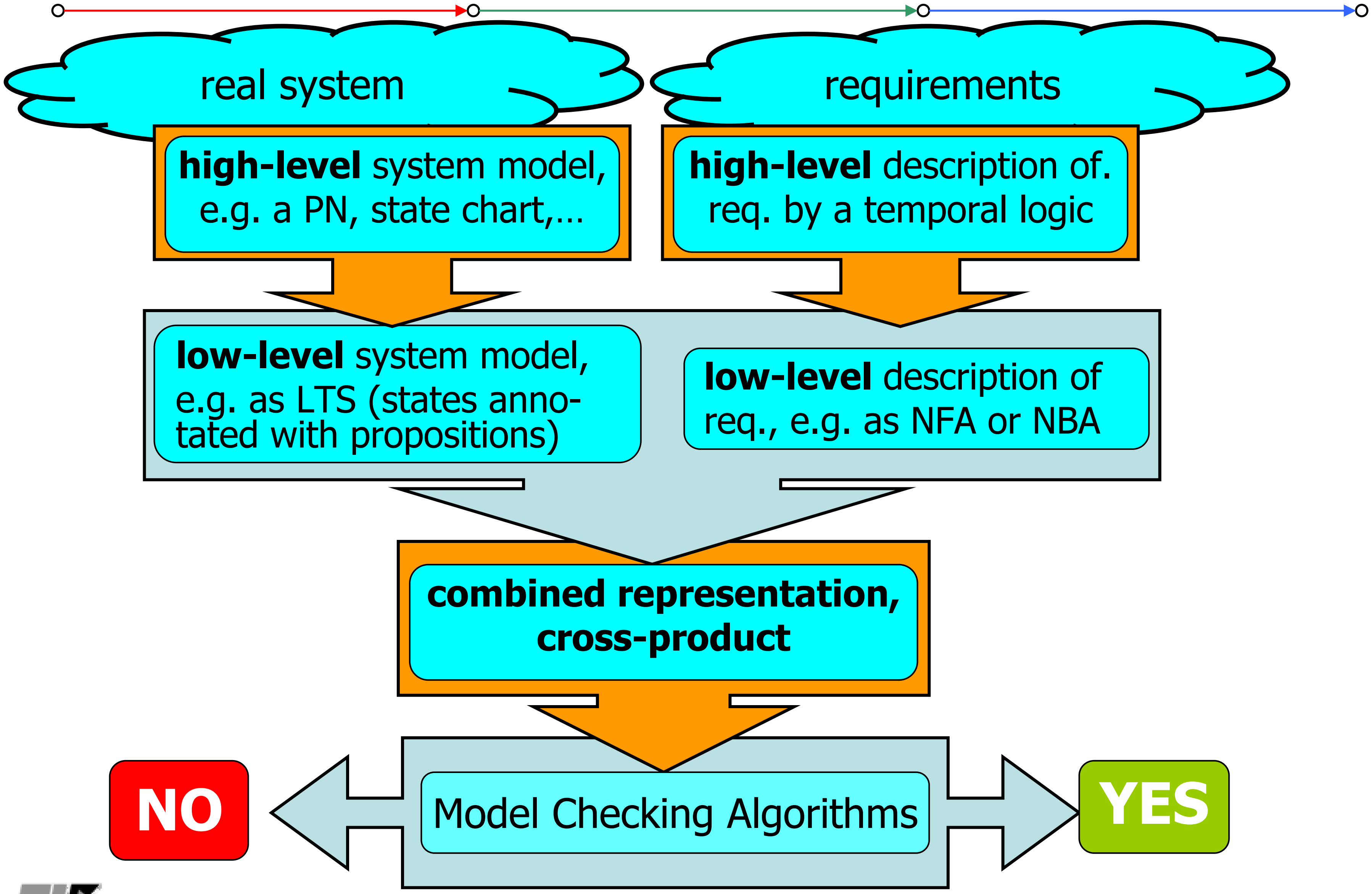
Where are we?

- SDL and MSC ✓
- Petri Nets
  - Notation
  - Behavioral Properties ✓
- Symbolic Analysis methods of finite models ✓
- **Introduction to model checking**
- **Timed automata (real-time)**
  - Notation
  - Semantics
  - Analysis

Model Checking (at glance)

- **Model checking** ist ein Verfahren zur vollautomatischen Ueberpruefung ob ein Systemmodell eine gegebene Eigenschaft erfüllt.
- Das Verfahren ist vollautomatisch, da es keiner Benutzerinteraktion bedarf, --im Gegensatz zu einigen deduktiven Verfahren, wie bspw. interaktives Theorembeweisen.
- Automatisierung setzt Berechnung voraus, was wiederum eine mathematische Formulierung des Problems bedingt. D.h. Systembeschreibung M und zu überprüfende Eigenschaft φ müssen mathematisch interpretierbar sein, so dass eine rechnergestützte Ueberprüfung, ob *M* ein Modell, also eine Realisierung von φ darstellt, moeglich ist.
- Die Systembeschreibung erfolgt formal, d.h. durch eine hochsprachliche Modellspezifikationsmethode oder direkt als ein beschriftetes Transitionssystem. Letzteres muss hierbei nicht notwendigerweise endlich sein, jedoch eine endlich Repräsentation besitzen, bspw. durch ein Quotiententransitionssystem (siehe Regionengraph im Falle von TA).
- Die nachzuweisende Eigenschaft ist dann durch eine temporal-logische Formel oder durch einen Beobachtungsautomaten anzugeben.

Model Checking: Automatic verification of system behaviours



real system

high-level system model, e.g. a PN, state chart,…

requirements

high-level description of. req. by a temporal logic

low-level system model, e.g. as LTS (states annotated with propositions)

low-level description of req., e.g. as NFA or NBA

combined representation, cross-product

Model Checking Algorithms

NO

YES

Temproal logics, important remarks

- Temporal extensions of *propositional* – or predicate logics by modalities have been found very useful for specifying requirements to be verified.
- The **elementary modalities** referring to the infinite behavior of reactive systems are
  - $\Diamond$ eventually (eventually in the future)
  - $\Box$ always (now and forever)
- Temporal logics are **time abstract**, i.e the above modalities allow ordering of state labels / event labels, but not a specification of state residence times or transition duration, as expected for real-time behaviors. However, real-time extension due exist, but will not be covert here.
- Time can be viewed as **linear** or **branching**, i.e. one either associate a single succesor moment for each instant in time or reasons over all alternative concurses, where in the latter case one obtain path structures and in the latter case tree structures to reason about.
- The discussion in this lecture is limited to linear time logics.

Model Checking: Historic development

- In the late 70ties Pnueli proposed the usage of temporal logics and related modal logics for the verification of complex computer systems.

- Two early contenders in formal verifications were Linear Temporal Logic (a linear time logic by Amir Pnueli and Zohar Manna) and Computation Tree Logic, a branching time logic by Edmund Clarke and E. Allen Emerson.

- Pioneering work in the model checking of temporal logic formulae was done by E. M. Clarke and E. A. Emerson in 1981 and by J. P. Queille and J. Sifakis in 1982.

- Clarke, Emerson, and Sifakis shared the 2007 Turing Award for their work on model checking

Linear Time Properties (Definitions)

- Via state space exploration high-level models can be transformed into labelled transition Systems (LTS).
- Extending earlier definition we assume that each state of the LTS is labelled by some set of atomic propositions.
- Thus we define a LTS as follows: A LTS is a tuple $(\mathcal{S}, \mathcal{A}ct, \longrightarrow, \mathcal{I}, \mathcal{A}P, \mathcal{L})$

  $\mathcal{S}$: is the set of reachable states

  $\mathcal{I}$: is the set of initial states

  $\mathcal{A}ct$: is the set of activity/action labels

  $\longrightarrow \subseteq \mathcal{S} \times \mathcal{A}ct \times \mathcal{S}$: is a transition relation

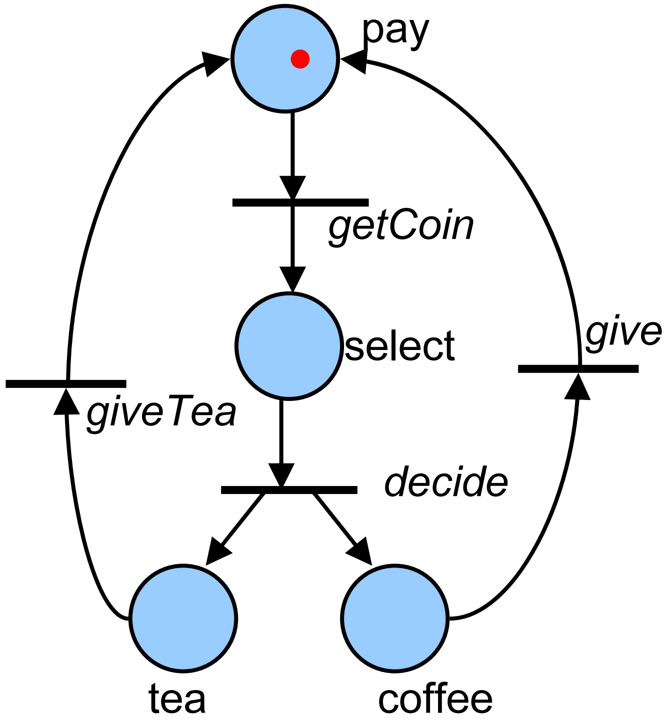
  $\mathcal{A}P$: is the set of atomic propositions

  $\mathcal{L} \to 2^{\mathcal{A}P}$: is the state labelling function

- In the following we emphasize a state-based approach, thus the transition labels are ignored, but also action-based approaches exisit.

