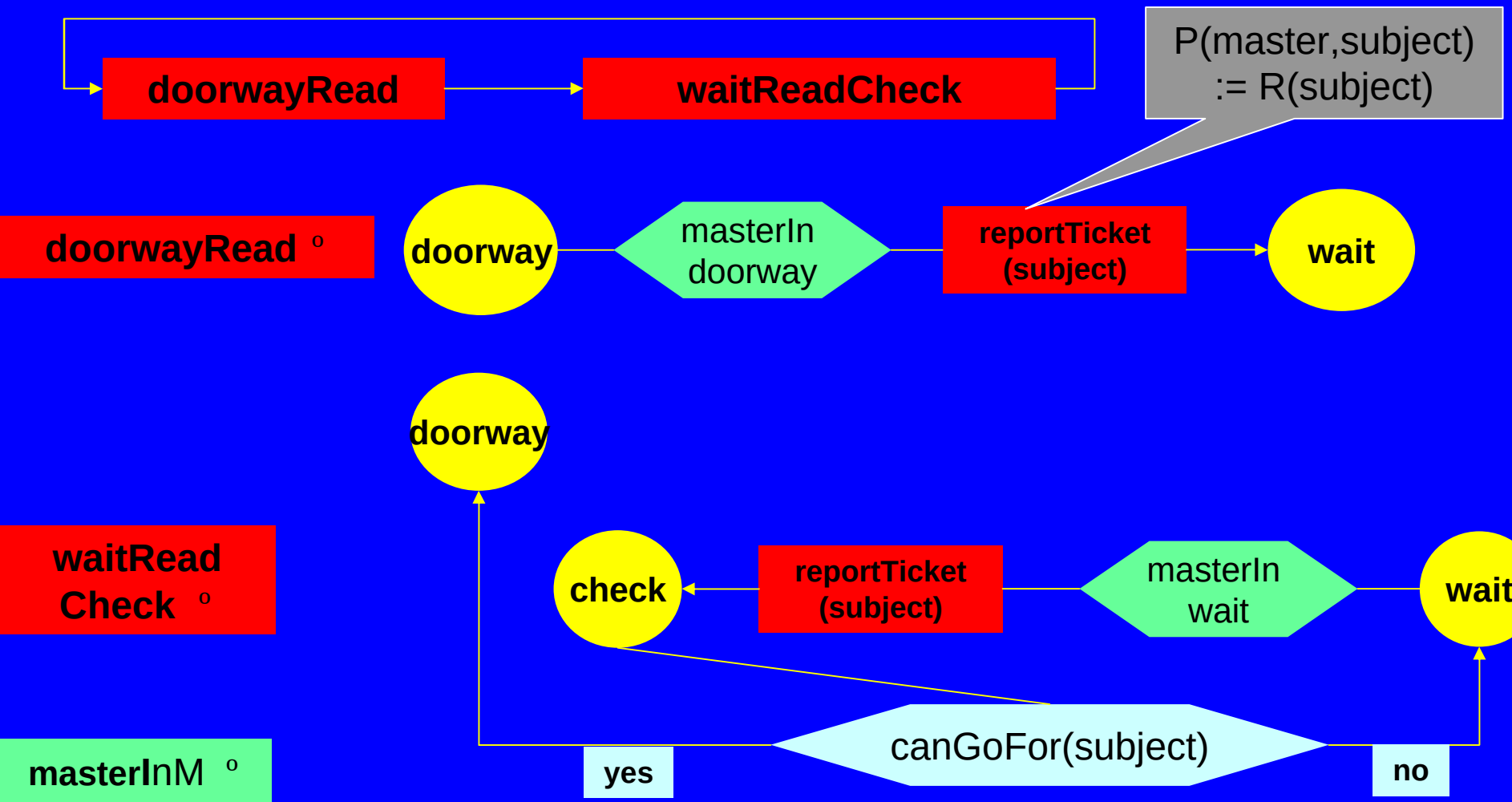
nextTicket ° 1 +
max {P(X) | X ∈ CUSTOMER**}**

canGetTicket & canGo
defined as terminated
doorwayRead resp.
waitReadCheck of all the
other tickets by the readers of
self

canGetTicket ° every reader of self is in mode wait

canGo ° every reader of self is in mode doorway

ASM Rules for any BasicBakery Reader self



notInterested(subject) ° $P_{master}(subject) = 0$

comesLater(subject) °
$P_{master}(master) < P_{master}(subject)$ or
$P_{master}(master) = P_{master}(subject)$ & master < subject

canGoFor(subject) °
notInterested(subject) or
comesLater(subject)

Projecting Atomic Actions in Runs of Asynchronous Multi-Agent Bakery ASM to Real-Time

- Assuming for "moves" (rule applications) of agents in runs of the Bakery ASM atomicity of actions (namely read/write to shared registers) performed at (non-negative) real-time moments,
  - moves take place in zero time (i.e. the state change due to a move at time t becomes effective at time t+ε for sufficiently small ε)
  - two moves of one agent take place at different moments (by the sequentiality of agents)
  - NB. The real-time ordering of moves by the moments they take place makes certain moves comparable, e.g. moves of the readers belonging to one master, which in Lamport's original formulation are intended to be independent (indeed done in parallel using a forall rule). To reflect this, one could allow moves of different agents to take place at the same moment. Instead, we define a high-level model which abstracts completely from readers.

