# Transition Systems and Linear-Time Properties

## Part #1 of Logic and Verification

*Joost-Pieter Katoen*

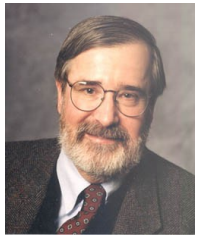
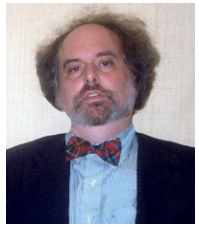
Software Modeling and Verification Group

RWTH Aachen University

MOVEP 2014, University of Nantes, July 7, 2014

# Model checking

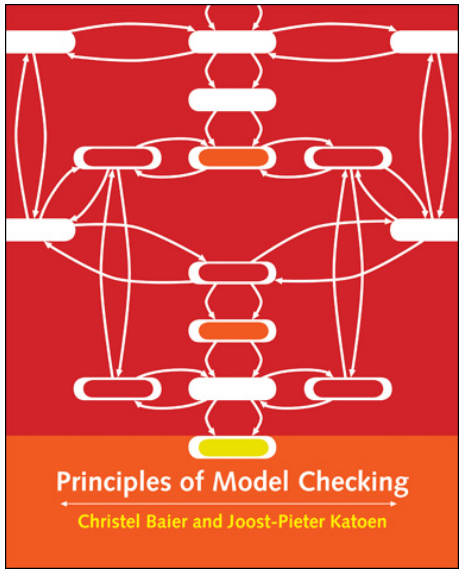
- Automated model-based verification and debugging technique
  - model of system = Kripke structure $\approx$ labeled transition system
  - properties expressed in temporal logic like LTL or CTL
  - provides counterexamples in case of property refutation
- Various striking examples
  - Needham-Schroeder 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

# 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

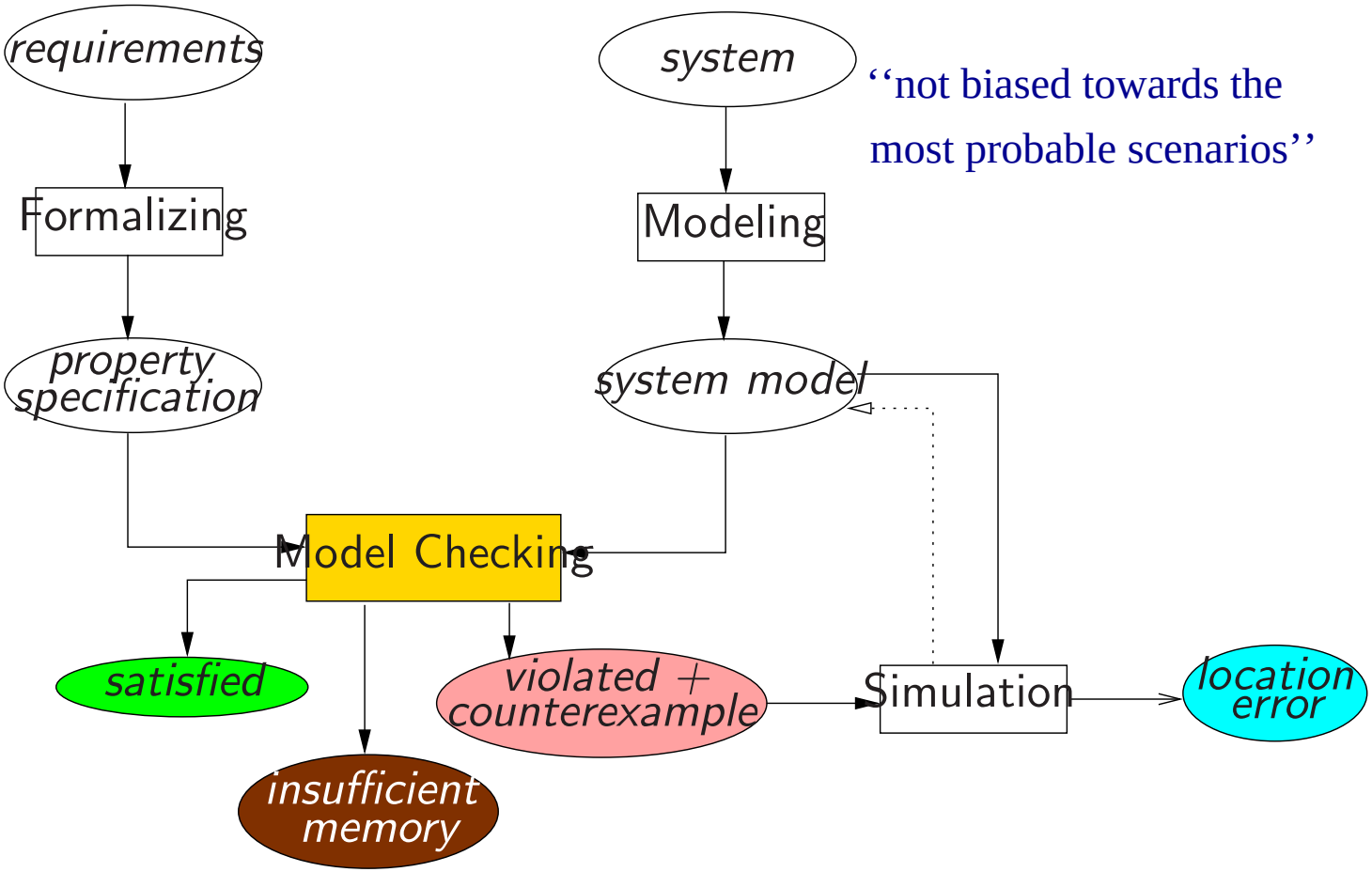
- Fairness
  - unconditional, strong and weak fairness

# 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:

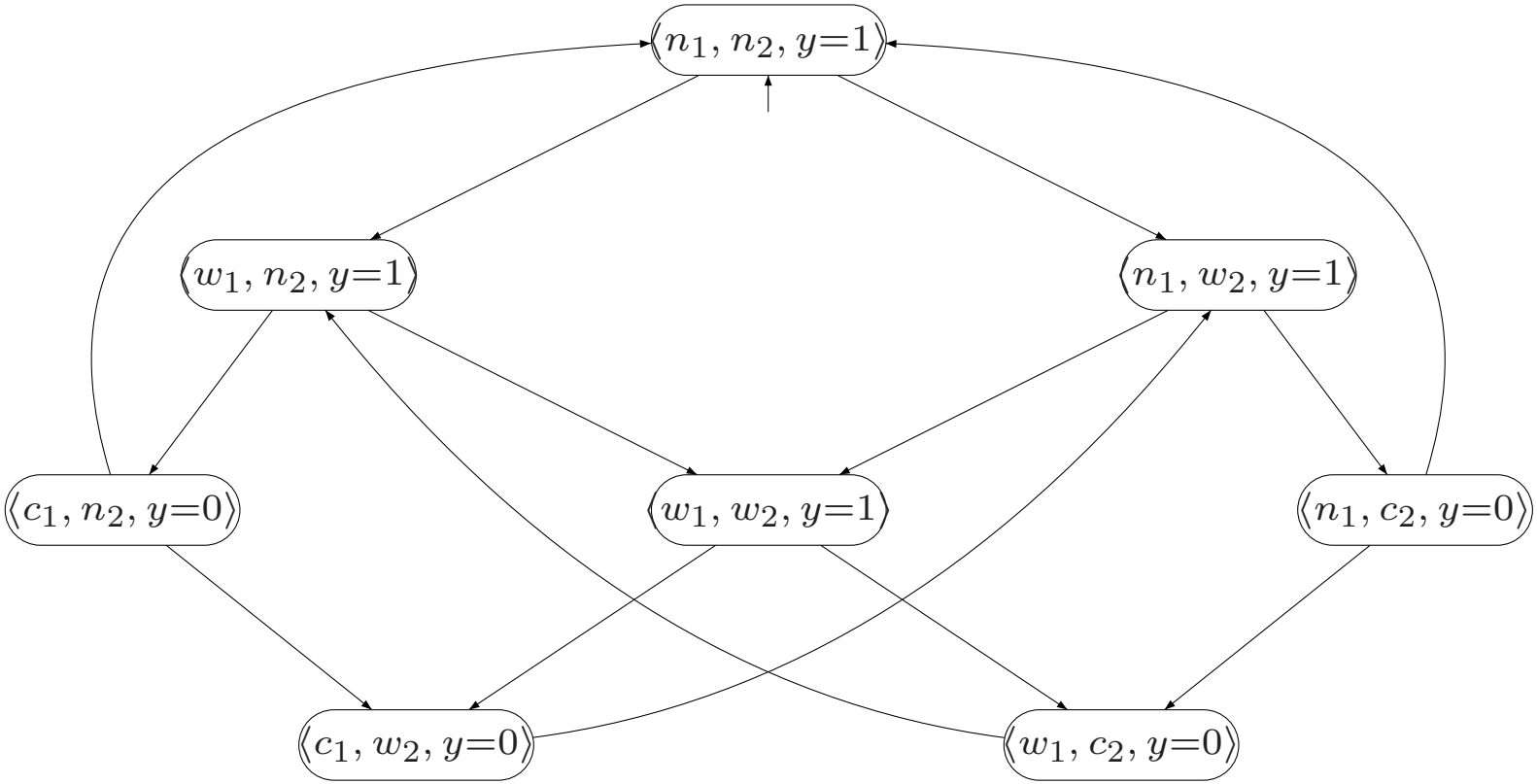
$$s_0\, s_1\, s_2 \ldots \quad \text{such that } s_i \in \mathit{Post}(s_{i-1}) \text{ for all } i > 0$$

- 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

# 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 $\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \mathit{Effect}(\alpha, \eta) \rangle$

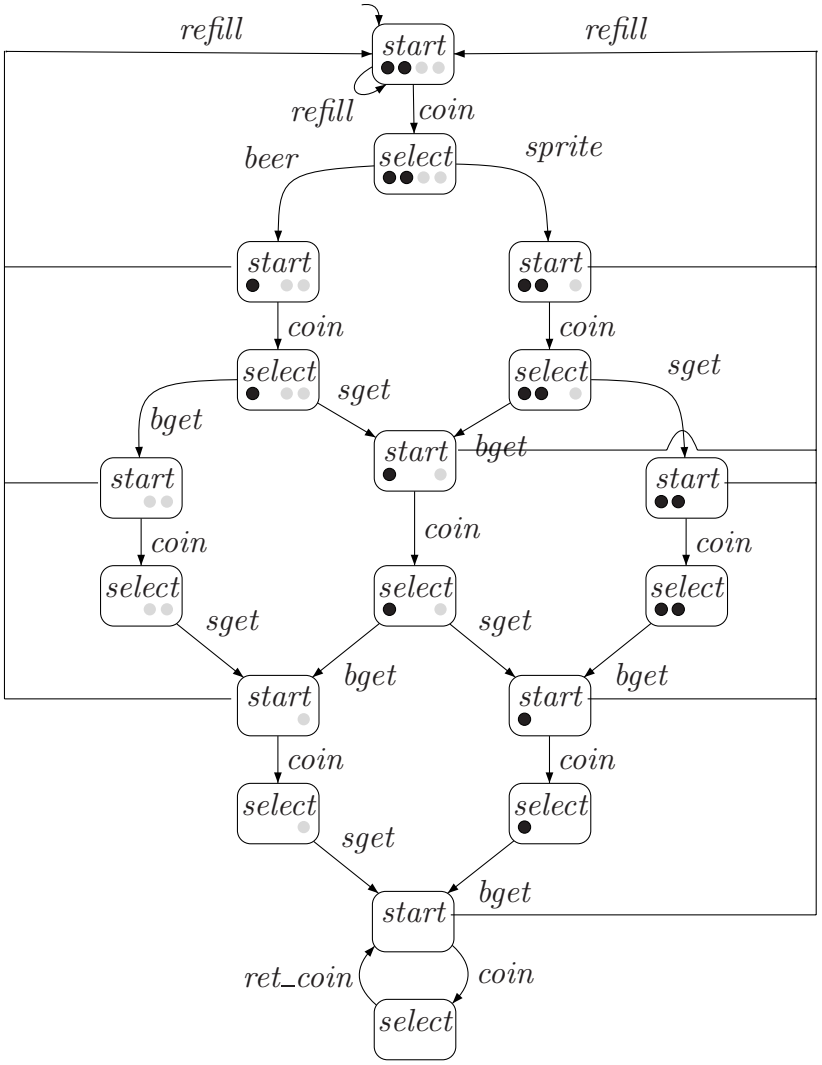
# Transition systems for program graphs

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

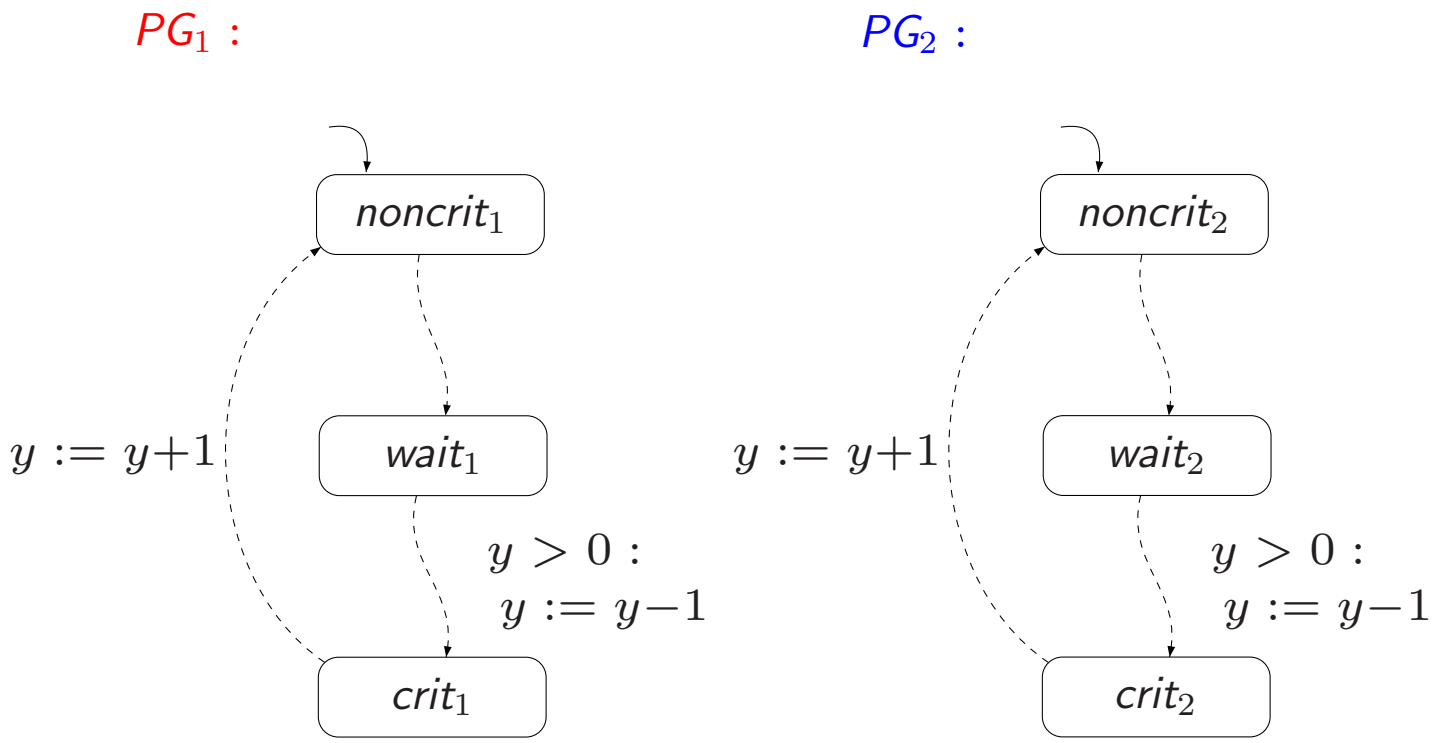
# Linear-time properties

- 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 $AP$):

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

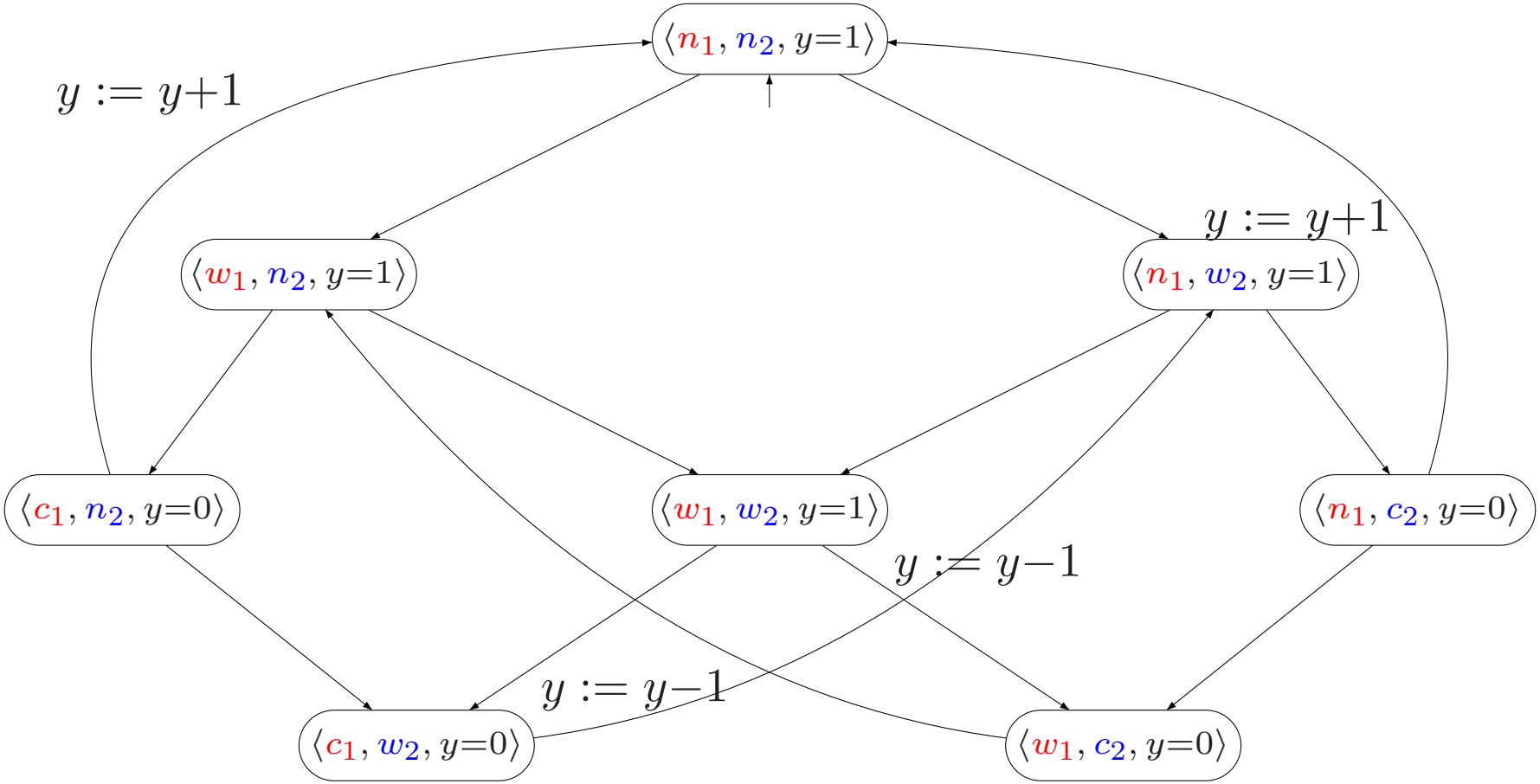
  - $TS$ satisfies the LT property $P$ if all its "observable" behaviors are admissible

# 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 $0 \leqslant i$

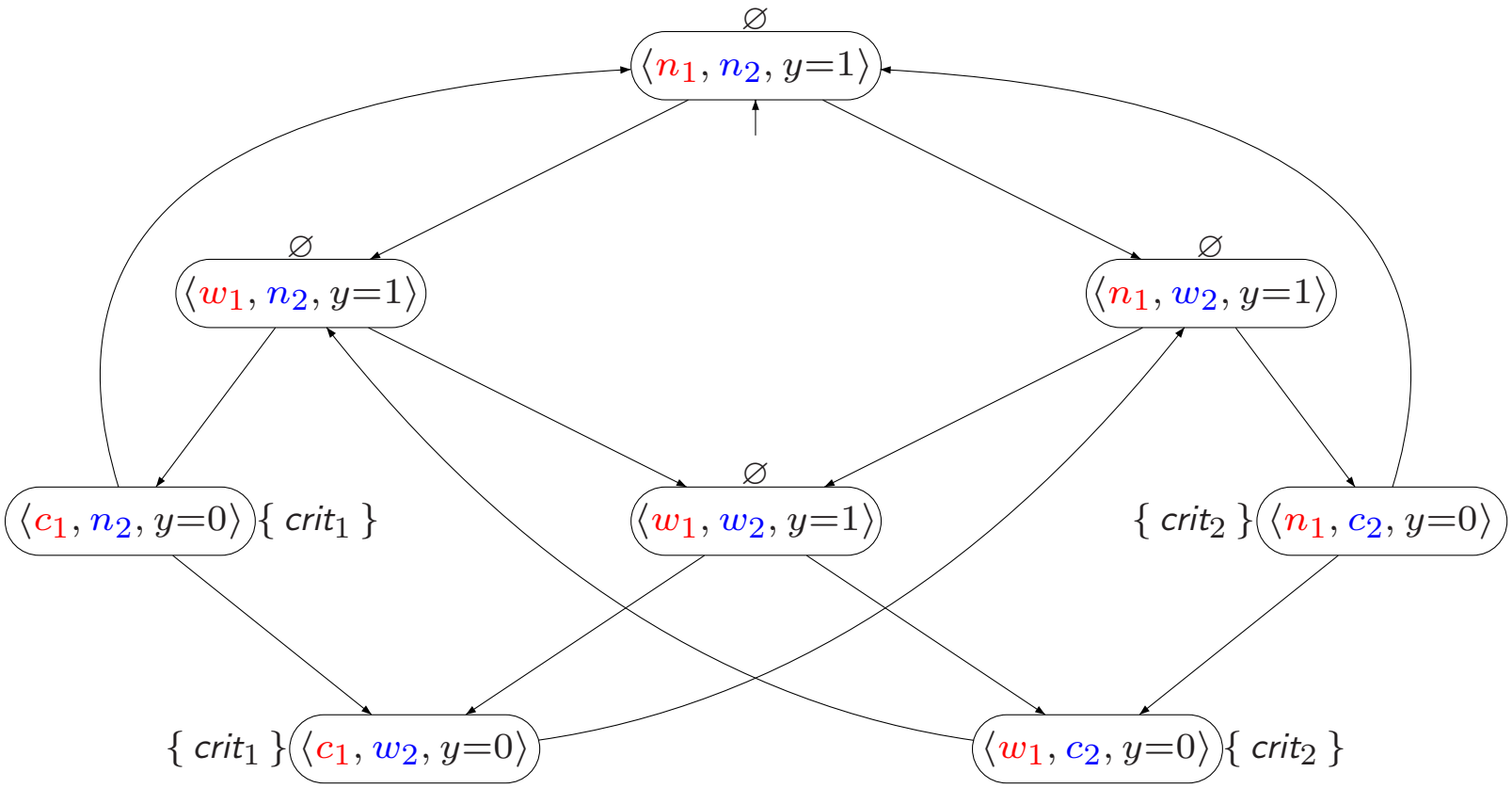
- Contained in $P_{mutex}$ are e.g., the infinite words:

$\{ crit_1 \}\, \{ crit_2 \}\, \{ crit_1 \}\, \{ crit_2 \}\, \{ crit_1 \}\, \{ crit_2 \} \ldots$ 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 $\mathit{BadPref}(P_{safe})$

# 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?

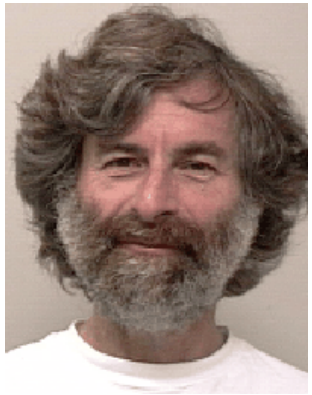
- Safety properties specify that "something bad never happens"
- Doing nothing easily fulfills a safety property
  - as this will never lead to a "bad" situation