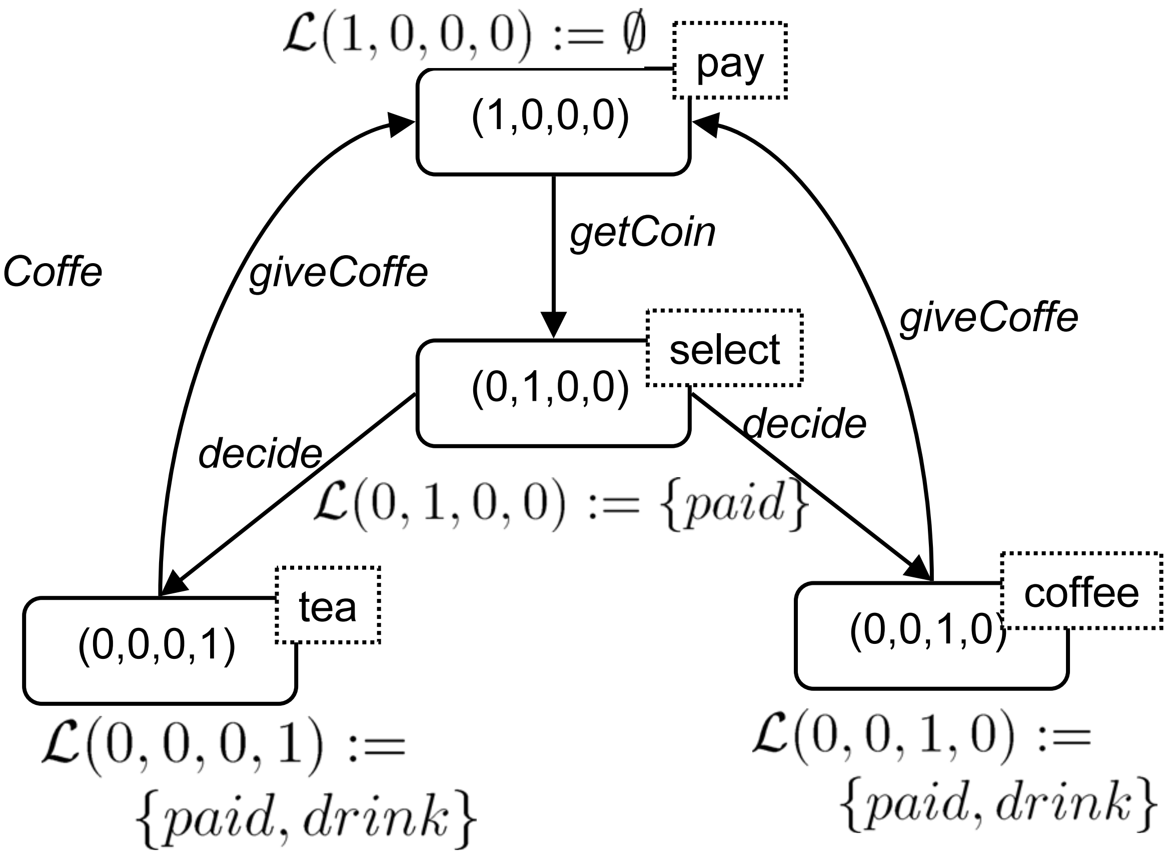
Linear Time Properties (Definitions)

A PN specifying a breverage vending machine



The underlying LTS, we omitted the act.-labels

$$TS := (\mathcal{S}, \mathcal{A}ct, \rightarrow, (1,0,0,0), \mathcal{AP} := \{paid, drink\}, \mathcal{L})$$


$\mathcal{L}(1,0,0,0) := \emptyset$
pay
(1,0,0,0)
getCoin
giveCoffe
giveCoffe
(0,1,0,0)
select
decide
decide
$\mathcal{L}(0,1,0,0) := \{paid\}$
(0,0,0,1)
tea
(0,0,1,0)
coffee
$\mathcal{L}(0,0,0,1) := \{paid, drink\}$
$\mathcal{L}(0,0,1,0) := \{paid, drink\}$

Linear Time Properties (Definitions)

- For each state s of a LTS TS one may define a post- and a preset:

$$Post(s) := \bigcap_{\alpha \in Acts} \{s' \in \mathcal{S} | (s, \alpha, s') \in \rightarrow_{TS}\}$$

$$Pre(s) := \bigcap_{\alpha \in Acts} \{s' \in \mathcal{S} | (s', \alpha, s) \in \rightarrow_{TS}\}$$

- These sets can be extended to the level of input sets:

$$Pre(\mathcal{C}) := \bigcap_{s \in \mathcal{C}} Pre(s) \qquad Post(\mathcal{C}) := \bigcap_{s \in \mathcal{C}} Post(s)$$

- and to the transitive closures $Pre^*(s), Pre^*(\mathcal{C}), Post^*(s), Post^*(\mathcal{C})$

- We have $Reach(TS) := Post^*(\mathcal{I})$

Linear Time Properties (Definitions)

- A LTS $\mathcal{TS}$ is denoted as deadlock-free iff ∀ s ∈ $\mathcal{S}$: Post(s) ≠ ø.

- A finite/infinite trace of a deadlock-free LTS $\mathcal{TS}$ is defined by a finite/infinite path as follows:

$$\hat{\pi} := s_0 s_1 s_2 \ldots s_n \text{ with } trace(\hat{\pi}) = \mathcal{L}(s_0)\mathcal{L}(s_1)\mathcal{L}(s_2)\ldots\mathcal{L}(s_n)$$

$$\pi := s_0 s_1 s_2 \ldots\ldots \text{ with } trace(\pi) = \mathcal{L}(s_0)\mathcal{L}(s_1)\mathcal{L}(s_2)\ldots\ldots$$

- Thus a trace of an LTS is a finite/infinite word over the alphabet 2^{AP}

- Traces can now be extended to sets of traces

$$trace(\Pi) := \{trace(\pi) | \pi \in \Pi\}$$

Linear Time Properties (Definitions)

- A finite path fragment $\hat{\pi}$ of a LTS TS is a finite sequence

$$\hat{\pi} := s_0 s_1 s_2 \ldots s_n \text{ such that } s_i \in Post(s_{i-1}) \forall i : 0 < i \leq n \text{ where } n > 0$$

- A infinite path fragment π of trace of a LTS TS is a infinite sequence

$$\pi := s_0 s_1 s_2 \ldots\ldots \text{ such that } s_i \in Post(s_{i-1}) \forall i : i > 0$$

- A maximal path fragment is an infinite p.f. or a finite p.f. ending in a deadlock state

- An initial path fragment is a p.f. starting in an initial state of the LTS

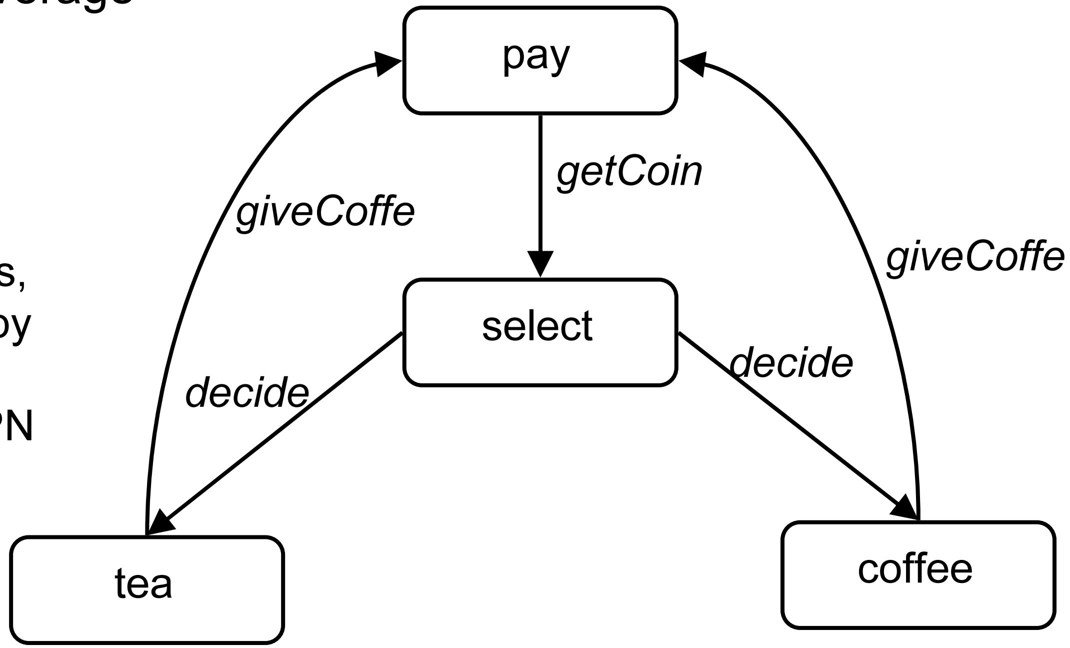
- A path of a LTS TS is an initial maximal p.f..

- Let Paths(TS) denote the set of all paths of TS and let Paths$_{\text{fin}}$(TS) denote the set of all finite paths of TS.