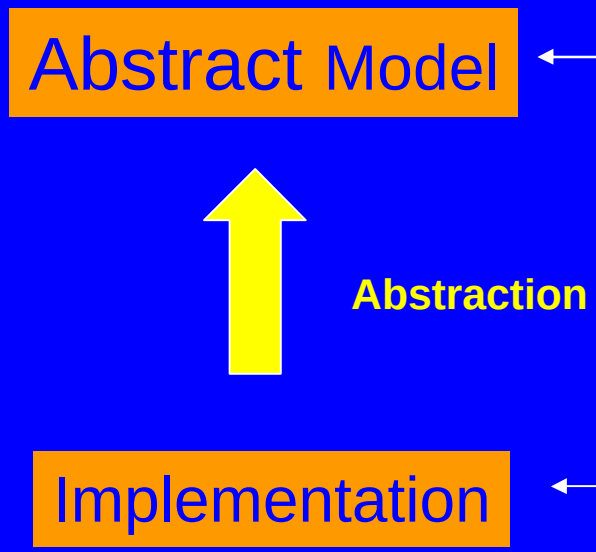
Diligence Assumption for Real-Time Runs of the Bakery ASM with Atomic Actions

- We assume that
  - no agent can become continuously enabled without eventually making a move
  - done will eventually become true each time the corresponding customer is in CS, whereas interested may eventually stay false forever
- Denote by S_t the state at time t, resulting from all moves taking place before t – by the finite-history property of runs of asynchronous multi-agent ASMs, there are only finitely many such moves.

- Definition of Asynchronous Multi-Agent ASMs
- Asynchronous Multi-agent Real-Time ASM with atomic actions: Lamport's Bakery Algorithm
- An Abstract Bakery Algorithm (for Analysis)
- Analysis of the Bakery ASMs
  - Correctness of the Abstraction
  - Correctness & Liveness of the Abstract Bakery Algorithm
- Refining Atomic to Durative Actions
  - Stable States Resulting from Durative Actions
  - Correctness and Fairness Proofs for Bakery Algorithms
- Exercises and References

Methodological Verification Scheme for an implementation to meet desired design properties

Principle: divide & conquer (ancient math paradigm)



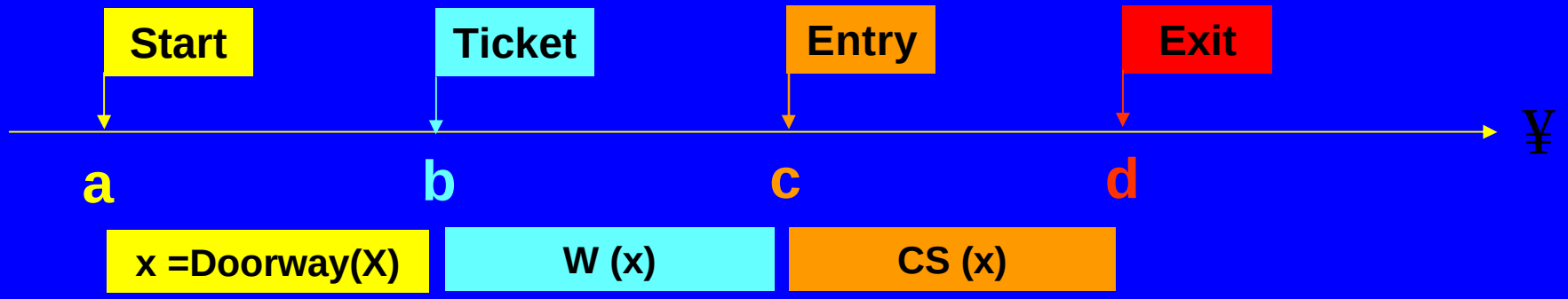
1. Build an abstract model & prove **properties** from assumptions (axioms)

