Concurrency Theory

Lecture 1: Motivation & Introduction

Dr. Stephan Mennicke

Institute for Theoretical Computer Science
Knowledge-Based Systems Group

April 7, 2025

TECHNISCHE UNIVERSITÄT DRESDEN

International Center for Computational Logic

Organization

Sessions

- Mondays DS4 (13.00–14.30), APB E005 planned as *lectures*
- Tuesdays DS3 (11.10–12.40), APB E005 planned as *exercise*
- **exception:** tomorrow (April 8) also planned as lecture

Non-Sessions

- April 21 & 22 (*Easter week*)
- June 9 & 10 (*Pentecost week*)

Webpage

https://iccl.inf.tu-dresden.de/web/Concurrency_Theory_(SS2025)

Lecture Notes

Slides will be published online.

Goals of the Course

1. Basic notions of **concurrency theory**
   - computation = *interaction*
   - (global) states vs. *processes*
2. Process semantics = **bisimilarity**
   - comparative semantics
   - congruence results
3. Process control
   - process calculi (here, Milner's CCS)
   - Petri nets
4. Advanced **proof tools**
   - coinductive proofs – the bisimulation proof method
   - weak simulation – undecidability, even for non-Turing-complete models
   - exploiting well-quasi orderings – deciding *non-halting* in infinite state spaces
   - inductive counting on steroids – how far can Petri nets count?

(Non-)Prerequisites

- No particular prior course needed
- Basic knowledge in *Theoretical Computer Science* helpful (e.g., computation, Turing machines, a bit of complexity theory)
- General mathematical skills

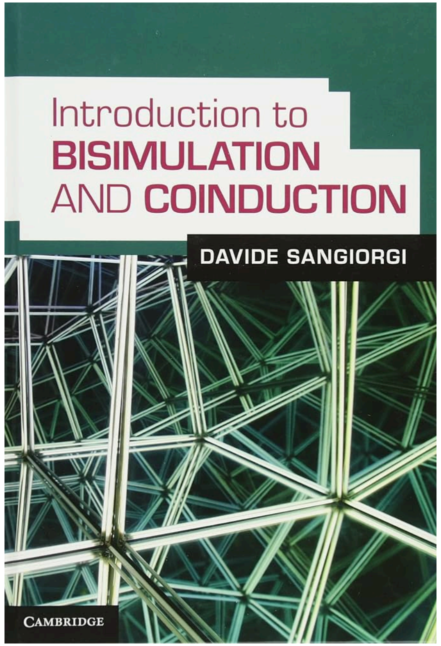
Example Skill

Let A be a set. We call $\mathcal{R} \subseteq A \times A$ an *equivalence relation* if

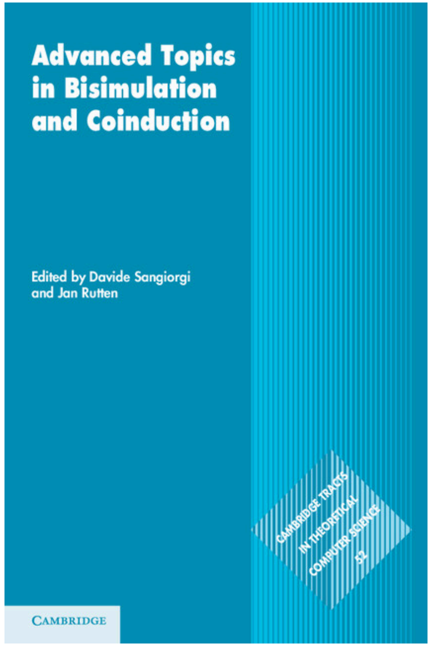
1. $\mathcal{R}$ is *reflexive*, $\forall a \in A : (a, a) \in \mathcal{R}$
2. $\mathcal{R}$ is *symmetric*, and $\forall a, b \in A : (a, b) \in \mathcal{R} \rightarrow (b, a) \in \mathcal{R}$
3. $\mathcal{R}$ is *transitive*. $\forall a, b, c \in A : (a, b) \in \mathcal{R} \wedge (b, c) \in \mathcal{R} \rightarrow (a, c) \in \mathcal{R}$

If $\mathcal{R}$ is binary, we write $a \mathrel{\mathcal{R}} b$ for $(a, b) \in \mathcal{R}$.

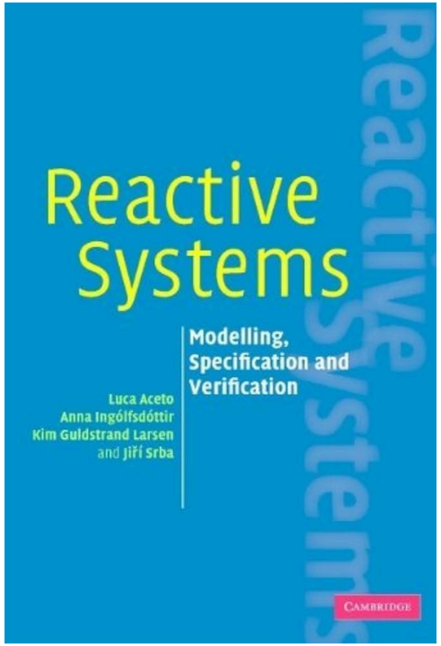
Reading List



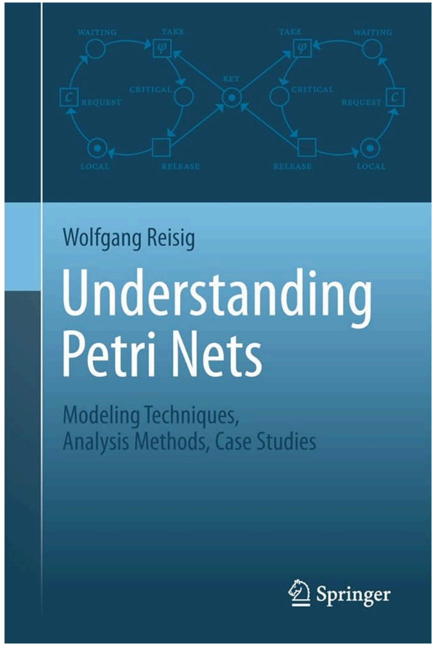
[1]



[2]

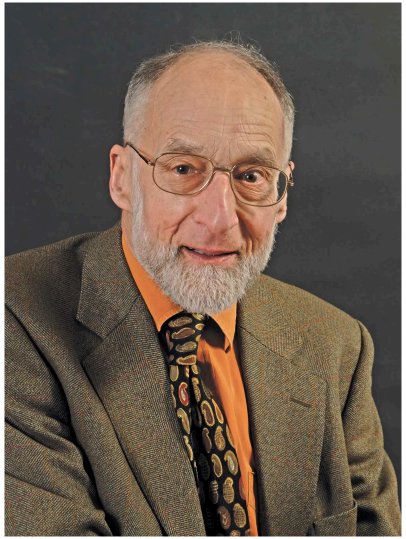


[3]



[4]

Three People



The Core Model of Concurrency Theory

What is a Process

Example. *The vending machine has*

- *a slot for consuming coins (€);*
- *two buttons, one for picking coffee (☕) and one for tea (🍵);*
- *one compartment for providing beverages (🥡).*

- What kind of description is this?
- Who **interacts** with the vending machine?
- When can interaction take place?
- How can interaction take place?