Linear Time Properties (Definitions)

- The LTS of the breverage vending maschine:

instead of state vectors, each state is labelled by the identifier of the marked place on the PN



$$\pi_1 := pay\, select\, tea\, pay\, select\, tea\ \ldots\ldots\ \text{path?}$$

$$\pi_2 := select\, tea\, pay\, select\, coffee\ \ldots\ldots\ \text{path?}$$

$$\hat{\pi}_1 := pay\, select\, tea\, pay\, select\, coffee.\ \text{path?}$$

Linear Time Properties

- A linear time (LT) property can be seen as the role-model trace that a (L)TS $\mathcal{TS}$ should exhibit, i.e. it is defined as requirement over all words over $\mathcal{AP}$ of $\mathcal{TS}$

- A LT prop. over the set of atomic propositions is a subset of $\left(2^{\mathcal{AP}}\right)^{\omega}$ which is the infinite concatenation of words $2^{\mathcal{AP}}$

- Remark: Solely infinite words matter, since we consider TS which do not have terminal states, i.e. which are deadlock-free

- Let $\mathcal{P}$ be a LT property and let $\mathcal{TS}$ be a TS (without terminal states):
  - $\mathcal{TS} \vDash \mathcal{P} \Leftrightarrow$ *Traces*$(\mathcal{TS}) \subseteq \mathcal{P}$ ($\mathcal{TS}$ satisfies $\mathcal{P}$ iff the set of all traces of $\mathcal{TS}$ is included in the set of all words induced by $\mathcal{P}$)
  - $s \vDash \mathcal{P} \Leftrightarrow$ *Traces*$(s) \subseteq \mathcal{P}$ (a state s satisfies $\mathcal{P}$ iff the set of all traces starting in s is included in the set of all words induced by $\mathcal{P}$)

What are the requirements to be verified ? (from intro)

1. **Safety**: A safety property to be verified asserts that a system under analysis never reaches a (set of) dedicated state, e.g. like error states, or in particular a deadlock. The mutual exclusion property is one of the most prominent examples of a safety property.
   (constraint on finite behaviour)

2. **Liveness or progress**: A liveness property guarantees that a system under analysis is executing a (set) of dedicated activities infinitely often (constraint on infinite behaviour)

At first we will look at so called state invariants. A state invariant requires that a property P holds for all reachable states!

Safety properties and Invariants

- An LT prop. $\mathcal{P}_{\text{inv}}$ over $\mathcal{AP}$ is an invariant if there is a propositional logic formulae Ψ over $\mathcal{AP}$ s.t.

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

- Ψ is called a invariant condition of $\mathcal{P}_{\text{inv}}$ and A_i are the atomic propositions of state i

$$\begin{aligned}
\mathcal{TS} \models \mathcal{P}_{inv} \quad &\Longleftrightarrow \quad trace(\pi) \in \mathcal{P}_{inv} \text{ for all paths } \pi \text{ in } \mathcal{TS} \\
&\Longleftrightarrow \quad \mathcal{L}(s) \models \Psi \text{ for all states } s \ \in \pi \\
&\Longleftrightarrow \quad \mathcal{L}(s) \models \Psi \text{ for all states } \in Reach(\mathcal{TS})
\end{aligned}$$

Remark:
The condition Ψ has to be fulfilled by all initial states and its satisfaction is invariant under all transitions in the reachable fragment of $\mathcal{TS}$

- How can we check this?

DFS for generating LTS of high-level model and check if Ψ holds for each state

Safety properties

- Some safety properties can not be verified by considering the set of reachable states only. But, infinite traces violating such properties have a finite prefix, where the violation takes place, i.e. no infinite word with such a ***bad prefix*** satisfy the property.

- An LT prop. $\mathcal{P}_{\text{safe}}$ over $\mathcal{AP}$ is called a safety property if for all words

$$\sigma \in \left(2^{AP}\right)^{\omega} \setminus \mathcal{P}_{safe} \qquad \text{there exists a finite prefix } \tilde{\sigma} \text{ of } \sigma \text{ s.t.}$$

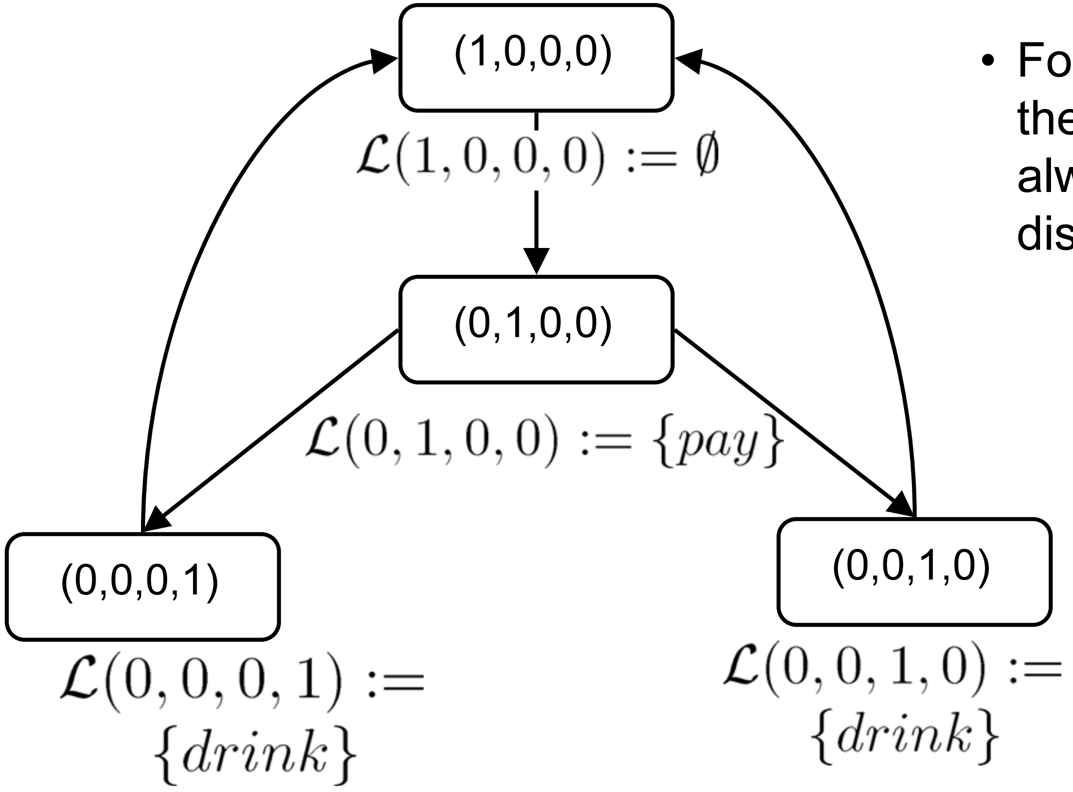
$$\mathcal{P}_{safe} \cap \{\sigma' \in \left(2^{AP}\right)^{\omega} \mid \tilde{\sigma} \text{ is a finite prefix of } \sigma'\} = \emptyset$$

<u>Remark:</u>

- Any such finite word $\hat{\sigma}$ is denoted **bad prefix**

- A safety property is a property the violation of which appears in a finite number of steps

Safety properties (Example)


(1,0,0,0)
$\mathcal{L}(1,0,0,0) := \emptyset$
(0,1,0,0)
$\mathcal{L}(0,1,0,0) := \{pay\}$
(0,0,0,1)
$\mathcal{L}(0,0,0,1) := \{drink\}$
(0,0,1,0)
$\mathcal{L}(0,0,1,0) := \{drink\}$

- For the breverage vending machine the number of inserted coins must always be at least the number of dispensed drinks.

$$|\{0 \le j \le i \mid pay \in A_j\}| \ge |\{0 \le j \le i \mid drink \in A_j\}|$$

<u>Bad prefixes are:</u>

ø {pay}{drink}{drink}, ø {pay}{drink}ø{pay}{drink}{drink}, etc.

Safety properties

- For a TS $\mathcal{TS}$ without terminal states and a safety property $\mathcal{P}_{\text{safe}}$ over $\mathcal{AP}$ we have:

$$\mathcal{T}S \models \mathcal{P}_{safe} \iff Trace_{fin}(\mathcal{T}S) \cap BadPref(\mathcal{P}_{safe}) = \emptyset$$

- $Trace_{fin}(\mathcal{T}S)$ is the set of finite traces starting in the initial state of $\mathcal{T}S$ and
- $BadPref(\mathcal{P}_{safe})$ is the set of bad prefixes induced by $\mathcal{P}_{safe}$, i.e. it adresses all words in $(2^{\mathcal{A}P})^{\omega} \setminus \mathcal{P}_{safe}$.

How-can the check such properties?

Model checking regular safety properties

- A safety property $\mathcal{P}_{\text{safe}}$ over $\mathcal{AP}$ is denoted as regular safety property if its set of bad prefixes constitutes a regular language over $2^{\mathcal{AP}}$.

