# Transition Systems and Linear-Time Properties

## Lecture #1 of Principles of Model Checking

*Joost-Pieter Katoen*

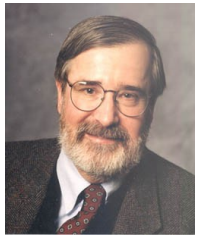
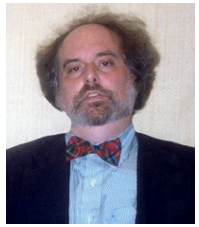
Software Modeling and Verification Group

affiliated to University of Twente, Formal Methods and Tools

University of Twente, September 4, 2012

# Model checking

- Automated model-based verification and debugging technique
  - model of system = Kripke structure $\approx$ labeled transition system
  - properties expressed in temporal logic like LTL or CTL
  - provides counterexamples in case of property refutation
- Various striking examples
  - Needham-Schroeder security protocol, storm surge barrier, C code
- 2008: Pioneers awarded prestigious ACM Turing Award

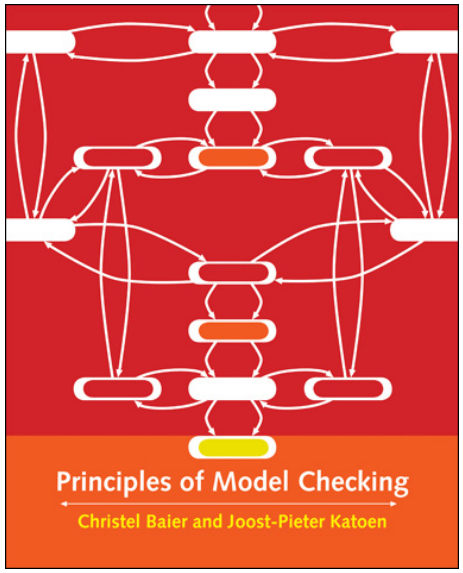

# Course topics

- Transition systems and linear-time properties
  - traces, safety, liveness, fairness
- Verifying regular linear-time properties
  - omega-regular languages, Büchi automata, nested depth-first search
- LTL model checking
  - syntax, semantics, automata, model-checking algorithm
- CTL model checking
  - syntax, semantics, CTL versus LTL, model-checking algorithm

# Course topics

- Abstraction
  - stutter (bi)simulation, LTL/CTL equivalence, minimisation algorithms
- Partial-order reduction
  - independence, ample sets, dynamic POR
- Probabilistic model checking
  - Markov chains, reachability probabilities

# Principles of Model Checking




Christel Baier

TU Dresden, Germany

Joost-Pieter Katoen

RWTH Aachen University, Germany,
and
University of Twente, the Netherlands

*"This book offers one of the most comprehensive introductions to logic model checking techniques available today. The authors have found a way to explain both basic concepts and foundational theory thoroughly and in crystal clear prose. Highly recommended for anyone who wants to learn about this important new field, or brush up on their knowledge of the current state of the art."*

(**Gerard J. Holzmann, NASA JPL, Pasadena**)

# Content of this lecture

- Introduction
  - why model checking?, how to model check?
- Transition systems
  - paths, traces, program graphs
- Linear time properties
  - safety, liveness, decomposition
- Fairness
  - unconditional, strong and weak fairness

# Content of this lecture

⇒ Introduction

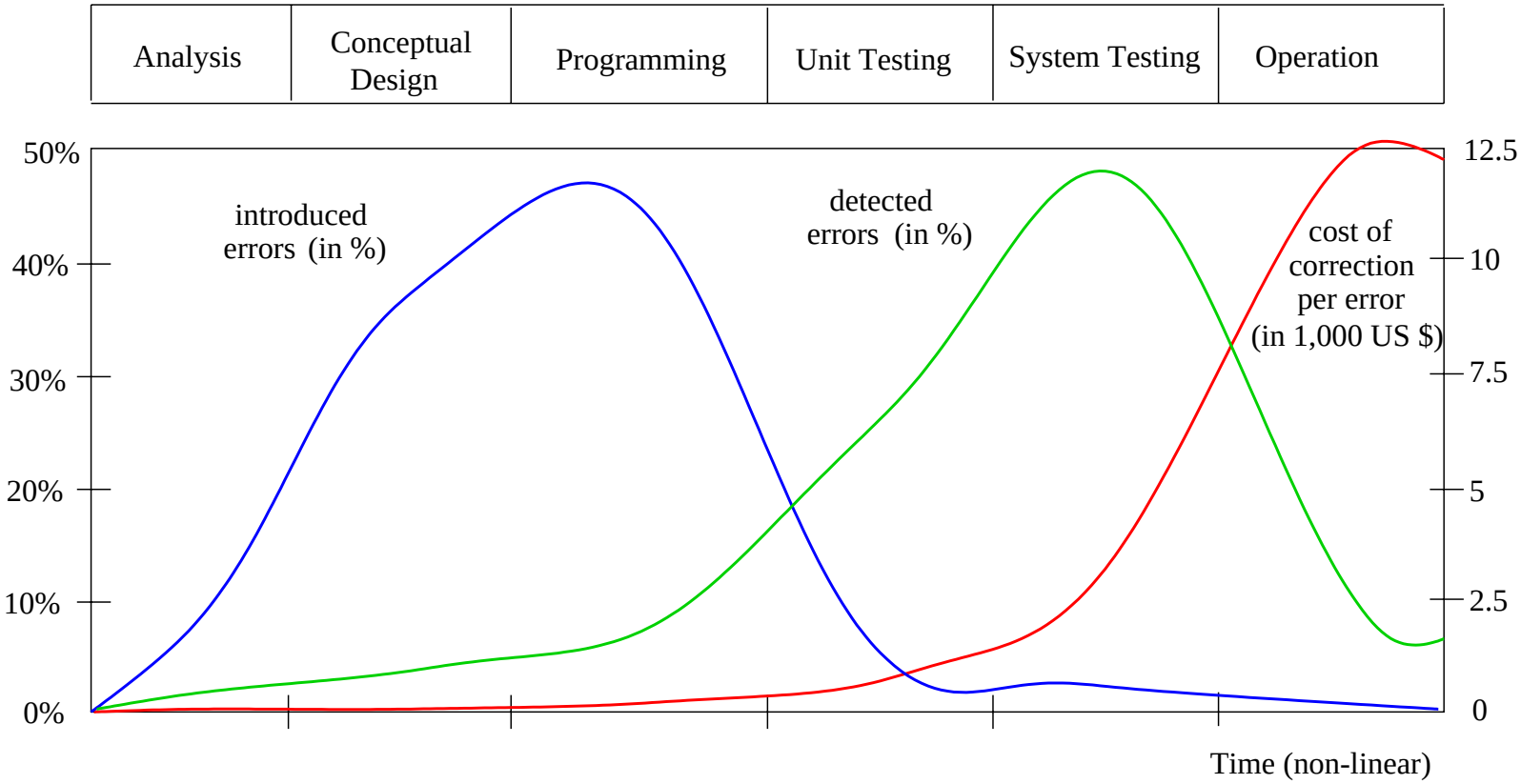
- why model checking?, how to model check?

