Model Checking

Linear versus Branching Time

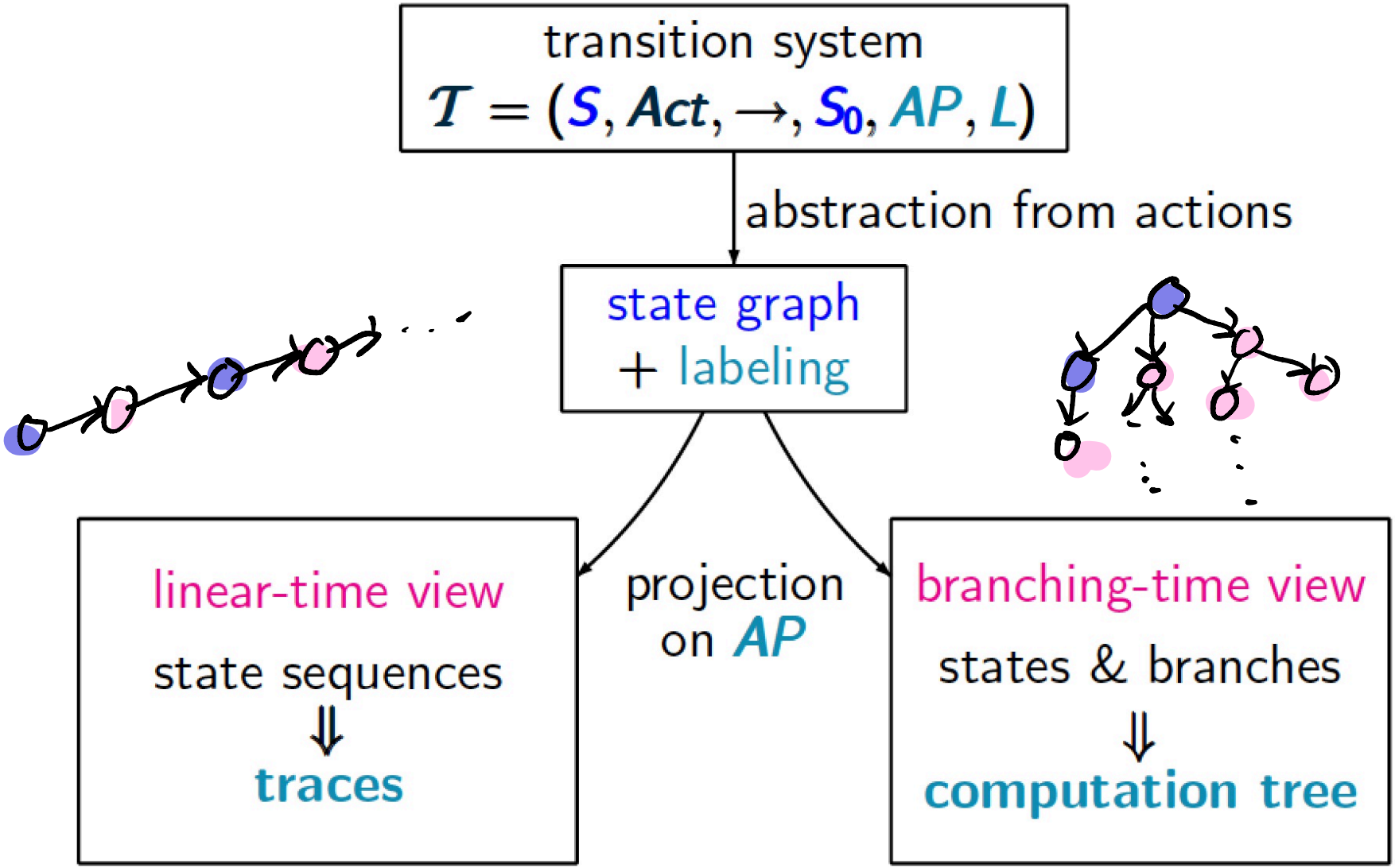
[Baier & Katoen, Chapter 6.3, 7.1, 7.2, 7.3.5]

Joost-Pieter Katoen and Tim Quatmann

Software Modeling and Verification Group

RWTH Aachen, SoSe 2022

Topic


transition system
$T = (S, Act, \rightarrow, S_0, AP, L)$
abstraction from actions
state graph
+ labeling
linear-time view
state sequences
traces
projection
on AP
branching-time view
states & branches
computation tree

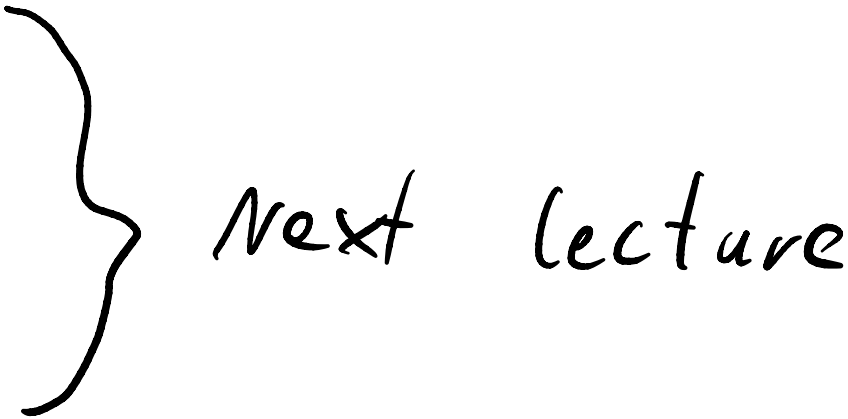
Linear Versus Branching Time

	linear time	branching time
behavior	path based traces	state based computation tree
temporal logic	**LTL** path formulas	**CTL** state formulas
model checking	PSPACE-complete $\mathcal{O}(size(\mathcal{T}) \cdot \exp(\lvert\varphi\rvert))$	PTIME - complete $\mathcal{O}(size(\mathcal{T}) \cdot \lvert\Phi\rvert)$
impl. relation	trace inclusion trace equivalence PSPACE-complete	simulation bisimulation PTIME

TS_1, TS_2 s.t. $tr(TS_1) \subseteq tr(TS_2) \Rightarrow \forall \varphi \in LTC : TS_2 \models \varphi \Rightarrow TS \models \varphi$

Overview

1. Reminder: Expressiveness of LTL, CTL, and CTL*
2. Complexity Considerations
3. Trace and Bisimulation Equivalence
4. Bisimilarity And CTL
5. CTL* Model Checking
6. Summary



Overview

LTL and CTL are Incomparable

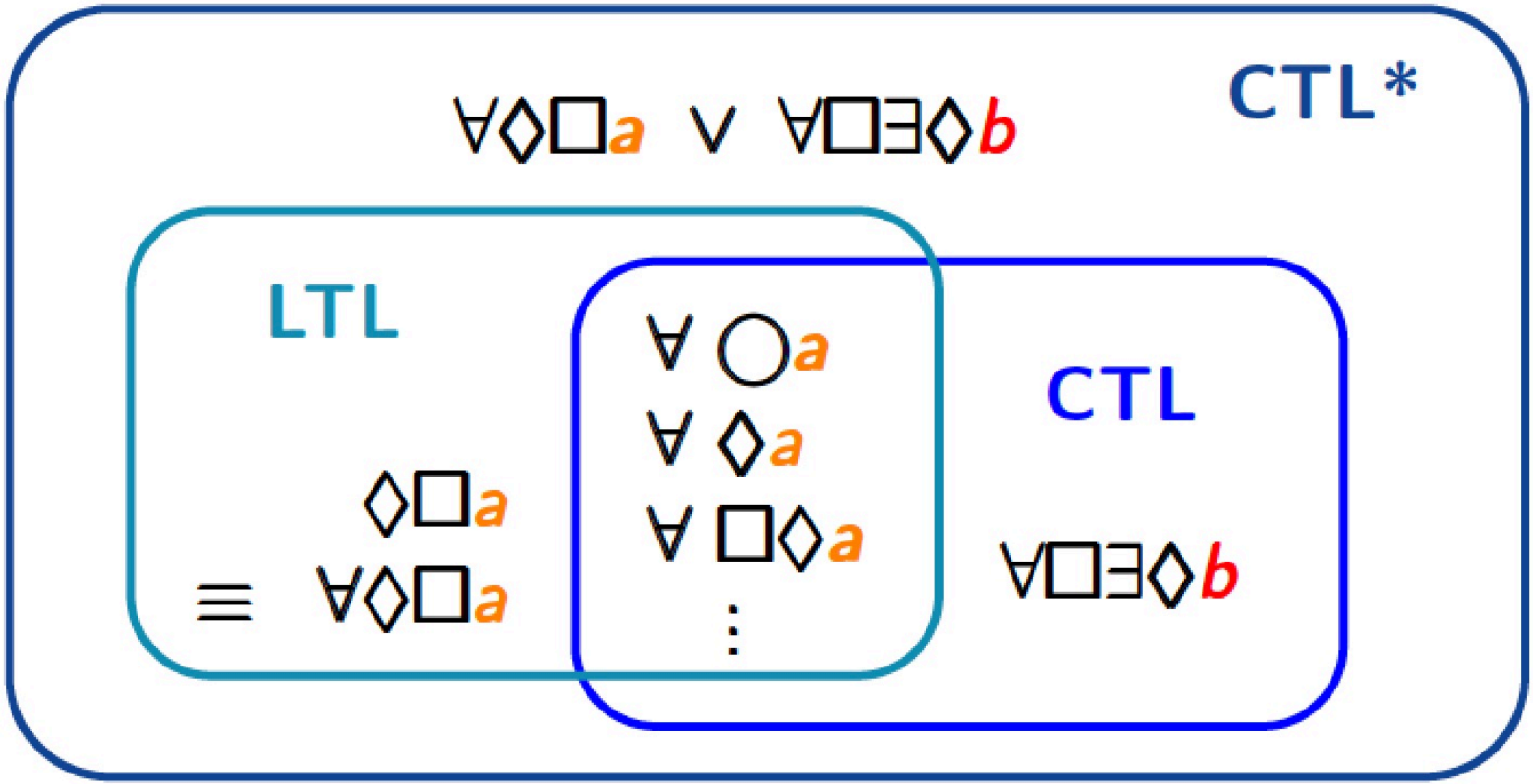
- Some LTL-formulas cannot be expressed in CTL, e.g.,
  - $\Diamond\Box\, a$
  - $\Diamond\,(a \wedge \bigcirc\, a)$

  There does not exist an equivalent CTL formula

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

  There does not exist an equivalent LTL formula

Relating LTL, CTL, and CTL*



Overview

CTL vs. LTL Model Checking

LTL model checking is PSPACE-complete
CTL model checking is PTIME-complete.

$\varphi \equiv \Phi$

- Take a property that can be expressed in both LTL and CTL
- Is CTL model checking more efficient?

Should we check $TS \models \varphi$ or $TS \models \Phi$?