- Every invariant is a regular safety property: Let Ψ be the state condition to be fulfilled by *Reach(s)*. Then the language of bad prefixes consists of the word

$$A_0 A_1 A_2 \ldots A_n \text{ s.t. } A_i \not\models \Psi \text{ for some } 0 \leq i \leq n.$$

Such languages can be characterized by

$$\Psi^*(\neg\Psi)\text{true}^*$$

which is a *regular expression*

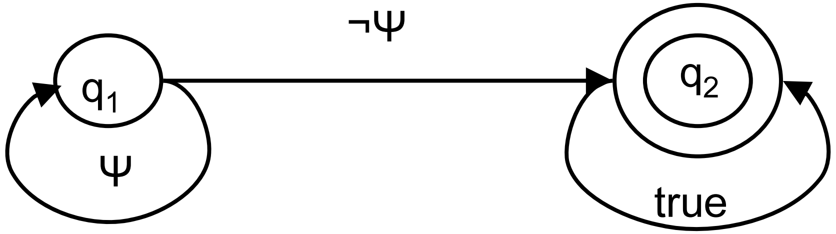
Model checking regular safety properties

$\mathcal{AP}$:= {a,b} Ψ := a v ¬b

- Ψ stands for the regular expresion {} v {a} v {a,b}
- ¬Ψ stands for the regular expression {b}
- true stands for {},{a},{a,b},{b}
- The bad prefixes over the state condition (a v ¬b) are given as

$$E = \underbrace{(\{\} \vee \{a\} \vee \{a,b\})^*}_{\Psi^*}\underbrace{\{b\}}_{(\neg\Psi)}\underbrace{(\{\}, \ldots \vee \{a,b\} \vee \{b\})^*}_{\text{true}^*}$$

- Thus the languag $\mathcal{L}$(E) consists of all words $A_1A_2\ldots A_n$ s.t. A_i = b for 0 < i ≤ n, since A $\nvDash$ a v ¬b iff A = b.
- Since $\mathcal{L}$ (E) is a regular language, we can construct an accepting NFA



Model checking regular safety properties

- Idea

$$\mathit{Traces}_{fin}(\mathcal{TS}) \cap \mathcal{L}(A) = \emptyset \Rightarrow \mathcal{TS} \models \mathcal{P}_{safe}$$

where A ist the automata accepting the minimal bad prefix of $\mathcal{P}_{\text{safe}}$

- One proceeds as follows:

1. Construct product automaton $\mathcal{TS} \otimes \mathcal{P}$:

$$\frac{\mathcal{TS}:\ \vec{s} \xrightarrow{\alpha} \vec{s} \wedge A:\ q \xrightarrow{\mathcal{L}(\vec{s}\,)} q'}{\mathcal{TS} \otimes A:\ (\vec{s}, q) \xrightarrow{\alpha} (\vec{s}', q')}$$

2. Check whether no state <s,q> is reachable where q ∈ F holds (F is the set of accepting or terminal states of A). Thus one needs simply to check the state condition Ψ := ¬F for the product-automaton. This can be done by a simple (naive) DFS reachabilty algorithm.

$$\Psi := \bigwedge_{q \in F} \neg q$$

Now we can check regular safety properties by means of invariants, referring to finite system behavior. Is this sufficient?

Liveness properties

- An <u>algorithm which does nothing fulfills a safety property</u>. Liveness or progress properties require that something good will happen in the future, e.g. a set of transitions will be eventually taken.

- A LT property $\mathcal{P}_{\text{live}}$ over $\mathcal{AP}$ is a liveness property whenever:

$$pref(\mathcal{P}_{live}) = \left(2^{AP}\right)^*$$

- **⇒ Thus a liveness property is an LT property $\mathcal{P}$ s.t. each finite word can be extended to an infinite word that satiesfy $\mathcal{P}$, such properties are violated by infinite runs**

Example (mutual exclusion from ex. class):

- each PN will eventually enter its critical section
- each PN will enter its critical section ∞-often
- each waiting PN will eventually enter its crit. section

Formally:

$$\bigwedge_i \overset{\infty}{\exists}\, j \geq 0 : crit_i \in A_j; \left(\overset{\infty}{\exists} \text{ stands for } \forall k \geq 0 : \exists j \geq k : ....\right)$$

ω-regular languages: Languages over infinite words

- An ω-regular expression G over the alphabet $\sum$ has the form:

$$G := E_1 F_1^\omega + \cdots + E_n F_n^\omega$$

where n ≥ 1 and $E_1,..,E_n, F_1,\ldots F_n$ are regular expressions over $\sum$ s.t. $\varepsilon \notin L(F_i)$ for all $1 \leq i \leq n$. The semantics of a ω-regular expression G is a language of **infinite words:**

$$\mathcal{L}_\omega(G) := \mathcal{L}(E_1).\mathcal{L}(F_1)^\omega \cup \ldots \cup \mathcal{L}(E_n).\mathcal{L}(F_n)^\omega$$

- where $L(E) \subseteq \sum^\omega$ denotes the language of finite words induced by a regular expression $E.$

- A language $L(E) \subseteq \sum^\omega$ is called ω-regular if $L \subseteq L_\omega(G)$ for some ω-regular expression G over the alphabet $\sum$ holds.

- ω-regular languages are closed under union, intersection and complement, e.g.

$$(2^{\mathcal{A}P})^\omega \setminus \mathcal{P}_{safe} = BadPref(\mathcal{P}_{safe}).(2^{\mathcal{A}P})^\omega$$

ω-regular properties

- Automata accepting ω-regular languages are denoted as ω-Automata, the Non-deterministic Buechi Automaton (NBA) is one of the simplest ω-Automata.

- The concept of ω-regular languages is very important, most relevant LT properties are ω-regular, thus they can be verified by employing the NBA.

- LT porperty over $\mathcal{AP}$ is called ω-regular if $\mathcal{P}$ is an ω-regular language over the alphabet $2^{\mathcal{AP}}$.

- Example: $\mathcal{AP} := \{a,b\}$, the invariant $\mathcal{P}_{\text{inv}}$ induced by the proposition Ψ := a ∨ ¬b.

$$\begin{aligned} \mathcal{P}_{inv} := \quad & \{A_0 A_1 A_2 \ldots \in (2^{\mathcal{A}P})^\omega \mid \forall i \geq 0 : (a \in A_i \text{ or } b \notin A_i)\} \\ & \{A_0 A_1 A_2 \ldots \in (2^{\mathcal{A}P})^\omega \mid \forall i \geq 0 : (A_i \in \{\{\},\{a\},\{a,b\}\} \end{aligned}$$

Non-deterministic Buechi Automaton

- A non-deterministic Buechi Automaton (NBA) A is a tuple: $(\mathcal{Q}, \Sigma, \mathcal{Q}_0, \delta, \mathcal{F})$

  $\mathcal{Q}$: is the set of states $\qquad$ Σ : is an alphabet

  $\mathcal{Q}_0 \subseteq \mathcal{Q}$: is the set of initial states

  $\delta : \mathcal{Q} \times \Sigma$: is the transition function, which induces: $\rightarrow \subseteq \mathcal{Q} \times \Sigma \times \mathcal{Q}$,

  which is given by $q \xrightarrow{A} q' \iff q' \in \delta(q, A)$

  $\mathcal{F} \subseteq \mathcal{Q}$: is the set of accepting or final state, denoted as acceptance set

- A run $\sigma := A_0 A_1 A_2 \ldots \in \Sigma^\omega$ addresses an infinite sequence of states

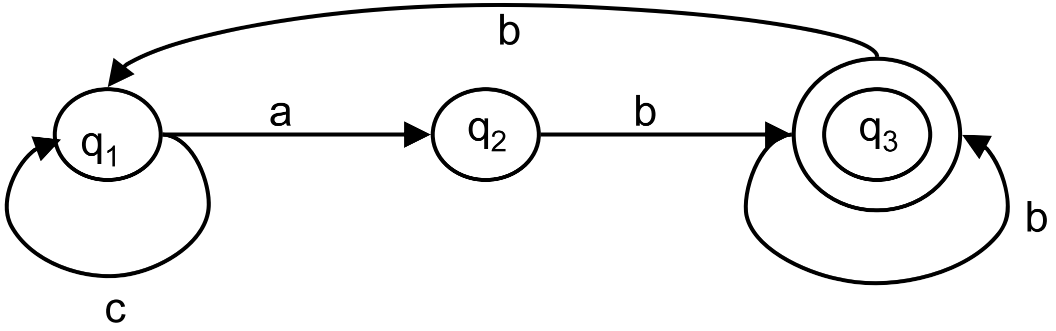
  $q_0 q_1 q_2 \ldots$ in $\mathcal{A}$ s.t. $q_0 \in \mathcal{Q}_0$ and $q_i \xrightarrow{A_i} q_{i+1}$ for $i \geq 0$, $A_i \in \Sigma$.

- A run is denoted as accepting if $q_i \in \mathcal{F}$ for ∞ many $i \in \mathbb{N}$

- The accepte language of $\mathcal{A}$ is the ω-regular language

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

Non-deterministic Buechi Automaton



- c^ω has only one run, namely $q_1q_1q_1q_{1....} = q_1^\omega$ since $q_1 \notin \mathscr{F}$ this run is a not accepting one.
- ab^ω yields the run $q_1q_2q_3q_{3....} = q_1q_2q_3^\omega$ since $q_3 \in \mathscr{F}$ and we visit q_3 ∞-often this run is accepting.
- What is about $(abb)^nc^\omega$?
- What is the ω-regular expression defining the accepting language of $\mathcal{A}$?

$$\mathbf{c^*ab(b^+ + bc^*ab)^\omega}$$

Model checking ω-regular properties

- Let $\mathcal{TS}$ be a finite TS without terminal states over $\mathcal{AP}$ and let $\mathcal{P}$ be an ω-regular property over $\mathcal{AP}$. Furthermore let $\mathcal{A}$ be a non-blocking NBA, i.e. δ(q, A) ≠ ø ∀ q ∈ Q and ∀ A ∈ ∑), that accepts $(2^{AP})^\omega \setminus \mathcal{P}$ It follows:

$$\mathcal{TS} \models \mathcal{P} \iff \mathcal{TS} \otimes \mathcal{A} \models \mathcal{P}_{pers(\mathcal{A})}$$

- $\mathcal{P}_{\text{pers}}$ is a persistence property. Such a property is an LT property which requires that some propositional logic formula Ψ "*eventually forever*" holds, i.e. formally

$$\mathcal{P}_{pers} := \{A_0A_1A_2\ldots \in (2^{AP})^\omega \mid \overset{\infty}{\forall} j : A_j \models \Psi\}$$

$$\text{where } \overset{\infty}{\forall} j \text{ means } \exists i \geq 0 : \forall j \geq i$$

- This gives, that infinite behaviors can be verified by persistence properties on the product automaton. How-do we do this?