- Transition systems
  - paths, traces, program graphs

- Linear time properties
  - safety, liveness, decomposition

- Fairness
  - unconditional, strong and weak fairness

# Catching software bugs: the sooner, the better

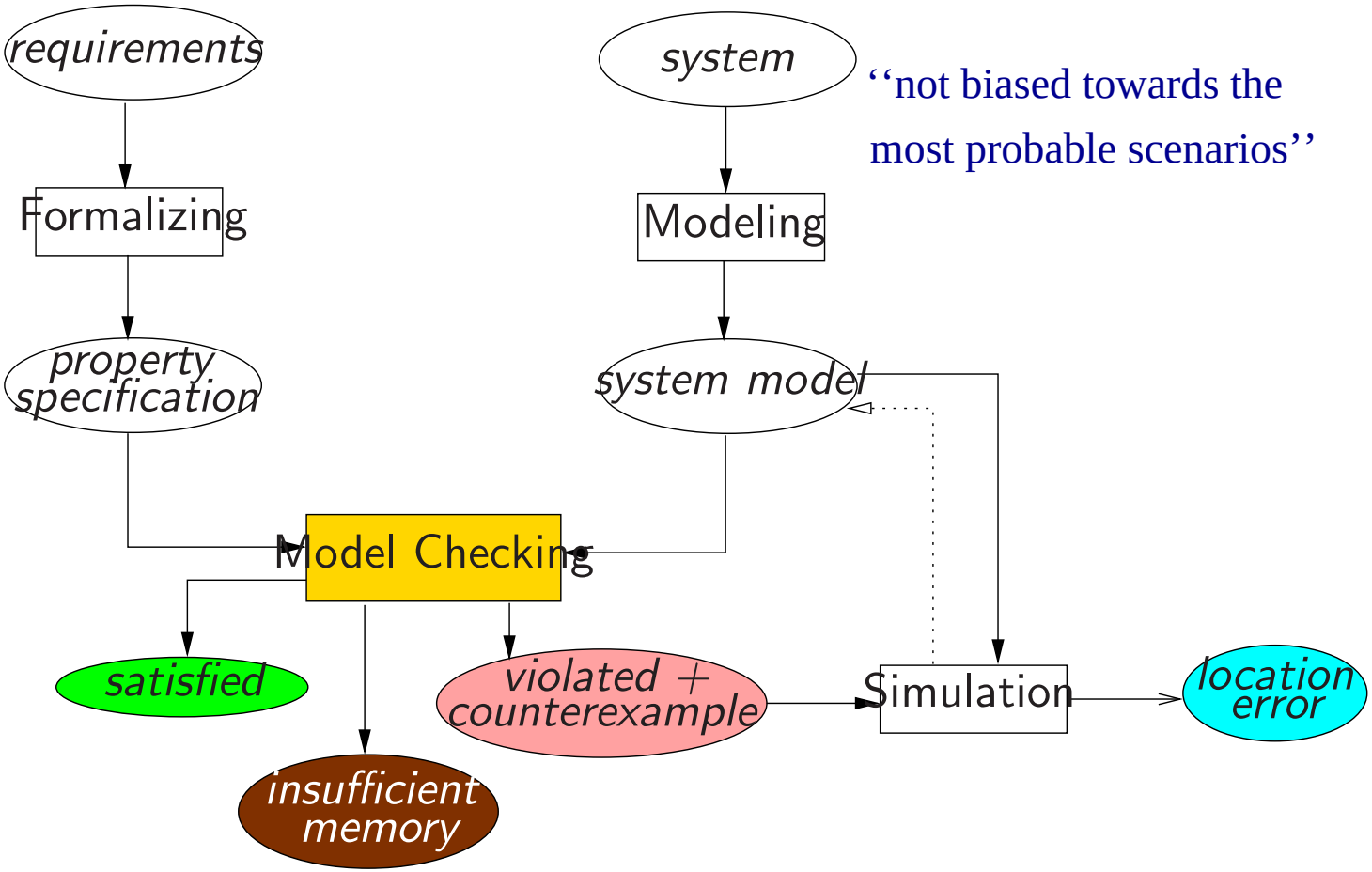


# Milestones in formal verification

- Mathematical approach towards program correctness (Turing, 1949)
- Syntax-based technique for sequential programs (Hoare, 1969)
  - for a given input, does a computer program generate the correct output?
  - based on compositional proof rules expressed in predicate logic
- Syntax-based technique for concurrent programs (Pnueli, 1977)
  - can handle properties referring to situations during the computation
  - based on proof rules expressed in temporal logic
- Automated verification of concurrent programs (Emerson & Clarke, 1981)
  - model-based instead of proof-rule based approach
  - does the concurrent program satisfy a given (logical) property?

*these formal techniques are not biased towards the most probable scenarios*

# Model checking overview



# The model checking process

- Modeling phase
  - model the system under consideration
  - as a first sanity check, perform some simulations
  - formalise the property to be checked
- Running phase
  - run the model checker to check the validity of the property in the model
- Analysis phase
  - property satisfied? $\rightarrow$ check next property (if any)
  - property violated? $\rightarrow$
    1. analyse generated counterexample by simulation
    2. refine the model, design, or property . . . and repeat the entire procedure
  - out of memory? $\rightarrow$ try to reduce the model and try again

# Content of this lecture

- Introduction
  - why model checking?, how to model check?

$\Rightarrow$ Transition systems
  - paths, traces, program graphs

- Linear time properties
  - safety, liveness, decomposition

- Fairness
  - unconditional, strong and weak fairness

# Transition systems

- Model to describe the behaviour of systems
- Digraphs where nodes represent *states*, and edges model *transitions*
- State:
  - the current colour of a traffic light
  - the current values of all program variables + the program counter
  - the current value of the registers together with the values of the input bits
- Transition: ("state change")
  - a switch from one colour to another
  - the execution of a program statement
  - the change of the registers and output bits for a new input

# Formal definition

A *transition system* $TS$ is a tuple $(S, Act, \rightarrow, I, AP, L)$ where

- $S$ is a set of states
- $Act$ is a set of actions
- $\longrightarrow \subseteq S \times Act \times S$ is a transition relation
- $I \subseteq S$ is a set of initial states
- $AP$ is a set of atomic propositions
- $L : S \rightarrow 2^{AP}$ is a labeling function

$S$ and $Act$ are either finite or countably infinite

Notation: $s \xrightarrow{\alpha} s'$ instead of $(s, \alpha, s') \in \longrightarrow$

# Paths

- An *infinite path fragment* $\pi$ is an infinite state sequence:

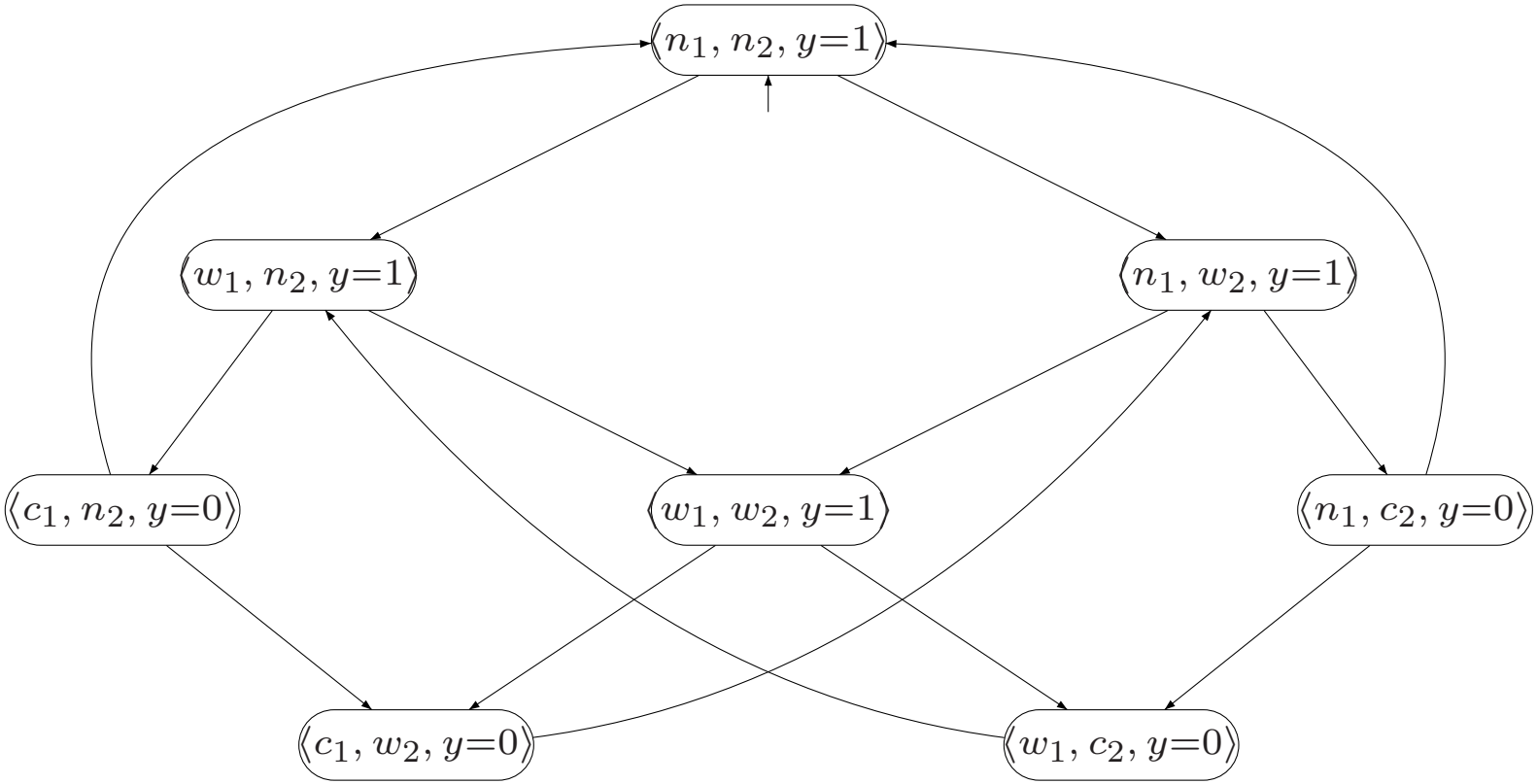
$$s_0\, s_1\, s_2 \ldots \quad \text{such that } s_i \in \mathit{Post}(s_{i-1}) \text{ for all } i > 0$$

- Notations for path fragment $\pi = s_0\, s_1\, s_2 \ldots$:
  - $\mathit{first}(\pi) = s_0 = \pi[0]$; let $\pi[j] = s_j$ denote the $j$-th state of $\pi$
  - $j$-th prefix $\pi[..j] = s_0\, s_1 \ldots s_j$ and $j$-th suffix $\pi[j..] = s_j\, s_{j+1} \ldots$

- A *path* of *TS* is an initial, maximal path fragment
  - a *maximal* path fragment cannot be prolonged
  - a path fragment is *initial* if $s_0 \in I$

- $\mathit{Paths}(s)$ is the set of maximal path fragments $\pi$ with $\mathit{first}(\pi) = s$