- **No!** LTL-formulae can be exponentially shorter than their CTL-equivalent

CTL Versus LTL

If Φ is equivalent to some LTL-formula then:

$\forall \quad \exists$

$\Phi \equiv \varphi$ where φ is obtained by removing all path quantifiers from Φ.
In particular, $|\varphi| \leq |\Phi| \leq 2 \cdot |\varphi|$.

But: φ is not necessarily the smallest equivalent LTL formula.

If P $\neq$ NP, then there is a sequence φ_n, $n \geq 0$ of LTL formulas such that:

- $|\varphi_n|$ is polynomial in n
- φ_n has an equivalent CTL formula Φ_n
- no CTL formula of polynomial length is equivalent to φ_n

Proof.

φ_n = the absence of a Hamiltonian path in a digraph on n vertices □

Reminder: The Hamilton Path Problem

A Hamilton path of a (directed and finite) graph $G = (V, E)$ is a path $\pi = v_1 \dots v_n$ that visits each vertex of G exactly once, i.e.,

$$\forall\, v \in V \,.\, \exists\, i \leq n \,.\, \big(v_i = v \text{ and } \forall\, j \neq i \,.\, v_j \neq v\big).$$

The Hamiltopn Path Problem (HAM)

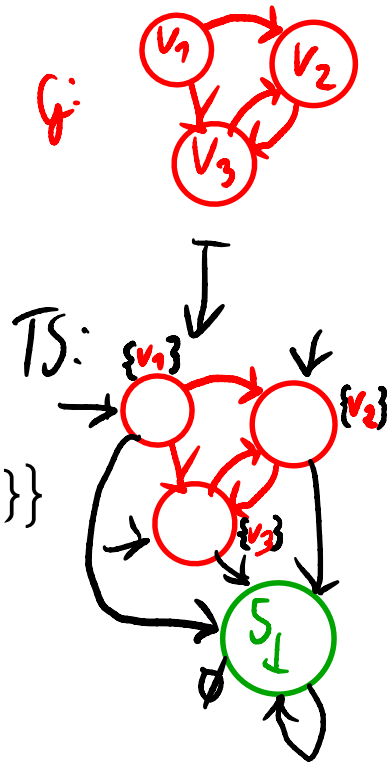
Given a graph G, does G have a Hamilton path?

HAM is NP-complete.

Reminder: Polynomial Reduction for $\texttt{HAM} \leq_{\text{poly}} \overline{\texttt{LTL}}$

Transform input $G = (V, E)$ for HAM to input (TS, φ) for LTL such that

$$G \text{ has a Hamilton path} \quad \text{iff} \quad TS \not\models \varphi$$



- $TS := (V \uplus \{s_\perp\},\ \{\alpha\},\ \rightarrow,\ V,\ V,\ L)$, where
  - $L(s_\perp) := \varnothing, \qquad \forall v \in V.\ L(v) := v$, and
  - $\rightarrow := \{(v, \alpha, v') \mid (v, v') \in E\} \cup \{(s, \alpha, \{s_\perp\}) \mid s \in V \uplus \{s_\perp\}\}$
- $\varphi = \neg\Big(\bigwedge_{v \in V} \Diamond v \wedge \Box\big(v \Rightarrow \bigcirc\, \Box \neg v\big)\Big)$

$|TS| + |\varphi|$ is linear in the size of the graph $|G|$

If P ≠ NP, then there is a sequence φ_n, $n \geq 0$ of LTL formulas such that:

1. $|\varphi_n|$ is polynomial in n
2. φ_n has an equivalent CTL formula Φ_n
3. no CTL formula of polynomial length is equivalent to φ_n

Proof:

For $n \in \mathbb{N}$ Let $V = \{v_1, \dots, v_n\}$ and

$$\varphi_n := \bigwedge_{i=1}^{n} \Big(\Diamond v_i \wedge \Box\big(v_i \Rightarrow \bigcirc \Box \neg v_i\big)\Big)$$

expresses that a graph on n vertices has a Hamilton path

① $|\varphi_n| \in O(\text{poly}(n))$

② $\text{words}(\varphi_n) = \{ \{v_{i_1}\}\{v_{i_2}\} \dots \{v_{i_n}\} \emptyset^\omega \mid (i_1, \dots, i_n)$ is a permutation of $[1 \dots n]\}$

In CTL, we can enumerate all perm. of $[1 \dots n]$

For example, if $n = 2$:

$$\Phi_2 = \big(v_1 \wedge \exists\bigcirc(v_2 \wedge \exists\bigcirc \Phi_\emptyset)\big) \quad (1,2)$$
$$\vee \big(v_2 \wedge \exists\bigcirc(v_1 \wedge \exists\bigcirc \Phi_\emptyset)\big) \quad (2,1)$$

where $\Phi_\emptyset = \bigwedge_{i=1}^{n} \neg v_i$

$$\Phi_3 = v_1 \wedge \exists 0 (v_2 \wedge \exists 0 (v_3 \wedge \exists 0\, \Phi_\emptyset)) \quad (1,2,3)$$

$$\vee\ v_1 \wedge \exists 0 (v_3 \wedge \exists 0 (v_2 \wedge \exists 0\, \Phi_\emptyset)) \quad (1,3,2)$$

$$\vee\ v_2 \wedge \exists 0 (v_1 \wedge \exists 0 (v_3 \wedge \exists 0\, \Phi_\emptyset)) \quad (2,1,3)$$

$$\vee \quad \cdots \quad (2,3,1)$$

$$\vee \quad \cdots \quad (3,1,2)$$

$$\vee \quad \cdots \quad (3,2,1)$$

In general:

$$\Phi_n = \bigvee \{ \psi_{(i_1 \ldots i_n)} \mid (i_1, \ldots, i_n) \text{ is a perm. of } [1..n] \}$$

$$\psi_{(i_1 \ldots i_n)} := v_{i_1} \wedge \exists 0 (\psi_{(i_2 \ldots i_n)})$$

$$\psi_{(i_1)} := v_{i_1} \wedge \exists 0 (\Phi_\emptyset)$$

$$Sat(\psi_{(i_1 \ldots i_n)}) = \begin{cases} \{v_{i_1}\} & \text{if } v_{i_1} v_{i_2} \ldots v_{i_n} \text{ is a Ham. path} \\ \emptyset & \text{else} \end{cases}$$

To show $\neg\varphi_n \equiv \neg\phi_n$

This is as follows:

$TS \not\models \neg\phi_n$

iff $\exists$ init. state s_0 s.t. $s_0 \not\models \neg\phi_n$

iff ——— " ——— $s_0 \models \phi_n$

iff $\exists$ perm. $(i_n \ldots i_n)$ of $\{1 \ldots n\}$

s.t. $\{v_{i_1}\} \ldots \{v_{i_n}\} \emptyset^\omega \in Traces(TS)$

iff $TS \not\models \neg\varphi_n$

③ Assume there is an equivalent formula Φ_n' such that $|\Phi_n'| \in O(poly(n))$

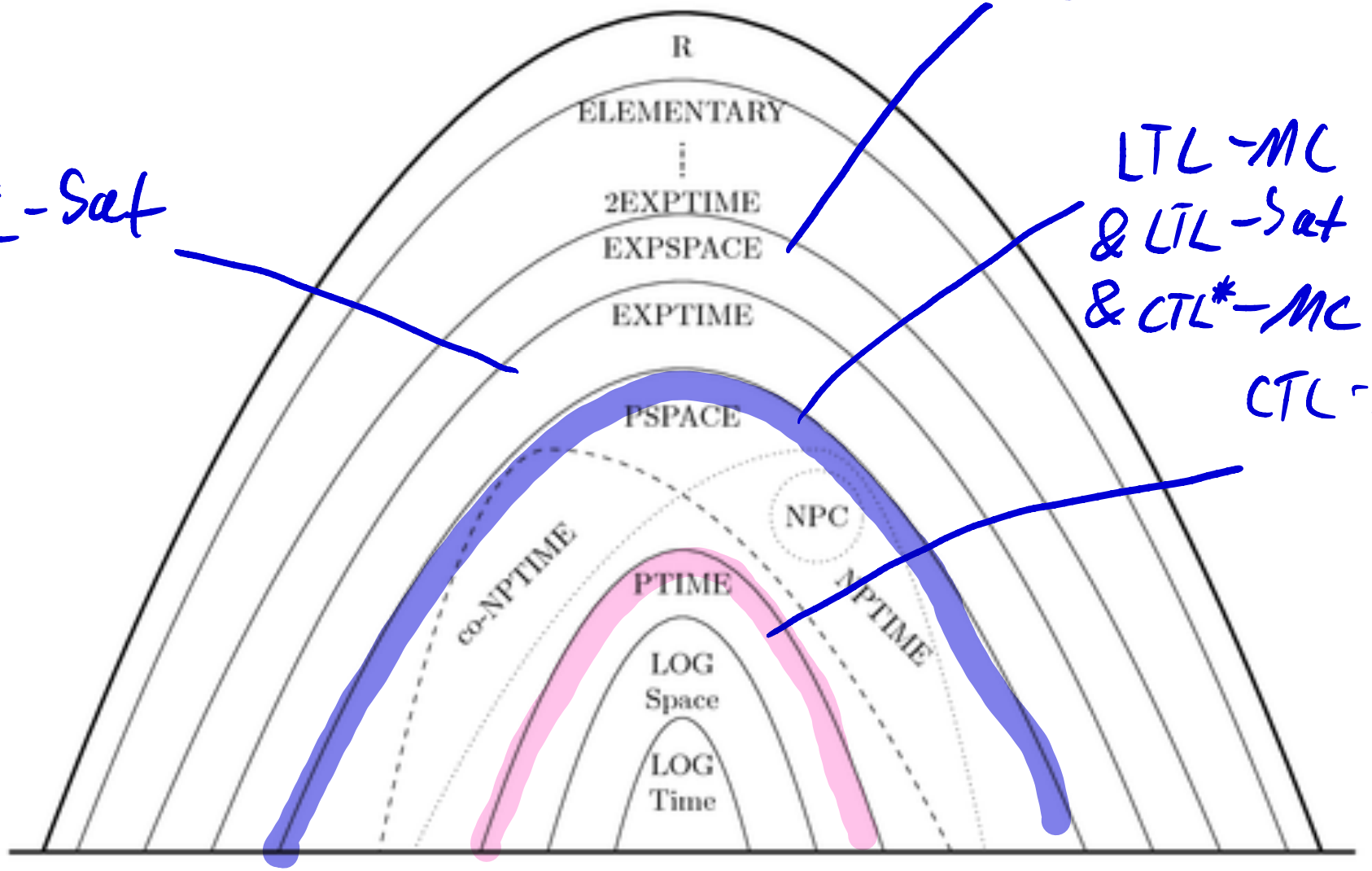
Then, since CTL-Model Checking is PTIME complete: $TS \models \Phi_n'$ can be checked in Poly. Time

$\rightsquigarrow$ HAM $\in$ PTIME

└ NP-complete

↳ This contradicts the assumption $P \neq NP$

Complexity Class Refresher



CTL*-Sat

LTL-MC & LTL-Sat & CTL*-MC

CTL-Sat

CTL-MC

MC = Model Checking

Satisfiability Problem

The LTL satisfiability problem is PSPACE-complete.

