Modeling & Verification

Strong Bisimilarity

Max Tschaikowski (tschaikowski@cs.aau.dk)
Slides courtesy of Giorgio Bacci

in the last Lecture

- CCS - the basic principles & motivations
- Examples
- CCS - formal definition (syntax & semantics)

in this Lecture

- Value-passing CCS
- Behavioural Equivalences (idea & motivations)
- Strong Bisimilarity
- Game characterisation of Bisimilarity

Value-passing CCS

Main Idea

In pure CCS communication is just synchronisation with no exchange of data. In many application, however, processes do exchange data when communicate.

To allow for the natural modelling of data exchange we extend the CCS language with two prefixing operations:

Input Prefix

$a(x).P$

x is a variable

Output Prefix

$\bar{a}(e).P$

e is an expression

Example

Defining Equations

$$\mathrm{Job}(\mathrm{amount}) \stackrel{\mathrm{def}}{=} \overline{\mathrm{pay}}(\mathrm{amount}).0$$

$$\mathrm{Worker} \stackrel{\mathrm{def}}{=} \mathrm{pay}(x).\overline{\mathrm{save}}(x/2).\mathrm{Worker}$$

$$\mathrm{Bank}(\mathrm{total}) \stackrel{\mathrm{def}}{=} \mathrm{save}(y).\mathrm{Bank}(\mathrm{total} + y)$$

$$\begin{array}{c}
\mathrm{Job}(6) \mid \mathrm{Worker} \mid \mathrm{Bank}(100) \\
\big\downarrow \tau \\
0 \mid \overline{\mathrm{save}}(3).\mathrm{Worker} \mid \mathrm{Bank}(100) \\
\big\downarrow \tau \\
0 \mid \mathrm{Worker} \mid \mathrm{Bank}(103)
\end{array}$$

Formal Semantics(*)

we have two new rules for the prefixing operators

(IN) $$\frac{}{a(x).P \xrightarrow{a(n)} P\{n/x\}}$$ *where $n \geq 0$*

substitution of all occurrences x with n

(OUT) $$\frac{}{\overline{a}(e).P \xrightarrow{\overline{a}(n)} P}$$ *where $v(e)=n$*

expressions need to be evaluated!

and one rule to deal with parametrised constants

(CON) $$\frac{P\{n_1/x_1,\ldots,n_m/x_m\} \xrightarrow{\alpha} P'}{K(e_1,\ldots,e_m) \xrightarrow{\alpha} P'}$$ *where $K(x_1,\ldots,x_m) \stackrel{def}{=} P$ and for all $1 \leq i \leq m$, $v(e_i)= n_i$*

(*) This only works for CCS expressions with no free occurrences of value variables

Encoding into pure CCS

As argued by Milner (1989), value-passing CCS is theoretically unnecessary: one can encode it into pure CCS

value-passing CCS

$$P \stackrel{\text{def}}{=} in(x).Q(x)$$

$$Q(x) \stackrel{\text{def}}{=} \overline{out}(x).P$$

pure CCS

$$P \stackrel{\text{def}}{=} \sum_{i \geq 0} in_i.Q_i$$

$$Q_i \stackrel{\text{def}}{=} \overline{out}_i.P$$

we need an infinite sum of processes

Expressivity of CCS

Fact

In CCS one can encode and simulate the computation of any Turing machine.

Hence, the CCS language is as expressive as any other programming language.

However, it is mainly used to describe the behaviour of reactive systems rather than to perform specific calculation

Behavioural Equivalences

idea & motivations

Implementation vs Specification

CCS can be used to describe both implementations of processes and specifications of their expected behaviours

Implementation

$CS \stackrel{\text{def}}{=} \overline{\text{pub}}.\overline{\text{coin}}.\text{coffee}.CS$

$CM \stackrel{\text{def}}{=} \text{coin}.\overline{\text{coffee}}.CM$

$Sys \stackrel{\text{def}}{=} (CM \mid CS)\backslash\{\text{coin},\text{coffee}\}$

Specification

$Spec \stackrel{\text{def}}{=} \overline{\text{pub}}.Spec$

Question: is the process **Sys** behaving according to the specification **Spec**?