# A mutual exclusion algorithm



# Traces

- Actions are mainly used to model the (possibility of) interaction
  - synchronous or asynchronous communication
- Here, focus on the states that are visited during executions
  - the states themselves are not "observable", but just their atomic propositions
- Traces are sequences of the form $L(s_0)\,L(s_1)\,L(s_2)\ldots$
  - just register the (set of) atomic propositions that are valid along the execution
- For transition systems without terminal states:
  - traces are infinite words over the alphabet $2^{AP}$, i.e., they are in $\left(2^{AP}\right)^{\omega}$
  - we will (mostly) assume that there are no terminal states

# Traces

- Let transition system $\mathit{TS} = (S, \mathit{Act}, \rightarrow, I, \mathit{AP}, L)$ without terminal states
- The *trace* of path fragment $\pi = s_0\, s_1 \ldots$ is $\mathit{trace}(\pi) = L(s_0)\, L(s_1) \ldots$
  - the trace of $\widehat{\pi} = s_0\, s_1 \ldots s_n$ is $\mathit{trace}(\widehat{\pi}) = L(s_0)\, L(s_1) \ldots L(s_n)$.
- The set of traces of a set $\Pi$ of paths: $\mathit{trace}(\Pi) = \{\, \mathit{trace}(\pi) \mid \pi \in \Pi \,\}$
- $\mathit{Traces}(s) = \mathit{trace}(\mathit{Paths}(s))$ $\qquad\qquad$ $\mathit{Traces}(\mathit{TS}) = \bigcup_{s \in I} \mathit{Traces}(s)$

# Example traces

Let $AP = \{ \mathit{crit}_1, \mathit{crit}_2 \}$

Example path:

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

The trace of this path is the infinite word:

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

The trace of the finite path fragment:

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

is:

$$\mathit{trace}(\widehat{\pi}) \;=\; \varnothing\, \varnothing\, \varnothing\, \{ \mathit{crit}_2 \}\, \varnothing\, \{ \mathit{crit}_1 \}$$

# Program graphs: A beverage vending machine

"Abstract" transitions:

$$start \xrightarrow{true:coin} select \qquad \text{and} \qquad start \xrightarrow{true:refill} start$$

$$select \xrightarrow{nsprite>0:sget} start \qquad \text{and} \qquad select \xrightarrow{nbeer>0:bget} start$$

$$select \xrightarrow{nsprite=0 \wedge nbeer=0:ret\_coin} start$$

| Action | Effect on variables |
|---|---|
| $coin$ | |
| $ret\_coin$ | |
| $sget$ | $nsprite := nsprite - 1$ |
| $bget$ | $nbeer := nbeer - 1$ |
| $refill$ | $nsprite := max;\ nbeer := max$ |

# Some preliminaries

- typed variables with a valuation that assigns values to variables
  - e.g., $\eta(x) = 17$ and $\eta(y) = -2$
- the set of Boolean conditions over *Var*
  - propositional logic formulas whose propositions are of the form "$\overline{x} \in \overline{D}$"
  - $(-3 < x \leqslant 5) \ \wedge \ (y = green) \ \wedge \ (x \leqslant 2{\cdot}x')$
- effect of the actions is formalized by means of a mapping:

$$\mathit{Effect} : \mathit{Act} \times \mathit{Eval}(\mathit{Var}) \rightarrow \mathit{Eval}(\mathit{Var})$$

  - e.g., $\alpha \equiv x := y{+}5$ and evaluation $\eta(x) = 17$ and $\eta(y) = -2$
  - $\mathit{Effect}(\alpha, \eta)(x) \ = \ \eta(y) + 5 = 3,$ and $\mathit{Effect}(\alpha, \eta)(y) \ = \eta(y) = -2$

# Program graphs

A *program graph* $PG$ over set $Var$ of typed variables is a tuple

$$(Loc, Act, Effect, \longrightarrow, Loc_0, g_0) \quad \text{where}$$

- $Loc$ is a set of *locations* with initial locations $Loc_0 \subseteq Loc$
- $Act$ is a set of actions
- $Effect : Act \times Eval(Var) \to Eval(Var)$ is the *effect* function
- $\longrightarrow \subseteq Loc \times \underbrace{Cond(Var)}_{\text{Boolean conditions over } Var} \times Act \times Loc$, transition relation
- $g_0 \in Cond(Var)$ is the initial *condition*.

Notation: $\ell \xrightarrow{g:\alpha} \ell'$ denotes $(\ell, g, \alpha, \ell') \in \longrightarrow$

# Beverage vending machine

- $\mathit{Loc} = \{\, start, select \,\}$ with $\mathit{Loc}_0 = \{\, start \,\}$
- $\mathit{Act} = \{\, bget, sget, coin, ret\_coin, refill \,\}$
- $\mathit{Var} = \{\, nsprite, \quad nbeer \,\}$ with domain $\{\, 0, 1, \ldots, max \,\}$
- $$\begin{array}{lcl} \mathit{Effect}(coin, \eta) & = & \eta \\ \mathit{Effect}(ret\_coin, \eta) & = & \eta \\ \mathit{Effect}(sget, \eta) & = & \eta[nsprite := nsprite - 1] \\ \mathit{Effect}(bget, \eta) & = & \eta[nbeer := nbeer - 1] \\ \mathit{Effect}(refill, \eta) & = & [nsprite := max, nbeer := max] \end{array}$$
- $g_0 = (nsprite = max \ \wedge \ nbeer = max)$

# From program graphs to transition systems

- Basic strategy: *unfolding*
  - state = location (current control) $\ell$ + data valuation $\eta$
  - initial state = initial location satisfying the initial condition $g_0$
- Propositions and labeling
  - propositions: "at $\ell$" and "$x \in D$" for $D \subseteq \mathit{dom}(x)$
  - $\langle \ell, \eta \rangle$ is labeled with "at $\ell$" and all conditions that hold in $\eta$
- $\ell \xrightarrow{g:\alpha} \ell'$ and $g$ holds in $\eta$ then $\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \mathit{Effect}(\alpha, \eta) \rangle$

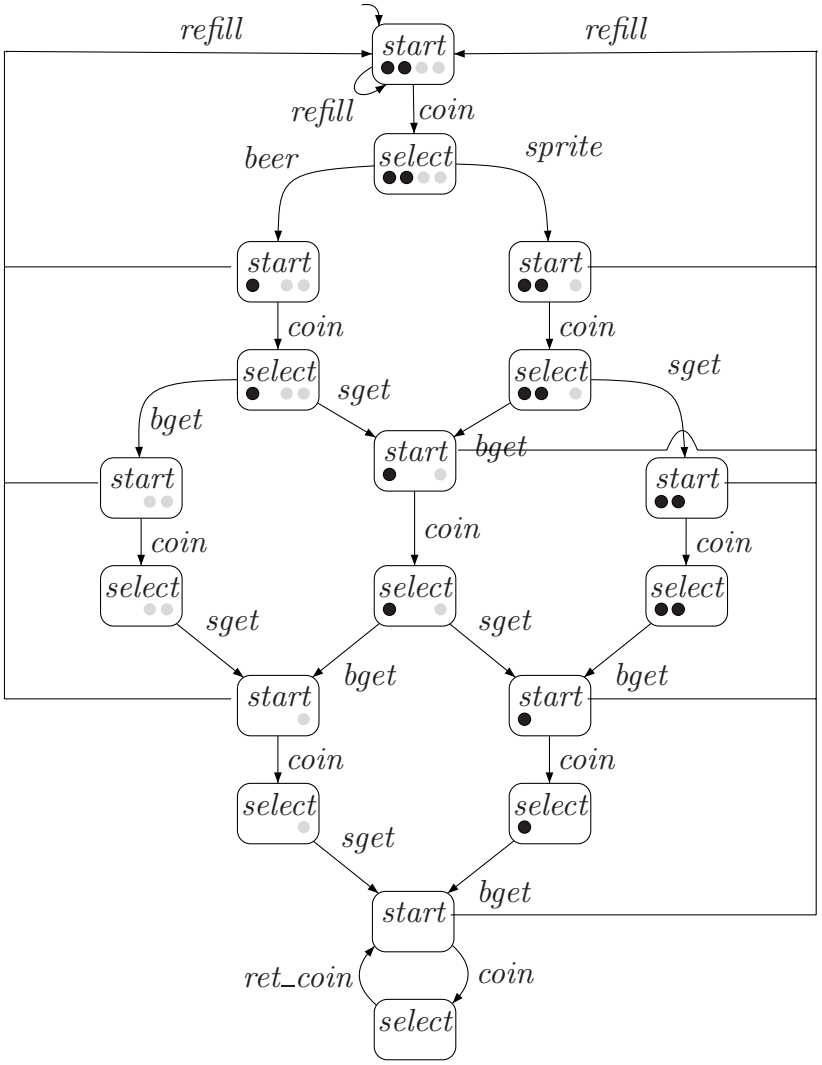
# Transition systems for program graphs

The transition system $\mathit{TS}(\mathit{PG})$ of program graph

$$\mathit{PG} = (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \longrightarrow, \mathit{Loc}_0, g_0)$$

over set $\mathit{Var}$ of variables is the tuple $(S, \mathit{Act}, \longrightarrow, I, \mathit{AP}, L)$ where

