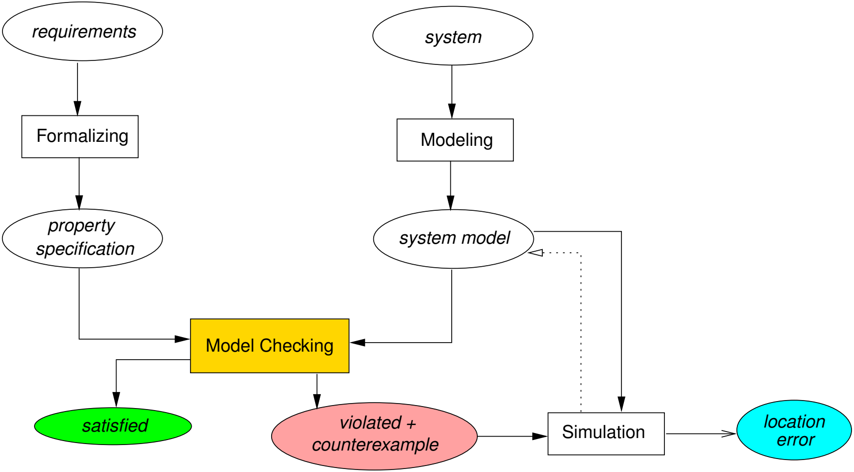
Linear-Time Properties

Hao Zheng

Department of Computer Science and Engineering
University of South Florida
Tampa, FL 33620
Email: zheng@cse.usf.edu
Phone: (813)974-4757
Fax: (813)974-5456

Recall Model Checking



We now consider the properties.

Overview

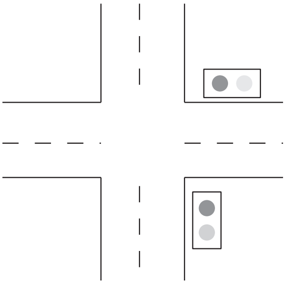
1. **Deadlock (Section 3.1)**
2. **Linear Time Behavior (Section 3.2)**
    - Executions, Paths, and Traces (Section 3.2.1 -3.2.2)
    - Linear-time Properties (Section 3.2.3 - 3.2.4)
3. **Safety and Invariants (Section 3.3.1 - 3.3.2)**
4. **Liveness Properties (Section 3.4.1)**
5. **Fairness (Section 3.5.1)**

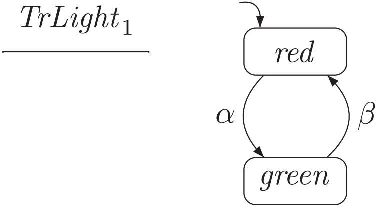
Contents

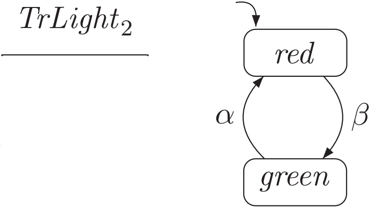
Deadlock

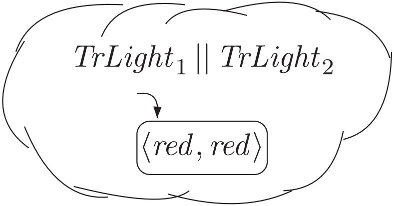
- Sequential programs without infinite loops terminate.
- For reactive systems, terminal states are undesirable and represent an error.
  - Embedded controllers need to operate without interruption for a long time.
- A *deadlock* occurs if a system stops while at least one component is in a (local) nonterminal state.
  - System has halted when at least one component should continue.
- Typically occurs when components mutually wait for each other.

Example Deadlock Situation

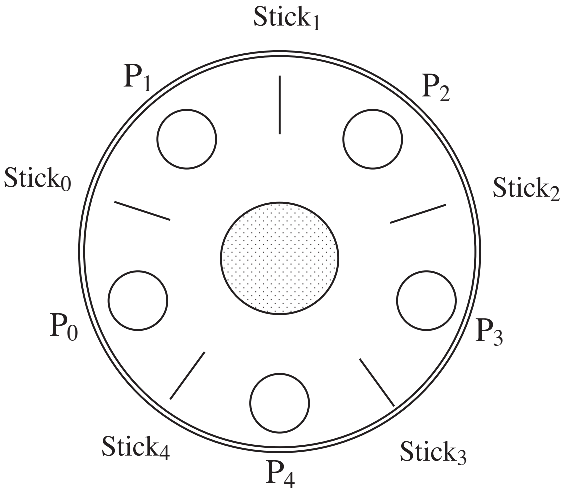








The Dining Philosophers



- Design a protocol which is deadlock-free.
- Design a protocol which is free of starvation.

A Transition System for the Dining Philosophers

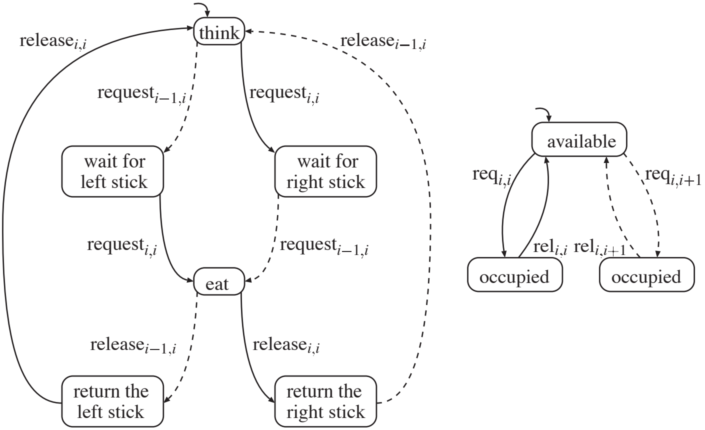


Figure 3.2: Transition systems for the ith philosopher and the ith stick.