Example. *The vending machine accepts a coin (€) and offers either coffee (☕) or (🍵). Depending on the choice, a beverage (🥡) is returned.*

What is a Process

Example. *The vending machine accepts a coin (€) and offers either coffee (☕) or (🍵). Depending on the choice, a beverage (🥡) is returned.*

Open Questions

1. Do we always have to insert € first?
2. Can we freely choose between ☕ and 🍵 ?
3. Does the machine return to its initial state once a 🥡 is returned?

Resolution

- *mathematically precise* description of **processes** and their **behavior**
- **behavior** is described by what we can observe by *interacting* with a **process**
- *interaction* here: very simple *handshake*

Suggestions?

1. (Nondeterministic) Finite Automata
2. Tree Transducers
3. Turing Machines
4. Minsky machines
5. Programming Language XYZ
6. (formal) grammars
7. Petri nets
8. …
9. …
10. …
11. …

in this course: **Labeled Transition Systems** (LTS*s*)

Labeled Transition Systems (LTS)

Definition 1 (Labeled Transition System) We call a triple $\mathcal{T} = (\mathsf{Pr}, \mathsf{Act}, \longrightarrow)$ a *labeled transition system* (or LTS for short) if Pr is a set of *processes*, Act a set of *actions* such that $\mathsf{Pr} \cap \mathsf{Act} = \emptyset$, and $\longrightarrow \subseteq \mathsf{Pr} \times \mathsf{Act} \times \mathsf{Pr}$ the *labeled transition relation* of $\mathcal{T}$.

Labeled Transition Systems (LTS)

Definition 1 (Labeled Transition System) We call a triple $\mathcal{T} = (\mathsf{Pr}, \mathsf{Act}, \longrightarrow)$ a *labeled transition system* (or LTS for short) if Pr is a set of *processes*, Act a set of *actions* such that $\mathsf{Pr} \cap \mathsf{Act} = \emptyset$, and $\longrightarrow \subseteq \mathsf{Pr} \times \mathsf{Act} \times \mathsf{Pr}$ the *labeled transition relation* of $\mathcal{T}$.

Processes Pr

- possibly infinite set
- abstract elements are $p, q, r, \ldots, p_0, p_1, p_2, \ldots$
- automaton analogy: *states*

Actions Act

- possibly infinite set
- disjoint from Pr
- elements are $a, b, c, \ldots, \alpha, \beta, \gamma, \ldots$
- thought of as *irreducible* active components of processes
- automaton analogy: *alphabet symbols*

LTS Examples

Definition 1 (Labeled Transition System) We call a triple $\mathcal{T} = (\mathsf{Pr}, \mathsf{Act}, \longrightarrow)$ a *labeled transition system* (or LTS for short) if Pr is a set of *processes*, Act a set of *actions* such that $\mathsf{Pr} \cap \mathsf{Act} = \emptyset$, and $\longrightarrow \subseteq \mathsf{Pr} \times \mathsf{Act} \times \mathsf{Pr}$ the *labeled transition relation* of $\mathcal{T}$.

Example. $(\{p, q, r\}, \{a, b, c\}, \{(p, a, q), (p, a, p), (q, b, p), (q, c, r)\})$

Example.
$(\{p, q, r_1, r_2\}, \{\text{☕}, \text{🍵}, \text{🥡}, \text{€}\}, \{(p, \text{€}, q), (q, \text{☕}, r_1), (q, \text{🍵}, r_2), (r_1, \text{🥡}, p), (r_2, \text{🥡}, p)\})$

Example. $(\mathbb{N} \times \mathbb{N}, \{a\}, \{(\langle 0, 0\rangle, a, \langle 0, n\rangle) \mid n \in \mathbb{N}\} \cup \{(\langle m, n\rangle, a, \langle m+1, n\rangle) \mid m < n\})$

LTS Notation and Somewhat Important Classes

Definition 1 (Labeled Transition System) We call a triple $\mathcal{T} = (\mathsf{Pr}, \mathsf{Act}, \longrightarrow)$ a *labeled transition system* (or LTS for short) if Pr is a set of *processes*, Act a set of *actions* such that $\mathsf{Pr} \cap \mathsf{Act} = \emptyset$, and $\longrightarrow \subseteq \mathsf{Pr} \times \mathsf{Act} \times \mathsf{Pr}$ the *labeled transition relation* of $\mathcal{T}$.

- write $p \xrightarrow{a} q$ for $(p, a, q) \in \longrightarrow$
- write $p \xrightarrow{a}$ if there is a $q \in \mathsf{Pr}$ such that $p \xrightarrow{a} q$
- write $p \not\xrightarrow{a}$ if there is no $q \in \mathsf{Pr}$ such that $p \xrightarrow{a} q$

LTS Notation and Somewhat Important Classes

Definition 1 (Labeled Transition System) We call a triple $\mathcal{T} = (\mathsf{Pr}, \mathsf{Act}, \longrightarrow)$ a *labeled transition system* (or LTS for short) if Pr is a set of *processes*, Act a set of *actions* such that $\mathsf{Pr} \cap \mathsf{Act} = \emptyset$, and $\longrightarrow \subseteq \mathsf{Pr} \times \mathsf{Act} \times \mathsf{Pr}$ the *labeled transition relation* of $\mathcal{T}$.