- $S = \mathit{Loc} \times \mathit{Eval}(\mathit{Var})$
- $\longrightarrow \subseteq S \times Act \times S$ is defined by the rule: $\dfrac{\ell \xrightarrow{g:\alpha} \ell' \quad \wedge \quad \eta \models g}{\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \mathit{Effect}(\alpha, \eta) \rangle}$
- $I = \{\langle \ell, \eta \rangle \mid \ell \in \mathit{Loc}_0, \eta \models g_0\}$
- $\mathit{AP} = \mathit{Loc} \cup \mathit{Cond}(\mathit{Var})$ and $L(\langle \ell, \eta \rangle) = \{\ell\} \cup \{g \in \mathit{Cond}(\mathit{Var}) \mid \eta \models g\}$.


refill
refill
start
refill
coin
beer
select
sprite
start
start
coin
coin
select
sget
select
sget
bget
start
bget
start
start
coin
coin
coin
select
select
select
sget
sget
bget
bget
start
start
coin
coin
select
select
sget
bget
start
ret_coin
coin
select

# Content of this lecture

- Introduction
  - why model checking?, how to model check?
- Transition systems
  - paths, traces, program graphs

$\Rightarrow$ Linear time properties

  - safety, liveness, decomposition
- Fairness
  - unconditional, strong and weak fairness

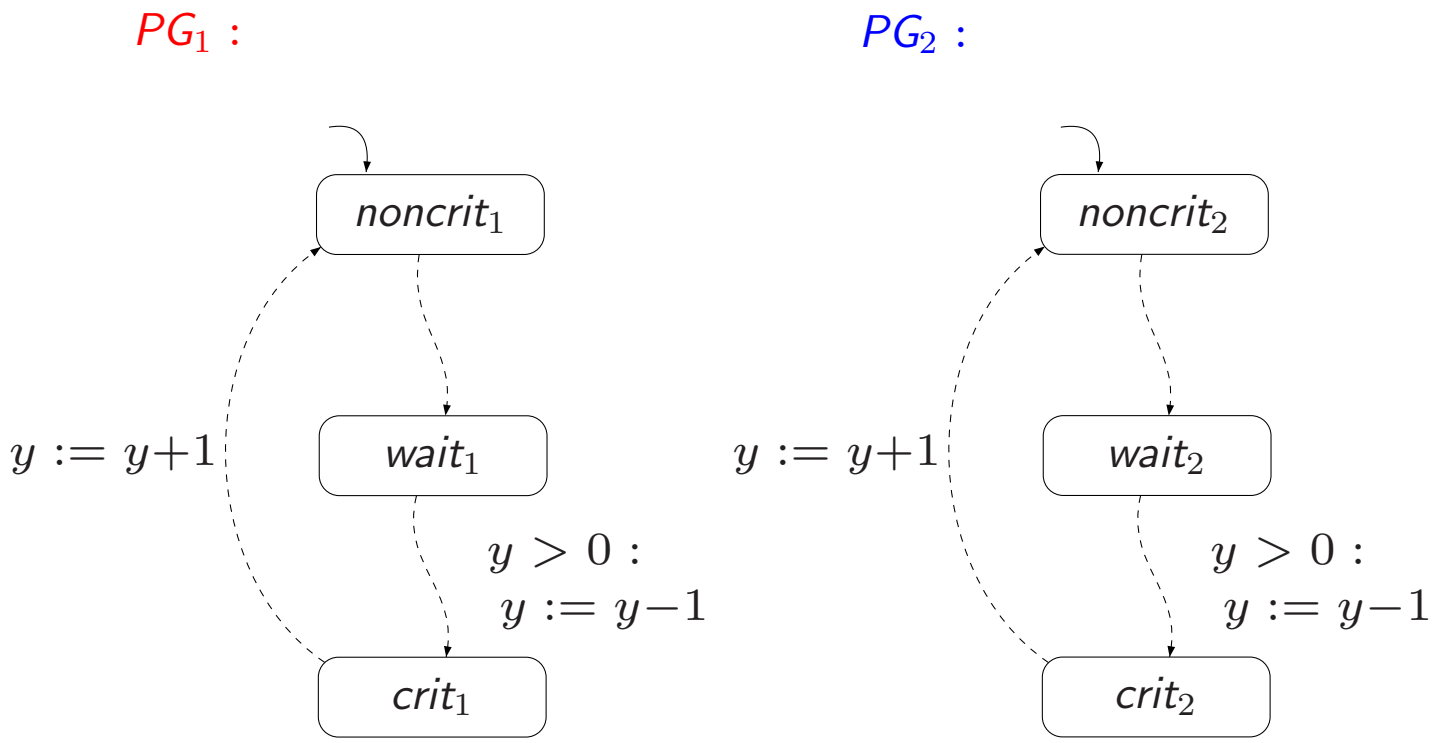
# Linear-time properties

- 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 $AP$):

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

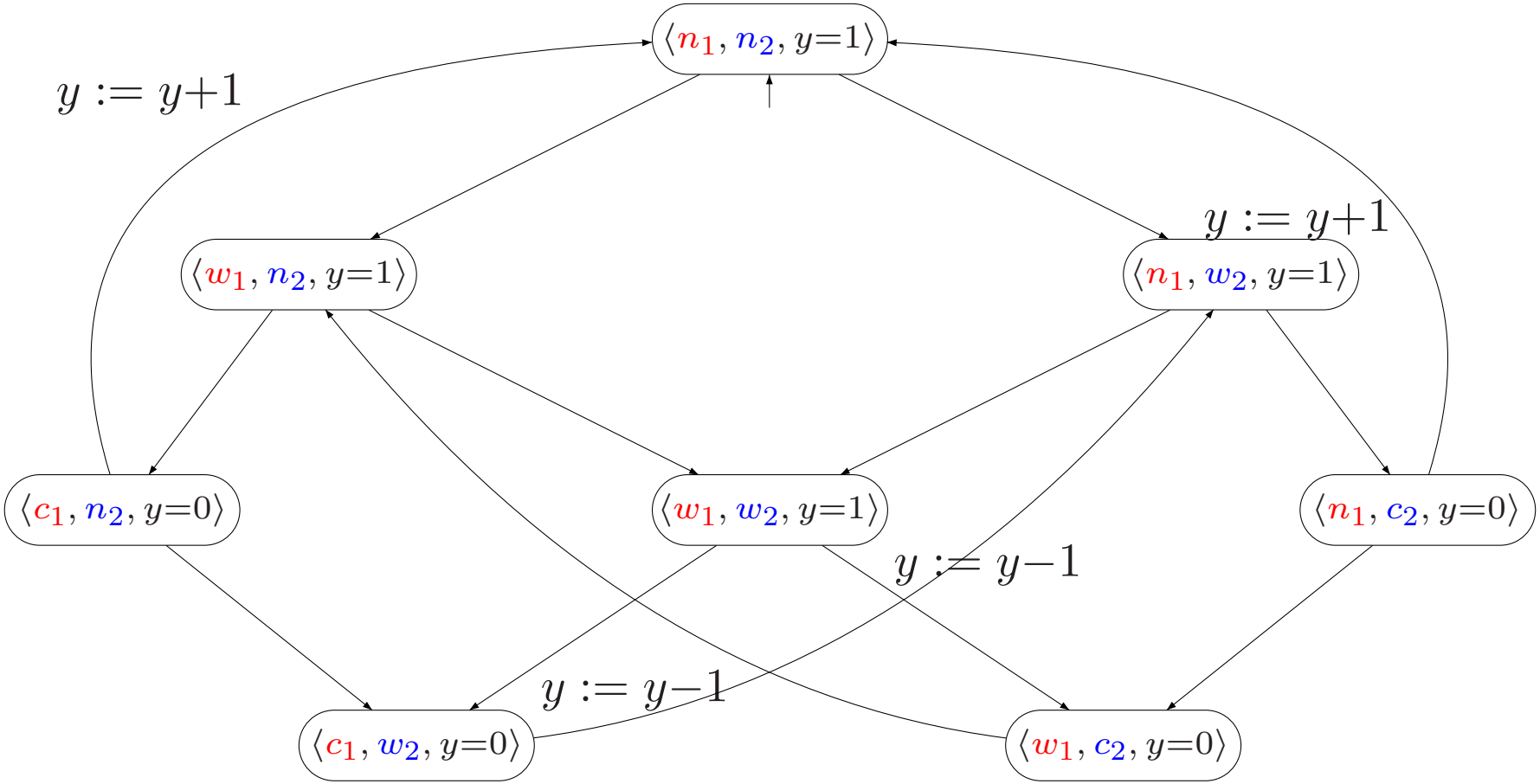
  - $TS$ satisfies the LT property $P$ if all its "observable" behaviors are admissible

# Semaphore-based mutual exclusion




$y{=}0$ 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 $AP = \{ crit_1, crit_2 \}$
  - other atomic propositions are not of any relevance for this property
- Formalization as LT property

$P_{mutex}$ = set of infinite words $A_0\, A_1\, A_2 \ldots$ with $\{ crit_1, crit_2 \} \not\subseteq A_i$ for all $0 \leqslant i$