$\Rightarrow$ 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

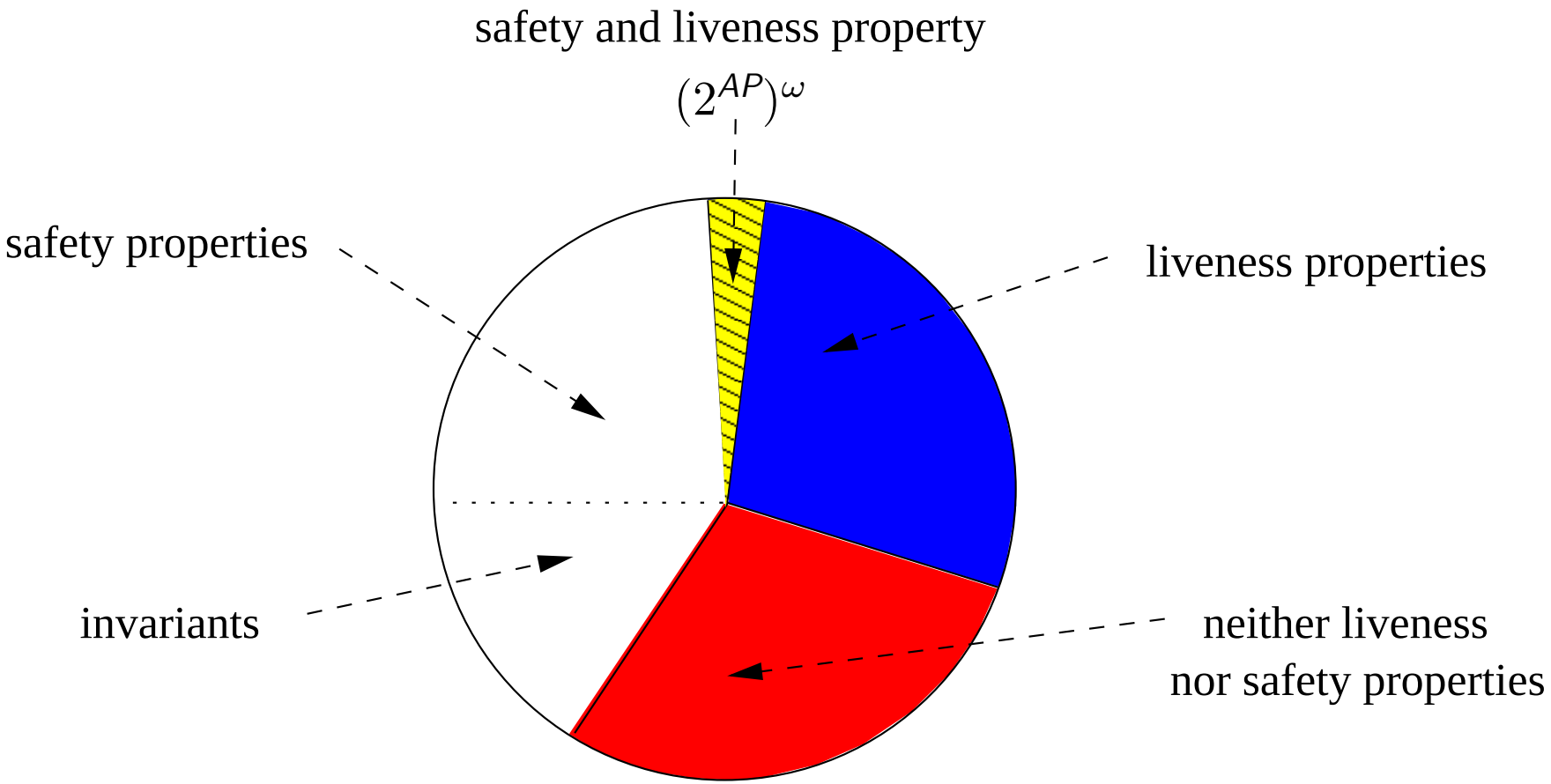
# Decomposition theorem

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

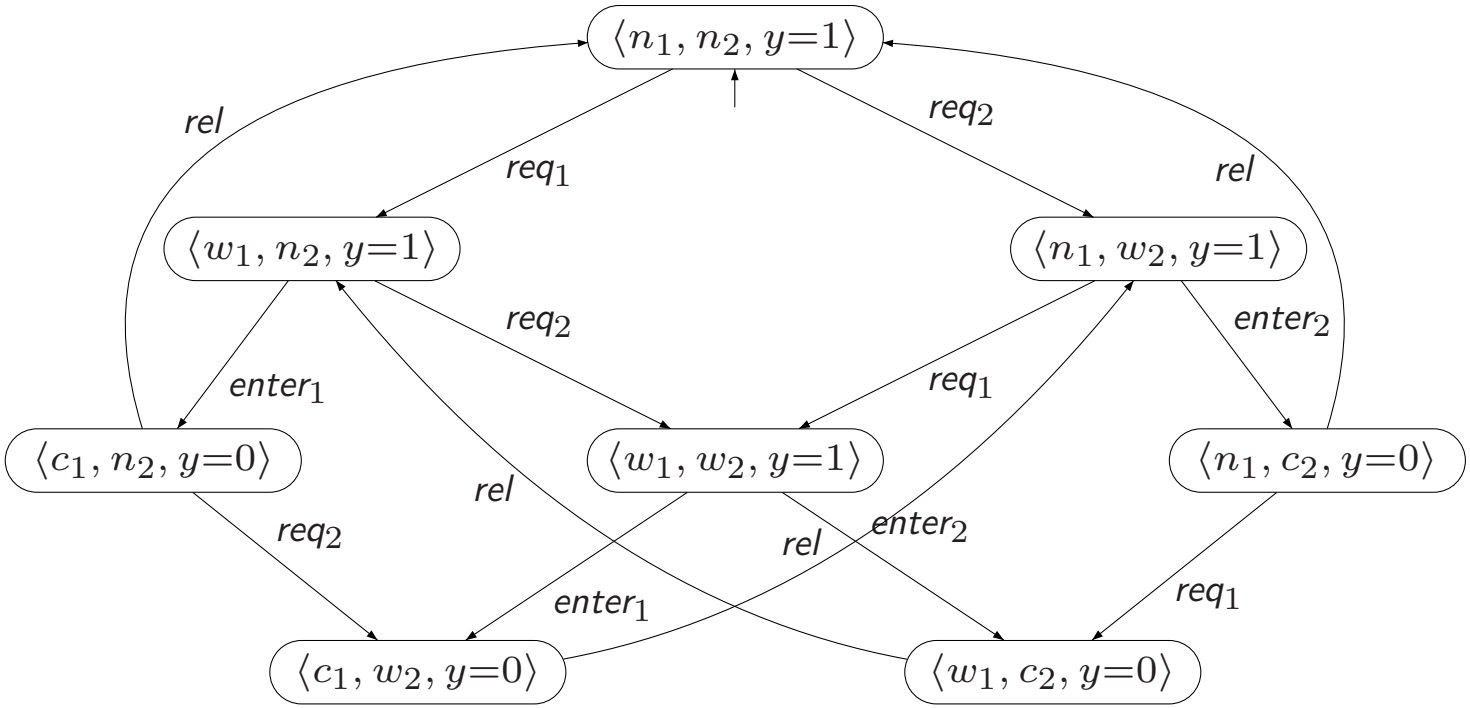
# Does this program always terminate?

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

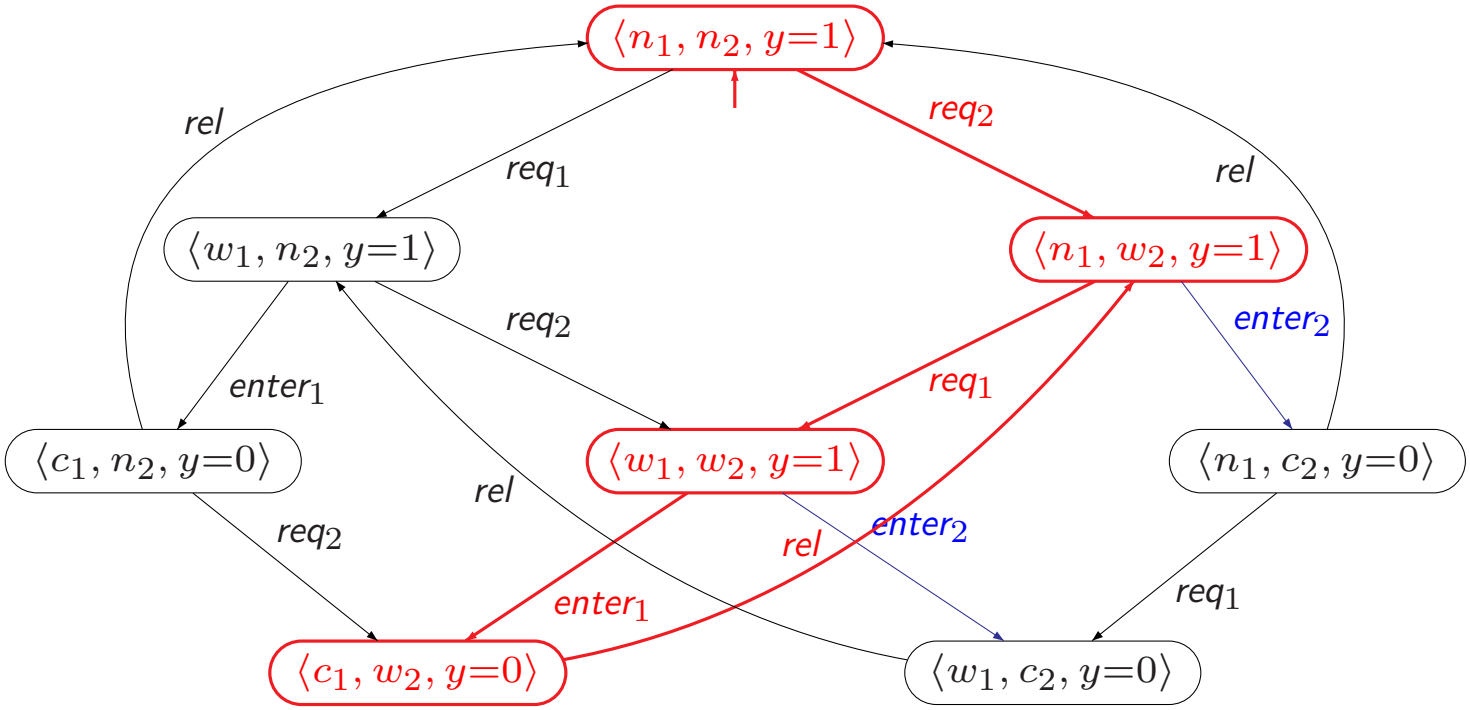
$$
\begin{array}{rcl}
\textit{where} & & \\
\textbf{proc}\ \mathsf{Inc} & = & \textbf{while}\ \langle\, x \geqslant 0\ \textbf{do}\ x := x + 1 \,\rangle\ \textbf{od} \\
\textbf{proc}\ \mathsf{Reset} & = & x := -1
\end{array}
$$

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

# Is it possible to starve?

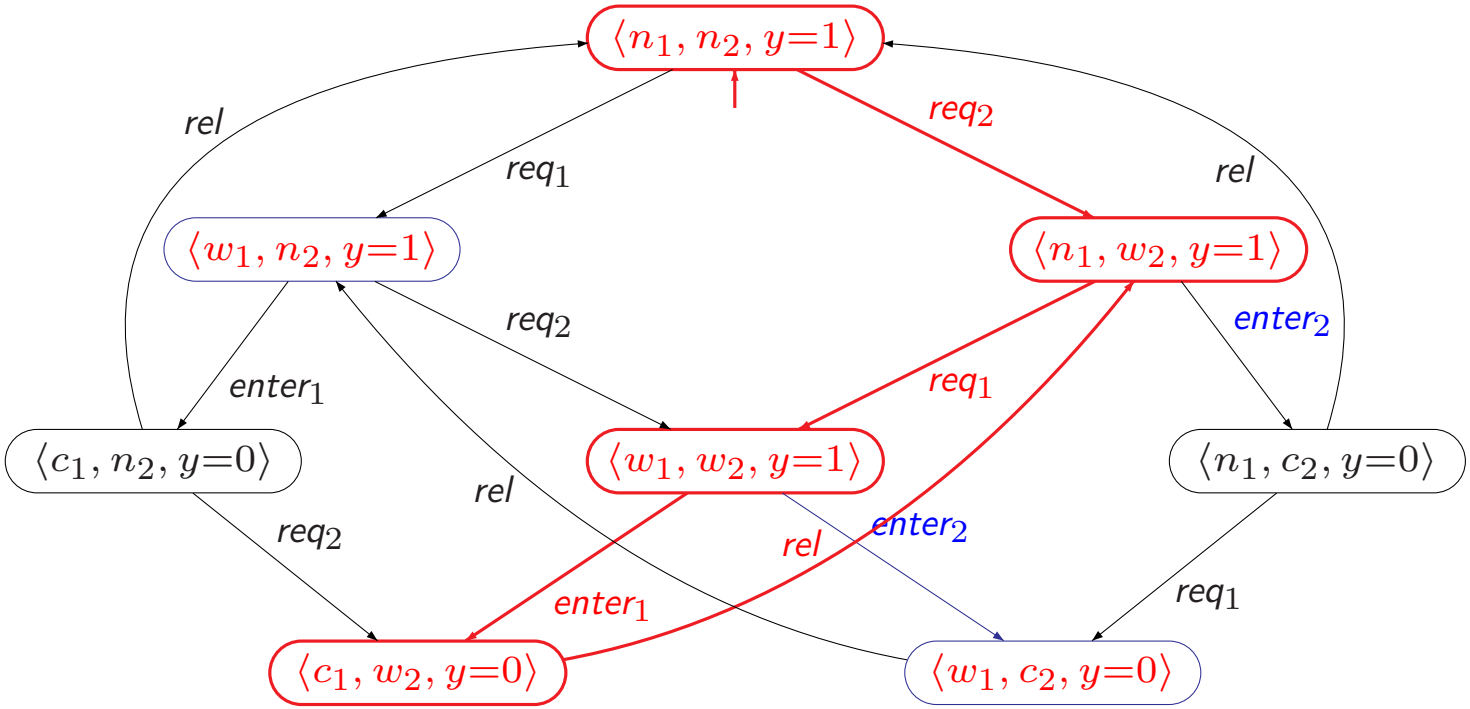


# Process two starves




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

# 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

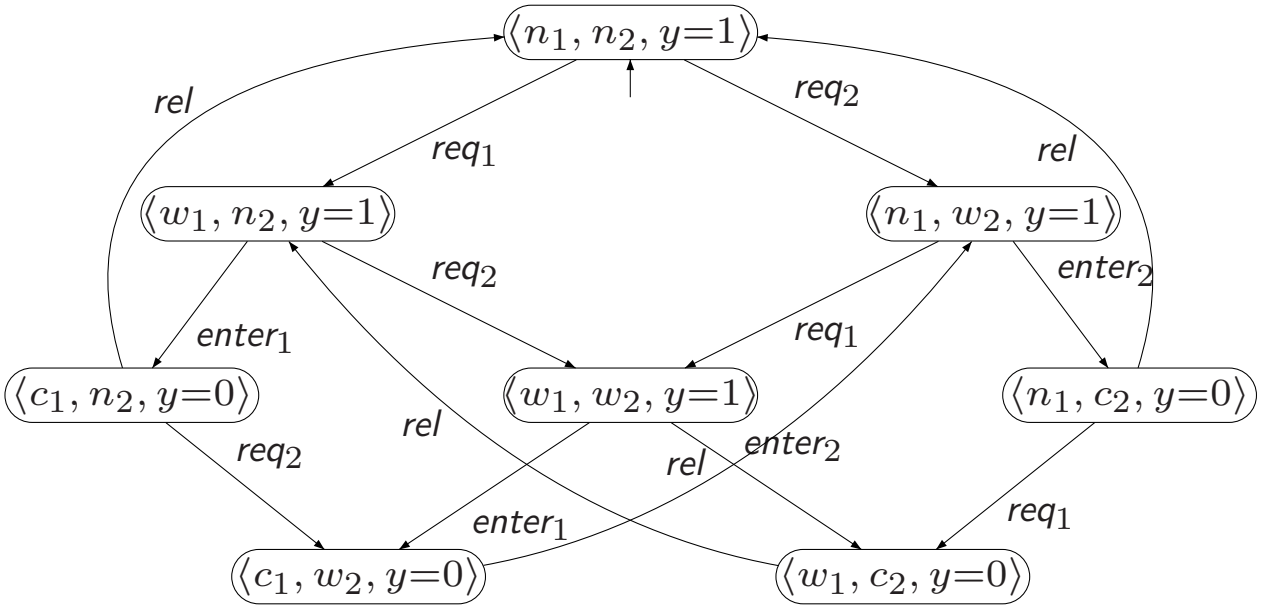
$$\mathcal{F} \;=\; (\mathcal{F}_{ucond}, \mathcal{F}_{strong}, \mathcal{F}_{weak})$$

  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 $A \in \mathcal{F}_{weak}$

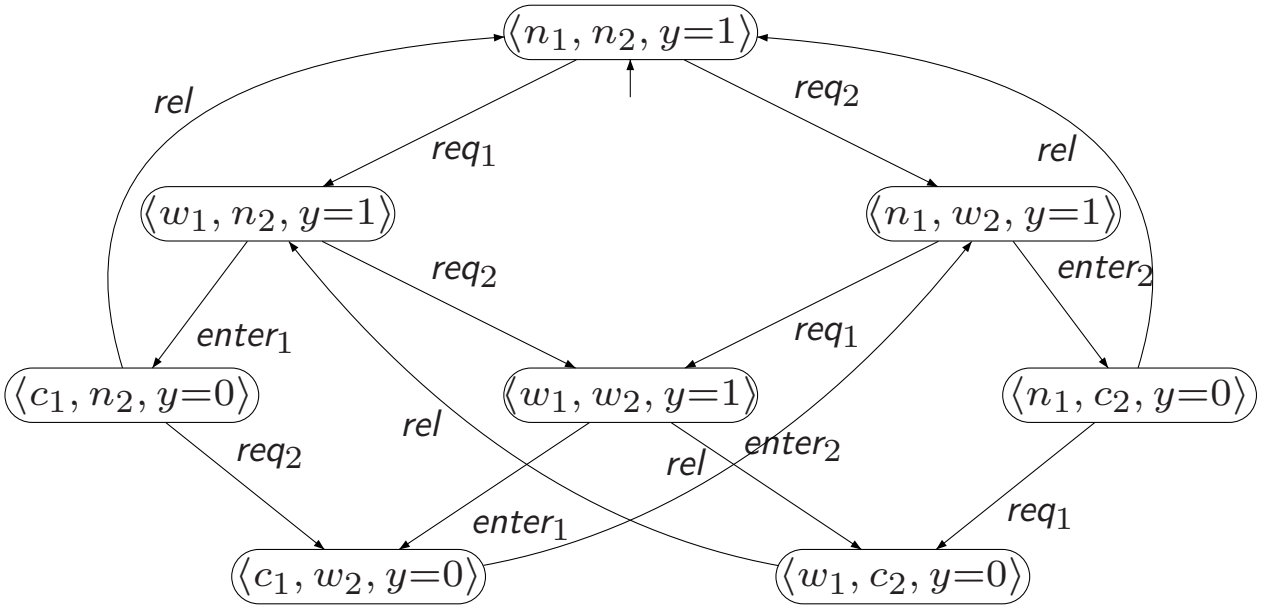
fairness assumption $(\varnothing, \mathcal{F}', \varnothing)$ denotes strong fairness; $(\varnothing, \varnothing, \mathcal{F}')$ weak, etc.

# Fairness for mutual exclusion



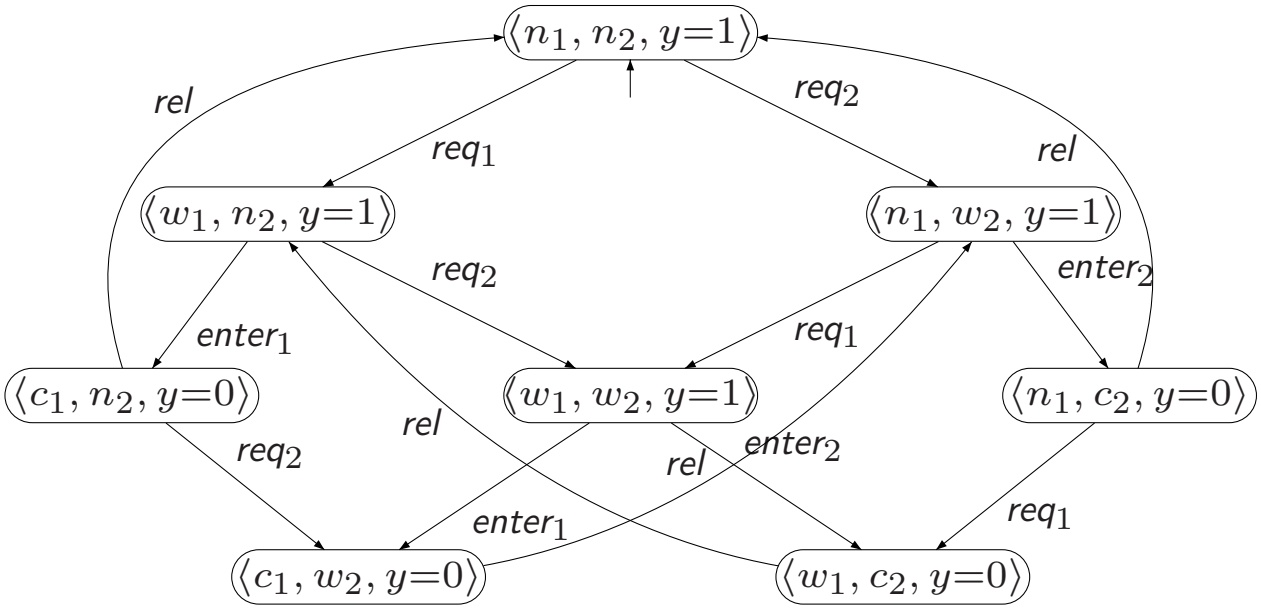

$$\mathcal{F} = (\varnothing, \underbrace{\{\{\, enter_1, enter_2 \,\}\}}_{\mathcal{F}_{strong}}, \varnothing)$$

# 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 $P$:

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

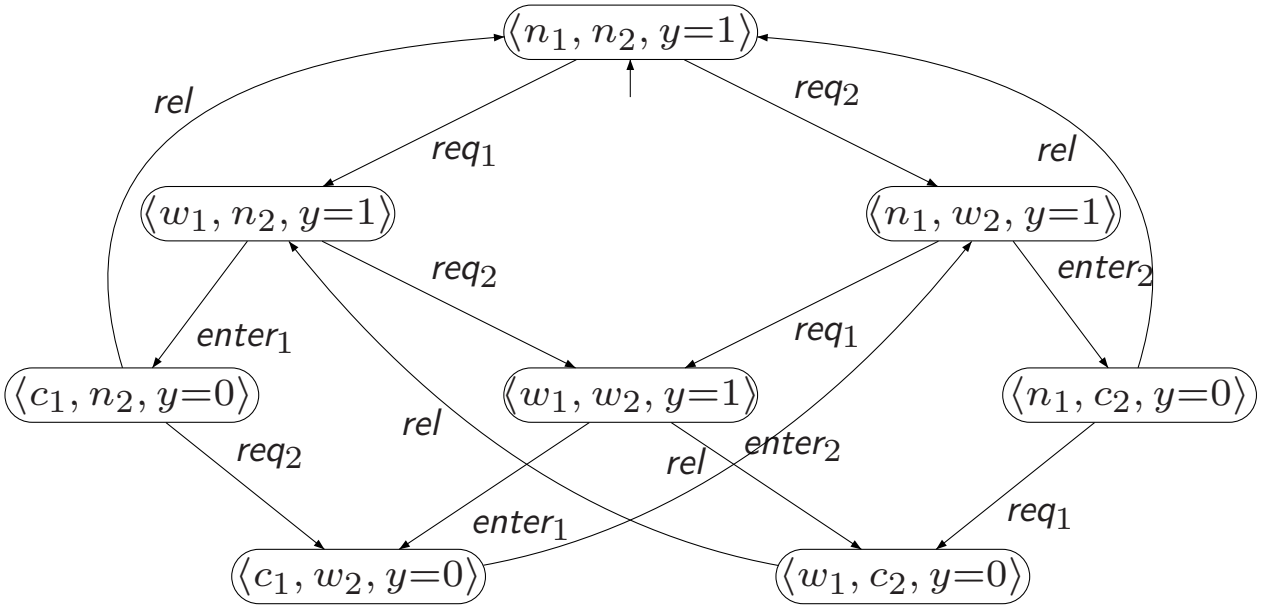
  - *TS* satisfies the LT property $P$ if *all* its observable behaviors are admissible

- *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

# Verifying Regular Linear-Time Properties

## Part #2 of Logic and Verification

*Joost-Pieter Katoen*

Software Modeling and Verification Group

RWTH Aachen University

MOVEP 2014, University of Nantes, July 7, 2014

# Content of this lecture

- Automata on finite words
  - refresh your memory
- Verifying regular safety properties
  - product construction, counterexamples
- Automata on infinite words
  - (generalised) Büchi automata, $\omega$-regular languages
- Verifying $\omega$-regular properties
  - nested depth first search

# Content of this lecture

⇒ Automata on finite words

- refresh your memory

- Verifying regular safety properties
  - product construction, counterexamples

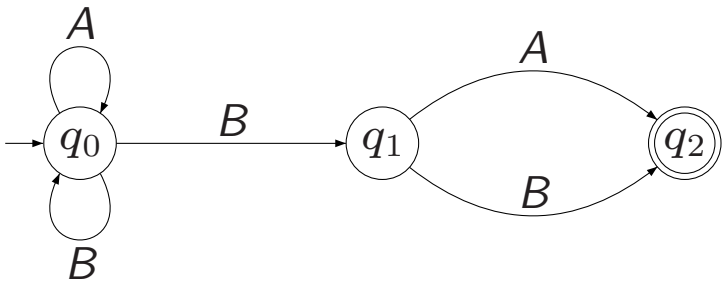
- Automata on infinite words
  - (generalised) Büchi automata, $\omega$-regular languages

- Verifying $\omega$-regular properties
  - nested depth first search

# Refresh your memory: Finite automata

A *nondeterministic finite automaton* (NFA) $\mathcal{A}$ is a tuple $(Q, \Sigma, \delta, Q_0, F)$ where:

- $Q$ is a finite set of states
- $\Sigma$ is an alphabet
- $\delta : Q \times \Sigma \to 2^Q$ is a transition function
- $Q_0 \subseteq Q$ a set of initial states
- $F \subseteq Q$ is a set of accept (or: final) states