Definition 2 An LTS $\mathcal{T} = (\mathsf{Pr}, \mathsf{Act}, \longrightarrow)$ is

image-finite if for each $p \in \mathsf{Pr}$ and $a \in \mathsf{Act}$, the set $\left\{p' \in \mathsf{Pr} \,\middle|\, p \xrightarrow{a} p'\right\}$ is finite.

finitely branching if it is image-finite and the set $\left\{\mu \in \mathsf{Act} \,\middle|\, p \xrightarrow{\mu}\right\}$ is finite.

finite-state if it has a finite number of states.

LTS Notation and Somewhat Important Classes

Definition 2 An LTS $\mathcal{T} = (\mathsf{Pr}, \mathsf{Act}, \longrightarrow)$ is

image-finite if for each $p \in \mathsf{Pr}$ and $a \in \mathsf{Act}$, the set $\left\{p' \in \mathsf{Pr} \,\middle|\, p \xrightarrow{a} p'\right\}$ is finite.

finitely branching if it is image-finite and the set $\left\{\mu \in \mathsf{Act} \,\middle|\, p \xrightarrow{\mu}\right\}$ is finite.

finite-state if it has a finite number of states.

Example. *Have* $(\{p, q, r\}, \{a, b, c\}, \longrightarrow)$ *such that*

$p \xrightarrow{a} q, p \xrightarrow{a} p, q \xrightarrow{b} p$, and $q \xrightarrow{c} r$ *finite-state and finitely branching*

Exercise. Show that finite-state implies finitely branching if the set of actions is finite.

LTS Notation and Somewhat Important Classes

Definition 2 An LTS $\mathcal{T} = (\mathsf{Pr}, \mathsf{Act}, \longrightarrow)$ is

image-finite if for each $p \in \mathsf{Pr}$ and $a \in \mathsf{Act}$, the set $\left\{ p' \in \mathsf{Pr} \;\middle|\; p \xrightarrow{a} p' \right\}$ is finite.

finitely branching if it is image-finite and the set $\left\{ \mu \in \mathsf{Act} \;\middle|\; p \xrightarrow{\mu} \right\}$ is finite.

finite-state if it has a finite number of states.

Example. *Have* $(\{p, q, r_1, r_2\}, \{\text{☕}, \text{🍵}, \text{🥡}, \text{€}\}, \longrightarrow)$ *such that* $p \xrightarrow{\text{€}} q$, $q \xrightarrow{\text{☕}} r_1$, $q \xrightarrow{\text{🍵}} r_2$, *and* $r_i \xrightarrow{\text{🥡}} p$ $(i = 1, 2)$.

finite-state

LTS Notation and Somewhat Important Classes

Definition 2 An LTS $\mathcal{T} = (\mathsf{Pr}, \mathsf{Act}, \longrightarrow)$ is

image-finite if for each $p \in \mathsf{Pr}$ and $a \in \mathsf{Act}$, the set $\left\{p' \in \mathsf{Pr} \,\middle|\, p \xrightarrow{a} p'\right\}$ is finite.

finitely branching if it is image-finite and the set $\left\{\mu \in \mathsf{Act} \,\middle|\, p \xrightarrow{\mu}\right\}$ is finite.

finite-state if it has a finite number of states.

Example. *Have* $(\mathbb{N} \times \mathbb{N}, \{a\}, \longrightarrow)$ *with*

$$\longrightarrow := \{(\langle 0, 0\rangle, a, \langle 0, n\rangle) \mid n \in \mathbb{N}\} \cup \{(\langle m, n\rangle, a, \langle m+1, n\rangle) \mid m < n\}$$

image-finite

Yet Another Well-Known Class

Definition 2 An LTS $\mathcal{T} = (\mathsf{Pr}, \mathsf{Act}, \longrightarrow)$ is

image-finite if for each $p \in \mathsf{Pr}$ and $a \in \mathsf{Act}$, the set $\left\{p' \in \mathsf{Pr} \,\middle|\, p \xrightarrow{a} p'\right\}$ is finite.

finitely branching if it is image-finite and the set $\left\{\mu \in \mathsf{Act} \,\middle|\, p \xrightarrow{\mu}\right\}$ is finite.

finite-state if it has a finite number of states.

Definition 3 An LTS $\mathcal{T}$ is *deterministic* if for all processes $p \in \mathsf{Pr}$ and all $a \in \mathsf{Act}$, $p \xrightarrow{a} p'$ and $p \xrightarrow{a} p''$ implies $p' = p''$.

Exercise. Show that deterministic implies image-finite.

Relations to Graph Theory

Process Graphs

Despite the fact, that we usually deal with graphs as *finite* objects, LTSs show certain commonalities with graphs: graph nodes vs. processes; (labeled) edges vs. transitions

Definition 4 A *process graph* is a directed, rooted, edge-labeled, and possibly infinite graph $G = (V, r, E)$ such that V is its set of *nodes*, $r \in V$ is called the *root of* G ($\mathsf{root}(G) := r$), and $E \subseteq V \times \mathsf{Act} \times V$ is its directed and labeled edge relation.

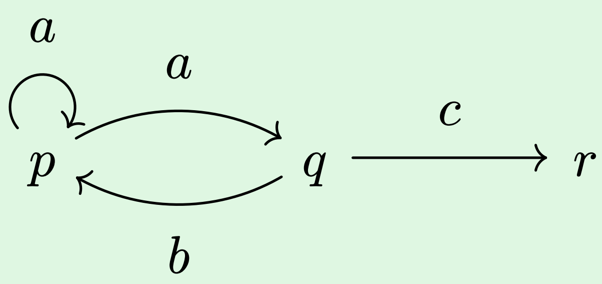
Definition 5 For LTS $(\mathsf{Pr}, \mathsf{Act}, \longrightarrow)$ and process $p \in \mathsf{Pr}$, define $G(p) := (V, p, E)$ by

1. V is the smallest set such that
   - $p \in V$ and
   - if $q \in V$ and $q \xrightarrow{a} r$, then $r \in V$;
2. $E := \left\{ (q, b, r) \;\middle|\; q, r \in V \text{ and } q \xrightarrow{b} r \right\}$

Process Graphs

Example. *Have* $(\{p, q, r\}, \{a, b, c\}, \longrightarrow)$ *such that* $p \xrightarrow{a} q$, $p \xrightarrow{a} p$, $q \xrightarrow{b} p$, and $q \xrightarrow{c} r$

$G(p)$:



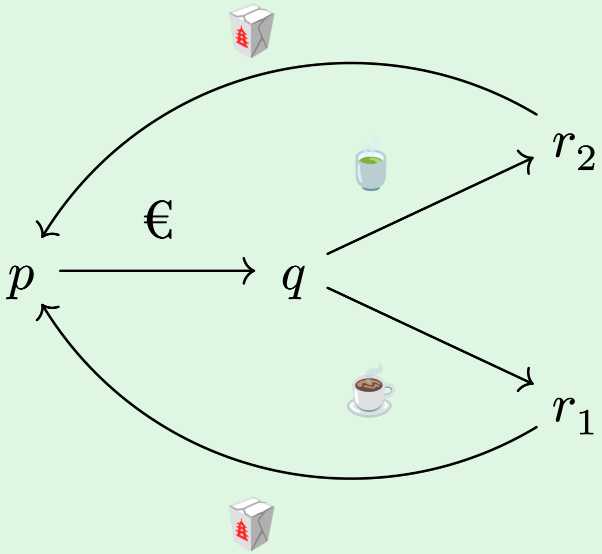
$G(r)$:

$$r$$

Process Graphs

Example. *Have* $(\{p, q, r_1, r_2\}, \{$☕, 🍵, 🥡, €$\}, \longrightarrow)$ *such that* $p \xrightarrow{\text{€}} q$, $q \xrightarrow{\text{☕}} r_1$, $q \xrightarrow{\text{🍵}} r_2$, *and* $r_i \xrightarrow{\text{🥡}} p$ $(i = 1, 2)$.

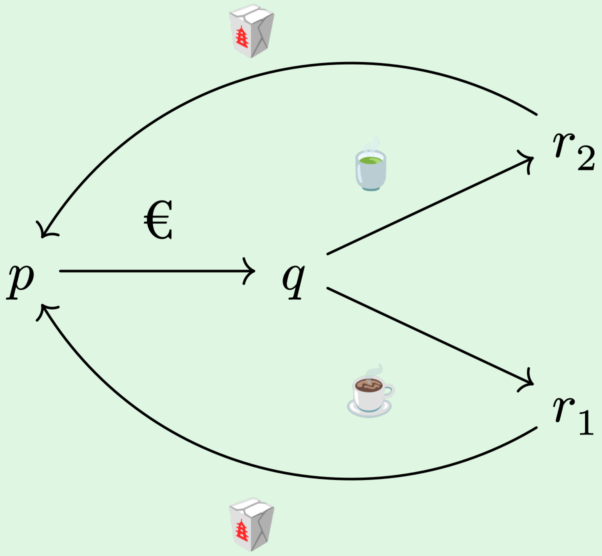
$G(p)$:



Process Graphs

Example. *Have* $(\{p, q, r_1, r_2\}, \{\text{☕}, \text{🍵}, \text{🥡}, \text{€}\}, \longrightarrow)$ *such that* $p \xrightarrow{\text{€}} q$, $q \xrightarrow{\text{☕}} r_1$, $q \xrightarrow{\text{🍵}} r_2$, *and* $r_i \xrightarrow{\text{🥡}} p$ $(i = 1, 2)$.

$G(r_1)$:



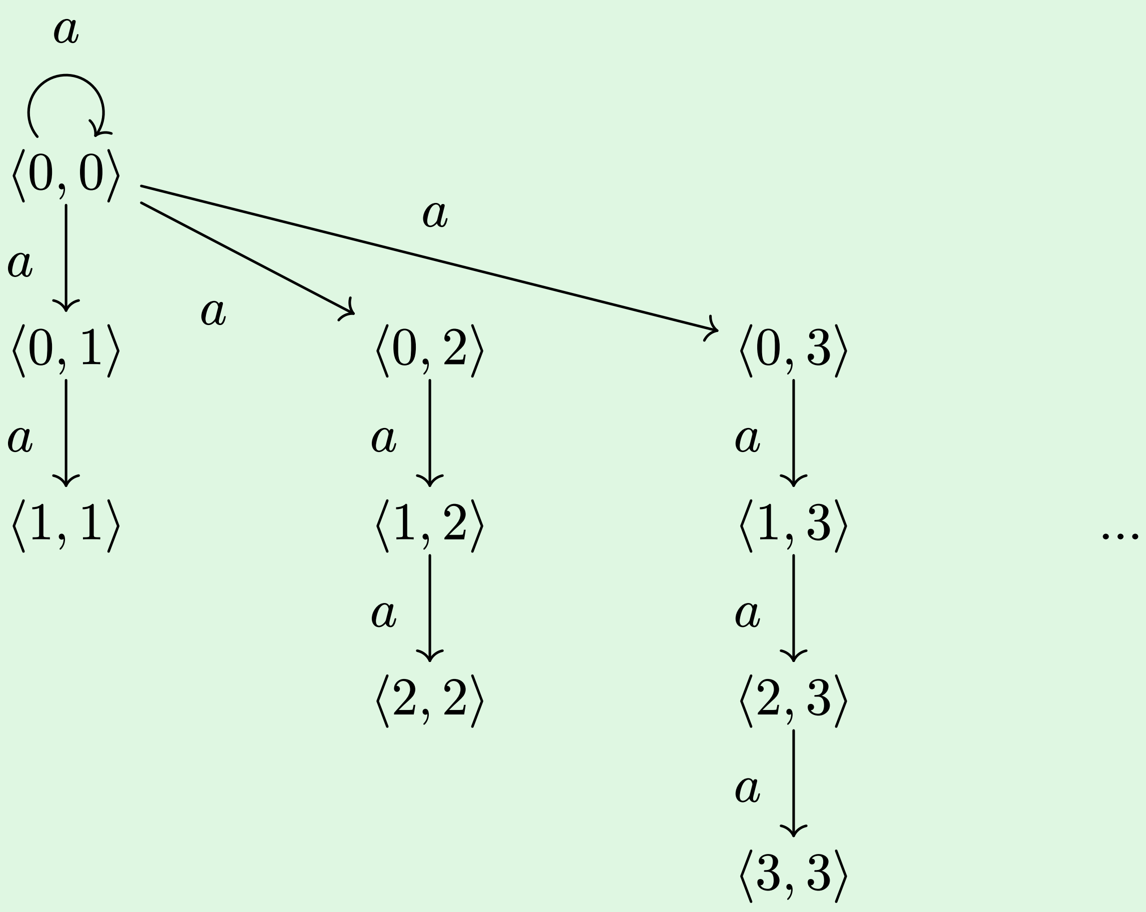
Process Graphs

Example. *Have* $(\mathbb{N} \times \mathbb{N}, \{a\}, \longrightarrow)$ *with*

$$\longrightarrow := \{(\langle 0, 0\rangle, a, \langle 0, n\rangle) \mid n \in \mathbb{N}\} \cup \{(\langle m, n\rangle, a, \langle m+1, n\rangle) \mid m < n\}$$

$G(\langle 0, 0\rangle)$*:*

Process Graphs



Process Equivalence 1.0: Process Graph Isomorphisms

Definition 4 A *process graph* is a directed, rooted, edge-labeled, and possibly infinite graph $G = (V, r, E)$ such that V is its set of *nodes*, $r \in V$ is called the *root of* G ($\mathsf{root}(G) := r$), and $E \subseteq V \times \mathsf{Act} \times V$ is its directed and labeled edge relation.