$$\mathsf{Phil}_4 \parallel \mathsf{Stick}_3 \parallel \mathsf{Phil}_3 \parallel \mathsf{Stick}_2 \parallel \mathsf{Phil}_2 \parallel \mathsf{Stick}_1 \parallel \mathsf{Phil}_1 \parallel \mathsf{Stick}_0 \parallel \mathsf{Phil}_0 \parallel \mathsf{Stick}_4$$

A Transition System for the Dining Philosophers

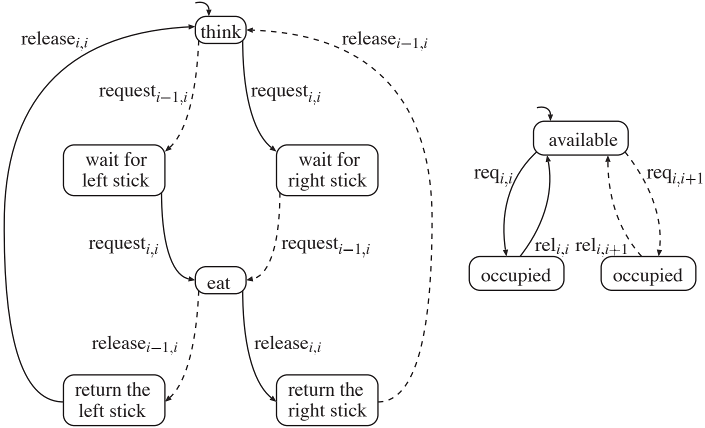


Figure 3.2: Transition systems for the ith philosopher and the ith stick.

$\text{Phil}_4 \parallel \text{Stick}_3 \parallel \text{Phil}_3 \parallel \text{Stick}_2 \parallel \text{Phil}_2 \parallel \text{Stick}_1 \parallel \text{Phil}_1 \parallel \text{Stick}_0 \parallel \text{Phil}_0 \parallel \text{Stick}_4$

$req_{4,4}, req_{3,3}, req_{2,2}, req_{1,1}, req_{0,0}$ leads to a deadlock.

Improved Transition System for the Stick

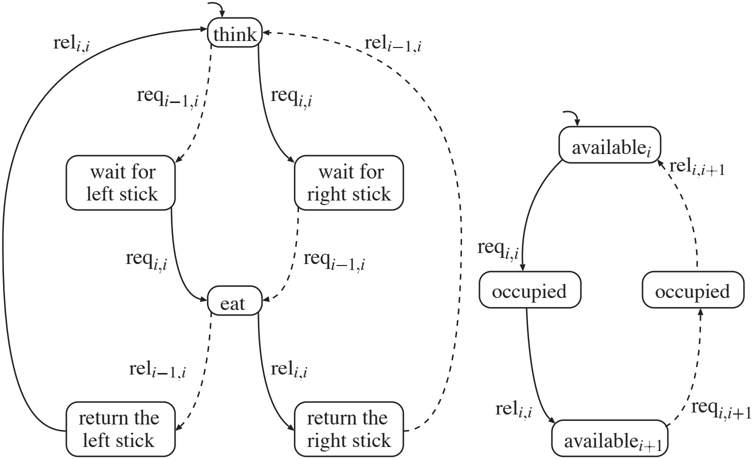


Figure 3.3: Improved variant of the ith philosopher and the ith stick.

- Even and odd numbered sticks start in different *available* states.

Contents

1. Deadlock (Section 3.1)
2. **Linear Time Behavior (Section 3.2)**
    - Executions, Paths, and Traces (Section 3.2.1 -3.2.2)
    - Linear-time Properties (Section 3.2.3 - 3.2.4)
3. Safety and Invariants (Section 3.3.1 - 3.3.2)
4. Liveness Properties (Section 3.4.1)
5. Fairness (Section 3.5.1)

Recall Executions

- A *finite execution fragment* ϱ of *TS* is an alternating sequence of states and actions ending with a state:

$$\varrho \ = \ s_0\,\alpha_1\,s_1\,\alpha_2\,\ldots\alpha_n\,s_n \text{ such that } s_i \xrightarrow{\alpha_{i+1}} s_{i+1} \text{ for all } 0 \leq i < n.$$

- An *infinite execution fragment* ρ of *TS* is an infinite, alternating sequence of states and actions:

$$\rho \ = \ s_0\,\alpha_1\,s_1\,\alpha_2\,s_2\,\alpha_3\ldots \text{ such that } s_i \xrightarrow{\alpha_{i+1}} s_{i+1} \text{ for all } 0 \leq i.$$

- An *execution* of *TS* is an *initial*, *maximal* execution fragment
  - An execution fragment is *initial* if $s_0 \in I$.
  - A maximal execution fragment can be finite, ending in a terminal state, or infinite.

Traces

- Let transition system $\mathit{TS} = (S, \mathit{Act}, \rightarrow, I, \mathit{AP}, L)$ without terminal states (i.e., all executions are infinite).
  - Terminal states are assumed to have self-loop transitions.
- The *trace* of execution $\rho \; = \; s_0\, \alpha_0\, s_1\, \alpha_1\, \ldots$ is

$$\mathit{trace}(\pi) = L(s_0)\, L(s_1) \ldots .$$

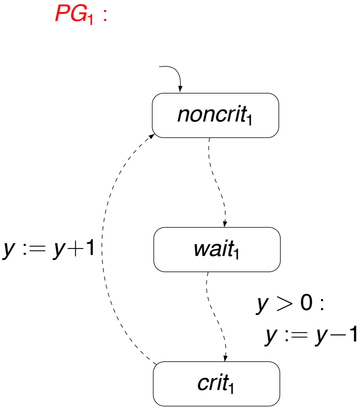
- The trace of $s_0\, \alpha_0\, s_1\, \alpha_1\, \ldots s_n$ is