# Language of an NFA

- NFA $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ and word $w = A_1 \dots A_n \in \Sigma^*$
- An accepted run for $w$ in $\mathcal{A}$ is a finite sequence $q_0\, q_1\, \dots\, q_n$ such that:
  - $q_0 \in Q_0$ and $q_i \xrightarrow{A_{i+1}} q_{i+1}$ for all $0 \leqslant i < n$, and $q_n \in F$
- $w \in \Sigma^*$ is accepted by $\mathcal{A}$ if there exists an accepting run for $w$
- $\mathcal{L}(\mathcal{A}) = \{ w \in \Sigma^* \mid$ there exists an accepting run for $w$ in $\mathcal{A} \}$
- NFA $\mathcal{A}$ and $\mathcal{A}'$ are *equivalent* if $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$

# Facts about finite automata

- They are as expressive as regular languages
- They are closed under $\cap$ and complementation
  - NFA $\mathcal{A} \otimes B$ (= cross product) accepts $\mathcal{L}(A) \cap \mathcal{L}(B)$
  - Total DFA $\overline{\mathcal{A}}$ (= swap all accept and normal states) accepts $\overline{\mathcal{L}(A)} = \Sigma^* \setminus \mathcal{L}(\mathcal{A})$
- They are closed under determinization (= removal of choice)
  - although at an exponential cost.....
- $\mathcal{L}(\mathcal{A}) = \varnothing?$ = check for a reachable accept state in $\mathcal{A}$
  - this can be done using a simple depth-first search
- For regular language $\mathcal{L}$ there is a unique minimal DFA accepting $\mathcal{L}$

# Content of this lecture

- Automata on finite words
  - refresh your memory

$\Rightarrow$ Verifying regular safety properties
  - product construction, counterexamples

- Automata on infinite words
  - (generalised) Büchi automata, $\omega$-regular languages

- Verifying $\omega$-regular properties
  - nested depth first search

# Safety properties

- LT property $P_{safe}$ over $AP$ is a *safety property* if
  - for all $\sigma \notin P_{safe}$ there exists a finite prefix $\widehat{\sigma}$ of $\sigma$ such that:

$$P_{safe} \cap \left\{ \sigma' \in \left(2^{AP}\right)^{\omega} \mid \widehat{\sigma} \in \mathit{pref}(\sigma) \right\} \;=\; \varnothing$$

- The set *BadPref* of *bad prefixes* for $P_{safe}$:

$$\mathit{BadPref}(P_{safe}) \;=\; \left(2^{AP}\right)^{*} \setminus \mathit{pref}(P_{safe})$$

- The set *MinBadPref* of *minimal bad prefixes* for $P_{safe}$:

$$\mathit{MinBadPref}(P_{safe}) \;=\; \{\, \sigma \in \left(2^{AP}\right)^{*} \mid \mathit{pref}(\sigma) \cap \mathit{BadPref}(P_{safe}) = \{\, \sigma \,\} \,\}$$

# Regular safety properties

- Definition:

  Safety property $P_{safe}$ is regular if *BadPref*$(P_{safe})$ is a regular language

- Or, equivalently:

  Safety property $P_{safe}$ is regular if there exists
  a finite automaton over the alphabet $2^{AP}$ recognizing *BadPref*$(P_{safe})$

# Some regular safety properties

- Every invariant (over $AP$) is a regular safety property
  - bad prefixes have form $\Phi^*(\neg\Phi)\text{true}^*$ for invariant condition $\Phi$
  - . . . where $\Phi$ stands for any $A \subseteq AP$ with $A \models \Phi$
- A regular safety property which is not an invariant:

  "a red light is immediately preceded by a yellow light"
- A non-regular safety property:

  "the number of inserted coins is at least the number of dispensed drinks"

# Peterson's banking system

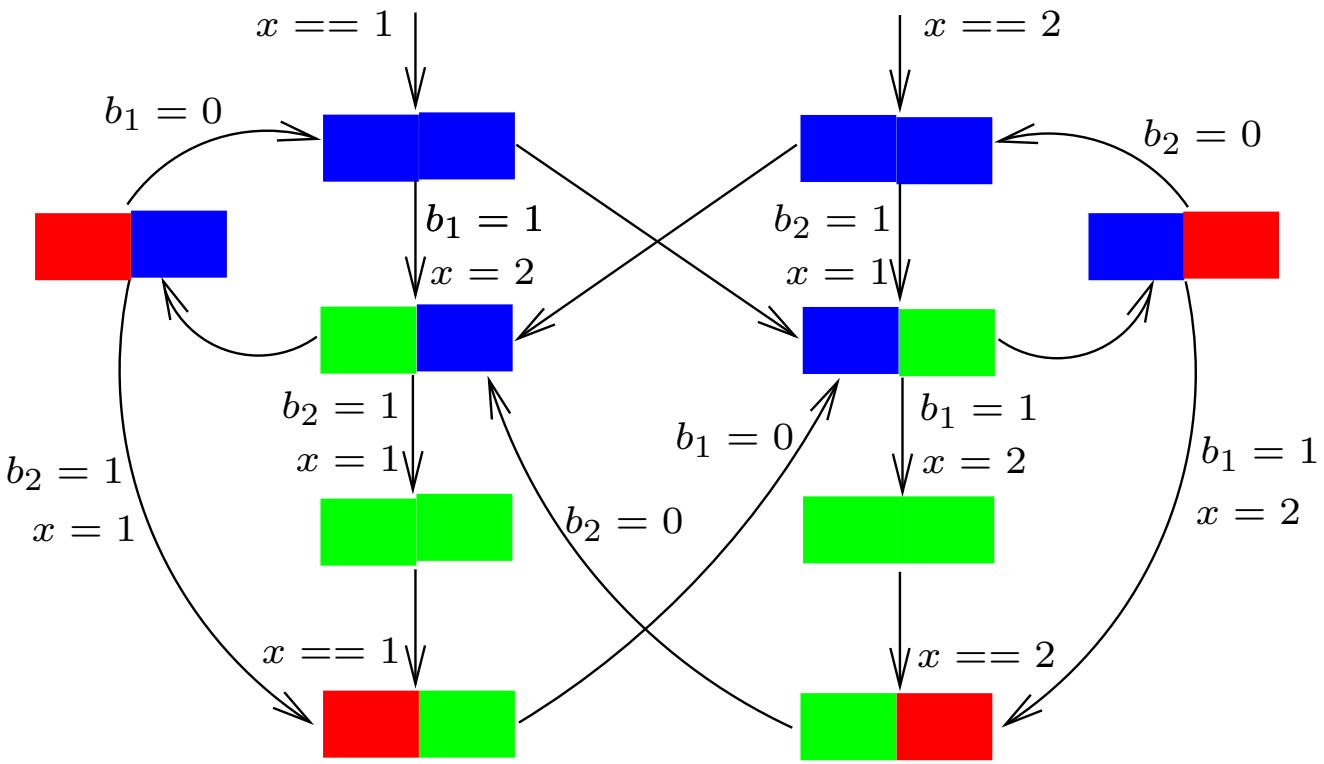
Person Left behaves as follows:

```
          while true {
              ......
rq :          b1, x = true, 2;
wt :          wait until(x == 1 || ¬ b2) {
cs :              ... @accountL ...}
              b1 = false;
              ......
          }
```

Person Right behaves as follows:

```
          while true {
              ......
rq :          b2, x = true, 1;
wt :          wait until(x == 2 || ¬ b1) {
cs :              ... @accountR ...}
              b2 = false;
              ......
          }
```

# Is the banking system safe?




Can we guarantee that only one person at a time has access to the bank account?

"always $\neg$ (@account$_L$ $\wedge$ @account$_R$)"

# Is the banking system safe?

- Safe = at most one person may have access to the account
- Unsafe: two persons have access to the account simultaneously
  - unsafe behaviour can be characterized by bad prefix
  - alternatively (in this case) by the finite automaton:


- $\mathit{Traces}(\mathit{TS}_{Pet}) \cap \mathit{BadPref}(P_{safe}) = \varnothing$?
  - intersection, complementation and emptiness of languages . . .

# Problem statement

Let

- $P_{safe}$ be a *regular* safety property over $AP$
- $\mathcal{A}$ be an NFA recognizing the bad prefixes of $P_{safe}$
  - assume that $\varepsilon \notin \mathcal{L}(\mathcal{A})$

  $\Rightarrow$ otherwise all finite words over $2^{AP}$ are bad prefixes and $P_{safe} = \varnothing$
- $TS$ be a *finite* transition system (over $AP$) without terminal states

How to establish whether $TS \models P_{safe}$?

# Basic idea of the algorithm

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

$$\text{if and only if} \quad \mathit{Traces}_{\mathit{fin}}(TS) \cap \mathcal{L}(\mathcal{A}) = \varnothing$$

$$\text{if and only if} \quad TS \otimes \mathcal{A} \models \text{“always”}\ \Phi$$

*But . . . . . this amounts to invariant checking on* $TS \otimes \mathcal{A}$

$\Rightarrow$ *checking regular safety properties can be done by depth-first search!*

# Synchronous product

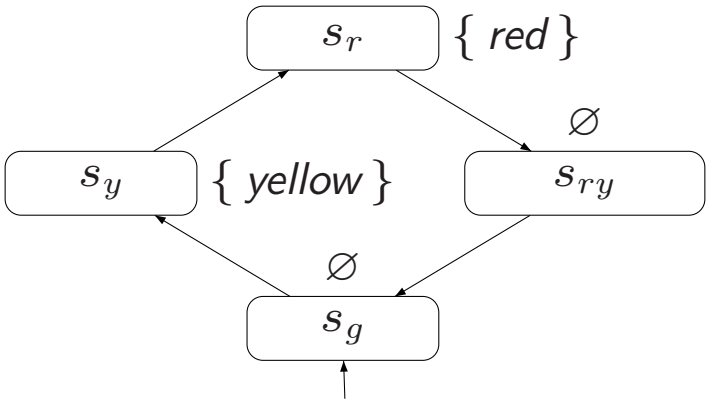
For transition system $TS = (S, Act, \rightarrow, I, AP, L)$ without terminal states and $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ an NFA with $\Sigma = 2^{AP}$ and $Q_0 \cap F = \varnothing$, let:

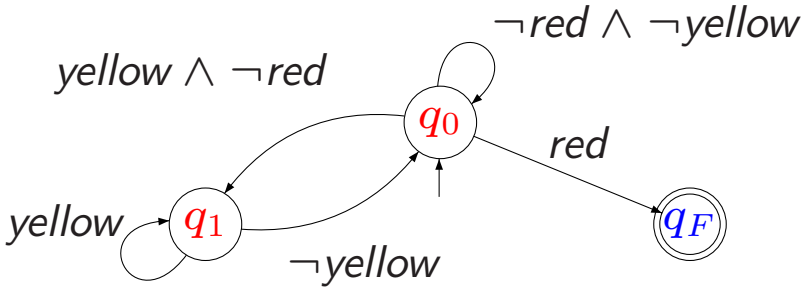
$$TS \otimes \mathcal{A} \;=\; (S', Act, \rightarrow', I', AP', L') \qquad \text{where}$$

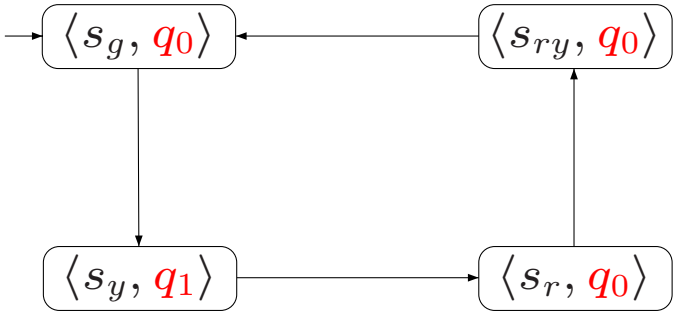
- $S' = S \times Q$, $AP' = Q$ and $L'(\langle s, q \rangle) \;=\; \{\, q \,\}$
- $\rightarrow'$ is the smallest relation defined by: $\dfrac{s \xrightarrow{\alpha} t \;\wedge\; q \xrightarrow{L(t)} p}{\langle s, q \rangle \xrightarrow{\alpha}' \langle t, p \rangle}$
- $I' = \{\, \langle s_0, q \rangle \mid s_0 \in I \;\wedge\; \exists q_0 \in Q_0.\; q_0 \xrightarrow{L(s_0)} q \,\}$

*without loss of generality it may be assumed that $TS \otimes \mathcal{A}$ has no terminal states*

# Example product



# Verification of regular safety properties

Let $TS$ over $AP$, NFA $\mathcal{A}$, and $P$ a regular safety property with $\mathcal{L}(\mathcal{A}) = \mathit{BadPref}(P)$

The following statements are equivalent:

(a) $TS \models P$

(b) $\mathit{Traces}_{\mathit{fin}}(TS) \cap \mathcal{L}(\mathcal{A}) = \varnothing$

(c) $TS \otimes \mathcal{A} \models P_{inv(A)} = \bigwedge_{q \in F} \neg\, q$

# Counterexamples

For each initial path fragment $\langle s_0, q_1 \rangle \ldots \langle s_n, q_{n+1} \rangle$ of $TS \otimes \mathcal{A}$:

$$q_1, \ldots, q_n \notin F \text{ and } q_{n+1} \in F \quad \Rightarrow \quad \underbrace{trace(s_0\, s_1 \ldots s_n)}_{\text{bad prefix for } P_{safe}} \in \mathcal{L}(\mathcal{A})$$

# Time complexity

The time and space complexity of checking $TS \models P_{safe}$ is in:

$$\mathcal{O}(|TS| \cdot |\mathcal{A}|)$$

where $\mathcal{A}$ is an NFA with $\mathcal{L}(\mathcal{A}) = \mathit{MinBadPref}(P_{safe})$

# Content of this lecture

- Automata on finite words
  - refresh your memory
- Verifying regular safety properties
  - product construction, counterexamples

$\Rightarrow$ Automata on infinite words

  - (generalised) Büchi automata, $\omega$-regular languages
- Verifying $\omega$-regular properties
  - nested depth first search

# Peterson's banking system

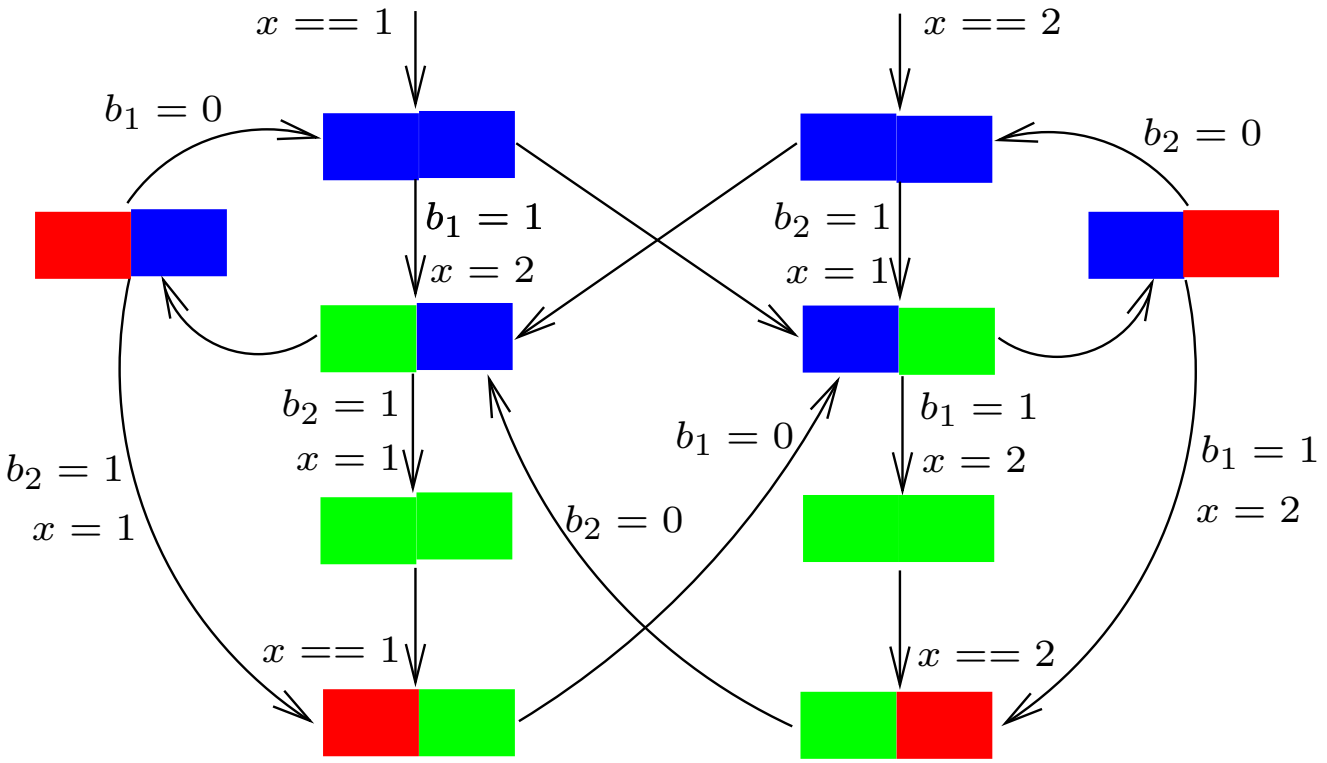
Person Left behaves as follows:

```
          while true {
              ......
rq :          b1, x = true, 2;
wt :          wait until(x == 1 || ¬ b2) {
cs :              ... @accountL ...}
              b1 = false;
              ......
          }
```

Person Right behaves as follows:

```
          while true {
              ......
rq :          b2, x = true, 1;
wt :          wait until(x == 2 || ¬ b1) {
cs :              ... @accountR ...}
              b2 = false;
              ......
          }
```

# Is the banking system live?

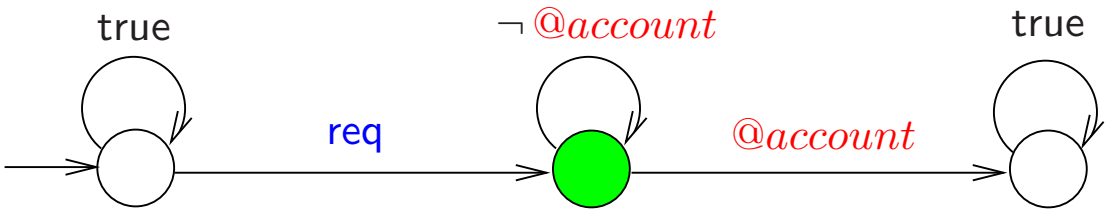



If someone wants to update the account, does he ever get the opportunity to do so?

"always ($req_L$ $\Rightarrow$ eventually @account$_L$) $\wedge$ always ($req_R$ $\Rightarrow$ eventually @account$_R$)"

# Is the banking system live?

- Live = when you want access to account, you eventually get it
- Not live: once you want access to the account, you never get it
  - unlive behaviour can be characterized as a (set of) infinite traces
  - or, equivalently, by a Büchi-automaton *Live*:




- Checking liveness: $\mathit{Traces}(\mathit{TS}_{Pet}) \cap \mathcal{L}_\omega(\overline{\mathit{Live}}) = \varnothing$?
  - (explicit) complementation, intersection and emptiness of Büchi automata!

# Syntax of $\omega$-regular expressions

- Regular expressions denote languages of finite words
- $\omega$-Regular expressions denote languages of infinite words
- An $\omega$-*regular expression* G over $\Sigma$ has the form:

$$\mathsf{G} \;=\; \mathsf{E}_1.\mathsf{F}_1^{\omega} \;+\; \ldots \;+\; \mathsf{E}_n.\mathsf{F}_n^{\omega} \quad \text{for } n > 0$$

where $\mathsf{E}_i, \mathsf{F}_i$ are regular expressions over $\Sigma$ with $\varepsilon \notin \mathcal{L}(\mathsf{F}_i)$

- Some examples: $(A + B)^*.B^{\omega}$, $(B^*.A)^{\omega}$, and $A^*.B^{\omega} + A^{\omega}$

# Semantics of $\omega$-regular expressions

- For $\mathcal{L} \subseteq \Sigma^*$ let $\mathcal{L}^\omega = \{ w_1 w_2 w_3 \ldots \mid \forall i \geqslant 0.\, w_i \in \mathcal{L} \}$
- Let $\omega$-regular expression $\mathsf{G} = \mathsf{E}_1.\mathsf{F}_1^\omega + \ldots + \mathsf{E}_n.\mathsf{F}_n^\omega$
- The *semantics* of G is a language $\mathcal{L}(\mathsf{G}) \subseteq \Sigma^\omega$:

$$\mathcal{L}_\omega(\mathsf{G}) = \mathcal{L}(\mathsf{E}_1).\mathcal{L}(\mathsf{F}_1)^\omega \cup \ldots \cup \mathcal{L}(\mathsf{E}_n).\mathcal{L}(\mathsf{F}_n)^\omega$$

- $\mathsf{G}_1$ and $\mathsf{G}_2$ are *equivalent*, denoted $\mathsf{G}_1 \equiv \mathsf{G}_2$, if $\mathcal{L}_\omega(\mathsf{G}_1) = \mathcal{L}_\omega(\mathsf{G}_2)$

# $\omega$-Regular languages

- $\mathcal{L}$ is $\omega$-regular if $\mathcal{L} = \mathcal{L}_\omega(\mathsf{G})$ for some $\omega$-regular expression G
- Examples over $\Sigma = \{ A, B \}$:
  - language of all words with infinitely many $A$s:

$$(B^*.A)^\omega$$

  - language of all words with finitely many $A$s:

$$(A + B)^*.B^\omega$$

  - the empty language

$$\varnothing^\omega$$

- $\omega$-Regular languages are closed under $\cup$, $\cap$, and complementation

# $\omega$-regular properties

- Definition:

  LT property $P$ over $AP$ is $\omega$-regular if

  $P$ is an $\omega$-regular language over the alphabet $2^{AP}$

- Or, equivalently:

  LT property $P$ over $AP$ is $\omega$-regular if $P$ is a language accepted by a nondeterministic Büchi automaton over $2^{AP}$

# Example $\omega$-regular properties

- Any invariant $P$ is an $\omega$-regular property
  - as $\Phi^\omega$ describes $P$ with invariant condition $\Phi$
- Any regular safety property $P$ is an $\omega$-regular property
  - as $\overline{P} = \mathit{BadPref}(P) . \left(2^{AP}\right)^\omega$ is $\omega$-regular
  - and the fact that $\omega$-regular languages are closed under complement
- Many liveness properties $P$ are $\omega$-regular properties

# Nondeterministic Büchi automata

- NFA (and DFA) are incapable of accepting infinite words
- Automata on infinite words
  - suited for accepting $\omega$-regular languages
  - we consider nondeterministic Büchi automata (NBA)
- Accepting runs have to "check" the entire input word $\Rightarrow$ are infinite

  $\Rightarrow$ acceptance criteria for infinite runs are needed
- NBA are like NFA, but have a distinct *acceptance criterion*
  - one of the accept states must be visited infinitely often

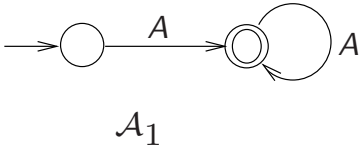
# Language of an NBA

- NBA $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ and word $\sigma = A_0 A_1 A_2 \ldots \in \Sigma^\omega$
- An accepted run for $\sigma$ in $\mathcal{A}$ is an infinite sequence $q_0\, q_1\, q_2 \ldots$ such that:
  - $q_0 \in Q_0$ and $q_i \xrightarrow{A_{i+1}} q_{i+1}$ for all $0 \leqslant i$, and
  - $q_i \in F$ for infinitely many $i$
- $\sigma \in \Sigma^\omega$ is accepted by $\mathcal{A}$ if there exists an accepting run for $\sigma$
- The accepted language of $\mathcal{A}$:

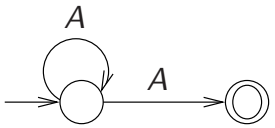
$$\mathcal{L}_\omega(\mathcal{A}) = \left\{ \sigma \in \Sigma^\omega \mid \text{ there exists an accepting run for } \sigma \text{ in } \mathcal{A} \right\}$$

- NBA $\mathcal{A}$ and $\mathcal{A}'$ are *equivalent* if $\mathcal{L}_\omega(\mathcal{A}) = \mathcal{L}_\omega(\mathcal{A}')$

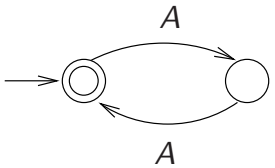
# NBA versus NFA



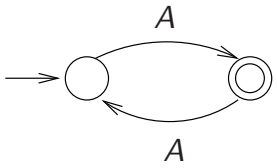

$\mathcal{A}_1$




$\mathcal{A}_2$




$\mathcal{A}_1$




$\mathcal{A}_2$

finite equivalence $\not\Rightarrow$ $\omega$-equivalence

$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$, but $\mathcal{L}_\omega(\mathcal{A}) \neq \mathcal{L}_\omega(\mathcal{A}')$

$\omega$-equivalence $\not\Rightarrow$ finite equivalence

$\mathcal{L}_\omega(\mathcal{A}) = \mathcal{L}_\omega(\mathcal{A}')$, but $\mathcal{L}(\mathcal{A}) \neq \mathcal{L}(\mathcal{A}')$

# NBA and $\omega$-regular languages

The class of languages accepted by NBA

agrees with the class of $\omega$-regular languages

*(1) any $\omega$-regular language is recognized by an NBA*

*(2) for any NBA $\mathcal{A}$, the language $\mathcal{L}_\omega(\mathcal{A})$ is $\omega$-regular*

# Checking non-emptiness

$$\mathcal{L}_\omega(\mathcal{A}) \neq \varnothing$$

if and only if

$$\underbrace{\exists q_0 \in Q_0.\ \exists q \in F.\ \exists w \in \Sigma^*.\ \exists v \in \Sigma^+.\ \ q \in \delta^*(q_0, w) \ \wedge \ q \in \delta^*(q, v)}$$

there is a reachable accept state on a cycle

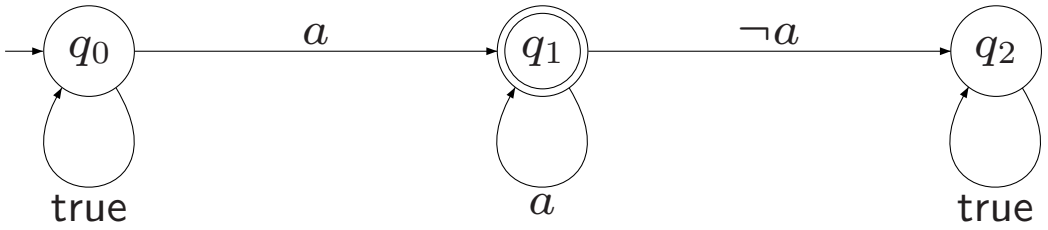
*The emptiness problem for NBA $\mathcal{A}$ can be solved in $\mathcal{O}(|\mathcal{A}|)$*

# NBA are more expressive than DBA

NFA and DFA are equally expressive but NBA and DBA are not!