Definition 6 Let $G = (V_G, r_G, E_G)$ and $H = (V_H, r_H, E_H)$ be process graphs. A *graph isomorphism* between G and H is a bijective function $f : V_G \to V_H$ such that

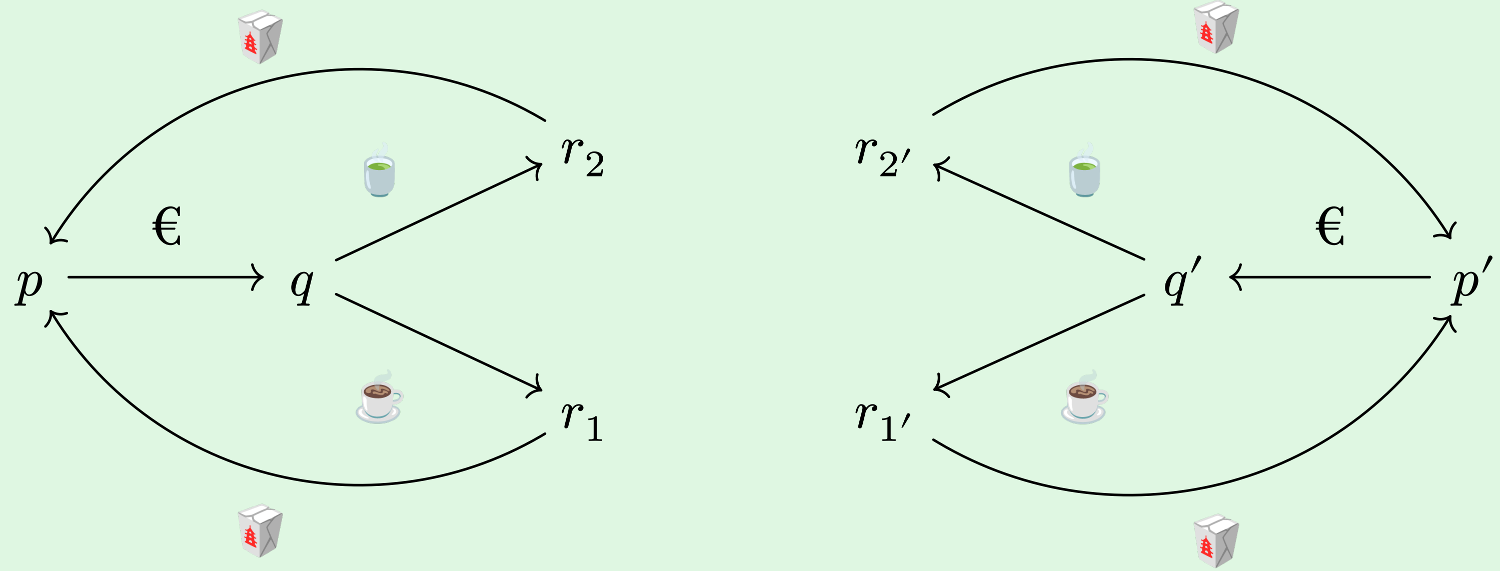
1. $f(r_G) = r_H$ and
2. $(u, a, v) \in E_G$ if and only if $(f(u), a, f(v)) \in E_H$.

Lemma 7 (Exercise ;)) If f is a graph isomorphism, f^{-1} is a graph isomorphism.

Process Equivalence 1.0: Process Graph Isomorphisms

Definition 8 Two processes $p, q \in \mathsf{Pr}$ are *isomorphic*, denoted by $p \leftrightarrow q$, if there is a graph isomorphism between $G(p)$ and $G(q)$.

Example.



Do Isomorphisms Capture Behavior?

Example. *Consider the following processes p and q*

$$\begin{tikzcd} p \arrow[r, "a"] & q \arrow[loop above, "a"] \end{tikzcd}$$

- $p \not\leftrightarrow q$ *because* $G(p)$ *and* $G(q)$ *have different node set cardinalities*
- *systematically, p is a* ***1-time*** *unfolding of the loop in q*
- *compare*

```
while (x != 0) x = x - 1; y = y * 2;
```

to

```
if (x != 0) { x = x - 1; while (x != 0) x = x - 1; } y = y * 2;
```

Process Graphs: Summary

The Good

- allow for graphical representations of LTSs and processes
- inherit useful notions and notations, e.g.,

Definition 9 Let $\mathcal{T} = (\mathsf{Pr}, \mathsf{Act}, \longrightarrow)$ be an LTS and $p \in \mathsf{Pr}$. Define the set of all reachable processes $\mathrm{Reach}_{\mathcal{T}}(p)$ by

1. $p \in \mathrm{Reach}_{\mathcal{T}}(p)$ and
2. if $q \in \mathrm{Reach}_{\mathcal{T}}(p)$ and $q \xrightarrow{a} q'$ for some $a \in \mathsf{Act}$, then $q' \in \mathrm{Reach}_{\mathcal{T}}(p)$.

Process $q \in \mathsf{Pr}$ is reachable from process $p \in \mathsf{Pr}$ if $q \in \mathrm{Reach}_{\mathcal{T}}(p)$.

The Bad

- natural process graph equivalence (i.e., $\leftrightarrow$) is too strong

Exercise. What about graph homomorphisms?