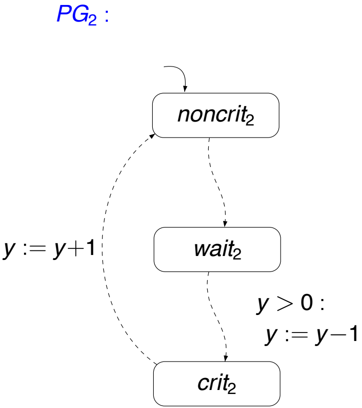
$$\mathit{trace}(\widehat{\pi}) = L(s_0)\, L(s_1) \ldots L(s_n).$$

- Traces of a TS are infinite words over the alphabet 2^{AP}, i.e.,

$$\mathit{Traces}(TS) \subseteq \left(2^{\mathit{AP}}\right)^{\omega}.$$

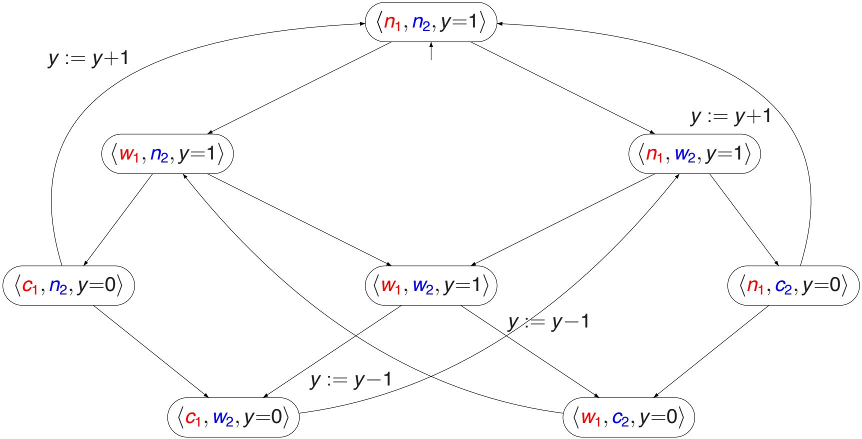
Semaphore-Based Mutual Exclusion



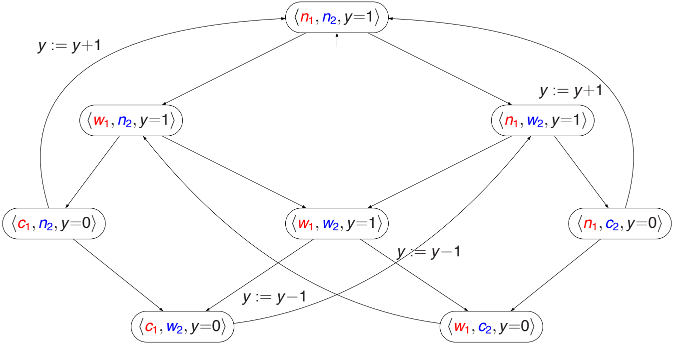


$y=0$ means "lock is currently possessed"; $y=1$ means "lock is free"

$TS(PG_1 \,|||\, PG_2)$


$\langle n_1, n_2, y{=}1\rangle$
$y := y{+}1$
$\langle w_1, n_2, y{=}1\rangle$
$y := y{+}1$
$\langle n_1, w_2, y{=}1\rangle$
$\langle c_1, n_2, y{=}0\rangle$
$\langle w_1, w_2, y{=}1\rangle$
$\langle n_1, c_2, y{=}0\rangle$
$y := y{-}1$
$y := y{-}1$
$\langle c_1, w_2, y{=}0\rangle$
$\langle w_1, c_2, y{=}0\rangle$

Example Traces

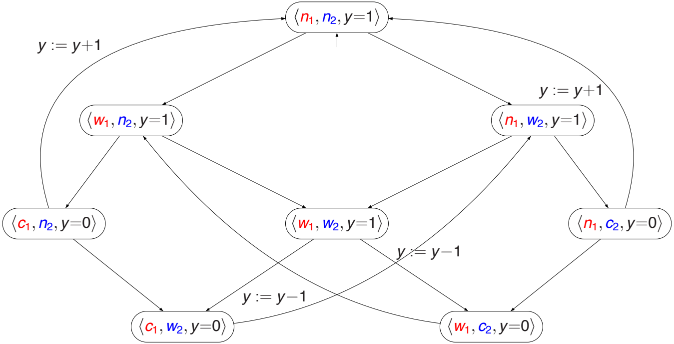


Let $AP = \{ \, crit_1, crit_2 \, \}$
The trace of the finite execution:

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

is: $trace(\widehat{\pi}) \; = \; \emptyset \, \emptyset \, \emptyset \, \{ \, crit_2 \, \} \, \emptyset \, \{ \, crit_1 \, \}$

Example Traces



Let $AP = \{ \mathit{crit}_1, \mathit{crit}_2 \}$
The trace of the infinite execution:

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

is: $\mathit{trace}(\pi) \; = \; \emptyset\,\emptyset\,\{ \mathit{crit}_1 \}\,\emptyset\,\emptyset\,\{ \mathit{crit}_2 \}\,\emptyset\,\emptyset\,\{ \mathit{crit}_1 \}\,\emptyset\,\emptyset\,\{ \mathit{crit}_2 \} \ldots$

Linear-Time Properties

- Linear-time properties specify the traces that a TS should only exhibit.

A *linear-time property* (LT property) P over *AP* is a subset of $\left(2^{AP}\right)^{\omega}$

- Finite words are not needed assuming there are *no terminal states*.
- A trace satisfies LT property P if it is included in P.

TS (over *AP*) *satisfies* LT property P (over *AP*):

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

- *TS* satisfies the LT property P if all its traces are admissible.

- Later, a logic will be introduced for specifying LT properties.