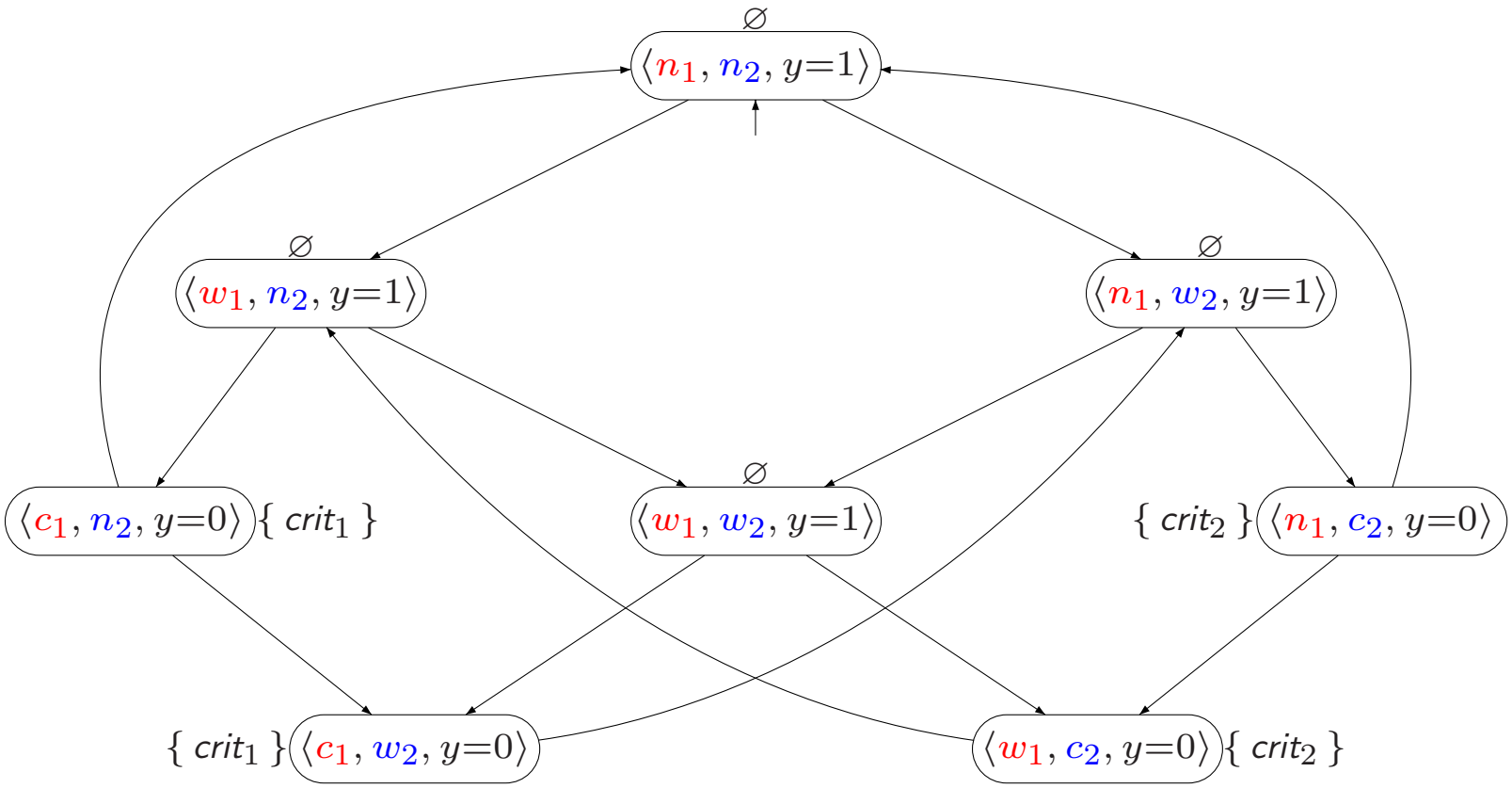
- Contained in $P_{mutex}$ are e.g., the infinite words:

$\{ crit_1 \}\, \{ crit_2 \}\, \{ crit_1 \}\, \{ crit_2 \}\, \{ crit_1 \}\, \{ crit_2 \} \ldots$ and
$\varnothing\, \varnothing\, \varnothing\, \varnothing\, \varnothing\, \varnothing\, \varnothing \ldots$

  - this does not apply to words of the form: $\{ crit_1 \}\, \varnothing\, \{ crit_1, crit_2 \} \ldots$

*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 $\{ crit_1, crit_2 \}$

# How to specify starvation freedom?

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

- Let $AP = \{ wait_1, crit_1, wait_2, crit_2 \}$
- Formalization as LT-property

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

$$\left( \overset{\infty}{\exists}\, j.\ wait_i \in A_j \right) \Rightarrow \left( \overset{\infty}{\exists}\, j.\ crit_i \in A_j \right) \quad \text{for each } i \in \{ 1, 2 \}$$

$\overset{\infty}{\exists}$ 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 $AP$):

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

if and only if

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

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

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:

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

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

- Note that

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

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

# 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 $\sigma$ 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

# Safety properties and finite traces

For transition system $TS$ without terminal states

and safety property $P_{safe}$:

$TS \models P_{safe}$ if and only if $Traces_{fin}(TS) \cap BadPref(P_{safe}) = \varnothing$

# Closure

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

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

  - if $\sigma = A_0\, A_1\, \ldots$ then $\mathit{pref}(\sigma) = \left\{ \varepsilon, A_0, A_0A_1, A_0A_1A_2, \ldots \right\}$

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

- The *closure* of LT property $P$:

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

  - 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?

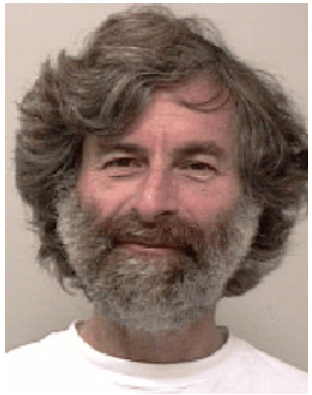
- Safety properties specify that "something bad never happens"
- Doing nothing easily fulfills a safety property
  - as this will never lead to a "bad" situation

$\Rightarrow$ Safety properties are complemented by liveness properties

  - that require some progress

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

# The meaning of liveness



[Lamport 2000]

The question of whether a real system satisfies a liveness property is meaningless; it can be answered only by observing the system for an infinite length of time, and real systems don't run forever.

Liveness is always an approximation to the property we really care about. We want a program to terminate within 100 years, but proving that it does would require addition of distracting timing assumptions.

So, we prove the weaker condition that the program eventually terminates. This doesn't prove that the program will terminate within our lifetimes, but it does demonstrate the absence of infinite loops.

# 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?*

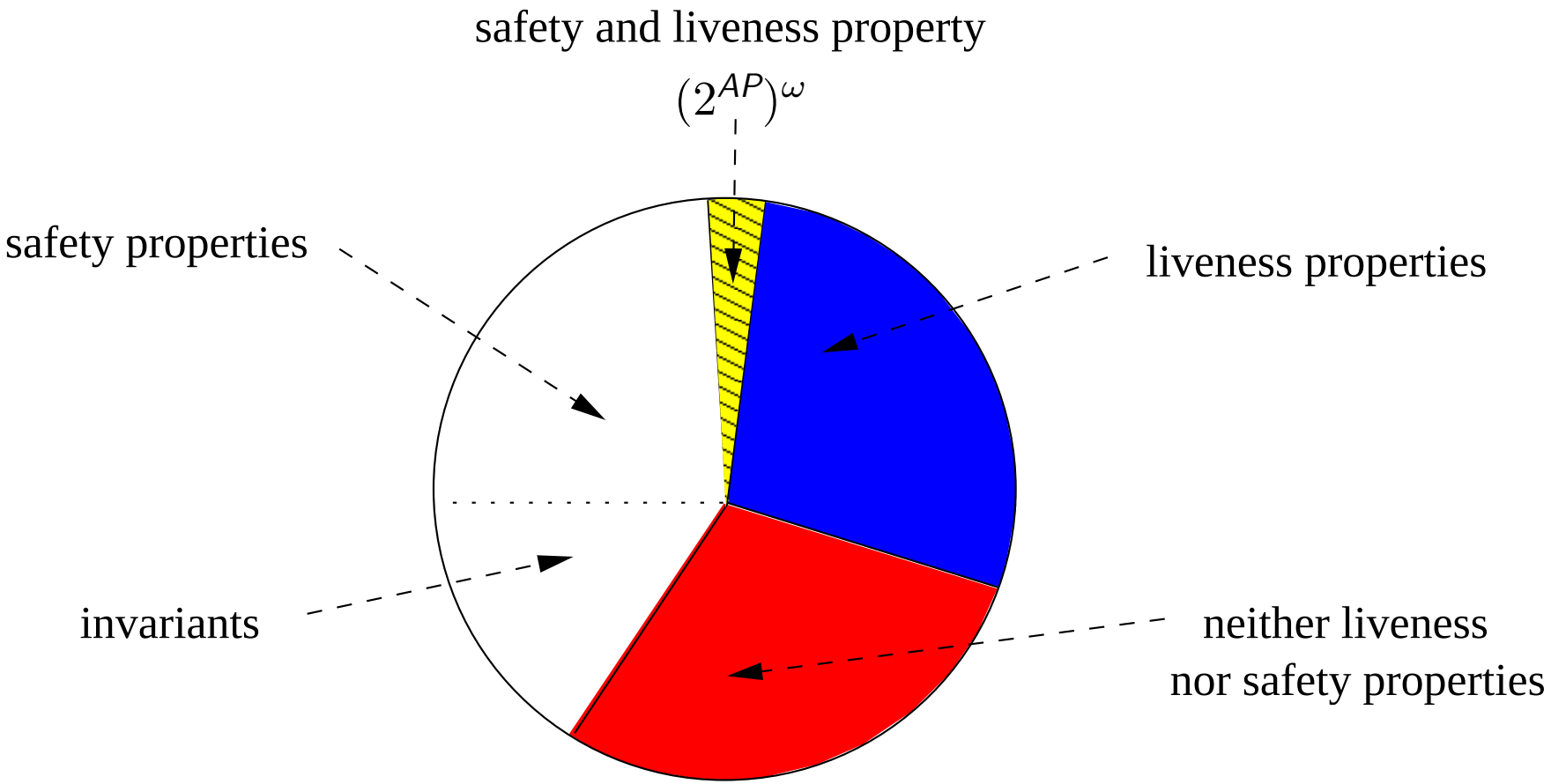
# Decomposition theorem

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



# Content of this lecture

- Introduction
  - why model checking?, how to model check?
- Transition systems
  - paths, traces, program graphs
- Linear time properties
  - safety, liveness, decomposition