Relations to Automata Theory

LTSs and Nondeterministic Finite Automata (NFAs)

	LTS	**NFA**
notation	$(\mathsf{Pr}, \mathsf{Act}, \longrightarrow)$	$(Q, \Sigma, \delta, Q_0, F)$
set of states	Pr possibly infinite	
set of symbols	Act possibly infinite called actions	
transitions	$\longrightarrow \subseteq \mathsf{Pr} \times \mathsf{Act} \times \mathsf{Pr}$	
where to start?	some process p of interest	
where to end?	**not important at all**	

LTSs and Nondeterministic Finite Automata (NFAs)

	LTS	**NFA**
notation	$(\mathsf{Pr}, \mathsf{Act}, \longrightarrow)$	$(Q, \Sigma, \delta, Q_0, F)$
set of states	Pr possibly infinite	Q always finite
set of symbols	Act possibly infinite called actions	Σ always finite called input alphabet
transitions	$\longrightarrow \subseteq \mathsf{Pr} \times \mathsf{Act} \times \mathsf{Pr}$	$\delta : Q \times \Sigma \to \mathbf{2}^Q$
where to start?	some process p of interest	any $q \in Q_0$
where to end?	**not important at all**	any $q \in F$

The Trace Semantics

Definition 1 (Labeled Transition System) We call a triple $\mathcal{T} = (\mathsf{Pr}, \mathsf{Act}, \longrightarrow)$ a *labeled transition system* (or LTS for short) if Pr is a set of *processes*, Act a set of *actions* such that $\mathsf{Pr} \cap \mathsf{Act} = \emptyset$, and $\longrightarrow \subseteq \mathsf{Pr} \times \mathsf{Act} \times \mathsf{Pr}$ the *labeled transition relation* of $\mathcal{T}$.

- *(partial) traces* are finite sequences of action labels $\sigma \in \mathsf{Act}^\star$
  - write $p \xrightarrow{\varepsilon} p$ for $p \in \mathsf{Pr}$ and $\varepsilon =$ empty word
  - write $p \xrightarrow{a\sigma} q$ for $p, q \in \mathsf{Pr}$
    if there is a $p' \in \mathsf{Pr}$ such that $p \xrightarrow{a} p'$ and $p \xrightarrow{\sigma} p'$ $a \in \mathsf{Act}, \sigma \in \mathsf{Act}^\star$
- an LTS $\mathcal{T}$ can be seen as an NFA such that all states are final $F = \mathsf{Pr}$

- there is a notion closer to that of automata theory
- $p \in \mathsf{Pr}$ is called a *deadlock (process)* if $p \overset{a}{\not\longrightarrow}$ for all $a \in \mathsf{Act}$
- *completed traces* are traces ending in deadlocks $F = \{p \in \mathsf{Pr} \mid p \text{ is a deadlock}\}$

The Trace Semantics

Note

We oftentimes leave the LTS $\mathcal{T}$ implicit and just talk about its processes. Whenever we state that p is a process, we assume there is an LTS in which p *lives*.

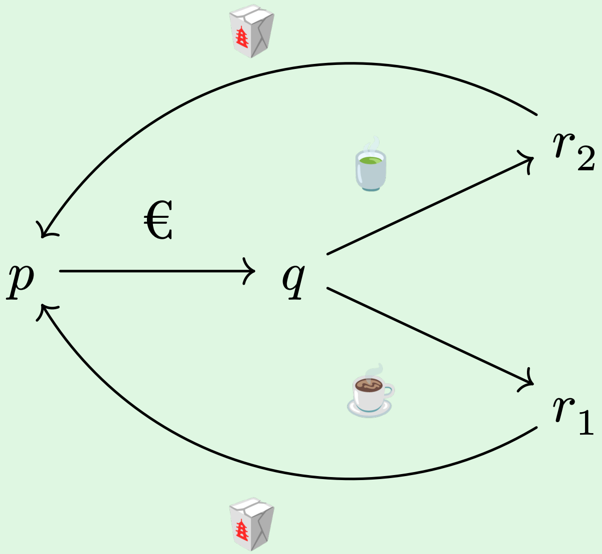
Definition 10 The *set of (partial) traces of* $p \in \mathsf{Pr}$, $\mathsf{traces}(p)$, is defined inductively

- $\varepsilon \in \mathsf{traces}(p)$ and
- if $\sigma \in \mathsf{traces}(q)$ and $p \xrightarrow{a} q$, then $a\sigma \in \mathsf{traces}(p)$.

$p \in \mathsf{Pr}$ is *(partial) trace equialent* to $q \in \mathsf{Pr}$, denoted $p \equiv_{\mathsf{tr}} q$, if $\mathsf{traces}(p) = \mathsf{traces}(q)$.

The Trace Semantics

Example.



$$\mathsf{traces}(p) = \{\varepsilon, \text{€}, \text{€☕}, \text{€🍵}, \text{€☕🥡}, \text{€🍵🥡}, \ldots\}$$

$$\mathsf{traces}(q) = \{\varepsilon, \text{☕}, \text{🍵}, \text{☕🥡}, \text{🍵🥡}, \ldots\}$$

$$\mathsf{traces}(p) = \{\varepsilon\} \cup \{\text{€}\} \circ \mathsf{traces}(q)$$

The Trace Semantics

Example. *Reconsider processes p and q and find that $p \equiv_{\mathsf{tr}} q$*

$$\begin{tikzcd} p \arrow[rr, "a"] & & q \arrow["a", loop above] \end{tikzcd}$$

$$\mathsf{traces}(p) = \{\varepsilon, a, aa, aaa, \dots\} = \mathsf{traces}(q)$$

Process (Equivalence) Relations

Definition 11 Any binary relation $\mathcal{R} \subseteq \mathsf{Pr} \times \mathsf{Pr}$ is called a *process relation*. $\mathcal{R}$ is a *process equivalence* if it is a process relation and an equivalence.

We have seen now two instances of process equivalences.

Theorem 12 $\leftrightarrow$ and $\equiv_{\mathsf{tr}}$ are process equivalences.

Proof: next slide ∎

Throughout the course, we will explore many more process equivalences, each time with a different set of requirements.

Isomorphic equivalence ($\leftrightarrow$) and trace equivalence ($\equiv_{\mathsf{tr}}$) form meaninful boundaries.

Trivial boundaries: $\mathcal{U} = \mathsf{Pr} \times \mathsf{Pr}$ (the *universal equivalence*) and $\emptyset$ (the *non-equivalence*).

A Proof of Theorem 12

Theorem 12 $\leftrightarrow$ and $\equiv_{\mathsf{tr}}$ are process equivalences.