3. Prove **assumptions** to hold in concrete model

2. Prove correctness of abstraction

Succession of Real-Time Intervals of Customers' Visits

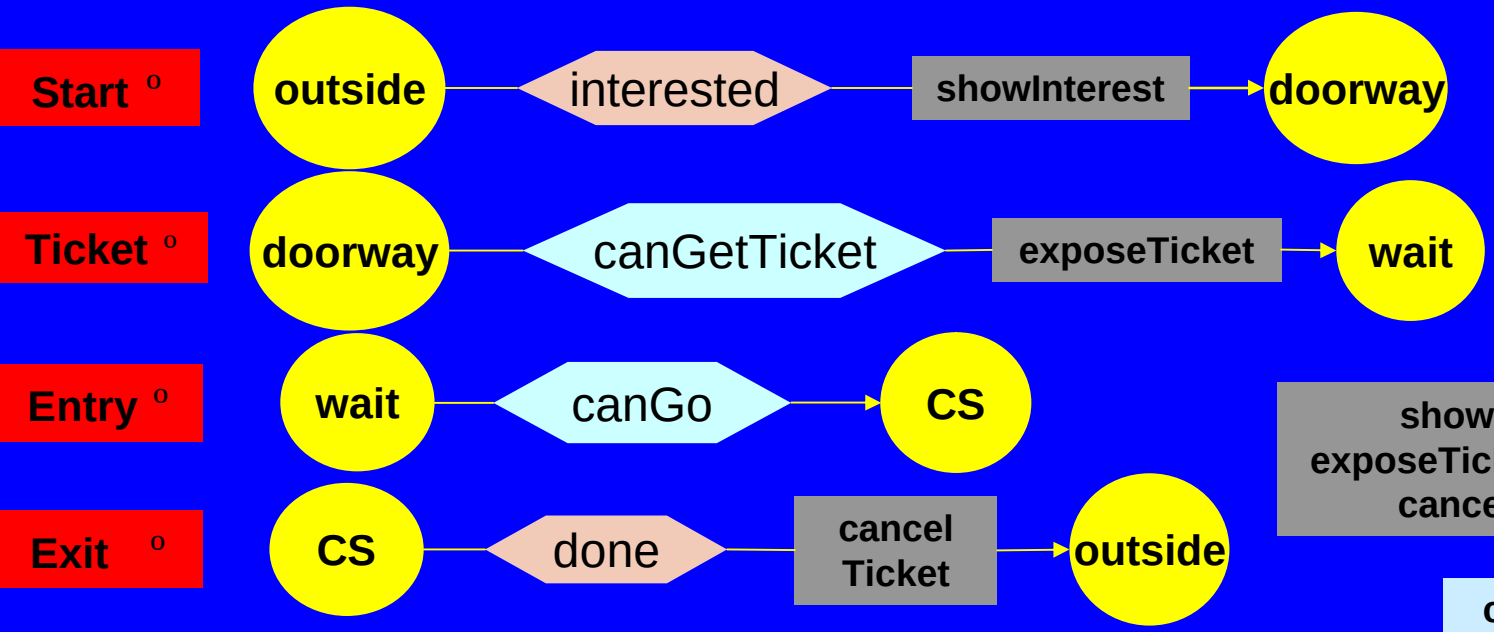
- If agent X executes Start and then Ticket at moments a,b
  - the open interval x = (a,b) is called a Doorway of X
    - during which the readers of X do doorwayRead
- If X after Ticket executes Entry and Exit at moments c,d
  - W(x) = (b,c) is called the Wait interval of x
    - during which the readers of X do waitReadCheck
  - CS(x) = (c,d) is called the CriticalSection interval of x
  - if after b X makes no more move, we set W(x) = (b,∞), CS (x) = undef. To prove: for each doorway x, W(x) is bounded & CS (x) is defined.
- By the assumption that no agent stalls forever, every doorway - if defined also every CS(x) - is bounded.



High-Level Bakery ASM: no readers, same customer rules but with monitored nextTicket, canGo

canGo ° Go(self)
with constrained monitored Go

nextTicket ° T(self) with
T: CUSTOMER ⟶ Nat
constrained monitored T



showInterest = R :=1
exposeTicket = R := nextTicket
cancelTicket = R :=1

canGetTicket = true

Doorway Ordering

- Goal: impose conditions on
  - assigning T(x) to customers X leaving a doorway x to enter W(x) - so that the doorway determines entry order into CS (Bakery Definability Property [Abraham 2001])
  - permitting customers to Go from wait interval to CS such that runs of the high-level Bakery ASM satisfy mutual exclusion in accessing CS, fairness, deadlock freedom.
- Notation: For f:CUSTOMER $\rightarrow$ Nat let
  - $f'(X) = N \times f(X) + id(X)$ if $f(X) > 0$ wlog $0 \leq id(X) < N$
    $= \infty$ else so that $T'(x) \neq T'(y)$ if $X \neq Y$
- Ordering of doorways x,y of agents $X \neq Y$:
  - $x \blacktriangleleft y$ iff $x \cap y \neq \emptyset$ & $T'(x) < T'(y)$ doorways overlap, tickets are ordered
  - $x \blacktriangleleft\blacktriangleleft y$ iff $x<y$ or $x \blacktriangleleft y$ (where $x<y$ iff $\forall a \in x\ \forall b \in y: a<b$)

    x comes earlier than or overlaps with y but has a smaller ticket
- Lemma. $\blacktriangleleft\blacktriangleleft$ is a linear order of doorways of different agents