$\Rightarrow$ Fairness

  - unconditional, strong and weak fairness

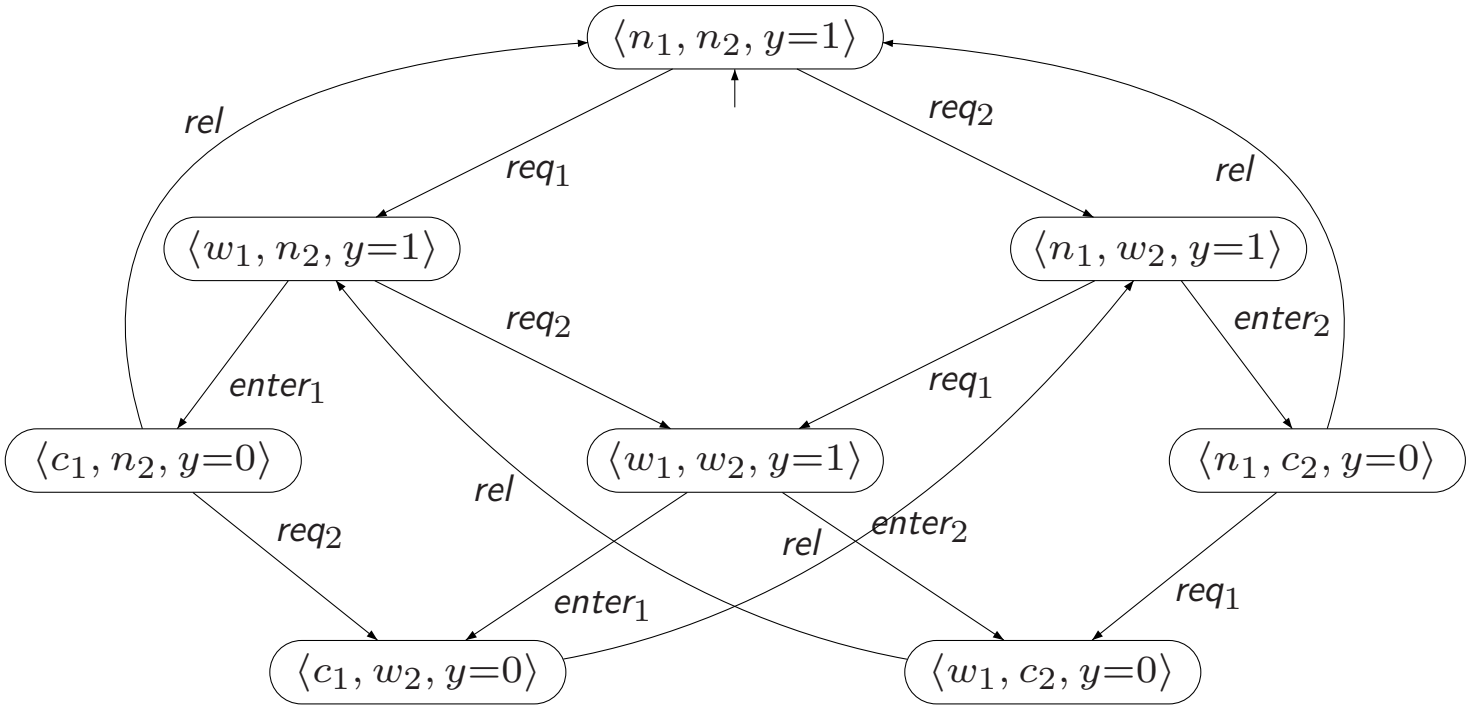
# Does this program always terminate?

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

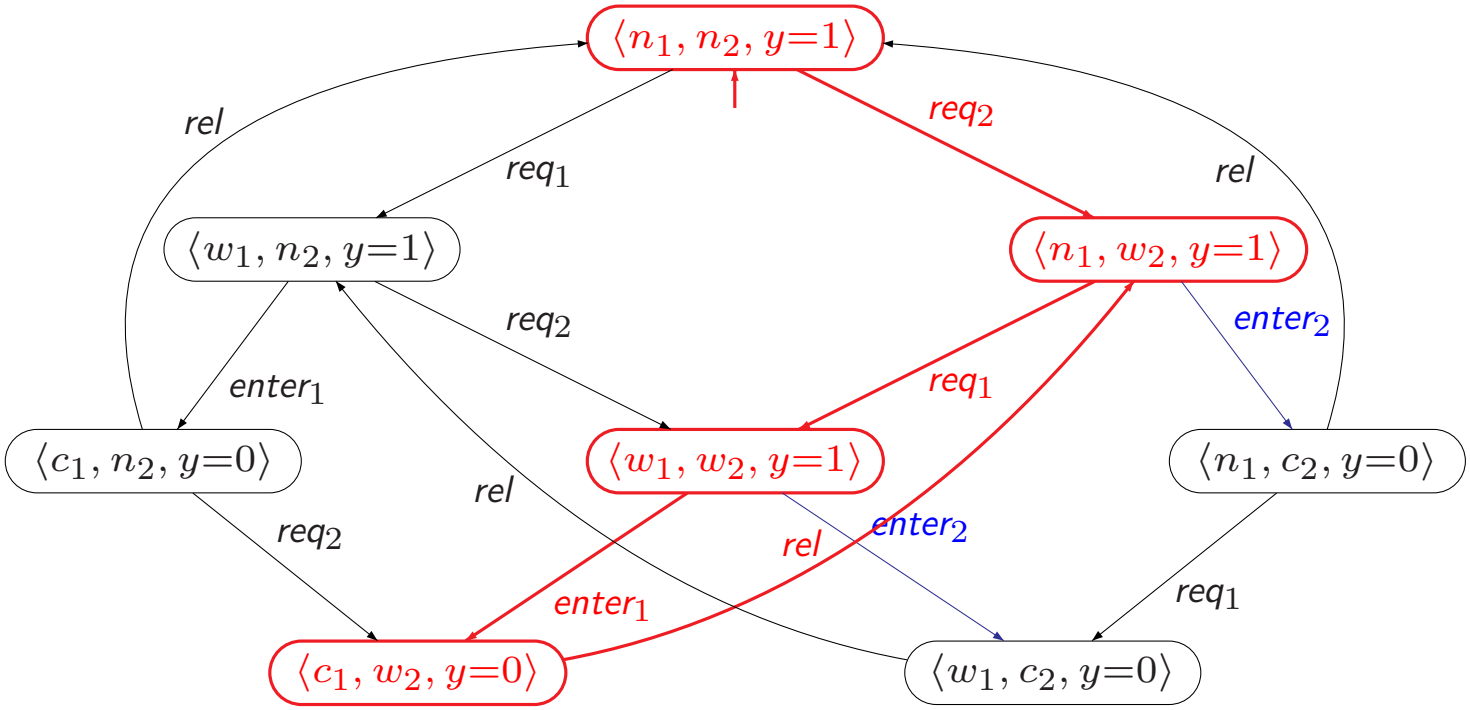
$$
\begin{array}{rcl}
\textit{where} & & \\
\textbf{proc}\ \mathsf{Inc} & = & \textbf{while}\ \langle\, x \geqslant 0\ \textbf{do}\ x := x + 1 \,\rangle\ \textbf{od} \\
\textbf{proc}\ \mathsf{Reset} & = & x := -1
\end{array}
$$