Proof: For all processes $p, q, r \in \mathsf{Pr}$,

1. $p \leftrightarrow p$ by $\mathrm{id} : \mathsf{Pr} \to \mathsf{Pr}$ ($\mathrm{id}(q) = q$ for all $q \in \mathsf{Pr}$) being an isomorphism.
2. $p \leftrightarrow q$ implies $q \leftrightarrow p$ since the inverse f^{-1} of an isomorphism f is an isomorphism (cf. Lemma 7).
3. $p \leftrightarrow q$ and $q \leftrightarrow r$ implies $p \leftrightarrow r$ since isomorphisms f and g compose to an isomorphism $g \circ f$.

For all processes $p, q, r \in \mathsf{Pr}$,

1. $p \equiv_{\mathsf{tr}} p$ as set equality is reflexive.
2. $p \equiv_{\mathsf{tr}} q$ implies $q \equiv_{\mathsf{tr}} p$ since set equality is symmetric.
3. $p \equiv_{\mathsf{tr}} q$ and $q \equiv_{\mathsf{tr}} r$ implies $p \equiv_{\mathsf{tr}} r$ since set equality is transitive.

∎

Towards a Spectrum of Process Equivalences

Theorem 13

$$\emptyset \overset{(1)}{\subsetneq} \leftrightarrow \overset{(2)}{\subsetneq} \equiv_{\mathsf{tr}} \overset{(3)}{\subsetneq} \mathcal{U} = \mathsf{Pr} \times \mathsf{Pr}$$

Proof: Parts (1) and (3) are clear. Proper inclusions stem from the examples we have seen.

Regarding (2), let $p, q \in \mathsf{Pr}$ such that $p \leftrightarrow q$. Then there is an isomorphism f between the graphs $G(p)$ and $G(q)$, meaning

1. $f(p) = q$ (since p and q are the roots of their respective process graphs) and
2. $p_1 \xrightarrow{a} p_2$ (and $p_1 \in \mathrm{Reach}(p)$) if and only if $f(p_1) \xrightarrow{a} f(p_2)$ ($f(p_1) \in \mathrm{Reach}(q)$)

Towards a Spectrum of Process Equivalences

For every trace $\sigma = a_1 a_2 \ldots a_n \in \mathsf{Act}^\star$,

$$
\begin{aligned}
\sigma \in \mathsf{traces}(p) \ &\text{iff } \exists p_1, \ldots, p_n \in \mathsf{Pr}\ .p \xrightarrow{a_1} p_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} p_n && \text{(by definition)} \\
&\text{iff } f(p) \xrightarrow{a_1} f(p_1) \xrightarrow{a_2} \cdots \xrightarrow{a_n} f(p_n) && (f \text{ is an isomorphism}) \\
&\text{iff } \exists q_1, \ldots, q_n \in \mathsf{Pr}\ .q \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n && (\text{take } q_1 = f(p_1) \ldots q_n = f(p_n)) \\
&\text{iff } \sigma \in \mathsf{traces}(q) && \text{(by definition)}
\end{aligned}
$$

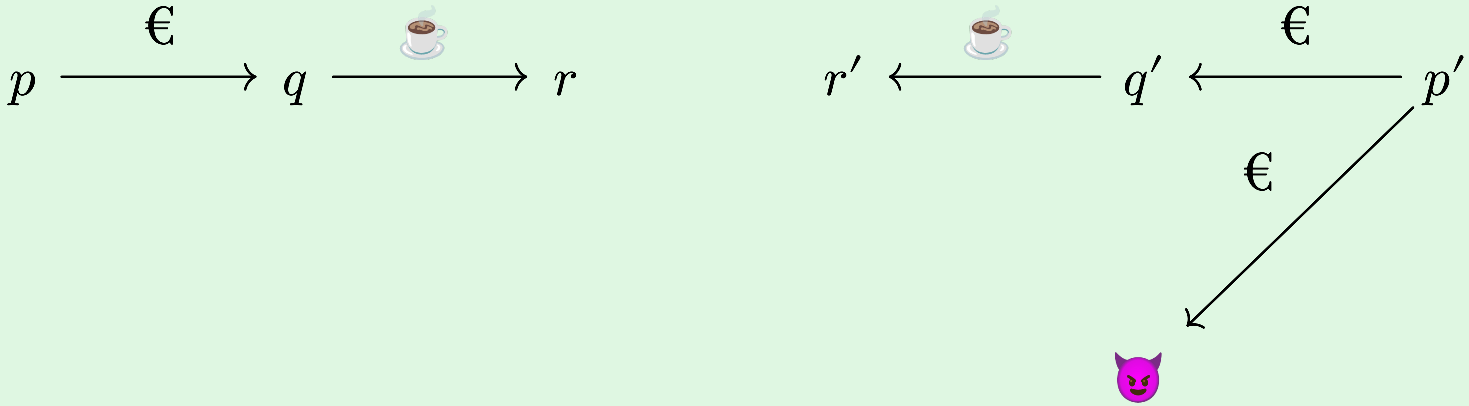
For $\leftrightarrow \neq \equiv_{\mathsf{tr}}$, reconsider p and q below, having $p \equiv_{\mathsf{tr}} q$ but $p \not\leftrightarrow q$.

$$
\begin{tikzcd}
p \arrow[rr, "a"] & & q \arrow[loop above, "a"]
\end{tikzcd}
$$

$\blacksquare$

Trace Equivalence: End of Story?

Example.



$$\mathsf{traces}(p) = \{\varepsilon, €, €\text{☕}\} = \{\varepsilon, €, €, €\text{☕}\} = \mathsf{traces}(p')$$

There is one trace, namely €, that is a completed trace of p' but not of p.

In other words, trace equivalence (i.e., $\equiv_{\mathsf{tr}}$) is **not** sensitive to deadlocks.

The Completed Trace Semantics

Definition 14 A process $p \in \mathsf{Pr}$ is a *deadlock* if $p \overset{a}{\not\longrightarrow}$ for all $a \in \mathsf{Act}$.

The set of *completed traces* of a process $p \in \mathsf{Pr}$, denoted by $\text{traces}_c(p)$ is the set of all traces $\sigma \in \mathsf{ctraces}(p)$ such that $p \xrightarrow{\sigma} q$ and q is a deadlock.

Processes $p, q \in \mathsf{Pr}$ are *completed trace equivalent*, denoted by $p \equiv_{\mathsf{ctr}} q$, if $p \equiv_{\mathsf{tr}} q$ and $\mathsf{ctraces}(p) = \mathsf{ctraces}(q)$.