Given φ

Is there a TS such that $TS \models \varphi$?

Satisfiability Problem

The LTL satisfiability problem is PSPACE-complete.

The LTL satisfiability problem is equally hard as
the LTL model checking problem.

Satisfiability Problem

The LTL satisfiability problem is PSPACE-complete.

The LTL satisfiability problem is equally hard as
the LTL model checking problem.

- The CTL satisfiability problem is EXPTIME-complete.
- The CTL* satisfiability problem is 2EXPTIME-complete.

Satisfiability Problem

The LTL satisfiability problem is PSPACE-complete.

The LTL satisfiability problem is equally hard as
the LTL model checking problem.

- The CTL satisfiability problem is EXPTIME-complete.
- The CTL* satisfiability problem is 2EXPTIME-complete.

The CTL satisfiability problem is harder than
the CTL model checking problem.
This also applies to CTL* (and many more logics)

Overview

Trace Equivalence

cf. Lec 3

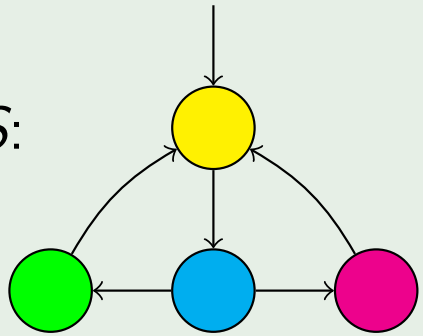
Definition: trace equivalence

Transition systems TS and TS' (both over AP) are trace equivalent iff they exhibit the same traces:

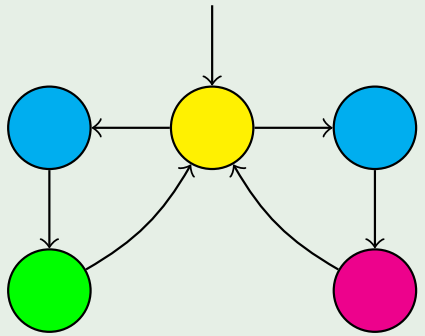
$$TS \equiv_{trace} TS' \quad \text{if and only if} \quad Traces(TS) = Traces(TS').$$

Example

TS:



TS':



- $Traces(TS) = Traces(TS') \quad \rightsquigarrow \quad TS \equiv_{trace} TS'$

Trace Equivalence and LT Properties

$TS \equiv_{trace} TS'$ if and only if TS and TS' satisfy the same LT properties:

$$TS \equiv_{trace} TS' \quad \text{if and only if} \quad \Big(\forall E \subseteq \big(2^{AP}\big)^{\omega}.\ TS \models E \text{ iff } TS' \models E \Big).$$

Logical Equivalence

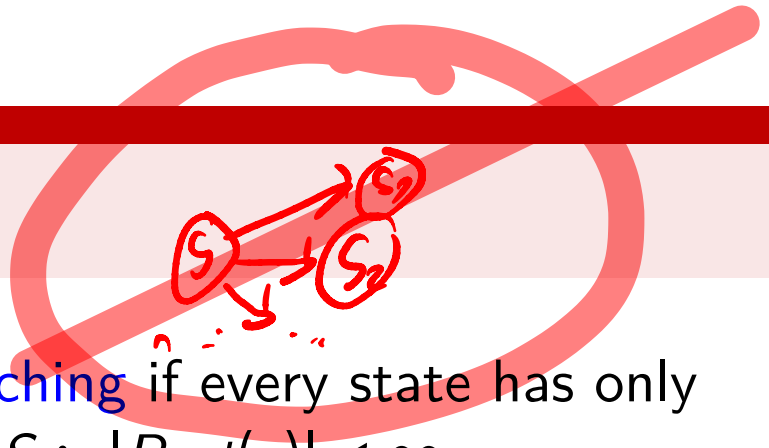
Definition: logical equivalence

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

- $TS \equiv_{LTL} TS'$ iff $\big(\forall \varphi \in \text{LTL}.\ TS \models \varphi \text{ iff } TS' \models \varphi\big)$
- $TS \equiv_{CTL} TS'$ iff $\big(\forall \Phi \in \text{CTL}.\ TS \models \Phi \text{ iff } TS' \models \Phi\big)$

In a similar way, $\equiv_L$ can be defined for logic L (such as CTL*, CTL$^+$, etc.).

$TS \equiv_{trace} TS'$ implies $TS \equiv_{LTL} TS'$.



A transition system is called finitely-branching if every state has only finitely many direct successors, i.e. $\forall s \in S:\ |Post(s)| < \infty$.

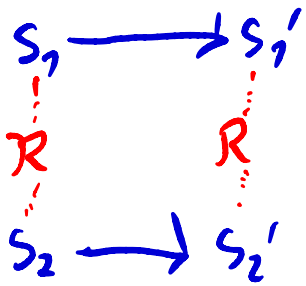
For finitely-branching TS and TS': $TS \equiv_{trace} TS'$ iff $TS \equiv_{LTL} TS'$.

Bisimulation

Definition: bisimulation relation

Let $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP, L_i)$, i=1, 2, be transition systems. The symmetric relation $\mathfrak{R} \subseteq \big((S_1 \cup S_2) \times (S_1 \cup S_2)\big)$ is a bisimulation for (TS_1, TS_2) whenever:

1. for all initial states $s_1 \in I_i$ there exists $s_2 \in I_{2-i}$ such that $(s_1, s_2) \in \mathfrak{R}$
2. for all $(s_1, s_2) \in \mathfrak{R}$ it holds:
   - 2.1 $L_1(s_1) = L_2(s_2)$, and
   - 2.2 $s_1' \in Post(s_1)$ implies $(s_1', s_2') \in \mathfrak{R}$ for some $s_2' \in Post(s_2)$.



$\mathfrak{R}$ is a bisim for (TS_1, TS_2)

$\Leftrightarrow$ $\mathfrak{R}^{-1}$ is a bisim. for (TS_2, TS_1)

$= \mathfrak{R}$ due to symmetry

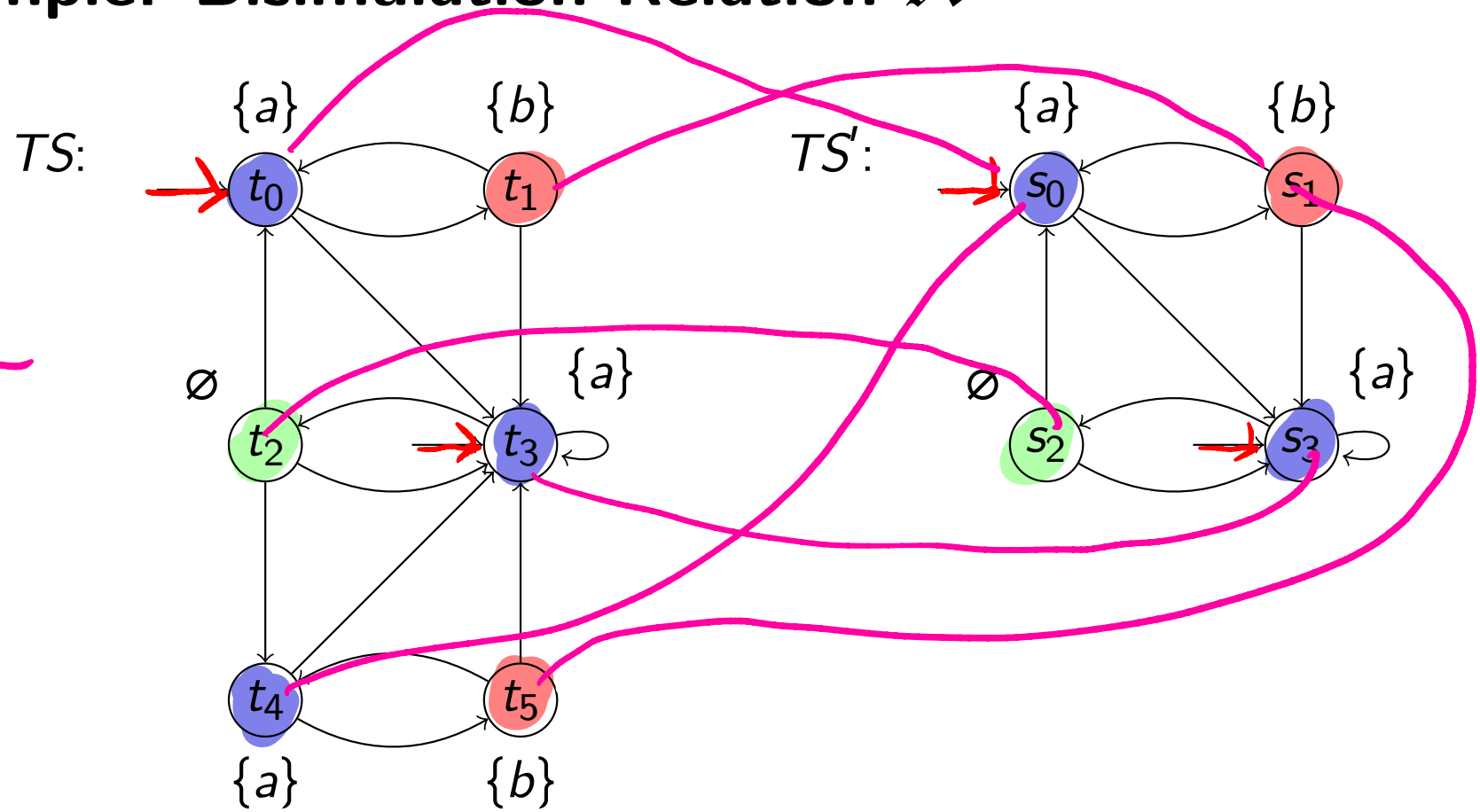
Visually

$$\begin{array}{lll} s_1 & \rightarrow & s_1' \\ \mathfrak{R} & & \\ s_2 & & \end{array} \quad \text{can be completed to} \quad \begin{array}{lll} s_1 & \rightarrow & s_1' \\ \mathfrak{R} & & \mathfrak{R} \\ s_2 & \rightarrow & s_2' \end{array}$$

and by symmetry

$$\begin{array}{lll} s_1 & & \\ \mathfrak{R} & & \\ s_2 & \rightarrow & s_2' \end{array} \quad \text{can be completed to} \quad \begin{array}{lll} s_1 & \rightarrow & s_1' \\ \mathfrak{R} & & \mathfrak{R} \\ s_2 & \rightarrow & s_2' \end{array}$$