$x$ is a shared integer variable that initially has value 0

# Is it possible to starve?

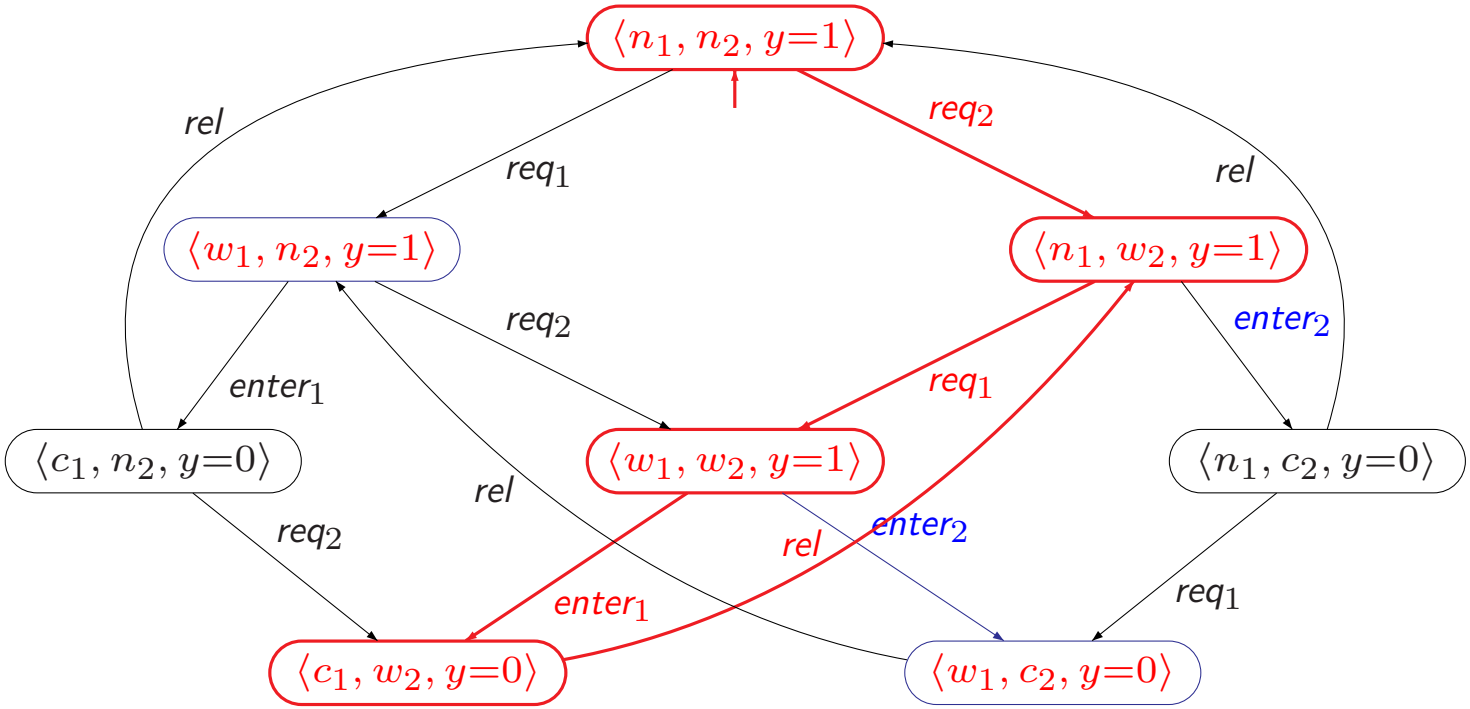


# Process two starves




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:

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

$x$ is a shared integer variable that initially has value 0

# Fairness constraints

For $TS = (S, Act, \rightarrow, I, AP, L)$ without terminal states, $A \subseteq Act$,
and infinite execution fragment $\rho = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots$ of $TS$:

1. $\rho$ 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. $\rho$ is *strongly $A$-fair* whenever:

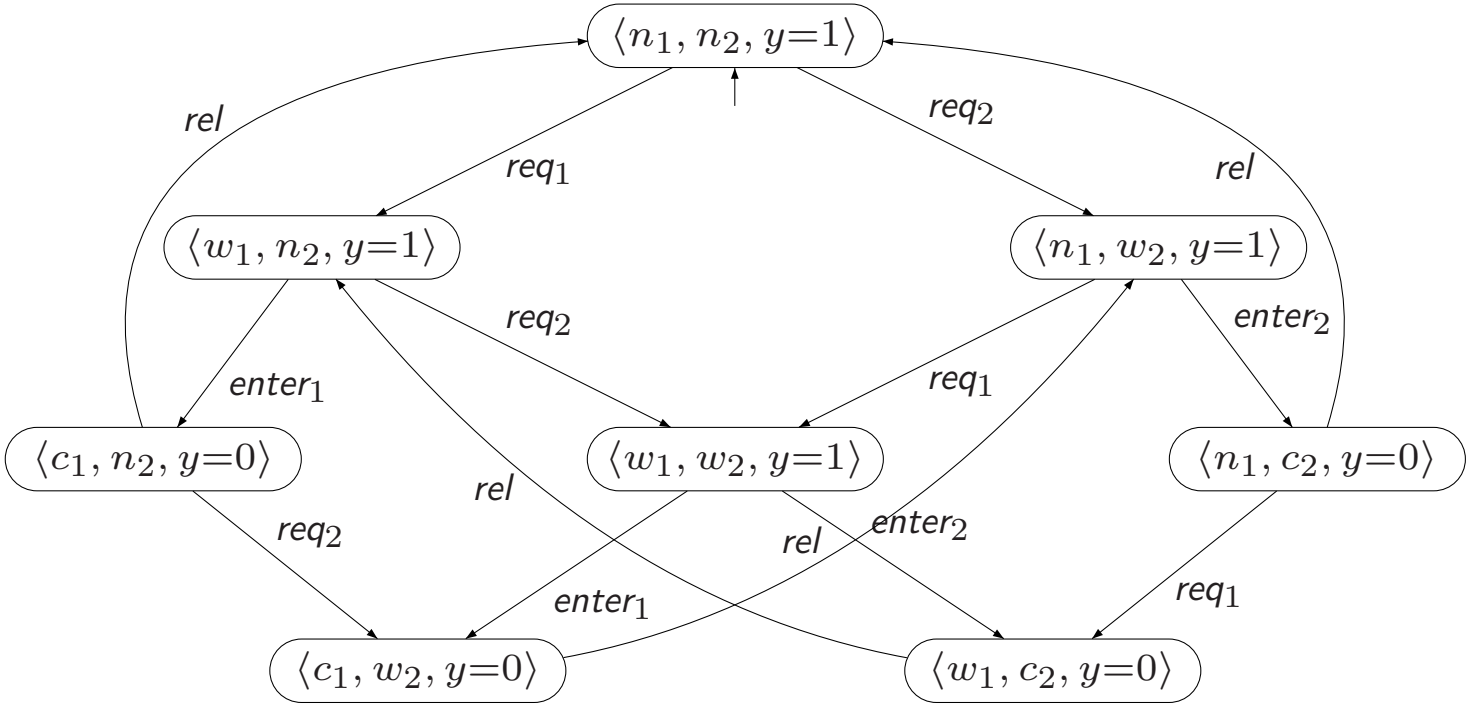
$$\underbrace{(\,\forall k \geqslant 0.\, \exists j \geqslant k.\, Act(s_j) \cap A \neq \varnothing\,)}_{\text{infinitely often } A \text{ is enabled}} \implies \underbrace{\forall k \geqslant 0.\, \exists j \geqslant k.\, \alpha_j \in A}_{\text{infinitely often } A \text{ is taken}}$$

3. $\rho$ is *weakly $A$-fair* whenever:

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

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

# 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

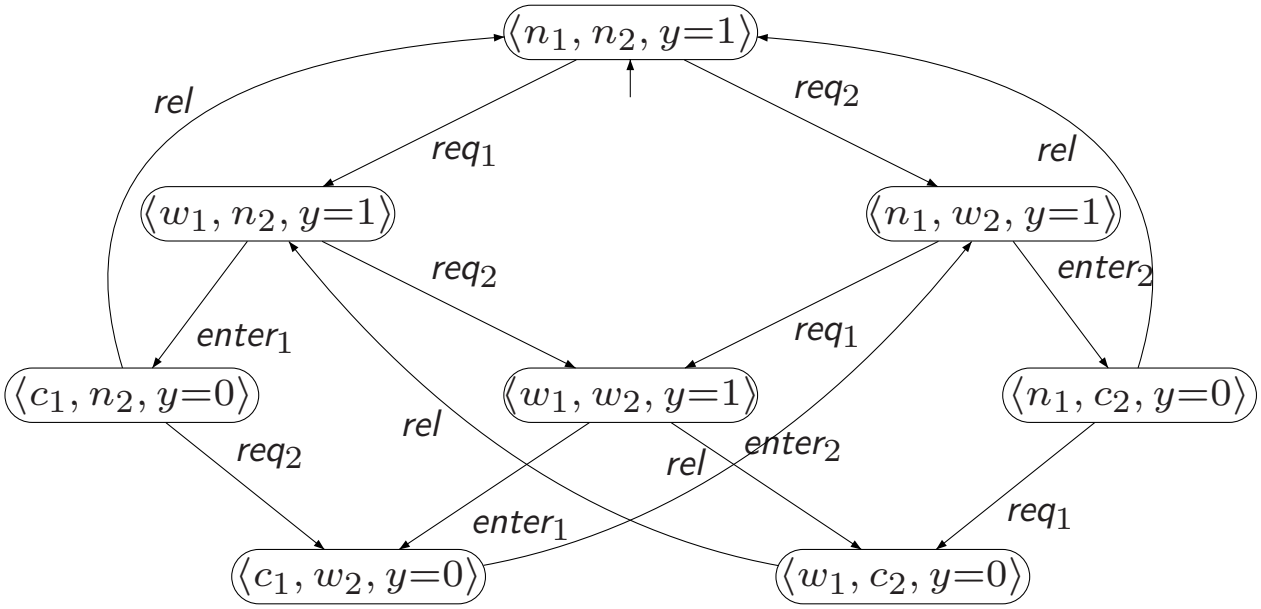
$$\mathcal{F} \;=\; (\mathcal{F}_{ucond}, \mathcal{F}_{strong}, \mathcal{F}_{weak})$$