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 relevant for this property.
- Formalization as LT property:

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

- Which of the following infinite words satisfies P_{mutex}?
  - $(\{ \mathit{crit}_1 \} \{ \mathit{crit}_2 \})^\omega$
  - $\{ \mathit{crit}_1 \} \{ \mathit{crit}_1 \} \{ \mathit{crit}_1 \} \ldots$
  - $\emptyset \, \emptyset \, \emptyset \ldots$
  - $\{ \mathit{crit}_1 \} \emptyset \{ \mathit{crit}_1, \mathit{crit}_2 \} \ldots$
  - $\emptyset \{ \mathit{crit}_1 \} \emptyset \, \emptyset \{ \mathit{crit}_1, \mathit{crit}_2 \} \emptyset \ldots$

How to Specify Mutual Exclusion?

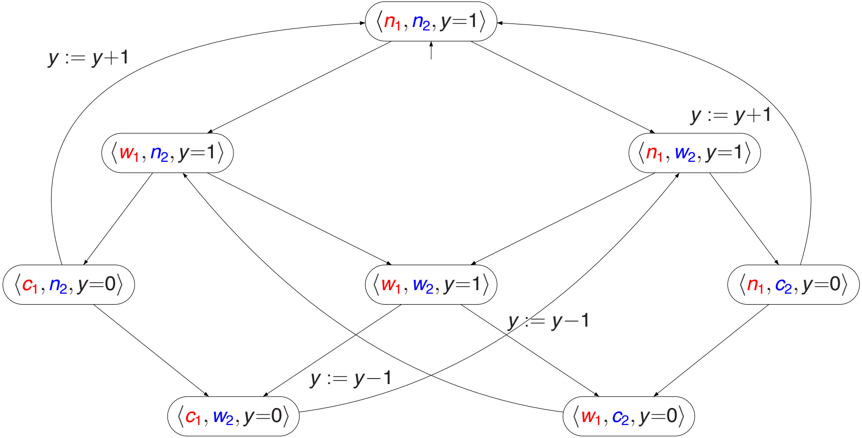
Always at most one process is in its critical section.

- Let $AP = \{ \mathit{crit}_1, \mathit{crit}_2 \}$
  - Other atomic propositions are not relevant for this property.
- Formalization as LT property:

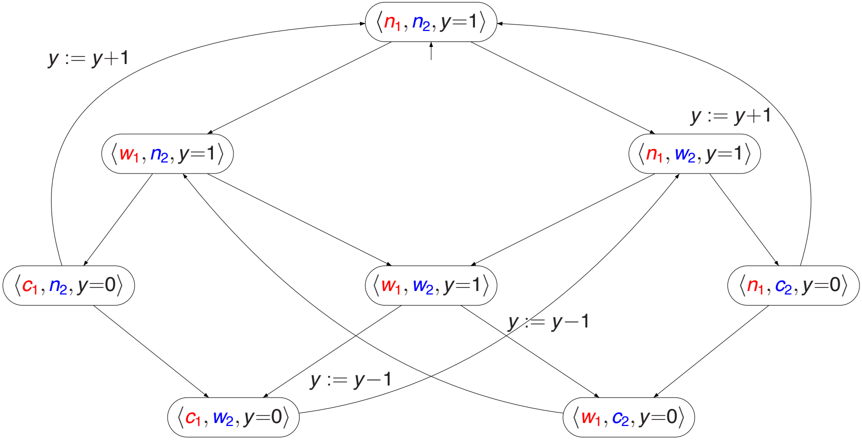
P_{mutex} = set of infinite words $A_0\, A_1\, A_2 \ldots$ with $\{ \mathit{crit}_1, \mathit{crit}_2 \} \not\subseteq A_i$ for all $0 \leq i$

- Which of the following infinite words satisfies P_{mutex}?
  - $(\{ \mathit{crit}_1 \} \{ \mathit{crit}_2 \})^\omega$ ✔
  - $\{ \mathit{crit}_1 \} \{ \mathit{crit}_1 \} \{ \mathit{crit}_1 \} \ldots$ ✔
  - $\emptyset\, \emptyset\, \emptyset \ldots$ ✔
  - $\{ \mathit{crit}_1 \}\, \emptyset\, \{ \mathit{crit}_1, \mathit{crit}_2 \} \ldots$ ✘
  - $\emptyset\, \{ \mathit{crit}_1 \} \emptyset\, \emptyset\, \{ \mathit{crit}_1, \mathit{crit}_2 \} \emptyset \ldots$ ✘

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.

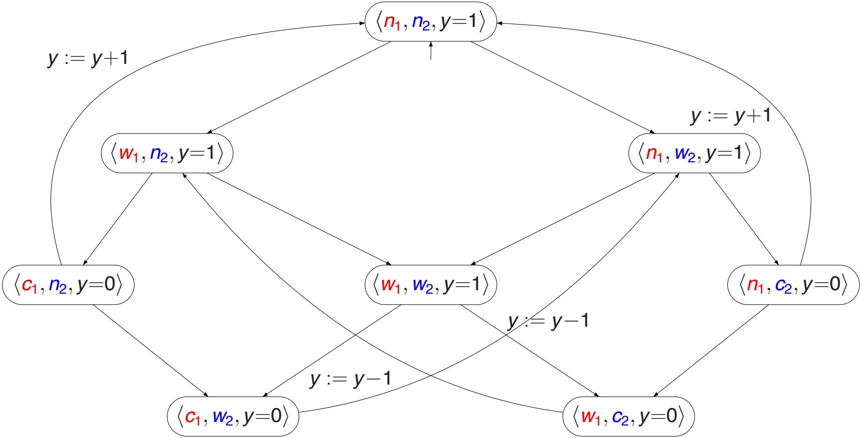
- Let $AP = \{ \mathit{wait}_1, \mathit{crit}_1, \mathit{wait}_2, \mathit{crit}_2 \}$
- Formalization #1:

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

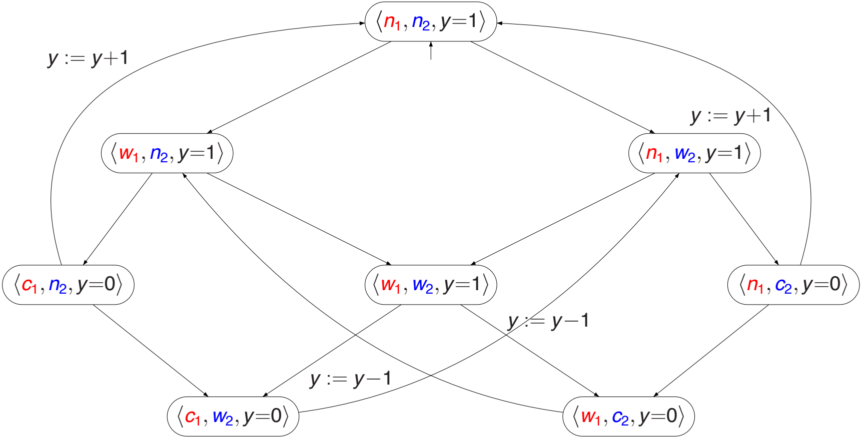
$$\forall j. \left(\mathit{wait}_i \in A_j \;\Rightarrow\; \exists k \geq j.\; \mathit{crit}_i \in A_k \right) \qquad \text{for each } i \in \{1, 2\}$$

  - However, it does not specify that a process should wait often.
  - This property holds if process i never wants to enter the critical section!

Does the Semaphore-Based Algorithm Satisfy $P_{nostarve}$?



Does the Semaphore-Based Algorithm Satisfy $P_{nostarve}$?



No. $\emptyset\ (\{\, wait_2 \,\}\,\{\, wait_1, wait_2 \,\}\,\{\, crit_1, wait_2 \,\})^{\omega} \in Traces(TS)$, but $\notin P_{nostarve}$.

3.2.4 Trace Inclusion and LT Properties

Theorem 3.15

Let *TS* and *TS′* be transition systems (over *AP*) without terminal states:

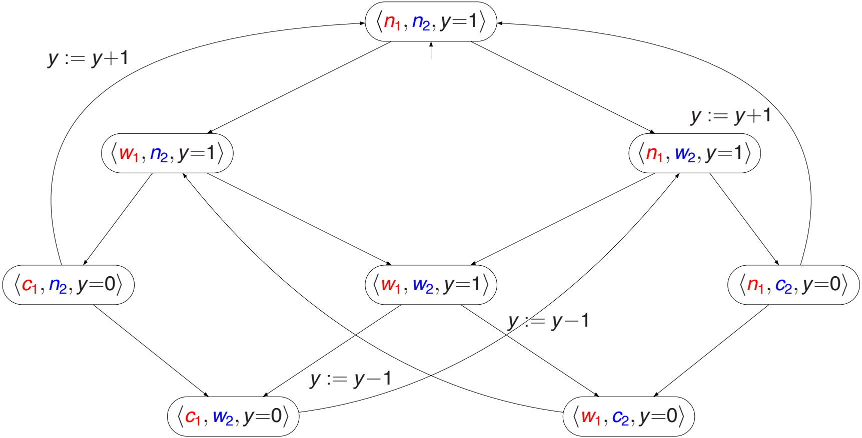
$$Traces(TS) \subseteq Traces(TS')$$

if and only if

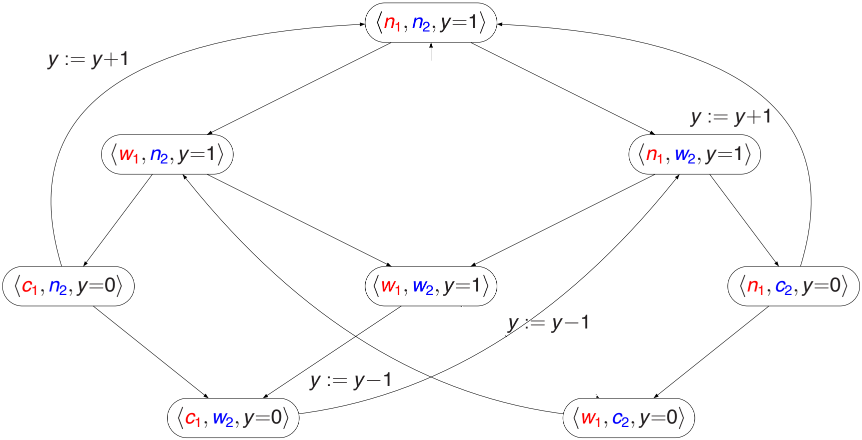
for any LT property P: $TS' \models P$ implies $TS \models P$

- $Traces(TS) \subseteq Traces(TS')$ means that *TS* is an implementation of *TS′*
  - *TS* is also referred to as *refinement* of *TS′*.

Mutual Exclusion Algorithm Revisited TS'



Mutual Exclusion Algorithm Revisited TS



This algorithm satisfies P_{mutex} as
$Traces(TS) \subset Traces(TS')$ and $TS' \models P_{mutex}$.

Corollary 3.18 Trace Equivalence and LT Properties

Let TS and TS' be transition systems (over AP) without terminal states:

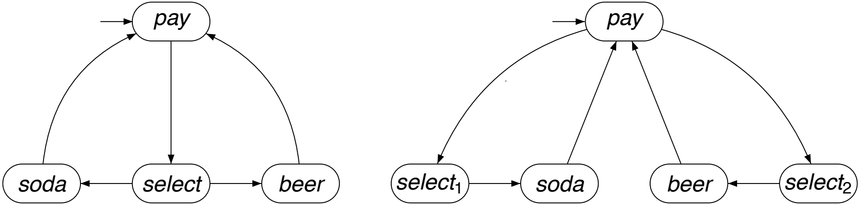
$$Traces(TS) = Traces(TS')$$

if and only if

TS and TS' satisfy the same LT properties

- TS and TS' cannot be distinguished by any LT properties.

Two Beverage Vending Machines



$$AP = \{ \, pay, soda, beer \, \}$$

There is no LT-property that can distinguish between these machines.