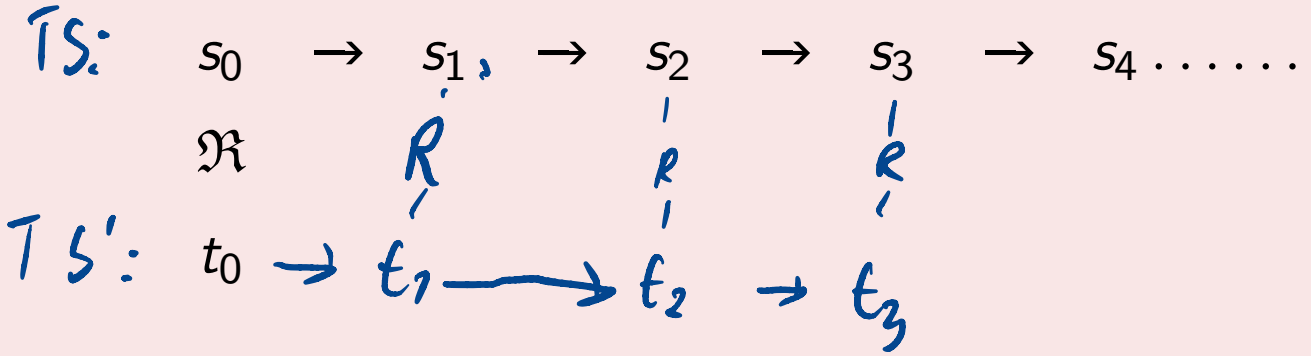
Example: Bisimulation Relation $\mathfrak{R}$



$$\mathfrak{R} = \{(t_0, s_0), (t_1, s_1), (t_2, s_2), (t_3, s_3), (t_4, s_0), (t_5, s_1)\} \cup$$
$$\{(s_0, t_0), (s_1, t_1), (s_2, t_2), (s_3, t_3), (s_0, t_4), (s_1, t_5)\}$$

Bisimulation on Paths

Whenever we have:



this can be completed to

$$\begin{array}{ccccccccccc} s_0 & \rightarrow & s_1 & \rightarrow & s_2 & \rightarrow & s_3 & \rightarrow & s_4 \ldots\ldots \\ \mathfrak{R} & & \mathfrak{R} & & \mathfrak{R} & & \mathfrak{R} & & \mathfrak{R} \\ t_0 & \rightarrow & t_1 & \rightarrow & t_2 & \rightarrow & t_3 & \rightarrow & t_4 \ldots\ldots \end{array}$$

Proof.

By induction on index i of state s_i. □

Bisimulation Equivalence

Definition: bisimulation equivalence

TS_1 and TS_2 are bisimulation equivalent (short: bisimilar), denoted $TS_1 \sim TS_2$, if there exists a bisimulation for (TS_1, TS_2). That is:

$$\sim \;=\; \bigcup \{\, \mathfrak{R} \mid \mathfrak{R} \text{ is a bisimulation on } (TS_1, TS_2) \,\}.$$

Bisimilarity ($\sim$) is an equivalence relation.

Proof.

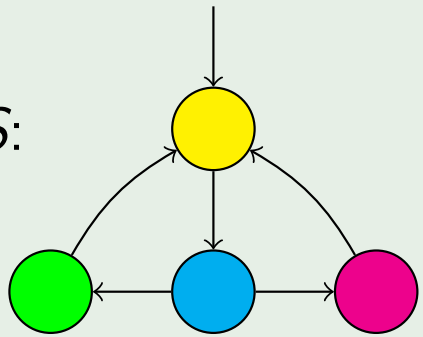
- (Reflexivity). The identity relation is a bisimulation for (TS, TS).
- (Symmetry). If $\mathfrak{R}$ is a bisimulation for (TS, TS'), then $\mathfrak{R}^{-1}$ is a bisimulation for (TS', TS).
- (Transitivity). If $\mathfrak{R}_{1,2}$ is a bisimulation for (TS_1, TS_2) and $\mathfrak{R}_{2,3}$ a bisimulation for (TS_2, TS_3), then $\mathfrak{R}_{2,3} \circ \mathfrak{R}_{1,2}$ is a bisimulation for (TS_1, TS_3). □

AP-Deterministic

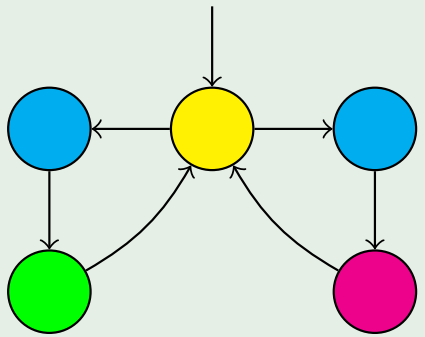
A transition system TS is *AP*-deterministic whenever $|I| \leq 1$ and for all $A \subseteq AP$ we have $|Post(s) \cap \{ s' \in S \mid L(s') = A \}| \leq 1$.

Example

TS:



TS':



- TS is *AP*-deterministic; TS' is **not** *AP*-deterministic

Bisimilarity And Trace Equivalence

1. $TS_1 \sim TS_2$ implies $TS_1 \equiv_{trace} TS_2$.
2. For AP-deterministic TS_1, TS_2: $TS_1 \sim TS_2$ iff $TS_1 \equiv_{trace} TS_2$.

Proof.

1. Follows from the fact that bisimulation carries over to infinite paths.
2. Left as exercise. □

Bisimilarity And Trace Equivalence

AP-det. implies finite branching (if AP is finite)

1. $TS_1 \sim TS_2$ implies $TS_1 \equiv_{trace} TS_2$.
2. For *AP*-deterministic TS_1, TS_2: $TS_1 \sim TS_2$ iff $TS_1 \equiv_{trace} TS_2$.

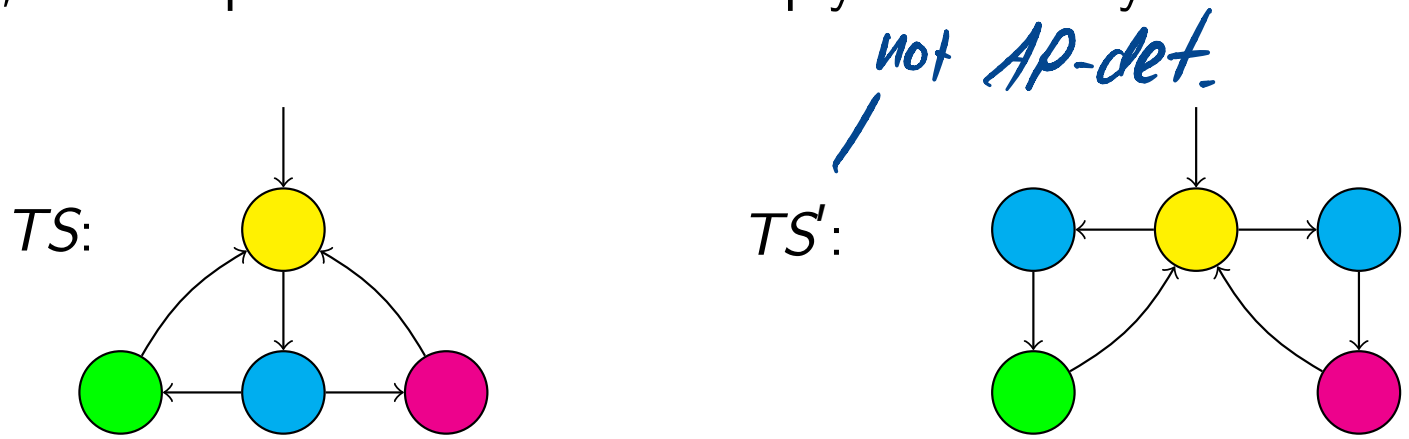
Proof.

1. Follows from the fact that bisimulation carries over to infinite paths.
2. Left as exercise. □

$TS_1 \sim TS_2$ implies $TS_1 \equiv_{LTL} TS_2$. The converse also holds for *AP*-deterministic transition systems.

Distinguishing Bisimilarity And Trace Equivalence

In general, trace equivalence does not imply bisimilarity!



- TS is AP-deterministic; TS' is **not** AP-deterministic
- $TS \equiv_{trace} TS'$
- But $TS \not\leftrightarrow TS'$
  There is no 🔵-state in TS' that has both a 🟢-successor and a 🔴-successor

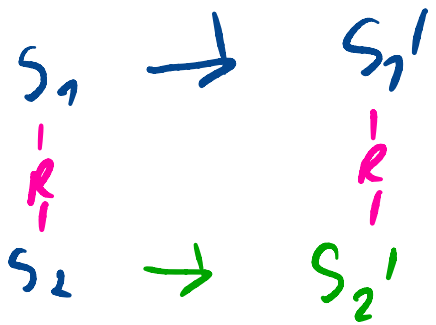
Bisimulation on States

Definition: bisimulation/bisimilarity on states

Symmetric relation $\mathfrak{R} \subseteq S \times S$ is a bisimulation on TS (with state space S) if for any $(s_1, s_2) \in \mathfrak{R}$:

1. $L(s_1) = L(s_2)$
2. $s_1' \in Post(s_1)$ then $(s_1', s_2') \in \mathfrak{R}$ for some $s_2' \in Post(s_2)$.

The states s_1 and s_2 are bisimilar, denoted $s_1 \sim_{TS} s_2$, if $(s_1, s_2) \in \mathfrak{R}$ for some bisimulation $\mathfrak{R}$ for TS.



Bisimulation on States

Definition: bisimulation/bisimilarity on states

Symmetric relation $\mathfrak{R} \subseteq S \times S$ is a bisimulation on TS (with state space S) if for any $(s_1, s_2) \in \mathfrak{R}$:

1. $L(s_1) = L(s_2)$
2. $s_1' \in Post(s_1)$ then $(s_1', s_2') \in \mathfrak{R}$ for some $s_2' \in Post(s_2)$.

The states s_1 and s_2 are bisimilar, denoted $s_1 \sim_{TS} s_2$, if $(s_1, s_2) \in \mathfrak{R}$ for some bisimulation $\mathfrak{R}$ for TS.

$s_1 \sim_{TS} s_2$ if and only if $TS_{s_1} \sim TS_{s_2}$ where TS_{s_i} denotes the transition system TS in which s_i is the only initial state.