Implementation Verification

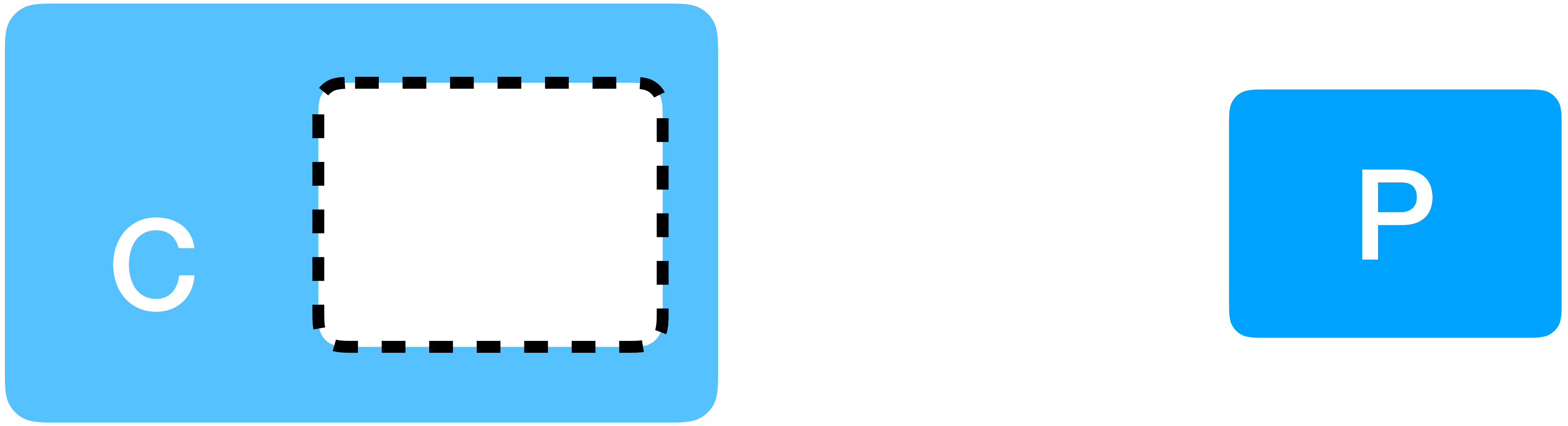
One way to check that **Sys** behaves according to **Spec** is to check that they are *behavioural equivalent*

Criteria for a good behavioural equivalence

- it should be based only on the behaviour (i.e., by looking only at the available actions)
- abstract from non-determinism
- it should not consider internal behaviour
- it should be at least a *preorder*