Model checking ω-regular properties

- To check if the high-level model fulfills a persitence property $\mathcal{P}$ over a propositional logic formula Ψ, we must check if there exists cycle containing a ¬Ψ-state within the product automaton $\mathcal{TS} \otimes \mathcal{A}$.
- If this is the case one can conclude $\mathcal{TS} \not\models \mathcal{P}$.

- Formally:
  Let $\mathcal{TS}$ be non-terminal finite TS over $\mathcal{TS}$, Ψ a propositional formula over $\mathcal{AP}$ and $\mathcal{P}_{\text{pers}}$ the persitence property "eventually forever Ψ".
  Then, the following statements are equivalent:

(a) $\mathcal{TS} \not\models \mathcal{P}_{pers}$

(b) There exists a reachable ¬Ψ-state s which belongs to a cycle:

$$\exists s \in Reach(\mathcal{TS}) : s \not\models \Psi \wedge s \text{ is on a cycle in } \mathcal{TS}$$

- Such states can be found by nested-DFS algorithm, the path to such a state automatically provides the counter example

Model checking ω-regular properties (at glance)

- Construct product automaton $\mathcal{TS} \otimes \mathcal{A}$:

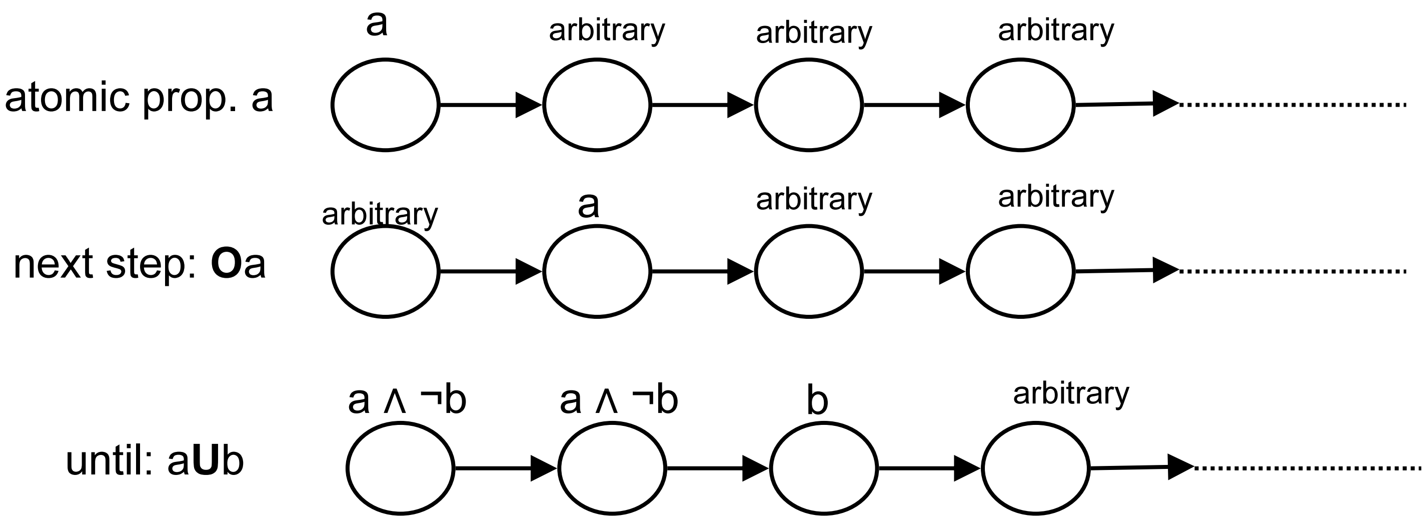
$$\frac{\mathcal{TS} :\ \vec{s} \xrightarrow{\alpha} \vec{s} \wedge A :\ q \xrightarrow{\mathcal{L}(\vec{s}\,)} q'}{\mathcal{TS} \otimes A :\ (\vec{s}, q) \xrightarrow{\alpha} (\vec{s}', q')}$$

- where $\mathcal{A}$ represents the negated property $(2^{AP})^{\omega} \setminus \mathcal{P}$
- Within the product automaton we check if there is a state <s,q> $\not\models \Psi$ and if this state is part of a cycle. This gives for $\Psi := \bigwedge_{q \in F} \neg q$ that we have a counter example that “eventually forever no accept state” holds.
- Thus we have a run, where $\neg\mathcal{P}$ holds forever
- We can conclude $\mathcal{TS} \not\models \mathcal{P}$
- Question: Why can we not simply check for a cycle containing acceptance states of the original ω-regular property?

Linear Temporal Logic

- Now (finally) we have the armamentarium for verifying properties specified by a temporal logic, here LTL.
- LTL-formuale over the set of $\mathcal{AP}$ atomic propositionsare formed as follows: $\varphi := true \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi_1 \bigcup \varphi_2$

  where O is the next-operator and U is the until-operator.
- Intuitive semantics of temporal modalities



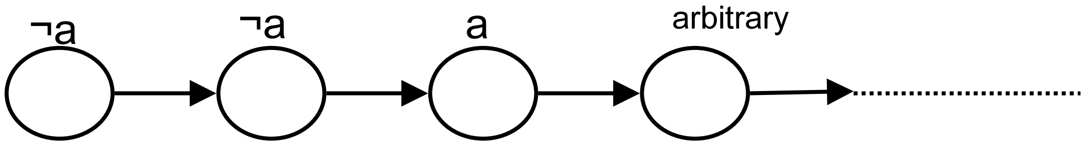
Linear Temporal Logic

- The until operator allows us now to define the elementary modalities until-operator.

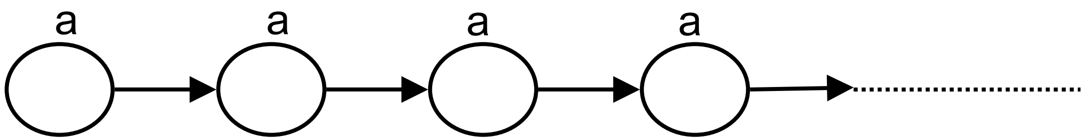
$$\begin{aligned}\Diamond\varphi &:= true \bigcup \varphi \\ \Box\varphi &:= \neg\Diamond\neg\varphi\end{aligned}$$

- Intuitive semantics of temporal modalities

eventually: ◊ a



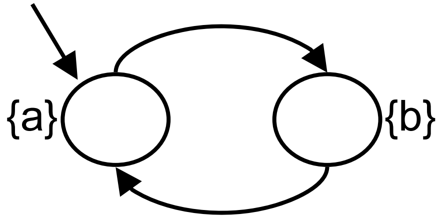
always: □a



Equivalence rules for LTL

- Duality law

$$\begin{aligned}
\neg \bigcirc \varphi &= \bigcirc \neg \varphi \\
\neg \Diamond \varphi &= \Box \neg \varphi \\
\neg \Box \varphi &= \Diamond \neg \varphi
\end{aligned}$$



$$\Diamond(a \wedge b) \overset{?}{=} \Diamond a \wedge \Diamond b$$

- Distributive law

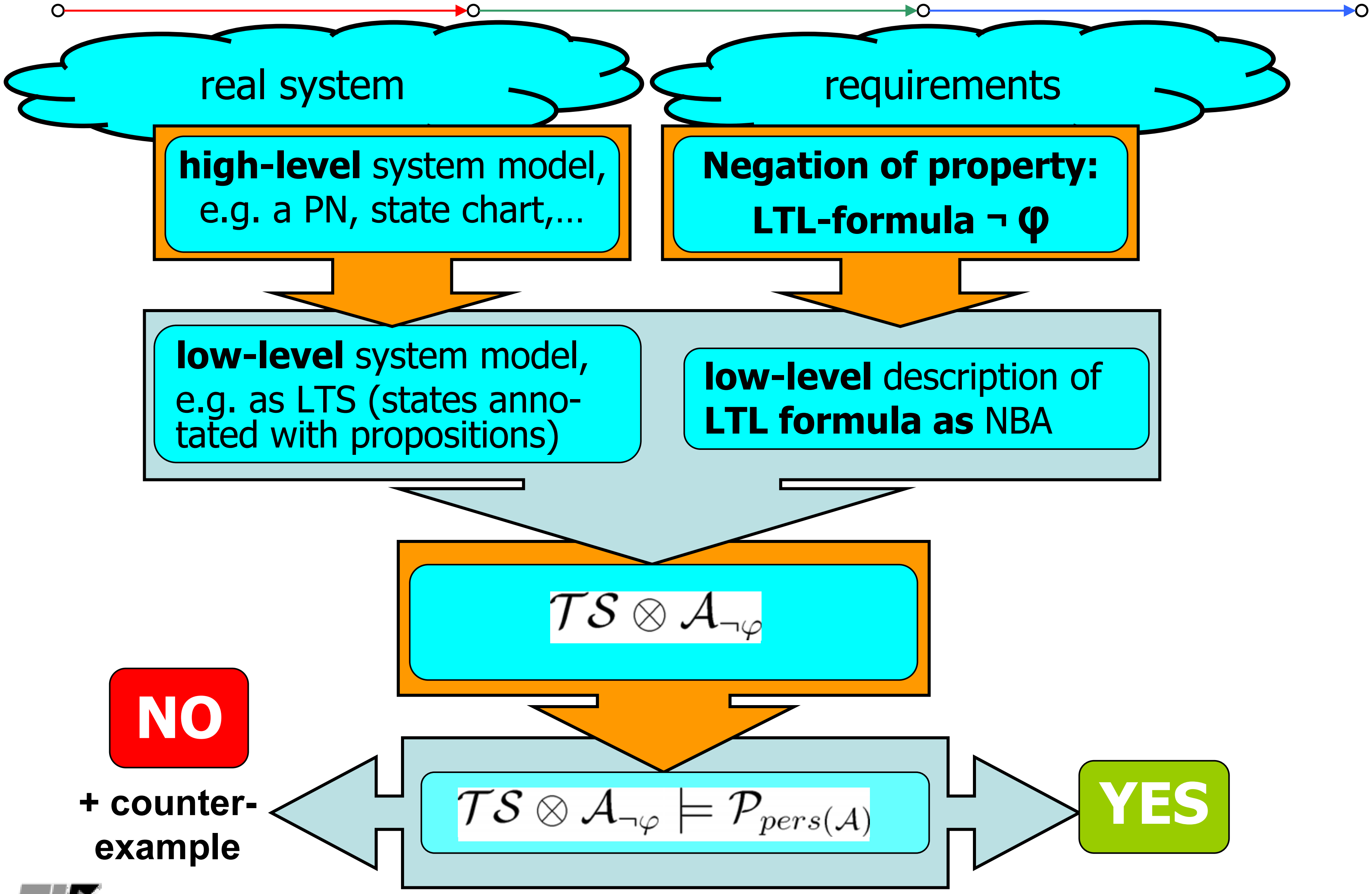
$$\begin{aligned}
\bigcirc(\varphi \bigcup \psi) &= (\bigcirc \varphi) \bigcup (\bigcirc \psi) \\
\Diamond(\varphi \vee \psi) &= \Diamond \vee \bigcup \Diamond \psi \\
\Box(\varphi \wedge \psi) &= \Box \varphi \wedge \Diamond \psi
\end{aligned}$$