There is no DBA that accepts $\mathcal{L}_\omega((A+B)^*B^\omega)$

# An LT property requiring nondeterminism




let $\{ a \} = AP$, i.e., $2^{AP} = \{A, B\}$ where $A = \{\}$ and $B = \{a\}$

"eventually for ever $a$" equals $(A + B)^* B^\omega \;=\; (\{\} + \{a\})^* \{a\}^\omega$

# Content of this lecture

- Automata on finite words
  - refresh your memory
- Verifying regular safety properties
  - product construction, counterexamples
- Automata on infinite words
  - (generalised) Büchi automata, $\omega$-regular languages

$\Rightarrow$ Verifying $\omega$-regular properties

  - nested depth first search

# $\omega$-regular properties

- Definition:

  LT property $P$ over $AP$ is $\omega$-regular if

  $P$ is an $\omega$-regular language over the alphabet $2^{AP}$

- Or, equivalently:

  LT property $P$ over $AP$ is $\omega$-regular if $P$ is a language accepted by a nondeterministic Büchi automaton over $2^{AP}$

# Basic idea of the algorithm

$TS \not\models P$ if and only if $\mathit{Traces}(TS) \not\subseteq P$

if and only if $\mathit{Traces}(TS) \cap \left(2^{AP}\right)^{\omega} \setminus P \neq \varnothing$

if and only if $\mathit{Traces}(TS) \cap \overline{P} \neq \varnothing$

if and only if $\mathit{Traces}(TS) \cap \mathcal{L}_{\omega}(\mathcal{A}) \neq \varnothing$

if and only if $TS \otimes \mathcal{A} \not\models \underbrace{\text{“eventually for ever” } \neg F}_{\text{persistence property}}$

where $\mathcal{A}$ is an NBA accepting the complement property $\overline{P} = \left(2^{AP}\right)^{\omega} \setminus P$

# Persistence property

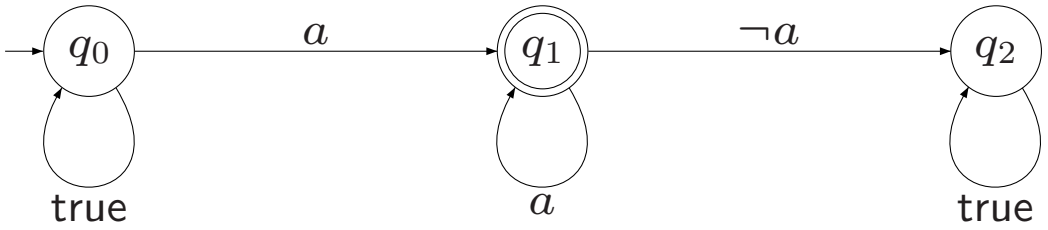
A *persistence property* over $AP$ is an LT property $P_{pers} \subseteq \left(2^{AP}\right)^{\omega}$ "eventually for ever $\Phi$" for some propositional logic formula $\Phi$ over $AP$:

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

$\Phi$ is called a persistence (or state) condition of $P_{pers}$

"$\Phi$ is an invariant after a while"

# Example persistence property




let $\{ a \} = AP$, i.e., $2^{AP} = \{A, B\}$ where $A = \{\}$ and $B = \{a\}$

"eventually for ever $a$" equals $(A + B)^* B^\omega \;=\; (\{\} + \{a\})^* \{a\}^\omega$

# Recall synchronous product

For transition system $TS = (S, Act, \rightarrow, I, AP, L)$ without terminal states and $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ a non-blocking NBA with $\Sigma = 2^{AP}$, let:

$$TS \otimes \mathcal{A} \;=\; (S', Act, \rightarrow', I', AP', L') \qquad \text{where}$$

- $S' = S \times Q$, $AP' = Q$ and $L'(\langle s, q \rangle) \;=\; \{\, q \,\}$
- $\rightarrow'$ is the smallest relation defined by: $\dfrac{s \xrightarrow{\alpha} t \;\wedge\; q \xrightarrow{L(t)} p}{\langle s, q \rangle \xrightarrow{\alpha}{}' \langle t, p \rangle}$
- $I' = \{\, \langle s_0, q \rangle \mid s_0 \in I \;\wedge\; \exists q_0 \in Q_0.\; q_0 \xrightarrow{L(s_0)} q \,\}$

# Verifying $\omega$-regular properties

Let:

- $TS$ be a transition system over $AP$
- $P$ be an $\omega$-regular property over $AP$, and
- $\mathcal{A}$ a non-blocking NBA such that $\mathcal{L}_\omega(\mathcal{A}) = \overline{P}$.

The following statements are equivalent:

(a) $TS \models P$

(b) $Traces(TS) \cap \mathcal{L}_\omega(\mathcal{A}) = \varnothing$

(c) $TS \otimes \mathcal{A} \models P_{pers(A)}$

*where* $P_{pers(A)}$ = *"eventually for ever* $\neg F$*"*

$\Rightarrow$ *checking $\omega$-regular properties is reduced to persistence checking!*

# Persistence checking and cycle detection

Let

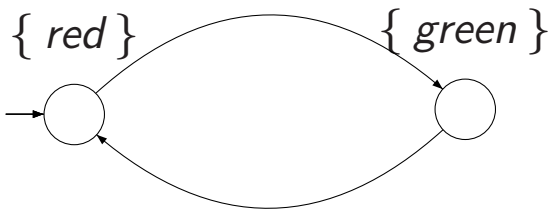
- $TS$ be a finite transition system without terminal states over $AP$
- $\Phi$ a propositional formula over $AP$, and
- $P_{pers}$ the persistence property "eventually for ever $\Phi$"

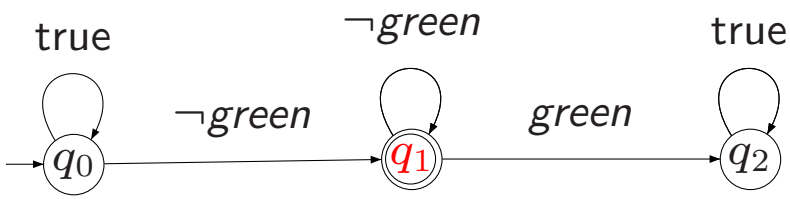
$$TS \not\models P_{pers}$$

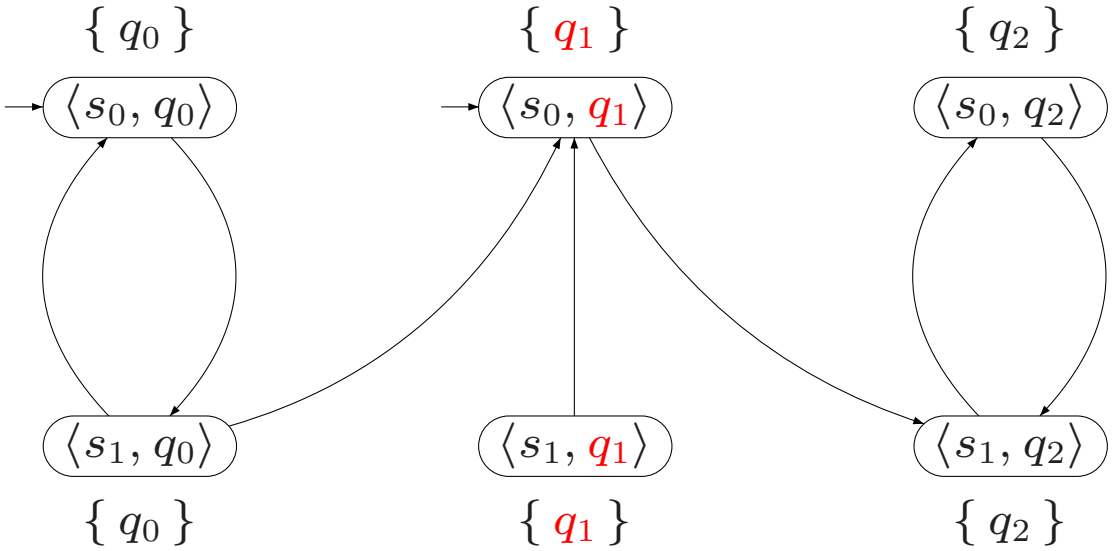
if and only if

$$\exists s \in \mathit{Reach}(TS).\ s \not\models \Phi \ \wedge\ s \text{ is on a cycle in } G(TS)$$

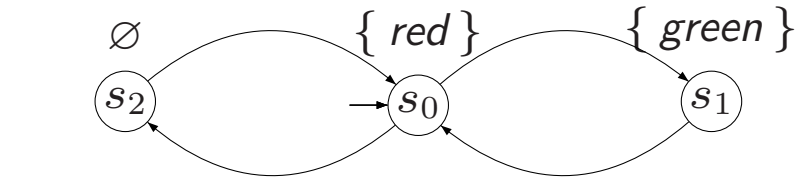
# Infinitely often green?

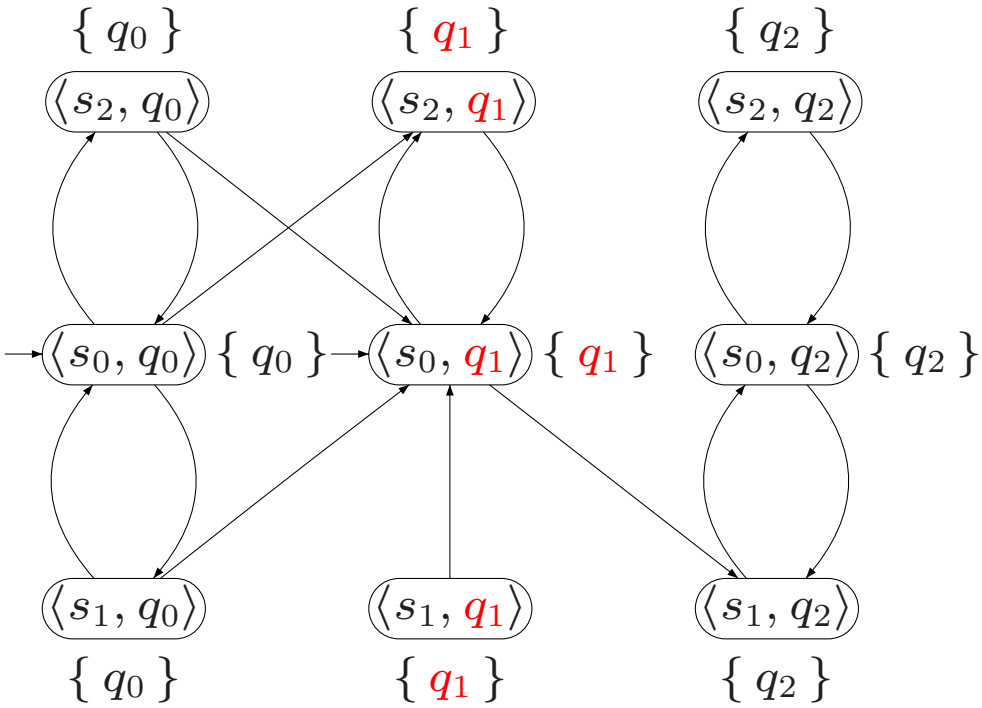


# Infinitely often green?



# Cycle detection

How to check for a reachable cycles containing a $\neg\Phi$-state?

- Alternative 1:
  - compute the strongly connected components (SCCs) in $G(TS)$
  - check whether one such SCC is reachable from an initial state
  - . . . that contains a $\neg\Phi$-state
  - "eventually for ever $\Phi$" is refuted if and only if such SCC is found

- Alternative 2:
  - *use a nested depth-first search*

  $\Rightarrow$ more adequate for an on-the-fly verification algorithm

  $\Rightarrow$ easier for generating counterexamples

# Nested depth-first search

- Idea: perform the two depth-first searches in an *interleaved* way
  - the outer DFS serves to encounter all reachable $\neg\Phi$-states
  - the inner DFS seeks for backward edges leading to a $\neg\Phi$-state
- *Nested DFS*
  - on full expansion of $\neg\Phi$-state $s$ in the outer DFS, start inner DFS
  - in inner DFS, visit all states reachable from $s$ *not visited* in the inner DFS yet
  - no backward edge found to $s$? continue the outer DFS (look for next $\neg\Phi$ state)
- *Counterexample generation*: DFS stack concatenation
  - stack $U$ for the outer DFS $=$ path fragment from $s_0 \in I$ to $s$ (in reversed order)
  - stack $V$ for the inner DFS $=$ a cycle from state $s$ to $s$ (in reversed order)

# Correctness of nested DFS

Let:

- $TS$ be a finite transition system over $AP$ without terminal states and
- $P_{pers}$ a persistence property

The nested DFS algorithm yields "no" if and only if $TS \not\models P_{pers}$

# Time complexity

The worst-case time complexity of nested DFS is in

$$\mathcal{O}((N+M) + N \cdot |\, \Phi \,|)$$

where $N$ is $\#$ reachable states in $TS$, and $M$ is $\#$ transitions in $TS$

# LTL Model Checking

## Part #3 of Logic and Verification

*Joost-Pieter Katoen*

Software Modeling and Verification Group

RWTH Aachen University

MOVEP 2014, University of Nantes, July 7, 2014

# Content of this lecture

- Linear temporal logic
  - syntax, semantics, specifying properties
- Equivalences and fairness
  - weak until, fairness in LTL
- LTL model checking
  - GNBA, from LTL to GNBA, complexity

# Content of this lecture

⇒ Linear temporal logic

- syntax, semantics, specifying properties

- Equivalences and fairness
  - weak until, fairness in LTL

- LTL model checking
  - GNBA, from LTL to GNBA, complexity

# LT properties

- An LT property is a set of infinite traces over $AP$
- Specifying such sets explicitly is often inconvenient
- Mutual exclusion is specified over $AP = \{ c_1, c_2 \}$ by

$$P_{mutex} = \text{set of infinite words } A_0\, A_1\, A_2 \ldots \text{ with } \{ c_1, c_2 \} \not\subseteq A_i \text{ for all } 0 \leqslant i$$

- Starvation freedom is specified over $AP = \{ c_1, w_1, c_2, w_2 \}$ by

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

$$\left(\overset{\infty}{\exists}\, j.\ w_1 \in A_j\right) \Rightarrow \left(\overset{\infty}{\exists}\, j.\ c_1 \in A_j\right) \wedge \left(\overset{\infty}{\exists}\, j.\ w_2 \in A_j\right) \Rightarrow \left(\overset{\infty}{\exists}\, j.\ c_2 \in A_j\right)$$

such properties can be specified succinctly using logic

# Linear Temporal Logic: Syntax

- Propositional logic
  - $a \in AP$ — atomic proposition
  - $\neg\phi$ and $\phi \wedge \psi$ — negation and conjunction
- Temporal operators
  - $\bigcirc \phi$ — neXt state fulfills $\phi$
  - $\phi \,\mathsf{U}\, \psi$ — $\phi$ holds Until a $\psi$-state is reached

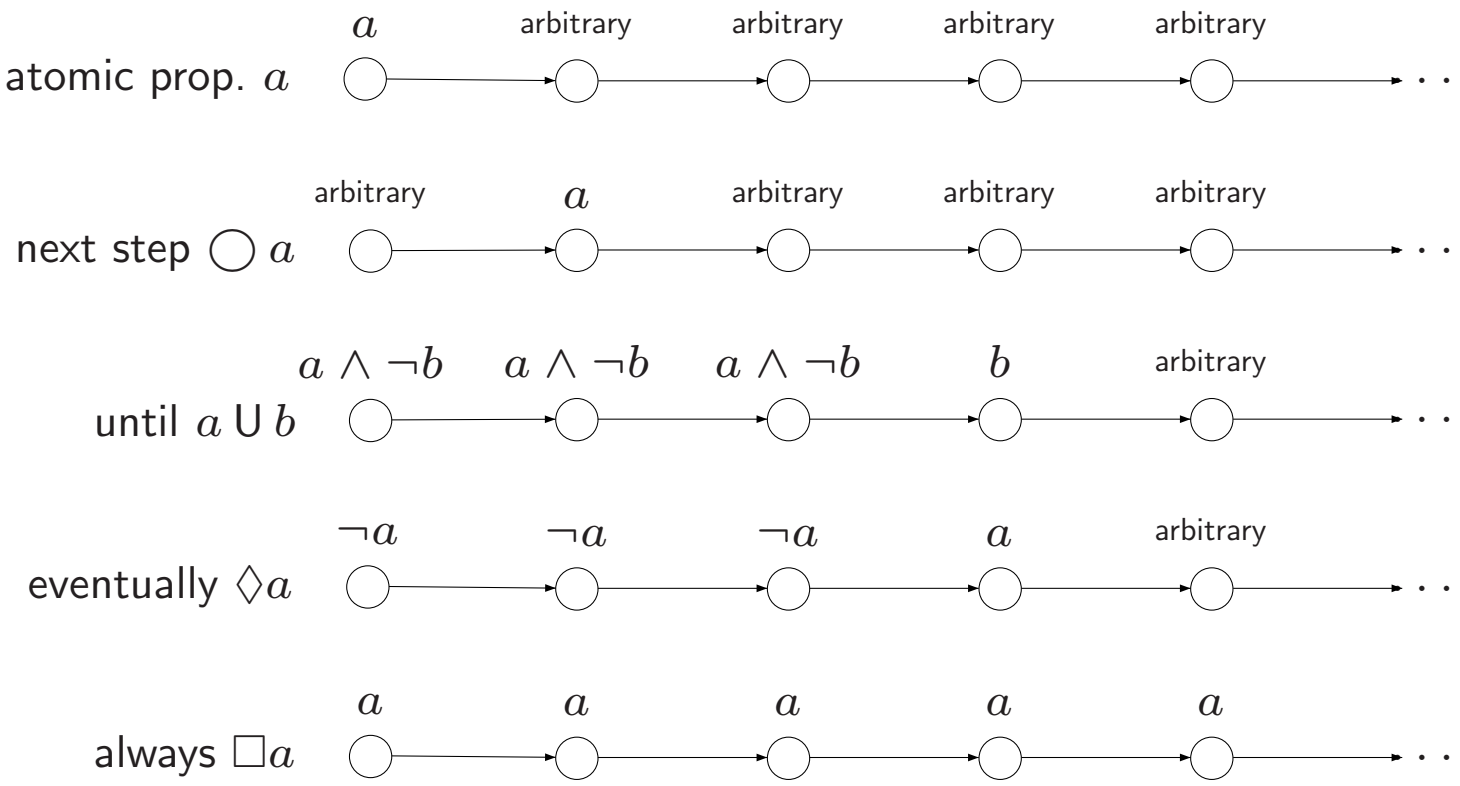
linear temporal logic is a logic for describing LT properties

# Derived operators

$$
\begin{aligned}
\phi \vee \psi &\equiv \neg(\neg\phi \wedge \neg\psi) \\
\phi \Rightarrow \psi &\equiv \neg\phi \vee \psi \\
\phi \Leftrightarrow \psi &\equiv (\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi) \\
\phi \oplus \psi &\equiv (\phi \wedge \neg\psi) \vee (\neg\phi \wedge \psi) \\
\mathsf{true} &\equiv \phi \vee \neg\phi \\
\mathsf{false} &\equiv \neg\,\mathsf{true} \\
\Diamond\,\phi &\equiv \mathsf{true}\,\mathsf{U}\,\phi \qquad \text{“sometimes in the future”} \\
\Box\,\phi &\equiv \neg\Diamond\neg\phi \qquad \text{“from now on for ever”}
\end{aligned}
$$

precedence order: the unary operators bind stronger than the binary ones.
$\neg$ and $\bigcirc$ bind equally strong. U takes precedence over $\wedge$, $\vee$, and $\rightarrow$

# Intuitive semantics



# LT properties

- Mutual exclusion is specified over $AP = \{ c_1, c_2 \}$ by

$$P_{mutex} \;=\; \text{set of infinite words } A_0\, A_1\, A_2 \ldots \text{ with } \{ c_1, c_2 \} \not\subseteq A_i \text{ for all } 0 \leqslant i$$

- In LTL: $\Box \neg (c_1 \wedge c_2)$

- Starvation freedom is specified over $AP = \{ c_1, w_1, c_2, w_2 \}$ by

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

$$\left(\overset{\infty}{\exists}\, j.\; w_1 \in A_j\right) \Rightarrow \left(\overset{\infty}{\exists}\, j.\; c_1 \in A_j\right) \wedge \left(\overset{\infty}{\exists}\, j.\; w_2 \in A_j\right) \Rightarrow \left(\overset{\infty}{\exists}\, j.\; c_2 \in A_j\right)$$

- In LTL: $(\Box \Diamond w_1 \;\Rightarrow\; \Box \Diamond c_1) \;\wedge\; (\Box \Diamond w_2 \;\Rightarrow\; \Box \Diamond c_2)$

# Semantics over words

The LT-property induced by LTL formula $\varphi$ over $AP$ is:

$$\mathit{Words}(\varphi) \;=\; \left\{\sigma \in \left(2^{AP}\right)^{\omega} \mid \sigma \models \varphi\right\}, \text{ where } \models \text{ is the smallest relation satisfying:}$$

$$\begin{array}{lcll}
\sigma & \models & \mathsf{true} & \\
\sigma & \models & a & \text{iff} \quad a \in A_0 \quad (\text{i.e., } A_0 \models a) \\
\sigma & \models & \varphi_1 \wedge \varphi_2 & \text{iff} \quad \sigma \models \varphi_1 \text{ and } \sigma \models \varphi_2 \\
\sigma & \models & \neg\, \varphi & \text{iff} \quad \sigma \not\models \varphi \\
\sigma & \models & \bigcirc \varphi & \text{iff} \quad \sigma[1..] = A_1 A_2 A_3 \ldots \models \varphi \\
\sigma & \models & \varphi_1 \,\mathsf{U}\, \varphi_2 & \text{iff} \quad \exists j \geqslant 0.\ \sigma[j..] \models \varphi_2 \ \text{ and } \ \sigma[i..] \models \varphi_1,\ 0 \leqslant i < j
\end{array}$$

for $\sigma = A_0 A_1 A_2 \ldots$ we have $\sigma[i..] = A_i A_{i+1} A_{i+2} \ldots$ is the suffix of $\sigma$ from index $i$ on

# Semantics of $\Box$, $\Diamond$, $\Box\Diamond$ and $\Diamond\Box$

$$\sigma \models \Diamond\varphi \quad \text{iff} \quad \exists j \geqslant 0.\ \sigma[j..] \models \varphi$$

$$\sigma \models \Box\varphi \quad \text{iff} \quad \forall j \geqslant 0.\ \sigma[j..] \models \varphi$$

$$\sigma \models \Box\Diamond\varphi \quad \text{iff} \quad \forall j \geqslant 0.\, \exists i \geqslant j.\ \sigma[i \ldots] \models \varphi$$

$$\sigma \models \Diamond\Box\varphi \quad \text{iff} \quad \exists j \geqslant 0.\, \forall i \geqslant j.\ \sigma[i \ldots] \models \varphi$$

# Semantics over paths and states

Let $TS = (S, Act, \rightarrow, I, AP, L)$ and $\varphi$ an LTL-formula over $AP$.

- For infinite path fragment $\pi$ of $TS$:

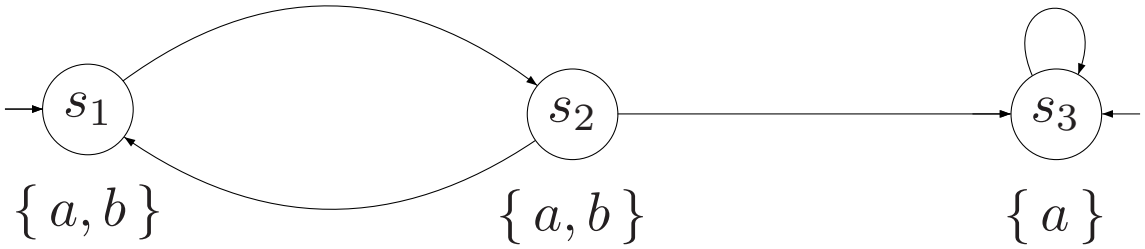
$$\pi \models \varphi \qquad \text{iff} \qquad trace(\pi) \models \varphi$$

- For state $s \in S$:

$$s \models \varphi \qquad \text{iff} \qquad \forall \pi \in Paths(s).\ \pi \models \varphi$$

- $TS$ satisfies $\varphi$, denoted $TS \models \varphi$, iff $Traces(TS) \subseteq Words(\varphi)$

# Example




$TS \models \Box a$

$TS \models \Box(\neg b \Rightarrow \Box(a \wedge \neg b))$

$TS \not\models \bigcirc(a \wedge b)$

$TS \not\models b \,\mathsf{U}\, (a \wedge \neg b)$

# Practical properties in LTL

- Reachability
  - simple reachability $\Diamond \psi$
  - conditional reachability $\phi \, \mathsf{U} \, \psi$
- Safety
  - invariant $\Box \phi$
- Liveness $\Box (\phi \Rightarrow \Diamond \psi)$ and others
- Fairness $\Box \Diamond \phi$ and others

# Semantics of negation

For paths, it holds $\pi \models \varphi$ if and only if $\pi \not\models \neg\varphi$ since:

$$\mathit{Words}(\neg\varphi) = \left(2^{AP}\right)^{\omega} \setminus \mathit{Words}(\varphi) \quad .$$

But: $\mathit{TS} \not\models \varphi$ and $\mathit{TS} \models \neg\varphi$ are *not* equivalent in general

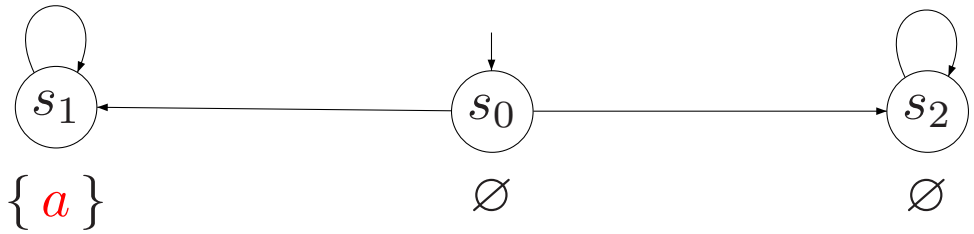
It holds: $\mathit{TS} \models \neg\varphi$ implies $\mathit{TS} \not\models \varphi$. Not always the reverse!

