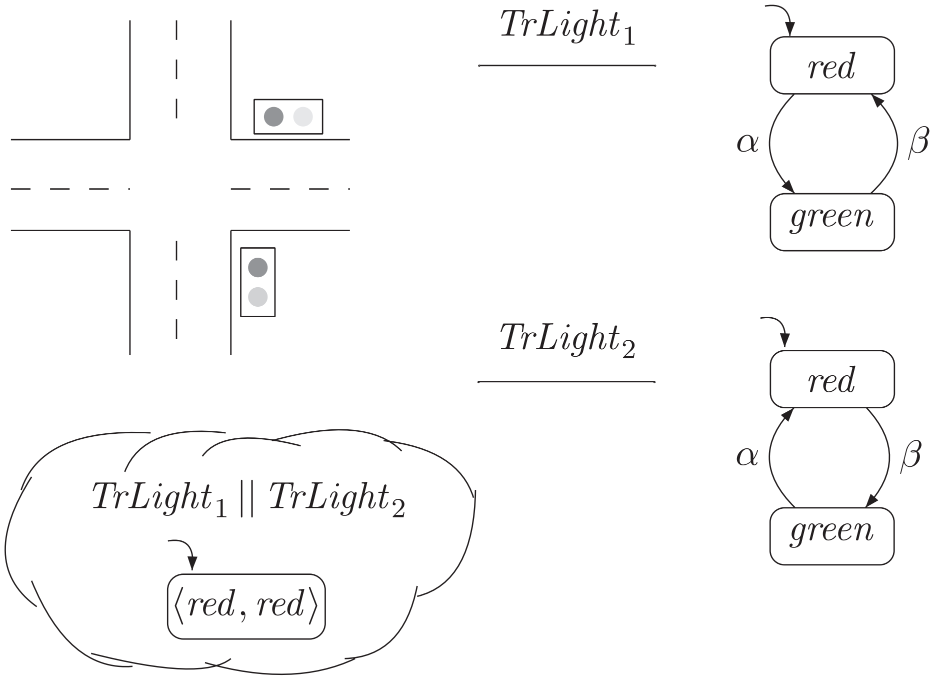
Linear-Time Properties

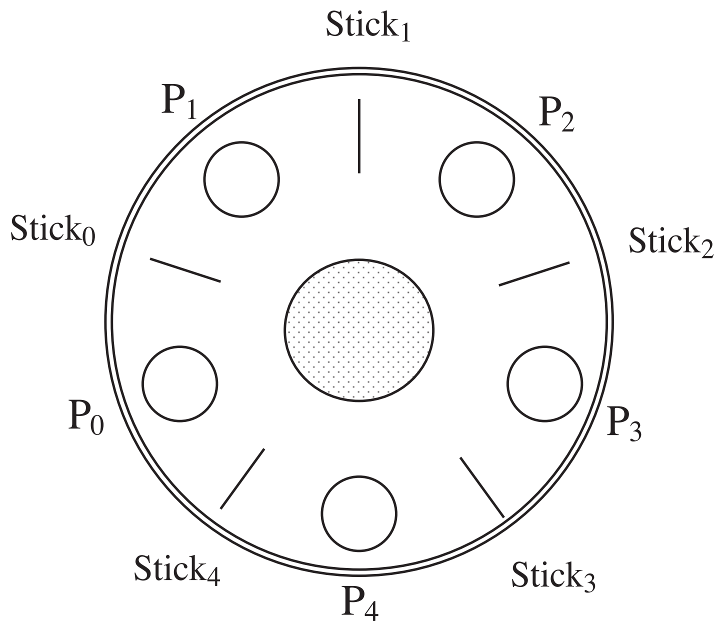
Deadlock

- A typical deadlock scenario occurs when components mutually wait for each other to progress
- both traffic lights start with a red light results in a deadlock

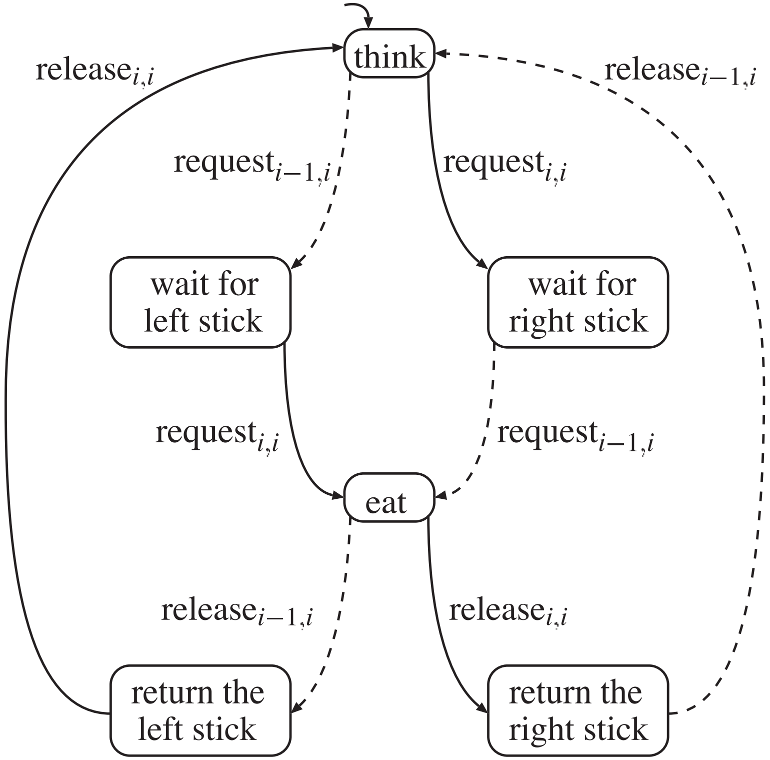


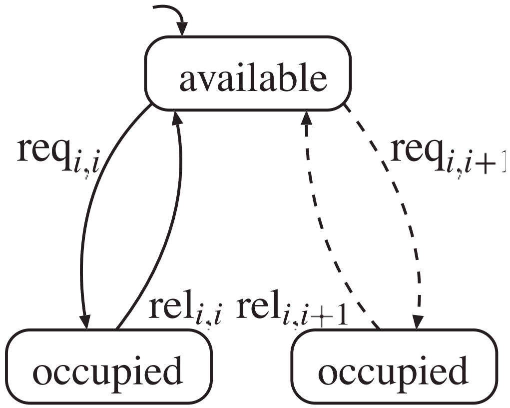
dining philosophers

- Five philosophers are sitting at a round table with a bowl of rice in the middle.
- For the philosophers life consists of thinking and eating .
- To take some rice out of the bowl, a philosopher needs two chopsticks.



TS of philosopher and stick





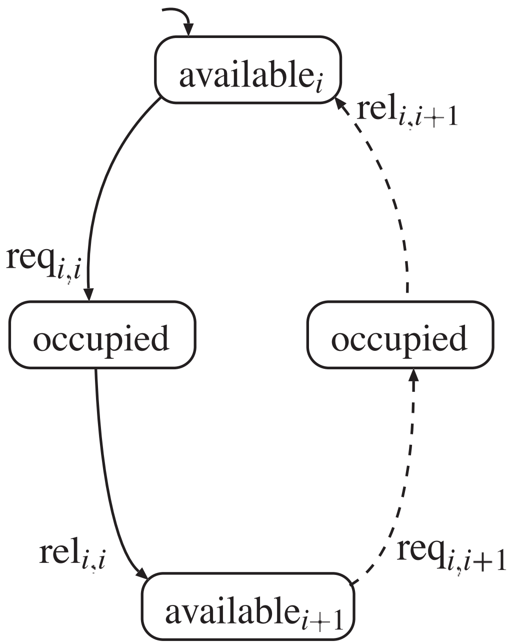
Phil4 || Stick3 || Phil3 || Stick2 || Phil2 || Stick1 || Phil1 || Stick0 || Phil0 || Stick4

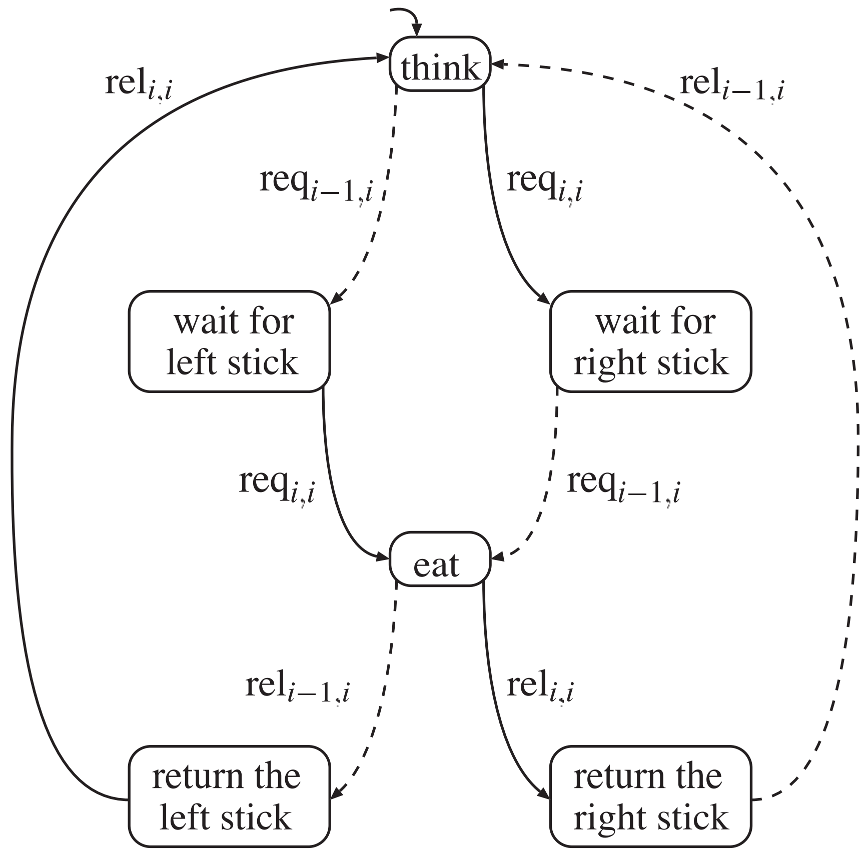
初始状态:
<think4,avail3,think3,avail2,think2,avail1,think1,avail0,think0,avail4>

<wait4,0,occ4,4,wait3,4,occ3,3,wait2,3,occ2,2,wait1,2,occ1,1,wait0,1,occ0,0>
死锁!

a solution of deadlock

- some sticks (e.g., the first, the third, and the fifth stick) start in state availablei,i, while the remaining sticks start in state availablei,i+1.





linear-time behavior

- analyze system
  - an action-based approach
  - a state-based approach ---we conside

Paths and State Graph

- The state graph of TS, notation G(TS), is the digraph (V,E) with vertices V = S and edgesE={(s,s')∈S×S| s' ∈Post(s)}.