Constraints on Monitored Functions T and Go

- C0: $T(x) \in Nat$, $T(x) > 1$ tickets are natural nos. >1
- C1: If $y<x$, then either $CS(y)<sup(x)$ or $T'(y)<T'(x)$

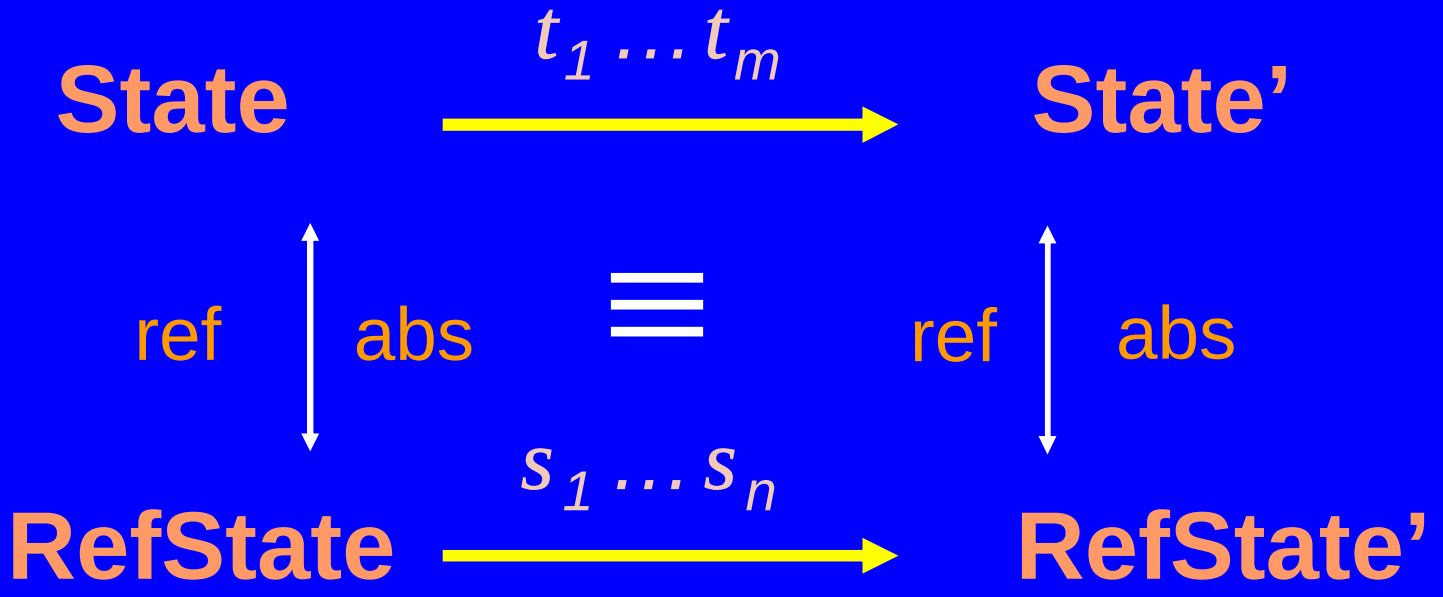
  If a doorway of Y comes before a doorway of X, then either Y has left the corresponding CS before X gets the ticket or the tickets of Y and X respect the temporal precedence of doorways with overlapping wait intervals
- C2: If Go (X) holds at any moment $t > sup(x)$, then for every $Y \neq X$ there is a moment $b \in W(x)$ such that $T'(x) < R'_b(Y)$

  X can leave its wait interval only if during waitReadCheck its ticket at some moment resulted smaller than the ticket posted in the register of any other customer
- C3 (induction principle for waiting intervals): If W(y) is bounded for all $y \lhd x$, then W(x) is bounded
  - $term_b$ denotes the value of term in S_b (the state at time b)

- Definition of Asynchronous Multi-Agent ASMs
- Asynchronous Multi-agent Real-Time ASM with atomic actions: Lamport's Bakery Algorithm

  An Abstract Bakery Algorithm (for Analysis)
- Analysis of the Bakery ASMs
  - Correctness of the Abstraction
  - Correctness & Liveness of the Abstract Bakery Algorithm
- Refining Atomic to Durative Actions
  - Stable States Resulting from Durative Actions
  - Correctness and Fairness Proofs for Bakery Algorithms
- Exercises and References