Note that:

$$\begin{aligned} \mathit{TS} \not\models \varphi \quad & \text{iff } \mathit{Traces}(\mathit{TS}) \not\subseteq \mathit{Words}(\varphi) \\ & \text{iff } \mathit{Traces}(\mathit{TS}) \setminus \mathit{Words}(\varphi) \neq \varnothing \\ & \text{iff } \mathit{Traces}(\mathit{TS}) \cap \mathit{Words}(\neg\varphi) \neq \varnothing \quad . \end{aligned}$$

$\mathit{TS}$ neither satisfies $\varphi$ nor $\neg\varphi$ if there are
paths $\pi_1$ and $\pi_2$ in $\mathit{TS}$ such that $\pi_1 \models \varphi$ and $\pi_2 \models \neg\varphi$

# Example




A transition system for which $TS \not\models \Diamond a$ and $TS \not\models \neg\Diamond a$

# Content of this lecture

- Linear temporal logic
  - syntax, semantics, specifying properties

$\Rightarrow$ Equivalences and fairness
  - weak until, fairness in LTL

- LTL model checking
  - GNBA, from LTL to GNBA, complexity

# Equivalence

LTL formulas $\phi, \psi$ are *equivalent*, denoted $\phi \equiv \psi$, if:

$$\mathit{Words}(\phi) = \mathit{Words}(\psi)$$

# Duality and idempotence laws

Duality:

$$\neg \Box \phi \equiv \Diamond \neg \phi$$
$$\neg \Diamond \phi \equiv \Box \neg \phi$$
$$\neg \bigcirc \phi \equiv \bigcirc \neg \phi$$

Idempotency:

$$\Box \Box \phi \equiv \Box \phi$$
$$\Diamond \Diamond \phi \equiv \Diamond \phi$$
$$\phi \,\mathsf{U}\, (\phi \,\mathsf{U}\, \psi) \equiv \phi \,\mathsf{U}\, \psi$$
$$(\phi \,\mathsf{U}\, \psi) \,\mathsf{U}\, \psi \equiv \phi \,\mathsf{U}\, \psi$$

# Expansion laws

Expansion:
$$\phi \,\mathsf{U}\, \psi \equiv \psi \vee (\phi \wedge \bigcirc(\phi \,\mathsf{U}\, \psi))$$
$$\Diamond \phi \equiv \phi \vee \bigcirc \Diamond \phi$$
$$\Box \phi \equiv \phi \wedge \bigcirc \Box \phi$$

proof on the black board

# Expansion for until

$P = \mathit{Words}(\varphi \,\mathsf{U}\, \psi)$ satisfies:

$$P \;=\; \mathit{Words}(\psi) \cup \big\{\, A_0A_1A_2\ldots \in \mathit{Words}(\varphi) \;\mid\; A_1A_2\ldots \in P \,\big\}$$

and is the *smallest* LT-property such that:

$$\mathit{Words}(\psi) \cup \big\{A_0A_1A_2\ldots \in \mathit{Words}(\varphi) \;\mid\; A_1A_2\ldots \in P\big\} \;\subseteq\; P \quad (*)$$

smallest LT-property satisfying condition (*) means that:
$P = \mathit{Words}(\varphi \,\mathsf{U}\, \psi)$ satisfies (*) and $\mathit{Words}(\varphi \,\mathsf{U}\, \psi) \subseteq P$ for each $P$ satisfying (*)

# LTL fairness constraints

Let $\Phi$ and $\Psi$ be propositional logic formulas over *AP*.

1. An *unconditional LTL fairness constraint* is of the form:

$$ufair \;=\; \Box\Diamond\Psi$$

2. A *strong LTL fairness condition* is of the form:

$$sfair \;=\; \Box\Diamond\Phi \;\longrightarrow\; \Box\Diamond\Psi$$

3. A *weak LTL fairness constraint* is of the form:

$$wfair \;=\; \Diamond\Box\Phi \;\longrightarrow\; \Box\Diamond\Psi$$

$\Phi$ stands for "something is enabled"; $\Psi$ for "something is taken"

# LTL fairness assumption

- *LTL fairness assumption* = conjunction of LTL fairness constraints
  - the fairness constraints are of any arbitrary type
- Strong fairness assumption: $sfair = \bigwedge_{0<i\leqslant k} \left(\Box\Diamond\Phi_i \longrightarrow \Box\Diamond\Psi_i\right)$
  - compare this to an action-based strong fairness constraint over $A$ with $|A| = k$
- General format: $fair = ufair \wedge sfair \wedge wfair$
- Rules of thumb:
  - strong (or unconditional) fairness assumptions are useful for solving contentions
  - weak fairness suffices for resolving nondeterminism resulting from interleaving

# Fair satisfaction

For state $s$ in transition system $TS$ (over $AP$) without terminal states, let

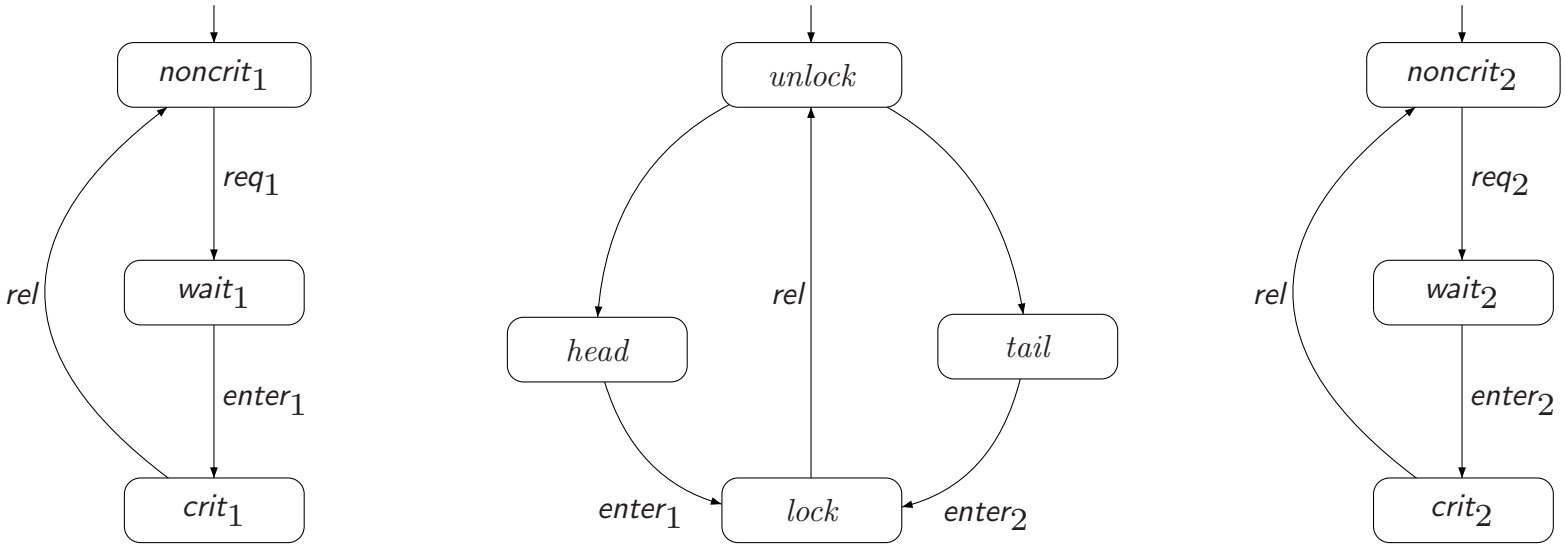
$$\begin{aligned} \mathit{FairPaths}_{fair}(s) &= \left\{ \pi \in \mathit{Paths}(s) \mid \pi \models fair \right\} \\ \mathit{FairTraces}_{fair}(s) &= \left\{ \mathit{trace}(\pi) \mid \pi \in \mathit{FairPaths}_{fair}(s) \right\} \end{aligned}$$

For LTL-formula $\varphi$, and LTL fairness assumption $fair$:

$$\begin{aligned} s \models_{fair} \varphi \quad &\text{if and only if} \quad \forall \pi \in \mathit{FairPaths}_{fair}(s).\, \pi \models \varphi \quad \text{and} \\ TS \models_{fair} \varphi \quad &\text{if and only if} \quad \forall s_0 \in I.\, s_0 \models_{fair} \varphi \end{aligned}$$

$\models_{fair}$ is the *fair satisfaction relation* for LTL; $\models$ the standard one for LTL

# Randomized arbiter




$$TS_1 \parallel Arbiter \parallel TS_2 \not\models \Box\Diamond\, crit_1$$

But: $TS_1 \parallel Arbiter \parallel TS_2 \models_{fair} \Box\Diamond crit_1 \wedge \Box\Diamond crit_2$ with $fair = \Box\Diamond head \wedge \Box\Diamond tail$

# Reducing $\models_{fair}$ to $\models$

For:

- transition system $TS$ without terminal states
- LTL formula $\varphi$, and
- LTL fairness assumption $fair$

it holds:

$$TS \models_{fair} \varphi \quad \text{if and only if} \quad TS \models (fair \rightarrow \varphi)$$

verifying an LTL-formula under a fairness assumption can be done using standard verification algorithms for LTL

# Content of this lecture

- Linear temporal logic
  - syntax, semantics, specifying properties
- Equivalences and fairness
  - weak until, fairness in LTL

$\Rightarrow$ LTL model checking

  - GNBA, from LTL to GNBA, complexity

# LTL model-checking problem

The following decision problem:

Given finite transition system $TS$ and LTL-formula $\varphi$:

yields "yes" if $TS \models \varphi$, and "no" (plus a counterexample) if $TS \not\models \varphi$

# NBA for LTL-formulae

# A first attempt

$$TS \models \varphi \quad \text{if and only if} \quad \mathit{Traces}(TS) \subseteq \underbrace{\mathit{Words}(\varphi)}_{\mathcal{L}_\omega(\mathcal{A}_\varphi)}$$

$$\text{if and only if} \quad \mathit{Traces}(TS) \cap \overline{\mathcal{L}_\omega(\mathcal{A}_\varphi)} = \varnothing$$

$$\text{if and only if} \quad \mathit{Traces}(TS) \cap \mathcal{L}_\omega(\overline{\mathcal{A}_\varphi}) = \varnothing$$

*but complementation of NBA is quadratically exponential*

*if $\mathcal{A}$ has $n$ states, $\overline{\mathcal{A}}$ has $c^{n^2}$ states in worst case*

# Observation

$TS \models \varphi$ if and only if $\mathit{Traces}(TS) \subseteq \mathit{Words}(\varphi)$

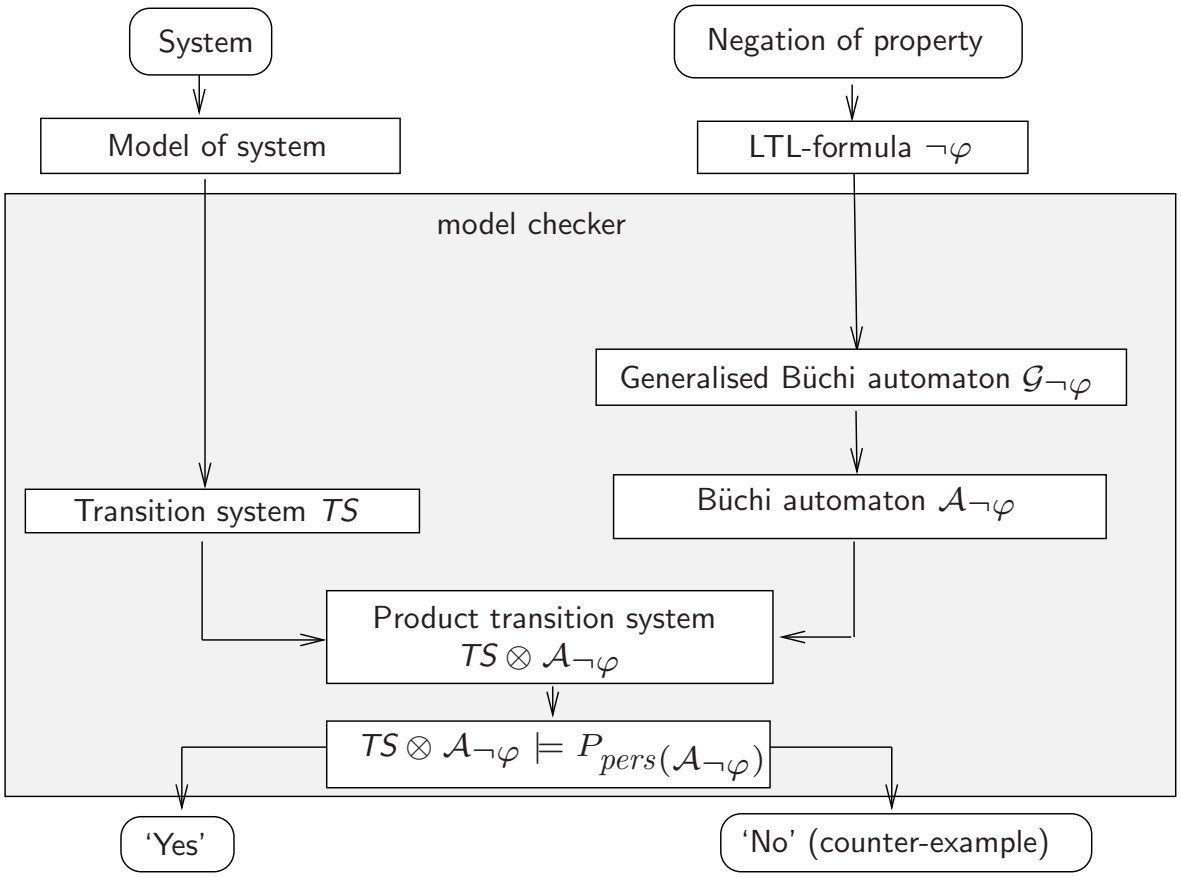
if and only if $\mathit{Traces}(TS) \cap \big((2^{AP})^\omega \setminus \mathit{Words}(\varphi)\big) = \varnothing$

if and only if $\mathit{Traces}(TS) \cap \underbrace{\mathit{Words}(\neg\varphi)}_{\mathcal{L}_\omega(\mathcal{A}_{\neg\varphi})} = \varnothing$

if and only if $TS \otimes \mathcal{A}_{\neg\varphi} \models \Diamond\Box\, \neg F$

*LTL model checking is thus reduced to persistence checking!*

# Overview of LTL model checking



# Generalized Büchi automata

- NBA are as expressive as $\omega$-regular languages
- Variants of NBA exist that are equally expressive
  - Muller, Rabin, Streett automata, and eneralized Büchi automata (GNBA)
- GNBA are like NBA, but have a distinct acceptance criterion
  - a GNBA requires to visit several sets $F_1, \ldots, F_k$ $(k \geqslant 0)$ infinitely often
  - for $k{=}0$, all runs are accepting; for $k{=}1$ it behaves like an NBA
- GNBA are useful to relate temporal logic and automata

# Generalized Büchi automata

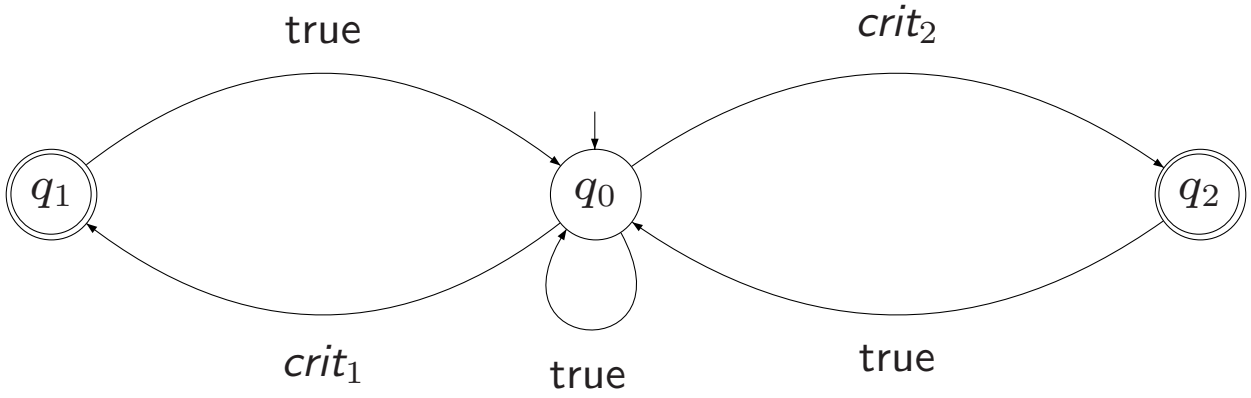
A generalized NBA (GNBA) $\mathcal{G}$ is a tuple $(Q, \Sigma, \delta, Q_0, \mathcal{F})$ where:

- $Q$, $\Sigma$, $\delta$ and $Q_0$ are as before, and
- $\mathcal{F} = \{ F_1, \ldots, F_k \}$ is a (possibly empty) subset of $2^Q$

# Language of a GNBA

- GNBA $\mathcal{G} = (Q, \Sigma, \delta, Q_0, \mathcal{F})$ and word $\sigma = A_0 A_1 A_2 \ldots \in \Sigma^\omega$
- A accepted run for $\sigma$ in $\mathcal{G}$ is an infinite sequence $q_0\, q_1\, q_2 \ldots$ such that:
  - $q_0 \in Q_0$ and $q_i \xrightarrow{A_i} q_{i+1}$ for all $0 \leqslant i$, and
  - for all $F \in \mathcal{F}$: $q_i \in F$ for infinitely many $i$
- $\mathcal{L}_\omega(\mathcal{G}) = \{\sigma \in \Sigma^\omega \mid$ there exists an accepting run for $\sigma$ in $\mathcal{G}\}$

# Example




A GNBA for the property "both processes are infinitely often in their critical section"

$$\mathcal{F} = \{ \{ q_1 \}, \{ q_2 \} \}$$

# From GNBA to NBA

For any GNBA $\mathcal{G}$ there exists an NBA $\mathcal{A}$ with:

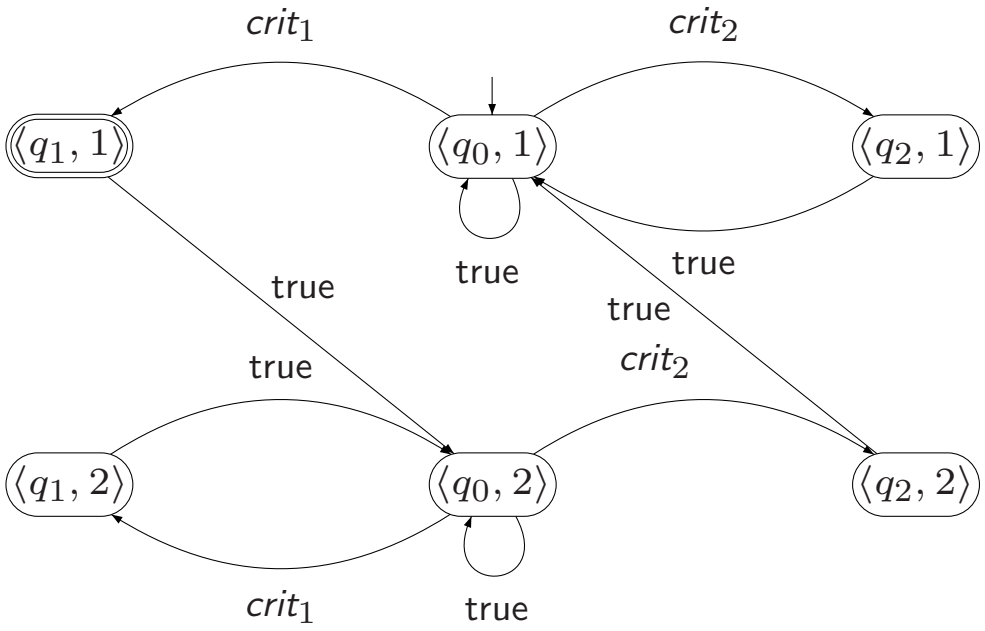
$\mathcal{L}_\omega(\mathcal{G}) = \mathcal{L}_\omega(\mathcal{A})$ and $|\mathcal{A}| = \mathcal{O}(|\mathcal{G}| \cdot |\mathcal{F}|)$

where $\mathcal{F}$ denotes the set of acceptance sets in $\mathcal{G}$

Sketch of transformation GNBA (with $|\mathcal{F}| = k$) into equivalent NBA:

- make $k$ copies of the GNBA
- initial states of NBA := the initial states in the first copy
- final states of NBA := accept set $F_1$ in the first copy
- on visiting in $i$-th copy a state in $F_i$, then move to the $(i{+}1)$-st copy

# Example


$crit_1$
$crit_2$
$\langle q_1, 1\rangle$
$\langle q_0, 1\rangle$
$\langle q_2, 1\rangle$
true
true
true
true
$crit_2$
true
$\langle q_1, 2\rangle$
$\langle q_0, 2\rangle$
$\langle q_2, 2\rangle$
$crit_1$
true

# From LTL to GNBA

GNBA $\mathcal{G}_\varphi$ over $2^{AP}$ for LTL-formula $\varphi$ with $\mathcal{L}_\omega(\mathcal{G}_\varphi) = \mathit{Words}(\varphi)$:

- Assume $\varphi$ only contains the operators $\wedge$, $\neg$, $\bigcirc$ and U
- States are elementary sets of sub-formulas in $\varphi$
  - for $\sigma = A_0 A_1 A_2 \ldots \in \mathit{Words}(\varphi)$, "expand" $A_i \subseteq AP$ with sub-formulas of $\varphi$
  - ... to obtain $\bar{\sigma} = B_0 B_1 B_2 \ldots$ such that

$$\psi \in B_i \qquad \text{if and only if} \qquad \sigma^i = A_i A_{i+1} A_{i+2} \ldots \models \psi$$

  - $\bar{\sigma}$ is intended to be a run in GNBA $\mathcal{G}_\varphi$ for $\sigma$
- Transitions are derived from semantics $\bigcirc$ and expansion law for U
- Accept sets guarantee that: $\bar{\sigma}$ is an accepting run for $\sigma$ iff $\sigma \models \varphi$

# From LTL to GNBA: the states (example)

- Let $\varphi = a \,\mathsf{U}\, (\neg a \wedge b)$ and $\sigma = \{a\}\{a,b\}\{b\}\ldots$
  - $B_i$ is a subset of $\{a, b, \neg a \wedge b, \varphi\} \cup \{\neg a, \neg b, \neg(\neg a \wedge b), \neg\varphi\}$
  - this set of formulas is also called the *closure* of $\varphi$

- Extend $A_0 = \{a\}$, $A_1 = \{a,b\}$, $A_2 = \{b\}$, ... as follows:
  - extend $A_0$ with $\neg b$, $\neg(\neg a \wedge b)$, and $\varphi$ as they hold in $\sigma^0 = \sigma$ (and no others)
  - extend $A_1$ with $\neg(\neg a \wedge b)$ and $\varphi$ as they hold in $\sigma^1$ (and no others)
  - extend $A_2$ with $\neg a$, $\neg a \wedge b$ and $\varphi$ as they hold in $\sigma^2$ (and no others)
  - ... and so forth
  - this is not effective and is performed on the automaton (not on words)

- Result:
  - $\bar{\sigma} = \underbrace{\{a, \neg b, \neg(\neg a \wedge b), \varphi\}}_{B_0} \underbrace{\{a, b, \neg(\neg a \wedge b), \varphi\}}_{B_1} \underbrace{\{\neg a, b, \neg a \wedge b, \varphi\}}_{B_2} \ldots$

# Closure

For LTL-formula $\varphi$, the set *closure*($\varphi$)

consists of all sub-formulas $\psi$ of $\varphi$ and their negation $\neg\psi$

(where $\psi$ and $\neg\neg\psi$ are identified)

for $\varphi = a \,\mathsf{U}\, (\neg a \wedge b)$, *closure*($\varphi$) $= \{\, a, b, \neg a, \neg b, \neg a \wedge b, \neg(\neg a \wedge b), \varphi, \neg\varphi \,\}$

can we take $B_i$ as any subset of *closure*($\varphi$)? no! they must be elementary

# Elementary sets of formulae

$B \subseteq \mathit{closure}(\varphi)$ is *elementary* if:

1. $B$ is *logically consistent* if for all $\varphi_1 \wedge \varphi_2, \psi \in \mathit{closure}(\varphi)$:
   - $\varphi_1 \wedge \varphi_2 \in B \;\Leftrightarrow\; \varphi_1 \in B$ and $\varphi_2 \in B$
   - $\psi \in B \;\Rightarrow\; \neg\psi \notin B$
   - true $\in \mathit{closure}(\varphi) \;\Rightarrow\;$ true $\in B$