- Post*(s) denote the states that are reachable in state graph G(TS) from s.

- $$Post^*(C) \;=\; \bigcup_{s \in C} Post^*(s).$$

- path, finite path, infinite path, maximal path, initial path

- a path of a TS

traces

- trace

$$s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \ldots$$

$$L(s_0)\, L(s_1)\, L(s_2) \ldots$$

- The traces of a transition system are thus words over the alphabet 2^{AP}

trace and trace fragment

Let $TS = (S, Act, \rightarrow, I, AP, L)$ be a transition system without terminal states. The *trace* of the infinite path fragment $\pi = s_0\, s_1 \ldots$ is defined as $\mathit{trace}(\pi) = L(s_0)\, L(s_1) \ldots$. The trace of the finite path fragment $\widehat{\pi} = s_0\, s_1 \ldots s_n$ is defined as $\mathit{trace}(\widehat{\pi}) = L(s_0)\, L(s_1) \ldots L(s_n)$.

$$\mathit{trace}(\Pi) \;=\; \{\, \mathit{trace}(\pi) \mid \pi \in \Pi \,\}.$$

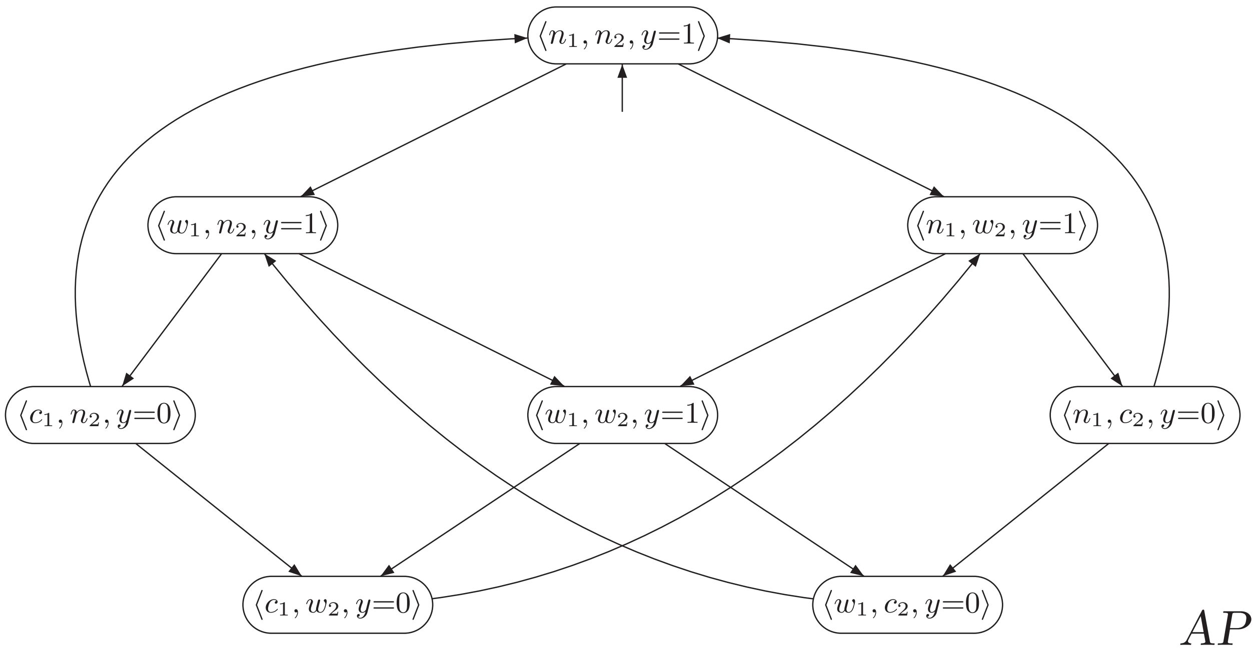
$$\mathit{Traces}(s) = \mathit{trace}(\mathit{Paths}(s))$$

$$\mathit{Traces}(TS) = \bigcup_{s \in I} \mathit{Traces}(s).$$

$$\mathit{Traces}_{\mathit{fin}}(s) = \mathit{trace}(\mathit{Paths}_{\mathit{fin}}(s))$$

$$\mathit{Traces}_{\mathit{fin}}(TS) = \bigcup_{s \in I} \mathit{Traces}_{\mathit{fin}}(s)$$

example semaphore-based mutual exclusion

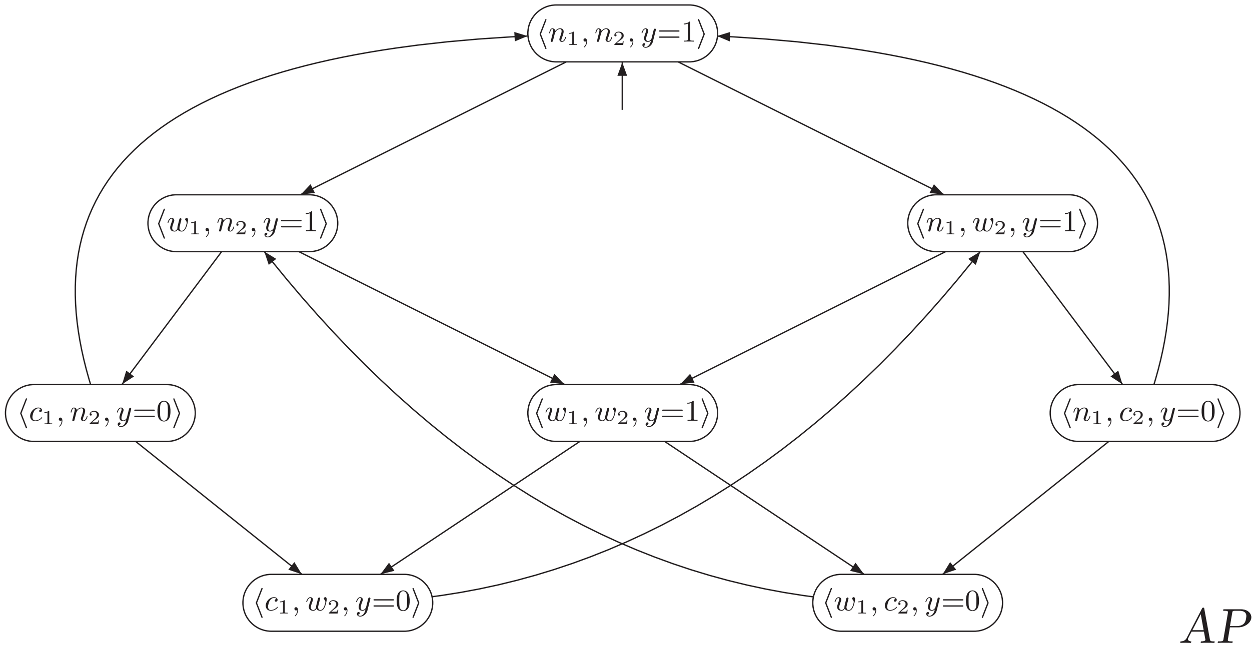


$$AP = \{ \, crit_1, crit_2 \, \}$$

$$\begin{array}{rcl} \pi & = & \langle n_1, n_2, y = 1\rangle \rightarrow \langle w_1, n_2, y = 1\rangle \rightarrow \langle c_1, n_2, y = 0\rangle \rightarrow \\ & & \langle n_1, n_2, y = 1\rangle \rightarrow \langle n_1, w_2, y = 1\rangle \rightarrow \langle n_1, c_2, y = 0\rangle \rightarrow \ldots \end{array}$$

$$trace(\pi) \; = \; \varnothing \, \varnothing \, \{ \, crit_1 \, \} \, \varnothing \, \varnothing \, \{ \, crit_2 \, \} \, \varnothing \, \varnothing \, \{ \, crit_1 \, \} \, \varnothing \, \varnothing \, \{ \, crit_2 \, \} \ldots .$$

example semaphore-based mutual exclusion



$AP = \{ \, crit_1, crit_2 \, \}$

$$\begin{aligned} \widehat{\pi} \quad = \quad & \langle n_1, n_2, y = 1\rangle \rightarrow \langle w_1, n_2, y = 1\rangle \rightarrow \langle w_1, w_2, y = 1\rangle \rightarrow \\ & \langle w_1, c_2, y = 0\rangle \rightarrow \langle w_1, n_2, y = 1\rangle \rightarrow \langle c_1, n_2, y = 0\rangle \end{aligned}$$

$$trace(\widehat{\pi}) \;=\; \varnothing \, \varnothing \, \varnothing \, \{ \, crit_2 \, \} \, \varnothing \, \{ \, crit_1 \, \}$$