- Expansion law

$$\begin{aligned}
\varphi \bigcup \psi &= \psi \vee (\varphi \vee \bigcirc(\varphi \bigcup \varphi)) \\
\Diamond \varphi &= \varphi \vee \bigcirc \Diamond \varphi \\
\Box \varphi &= \varphi \wedge \bigcirc \Box \varphi
\end{aligned}$$

- ....

Model Checking: Automatic verification of system behaviours


real system
requirements
high-level system model, e.g. a PN, state chart,…
Negation of property: LTL-formula ¬ φ
low-level system model, e.g. as LTS (states annotated with propositions)
low-level description of **LTL formula as** NBA
$\mathcal{TS} \otimes \mathcal{A}_{\neg\varphi}$
$\mathcal{TS} \otimes \mathcal{A}_{\neg\varphi} \models \mathcal{P}_{pers(\mathcal{A})}$
NO
+ counter-example
YES

Symbolic model checking (at glance)

- **State-of-the-art**
  - Transition relation (TR) can be represented by BDDs, requiring complex procedures for deriving BDD from high-level model descriptions.
  - NBA for representing negated property to be verified are small, can thus easily be transformed into a BDD-based representation
  - BDD-manipulating algorithms allow the execution of MC

  Depending on the modelled system BDDs may explode in the number of allocated nodes (add-function y := x + p)

- **New trends: SAT-Solvers have shown to be of value in such cases**

Beyond BDDs: SAT-Solvers (at glance)

- Satisfiability: Does there exists an assignment to the variables of a formula α of propositional logics, so that the formula evaluates to true
- 3-SAT: In general this problem is NP-complete (Cook 1972)

  => One may not expect always efficient computations. But in practice SAT-solver have shown to be very powerful, outperform BDDs

- Employing SAT-based MC:
  - Encode TR as boolean formula (unfolding of loops, each step in TR is encoded by a new set of variables, k-steps within TR

$$I(s_0) \wedge \bigwedge_{i:=0}^{k-1} T(s_i, s_{i-1})$$

  - Encode properties to be checked as boolean equation
  - Check if the obtained overall formula is satisfiable

Beyond verification of functional properties

- Probabilistic/Stochastic MC: System descpritions and formulae to be verified are extended with propabilities or intervals thereof, e.g. pCTL:

$\varphi := true \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \mathbb{P}_J(\phi)$ where $J \subseteq [0,1]$ is an interval on the rationals and ϕ is a path formula defined by
$\phi := \bigcirc\varphi \mid \varphi_1 \bigcup \varphi_2 \mid \varphi_1 \bigcup^{\leq n} \varphi_2$ with $n \in \mathbb{N}$ as step bound

- Timed extensions have been developed: allowing the correctness of properties not only functional and quantiatively, but real-time-wise, e.g. Timed CTL model checking for timed systems, i.e. for systems specified by Timed Automata.

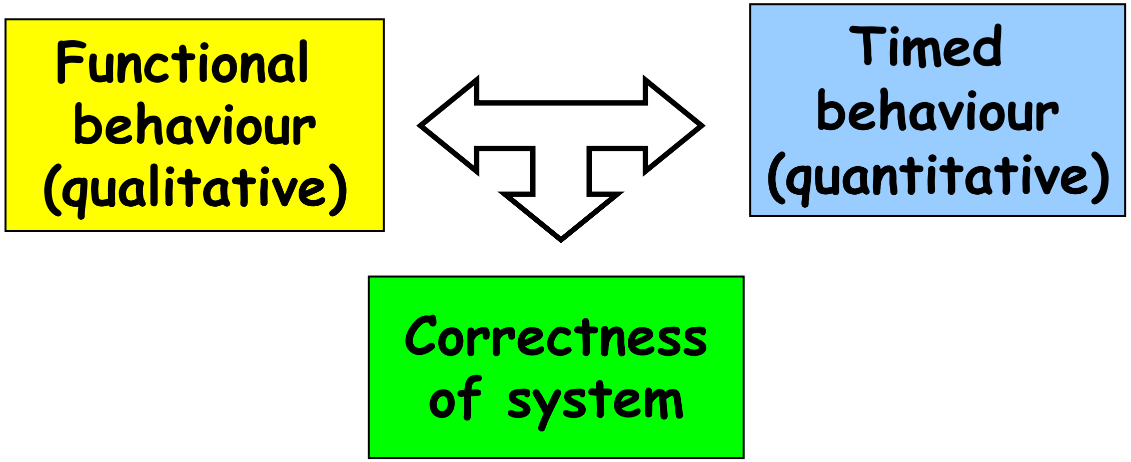
$\varphi := true \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \exists\phi \mid \forall\phi$
where ϕ is a path formula defined by $\phi := \varphi \bigcup^{J} \varphi$,
J is an interval, constraining clocks

Where are we?

- **SDL and MSC** ✓
- **Petri Nets** ✓
  - Notation
  - Behavioral Properties
- **Symbolic Analysis methods of finite models** ✓
- **Introduction to model checking** ✓
- **Timed automata (real-time)**
  - Notation
  - Semantics
  - Analysis

Introduction

Correctness in time-critical systems depends on the correctness of the produced output as well as on its availability just in time.


Functional behaviour (qualitative)
Timed behaviour (quantitative)
Correctness of system

- **SDL:** Time via timer signals or continuous signal. Re-action depends on position of timer signal in input-queue and/or on scheduling of processes ⇒ **uncoordinated delay !**

- **PN:** time does not exists => standard extension Timed PN (~TA)

Timed Automata (Alur, Dill '91)

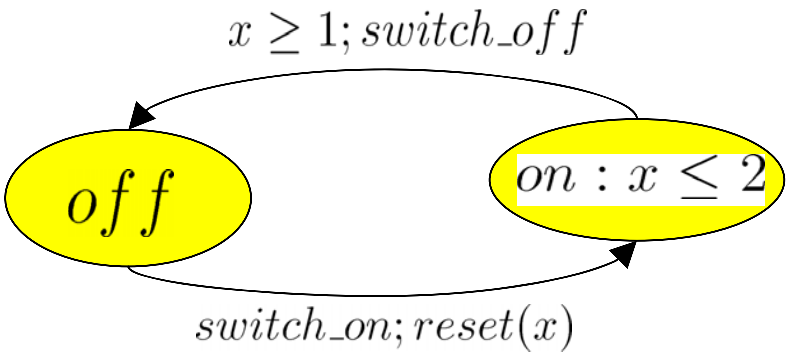
Informally speaking:

- TA is a program graph the variables of which are **clocks.**
- The value of clocks serve as conditions for taking an edge, emanating from the current state (location).
- TA can be used to **reason about timed, functional or combined properties of systems**
- Enables description of **very complex systems,** e.g. scheduling with pre-emption, etc.
- **Compositional by construction**: i.e. build system as network of TAs,, communication via synchronization (= joint execution of dedicated (enabled) transitions.).