Example: Bisimulation on TS

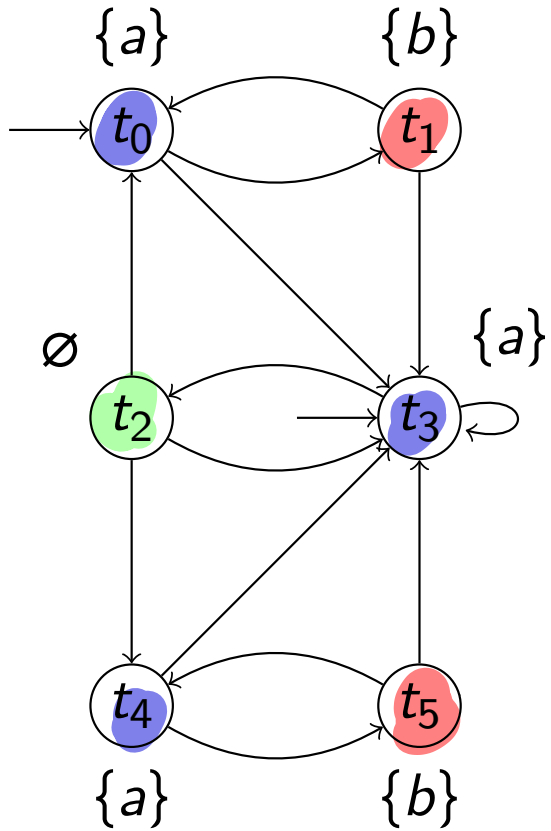
$t_0 \not\sim_{TS} t_3$

because

$t_0 \not\to$ ●

$t_3 \to$ ●

TS:



$$\sim_{TS} = \Big\{(t_0, t_4),\ (t_1, t_5)\Big\} \cup \Big\{(t_4, t_0),\ (t_5, t_1)\Big\} \cup \Big\{(t_i, t_i) \;\Big|\; 0 \leq i \leq 5\Big\}$$

Overview

Logical Equivalence

Definition: logical equivalence

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

- $TS \equiv_{LTL} TS'$ iff $\big(\forall \varphi \in \text{LTL}.\ TS \models \varphi \text{ iff } TS' \models \varphi\big)$
- $TS \equiv_{CTL} TS'$ iff $\big(\forall \Phi \in \text{CTL}.\ TS \models \Phi \text{ iff } TS' \models \Phi\big)$

In a similar way, $\equiv_L$ can be defined for logic L (such as CTL^*, CTL^+, etc.).

$TS \equiv_{trace} TS'$ implies $TS \equiv_{LTL} TS'$.

Logical Equivalence

Definition: logical equivalence

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

- $TS \equiv_{LTL} TS'$ iff $\big(\forall \varphi \in \text{LTL}.\ TS \models \varphi \text{ iff } TS' \models \varphi\big)$
- $TS \equiv_{CTL} TS'$ iff $\big(\forall \Phi \in \text{CTL}.\ TS \models \Phi \text{ iff } TS' \models \Phi\big)$

In a similar way, $\equiv_L$ can be defined for logic L (such as CTL*, CTL$^+$, etc.).

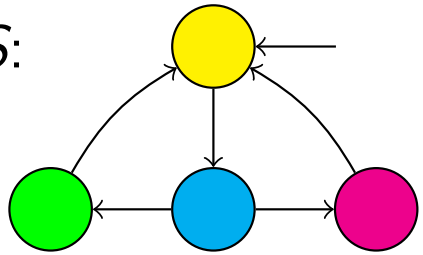
$TS \equiv_{trace} TS'$ implies $TS \equiv_{LTL} TS'$.

A transition system is called finitely-branching if every state has only finitely many direct successors, i.e. $\forall s \in S : |Post(s)| < \infty$.

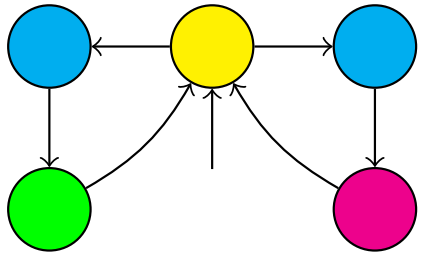
For finitely-branching TS and TS': $TS \equiv_{trace} TS'$ iff $TS \equiv_{LTL} TS'$.

Example: Linear Time

TS:



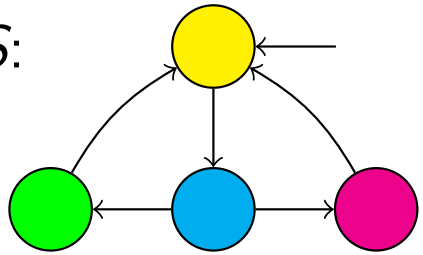
TS':



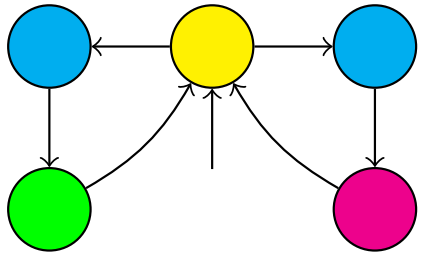
- $TS \equiv_{trace} TS'$
- $TS \equiv_{LTL} TS'$

Example: Linear Time

TS:

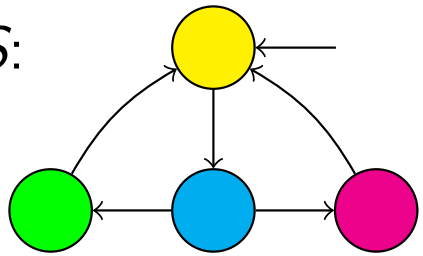


TS':

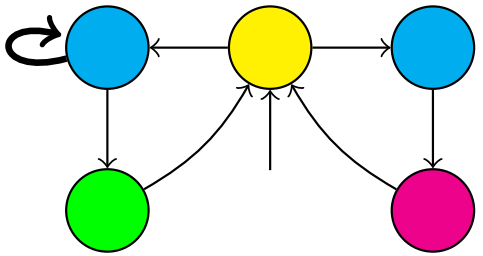


- $TS \equiv_{trace} TS'$
- $TS \equiv_{LTL} TS'$

TS:



TS':



- $TS \not\equiv_{trace} TS'$
- $TS \not\equiv_{LTL} TS'$

$TS \models \bigcirc\bigcirc\neg$● but $TS' \not\models \bigcirc\bigcirc\neg$●

A distinguishing LTL formula

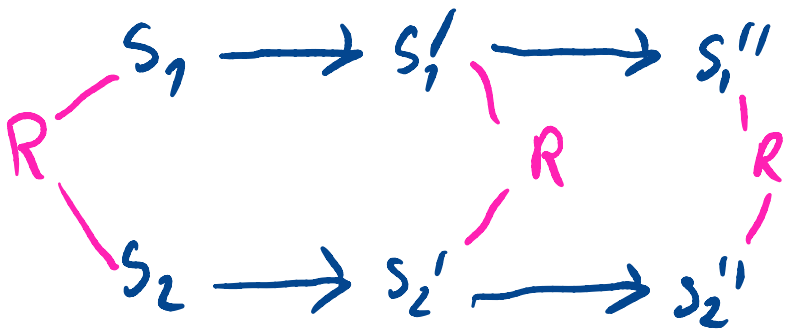
Bisimulation on States

Definition: bisimulation/bisimilarity on states

Symmetric relation $\mathfrak{R} \subseteq S \times S$ is a bisimulation on TS (with state space S) if for any $(s_1, s_2) \in \mathfrak{R}$:

1. $L(s_1) = L(s_2)$
2. $s_1' \in Post(s_1)$ then $(s_1', s_2') \in \mathfrak{R}$ for some $s_2' \in Post(s_2)$.

The states s_1 and s_2 are bisimilar, denoted $s_1 \sim_{TS} s_2$, if $(s_1, s_2) \in \mathfrak{R}$ for some bisimulation $\mathfrak{R}$ for TS.



Bisimulation on States

Definition: bisimulation/bisimilarity on states

Symmetric relation $\mathfrak{R} \subseteq S \times S$ is a bisimulation on TS (with state space S) if for any $(s_1, s_2) \in \mathfrak{R}$:

1. $L(s_1) = L(s_2)$
2. $s_1' \in Post(s_1)$ then $(s_1', s_2') \in \mathfrak{R}$ for some $s_2' \in Post(s_2)$.

The states s_1 and s_2 are bisimilar, denoted $s_1 \sim_{TS} s_2$, if $(s_1, s_2) \in \mathfrak{R}$ for some bisimulation $\mathfrak{R}$ for TS.

TS with $I := \{s_i\}$

$s_1 \sim_{TS} s_2$ if and only if $TS_{s_1} \sim TS_{s_2}$ where TS_{s_i} denotes the transition system TS in which s_i is the only initial state.

Bisimulation on Paths

Whenever we have:

$$\begin{array}{ccccccccc} s_0 & \rightarrow & s_1 & \rightarrow & s_2 & \rightarrow & s_3 & \rightarrow & s_4 \ldots\ldots \\ \mathfrak{R} & & & & & & & & \\ t_0 & & & & & & & & \end{array}$$

this can be completed to

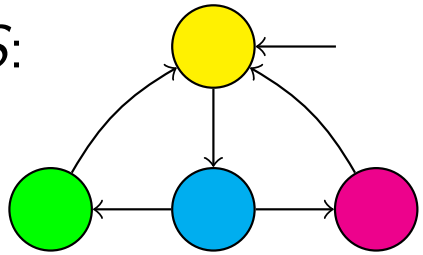
$$\begin{array}{lccccccccc} \pi_1 = & s_0 & \rightarrow & s_1 & \rightarrow & s_2 & \rightarrow & s_3 & \rightarrow & s_4 \ldots\ldots \\ & \mathfrak{R} & & \mathfrak{R} & & \mathfrak{R} & & \mathfrak{R} & & \mathfrak{R} \\ \pi_2 = & t_0 & \rightarrow & t_1 & \rightarrow & t_2 & \rightarrow & t_3 & \rightarrow & t_4 \ldots\ldots \end{array}$$

Proof.

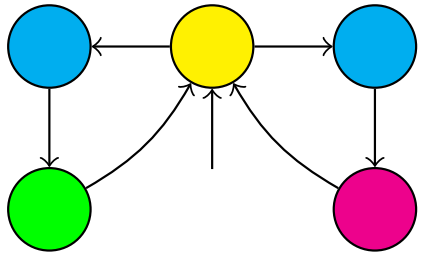
By induction on index i of state s_i. □

Example: Branching Time

TS:



TS':

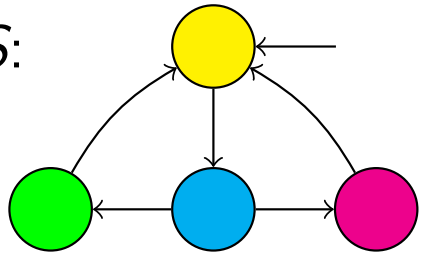


We have $TS \not\leftrightarrow TS'$ because ...

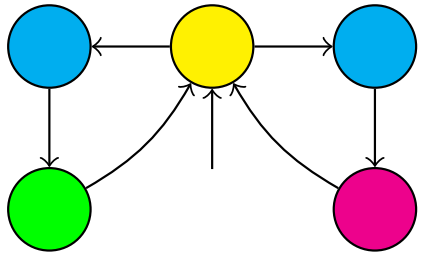
- there is no ●-state in TS' that has both a ●-successor and a ●-successor

Example: Branching Time

TS:



TS':



We have $TS \not\sim TS'$ because ...

