Goals of the lecture

- Time domain vs Causality domain
- Lamport's Mutual Exclusion Algorithm
- Formal Verification
  - Key Lemmas
  - Safety
  - Liveness
  - Fairness

References: Lamport 79,
Garg and Tomlinson 94

Time domain vs Causality domain

- most problems require causality domain
  - accounts for variable execution schedule
- problems in causality domain easier
  - mutual exclusion
  - ordering of messages
  - observing a global property

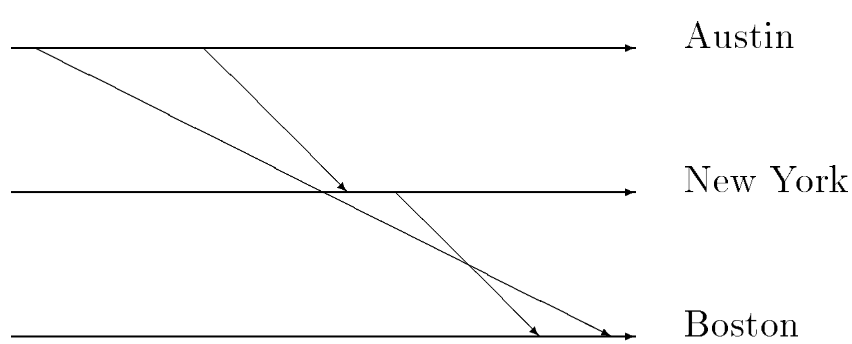
Properties of the Mutual Exclusion Algorithm

- a fixed number of processes
- a shared resource called the critical section (CS).
- Task is to coordinate processes.
- Requirements are:

Safety: Two processes should not use the CS simultaneously.

Liveness: Every request for the CS is eventually granted.

Fairness: Requests must be granted in the order they are made.



Formal Specification

Lamport's algorithm assumes that all channels are FIFO

$$s \prec t \wedge s \leadsto u \wedge t \leadsto v \Rightarrow \neg(v \prec u)$$

- $req(s) = P_{s.p}$ has requested the critical section
- $cs(s) = P_{s.p}$ has permission to enter the critical section in s
- Cooperation assumption:

$$cs(s) \Rightarrow (\exists t : s \prec t : \neg req(t))$$

Formal Requirements

$$s \| t \Rightarrow \neg(cs(s) \wedge cs(t)) \qquad \textbf{(Safety)}$$
$$req(s) \Rightarrow (\exists t :: s \prec t \wedge cs(t)) \qquad \textbf{(Liveness)}$$

$$next\_cs(s) = min\{t \mid s \prec t \wedge cs(t)\}$$
$$req\_start(s) = req(s) \wedge \neg req(s.prev)$$

$req\_start(s) = P_{s.p}$ made a request for the CS in state s.

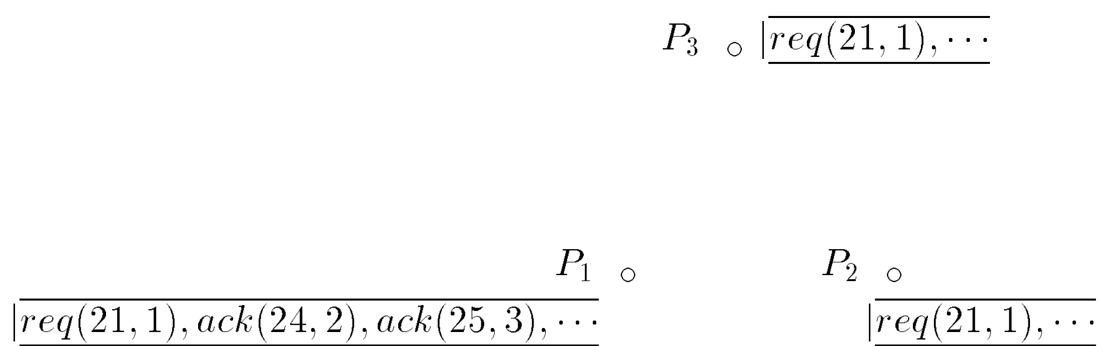
$$(req\_start(s) \wedge req\_start(t) \wedge s \rightarrow t) \Rightarrow next\_cs(s) \rightarrow next\_cs(t)$$

(Fairness)

- $next\_cs(s)$ and $next\_cs(t)$ exist due to liveness.
- $next\_cs(s)$ and $next\_cs(t)$ are not concurrent due to safety.

Informal Specification of the Mutual Exclusion Algorithm

- **request CS**: send a timestamped message to all other processes and add a timestamped request to the queue.
- **On receiving a request**: the request and its timestamp is stored in the queue and an acknowledgment is returned.
- **To release the CS**: send a release message to all other processes.
- **On receiving a "release"**: delete the corresponding request from the queue.