Linear-time properties

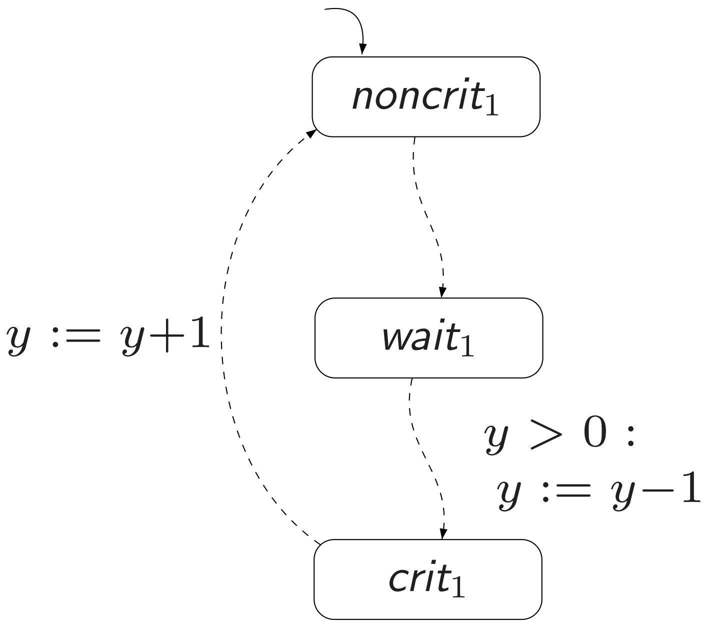
- Linear-time properties specify the traces that a TS must exhibit
  - LT-property specifies the admissible behaviour of the system
  - later, a logical formalism will be introduced for specifying LT properties
- A *linear-time property* (LT property) over *AP* is a subset of $\left(2^{AP}\right)^{\omega}$
  - finite words are not needed, as it is assumed that there are no terminal states
- *TS* (over *AP*) *satisfies* LT-property P (over *AP*):

$$TS \models P \quad \text{if and only if} \quad Traces(TS) \subseteq P$$

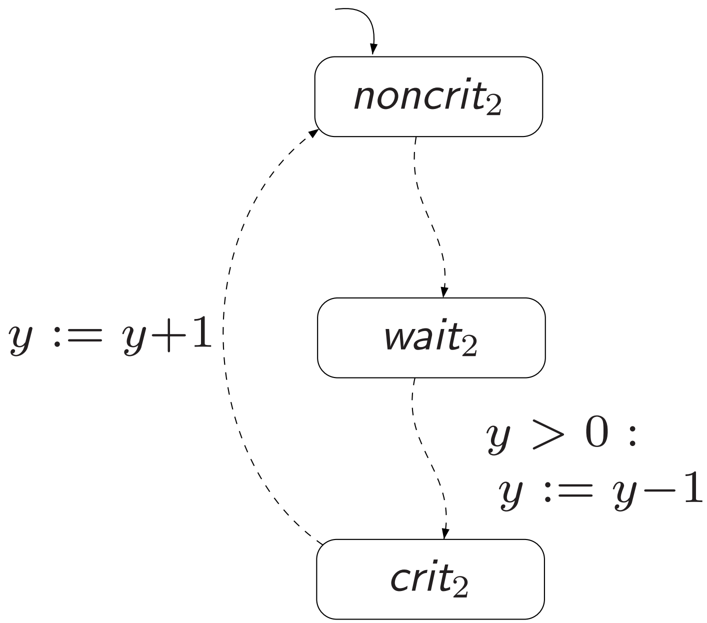
  - *TS* satisfies the LT property P if all its "observable" behaviors are admissible

Semaphore-based mutual exclusion

PG_1 :

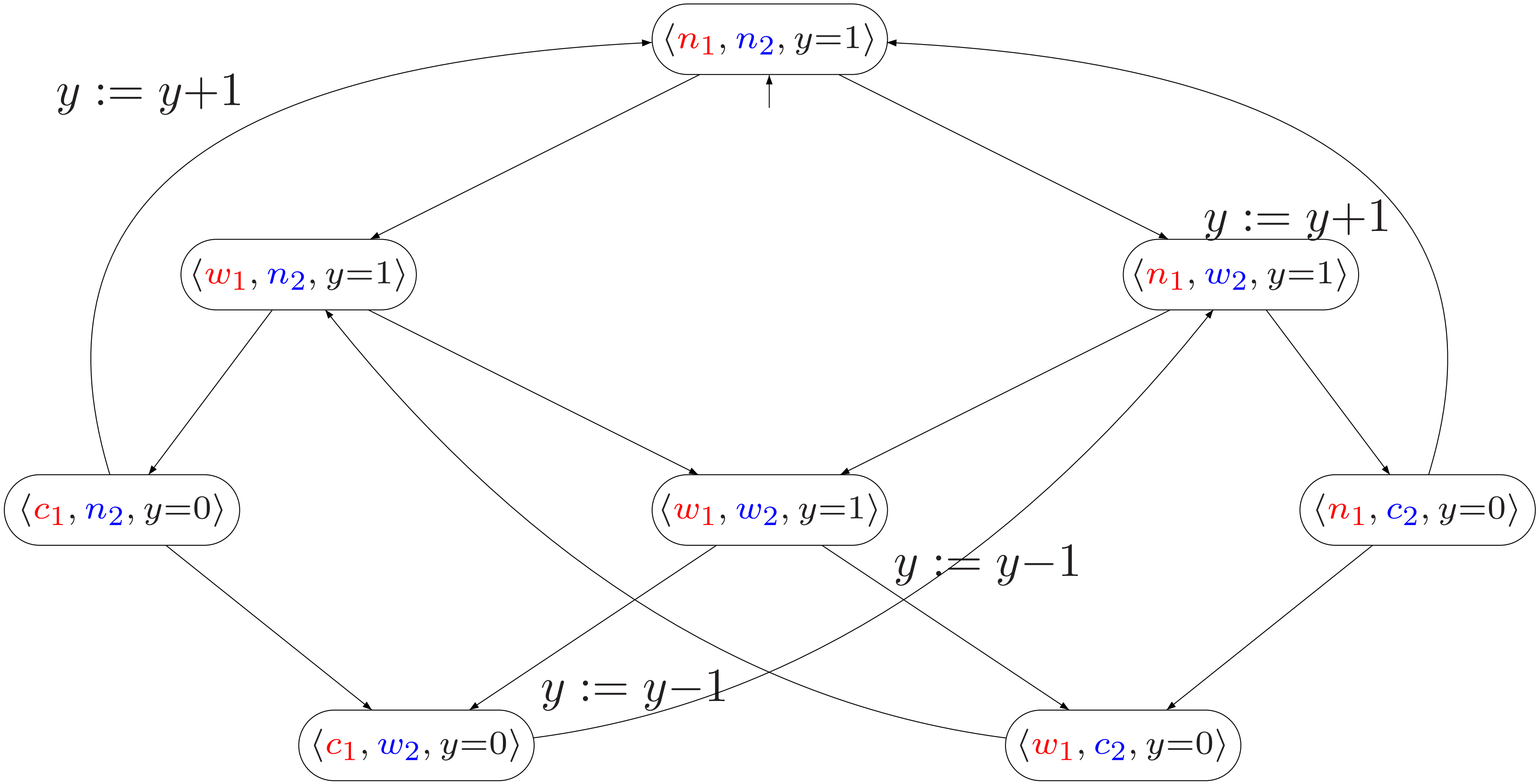


PG_2 :



$y=0$ means "lock is currently possessed"; $y=1$ means "lock is free"

Transition system



How to specify mutual exclusion?

"Always at most one process is in its critical section"

- Let $AP = \{ \mathit{crit}_1, \mathit{crit}_2 \}$
  - other atomic propositions are not of any relevance for this property

- Formalization as LT property

$$P_{mutex} = \text{set of infinite words } A_0 \, A_1 \, A_2 \ldots \text{ with } \{ \mathit{crit}_1, \mathit{crit}_2 \} \not\subseteq A_i \text{ for all } 0 \leqslant i$$

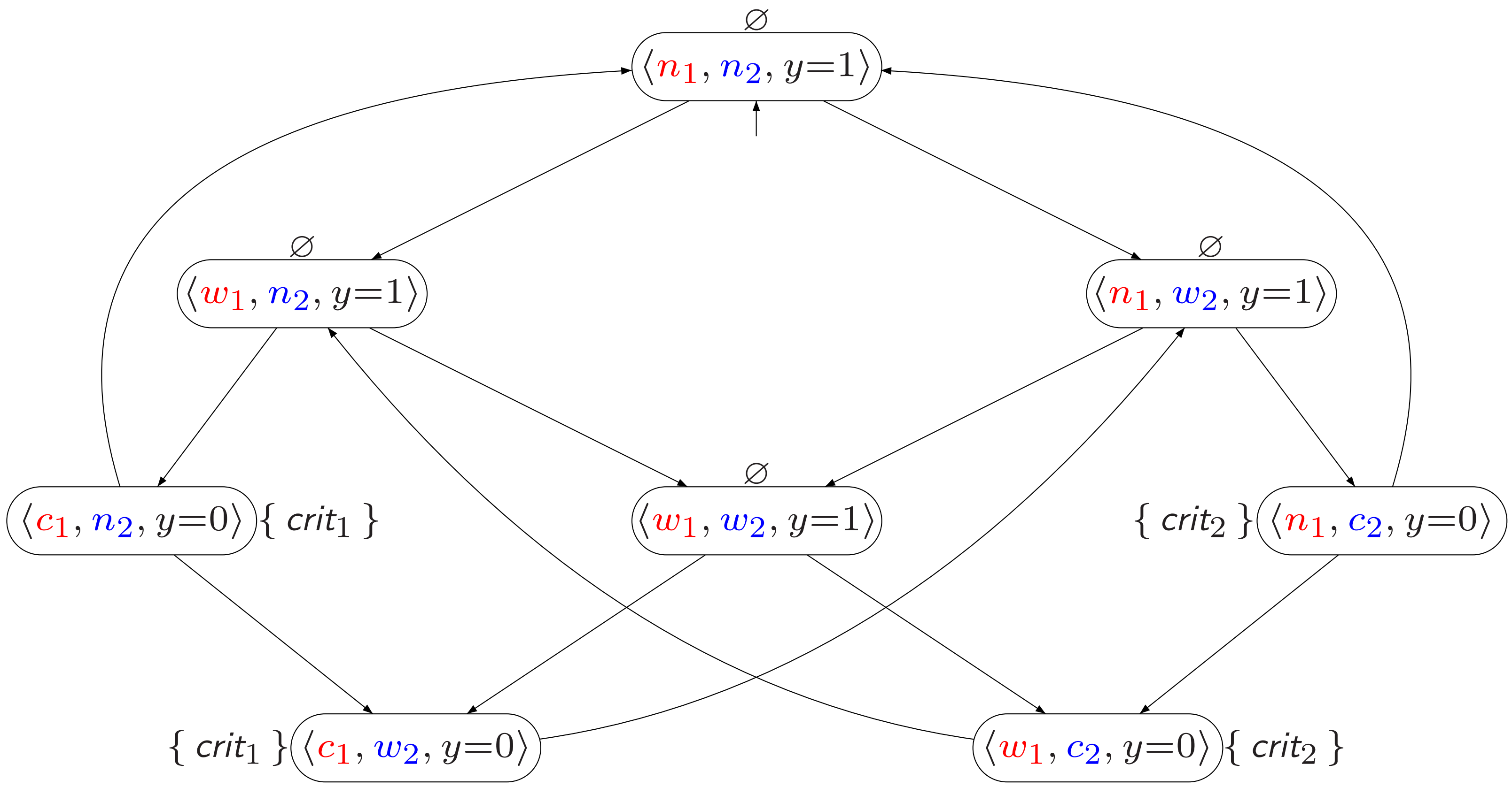
- Contained in P_{mutex} are e.g., the infinite words:

$$\{ \mathit{crit}_1 \} \{ \mathit{crit}_2 \} \{ \mathit{crit}_1 \} \{ \mathit{crit}_2 \} \{ \mathit{crit}_1 \} \{ \mathit{crit}_2 \} \ldots \quad \text{and}$$
$$\varnothing \, \varnothing \, \varnothing \, \varnothing \, \varnothing \, \varnothing \, \varnothing \ldots$$