Contents

1. **Deadlock (Section 3.1)**
2. **Linear Time Behavior (Section 3.2)**
    - Executions, Paths, and Traces (Section 3.2.1 -3.2.2)
    - Linear-time Properties (Section 3.2.3 - 3.2.4)
3. **Safety and Invariants (Section 3.3.1 - 3.3.2)**
4. **Liveness Properties (Section 3.4.1)**
5. **Fairness (Section 3.5.1)**

3.3 Safety and Invariants

- Safety properties $\approx$ "nothing bad should happen". [Lamport 1977]
- Typical safety property: mutual exclusion property.
  - The bad thing (having > 1 process in the critical section) never occurs.
- Another typical safety property is deadlock freedom.

$\Rightarrow$ These properties are in fact invariants.

- An invariant is an LT property that is given by a condition Φ for the states and requires that Φ holds for all reachable states (e.g., for mutex property $\Phi \equiv \neg crit_1 \ \vee \ \neg crit_2$).

Read section 3.3.1. Skim over 3.3.2. Ignore 3.3.3.

3.3.1 Invariants

Definition 3.20 Invariant

An LT property P_{inv} over *AP* is an *invariant* if there is a propositional logic formula Φ over *AP* such that:

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

where Φ is called an *invariant condition* of P_{inv}.

Example Invariants

- Mutual exclusion

$$\Phi \quad = \quad \neg crit_1 \vee \neg crit_2$$

- Deadlock freedom in Dining Philosophers

$$\Phi \quad = \quad \neg wait_0 \vee \ldots \vee \neg wait_4$$

Deadlock is avoided if at least one philosopher is not waiting to pick up sticks.

Notes on Invariants

$$\begin{array}{lll} \mathit{TS} \models P_{inv} & \text{iff} & \mathit{trace}(\pi) \in P_{inv} \text{ for all paths } \pi \text{ in } \mathit{TS} \\ & \text{iff} & L(s) \models \Phi \text{ for all states } s \text{ that belong to a path of } \mathit{TS} \\ & \text{iff} & L(s) \models \Phi \text{ for all states } s \in \mathit{Reach}(\mathit{TS}) \end{array}$$

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

Checking an Invariant

- Checking an invariant for the propositional formula Φ
  - $=$ Check the validity of Φ in every reachable state.
  - $\Rightarrow$ Use a slight modification of standard graph traversal algorithms (i.e., *depth-first search* (DFS) and *breadth-first search* (BFS)).
  - Provided that the given transition system *TS* is *finite*.
- Perform a forward depth-first search:
  - If any state s is found with $s \not\models \Phi \Rightarrow$ the invariance of Φ is violated.
- Alternative is to perform a backward search:
  - Starts with all states where Φ does not hold.
  - Calculates (by a DFS or BFS) the set $\bigcup_{s \in S, s \not\models \Phi} \mathit{Pre}^*(s)$.
  - If there is a $init \in I$ such that $init \in \bigcup_{s \in S, s \not\models \Phi} \mathit{Pre}^*(s)$, then Φ is violated.

Algorithm 3: A Naive Invariant Checking Algorithm

```
$R = \emptyset$;                              // Set of reachable states
$U = \emptyset$;                              // Stack of states
bool $b := true$;                             // All states in $R$ satisfies $\Phi$
foreach $s \in I$ do
    if $s \notin R$ then
        $visit(s)$;                           // $visit(s)$ shown on the next slide
return $b$;
```

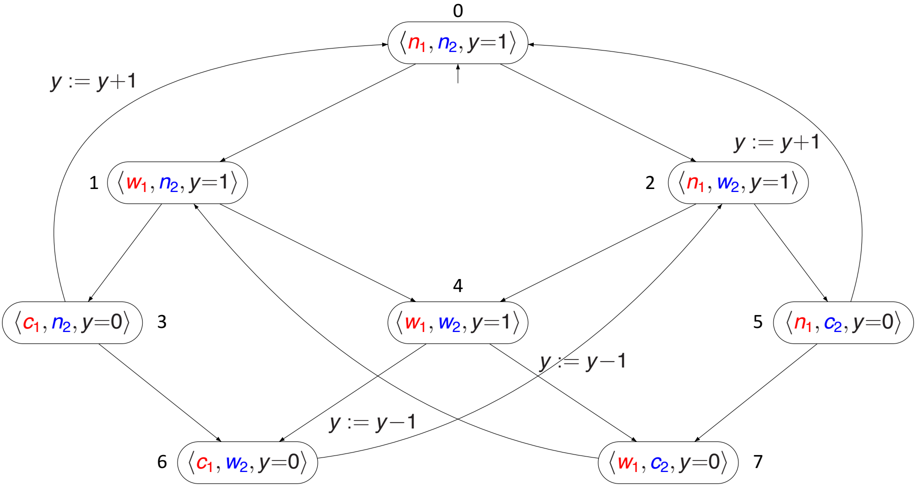
The $visit(s)$ Procedure

```
$push(s, U)$;                                    // Set of reachable states
$R := R \cup \{s\}$;                             // Stack of states
bool $b := true$;                                // All states in $R$ satisfies $\Phi$
while $U \neq \emptyset$ do
    $s' = top(U)$;
    if $Post(s') \subseteq R$ then
        $pop(U)$;
        $b := b \wedge (s' \models \Phi)$;
    else
        Let $s'' \in Post(s') \backslash R$;
        $push(s'', U)$;
        $R := R \cup \{s''\}$;
return b;
```

Error indication is state that refutes Φ.