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

- Execution $\rho$ 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 $A \in \mathcal{F}_{weak}$

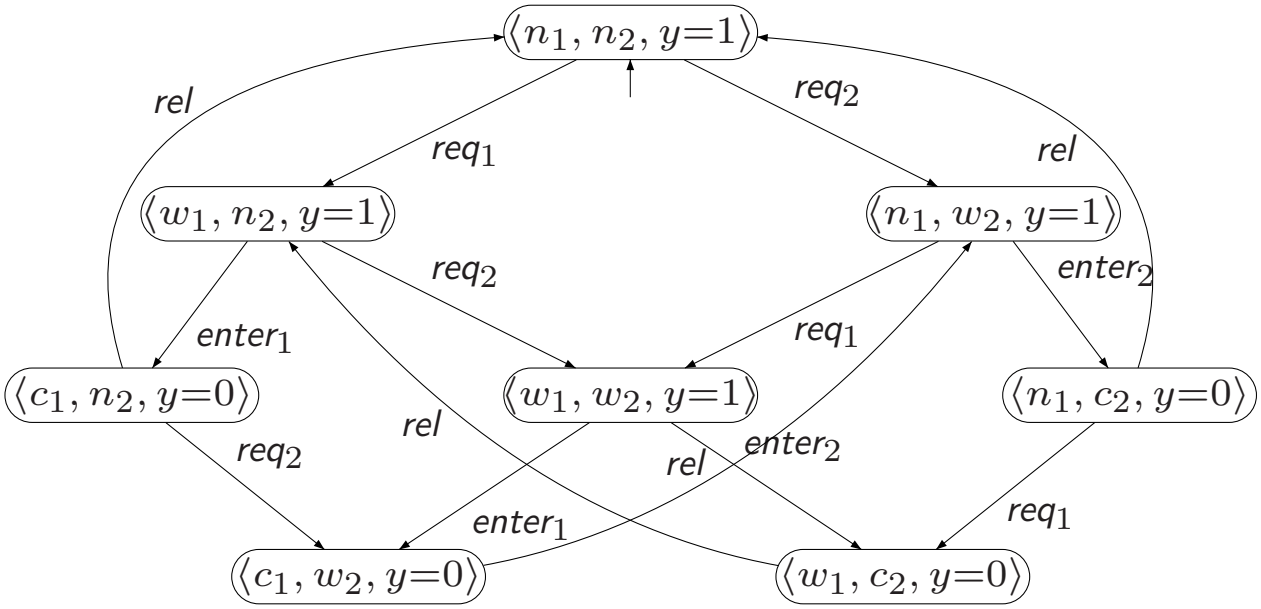
fairness assumption $(\varnothing, \mathcal{F}', \varnothing)$ denotes strong fairness; $(\varnothing, \varnothing, \mathcal{F}')$ weak, etc.

# Fairness for mutual exclusion



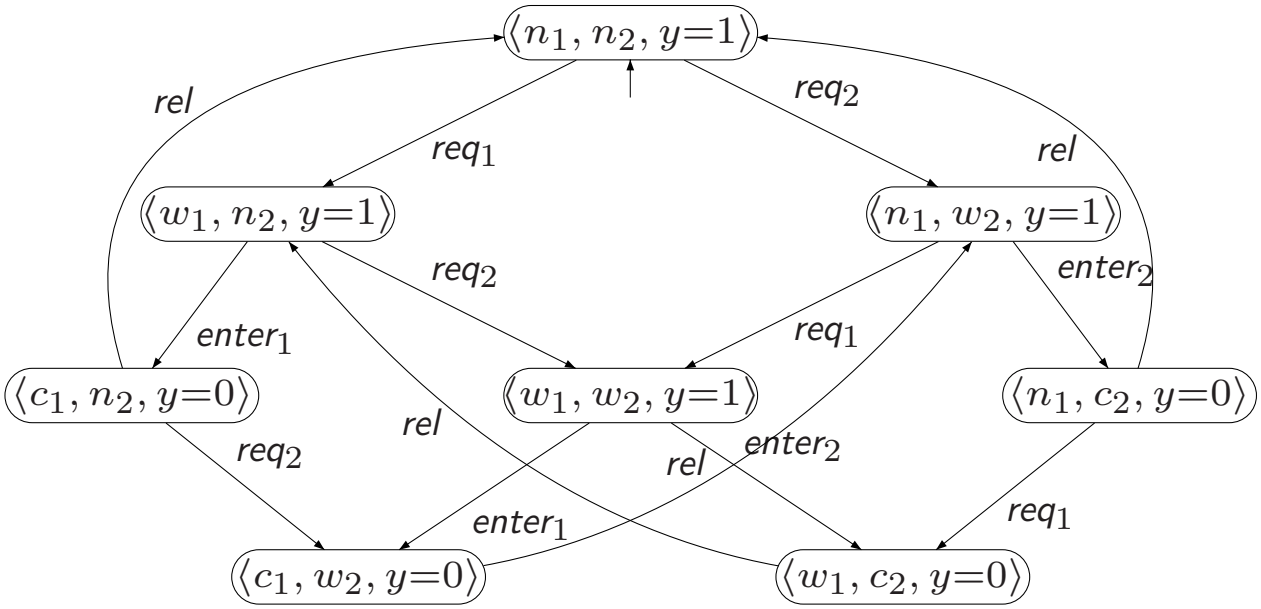

$$\mathcal{F} = (\varnothing, \underbrace{\{\{\, enter_1, enter_2 \,\}\}}_{\mathcal{F}_{strong}}, \varnothing)$$

# Fairness for mutual exclusion




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

# Fairness for mutual exclusion




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

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

# Fair paths and traces

- Path $s_0 \rightarrow s_1 \rightarrow s_2 \ldots$ 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 $\sigma$ is *$\mathcal{F}$-fair* if there exists an $\mathcal{F}$-fair path $\pi$ 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 $P$:

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

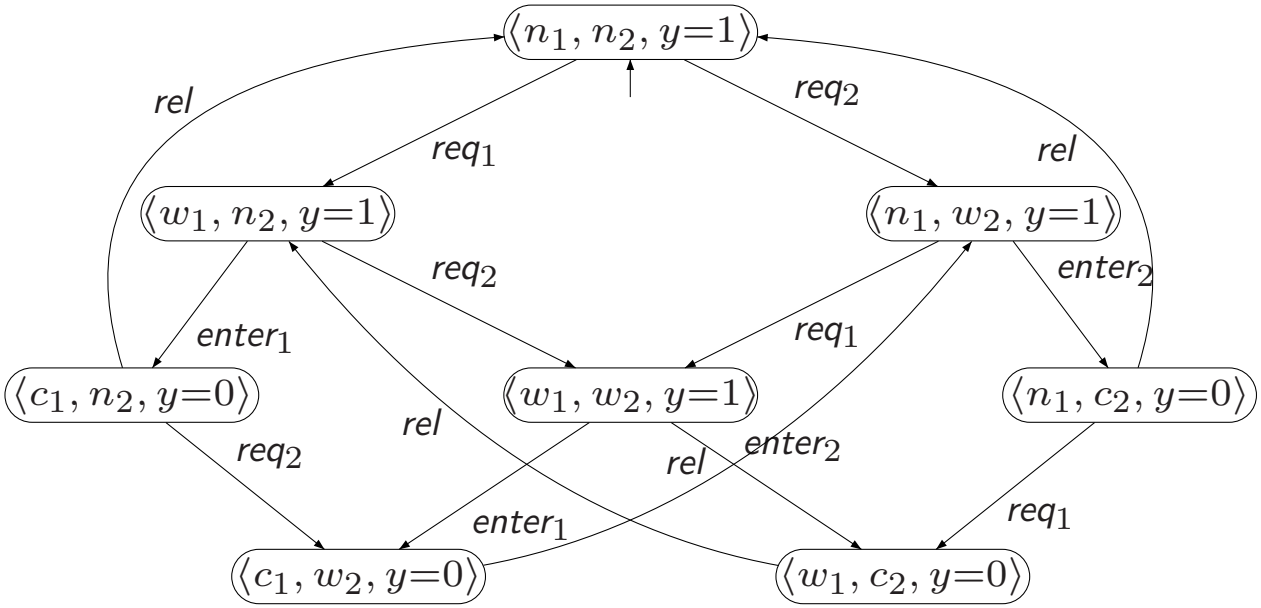
  - *TS* satisfies the LT property $P$ if *all* its observable behaviors are admissible

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

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

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

# Fairness for mutual exclusion




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

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

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

# Fairness and safety properties

For $TS$ and safety property $P_{safe}$ (both over $AP$)
suhc that for any $s \in \mathit{Reach}(TS)$: $\mathit{FairPaths}_{\mathcal{F}}(s) \neq \varnothing$:
$TS \models P_{safe}$ if and only if $TS \models_{\mathcal{F}} P_{safe}$

Safety properties are thus preserved by "realizable" fairness assumptions