  - this does not apply to words of the form: $\{ \mathit{crit}_1 \} \varnothing \{ \mathit{crit}_1, \mathit{crit}_2 \} \ldots$

Does the semaphore-based algorithm satisfy P_{mutex}?

Does the semaphore-based algorithm satisfy P_{mutex}?



Yes as there is no reachable state labeled with $\{ \mathit{crit}_1, \mathit{crit}_2 \}$

How to specify starvation freedom?

"A process that wants to enter the critical section is eventually able to do so'"

- Let $AP = \{ \mathit{wait}_1, \mathit{crit}_1, \mathit{wait}_2, \mathit{crit}_2 \}$
- Formalization as LT-property

$$P_{nostarve} = \text{ set of infinite words } A_0\, A_1\, A_2 \ldots \text{ such that:}$$

$$\Big(\overset{\infty}{\exists}\, j.\ \mathit{wait}_i \in A_j \Big) \ \Rightarrow\ \Big(\overset{\infty}{\exists}\, j.\ \mathit{crit}_i \in A_j \Big) \qquad \text{for each } i \in \{ 1, 2 \}$$

$\overset{\infty}{\exists}$ stands for "there are infinitely many".

Does the semaphore-based algorithm satisfy $P_{nostarve}$?

No. The trace

$$\varnothing \{ \mathit{wait}_2 \} \{ \mathit{wait}_1, \mathit{wait}_2 \} \{ \mathit{crit}_1, \mathit{wait}_2 \}$$
$$\{ \mathit{wait}_2 \} \{ \mathit{wait}_1, \mathit{wait}_2 \} \{ \mathit{crit}_1, \mathit{wait}_2 \} \ldots$$

is a possible trace of the transition system but not in $P_{nostarve}$

Trace equivalence and LT properties

For TS and TS' be transition systems (over AP):

$$\mathit{Traces}(TS) \subseteq \mathit{Traces}(TS')$$

if and only if

for any LT property P: $TS' \models P$ implies $TS \models P$

$$\mathit{Traces}(TS) = \mathit{Traces}(TS')$$

if and only if

TS and TS' satisfy the same LT properties

Invariants

- LT property P_{inv} over *AP* is an *invariant* if it has the form:

$$P_{inv} = \left\{ A_0 A_1 A_2 \ldots \in \left(2^{AP}\right)^{\omega} \mid \forall j \geqslant 0.\ A_j \models \Phi \right\}$$

  - where Φ is a propositional logic formula Φ over *AP*
  - Φ is called an *invariant condition* of P_{inv}

- Note that

$TS \models P_{inv}$	iff	$trace(\pi) \in P_{inv}$ for all paths π in *TS*
	iff	$L(s) \models \Phi$ for all states s that belong to a path of *TS*
	iff	$L(s) \models \Phi$ for all states $s \in Reach(TS)$

- Φ has to be fulfilled by all initial states and
  - satisfaction of Φ is invariant under all transitions in the reachable fragment of *TS*

algorithm

Algorithm 3 Naïve invariant checking by forward depth-first search

Input: finite transition system TS and propositional formula Φ
Output: true if TS satisfies the invariant "always Φ", otherwise false

set of state $R := \varnothing$; (* the set of visited states *)
stack of state $U := \varepsilon$; (* the empty stack *)
bool $b :=$ true; (* all states in R satisfy Φ *)
for all $s \in I$ **do**
    if $s \notin R$ **then**
        visit(s) (* perform a dfs for each unvisited initial state *)
    fi
od
return b

procedure visit (state s)
    $push(s, U)$; (* push s on the stack *)
    $R := R \cup \{ s \}$; (* mark s as reachable *)
    repeat
        $s' := top(U)$;
        if $Post(s') \subseteq R$ **then**
            $pop(U)$;
            $b := b \wedge (s' \models \Phi)$; (* check validity of Φ in s' *)
        else
            let $s'' \in Post(s') \setminus R$
            $push(s'', U)$;
            $R := R \cup \{ s'' \}$; (* state s'' is a new reachable state *)
        fi
    until $(U = \varepsilon)$
endproc

algorithm

Algorithm 4 Invariant checking by forward depth-first search

Input: finite transition system TS and propositional formula Φ
Output: "yes" if $TS \models$ "always Φ", otherwise "no" plus a counterexample

set of states $R := \varnothing$; (* the set of reachable states *)
stack of states $U := \varepsilon$; (* the empty stack *)
bool $b :=$ true; (* all states in R satisfy Φ *)
while $(I \setminus R \neq \varnothing \ \wedge \ b)$ **do**
  let $s \in I \setminus R$; (* choose an arbitrary initial state not in R *)
  visit(s); (* perform a DFS for each unvisited initial state *)
od
if b **then**
  return("yes") (* $TS \models$ "always Φ" *)
else
  return("no", reverse(U)) (* counterexample arises from the stack content *)
fi

procedure visit (state s)
  $push(s, U)$; (* push s on the stack *)
  $R := R \cup \{ s \}$; (* mark s as reachable *)
  repeat
    $s' := top(U)$;
    if $Post(s') \subseteq R$ **then**
      $pop(U)$;
      $b := b \ \wedge \ (s' \models \Phi)$; (* check validity of Φ in s' *)
    else
      let $s'' \in Post(s') \setminus R$
      $push(s'', U)$;
      $R := R \cup \{ s'' \}$; (* state s'' is a new reachable state *)
    fi
  until $((U = \varepsilon) \ \vee \ \neg b)$
endproc

Safety properties

- Safety properties may impose requirements on finite path fragments
  - and cannot be verified by considering the reachable states only

- A safety property which is not an invariant:
  - consider a cash dispenser, also known as automated teller machine (ATM)
  - property "money can only be withdrawn once a correct PIN has been provided"

  $\Rightarrow$ not an invariant, since it is not a state property

- But a safety property:
  - any infinite run violating the property has a finite prefix that is "bad"
  - i.e., in which money is withdrawn without issuing a PIN before

Safety properties

- LT property P_{safe} over *AP* is a *safety property* if
  - for all $\sigma \in \left(2^{AP}\right)^{\omega} \setminus P_{safe}$ there exists a finite prefix $\widehat{\sigma}$ of σ such that:

$$P_{safe} \cap \left\{ \sigma' \in \left(2^{AP}\right)^{\omega} \mid \widehat{\sigma} \text{ is a prefix of } \sigma' \right\} \;=\; \varnothing$$

- Path fragment $\widehat{\sigma}$ is a bad prefix of P_{safe}
  - let $\mathit{BadPref}(P_{safe})$ denote the set of bad prefixes of P_{safe}
- Path fragment $\widehat{\sigma}$ is a minimal bad prefix for P_{safe}:
  - if $\widehat{\sigma} \in \mathit{BadPref}(P_{safe})$ and no proper prefix of $\widehat{\sigma}$ is in $\mathit{BadPref}(P_{safe})$

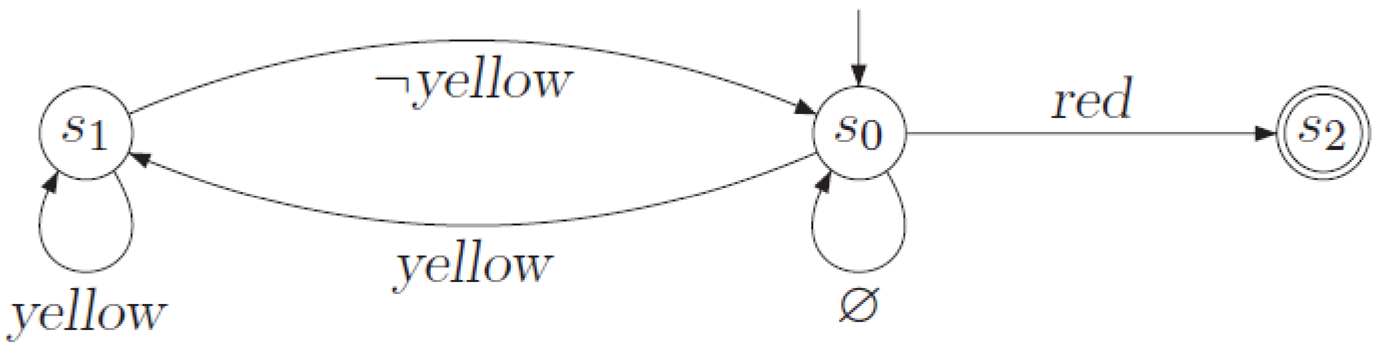
Example safety properties

- Property: a red phase must be preceded immediately by a yellow phase.

  infinite words $\sigma = A_0 A_1 \ldots$

  $red \in A_i$ implies $i > 0$ and $yellow \in A_{i-1}$.