Correct Refinement/Abstraction Step



≡ defined

- between states of interest
- reached by comp segments of interest
- relating the locations of interest

Abstraction Relation bw Lamport & High-Level Bakery ASM

- State (data) mapping (abstracting from READER states & P) :
  - bw homonymous CUSTOMER, mode, R, interested, done
  - nextTicket $\mapsto$ T, canGo $\mapsto$ Go
  - corresponding states of interest: states in which a move Start, Ticket, Entry, Exit respectively is executed
- Mapping of computation segments (move sequences):
  - Start of X followed by one doorwayRead move of each reader of X - i.e. until canGetTicket (X) becomes true - $\mapsto$ Start of X
  - Ticket of X $\mapsto$ Ticket of X Exit of X $\mapsto$ Exit of X
  - WaitReadCheck moves of every reader of X = master – i.e. until canGo(X) - followed by Entry of X $\mapsto$ Entry of X
- Mapping & equivalence of locations of interest: via C0-C3

Lemma: Lamport Bakery ASM correctly implements high-level Bakery ASM

- To show: C0-C3 satisfied by Lamport Bakery ASM with
  - Go(X) = canGo(X) at some t in the Lamport Bakery ASM
  - T(X) = 1 + max {P_X(Y) | Y ∈ CUSTOMER} = nextTicket
- ad C0: when nextTicket is taken at sup(x), X contributes to the maximum with value P_x (X) = 1 (set in Start move), so that T(X) >1
- ad C1: Let t = time of the reader(X,Y)-move during x
  - Case 1. Y applies Exit from CS(y) before this reading time t, i.e. during (sup(y),t). Then CS(y)<t<sup(x).
  - Case 2. Otherwise. Then R_t(y)=T(y) and therefore
    T(x) ≥1+ R_t(y)>T(y)>0. Hence T ’(x) > T ’(y).
- ad C2: can-go(X) becomes true at t>sup(x) when for every Y ≠ X, $reader_X$(Y) has finished his waitReadCheck moves in W(x). Thus for each Y ≠ X, the moment of the last wait-reading move of $reader_X$(Y) is a moment b ∈ W(x) such that T’(x) <R'_b(Y), as established by the following waitCheck move of $reader_X$(Y)

Proof: Lamport Bakery ASM correctly implements the high-level Bakery ASM

- ad C3: Proof by contradiction: assume W(y) bounded for all $y \blacktriangleleft x$ but W(x) unbounded.
  - Claim1: $\exists$ b $\in$ W(x) later than any sup(CS(y)) (if $y \blacktriangleleft x$) and later than any sup(y) (if $x \blacktriangleleft y$)
  - Claim 2: Claim 1 implies that every reader(X,Y) finishes his waitReadCheck moves in W(x), so that W(x) is bounded.
- Proof 2. If reader(X,Y) finishes his waitReadCheck moves before b $\in$ W(x), it finishes them in W(x). Otherwise: by defn of b in Claim 1, no Y $\neq$ X has mode doorway at t $\geq$ b. Therefore at t $\geq$ b, reader(X,Y) waitReads 0 or T(y) for some $y \blacktriangleleft\blacktriangleleft x$. In both cases the next waitCheck succeeds (in the second case by T '(x) < T '(y), true by C1 if x<y & by defn of $\blacktriangleleft$ if $x \blacktriangleleft y$) .
- Proof 1: By run co-finiteness there are only finitely many $y \blacktriangleleft\blacktriangleleft x$, all with sup(CS(y)) < ∞ (since W(y) is bounded & no module stalls forever). Also for each Y there is at most one $y \blacktriangleleft\blacktriangleleft x$, since $y \blacktriangleleft\blacktriangleleft x$ implies T '(x) < T '(y) - in case of y>x by C1 & unbounded W(x).

- Definition of Asynchronous Multi-Agent ASMs
- Asynchronous Multi-agent Real-Time ASM with atomic actions: Lamport's Bakery Algorithm
- An Abstract Bakery Algorithm (for Analysis)
- Analysis of the Bakery ASMs
  - Correctness of the Abstraction
  - Correctness (Mutual Exclusion & Fairness) & Liveness of High-Level Bakery ASM
- Refining Atomic to Durative Actions
  - Stable States Resulting from Durative Actions
  - Correctness and Fairness Proofs for Bakery Algorithms
- Exercises and References

Correctness proof for the high-level Bakery ASM

- FCFS & Mutual Exlusion Lemma: If $y \blacktriangleleft\blacktriangleleft x$ and W(x) is bounded, then W(y) is bounded and CS(y) < CS(x).
- Indirect proof. Since W(x) is bounded, by C2 there is for Y some $b \in W(x)$ such that $T'(x) < R'_b(Y)$.
  - Claim 2: $T'(y) < T'(x)$ follows from assumption not CS(y) < CS(x).
  - Claim 1: sup(y) < b
- For some ε, $R'_{\mathbf{sup(y)+}\,\varepsilon}(Y) = T'(y) < T'(x) < R'_b(Y)$ implies that Y must be writing to its register R(Y) sometime in (sup(y),b). But the first register write after sup(y) takes place in an Exit-move. Therefore Y has left CS(y) during W(x) - implying CS(y) < CS(x) – and W(y) is bounded, contradicting the assumption.
- Proof 2: Follows for $y \blacktriangleleft x$ from the defn of $\blacktriangleleft$, for y<x from C1.