2. $B$ is *locally consistent* if for all $\varphi_1 \,\mathsf{U}\, \varphi_2 \in \mathit{closure}(\varphi)$:
   - $\varphi_2 \in B \;\Rightarrow\; \varphi_1 \,\mathsf{U}\, \varphi_2 \in B$
   - $\varphi_1 \,\mathsf{U}\, \varphi_2 \in B$ and $\varphi_2 \notin B \;\Rightarrow\; \varphi_1 \in B$

3. $B$ is *maximal*, i.e., for all $\psi \in \mathit{closure}(\varphi)$:
   - $\psi \notin B \;\Rightarrow\; \neg\psi \in B$

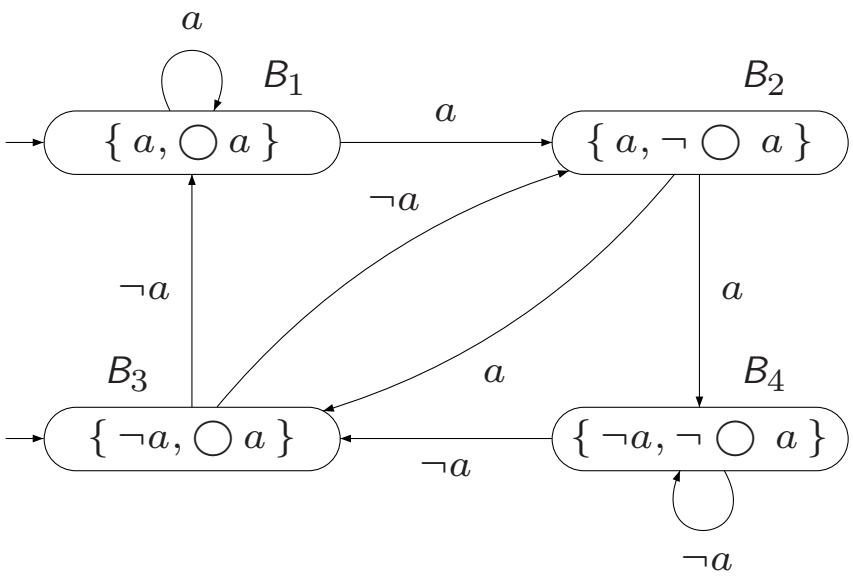
# The GNBA of LTL-formula $\varphi$

For LTL-formula $\varphi$, let $\mathcal{G}_\varphi = (Q, 2^{AP}, \delta, Q_0, \mathcal{F})$ where

- $Q =$ all elementary sets $B \subseteq \mathit{closure}(\varphi)$ , $Q_0 = \{ B \in Q \mid \varphi \in B \}$
- $\mathcal{F} = \Big\{ \{ B \in Q \mid \varphi_1 \,\mathsf{U}\, \varphi_2 \notin B \text{ or } \varphi_2 \in B \} \mid \varphi_1 \,\mathsf{U}\, \varphi_2 \in \mathit{closure}(\varphi) \Big\}$
- The transition relation $\delta : Q \times 2^{AP} \to 2^Q$ is given by:
  - If $A \neq B \cap AP$ then $\delta(B, A) = \varnothing$
  - $\delta(B, B \cap AP)$ is the set $B'$ of all elementary sets of formulas satisfying:
    (i) For every $\bigcirc \psi \in \mathit{closure}(\varphi)$: $\bigcirc \psi \in B \quad \Leftrightarrow \quad \psi \in B'$, and
    (ii) For every $\varphi_1 \,\mathsf{U}\, \varphi_2 \in \mathit{closure}(\varphi)$:

$$\varphi_1 \,\mathsf{U}\, \varphi_2 \in B \quad \Leftrightarrow \quad \Big( \varphi_2 \in B \;\vee\; (\varphi_1 \in B \;\wedge\; \varphi_1 \,\mathsf{U}\, \varphi_2 \in B') \Big)$$

# GNBA for LTL-formula $\bigcirc a$




$Q_0 = \{ B_1, B_3 \}$ since $\bigcirc a \in B_1$ and $\bigcirc a \in B_3$

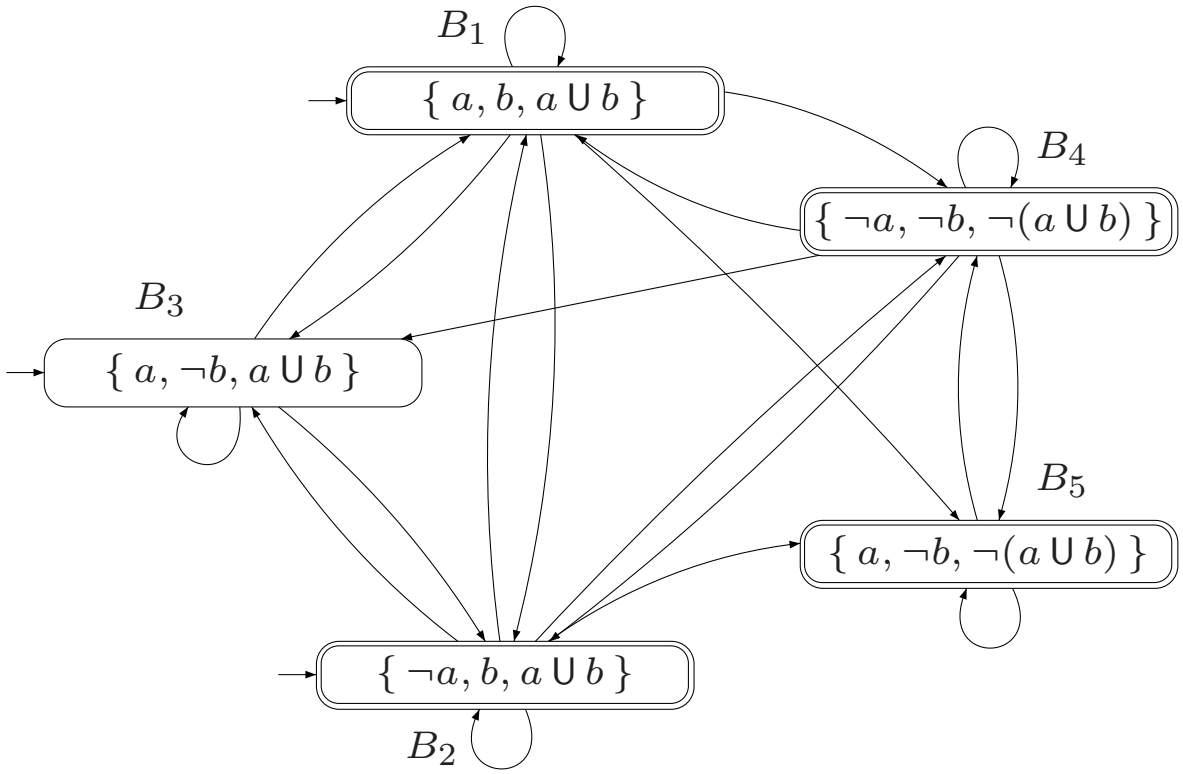
$\delta(B_2, \{ a \}) = \{ B_3, B_4 \}$ as $B_2 \cap \{ a \} = \{ a \}$, $\neg \bigcirc a = \bigcirc \neg a \in B_2$, and $\neg a \in B_3, B_4$

$\delta(B_1, \{ a \}) = \{ B_1, B_2 \}$ as $B_1 \cap \{ a \} = \{ a \}$, $\bigcirc a \in B_1$ and $a \in B_1, B_2$

$\delta(B_4, \{ a \}) = \varnothing$ since $B_4 \cap \{ a \} = \varnothing \neq \{ a \}$

The set $\mathcal{F}$ is empty, since $\varphi = \bigcirc a$ does not contain an until-operator

# GNBA for LTL-formula $a \mathsf{U} b$




justification: on the black board

# NBA are more expressive than LTL

*Corollary: every LTL-formula expresses an $\omega$-regular property*

*But: there exist $\omega$-regular properties that cannot be expressed in LTL*

Example: there is no LTL formula $\varphi$ with $\mathit{Words}(\varphi) = P$ for the LT-property:

$$P = \left\{ A_0 A_1 A_2 \ldots \in \left(2^{\{a\}}\right)^\omega \mid a \in A_{2i} \text{ for } i \geqslant 0 \right\}$$

But there exists an NBA $\mathcal{A}$ with $\mathcal{L}_\omega(\mathcal{A}) = P$

$\Rightarrow$ *there are $\omega$-regular properties that cannot be expressed in LTL!*

# Complexity for LTL to NBA

For any LTL-formula $\varphi$ (over $AP$) there exists an NBA $\mathcal{A}_\varphi$
with $\mathit{Words}(\varphi) = \mathcal{L}_\omega(\mathcal{A}_\varphi)$ and
which can be constructed in time and space in $2^{\mathcal{O}(|\varphi| \cdot \log |\varphi|)}$

*Justification complexity: next slide*

# Time and space complexity

- States GNBA $\mathcal{G}_\varphi$ are elementary sets of formulae in *closure*($\varphi$)
  - sets $B$ can be represented by bit vectors with single bit per subformula $\psi$ of $\varphi$
- The number of states in $\mathcal{G}_\varphi$ is bounded by $2^{|\mathsf{subf}(\varphi)|}$
  - where $\mathsf{subf}(\varphi)$ denotes the set of all subformulae of $\varphi$
  - $|\mathsf{subf}(\varphi)| \leqslant 2{\cdot}|\varphi|$; so, the number of states in $\mathcal{G}_\varphi$ is bounded by $2^{\mathcal{O}(|\varphi|)}$
- The number of accepting sets of $\mathcal{G}_\varphi$ is bounded above by $\mathcal{O}(|\varphi|)$
- The number of states in NBA $\mathcal{A}_\varphi$ is thus bounded by $2^{\mathcal{O}(|\varphi|)} \cdot \mathcal{O}(|\varphi|)$
- $2^{\mathcal{O}(|\varphi|)} \cdot \mathcal{O}(|\varphi|) \;=\; 2^{\mathcal{O}(|\varphi| \log |\varphi|)}$ qed

# Lower bound

There exists a family of LTL formulas $\varphi_n$ with $|\varphi_n| = \mathcal{O}(poly(n))$
such that every NBA $\mathcal{A}_{\varphi n}$ for $\varphi_n$ has at least $2^n$ states

# Proof (1)

Let $AP$ be non-empty, that is, $|2^{AP}| \geqslant 2$ and:

$$\mathcal{L}_n \;=\; \Big\{\, A_1 \ldots A_n\, A_1 \ldots A_n\, \sigma \mid A_i \subseteq AP \wedge \sigma \in \Big(2^{AP}\Big)^{\omega} \,\Big\}, \qquad \text{for } n \geqslant 0$$

It follows $\mathcal{L}_n \;=\; \mathit{Words}(\varphi_n)$ where $\varphi_n \;=\; \bigwedge_{a \in AP} \bigwedge_{0 \leqslant i < n} (\bigcirc^i a \longleftrightarrow \bigcirc^{n+i} a)$

$\varphi_n$ is an LTL formula of polynomial length: $|\varphi_n| \;\in\; \mathcal{O}\Big(|AP| \cdot n\Big)$

However, any NBA $\mathcal{A}$ with $\mathcal{L}_\omega(\mathcal{A}) = \mathcal{L}_n$ has at least $2^n$ states

# Proof (2)

Claim: any NBA $\mathcal{A}$ for $\bigwedge_{a \in AP} \bigwedge_{0 \leqslant i < n} (\bigcirc^i a \longleftrightarrow \bigcirc^{n+i} a)$ has at least $2^n$ states

Words of the form $A_1 \ldots A_n \, A_1 \ldots A_n \, \varnothing \, \varnothing \, \varnothing \ldots$ are accepted by $\mathcal{A}$

$\mathcal{A}$ thus has for every word $A_1 \ldots A_n$ of length $n$, a state $q(A_1 \ldots A_n)$, say, which can be reached from an initial state by consuming $A_1 \ldots A_n$

From $q(A_1 \ldots A_n)$, it is possible to visit an accept state infinitely often by accepting the suffix $A_1 \ldots A_n \, \varnothing \, \varnothing \, \varnothing \ldots$

If $A_1 \ldots A_n \neq A'_1 \ldots A'_n$ then

$$A_1 \ldots A_n \, A'_1 \ldots A'_n \, \varnothing \, \varnothing \, \varnothing \ldots \notin \mathcal{L}_n = \mathcal{L}_\omega(\mathcal{A})$$

Therefore, the states $q(A_1 \ldots A_n)$ are all pairwise different

Given $|2^{AP}|$ possible sequences $A_1 \ldots A_n$, NBA $\mathcal{A}$ has $\geqslant \left(\left|2^{AP}\right|\right)^n \geqslant 2^n$ states

# Complexity for LTL model checking

The time and space complexity of LTL model checking is in $\mathcal{O}\left(|TS| \cdot 2^{|\varphi|}\right)$

# Theoretical complexity

The LTL model-checking problem is PSPACE-complete

# CTL Model Checking

## Part #4 of Logic and Verification

*Joost-Pieter Katoen*

Software Modeling and Verification Group

RWTH Aachen University

MOVEP 2014, University of Nantes, July 7, 2014

# Content of this lecture

- Computation tree logic
  - syntax, semantics, equational laws
- CTL model checking
  - recursive descent, backward reachability, complexity
- Comparing LTL and CTL
  - what can be expressed in CTL? what in LTL?, efficiency
- Fairness
  - fair CTL semantics, model checking

# Content of this lecture

⇒ Computation tree logic

- syntax, semantics, equational laws

- CTL model checking
  - recursive descent, backward reachability, complexity

- Comparing LTL and CTL
  - what can be expressed in CTL? what in LTL?, efficiency

- Fairness
  - fair CTL semantics, model checking

# Linear and branching temporal logic

- *Linear* temporal logic:

  "statements about (all) paths starting in a state"

  - $s \models \Box(x \leqslant 20)$ iff for all possible paths starting in $s$ always $x \leqslant 20$

- *Branching* temporal logic:

  "statements about all or some paths starting in a state"

  - $s \models \forall\Box(x \leqslant 20)$ iff for **all** paths starting in $s$ always $x \leqslant 20$
  - $s \models \exists\Box(x \leqslant 20)$ iff for **some** path starting in $s$ always $x \leqslant 20$
  - nesting of path quantifiers is allowed

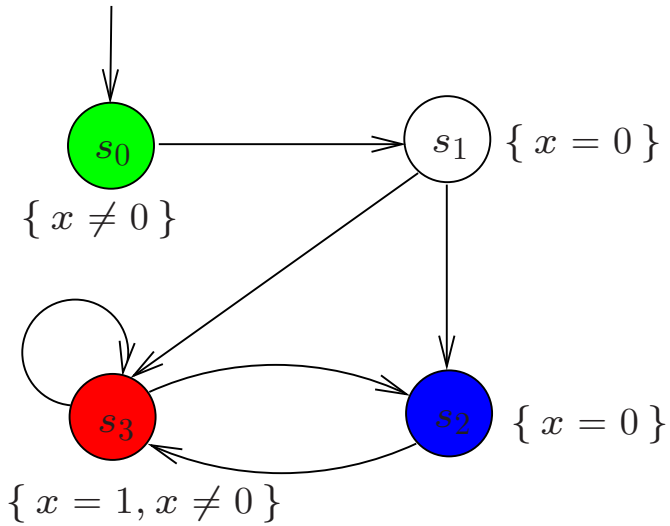
- Checking $\exists\varphi$ in LTL can be done using $\forall\neg\varphi$

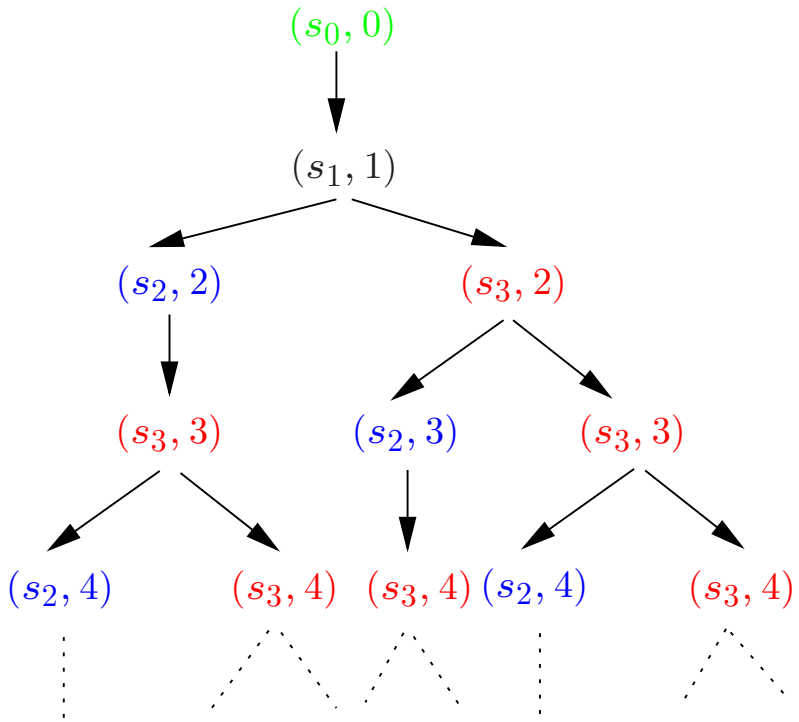
  - ... but this does not work for nested formulas such as $\forall\Box\exists\Diamond a$

# Linear versus branching temporal logic

- Semantics is based on a branching notion of time
  - an infinite tree of states obtained by unfolding transition system
  - one "time instant" may have several possible successor "time instants"
- Incomparable expressiveness
  - there are properties that can be expressed in LTL, but not in CTL
  - there are properties that can be expressed in most branching, but not in LTL
- Distinct model-checking algorithms, and their time complexities
- Distinct equivalences (pre-orders) on transition systems
  - that correspond to logical equivalence in LTL and branching temporal logics

# Transition systems and trees



| "behavior" in a state $s$ | path-based: $\mathit{trace}(s)$ | state-based: computation tree of $s$ |
|---|---|---|
| temporal logic | LTL: path formulas $\varphi$ $s \models \varphi$ iff $\forall \pi \in \mathit{Paths}(s).\, \pi \models \varphi$ | CTL: state formulas existential path quantification $\exists\varphi$ universal path quantification: $\forall\varphi$ |
| complexity of the model checking problems | PSPACE–complete $\mathcal{O}\left(\lvert \mathit{TS}\rvert \cdot 2^{\lvert\varphi\rvert}\right)$ | PTIME $\mathcal{O}\left(\lvert \mathit{TS}\rvert \cdot \lvert\Phi\rvert\right)$ |
| implementation-relation | trace inclusion and the like (proof is PSPACE-complete) | simulation and bisimulation (proof in polynomial time) |

# Computation tree logic

modal logic over infinite trees [Clarke & Emerson 1981]

- Statements over states
  - $a \in AP$ — atomic proposition
  - $\neg\, \Phi$ and $\Phi \wedge \Psi$ — negation and conjunction
  - $\exists\varphi$ — there *exists* a path fulfilling $\varphi$
  - $\forall\varphi$ — *all* paths fulfill $\varphi$
- Statements over paths
  - $\bigcirc\Phi$ — the next state fulfills $\Phi$
  - $\Phi$ U $\Psi$ — $\Phi$ holds until a $\Psi$-state is reached

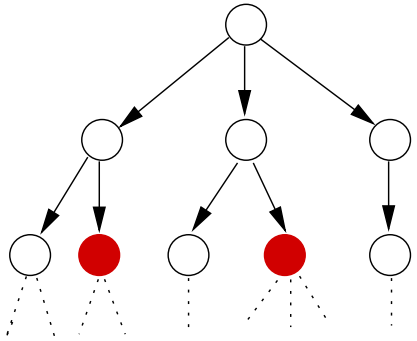
$\Rightarrow$ note that $\bigcirc$ and U *alternate* with $\forall$ and $\exists$

# Derived operators

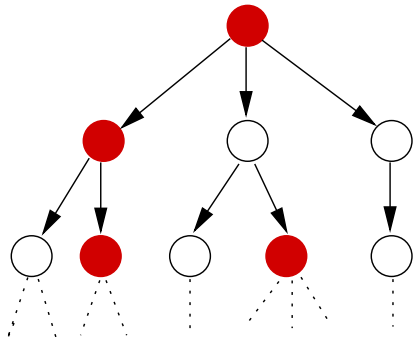
| | | | |
|---|---|---|---|
| potentially $\Phi$: | $\exists\Diamond\Phi$ | $=$ | $\exists(\mathsf{true}\,\mathsf{U}\,\Phi)$ |
| inevitably $\Phi$: | $\forall\Diamond\Phi$ | $=$ | $\forall(\mathsf{true}\,\mathsf{U}\,\Phi)$ |
| potentially always $\Phi$: | $\exists\Box\Phi$ | $:=$ | $\neg\forall\Diamond\neg\Phi$ |
| invariantly $\Phi$: | $\forall\Box\Phi$ | $=$ | $\neg\exists\Diamond\neg\Phi$ |
| weak until: | $\exists(\Phi\,\mathsf{W}\,\Psi)$ | $=$ | $\neg\forall\big((\Phi\wedge\neg\Psi)\,\mathsf{U}\,(\neg\Phi\wedge\neg\Psi)\big)$ |
| | $\forall(\Phi\,\mathsf{W}\,\Psi)$ | $=$ | $\neg\exists\big((\Phi\wedge\neg\Psi)\,\mathsf{U}\,(\neg\Phi\wedge\neg\Psi)\big)$ |

the boolean connectives are derived as usual

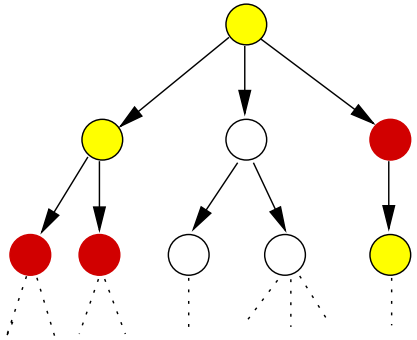
# Visualization of semantics



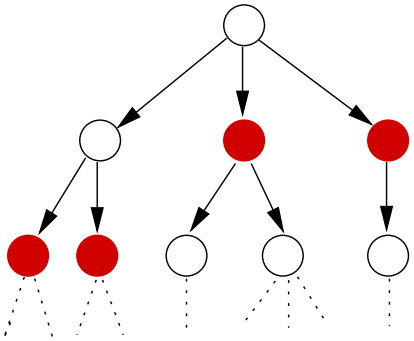
$\exists \Diamond red$



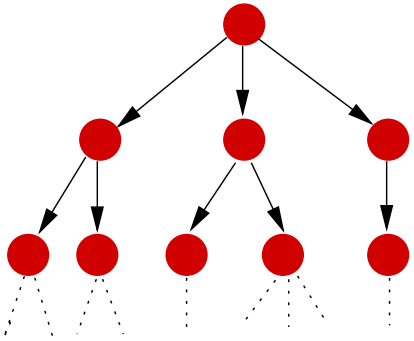
$\exists \Box red$



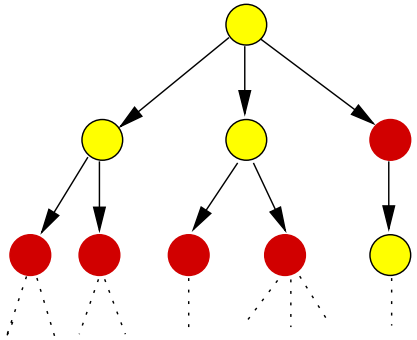
$\exists (yellow \,\mathsf{U}\, red)$



$\forall \Diamond red$



$\forall \Box red$



$\forall (yellow \,\mathsf{U}\, red)$

# Semantics of CTL state-formulas

Defined by a relation $\models$ such that

$s \models \Phi$ if and only if formula $\Phi$ holds in state $s$

$$
\begin{array}{lll}
s \models a & \text{iff} & a \in L(s) \\
s \models \neg\Phi & \text{iff} & \neg\,(s \models \Phi) \\
s \models \Phi \wedge \Psi & \text{iff} & (s \models \Phi) \wedge (s \models \Psi) \\
s \models \exists\varphi & \text{iff} & \pi \models \varphi \text{ for } \textit{some} \text{ path } \pi \text{ that starts in } s \\
s \models \forall\varphi & \text{iff} & \pi \models \varphi \text{ for } \textit{all} \text{ paths } \pi \text{ that start in } s
\end{array}
$$

# Semantics of CTL path-formulas

Define a relation $\models$ such that

$\pi \models \varphi$ if and only if path $\pi$ satisfies $\varphi$

$$\pi \models \bigcirc \Phi \qquad \text{iff } \pi[1] \models \Phi$$

$$\pi \models \Phi \,\mathsf{U}\, \Psi \qquad \text{iff } (\exists\, j \geqslant 0.\, \pi[j] \models \Psi \ \wedge\ (\forall\, 0 \leqslant k < j.\, \pi[k] \models \Phi))$$

where $\pi[i]$ denotes the state $s_i$ in the path $\pi$

# Transition system semantics

- For CTL-state-formula $\Phi$, the *satisfaction set* $Sat(\Phi)$ is defined by:

$$Sat(\Phi) \;=\; \{\, s \in S \mid s \models \Phi \,\}$$

- *TS* satisfies CTL-formula $\Phi$ iff $\Phi$ holds in all its initial states:

$$TS \models \Phi \quad \text{if and only if} \quad \forall s_0 \in I.\, s_0 \models \Phi$$

- Point of attention: $TS \not\models \Phi$ and $TS \not\models \neg\Phi$ is possible!
  - because of several initial states, e.g. $s_0 \models \exists\Box\Phi$ and $s_0' \not\models \exists\Box\Phi$