- AP={red, yellow}
  - $\varnothing$ $\varnothing$\{red\}, $\varnothing$\{red\} minimal bad prefixes
  - \{yellow\} \{yellow\}\{red\}\{red\} $\varnothing$ \{red\}: bad prefix, but not minimal



Example safety properties（cont'）

- Property: The number of inserted coins is always at least the number of dispensed drinks

  infinite words $A_0\, A_1\, A_2 \ldots$

  $$|\{ 0 \leqslant j \leqslant i \mid pay \in A_j \}| \;\geqslant\; |\{ 0 \leqslant j \leqslant i \mid drink \in A_j \}|$$

- AP={pay, drink}
- ∅{pay}{drink}{drink}
- ∅{pay}{drink} ∅{pay }{drink}{drink}
- Bad prefixes

Safety properties and finite traces

For transition system *TS* without terminal states

and safety property P_{safe}:

$$\mathit{TS} \models P_{safe} \quad \text{if and only if} \quad \mathit{Traces}_{\mathit{fin}}(\mathit{TS}) \cap \mathit{BadPref}(P_{safe}) = \varnothing$$

Closure

- For trace $\sigma \in \left(2^{AP}\right)^{\omega}$, let $pref(\sigma)$ be the set of *finite prefixes* of σ:

$$pref(\sigma) = \{\, \widehat{\sigma} \in \left(2^{AP}\right)^{*} \mid \widehat{\sigma} \text{ is a finite prefix of } \sigma \,\}$$

  – if $\sigma = A_0 A_1 \ldots$ then $pref(\sigma) = \left\{ \varepsilon, A_0, A_0 A_1, A_0 A_1 A_2, \ldots \right\}$

- For property P we have: $pref(P) = \bigcup_{\sigma \in P} pref(\sigma)$

- The *closure* of LT property P:

$$closure(P) \;=\; \left\{\sigma \in \left(2^{AP}\right)^{\omega} \mid pref(\sigma) \;\subseteq\; pref(P)\right\}$$

  – the set of infinite traces whose finite prefixes are also prefixes of P, or
  – infinite traces in the closure of P do not have a prefix that is not a prefix of P

Safety properties and closures

For any LT property P over *AP*:

P is a safety property if and only if *closure*$(P) \;=\; P$

Why liveness?

- Safety properties specify that "something bad never happens"
- Doing nothing easily fulfills a safety property
  - as this will never lead to a "bad" situation

⇒ Safety properties are complemented by liveness properties
  - that require some progress

- Liveness properties assert that:
  - "something good" will happen eventually [Lamport 1977]

Liveness properties

LT property P_{live} over *AP* is a *liveness* property whenever

$$\mathit{pref}(P_{live}) = \left(2^{AP}\right)^*$$

- A liveness property is an LT property
  - that *does not rule out any prefix*
- Liveness properties are violated in "infinite time"
  - whereas safety properties are violated in finite time
  - finite traces are of no use to decide whether P holds or not
  - any finite prefix can be extended such that the resulting infinite trace satisfies P

Liveness properties for mutual exclusion

- Eventually:
  - each process will eventually enter its critical section
- Repeated eventually:
  - each process will enter ist critical section infinitely often
- Starvation freedom:
  - each waiting process will eventually enter its critical section

Safety vs. liveness

- Are safety and liveness properties disjoint? Yes
- Is any linear-time property a safety or liveness property? No
- But:

for any LT property P an equivalent LT property P' exists

which is a conjunction of a safety and a liveness property

A non-safety and non-liveness property

"the machine provides infinitely often beer
after initially providing sprite three times in a row"

- This property consists of *two* parts:
  - it requires beer to be provided infinitely often

  $\Rightarrow$ as any finite trace fulfills this, it is a liveness property
  - the first three drinks it provides should all be sprite

  $\Rightarrow$ bad prefix = one of first three drinks is beer; this is a safety property

- Property is thus a conjunction of a safety *and* a liveness property

does this apply to all such properties?

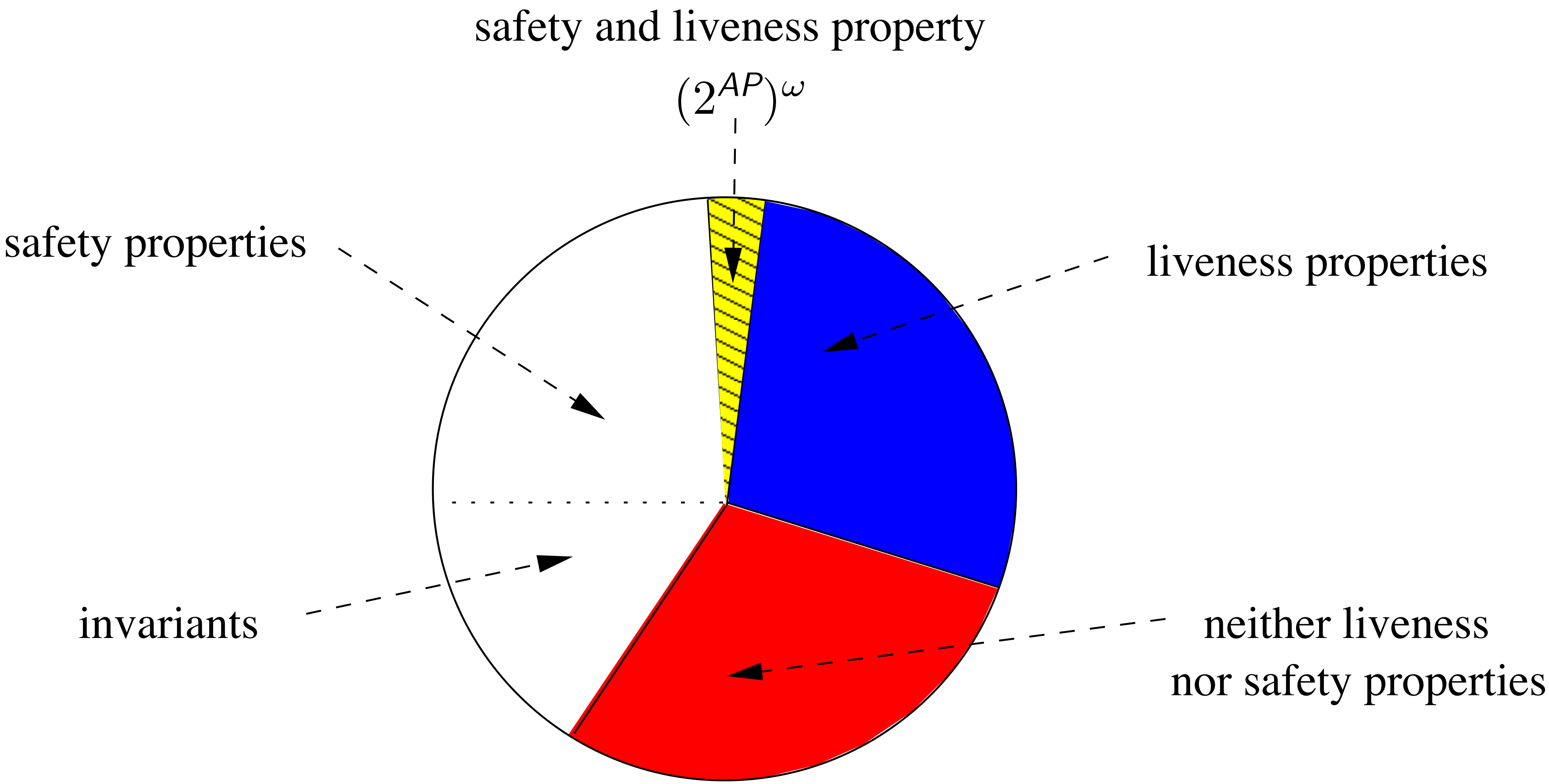
Decomposition theorem

For any LT property P over AP there exists
a safety property P_{safe} and a liveness property P_{live}
(both over AP) such that:

$$P = P_{safe} \cap P_{live}$$

Proposal: $P \; = \; \underbrace{\mathit{closure}(P)}_{=P_{safe}} \cap \underbrace{\left(P \cup \left(\left(2^{AP}\right)^{\omega} \setminus \mathit{closure}(P)\right)\right)}_{=P_{live}}$

Classification of LT properties



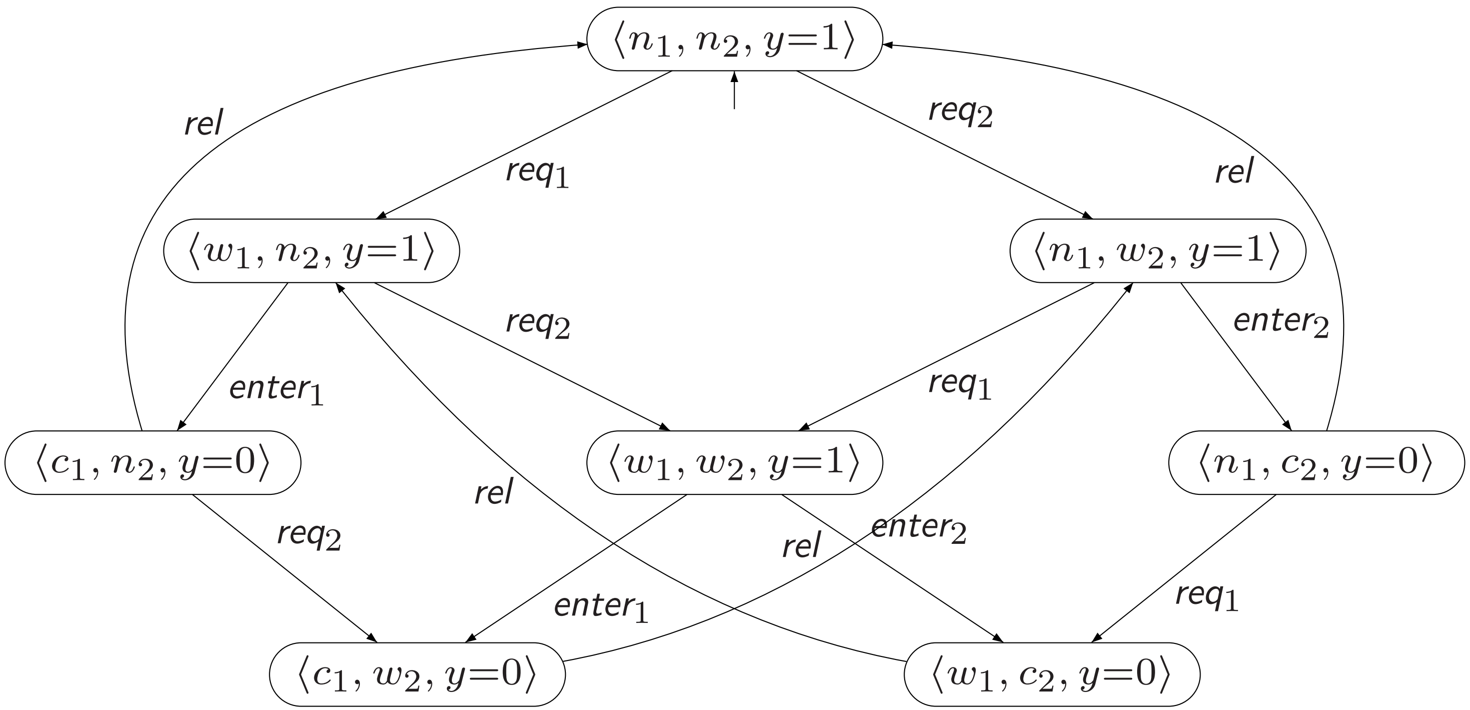
Does this program always terminate?