Informal Specification [Contd.]

- can access CS if
  - it has a request in the queue with timestamp t, and
  - t is less than all other requests in the queue, and
  - it has received a message from every other process with timestamp greater than t.

P_3 ○ $|req(21,1), req(24,2)\cdots$

P_1 ○ $|req(21,1), req(24,2), ack(25,3), \cdots$

P_2 ○ $|req(21,1), req(24,2)\cdots$

Formal Description

- Local variables in each state s:
  $s.q[1..n]$: integer initially ∞
  $s.v$: DDClock
- To request the critical section in t where $s \prec_1 t$:
  $t.q[t.p] = s.v[t.p]$
  for all $j : j \neq t.p$: send "request" to P_j
- On receiving "request" in state t sent from state u $(u \leadsto t)$:
  $t.q[u.p] = u.q[u.p]$
  send ack to $u.p$
- To release the critical section in state t:
  $t.q[t.p] = \infty$
  for all $j \neq t.p$, send "release" to P_j
- On receiving "release" sent from state u:
  $t.q[u.p] = \infty$

Formal Description [Contd.]

State s has permission to access the critical section when

- there is a request from $P_{s.p}$ with timestamp less than all other requests
- and $P_{s.p}$ has received a message from every other process with a timestamp greater than the timestamp of its own request.

Formal description of $CS(s) \equiv$

$$\forall j : j \neq s.p : (s.q[s.p], s.p) < (s.v[j], j) \ \wedge \ (s.q[s.p], s.p) < (s.q[j], j).$$

Proof of Correctness

We define the predicate

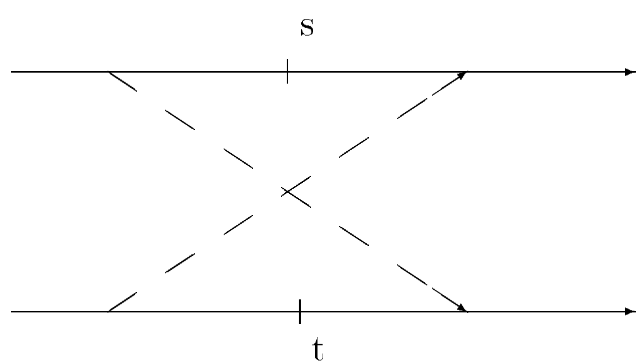
$$msg(s,t) \equiv (\exists u, t' : u \leadsto t' \wedge u \prec s \wedge t \prec t')$$

That is, there exists a message which was sent by $P_{s.p}$ before s and received by $P_{t.p}$ after t.

Lemma 1 *Assume FIFO.* $\forall s,t : s.p \neq t.p : s \not\rightarrow t \wedge \neg msg(s,t) \Rightarrow t.q[s.p] = s.q[s.p]$.

The following Lemma is crucial in proving the safety property.

Lemma 2 $\forall s,t : s.p \neq t.p : s \not\rightarrow t \wedge s.q[s.p] < t.v[s.p] \Rightarrow t.q[s.p] = s.q[s.p]$

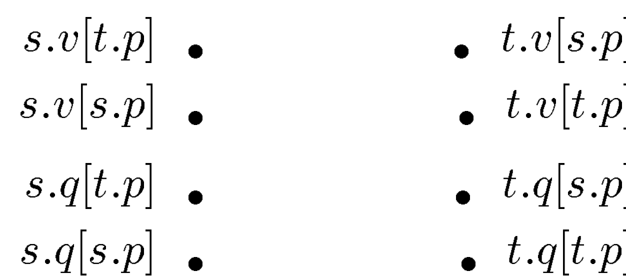


Safety Property

Lemma 3 (Safety) $s.p \neq t.p \wedge s \| t \Rightarrow \neg(cs(s) \wedge cs(t))$.

Proof: We will show that $(s \| t) \wedge cs(s) \wedge cs(t)$ implies false.
Case 1: $t.v[s.p] < s.q[s.p] \wedge s.v[t.p] < t.q[t.p]$
We get the following cycle.

$s.q[s.p]$
$<$ { $cs(s) \wedge s.p \neq t.p$ }
$s.v[t.p]$
$<$ { this case }
$t.q[t.p]$
$<$ { $cs(t) \wedge s.p \neq t.p$ }
$t.v[s.p]$
$<$ { this case }
$s.q[s.p]$.

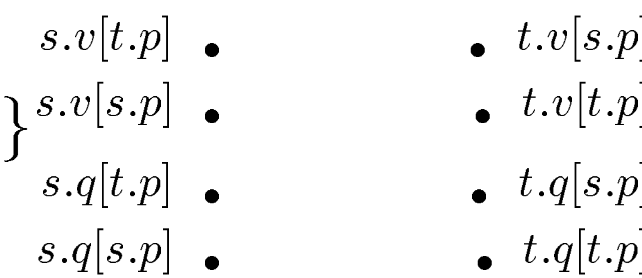


Safety Property [Contd.]