# CTL equivalence

CTL-formulas $\Phi$ and $\Psi$ (over $AP$) are *equivalent*, denoted $\Phi \equiv \Psi$
if and only if $Sat(\Phi) = Sat(\Psi)$ for all transition systems $TS$ over $AP$

$$\Phi \equiv \Psi \quad \text{iff} \quad (TS \models \Phi \quad \text{if and only if} \quad TS \models \Psi)$$

# Expansion laws

Recall in LTL: $\varphi \,\mathsf{U}\, \psi \;\equiv\; \psi \;\vee\; (\varphi \wedge \bigcirc\, (\varphi \,\mathsf{U}\, \psi))$

In CTL:

$$\forall(\Phi \,\mathsf{U}\, \Psi) \;\equiv\; \Psi \;\vee\; (\Phi \;\wedge\; \forall \bigcirc \forall(\Phi \,\mathsf{U}\, \Psi))$$

$$\forall\Diamond\Phi \;\equiv\; \Phi \;\vee\; \forall \bigcirc \forall\Diamond\Phi$$

$$\forall\Box\Phi \;\equiv\; \Phi \;\wedge\; \forall \bigcirc \forall\Box\Phi$$

$$\exists(\Phi \,\mathsf{U}\, \Psi) \;\equiv\; \Psi \;\vee\; (\Phi \;\wedge\; \exists \bigcirc \exists(\Phi \,\mathsf{U}\, \Psi))$$

$$\exists\Diamond\Phi \;\equiv\; \Phi \;\vee\; \exists \bigcirc \exists\Diamond\Phi$$

$$\exists\Box\Phi \;\equiv\; \Phi \;\wedge\; \exists \bigcirc \exists\Box\Phi$$

# Content of this lecture

- Computation tree logic
  - syntax, semantics, equational laws

$\Rightarrow$ CTL model checking
  - recursive descent, backward reachability, complexity

- Comparing LTL and CTL
  - what can be expressed in CTL? what in LTL?, efficiency

- Fairness
  - fair CTL semantics, model checking

# Existential normal form (ENF)

The set of CTL formulas in *existential normal form* (ENF) is given by:

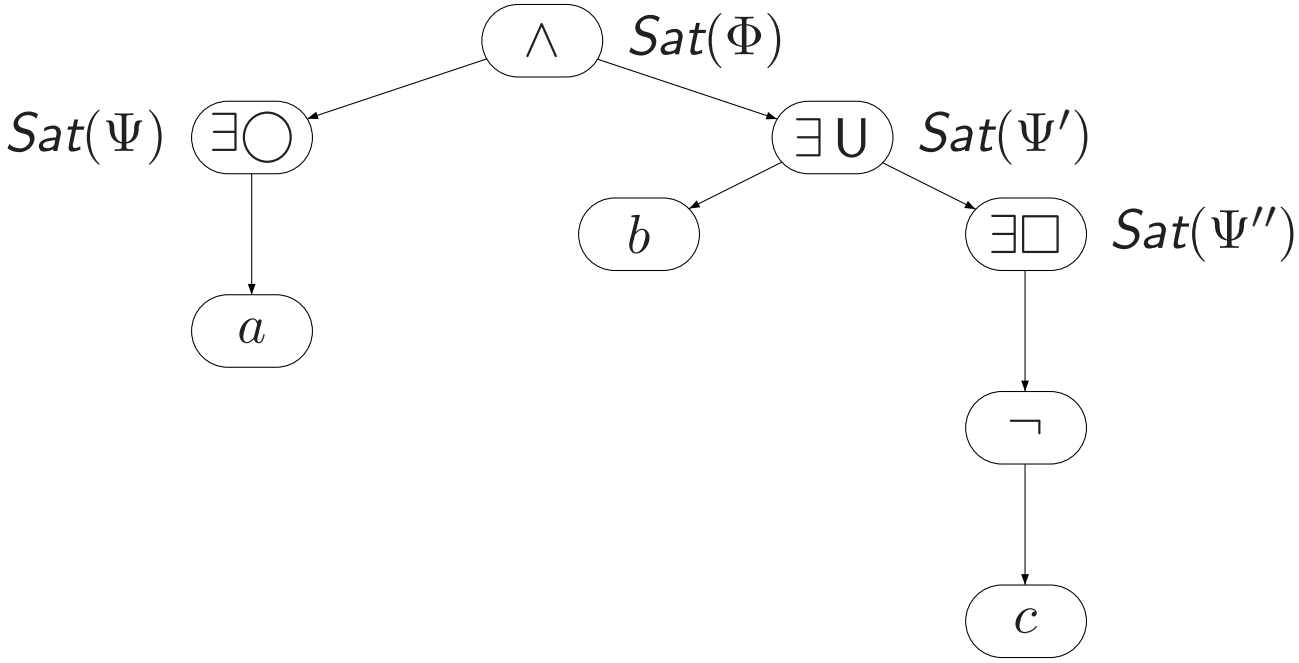
$$\Phi \quad ::= \quad \text{true} \;\Big|\; a \;\Big|\; \Phi_1 \wedge \Phi_2 \;\Big|\; \neg\Phi \;\Big|\; \exists\bigcirc\Phi \;\Big|\; \exists(\Phi_1 \,\mathsf{U}\, \Phi_2) \;\Big|\; \exists\Box\Phi$$

For each CTL formula, there exists an equivalent CTL formula in ENF

# Model checking CTL

- Convert the formula $\Phi'$ into an equivalent $\Phi$ in ENF
- How to check whether *TS* satisfies $\Phi$?
  - compute *recursively* the set $Sat(\Phi)$ of states that satisfy $\Phi$
  - check whether all initial states belong to $Sat(\Phi)$
- Recursive bottom-up computation:
  - consider the parse-tree of $\Phi$
  - start to compute $Sat(a)$, for all leafs in the tree
  - then go one level up in the tree and check the formula of these nodes
  - then go one level up and check the formula of these nodes
  - and so on..... until the root of the tree (i.e., $\Phi$) is checked

# Example




$$\Phi \;=\; \underbrace{\exists\bigcirc a}_{\Psi} \;\wedge\; \underbrace{\exists(b\,\mathsf{U}\,\underbrace{\exists\Box\neg c}_{\Psi''})}_{\Psi'} \quad .$$

# Characterization of *Sat* (1)

For all $CTL$ formulas $\Phi, \Psi$ over $AP$ it holds:

$$
\begin{aligned}
Sat(\text{true}) &= S \\
Sat(a) &= \{\, s \in S \mid a \in L(s) \,\}, \text{ for any } a \in AP \\
Sat(\Phi \wedge \Psi) &= Sat(\Phi) \cap Sat(\Psi) \\
Sat(\neg \Phi) &= S \setminus Sat(\Phi) \\
Sat(\exists \bigcirc \Phi) &= \{\, s \in S \mid Post(s) \cap Sat(\Phi) \neq \varnothing \,\}
\end{aligned}
$$

where $TS = (S, Act, \rightarrow, I, AP, L)$ is a transition system without terminal states

# Characterization of *Sat* (2)

For all $CTL$ formulas $\Phi, \Psi$ over $AP$ it holds:

- $Sat(\exists(\Phi \,\mathsf{U}\, \Psi))$ is the smallest subset $T$ of $S$, such that:

  (1) $Sat(\Psi) \subseteq T$ and

  (2) $s \in Sat(\Phi)$ and $Post(s) \cap T \neq \varnothing$ implies $s \in T$

- $Sat(\exists\Box\Phi)$ is the largest subset $T$ of $S$, such that:

  (3) $T \subseteq Sat(\Phi)$ and

  (4) $s \in T$ implies $Post(s) \cap T \neq \varnothing$

where $TS = (S, Act, \rightarrow, I, AP, L)$ is a transition system without terminal states

# Computation of *Sat*

**switch**$(\Phi)$:

| | | |
|---|---|---|
| $a$ | : | **return** $\{ s \in S \mid a \in L(s) \}$; |
| $\ldots$ | : | $\ldots\ldots$ |
| $\exists\bigcirc\Psi$ | : | **return** $\{ s \in S \mid \mathit{Post}(s) \cap \mathit{Sat}(\Psi) \neq \varnothing \}$; |
| $\exists(\Phi_1 \,\mathsf{U}\, \Phi_2)$ | : | $T := \mathit{Sat}(\Phi_2)$; (* compute the smallest fixed point *)<br>**while** $\mathit{Sat}(\Phi_1) \setminus T \cap \mathit{Pre}(T) \neq \varnothing$ **do**<br>**let** $s \in \mathit{Sat}(\Phi_1) \setminus T \cap \mathit{Pre}(T)$;<br>$T := T \cup \{ s \}$;<br>**od**;<br>**return** $T$; |
| $\exists\Box\Psi$ | : | $T := \mathit{Sat}(\Psi)$; (* compute the greatest fixed point *)<br>**while** $\exists s \in T.\ \mathit{Post}(s) \cap T = \varnothing$ **do**<br>**let** $s \in \{ s \in T \mid \mathit{Post}(s) \cap T = \varnothing \}$;<br>$T := T \setminus \{ s \}$;<br>**od**;<br>**return** $T$; |

**end switch**

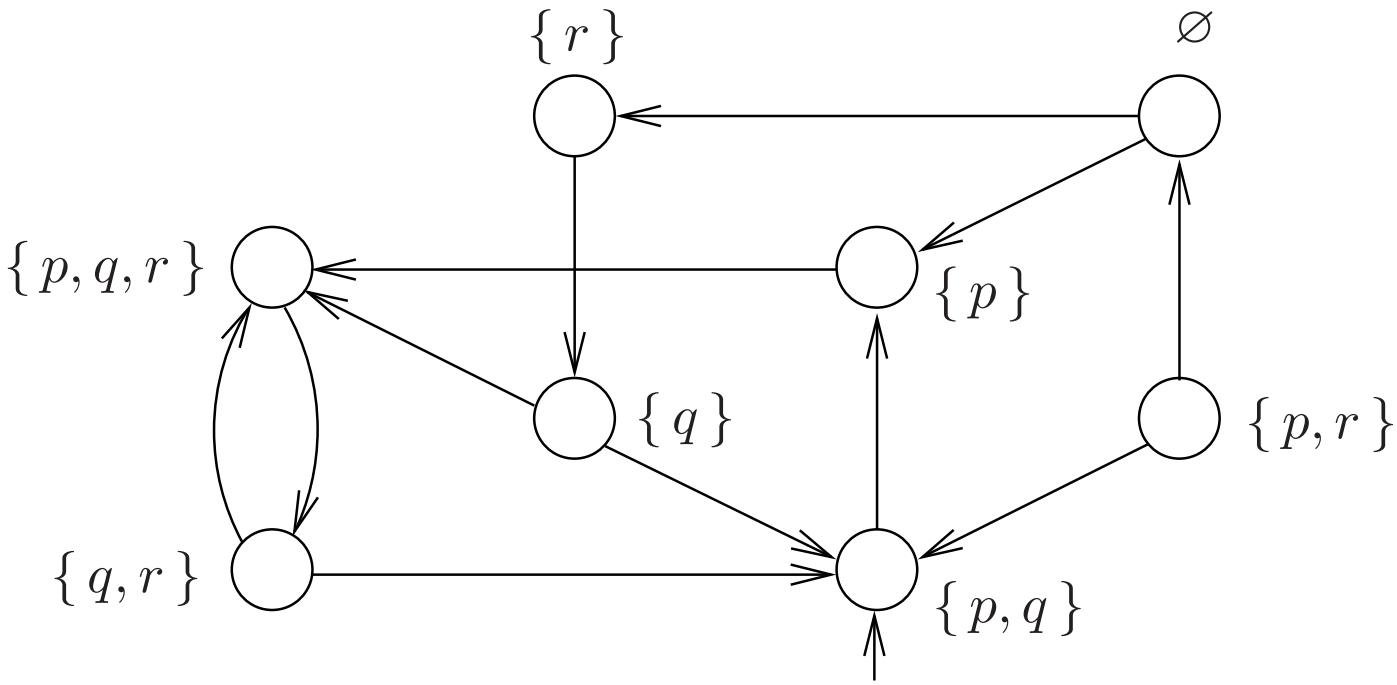
# Computing $Sat(\exists(\Phi \,\mathsf{U}\, \Psi))$

*Input:* finite transition system $TS$ with state-set $S$ and CTL-formula $\exists(\Phi \,\mathsf{U}\, \Psi)$
*Output:* $Sat(\exists(\Phi \,\mathsf{U}\, \Psi))$

---

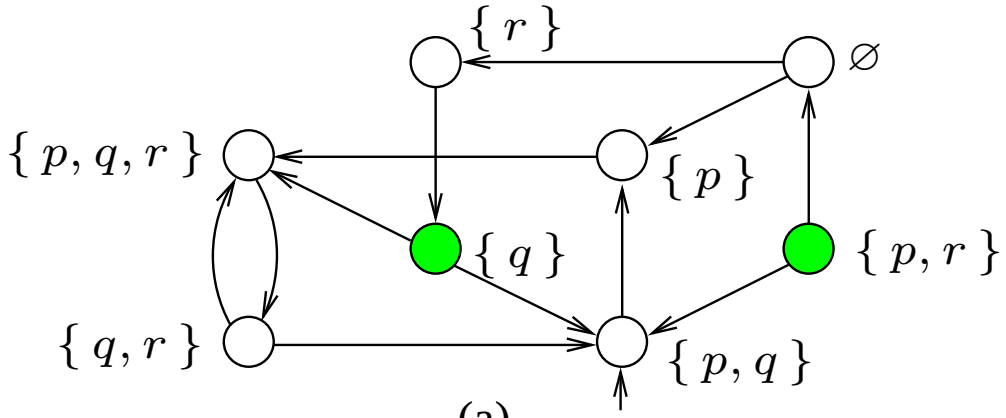
$E := Sat(\Psi)$; (* $E$ administers the states $s$ with $s \models \exists(\Phi \,\mathsf{U}\, \Psi)$ *)
$T := E$; (* $T$ contains the already visited states $s$ with $s \models \exists(\Phi \,\mathsf{U}\, \Psi)$ *)
**while** $E \neq \varnothing$ **do**
  **let** $s' \in E$;
  $E := E \setminus \{ s' \}$;
  **for all** $s \in Pre(s')$ **do**
    **if** $s \in Sat(\Phi) \setminus T$ **then** $E := E \cup \{ s \}; T := T \cup \{ s \};$ **fi**
  **od**
**od**
**return** $T$

# Example

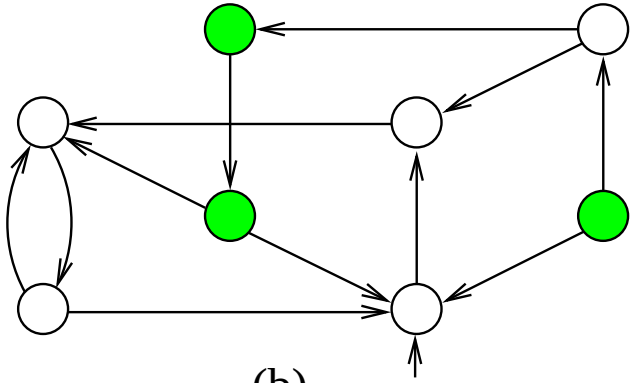


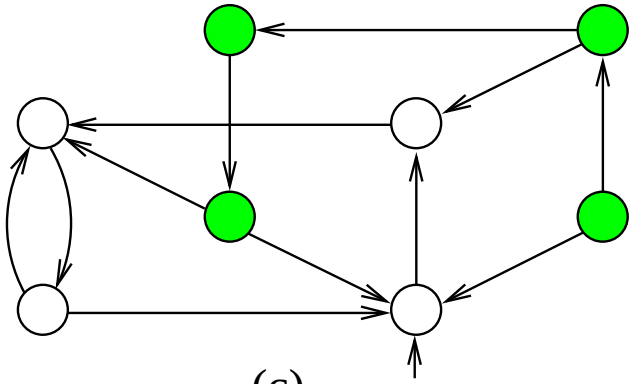

let's check the CTL-formula $\exists\Diamond((p = r) \wedge (p \neq q))$

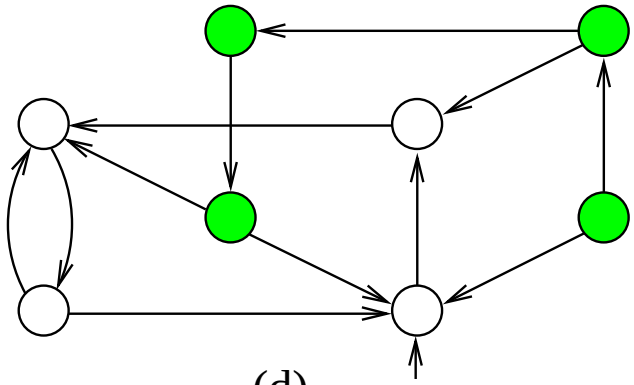
# The computation in snapshots



(a)


(b)


(c)


(d)

# Computing $\mathit{Sat}(\exists\Box\Phi)$

$E := S \setminus \mathit{Sat}(\Phi)$; (* $E$ contains any not visited $s'$ with $s' \not\models \exists\Box\Phi$ *)

$T := \mathit{Sat}(\Phi)$; (* $T$ contains any $s$ for which $s \models \exists\Box\Phi$ has not yet been disproven *)

**for all** $s \in \mathit{Sat}(\Phi)$ **do** $c[s] := \mid \mathit{Post}(s) \mid$; **od** (* initialize array $c$ *)

**while** $E \neq \varnothing$ **do**

(* loop invariant: $c[s] = \mid \mathit{Post}(s) \cap (T \cup E) \mid$ *)

  **let** $s' \in E$; (* $s' \not\models \Phi$ *)

  $E := E \setminus \{ s' \}$; (* $s'$ has been considered *)

  **for all** $s \in \mathit{Pre}(s')$ **do**

    **if** $s \in T$ **then**

      $c[s] := c[s] - 1$; (* update counter $c[s]$ for predecessor $s$ of $s'$ *)

      **if** $c[s] = 0$ **then**

        $T := T \setminus \{ s \}$; $E := E \cup \{ s \}$; (* $s$ does not have any successor in $T$ *)

      **fi**

    **fi**

  **od**

**od**

**return** $T$

# Alternative algorithm

1. Consider only state $s$ if $s \models \Phi$, otherwise *eliminate* $s$
   - change $TS$ into $TS[\Phi] = (S', Act, \rightarrow', I', AP, L')$ with $S' = Sat(\Phi)$,
   - $\rightarrow' = \rightarrow \cap (S' \times Act \times S')$, $I' = I \cap S'$, and $L'(s) = L(s)$ for $s \in S'$

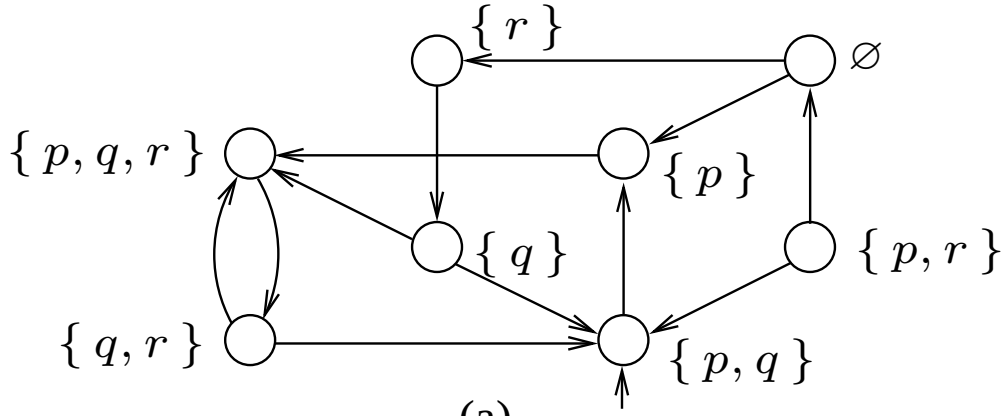
   $\Rightarrow$ all removed states will not satisfy $\exists\Box\Phi$, and thus can be safely removed

2. Determine all *non-trivial strongly connected components* in $TS[\Phi]$
   - non-trivial SCC = maximal, connected subgraph with at least one transition

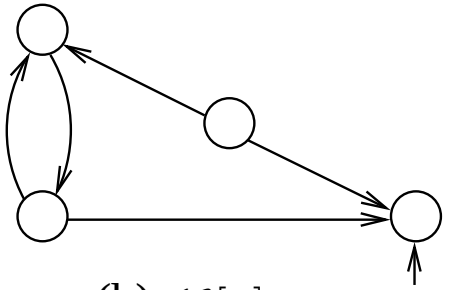
   $\Rightarrow$ any state in such SCC satisfies $\exists\Box\Phi$

3. $s \models \exists\Box\Phi$ is equivalent to "some *SCC is reachable* from $s$"
   - this search can be done in a backward manner

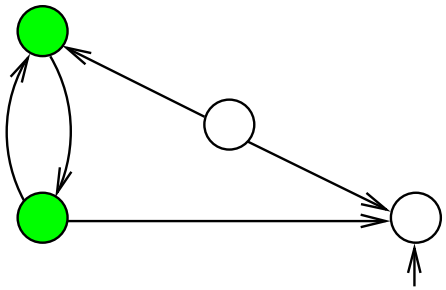
# Example



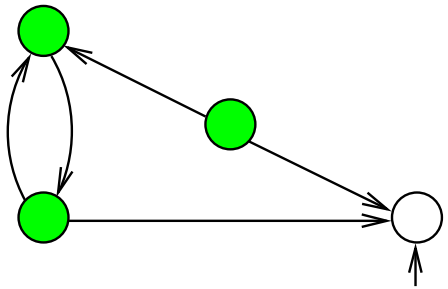

(a)



(b) $\mathcal{K}[q]$



(c) SCC



(d)

# Time complexity

For transition system $TS$ with $N$ states and $K$ transitions,
and CTL formula $\Phi$, the CTL model-checking problem $TS \models \Phi$
can be determined in time $\mathcal{O}(\mid \Phi \mid \cdot (N + M))$

this applies to both algorithm for existential until-formulas

# Content of this lecture

- Computation tree logic
  - syntax, semantics, equational laws
- CTL model checking
  - recursive descent, backward reachability, complexity

$\Rightarrow$ Comparing LTL and CTL

  - what can be expressed in CTL?, what in LTL?, efficiency

- Fairness
  - fair CTL semantics, model checking

# Equivalence of LTL and CTL formulas

- CTL-formula $\Phi$ and LTL-formula $\varphi$ (both over $AP$) are *equivalent*, denoted $\Phi \equiv \varphi$, if for any transition system $TS$ over $AP$:

$$TS \models \Phi \quad \text{if and only if} \quad TS \models \varphi$$

- Let $\Phi$ be a CTL-formula, and $\varphi$ the LTL-formula that is obtained by eliminating all path quantifiers in $\Phi$. Then:

$\Phi \equiv \varphi$ or there does not exist any LTL-formula that is equivalent to $\Phi$

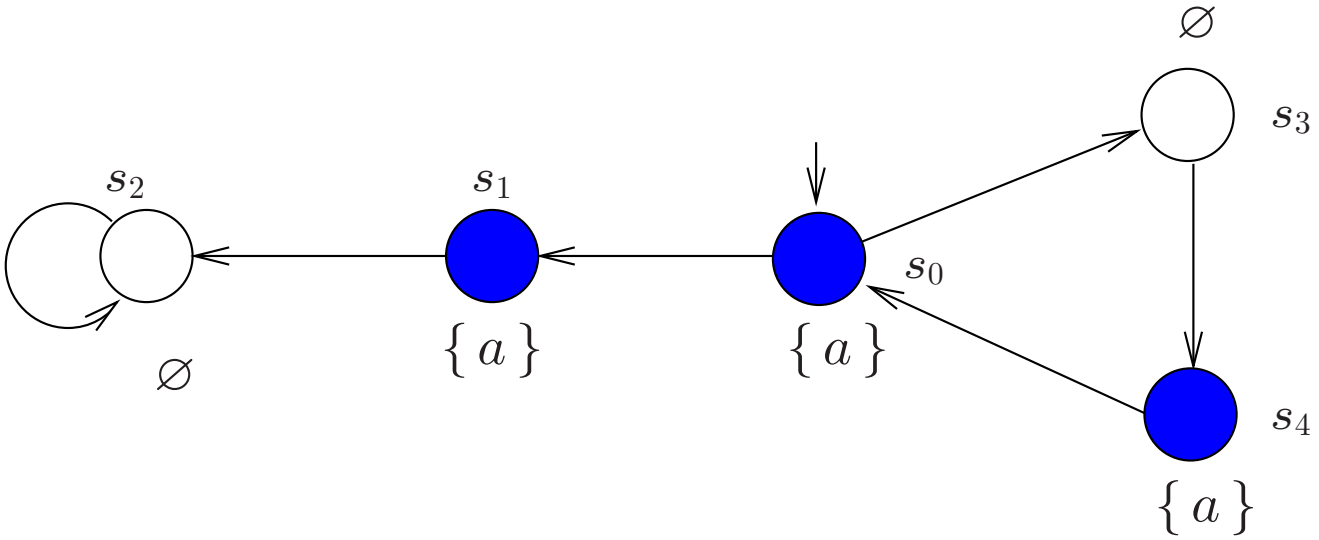
# LTL and CTL are incomparable

- Some LTL-formulas cannot be expressed in CTL, e.g.,
  - $\Diamond\Box a$
  - $\Diamond(a \ \wedge \ \bigcirc a)$
- Some CTL-formulas cannot be expressed in LTL, e.g.,
  - $\forall\Diamond\forall\Box a$
  - $\forall\Diamond(a \wedge \forall\bigcirc a)$
  - $\forall\Box\exists\Diamond a$