$$\text{Inc} \,|||\, \text{Reset}$$

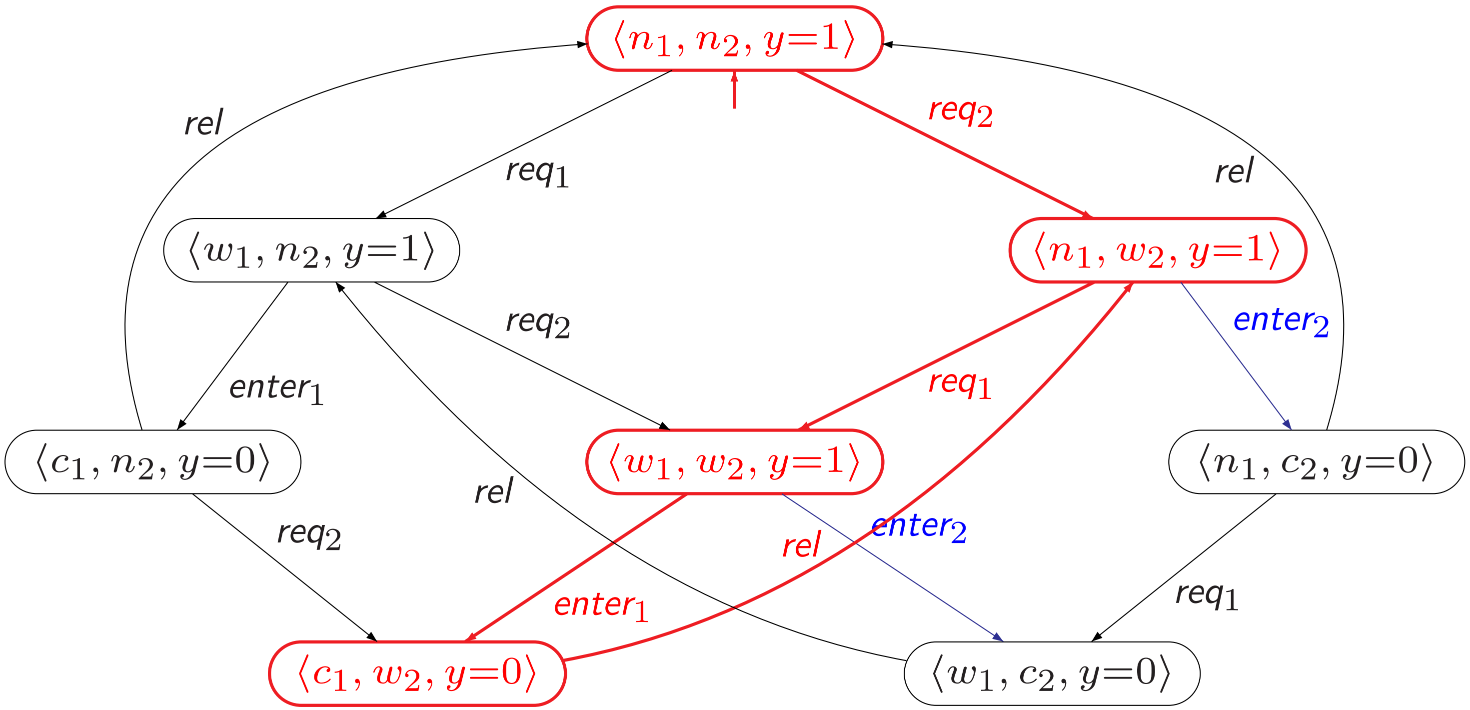
$$\begin{array}{rcl} & where & \\ \textbf{proc}\ \text{Inc} & = & \textbf{while}\ \langle\, x \geqslant 0\ \textbf{do}\ x := x + 1\,\rangle\ \textbf{od} \\ \textbf{proc}\ \text{Reset} & = & x := -1 \end{array}$$

x is a shared integer variable that initially has value 0

Is it possible to starve?

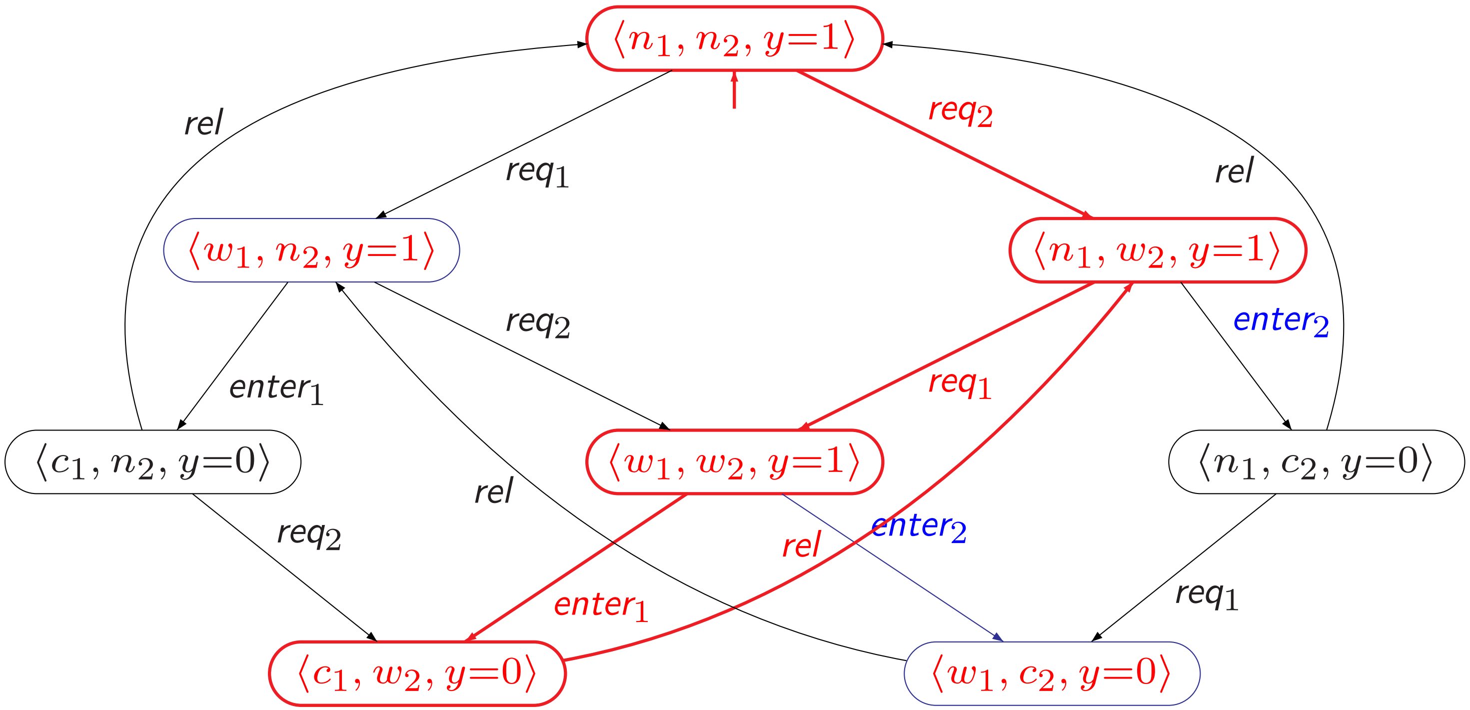


Process two starves



Is it fair that process two has infinitely many possibilities to enter the critical section, but never enters it?

Process two starves



Is it fair that process two has infinitely many possibilities
to enter the critical section, but only enters it finitely often?