$s_0\, s_1 \ldots s_n$ *with* $s_i \models \Phi$ *($i \neq n$) and* $s_n \not\models \Phi$ *is a counter-example.*

DFS Illustration



Time Complexity

- The time complexity for invariant checking is:

$$\mathcal{O}(\ N * (1 + |\Phi|) + M\ )$$

where

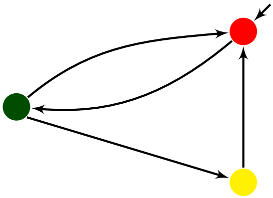
- N is the number of reachable states,
- M is the number of transitions in the reachable fragment of *TS*, and
- $|\Phi|$ is the length of Φ.

3.3.2 Safety Properties

- Safety properties can be translated to invariants.
  - Safety properties need to hold in every state!
- A *TS* violates a LT safety property P_{safe} if there is a $BadPref$ such that

$$BadPref \bullet (2^{AP})^{\omega} \notin P_{safe}.$$

  where $BadPref$ is a finite trace that ends with a violation to P_{safe}.
- The idea is different from what is given in the book.



P_{safe}: *red preceded immediately by yellow.*

Counter-example for P_{safe}: 🔴🟢🔴

Contents

Liveness Properties

- 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.
  - Safety violations are characterized by finite traces.
- Liveness properties assert that ”something good will happen”.
  - Liveness violations are characterized by infinite traces.

Example Liveness Properties

- “If the tank is empty, the outlet valve will eventually be closed”.
- “If the outlet valve is open and the request signal disappears, the outlet valve will eventually be closed”.
- “If the tank is full and a request is present, the outlet valve will eventually be opened”.
- “The program terminates within 31 computational steps”.

  ⇒ A finite trace may violate this; this is a safety property!
- “The program eventually terminates”.

Liveness Properties for Mutual Exclusion

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

How to formalize these properties?

Liveness Properties for Mutual Exclusion

$P = \{ A_0\, A_1\, A_2 \ldots \mid A_j \subseteq AP \;\;\&\;\; \ldots \}$ and $AP = \{wait_1, crit_1, wait_2, crit_2\}$

- Eventually:

  Each process will eventually enter its critical section.

$$(\exists j \geq 0.\ crit_1 \in A_j) \;\wedge\; (\exists j \geq 0.\ crit_2 \in A_j)$$

- Repeated eventually:

  Each process will enter its critical section infinitely often.

$$\left(\overset{\infty}{\exists}\ j \geq 0.\ crit_1 \in A_j\right) \;\wedge\; \left(\overset{\infty}{\exists}\ j \geq 0.\ crit_2 \in A_j\right)$$

- Starvation freedom:

  Each waiting process will eventually enter its critical section.

$$\begin{aligned} &\forall j \geq 0.\ (wait_1 \in A_j \;\Rightarrow\; (\exists k > j.\ crit_1 \in A_k)) \;\;\wedge \\ &\forall j \geq 0.\ (wait_2 \in A_j \;\Rightarrow\; (\exists k > j.\ crit_2 \in A_k)) \end{aligned}$$

Summary LT Properties

- LT properties are finite sets of infinite words over 2^{AP} ($=$ traces).
- An invariant requires a condition Φ to hold in any reachable state.
- Each trace refuting a safety property has a finite prefix causing this.
  - Invariants are safety properties with bad prefix $\Phi^*(\neg\Phi)$.
  - A safety property is regular iff its set of bad prefixes is a regular language.

  $\Rightarrow$ Safety properties constrain finite behaviors.
- A liveness property does not rule out finite behavior, liveness properties constrain infinite behaviors.

Contents

1. **Deadlock (Section 3.1)**
2. **Linear Time Behavior (Section 3.2)**
   - Executions, Paths, and Traces (Section 3.2.1 -3.2.2)
   - Linear-time Properties (Section 3.2.3 - 3.2.4)
3. **Safety and Invariants (Section 3.3.1 - 3.3.2)**
4. **Liveness Properties (Section 3.4.1)**
5. **Fairness (Section 3.5.1)**

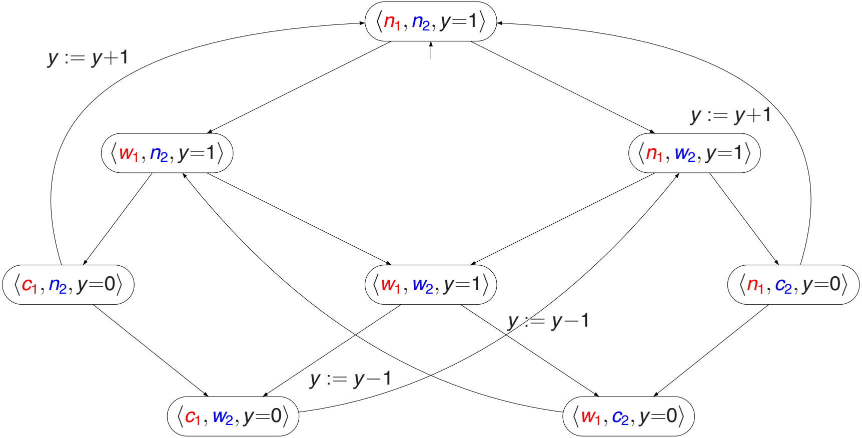
Does this Program Always Terminate?

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

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

x is a shared integer variable that initially has value 0

Is it Possible to Starve?



Can either process enter its critical section if it wants to?

Fairness

- Starvation freedom is often considered under process fairness.
  - ⇒ There is a fair scheduling of the execution of processes.
- Fairness is typically needed to prove liveness.
  - Not needed for safety properties!
  - To prove liveness or 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 behaviors, but some traces—that are unfair—refute the liveness property.

3.5.1 Fairness Constraints

- *Unconditional fairness*

  An activity is executed infinitely often.
- *Strong fairness (compassion)*

  If an activity is *infinitely often* enabled (not necessarily always!) then it has to be executed infinitely often.
- *Weak fairness (justice)*

  If an activity is *continuously enabled* (no temporary disabling!) then it has to be executed infinitely often.

We will use actions to distinguish fair and unfair behaviors.
A state-based notion of fairness could also be defined.