  Proof 1: $y \blacktriangleleft\blacktriangleleft x$ implies $\inf(y) \leq \sup(x) < b$. But $b \in (\inf(y),\sup(y)]$ would imply $R_b(Y)=1$, contradicting $1< T(x)$ & $T'(x)<R'_b(Y)$. Therefore b>sup(y).

Deadlock Freedom proof for the high-level Bakery ASM

- **Liveness Lemma**: Every W(x) is bounded.
- Proof: follows from C3, the induction principle for the ordering of doorways by ⋖ (see transitivity lemma below for ⋖) which is well-founded by the co-finiteness property.
- **Transitivity Lemma.** ⋖ is transitive.
- Proof by contradiction. Assume $x \lessdot y \lessdot z \lessdot x$ and let n be the number of occurrences of < in this chain.
  - In case n=0 or n=2,3 a contradiction follows from the transitivity and non-reflexivity of the ordering < of reals and of real intervals.
  - Case n=1: wlog assume $x \lessdot y \lessdot z < x$, so that $T'(x) < T'(y) < T'(z)$. By the FCFS lemma, $x \lessdot y \lessdot z \lessdot x$ implies that W(x) ,W(y), W(z) are infinite, so that CS(z) is undefined. C1 then implies $T'(z) < T'(x)$: a contradiction.
- **Resumé**: Doorways are linearly ordered by ⋖. Every waiting interval is bounded, $x \lessdot y$ implies CS(x)<CS(y).

- Definition of Asynchronous Multi-Agent ASMs
- Asynchronous Multi-agent Real-Time ASM with atomic actions: Lamport's Bakery Algorithm
- An Abstract Bakery Algorithm (for Analysis)
- Analysis of the Bakery ASMs
  - Correctness of the Abstraction
  - Correctness & Liveness of the Abstract Bakery Algorithm
- **Refining Atomic to Durative Actions**
  - Stable States Resulting from Durative Actions
  - Correctness and Fairness Proofs for Bakery Algorithms
- Exercises and References

Real-Time Asynchronous Multi-Agent ASM Runs with Durative Actions

- … are runs of asynchronous multi-agent ASMs s.t.
  - every move μ of any agent (in particular reading of and writing to shared registers) takes place during a non-empty open real-time interval, say Time (μ) = (a,b) with $0 \leq a < b$ (b real or ∞)
  - moves are ordered by the ordering of their (possibly overlapping) time intervals
  - every agent satisfies a separation principle for moves concerning monitored and controlled locations, establishing which moves take place without changes of which monitored locations (to guarantee well defined values of monitored locations during the intervals in which moves of agents take place) and assuring the coherence condition of partial order runs
- Assumptions are the same as for atomic actions:
  - every agent which is enabled will eventually make a move
  - on interested and done

Lamport Bakery Separation Principle for Monitored and Controlled Locs

Denote by $S_t(A)$ the local state of agent A at time t (global state restricted to locs of A). Separation conditions below guarantee the coherence condition for customer & reader moves.

- No customer move & no reader waitCheck move does overlap with any monitored move for any of its monitored locs
  - assuring $S_b(A)$ = the result of applying μ to $S_a(A)$ for these moves μ of agents A with Time (μ) = (a,b)
- No read-move does overlap with a move concerning the mode of its master, but it may overlap with a write-move concerning the register of its subject. Thus impose Register Regularity:
- let Time (μ) = (a,b) for a read-move μ of r = reader (X,Y). The read-result stored in P(X,Y) in state $S_b(r)$ is the value of R(Y) at any of
  - either Y's last passive moment $\leq$ a before starting the read-move
  - or a passive moment of Y in the read-move interval (a,b)