Fairness

- Starvation freedom is often considered under process fairness

$\Rightarrow$ there is a fair scheduling of the execution of processes

- Fairness is typically needed to prove liveness
  - to prove some form of progress, progress needs to be possible
- Fairness is concerned with a fair resolution of nondeterminism
  - such that it is not biased to consistently ignore a possible option
- Problem: liveness properties constrain infinite behaviours
  - but some traces—that are unfair—refute the liveness property

Fairness constraints

- What is wrong with our examples? Nothing!
  - interleaving: not realistic as in no processor is infinitely faster than another
  - semaphore-based mutual exclusion: level of abstraction

- Rule out "unrealistic" executions by imposing *fairness constraints*
  - what to rule out? $\Rightarrow$ different kinds of fairness constraints

- "A process gets its turn infinitely often"
  - always *unconditional fairness*
  - if it is enabled infinitely often *strong fairness*
  - if it is continuously enabled from some point on *weak fairness*

Fairness

This program terminates under unconditional (process) fairness:

$$
\begin{aligned}
\textbf{proc}\ \mathsf{Inc} \quad &= \quad \textbf{while}\ \langle\, x \geqslant 0\ \textbf{do}\ x := x + 1\,\rangle\ \textbf{od} \\
\textbf{proc}\ \mathsf{Reset} \quad &= \quad x := -1
\end{aligned}
$$

x is a shared integer variable that initially has value 0

Fairness constraints

For $\mathit{TS} = (S, \mathit{Act}, \longrightarrow, I, \mathit{AP}, L)$ without terminal states, $A \subseteq \mathit{Act}$,

and infinite execution fragment $\rho = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots$ of TS:

1. ρ is *unconditionally A-fair* whenever: true $\implies \underbrace{\forall k \geqslant 0.\, \exists j \geqslant k.\, \alpha_j \in A}_{\text{infinitely often } A \text{ is taken}}$

2. ρ is *strongly A-fair* whenever:

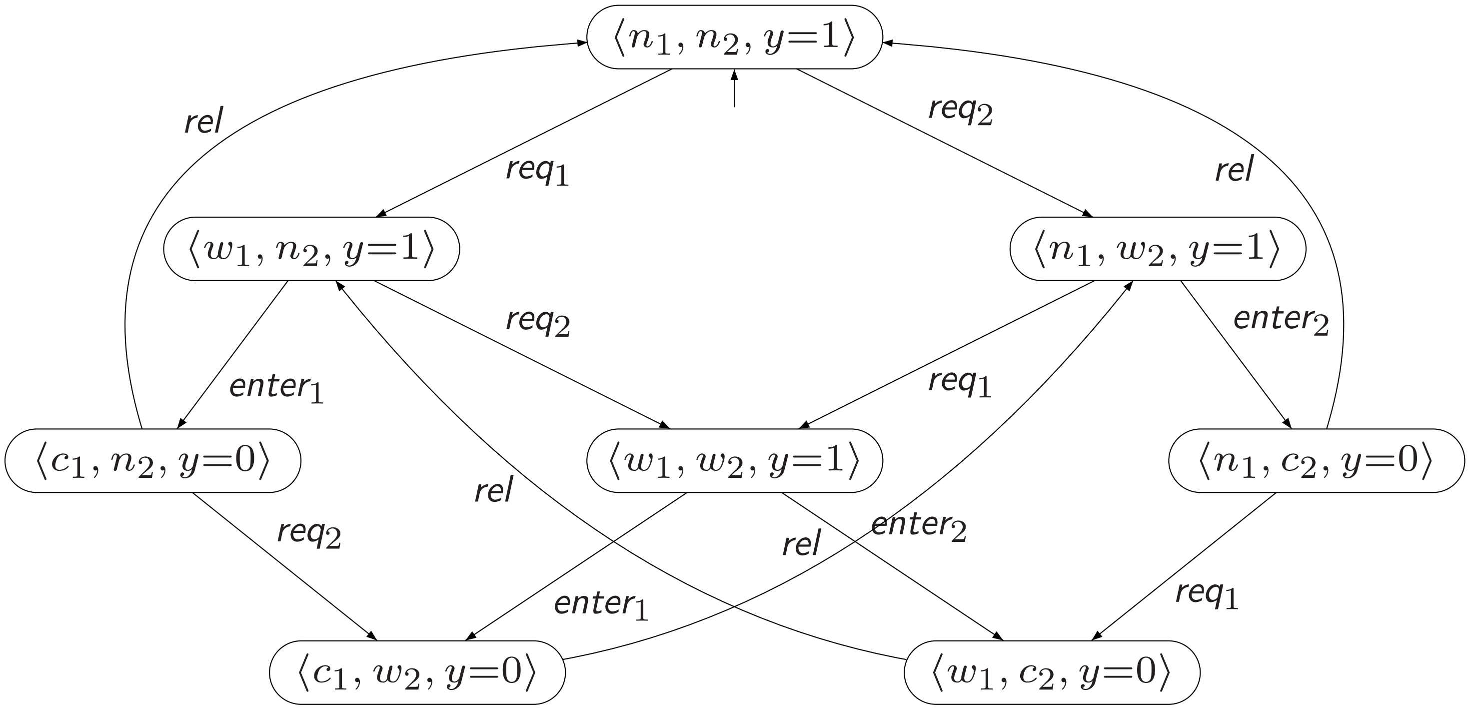
$$\underbrace{(\forall k \geqslant 0.\, \exists j \geqslant k.\, \mathit{Act}(s_j) \cap A \neq \varnothing)}_{\text{infinitely often } A \text{ is enabled}} \implies \underbrace{\forall k \geqslant 0.\, \exists j \geqslant k.\, \alpha_j \in A}_{\text{infinitely often } A \text{ is taken}}$$

3. ρ is *weakly A-fair* whenever:

$$\underbrace{(\exists k \geqslant 0.\, \forall j \geqslant k.\, \mathit{Act}(s_j) \cap A \neq \varnothing)}_{A \text{ is eventually always enabled}} \implies \underbrace{\forall k \geqslant 0.\, \exists j \geqslant k.\, \alpha_j \in A}_{\text{infinitely often } A \text{ is taken}}$$

where $\mathit{Act}(s) = \left\{ \alpha \in \mathit{Act} \mid \exists s' \in S.\, s \xrightarrow{\alpha} s' \right\}$

Example (un)fair executions



Which fairness notion to use?

- Fairness constraints aim to rule out "unreasonable" runs
- Too strong? $\Rightarrow$ relevant computations ruled out

  verification yields:
  - "false": error found
  - "true": don't know as some relevant execution may refute it
- Too weak? $\Rightarrow$ too many computations considered

  verification yields:
  - "true": property holds
  - "false": don't know, as refutation maybe due to some unreasonable run

often a combination of several fairness constraints is used

Fairness assumptions

- A *fairness assumption* for *Act* is a triple

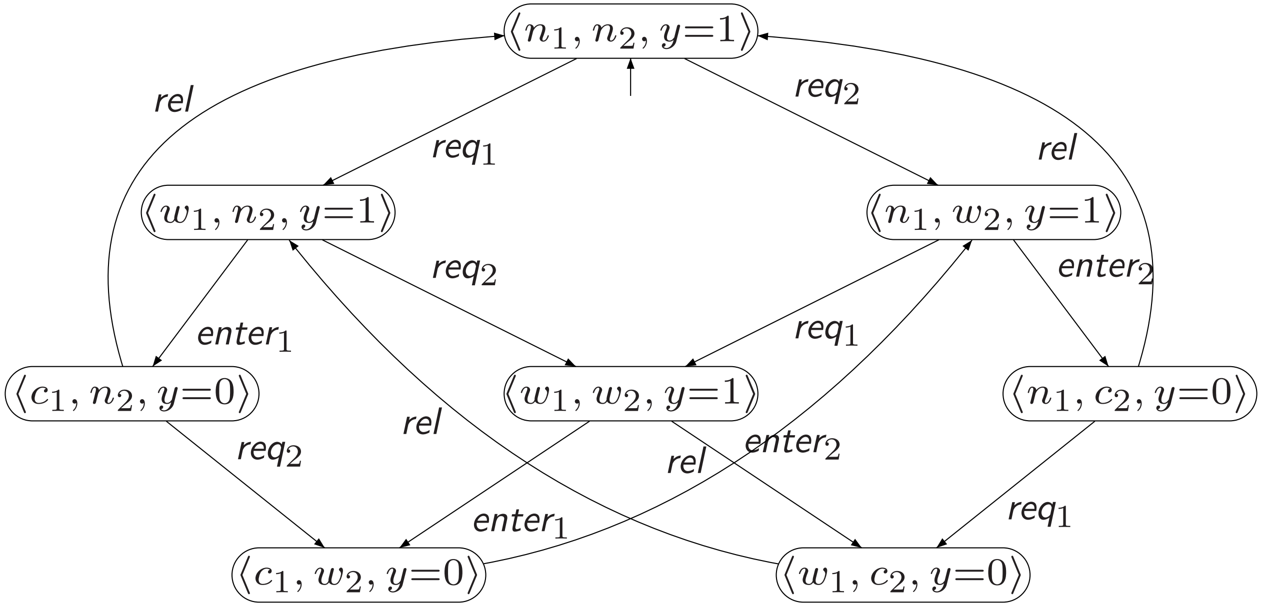
$$\mathcal{F} \; = \; (\mathcal{F}_{ucond}, \mathcal{F}_{strong}, \mathcal{F}_{weak})$$

  with $\mathcal{F}_{ucond}, \mathcal{F}_{strong}, \mathcal{F}_{weak} \subseteq 2^{Act}$

- Execution ρ is $\mathcal{F}$-fair if:
  - it is unconditionally A-fair for all $A \in \mathcal{F}_{ucond}$, and
  - it is strongly A-fair for all $A \in \mathcal{F}_{strong}$, and
  - it is weakly A-fair for all $A \in \mathcal{F}_{weak}$

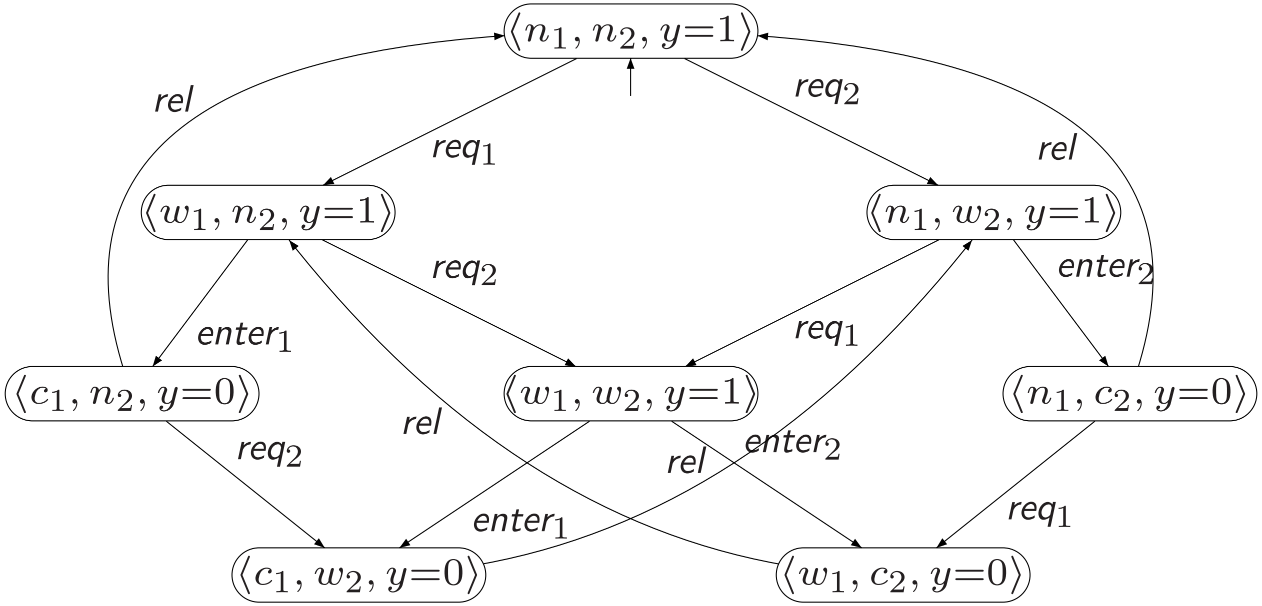
fairness assumption $(\varnothing, \mathcal{F}', \varnothing)$ denotes strong fairness; $(\varnothing, \varnothing, \mathcal{F}')$ weak, etc.