Fairness Definition

For $\mathit{TS} = (S, \mathit{Act}, \rightarrow, I, \mathit{AP}, L)$ 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*:

❶ ρ is *unconditionally A-fair* whenever:

$$\text{true} \implies \underbrace{\forall k \geq 0.\, \exists j \geq k.\ \alpha_j \in A}_{\text{infinitely often } A \text{ is taken}}$$

❷ ρ is *strongly A-fair* whenever:

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

❸ ρ is *weakly A-fair* whenever:

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

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

Fairness Example

This program terminates under unconditional (process) fairness on process IDs:

$$\begin{aligned} \textbf{proc } \mathsf{Inc} &= \textbf{while } x \geq 0 \textbf{ do } x := x + 1 \textbf{ od} \\ \textbf{proc } \mathsf{Reset} &= x := -1 \end{aligned}$$

x is a shared integer variable that initially has value 0

Another Fairness Example

Does the following property holds?

x eventually becomes negative.

$$\begin{array}{rcl}\textbf{proc}\ \text{Inc} & = & \textbf{while}\ x \geq 0\ \textbf{do}\ \alpha_1 : x := x + 1\ \textbf{od} \\ \textbf{proc}\ \text{Reset} & = & \textbf{while}\ x < 0\ \textbf{do}\ \alpha_2 : x := -1\ \textbf{od}\end{array}$$

x is a shared integer variable that initially has value 0

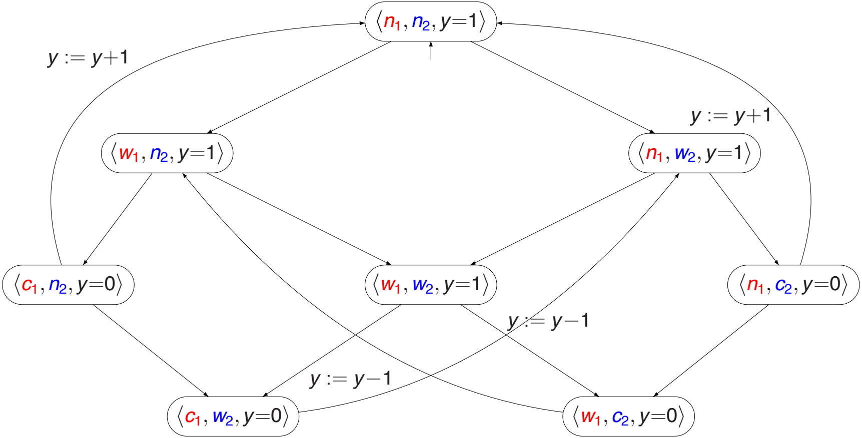
1. Using unconditional fairness on α_1 and α_2.
2. Using strong fairness on α_1 and α_2.

Which Fairness Notion to Use?

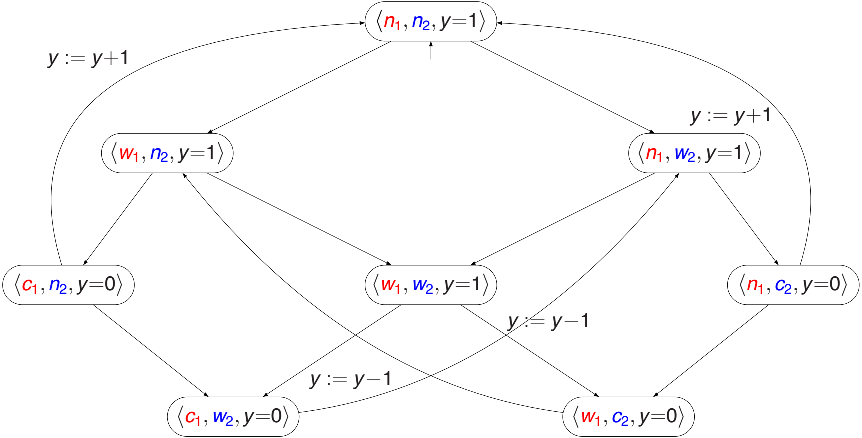
unconditional A-fairness $\Longrightarrow$ strong A-fairness $\Longrightarrow$ weak A-fairness

- Fairness constraints aim to rule out "unreasonable" executions.
- 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 may be due to some unreasonable run.

Example (Un)fair Executions

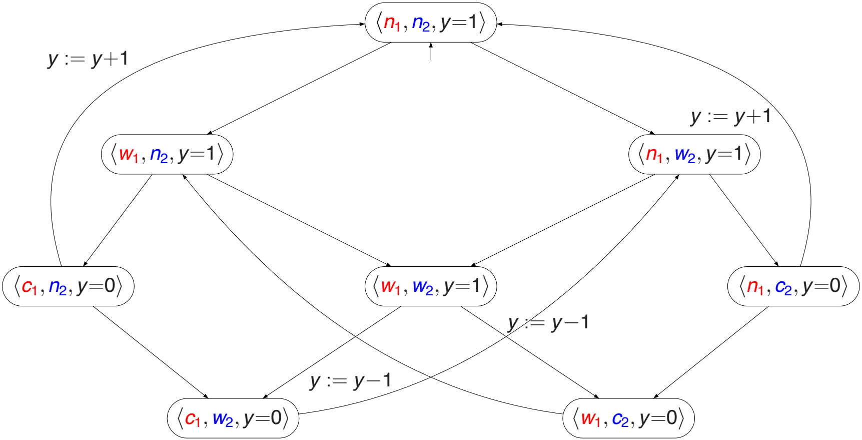


Fairness for Mutual Exclusion



$$\mathcal{F} = (\underbrace{\{enter_1, enter_2\}}_{\mathcal{F}_{uncondirional}}, \emptyset, \emptyset)$$

Fairness for Mutual Exclusion



$$\mathcal{F} = (\emptyset, \underbrace{\{enter_1, enter_2\}}_{\mathcal{F}_{strong}}, \emptyset)$$