- there is no ●-state in TS' that has both a ●-successor and a ●-successor
- expressed in CTL: $TS \models \Phi$ but $TS' \not\models \Phi$ where

$$\Phi \;=\; \exists\Diamond\Big(\bullet \wedge (\exists\bigcirc\,\bullet) \wedge (\exists\bigcirc\,\bullet)\Big)$$

["Distinguishing formula"

Bisimilarity And CTL

Theorem: Bisimilarity, CTL and CTL*

Let TS be a finitely branching[1] transition system and s, s' states in TS. The following statements are equivalent:

1. $s \sim_{TS} s'$
2. s and s' are CTL-equivalent, i.e., $s \equiv_{CTL} s'$
3. s and s' are CTL*-equivalent, i.e., $s \equiv_{CTL^*} s'$.

$s \not\sim s'$ iff $s \not\equiv_{CTL^*} s'$ iff $\exists \Phi \in CTL^*$:

$s \models \Phi$ but $s' \not\models \Phi$

[1] This means that every state has only finitely many direct successors. This theorem does not hold for arbitrary infinite-state transition systems.

Bisimilarity And CTL

Theorem: Bisimilarity, CTL and CTL*

Let TS be a finitely branching[1] transition system and s, s' states in TS. The following statements are equivalent:

1. $s \sim_{TS} s'$
2. s and s' are CTL-equivalent, i.e., $s \equiv_{CTL} s'$
3. s and s' are CTL*-equivalent, i.e., $s \equiv_{CTL^*} s'$.

Proof.

1. $s \equiv_{CTL} s'$ implies $s' \sim s'$ (see lecture notes)
2. $s \sim s'$ implies $s \equiv_{CTL^*} s'$ (see lecture notes)
3. $s \equiv_{CTL^*} s;$ implies $s \equiv_{CTL} s'$ (trivial since $CTL \subseteq CTL^*$)

□

[1] This means that every state has only finitely many direct successors. This theorem does not hold for arbitrary infinite-state transition systems.

② To show: $s_1 \sim s_2$ implies $s_1 \equiv_{CTL^*} s_2$

Two parts: (a) $s_1 \sim s_2$ implies $\forall \Phi \in CTL^*$ (state form.): $s_1 \models \Phi$ iff $s_2 \models \Phi$

(b) $\pi_1 \sim \pi_2$ implies $\forall \varphi \in CTL^*$ (path form.): $\pi_1 \models \varphi$ iff $\pi_2 \models \varphi$

π_1 & π_2 are state-wise bisimilar

Proof: simultaneously show (a) and (b) by induction on Φ and φ. Assume $s_1 \sim s_2$ and $\pi_1 \sim \pi_2$

base:
- case $\Phi = a$: $s_1 \models a$ iff $a \in L(s_1)$
  $s_1 \sim s_2$ iff $a \in L(s_2)$ iff $s_2 \models a$ ✓

hypotheses: (a) and (b) hold for fixed Φ', Φ_1, Φ_2, φ', φ_1, φ_2

step:
- case $\Phi = \exists \varphi'$: By symmetry we only need to show that $s_1 \models \exists \varphi'$ imply $s_2 \models \exists \varphi'$

> **Path lifting lemma**: $s_1 \sim s_2$ and $\pi_1 \in Paths(s_1)$ implies $\exists \pi_2 \in Paths(s_2) : \pi_1 \sim \pi_2$

Let $s_1 \models \exists \varphi'$. Then $\pi_1' \models \varphi'$ for some $\pi_1' \in Paths(s_1)$
$\exists \pi_2' \in Paths(s_2) : \pi_1' \sim \pi_2'$. By I.H. on φ' we get that $\pi_2' \models \varphi'$. Thus $s_2 \models \exists \varphi'$

- case $\varphi = \varphi_1 \, U \, \varphi_2$:
  $\pi_1 \models \varphi_1 \, U \, \varphi_2$ iff (CSL^* semantics) $\exists j : \pi_1[j..] \models \varphi_2$ and $\forall i < j : \pi_1[i..] \models \varphi_1$
  iff (I.H.) $\exists j : \pi_2[j..] \models \varphi_2$ and $\forall i < j : \pi_2[i..] \models \varphi_1$
  iff $\pi_2 \models \varphi_1 \, U \, \varphi_2$

- other cases ($\Phi = \Phi_1 \wedge \Phi_2$, $\varphi = \Phi'$, $\varphi = \neg\varphi$, ...) are omitted here

(7) To show: $s \equiv_{CTL} s'$ implies $s \sim s'$

$\exists$ bisimulation R: $(s,s') \in R$

Proof: Let $R = \{(s,s') \in S \times S \mid s \equiv_{CTL} s'\}$

Claim: R is a bisimulation.

$\rightsquigarrow s \equiv_{CTL} s' \Rightarrow (s,s') \in R \Rightarrow s \sim s'$

- R is symmetric ✓

Let $(s_1, s_2) \in R$

1. As $s_1 \equiv_{CTL} s_2$, we have $\forall a \in AP: s_1 \models a$ iff $s_2 \models a$

   we thus get $L(s_1) = L(s_2)$

2. Let $s_1' \in Post(s_1)$. Since TS. is finitely branching, $Post(s_2) = \{t_1, \dots, t_k\}$ is finite. we need to show that $\exists j \in \{1, \dots, k\}: (s_1', t_j) \in R$.

Towards a contradiction assume that $\forall j: (s_1', t_j) \notin R$

From the def. of $\mathcal{R} = \equiv_{CTL}$ we get that there are CTL formulas $\Psi_1, \dots, \Psi_k$ such that

$\forall j: s_1' \models \Psi_j$ and $t_j \not\models \Psi_j$.

Consider $\Phi := \Psi_1 \wedge \dots \wedge \Psi_k$. we get

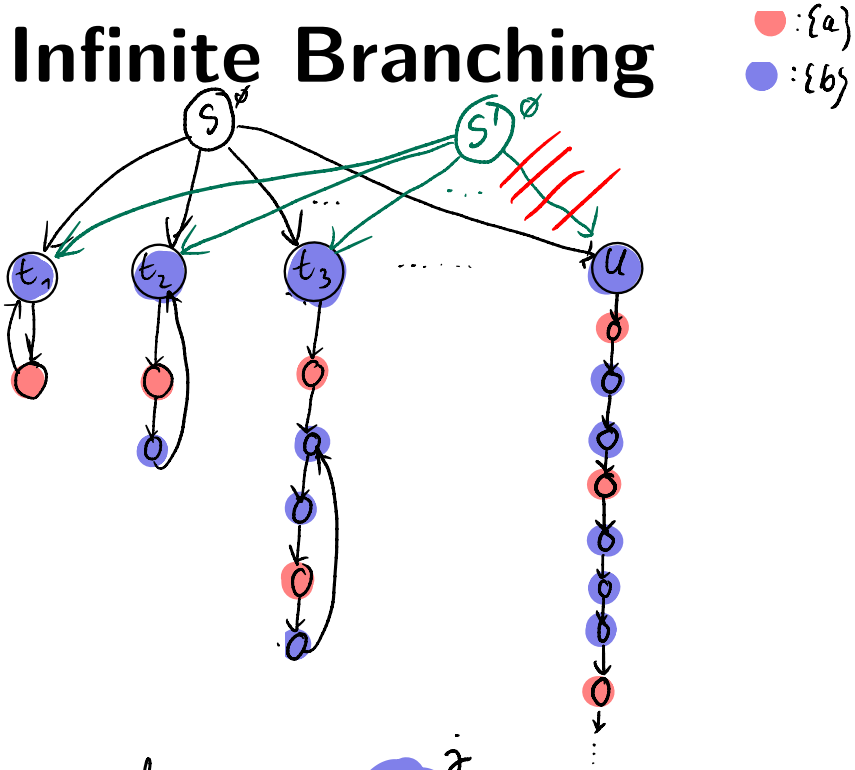
$s_1' \models \Phi$ but $\forall t_j \in Post(s_2): t_j \not\models \Phi$

since $s_1' \in Post(s_1): s_1 \models \exists \bigcirc \Phi$ but $s_2 \not\models \exists \bigcirc \Phi$

↯ contradiction to $(s_1, s_2) \in R$ i.e. $s_1 \equiv_{CTL} s_2$

□

Infinite Branching



So... for $w_j := \{b\}^j \{a\}$:

- Traces $(t_i) = \{w_1 w_2 \cdots w_{i-1} (w_i)^\omega\}$
- Traces $(u) = \{w_1 w_2 w_3 \cdots\}$
- Post $(s) = \{t_i \mid i \in \mathbb{N} \setminus \{0\}\} \cup \{u\}$
- Post $(s') = \{t_i \mid i \in \mathbb{N} \setminus \{0\}\}$

Claim: $s \not\sim s'$ but $s \equiv_{CTL} s'$

① ②

① $s \not\sim s'$ because $s \to u$ but

$\forall t_i \in post(s') : t_i \not\sim u$

(follows from path lifting Lemma)

② key insight:

$\forall \Phi \in CTL . \exists i \geq 0:$

$t_i \models \Phi$ iff $u \models \Phi$

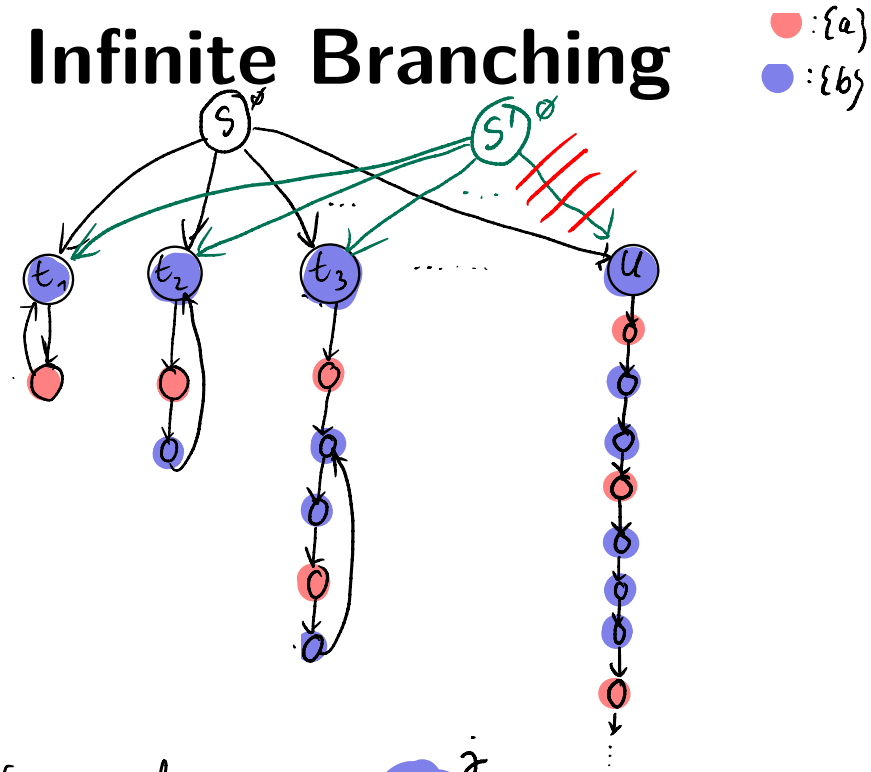
Intuition: If $n \in \mathbb{N}$ is the number of $\bigcirc$-formulae in Φ, then Φ can not distinguish between sequences of