Case 2: $s.q[s.p] < t.v[s.p] \wedge t.q[t.p] < s.v[t.p]$

We get the following cycle.

$$
\begin{array}{ll}
 & s.q[s.p] \\
< & \{\ cs(s) \wedge s.p \neq t.p\ \} \\
 & s.q[t.p] \\
= & \{\ t.q[t.p] < s.v[t.p],\ t \not\rightarrow s,\ \text{Lemma 2}\ \} \\
 & t.q[t.p] \\
< & \{\ cs(t) \wedge s.p \neq t.p\ \} \\
 & t.q[s.p] \\
= & \{\ s.q[s.p] < t.v[s.p],\ s \not\rightarrow t,\ \text{Lemma 2}\ \} \\
 & s.q[s.p].
\end{array}
$$

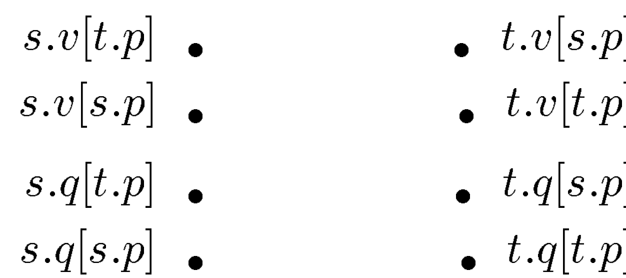


Safety Property [Contd.]

Case 3: $s.q[s.p] < t.v[s.p] \wedge s.v[t.p] < t.q[t.p]$

We get the following cycle.

$$
\begin{array}{l}
\quad s.q[s.p] \\
< \{\ cs(s) \wedge s.p \neq t.p\ \} \\
\quad s.v[t.p] \\
< \{\ \text{this case}\ \} \\
\quad t.q[t.p] \\
< \{\ cs(t) \wedge s.p \neq t.p\ \} \\
\quad t.q[s.p] \\
= \{\ s.q[s.p] < t.v[s.p],\ s \not\rightarrow t,\ \text{Lemma 2}\ \} \\
\quad s.q[s.p].
\end{array}
$$



Case 4: Similar to case 3. ∎

Liveness Property

Lemma 4 (Liveness) $req(s) \Rightarrow \exists t : s \prec t \wedge cs(t)$

Proof: $req(s)$ is equivalent to $s.q[s.p] \neq \infty$. $s.q[s.p] \neq \infty$ implies that there exists $s_1 \in P_{s.p}$ such that
$s_1.v[s.p] = s.q[s.p] \wedge event(s_1) = request$.
We show existence of the required t with the following two claims:
Claim 1:
$\exists t_1 : \forall j \neq s.p : t_1.v[j] > s.q[s.p] \wedge s.q[s.p] = t_1.q[s.p]$
Claim 2:
$\exists t_2 : \forall j \neq s.p : t_2.q[j] > s.q[s.p] \wedge s.q[s.p] = t_2.q[s.p]$

Fairness Property

Lemma 5 (Fairness) $(req\_start(s) \wedge req\_start(t) \wedge s \rightarrow t)$ $\Rightarrow (next\_cs(s) \rightarrow next\_cs(t))$

Proof:
Let $s' = next\_cs(s)$ be state in which critical section is acquired, and let s'' be state which it is released. Let $t' = next\_cs(t)$.
Let r be the state in $P_{t.p}$ which received the request message sent from s.

Fairness Property [Contd.]

We know the following facts:

1. $r \preceq t$, due to FIFO channels.
2. $t.v[t.p] = t.q[t.p]$, due to request event at t.
3. $s.v[s.p] < t.v[t.p]$, since $s \rightarrow t$ $(DD2)$.
4. $s.q[s.p] = s.v[s.p]$, due to request event at s.
5. $r.q[s.p] = s.q[s.p]$, due to receiving request at r.
6. $r.q[s.p] < t.q[t.p]$, from $2, 3, 4, 5$.
7. $t.q[t.p] = t'.q[t.p]$, by defn of t'.
8. $t'.q[t.p] \leq t'.q[s.p]$, since $cs(t')$.
9. $r.q[s.p] < t'.q[t.p] \leq t'.q[s.p]$, from $6, 7, 8$.

This means that $q[s.p]$ must be increased between r and t'. That can only happen when $P_{t.p}$ receives the release message sent from s''. Thus $s'' \rightarrow t'$. And since $s' \rightarrow s''$, we conclude $s' \rightarrow t'$. ∎