Example: A simple switch

In a nutshell:

- A timed automaton $\mathcal{TA}$ is a directed action-labeled graph, which is extended with non-negative clocks. Conditions imposed on the clocks (time constraints) steers the execution of the TA.
- Time constraints may be associated with edges (guards) or with locations (invariants)



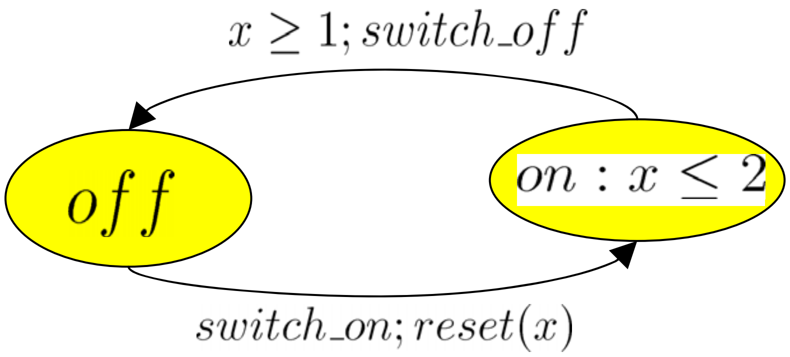
Remark:
only invariants enforce execution of transitions

Timed Automata (TA)

- is a tuple $(\mathcal{L}oc, l_0, \mathcal{C}, \mathcal{A}ct, \longrightarrow, \mathcal{I}nv)$ where
  1. $\mathcal{L}oc$ is a set of locations, with l_0 as initial one.
  2. $\mathcal{C}$ is a set of clocks, with $\mathcal{CC}$ as set of clock constraints $g := \{c < x; c \leq x; c > x; c \geq x; g \wedge g\}$.
  3. $\mathcal{A}ct$ as set of action labels.
  4. $\longrightarrow \subseteq \mathcal{L}oc \times \mathcal{A}ct \times \mathcal{CC}(C) \times 2^{|C|} \times \mathcal{L}oc$
  5. $\mathcal{I}nv : \mathcal{L}oc \rightarrow \mathcal{CC}(\mathcal{C})$ for equipping location with (time) invariants
- A clock valuation is a function $\eta : \mathcal{C} \rightarrow \mathbb{R}_{\geq 0}$ assigning a value to each $x \in \mathcal{C}$, where $\forall x \in \mathcal{C} : \eta_0(x) = 0$ holds.

Example

- Each edge of a TA is equipped with an action-label $a \in \mathcal{A}ct$
- a clock condition or guard $g \in \mathcal{CC}$
- a set of clocks $c_i \in \mathcal{C}$ to be reset, once the edge is taken.
- With the locations $l_1, l_2 \in \mathcal{LOC}$ this yields the notation



$$l_2 \xrightarrow{a,g,C} l_1$$

What is the guard, the label, set of clocks?

Semantics of Timed Automata (1)

There are two ways for a TA to proceed:

1. Take the enabled transitions emanating from the current location (=> discrete transitions)
2. Let time progress (=> delay transition)

=> This yields an (infinite) labelled transition system (LTS)

Let $\mathcal{T}A = (\mathcal{L}oc, l_0, \mathcal{C}, \mathcal{A}ct, \longrightarrow, \mathcal{I}nv)$ be a TA, it defines an (infinite) LTS $\mathcal{T}S(\mathcal{T}A) = (\mathcal{S}, \mathcal{A}ct', \longrightarrow, \mathcal{I})$, where

- $\mathcal{S} \subseteq \mathcal{L}oc \times Eval(\mathcal{C})$.
- $\mathcal{A}ct' = \mathcal{A}ct \cup \mathbb{R}_{\geq 0}$.
- $\mathcal{I} = \{< l_0, \eta > | l_o \in \mathcal{L}oc \wedge \eta(x) = 0 \; \forall x \in \mathcal{C}\}$ and
- $\longrightarrow$ as transition relation.

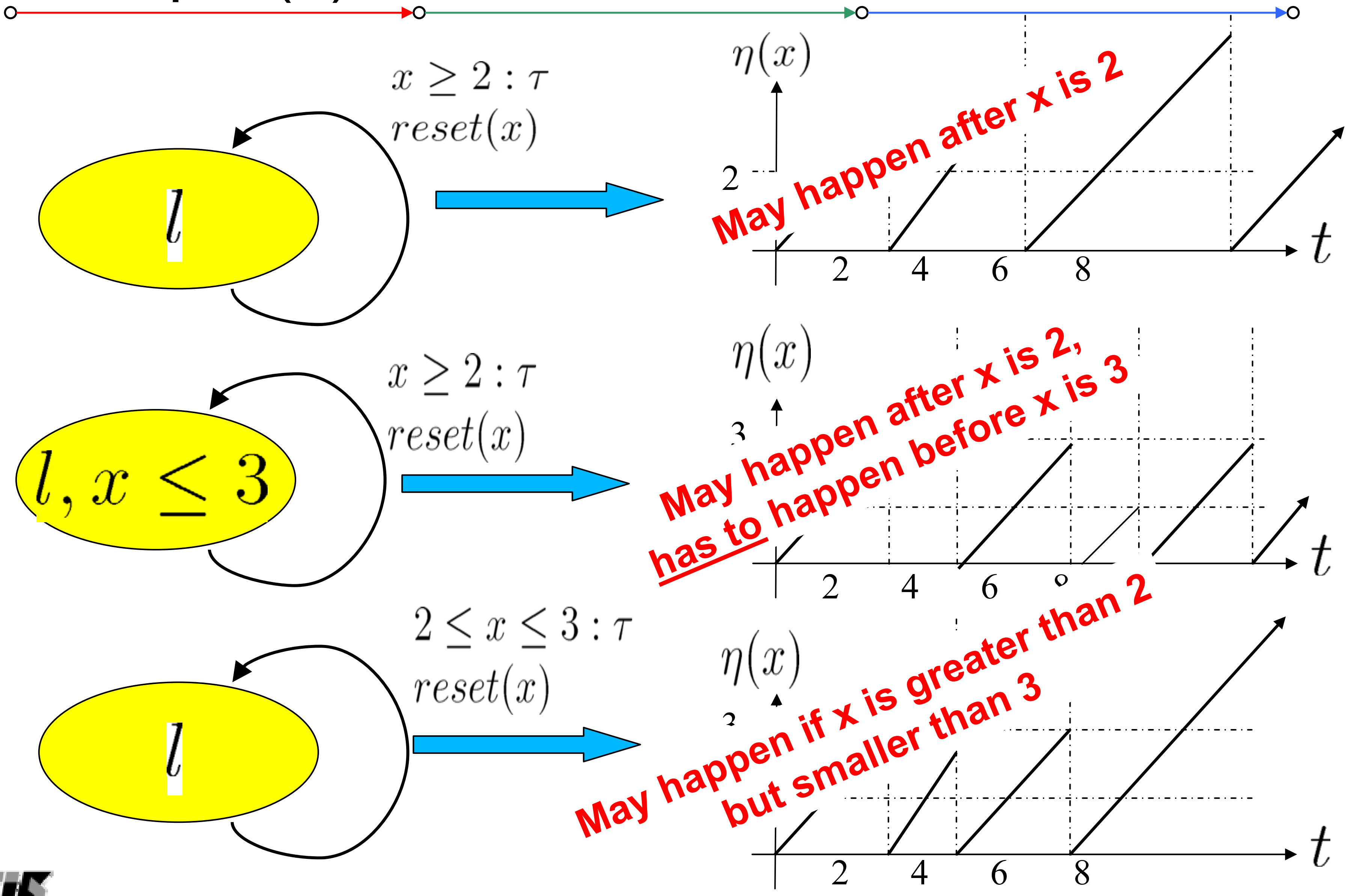
Multiple actions in zero time

- Transitions within TA (and their LTS) are instantaneous, i.e. they happen in zero time units. The elapse of time takes only place in locations (states)

  ⇒ At a single instant of time, several edges can be taken, but each transition of the LTS refers to the execution of a single action.

Example ⟶

Example (2)



May happen after x is 2

May happen after x is 2, <u>has to</u> happen before x is 3

May happen if x is greater than 2 but smaller than 3

Semantics of Timed Automata (2)

The transitions are defined as follows:

1. *discrete transitions*: $< l, \eta > \xrightarrow{\alpha} < l', \eta' >$ if the following conditions hold:

   (a) there is a transition in $\mathcal{TA} : l \overset{g:\alpha,\mathfrak{D}}{\longrightarrow} l'$

   (b) $\eta \models g$

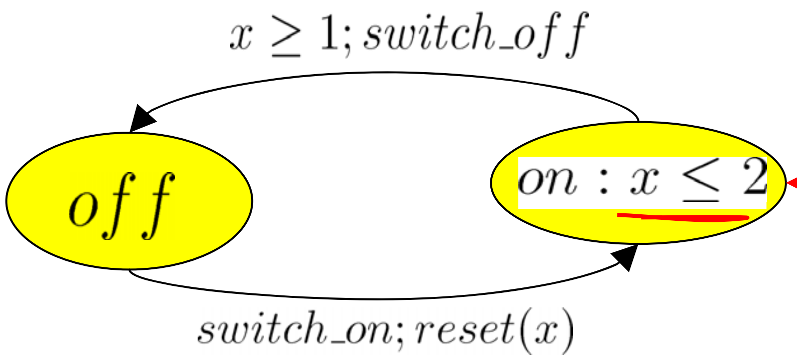
   (c) $\eta' = reset\ \mathfrak{D}\ in\ \eta$

   (d) $\eta' \models \mathcal{I}nv(l')$

2. *delay transition*:
   $< l, \eta > \xrightarrow{d} < l, \eta + d >$ for $d \in \mathbb{R}_{\geq 0}$ if $\eta + d \models \mathcal{I}nv(l)$
   (=> one can stay in l as long as $\mathcal{I}nv(l)$ is not violated).

=> Time is only spent in locations, transitions are time-less!