$n+1$ many $\{b\}$ and

$n+j$ many $\{b\}$ for $j \geq 1$

$\rightsquigarrow t_{n+1} \models \Phi$ iff $u \models \Phi$

Infinite Branching



So... for $w_i := \{b\}^i \{a\}$:

- Traces $(t_i) = \{w_1 w_2 \cdots w_{i-1} (w_i)^\omega\}$
- Traces $(u) = \{w_1 w_2 w_3 \cdots\}$
- Post $(s) = \{t_i \mid i \in \mathbb{N} \setminus \{0\}\} \cup \{u\}$
- Post $(s') = \{t_i \mid i \in \mathbb{N} \setminus \{0\}\}$

Claim: $s \not\sim s'$ but $s \equiv_{CTL} s'$

Lifting to Transition Systems

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

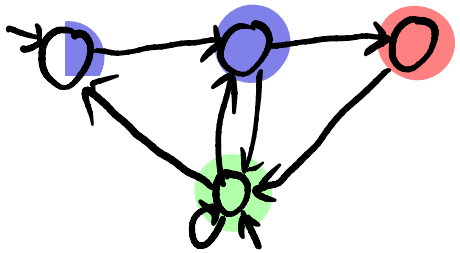
$$TS \sim TS' \quad \text{iff} \quad TS \equiv_{CTL} TS' \quad \text{iff} \quad TS \equiv_{CTL^*} TS'.$$

if $TS_1 \not\equiv_{CTL^*} TS_2 \longrightarrow TS \not\sim TS'$

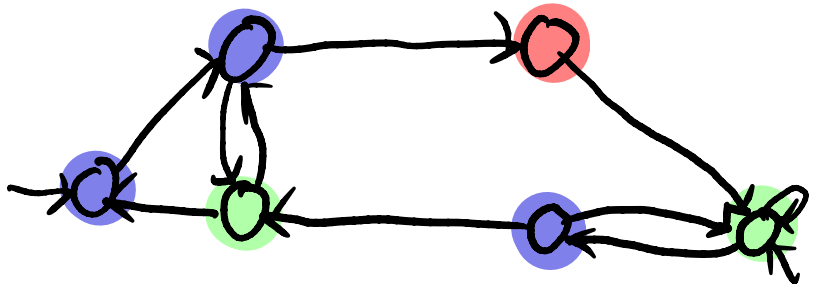
if $TS_1 \not\equiv_{CTL} TS_2 \longrightarrow TS \not\sim TS'$

Example

TS_1:



TS_2:

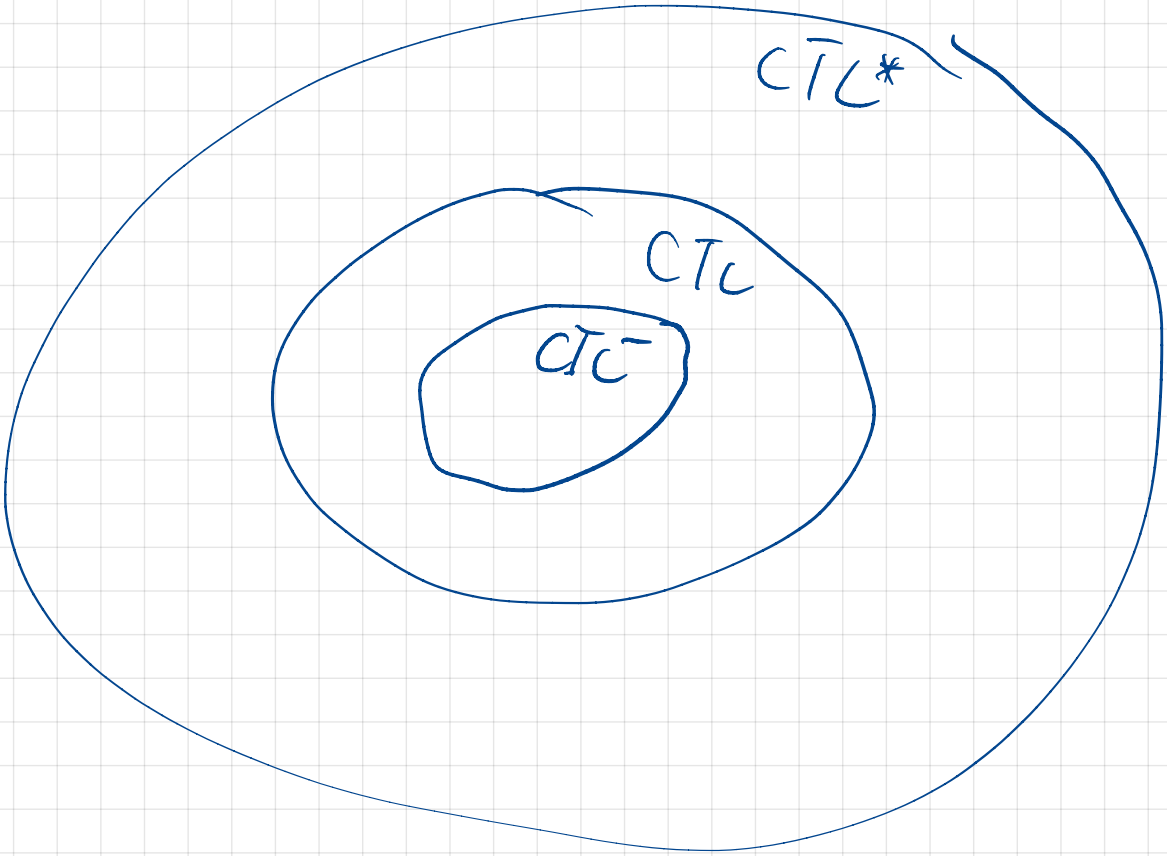


Question: $TS_1 \sim TS_2$? No!

Consider $\Phi = \bullet \Rightarrow (\exists \bigcirc (\bullet \wedge (\exists \bigcirc \bullet)))$

$TS_1 \models \Phi$ but $TS_2 \not\models \Phi$

Thus $TS_1 \not\sim TS_2$



$\equiv_{CTL^*} \quad = \quad \equiv_{CTL} \quad = \quad \equiv_{CTL^-}$

A Simple Fragment of CTL

Definition: CTL$^-$

CTL$^-$ state-formulas with $a \in AP$ obey the grammar:

$$\Phi \ ::= \ \text{true} \ \Big| \ a \ \Big| \ \Phi_1 \wedge \Phi_2 \ \Big| \ \neg\Phi \ \Big| \ \exists\bigcirc\Phi \ \Big| \ \forall\bigcirc\Phi$$

υ

No until-modalities, so no $\Box$ and no $\Diamond$

A Simple Fragment of CTL

Definition: CTL$^-$

CTL$^-$ state-formulas with $a \in AP$ obey the grammar:

$$\Phi \ ::= \ \text{true} \ \Big| \ a \ \Big| \ \Phi_1 \wedge \Phi_2 \ \Big| \ \neg\Phi \ \Big| \ \exists\bigcirc\Phi \ \Big| \ \forall\bigcirc\Phi$$

No until-modalities, so no $\Box$ and no $\Diamond$

1. CTL$^-$ is strictly less expressive than CTL (and than CTL*).
2. CTL$^-$ equivalence coincides with CTL (and CTL*) equivalence.

Proof.

Follows from the fact that in the proof of equivalence of $\sim$, $\equiv_{CTL}$ and $\equiv_{CTL^*}$ only CTL$^-$-formulas are used. In particular, no until-modalities are used. $\square$

The Importance of These Results

- CTL^--, CTL- and CTL^*-equivalence coincide
  - despite the fact that these logics have different expressivity
- Bisimilar transition systems preserve the same CTL^* formulas
  - and thus the same LTL formulas (and LT properties)
- Non-bisimilarity can be shown by a single CTL^- formula Φ
  - $TS_1 \models \Phi$ and $TS_2 \not\models \Phi$ implies $TS_1 \not\sim TS_2$
- One does not even need to use an until-modality

On Complexity

Given, TS_1, TS_2 finite does $TS_1 \equiv_{Trace} TS_2$?

The decision problem whether two finite transition systems are trace equivalent is PSPACE-complete.

Proof.

Reduction from language equivalence of nondeterministic finite-state automata. □

The decision problem whether two finite transition systems are bisimilar is PTIME-complete.

Proof.

A polynomial-time algorithm will be dealt with in an upcoming lecture. PTIME-hardness is outside the scope of this lecture. □

Overview

Reminder: Syntax of CTL*

Definition: Syntax CTL*

- CTL* state-formulas with $a \in AP$ obey the grammar:

$$\Phi \ ::= \ \text{true} \ \Big| \ a \ \Big| \ \Phi_1 \wedge \Phi_2 \ \Big| \ \neg\Phi \ \Big| \ \exists\varphi$$

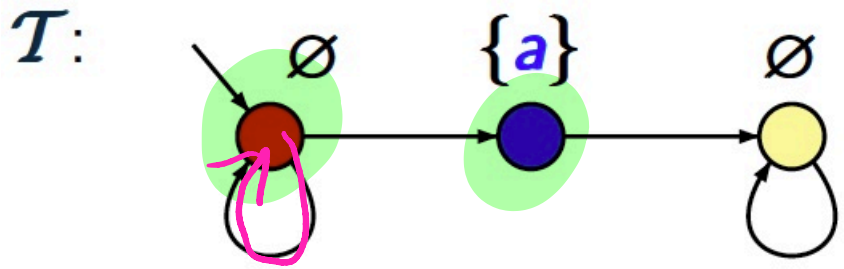
- and φ is a CTL* path-formula formed by the grammar:

$$\varphi \ ::= \ \Phi \ \Big| \ \varphi_1 \wedge \varphi_2 \ \Big| \ \neg\varphi \ \Big| \ \bigcirc\varphi \ \Big| \ \varphi_1 \, \mathsf{U} \, \varphi_2$$

where Φ is a CTL* state-formula, and φ, φ_1 and φ_2 are path-formulas.

in CTL*: $\forall\varphi \ = \ \neg\exists\neg\varphi$. This does not hold in CTL.