$\Rightarrow$ Cannot be expressed $=$ there does not exist an equivalent formula

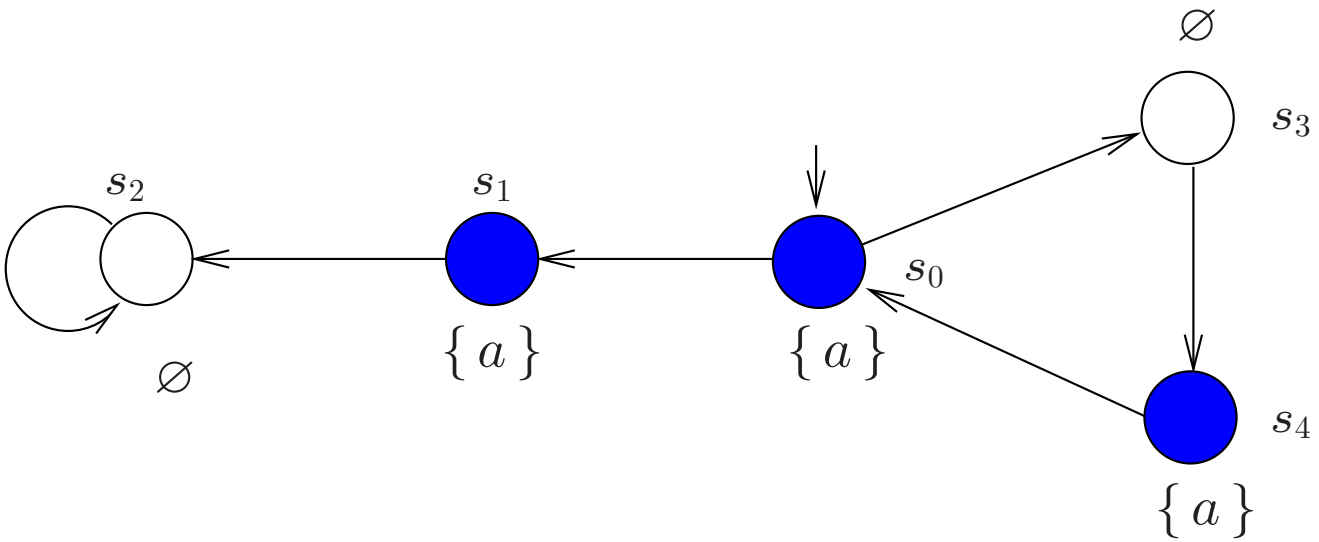
# Comparing LTL and CTL (1)

$\Diamond(a \wedge \bigcirc a)$ is not equivalent to $\forall\Diamond(a \wedge \forall\bigcirc a)$



# Comparing LTL and CTL (1)

$\Diamond(a \ \wedge \ \bigcirc a)$ is not equivalent to $\forall\Diamond(a \ \wedge \ \forall\bigcirc a)$

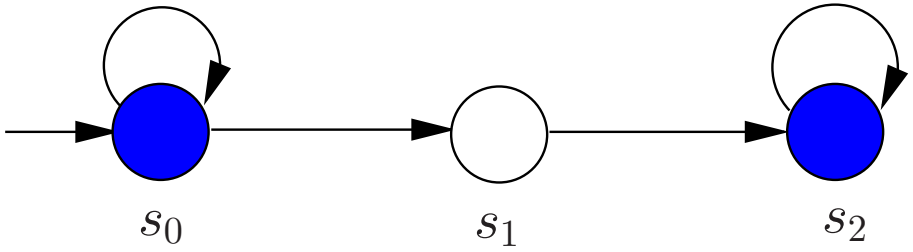



$s_0 \models \Diamond(a \ \wedge \ \bigcirc a)$ but $s_0 \not\models \forall\Diamond(a \ \wedge \ \forall\bigcirc a)$

since path $s_0\, s_1\, (s_2)^\omega$ violates $\Diamond(a \ \wedge \ \forall\bigcirc a)$

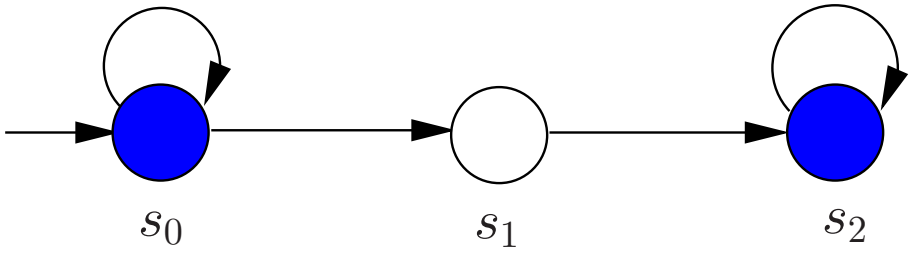
# Comparing LTL and CTL (2)

$\forall\Diamond\forall\Box a$ is not equivalent to $\Diamond\Box a$



# Comparing LTL and CTL (2)

$\forall\Diamond\forall\Box a$ is not equivalent to $\Diamond\Box a$

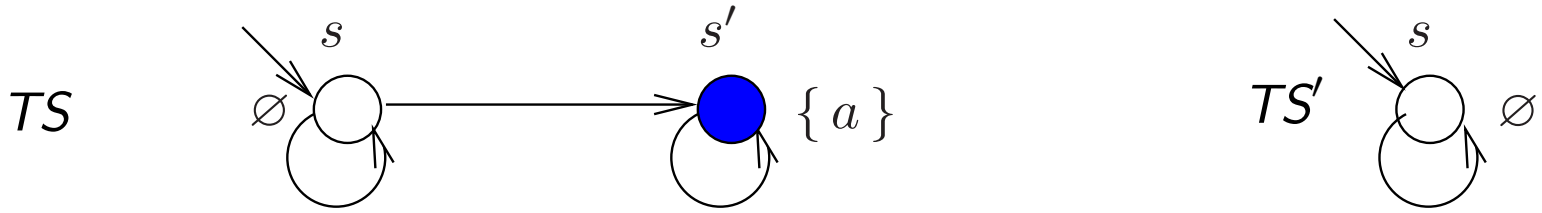



$s_0 \models \Diamond\Box a$ but $s_0 \not\models \forall\Diamond\forall\Box a$

since path $s_0^\omega$ violates $\Diamond\forall\Box a$

# Comparing LTL and CTL (3)

- No LTL-formula $\varphi$ is equivalent to $\forall\Box\exists\Diamond a$
- This is shown by contradiction: assume $\varphi \equiv \forall\Box\exists\Diamond a$; let:



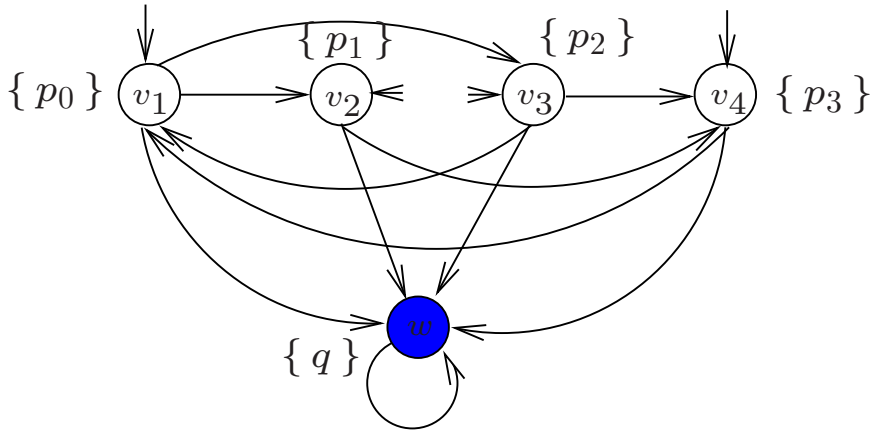

- $TS \models \forall\Box\exists\Diamond a$, and thus—by assumption—$TS \models \varphi$
- $Paths(TS') \subseteq Paths(TS)$, thus $TS' \models \varphi$
- But $TS' \not\models \forall\Box\exists\Diamond a$ as path $s^\omega \not\models \Box\exists\Diamond a$

# Model-checking LTL versus CTL

- Let *TS* be a transition system with $N$ states and $M$ transitions
- Model-checking LTL-formula $\Phi$ has time-complexity $\mathcal{O}((N+M)\cdot 2^{|\,\Phi\,|})$
  - linear in the state space of the system model
  - exponential in the length of the formula
- Model-checking CTL-formula $\Phi$ has time-complexity $\mathcal{O}((N+M)\cdot|\,\Phi\,|)$
  - linear in the state space of the system model and the formula
- Is model-checking CTL more efficient? No!

# Model-checking LTL versus CTL

$\Rightarrow$ LTL-formulae can be *exponentially shorter* than their equivalent in CTL




- Existence of Hamiltonian path in LTL: $\neg \left( (\Diamond p_0 \wedge \ldots \wedge \Diamond p_3) \ \wedge \ \bigcirc^4 q \right)$
- In CTL, all possible ($= 4!$) routes need to be encoded

# Content of this lecture

- Computation tree logic
  - syntax, semantics, equational laws
- CTL model checking
  - recursive descent, backward reachability, complexity
- Comparing LTL and CTL
  - what can be expressed in CTL?, what in LTL?, efficiency

$\Rightarrow$ Fairness

  - fair CTL semantics, model checking

# Fairness constraints in CTL

- For LTL it holds: $TS \models_{fair} \varphi$ if and only if $TS \models (fair \rightarrow \varphi)$
- An analogous approach for CTL is not possible!
- Formulas form $\forall(fair \rightarrow \varphi)$ and $\exists(fair \wedge \varphi)$ needed
- But: boolean combinations of path formulae are not allowed in CTL
- and: e.g., strong fairness constraints $\Box\Diamond b \rightarrow \Box\Diamond c \equiv \Diamond\Box\neg b \vee \Diamond\Box c$
  - cannot be expressed in CTL since persistence properties cannot
- Solution: change the semantics of CTL by ignoring unfair paths

# CTL fairness constraints

- A *strong CTL fairness constraint* is a formula of the form:

$$sfair = \bigwedge_{0<i\leqslant k} (\Box\Diamond\Phi_i \rightarrow \Box\Diamond\Psi_i)$$

  - where $\Phi_i$ and $\Psi_i$ (for $0 < i \leqslant k$) are CTL-formulas over *AP*
  - weak and unconditional CTL fairness constraints are defined analogously, e.g.

$$ufair = \bigwedge_{0<i\leqslant k} \Box\Diamond\Psi_i \quad \text{and} \quad wfair = \bigwedge_{0<i\leqslant k} (\Diamond\Box\Phi_i \rightarrow \Box\Diamond\Psi_i)$$

  - a CTL fairness assumption $fair$ is a combination of $ufair$, $sfair$ and $wfair$

$\Rightarrow$ a CTL fairness constraint is an LTL formula over CTL state formulas!

  - note that $s \models \Phi_i$ and $s \models \Psi_i$ refer to standard (unfair!) CTL semantics

# Semantics of fair CTL

For CTL fairness assumption $fair$, relation $\models_{fair}$ is defined by:

$$
\begin{array}{lll}
s \models_{fair} a & \text{iff} & a \in \mathit{Label}(s) \\
s \models_{fair} \neg \Phi & \text{iff} & \neg\,(s \models_{fair} \Phi) \\
s \models_{fair} \Phi \vee \Psi & \text{iff} & (s \models_{fair} \Phi) \vee (s \models_{fair} \Psi) \\
s \models_{fair} \exists\varphi & \text{iff} & \pi \models_{fair} \varphi \text{ for } \textit{some fair} \text{ path } \pi \text{ that starts in } s \\
s \models_{fair} \forall\varphi & \text{iff} & \pi \models_{fair} \varphi \text{ for } \textit{all fair} \text{ paths } \pi \text{ that start in } s
\end{array}
$$

$$
\begin{array}{ll}
\pi \models_{fair} \bigcirc\Phi & \text{iff } \pi[1] \models_{fair} \Phi \\
\pi \models_{fair} \Phi \,\mathsf{U}\, \Psi & \text{iff } (\exists\, j \geqslant 0.\, \pi[j] \models_{fair} \Psi \;\wedge\; (\forall\, 0 \leqslant k < j.\, \pi[k] \models_{fair} \Phi))
\end{array}
$$

$\pi$ is a fair path iff $\pi \models_{LTL} fair$ for CTL fairness assumption $fair$

# Transition system semantics

- For CTL-state-formula $\Phi$, and fairness assumption $fair$:

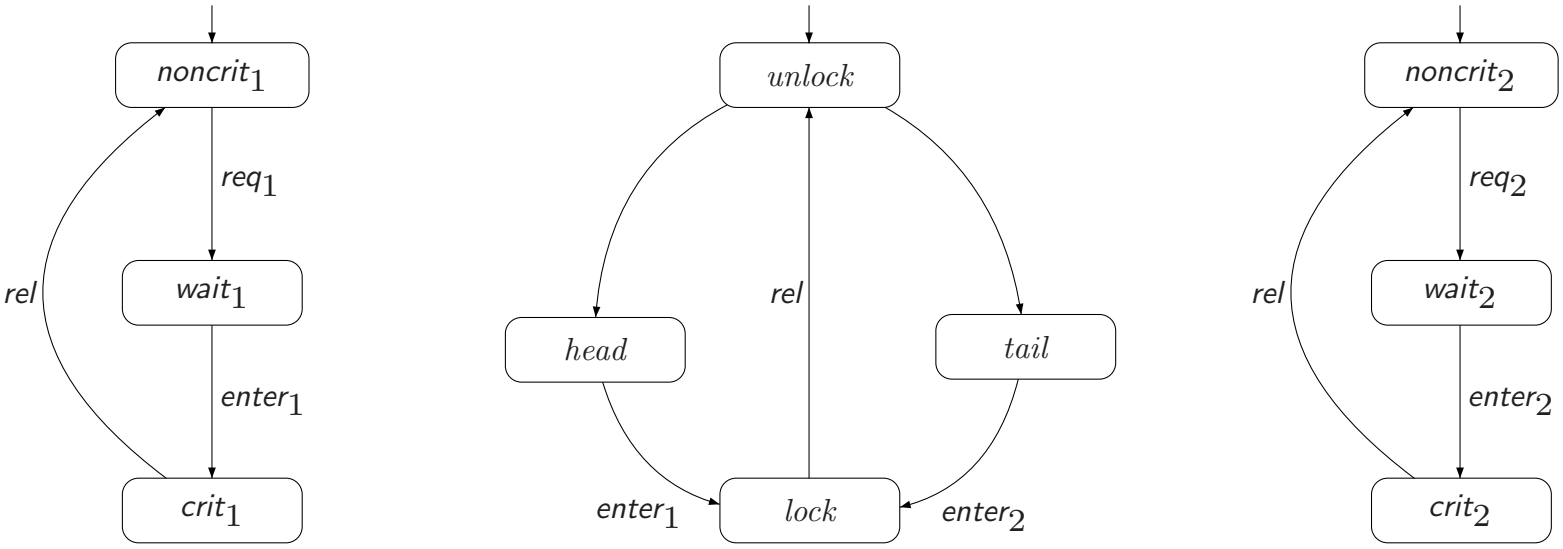
$$Sat_{fair}(\Phi) \;=\; \{\, s \in S \mid s \models_{fair} \Phi \,\}$$

- *TS* satisfies CTL-formula $\Phi$ iff $\Phi$ holds in all its initial states:

$$TS \models_{fair} \Phi \quad \text{if and only if} \quad \forall s_0 \in I.\, s_0 \models_{fair} \Phi$$

  - this is equivalent to $I \subseteq Sat_{fair}(\Phi)$

# Randomized arbiter




$$TS_1 \parallel Arbiter \parallel TS_2 \not\models (\forall\Box\forall\Diamond\, crit_1) \wedge (\forall\Box\forall\Diamond\, crit_2)$$

But: $TS_1 \parallel Arbiter \parallel TS_2 \models_{fair} \forall\Box\forall\Diamond crit_1 \wedge \forall\Box\forall\Diamond crit_2$ with

$$fair = \Box\Diamond head \wedge \Box\Diamond tail$$

# Fair CTL model-checking problem

For:

- finite transition system $TS$ without terminal states
- CTL formula $\Phi$ in ENF, and
- CTL fairness assumption $fair$

establish whether or not:

$$TS \models_{fair} \Phi$$

use bottom-up procedure à la CTL to determine $Sat_{fair}(\Phi)$
using as much as possible standard CTL model-checking algorithms

# CTL fairness constraints

- A *strong CTL fairness constraint*: $sfair = \bigwedge_{0<i\leqslant k} (\Box\Diamond\Phi_i \rightarrow \Box\Diamond\Psi_i)$
  - where $\Phi_i$ and $\Psi_i$ (for $0 < i \leqslant k$) are CTL-formulas over *AP*
- Replace the CTL state-formulas in $sfair$ by fresh atomic propositions:

$$sfair := \bigwedge_{0<i\leqslant k} (\Box\Diamond a_i \rightarrow \Box\Diamond b_i)$$

  - where $a_i \in L(s)$ if and only if $s \in Sat(\Phi_i)$ (not $Sat_{fair}(\Phi_i)$!)
  - ... $b_i \in L(s)$ if and only if $s \in Sat(\Psi_i)$ (not $Sat_{fair}(\Psi_i)$!)
  - (for unconditional and weak fairness this goes similarly)
- Note: $\pi \models fair$ iff $\pi[j..] \models fair$ for some $j \geqslant 0$ iff $\pi[j..] \models fair$ for all $j \geqslant 0$

# Results for $\models_{fair}$ (1)

$s \models_{fair} \exists \bigcirc a$ if and only if $\exists s' \in \mathit{Post}(s)$ with $s' \models a$ and $\mathit{FairPaths}(s') \neq \varnothing$

$s \models_{fair} \exists (a \,\mathsf{U}\, a')$ if and only if there exists a finite path fragment

$$s_0\, s_1\, s_2 \ldots s_{n-1} s_n \in \mathit{Paths}_{fin}(s) \quad \text{with } n \geqslant 0$$

such that $s_i \models a$ for $0 \leqslant i < n$, $s_n \models a'$, and $\mathit{FairPaths}(s_n) \neq \varnothing$

# Results for $\models_{fair}$ (2)

$s \models_{fair} \exists \bigcirc a$ if and only if $\exists s' \in \mathit{Post}(s)$ with $s' \models a$ and $\underbrace{\mathit{FairPaths}(s') \neq \varnothing}_{s' \models_{fair} \exists \Box \mathsf{true}}$

$s \models_{fair} \exists (a \,\mathsf{U}\, a')$ if and only if there exists a finite path fragment

$$s_0\, s_1\, s_2 \ldots s_{n-1} s_n \in \mathit{Paths}_{fin}(s) \quad \text{with } n \geqslant 0$$

such that $s_i \models a$ for $0 \leqslant i < n$, $s_n \models a'$, and $\underbrace{\mathit{FairPaths}(s_n) \neq \varnothing}_{s_n \models_{fair} \exists \Box \mathsf{true}}$

# Basic algorithm

- Determine $\mathit{Sat}_{\mathit{fair}}(\exists\Box\mathsf{true}) \;=\; \{\, s \in S \mid \mathit{FairPaths}(s) \neq \varnothing \,\}$
- Introduce an atomic proposition $a_{\mathit{fair}}$ and adjust labeling where:
  - $a_{\mathit{fair}} \in L(s)$ if and only if $s \in \mathit{Sat}_{\mathit{fair}}(\exists\Box\mathsf{true})$
- Compute the sets $\mathit{Sat}_{\mathit{fair}}(\Psi)$ for all subformulas $\Psi$ of $\Phi$ (in ENF) by:

$$
\begin{array}{rcl}
\mathit{Sat}_{\mathit{fair}}(a) & = & \{\, s \in S \mid a \in L(s) \,\} \\
\mathit{Sat}_{\mathit{fair}}(\neg a) & = & S \setminus \mathit{Sat}_{\mathit{fair}}(a) \\
\mathit{Sat}_{\mathit{fair}}(a \wedge a') & = & \mathit{Sat}_{\mathit{fair}}(a) \cap \mathit{Sat}_{\mathit{fair}}(a') \\
\mathit{Sat}_{\mathit{fair}}(\exists\bigcirc a) & = & \mathit{Sat}\,(\exists\bigcirc(a \wedge a_{\mathit{fair}})) \\
\mathit{Sat}_{\mathit{fair}}(\exists(a \,\mathsf{U}\, a')) & = & \mathit{Sat}\,\big(\exists(a \,\mathsf{U}\, (a' \wedge a_{\mathit{fair}}))\big) \\
\mathit{Sat}_{\mathit{fair}}(\exists\Box a) & = & \ldots\ldots
\end{array}
$$

- Thus: model checking CTL under fairness constraints is
  - CTL model checking + algorithm for computing $\mathit{Sat}_{\mathit{fair}}(\exists\Box a)$!

# Model checking CTL with fairness

The model-checking problem for CTL with fairness can be reduced to:

(1) the model-checking problem for CTL (without fairness), and

(2) the problem of computing $Sat_{fair}(\exists\Box a)$ for $a \in AP$

note that $\exists\Box\text{true}$ is a special case of $\exists\Box a$

thus a single algorithm suffices for $Sat_{fair}(\exists\Box a)$ and $Sat_{fair}(\exists\Box\text{true})$

# Core model-checking algorithm

(* states are assumed to be labeled with $a_i$ and $b_i$ *)

compute $\mathit{Sat}_{\mathit{fair}}(\exists\Box\text{true}) = \{ s \in S \mid \mathit{FairPaths}(s) \neq \varnothing \}$
**forall** $s \in \mathit{Sat}_{\mathit{fair}}(\exists\Box\text{true})$ **do** $L(s) := L(s) \cup \{ a_{\mathit{fair}} \}$ **od**

(* compute $\mathit{Sat}_{\mathit{fair}}(\Phi)$ *)

**for all** $0 < i \leqslant |\,\Phi\,|$ **do**
  **for all** $\Psi \in \mathit{Sub}(\Phi)$ with $|\,\Psi\,| = i$ **do**
    **switch**($\Psi$):

| | | |
|---|---|---|
| true | : | $\mathit{Sat}_{\mathit{fair}}(\Psi) := S$; |
| $a$ | : | $\mathit{Sat}_{\mathit{fair}}(\Psi) := \{ s \in S \mid a \in L(s) \}$; |
| $a \wedge a'$ | : | $\mathit{Sat}_{\mathit{fair}}(\Psi) := \{ s \in S \mid a, a' \in L(s) \}$; |
| $\neg a$ | : | $\mathit{Sat}_{\mathit{fair}}(\Psi) := \{ s \in S \mid a \notin L(s) \}$; |
| $\exists\bigcirc a$ | : | $\mathit{Sat}_{\mathit{fair}}(\Psi) := \mathit{Sat}(\exists\bigcirc(a \wedge a_{\mathit{fair}}))$; |
| $\exists(a \,\mathsf{U}\, a')$ | : | $\mathit{Sat}_{\mathit{fair}}(\Psi) := \mathit{Sat}(\exists(a \,\mathsf{U}\, (a' \wedge a_{\mathit{fair}})))$; |
| $\exists\Box a$ | : | compute $\mathit{Sat}_{\mathit{fair}}(\exists\Box a)$ |

    **end switch**
    replace all occurrences of $\Psi$ (in $\Phi$) by the fresh atomic proposition $a_\Psi$
    **forall** $s \in \mathit{Sat}_{\mathit{fair}}(\Psi)$ **do** $L(s) := L(s) \cup \{ a_\Psi \}$ **od**
  **od**
**od**
**return** $I \subseteq \mathit{Sat}_{\mathit{fair}}(\Phi)$

# Characterization of $Sat_{fair}(\exists\Box a)$

$$s \models_{sfair} \exists\Box a \quad \text{where} \quad sfair = \bigwedge_{0<i\leqslant k} (\Box\Diamond a_i \rightarrow \Box\Diamond b_i)$$

iff there exists a finite path fragment $s_0 \ldots s_n$ and a cycle $s'_0 \ldots s'_r$ with:

1. $s_0 = s$ and $s_n = s'_0 = s'_r$
2. $s_i \models a$, for any $0 \leqslant i \leqslant n$, and $s'_j \models a$, for any $0 \leqslant j \leqslant r$, and
3. $Sat(a_i) \cap \{ s'_1, \ldots, s'_r \} = \varnothing$ or $Sat(b_i) \cap \{ s'_1, \ldots, s'_r \} \neq \varnothing$ for $0 < i \leqslant k$

# Computing $Sat_{fair}(\exists\Box a)$

- Consider only state $s$ if $s \models a$, otherwise *eliminate* $s$
  - change $TS$ into $TS[a] = (S', Act, \rightarrow', I', AP, L')$ with $S' = Sat(a)$,
  - $\rightarrow' = \rightarrow \cap (S' \times Act \times S')$, $I' = I \cap S'$, and $L'(s) = L(s)$ for $s \in S'$

  $\Rightarrow$ each infinite path fragment in $TS[a]$ satisfies $\Box a$

- $s \models_{fair} \exists\Box a$ iff there is a non-trivial SCC $D$ in $TS[a]$ reachable from $s$:

$$D \cap Sat(a_i) = \varnothing \quad \text{or} \quad D \cap Sat(b_i) \neq \varnothing \quad \text{for} \quad 0 < i \leqslant k \qquad (*)$$

- $Sat_{sfair}(\exists\Box a) = \{ s \in S \mid Reach_{TS[a]}(s) \cap T \neq \varnothing \}$
  - $T$ is the union of all non-trivial SCCs $C$ that contain $D$ satisfying (*)

how to compute the set $T$ of SCCs?

# Time complexity

For transition system $TS$ with $N$ states and $M$ transitions,

CTL formula $\Phi$, and CTL fairness constraint $fair$ with $k$ conjuncts,

the CTL model-checking problem $TS \models_{fair} \Phi$

can be determined in time $\mathcal{O}(\mid \Phi \mid \cdot (N + M) \cdot k)$