Example



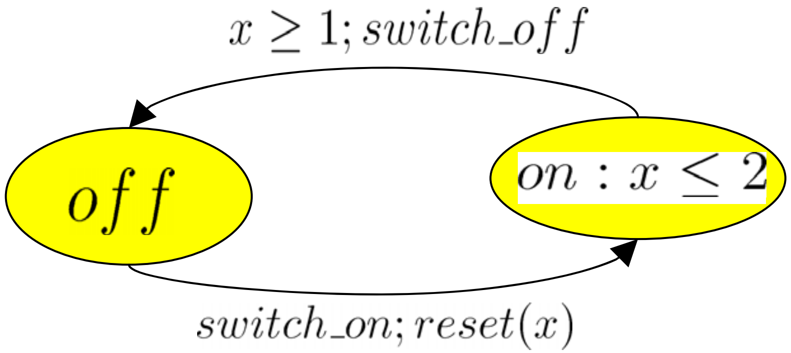
- The state space of *TS(Switch)*:

$$S := \{< off, t > |t \in \mathbb{R}_{\geq 0}\} \cup \{< on, t > |t \in \mathbb{R}_{\geq 0}\}$$

- The set of reachable states, starting from $< off, 0 >$ is defined as follows:

$$S := \{< off, t > |t \in \mathbb{R}_{\geq 0}\} \cup \{< on, t > |0 \leq t \leq 2\}$$

Example



TS(Switch):

$$\begin{array}{llll}
<off,t> & \xrightarrow{d} & <on,t+d> & \forall t \geq 0 \wedge d \geq 0 \\
<off,t> & \xrightarrow{switch\_on} & <on,0> & \forall t \geq 0 \\
<on,t> & \xrightarrow{d} & <on,t+d> & \forall t \geq 0 \wedge d \geq 0 \\
<on,t> & \xrightarrow{switch\_off} & <off,t> & 1 \leq t \leq 2
\end{array}$$

LTS: Finite semantics models for TA

The LTS underlying a TA gives an infinite model of the system

=> To keep things analyzable, one considers therefore a finite quotient of LTS, denoted as region graph

Idea: Merge states satisfying

- the same clock constraints and
- from which "similar" time-divergent paths (= paths on which time always progress) emanate

Requires definition of an equivalence class →

Integral - and fractional parts of clocks

1. Integral part:

$$\lfloor d \rfloor = max\{c \in \mathbb{N} | c \leq d\}$$

2. Fractional part:

$$frac(d) = d - \lfloor d \rfloor$$

Idea:

1. Agree on integral part $\lfloor d \rfloor$ $\Rightarrow$ clks $\models$ same constraints

2. frac. part gives order of clks changing next

The region graph (1)

1) Clock valuations η, η' are equivalent (η ≈ η') iff

- for any $x \in \mathcal{C}$: $\eta(x) > c_x$ and $\eta'(x) > c_x$ holds, or
- for any $x, y \in \mathcal{C}$ with $\eta(x), \eta'(x) \leq c_x$ and $\eta(y), \eta'(y) \leq c_y$ it holds:
  - $\lfloor \eta(x) \rfloor = \lfloor \eta'(x) \rfloor$ and $frac(\eta(x)) = 0 \iff frac(\eta'(x)) = 0$
  - $frac(\eta(x)) \leq frac(\eta(y)) \iff frac(\eta'(x)) \leq frac(\eta'(y))$

2) A clock region [η] is defined by

$$[\eta] = \{\eta' \in Eval(\mathcal{C}) | \eta \approx \eta'\}$$

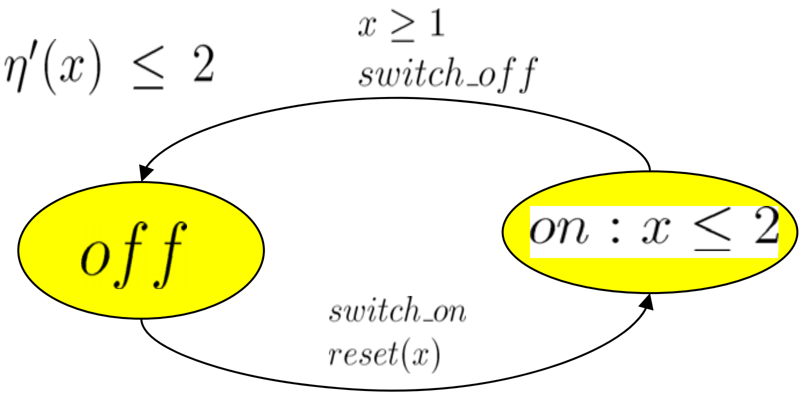
3) The state region [s] for $s =< l, \eta > \in TS(TA)$ is given by

$$[s] =< l, [\eta] >= \{< l, \eta' > | \eta' \in [\eta]\}$$

Example

1. The requirement $\eta(x), \eta'(x) > 2$ or $\eta(x), \eta'(x) \leq 2$ yields the regions:

$$x \in [0,2] \text{ or } x \in (2, \infty)$$



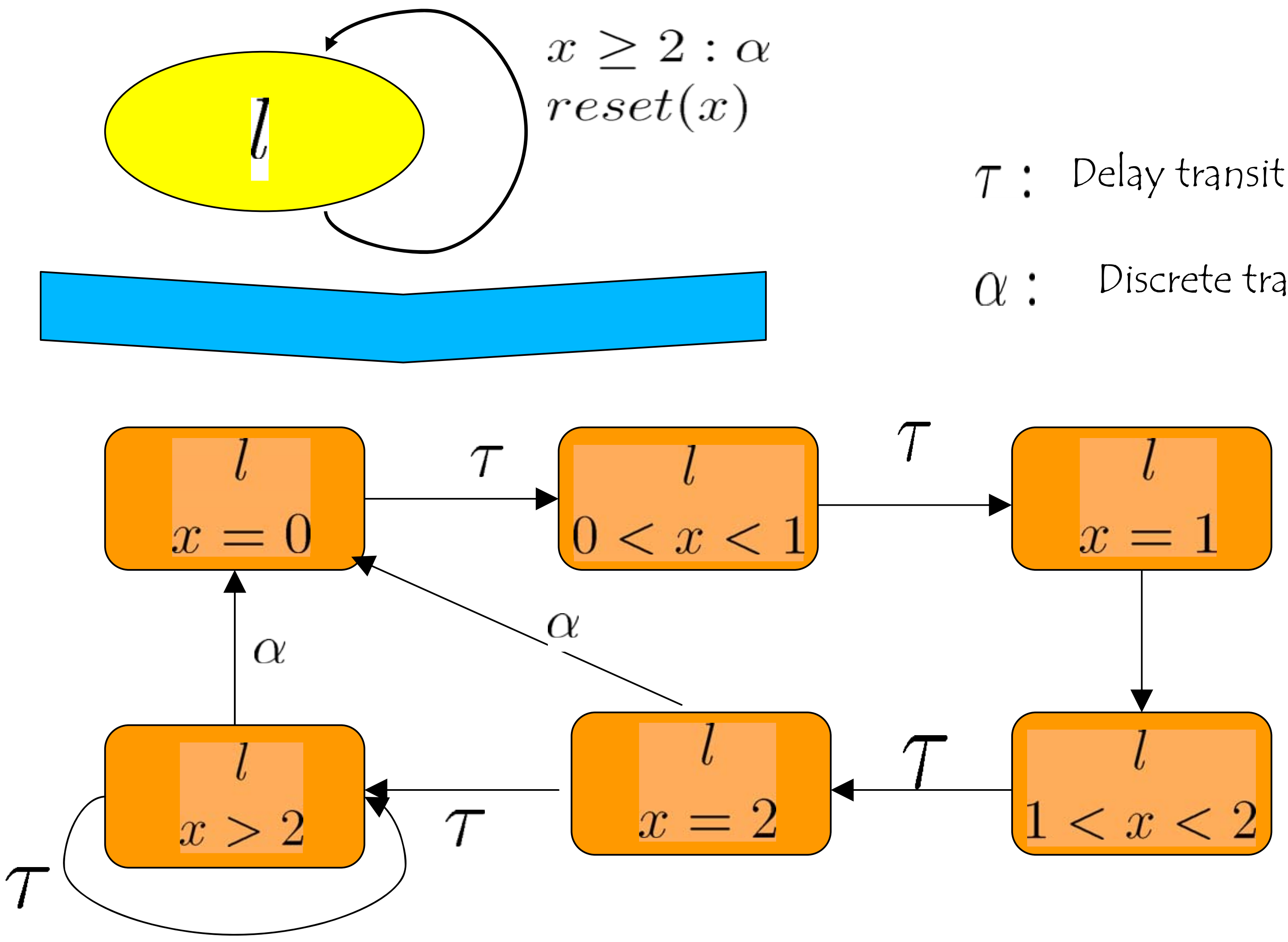
2. For $\eta(x), \eta'(x) \leq 2$ we have

   - that the integral parts of the valuations must agree and that $frac(\eta(x)) = 0 <=> frac(\eta'(x))$. This gives:

     $$[0,0]; (0,1); [1,1]; (1,2); [2,2]; (2, \infty)$$

   - or we have $frac(\eta(x)) \leq frac(\eta(y))$
     $$\iff frac(\eta'(x)) \leq frac(\eta'(y)),$$ which trivially holds, since $|\mathcal{C}| = 1$.

Region transition system



The region graph (2)

Upper and lower bound on the number of regions

$$|\mathcal{C}|! \times \prod_{x \in \mathcal{C}} c_x \leq \frac{|Eval(\mathcal{C})|}{|\approx|} \leq |\mathcal{C}|! \times 2^{|\mathcal{C}|-1} \times \prod_{x \in \mathcal{C}} (2c_x + 2)$$