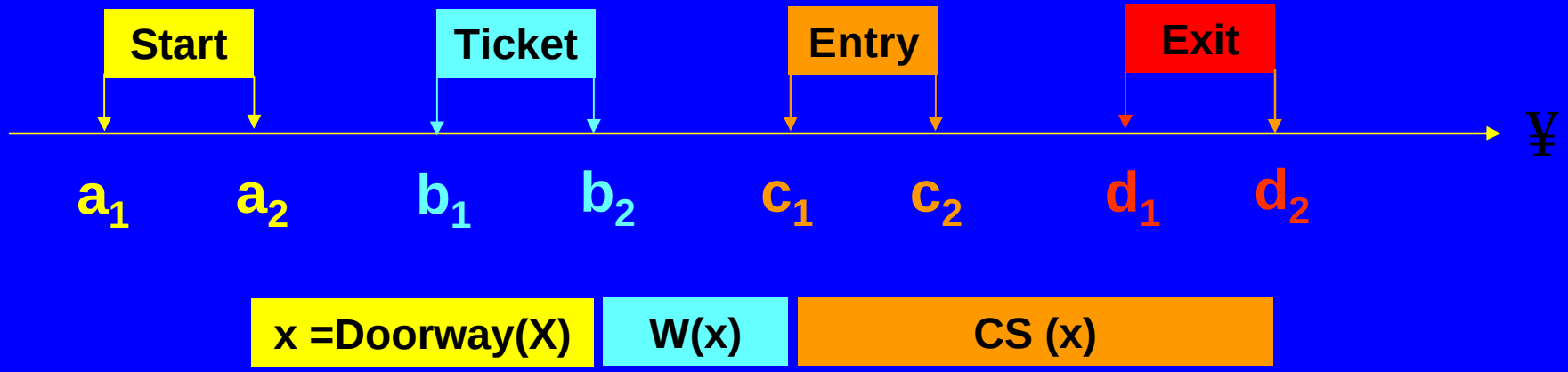
where A is passive in I iff I $\cap$ Time (μ) = $\emptyset$ for every move μ of A

Stability & Sequentiality Lemma for Agents with Durative Actions

- Stability Lemma. If [a,b] is a passive interval of an agent A in a run of the Lamport Bakery ASM with durative actions, then Controlled-S_b(A) = Controlled-S_a(A), where Controlled-S_c(A) is the restriction of S_c(A) to the controlled locations of A.
  - Proof: by definition only A can update its controlled locs.
  - NB: Meantime the monitored locs may change their values.
- Sequentiality Lemma. For each agent A, the set of duration times of moves of A is linearly ordered by the < relation on open real intervals.
  - Proof: follows from the sequentiality condition of partial order runs.
- NB. If A executes μ during (a,b) and then μ' during (c,d), the interval [b ,c] is passive for A.

Refining Real-Time Intervals of Customers' Visits for Durative Actions

- If agent X executes Start during (a_1,a_2) and then Ticket during (b_1,b_2), the Doorway of X starts when Start is finished and lasts until Ticket has finished, i.e. $x = (a_2,b_2)$
- If X after Ticket executes Entry during (c_1,c_2) and Exit during (d_1,d_2), Wait interval of x starts when Ticket has finished and lasts until the beginning of Entry: $W(x) = (b_2,c_1)$
  - CS(x) = (c_1,d_2) starts when Entry begins and lasts until Exit is finished



Refining Go-Constraint C2 for Durative Actions

- C0,C1,C3 remain as in the high-level ASM with atomic actions

C2 has to be adapted to regular register reading when checking ticket values in a waiting interval against values posted by competitors:

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

  X can leave its wait interval only if for every other customer Y, its ticket at some passive moment for Y, satisfying the register regularity principle, resulted smaller than the ticket posted in the register of Y

- Definition of Asynchronous Multi-Agent ASMs
- Asynchronous Multi-agent Real-Time ASM with atomic actions: Lamport's Bakery Algorithm
- An Abstract Bakery Algorithm (for Analysis)
- Analysis of the Bakery ASMs
  - Correctness of the Abstraction
  - Correctness & Liveness of the Abstract Bakery Algorithm
- Refining Atomic to Durative Actions
  - Stable States Resulting from Durative Actions
  - Correctness and Fairness Proofs for Bakery Algorithms with Durative Actions
- Exercises and References

High-Level axioms C0-C3 hold for Lamport Bakery ASM with durative actions

- C0, C3 carry over from atomic to durative actions without change.
- ad C1: Let μ be the doorwayRead move of reader (X,Y) during x.
  - Case 1. Y makes an Exit-move ν from CS(y) ending not later than μ, i.e. such that sup Time(ν) ≤ sup Time (μ) . Then CS(y) < sup Time (ν) ≤ sup Time (μ) ≤ sup(x).
  - Case 2. Otherwise. Let Time (μ) = (a,b).
    - If Y makes an Exit-move ν from CS(y), set e = sup Time (ν) > sup Time (μ)=b, otherwise set e = ∞ > b.
    - Then R(Y)=T(y) holds over [sup(y),e) ⊇ [sup(y),b),
    - in particular $R_t(Y)=T(y)$ for the moment chosen via the register regularity principle for μ (NB. Y is passive at sup(y)).
    - Therefore $T(x) \geq 1 + R_t(Y) > T(y) > 0$. Hence T ’(x) > T ’(y).
- ad C2: Go(X) becomes true when every reader(X,Y) finishes his waitReadCheck moves. For Y ≠ X consider the last waitRead move μ of reader(X,Y) during W(x). As desired b take the moment chosen via the register regularity principle for μ.