Example



CTL CTL*

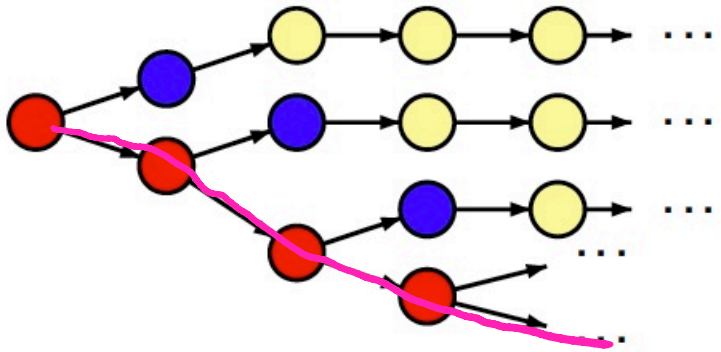
$\exists\Box\exists\Diamond a \not\equiv \exists\Box\Diamond a$

$\mathcal{T} \not\models \exists\Box\Diamond a$

$\mathcal{T} \models \exists\Box\exists\Diamond a$ note: $Sat(\exists\Diamond a) = \{ \bullet, \bullet \}$

hence: $\bullet\bullet\bullet\ldots \models \Box\exists\Diamond a$

computation tree:



Embedding LTL

iff $\forall \pi \in Paths(s) : \pi \models \varphi$

For LTL formula φ and TS without terminal states (both over AP) and for each $s \in S$:

$\underbrace{s \models \varphi}_{\text{LTL semantics}}$ if and only if $\underbrace{s \models \forall\varphi}_{\text{CTL}^* \text{ semantics}}$

iff $s \models \neg\exists\neg\varphi$

In particular:

$$TS \models_{LTL} \varphi \quad \text{if and only if} \quad TS \models_{CTL*} \forall\varphi$$

CTL* Model Checking

CTL* formula (without A)

[Emerson & Lei, 1985]

- Adopt a recursive descent over the parse tree of Φ (as for CTL)
- Replace maximal proper state sub-formula Ψ by new proposition a_Ψ
  - adjust labeling such that $a_\Psi \in L(s)$ if and only if $s \in Sat(\Psi)$

state-subformula that does not contain any CTL* path formula

CTL* Model Checking

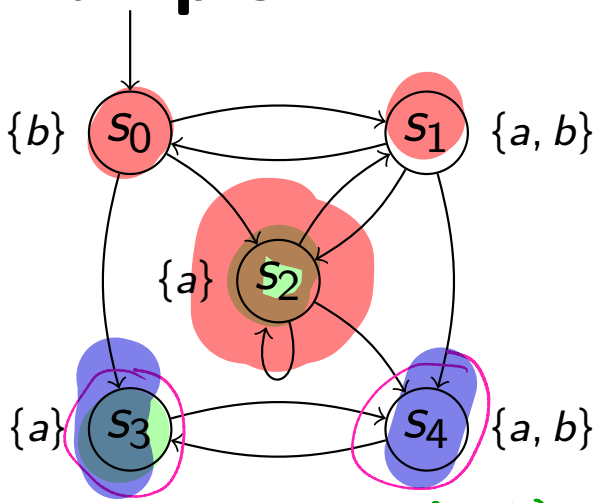
[Emerson & Lei, 1985]

- Adopt a recursive descent over the parse tree of Φ (as for CTL)
- Replace maximal proper state sub-formula Ψ by new proposition a_Ψ
  - adjust labeling such that $a_\Psi \in L(s)$ if and only if $s \in Sat(\Psi)$
- In the end, this yields an LTL formula
- Most interesting case: formulas of the form $\exists\varphi$

$$s \models_{CTL^*} \exists\varphi \quad \text{iff} \quad \underbrace{s \not\models_{CTL^*} \forall\neg\varphi}_{\text{CTL}^* \text{ semantics}} \quad \text{iff} \quad \underbrace{s \not\models_{LTL} \neg\varphi}_{\text{LTL semantics}}$$

$$Sat_{CTL*}(\exists\varphi) = S \setminus Sat_{LTL}(\neg\varphi) = S \setminus \{ s \in S \mid s \models_{LTL} \neg\varphi \}$$

Example



- $Sat(a \wedge \neg b) = \{s_2, s_3\}$
- $s_3 \models b \,U\, \Box a$
  $s_4 \not\models$

CTL* formula

$$\Phi = \exists\Big(\bigcirc(a \wedge \neg b) \wedge \bigcirc \forall(b \,U\, \Box a)\Big)$$

↳ eliminate $\forall$

$$\Phi = \exists(\bigcirc(\underbrace{a \wedge \neg b}_{\psi_1}) \wedge \bigcirc \neg\exists\neg(b \,U\, \Box a))$$

$Sat(\psi_1) = \{s_1, s_3\}$ ⇝ introduce new AP a_{ψ_1}

$$\Phi = \exists(\bigcirc\, a_{\psi_1} \wedge \bigcirc \neg\exists \underbrace{\neg(b \,U\, \Box a)}_{\psi_2 \in LTL})$$

we use LTL model checking to compute $Sat(\exists\psi_2)$

$s \models_{CTL^*} \exists\psi_2$ iff $s \not\models_{CTL^*} \forall\neg\psi_2$ iff $s \not\models_{LTL} \neg\psi_2$

$s \not\models_{LTL} b \,U\, \Box a$

Thus $Sat(\exists\psi_2) = S \setminus Sat_{LTL}(b \,U\, \Box a) = S \setminus \{s_2, s_4\} = \{s_0, s_1, s_3\}$

Introduce a_{ψ_2} as a new AP

$$= \exists(\bigcirc\, a_{\psi_1} \wedge \bigcirc \underbrace{\neg\, a_{\psi_2}}_{\psi_3})$$

$$= \exists(\underbrace{\bigcirc\, a_{\psi_1} \wedge \bigcirc\, a_{\psi_3}}_{\psi_4})$$

$Sat(\exists\psi_4) = S \setminus Sat_{LTL}(\neg\psi_4)$

$= S \setminus Sat_{LTL}(\bigcirc \neg a_{\psi_1} \vee \bigcirc \neg a_{\psi_3})$

$= S \setminus \{s_3, s_2, s_1\}$

$= \{s_0, s_4\} = Sat(\Phi)$

$I \subseteq \{s_0, s_4\}$ ⇝ $TS \models \Phi$

Complexity

The CTL* model-checking algorithm for finite transition system TS and CTL*-formula Φ has a time complexity in $O(|TS| \cdot 2^{|\Phi|})$.

Proof.

The recursive descent is linear in $|\Phi|$. The most expensive procedure for a node, i.e., sub-formula $\Psi = \exists\varphi$ of Φ, in the parse tree is in $O(|TS| \cdot 2^{|\Psi|})$. □

Complexity

The CTL* model-checking algorithm for finite transition system TS and CTL*-formula Φ has a time complexity in $O(|TS| \cdot 2^{|\Phi|})$.

Proof.

The recursive descent is linear in $|\Phi|$. The most expensive procedure for a node, i.e., sub-formula $\Psi = \exists\varphi$ of Φ, in the parse tree is in $O(|TS| \cdot 2^{|\Psi|})$. □

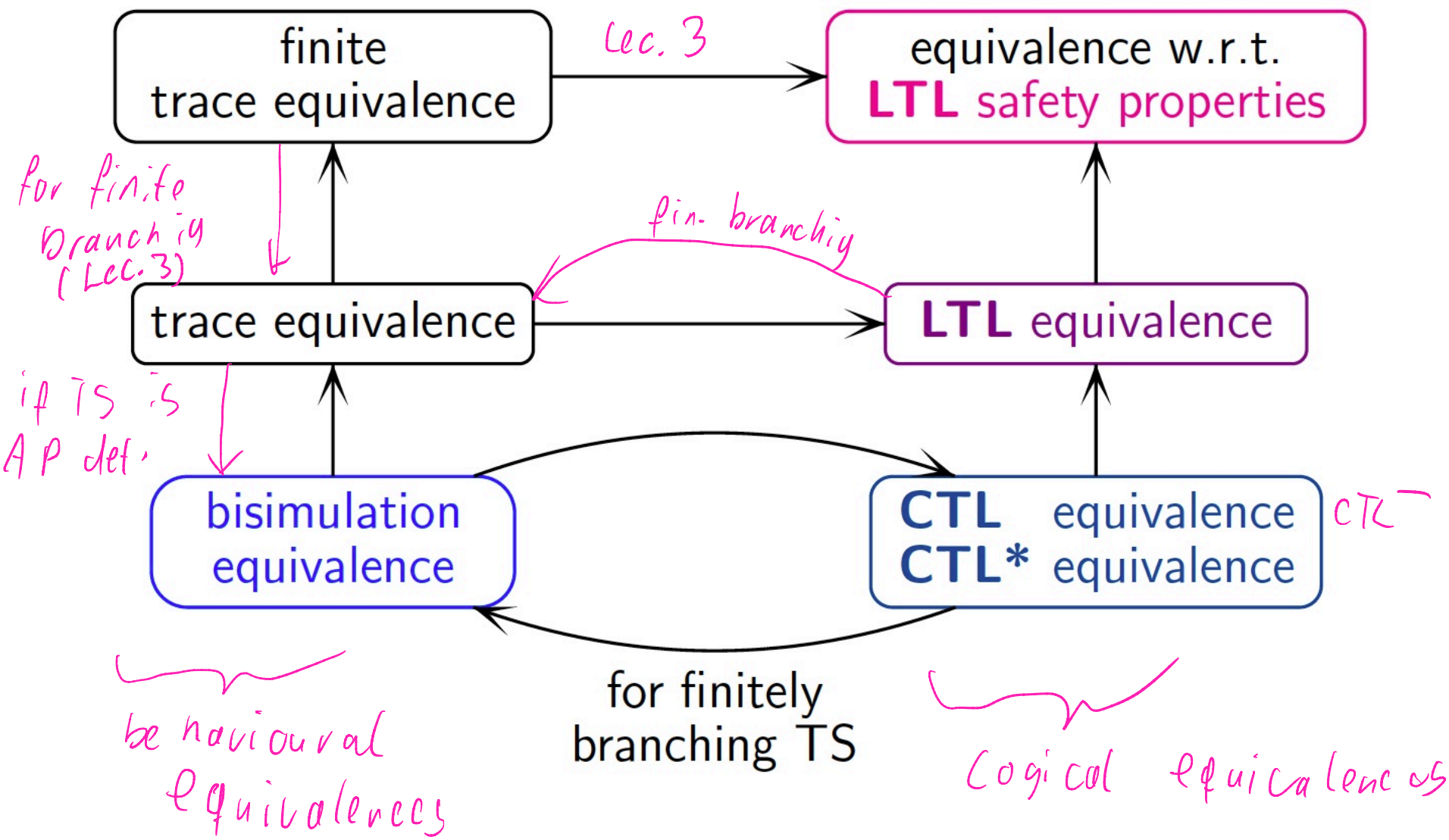
The CTL* model-checking problem is PSPACE-complete.

Proof.

Outside the scope of this lecture series. □

Overview

Summary: Equivalences



Complexity Overview

	CTL	LTL	CTL*
model checking	PTIME	PSPACE	PSPACE
algorithmic complexity	$\|TS\| \cdot \|\Phi\|$	$\|TS\| \cdot \exp(\|\varphi\|)$	$\|TS\| \cdot \exp(\|\Phi\|)$
satisfiability	EXPTIME	PSPACE	2EXPTIME
equivalence	bisimilarity	trace equivalence	bisimilarity
equivalence checking	PTIME	PSPACE	PTIME

All theoretical complexity indications are complete.

Next Lecture

Monday June 13, 10:30