Fairness for mutual exclusion



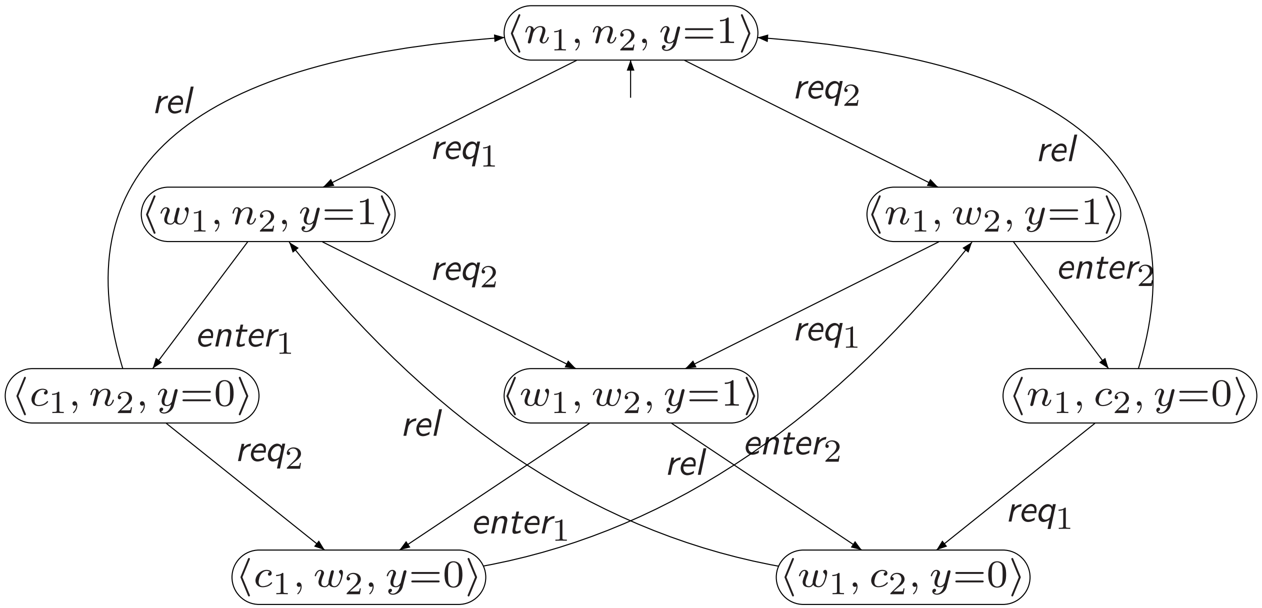
$$\mathcal{F} = (\varnothing, \underbrace{\Big\{ \{ \textit{enter}_1, \textit{enter}_2 \} \Big\}}_{\mathcal{F}_{strong}}, \varnothing)$$

Fairness for mutual exclusion



$$\mathcal{F}' = (\varnothing, \underbrace{\big\{ \{ \textit{enter}_1 \}, \{ \textit{enter}_2 \} \big\}}_{\mathcal{F}_{strong}}, \varnothing)$$

Fairness for mutual exclusion



$$\mathcal{F}'' = \Big(\varnothing, \underbrace{\Big\{ \{ \textit{enter}_1 \}, \{ \textit{enter}_2 \} \Big\}}_{\mathcal{F}_{strong}}, \underbrace{\Big\{ \{ \textit{req}_1 \}, \{ \textit{req}_2 \} \Big\}}_{\mathcal{F}_{weak}} \Big)$$

in any $\mathcal{F}''$-fair execution each process infinitely often requests access

Fair paths and traces

- Path $s_0 \longrightarrow s_1 \longrightarrow s_2 \ldots$ is *$\mathcal{F}$-fair* if
  - there exists an $\mathcal{F}$-fair execution $s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \ldots$
  - $\mathit{FairPaths}_{\mathcal{F}}(s)$ denotes the set of $\mathcal{F}$-fair paths that start in s
  - $\mathit{FairPaths}_{\mathcal{F}}(\mathit{TS}) = \bigcup_{s \in I} \mathit{FairPaths}_{\mathcal{F}}(s)$

- Trace σ is *$\mathcal{F}$-fair* if there exists an $\mathcal{F}$-fair path π with $\mathit{trace}(\pi) = \sigma$
  - $\mathit{FairTraces}_{\mathcal{F}}(s) = \mathit{trace}(\mathit{FairPaths}_{\mathcal{F}}(s))$
  - $\mathit{FairTraces}_{\mathcal{F}}(\mathit{TS}) = \mathit{trace}(\mathit{FairPaths}_{\mathcal{F}}(\mathit{TS}))$

Fair satisfaction

- *TS* *satisfies* LT-property P:

$$\mathit{TS} \models P \quad \text{if and only if} \quad \mathit{Traces}(\mathit{TS}) \subseteq P$$

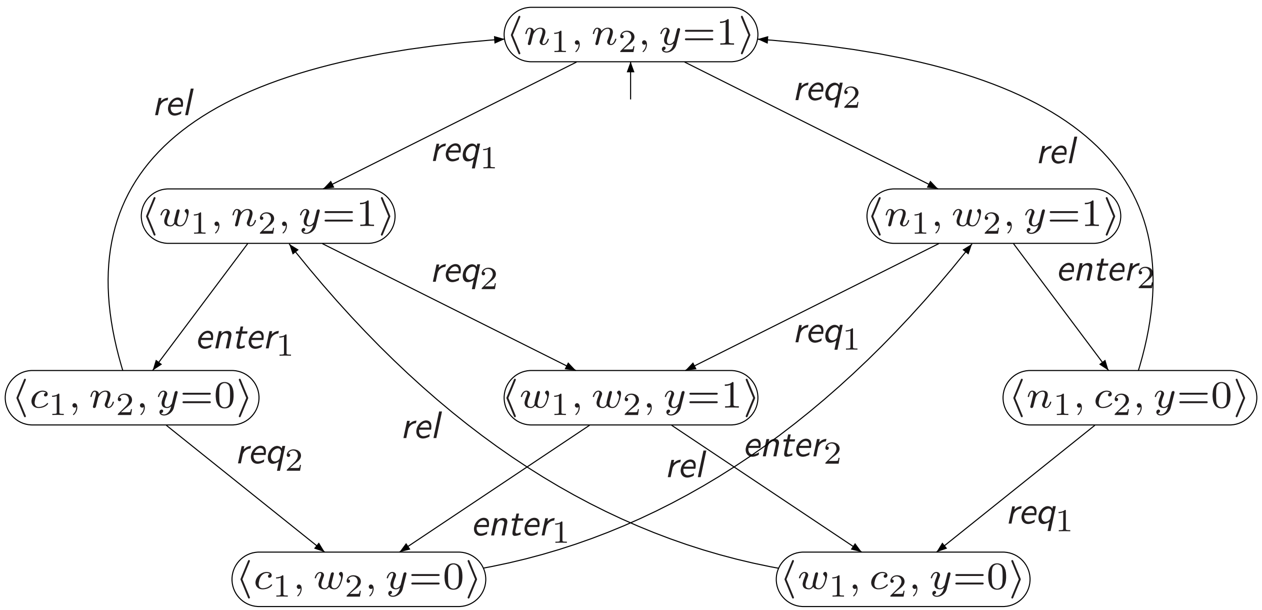
  – *TS* satisfies the LT property P if *all* its observable behaviors are admissible

- *TS* *fairly satisfies* LT-property P wrt. fairness assumption $\mathcal{F}$:

$$\mathit{TS} \models_{\mathcal{F}} P \quad \text{if and only if} \quad \mathit{FairTraces}_{\mathcal{F}}(\mathit{TS}) \subseteq P$$

  – if all paths in *TS* are $\mathcal{F}$-fair, then $\mathit{TS} \models_{\mathcal{F}} P$ if and only if $\mathit{TS} \models P$
  – if some path in *TS* is not $\mathcal{F}$-fair, then possibly $\mathit{TS} \models_{\mathcal{F}} P$ but $\mathit{TS} \not\models P$

Fairness for mutual exclusion



$\mathit{TS} \not\models$ "every process enters its critical section infinitely often"

and $\mathit{TS} \not\models_{\mathcal{F}'}$ "every . . . often"

but $\mathit{TS} \models_{\mathcal{F}''}$ "every . . . often"

Fairness and safety properties

For *TS* and safety property P_{safe} (both over *AP*)

suhc that for any $s \in \mathit{Reach}(\mathit{TS})$: $\mathit{FairPaths}_{\mathcal{F}}(s) \neq \varnothing$:

$$\mathit{TS} \models P_{safe} \quad \text{if and only if} \quad \mathit{TS} \models_{\mathcal{F}} P_{safe}$$

Safety properties are thus preserved by "realizable" fairness assumptions