Correctness proof for the high-level Bakery ASM with durative actions

- Only the proof of FCFS & Mutual Exlusion Lemma has to be adapted: If $y \blacktriangleleft\!\blacktriangleleft x$ and W(x) is bounded, then W(y) is bounded & CS(y) < CS(x).
- Proof by contradiction. For Y choose b by C2, passive for Y.
  - Claim1: T ’(y) < T ’(x) Claim 2: sup(y) < b
- for some $\varepsilon>0$, $R'_{sup(y)+\varepsilon}(Y) = T'(y) < T'(x) < R'_b(Y)$ implies that Y must be writing to its register R(Y) sometime in (sup(y),b), so that by C2 this write starts before sup W(x). The first register write after sup(y) takes place in an Exit-move. Therefore Y has left CS(y) during W(x) - implying CS(y) < CS(x) – & W(y) is bounded, contradicting the assumption.
- Proof 1: Follows for $y \blacktriangleleft x$ from the defn of $\blacktriangleleft$, for y<x from C1.
  Proof 2: $y \blacktriangleleft\!\blacktriangleleft x$ implies inf(y) ≤ b (since by regularity of register reads b<inf(y) forces b to be the last passive moment ≤ inf(W(x)), whereas inf(y) is passive for Y & inf(y)<sup(x)=inf W(x)). $b \in [inf(y),sup(y))$ would imply $R_b(Y)=1$, contradicting $1< T(x)\&T'(x) <R'_b(Y)$. Finally $b \neq sup(y)$ since othw $R_b(Y)=T(y)$ contradicting Claim 1.

Exercises to Bakery ASM

- Lamport's algorithm works with tickets (time stamps) associated to each customer instead of reading values from a global system clock. Define a Bakery ASM which uses global clock values instead of tickets.
  - For a solution see U. Abraham: Bakery Algorithms. Manuscript, Nov 19, 2001, Fig. 3, pg. 6
- Show that the Bakery ASM with durative actions remains correct and fair if the register regularity condition is changed to Lamport's register regularity condition [Lamport 1986] where as the read-result stored in P(X,Y) in state $S_b(r)$ also the value of R(Y) at Y's first passive moment $\geq b$ can be taken.
  - For a solution see [Börger, Gurevich, Rosenzweig 1995]

Exercise to Bakery ASM

- Show that Lamport's register regularity condition allows overtaking of readers to happen. Hint: Consider a reader and a writer where for two reads overlapping with a write, the first read gets the later write value and the second read the earlier one.
- Show the correctness and fairness of the Bakery ASM with durative actions also for safe registers (where a read that is concurrent (overlapping) with a write may return any value, but otherwise returns the result of the last write preceding it , see [Lamport 1986]). Hint: adapt the above proofs, using C0-C3, or see [Abraham 2001].
- Lift the Bakery ASMs and their analysis to general partially ordered runs of distributed ASMs, which are not mapped any more to continuous real-time.
  - Hint: Use the analysis of initial segments of distributed runs. For a solution see [Gurevich,Rosenzweig 2000]

References

- U. Abraham: Bakery Algorithms. Manuscript, Nov 19, 2001, pp.41
  - Bakery algorithm with bounded tickets
- U. Abraham: Models for Concurrency. Gordon & Breach 1999
- E. Börger, Y. Gurevich, and D. Rosenzweig: The Bakery Algorithm: Yet another Specification and Verification.
  - In: Specification and Validation Methods (Ed. E. Börger). Oxford University Press, 1995, 231-243
- E. Börger, R. Stärk: Abstract State Machines. A Method for High-Level System Design and Analysis
  - Springer-Verlag 2003, see http://www.di.unipi.it/AsmBook

References

- J. Cohen and A. Slissenko: On Verification of Refinements of Timed Distributed Algorithms.
  - In: Abstract State Machines. Theory and Applications. (Ed. Y.Gurevich et al). Springer LNCS 1912, 2000, 34-49
- Y. Gurevich and D. Rosenzweig: Partially Ordered Runs: A Case Study.
  - In: Abstract State Machines. Theory and Applications. (Ed. Y.Gurevich et al). Springer LNCS 1912, 2000, 131-150
- L. Lamport: A new solution of Dijkstra's concurrent programming problem.
  - In: Comm.ACM vol.17,8 (1974), 453-455
- L. Lamport: On Interprocess Communication. Part I: Basic Formalism, Part II: Algorithms.
  - In. Distributed Computing 1 (1986), pp. 77-101