Theorem 15

$$\leftrightarrow \quad \overset{(1)}{\subsetneq} \quad \equiv_{\mathsf{ctr}} \quad \overset{(2)}{\subsetneq} \quad \equiv_{\mathsf{tr}}$$

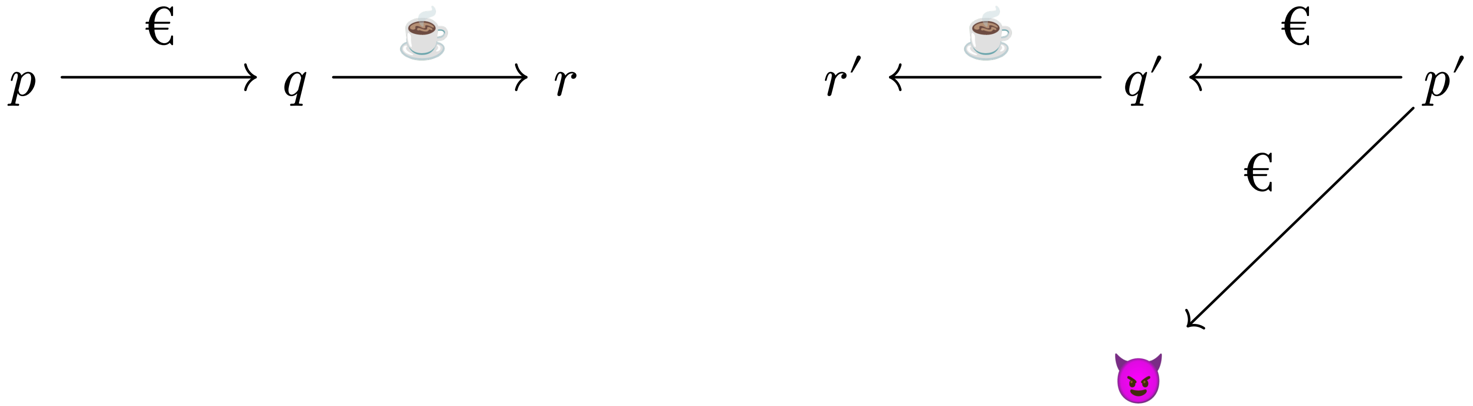
Proof of Theorem 15

Theorem 15

$$\leftrightarrow \quad \overset{(1)}{\subsetneq} \quad \equiv_{\mathsf{ctr}} \quad \overset{(2)}{\subsetneq} \quad \equiv_{\mathsf{tr}}$$

Regarding (2),

- observe that trace equivalence is part of the definition of $\equiv_{\mathsf{ctr}}$;
- furthermore, 😈 serves as a counterexample, proving $\equiv_{\mathsf{ctr}} \neq \equiv_{\mathsf{tr}}$.



Proof of Theorem 15

Towards (1),

- observe that a deadlock process $p \in \mathsf{Pr}$ can only be isomorphic to other deadlock processes;
- in fact, all deadlock processes are isomorphic;
- hence, any completed trace of $p \in \mathsf{Pr}$ must be a a completed trace of $f(p)$ (by the same arguments as in proof of Theorem 13);
- also, $\leftrightarrow \neq \equiv_{\mathsf{ctr}}$ (e.g., p_0 and q_0 below).

$$\begin{tikzcd}
 & & & & p_1 \\
q_1 & q_0 \arrow[l, "a"'] & \overset{\equiv_{\mathsf{ctr}}}{\not\leftrightarrow} & p_0 \arrow[ur, "a"] \arrow[dr, "a"'] & \\
 & & & & p_2
\end{tikzcd}$$

∎

Completed Traces: End of Story?

Definition 14 A process $p \in \mathsf{Pr}$ is a *deadlock* if $p \not\xrightarrow{a}$ for all $a \in \mathsf{Act}$.

The set of *completed traces* of a process $p \in \mathsf{Pr}$, denoted by $\text{traces}_c(p)$ is the set of all traces $\sigma \in \mathsf{ctraces}(p)$ such that $p \xrightarrow{\sigma} q$ and q is a deadlock.

Processes $p, q \in \mathsf{Pr}$ are *completed trace equivalent*, denoted by $p \equiv_{\mathsf{ctr}} q$, if $p \equiv_{\mathsf{tr}} q$ and $\mathsf{ctraces}(p) = \mathsf{ctraces}(q)$.

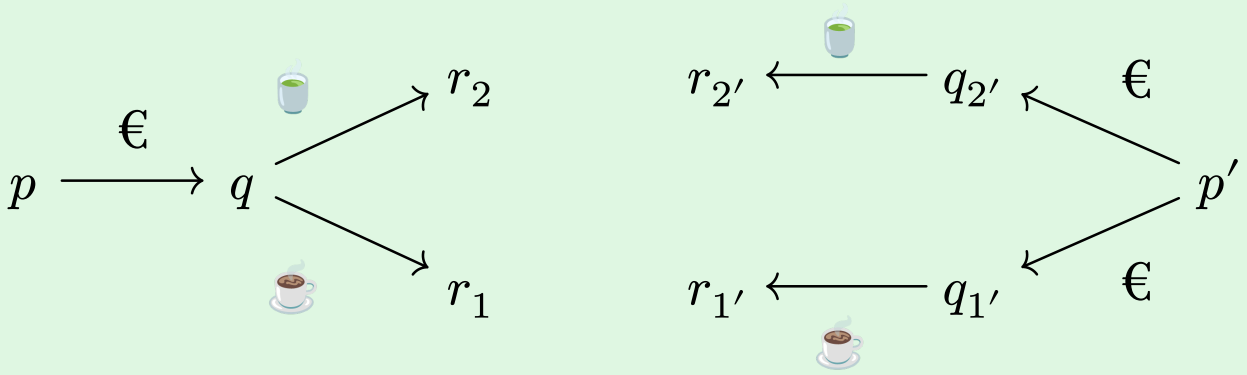
Theorem 15

$$\leftrightarrow \quad \overset{(1)}{\subsetneq} \quad \equiv_{\mathsf{ctr}} \quad \overset{(2)}{\subsetneq} \quad \equiv_{\mathsf{tr}}$$

$\equiv_{\mathsf{ctr}}$ preserves traces (2) and deadlocks (😈)

Completed Traces are Insensitive for Nondeterminism

Example.



Outlook

1. We are looking for the intimate connection between nondeterminism and interaction.
2. We are aiming at equivalences going beyond *linear time*.
3. **Resolution:** *branching time* and, more specifically, *bisimilarity*