  reflexive and transitive
- it should be a *congruence*

Congruence



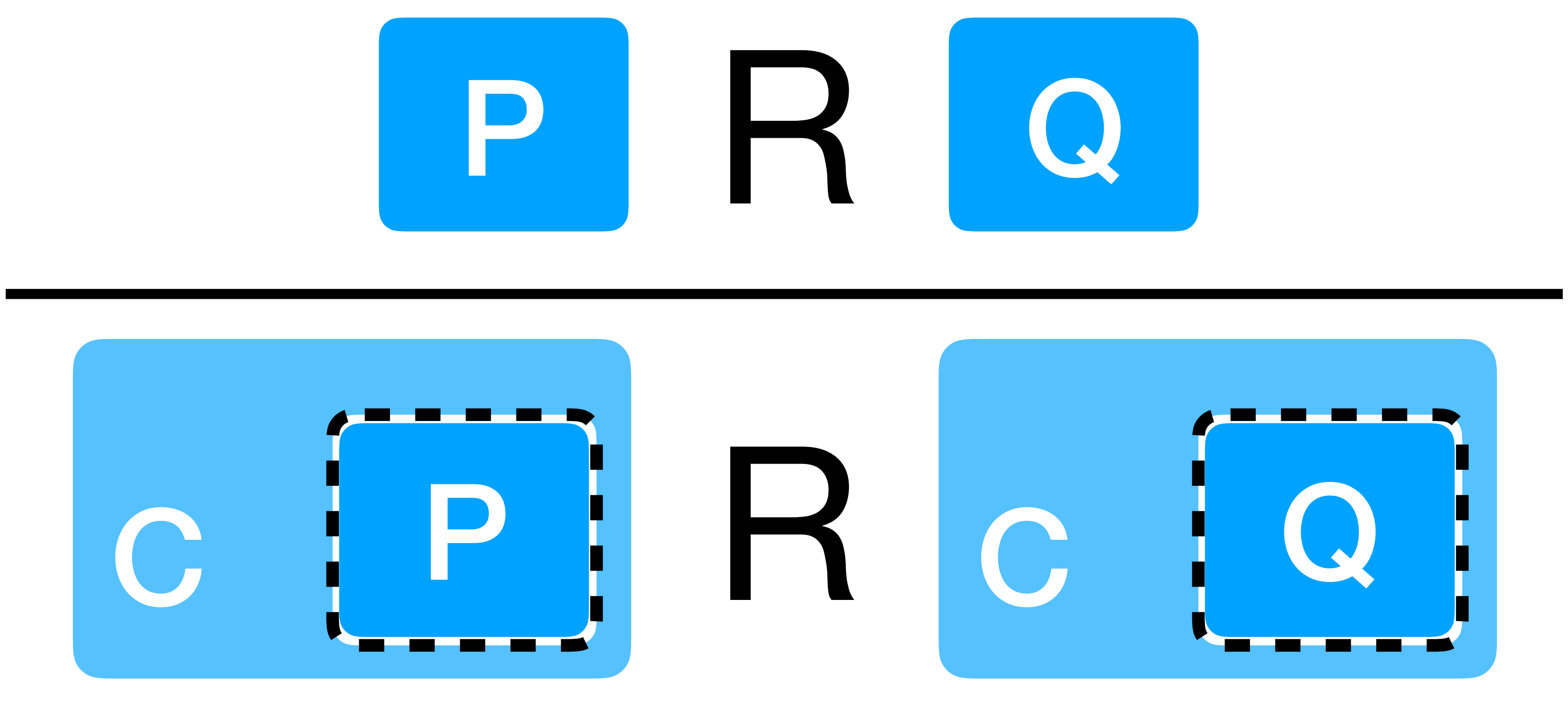
C[P]

A context is a CCS program fragment with a "hole"

Congruence

Definition

A relation R ⊆ $\mathcal{P}\times\mathcal{P}$ is said to be a *congruence* if, for each context C[ ], P R Q implies C[P] R C[Q].



Trace Equivalence

a first attempt...

The semantic of processes is given as LTS, which are essentially automata. The classic theory of automata suggests a ready-made notion of equivalence:

Definition (Trace Equivalence)

Let (Proc, Act, $\{\xrightarrow{\alpha} \mid \alpha \in \text{Act}\}$) be an LTS.

For each s∈Proc, define

$$\text{Trace}(s) = \{w \in \text{Act}^* \mid \exists s' \in \text{Proc}.\ \ s \xrightarrow{w} s'\}\ .$$

We say that s,s'∈ Proc are *trace equivalent*, written $s \equiv_t s'$, if and only if, Trace(s) = Trace(s').

Trace Equivalence (example)

Consider the processes

$$\mathrm{CTM} \stackrel{\mathrm{def}}{=} \mathrm{coin}.(\overline{\mathrm{coffee}}.\mathrm{CTM} + \overline{\mathrm{tea}}.\mathrm{CTM})$$

$$\mathrm{CTM}' \stackrel{\mathrm{def}}{=} \mathrm{coin}.\overline{\mathrm{coffee}}.\mathrm{CTM}' + \mathrm{coin}.\overline{\mathrm{tea}}.\mathrm{CTM}'$$

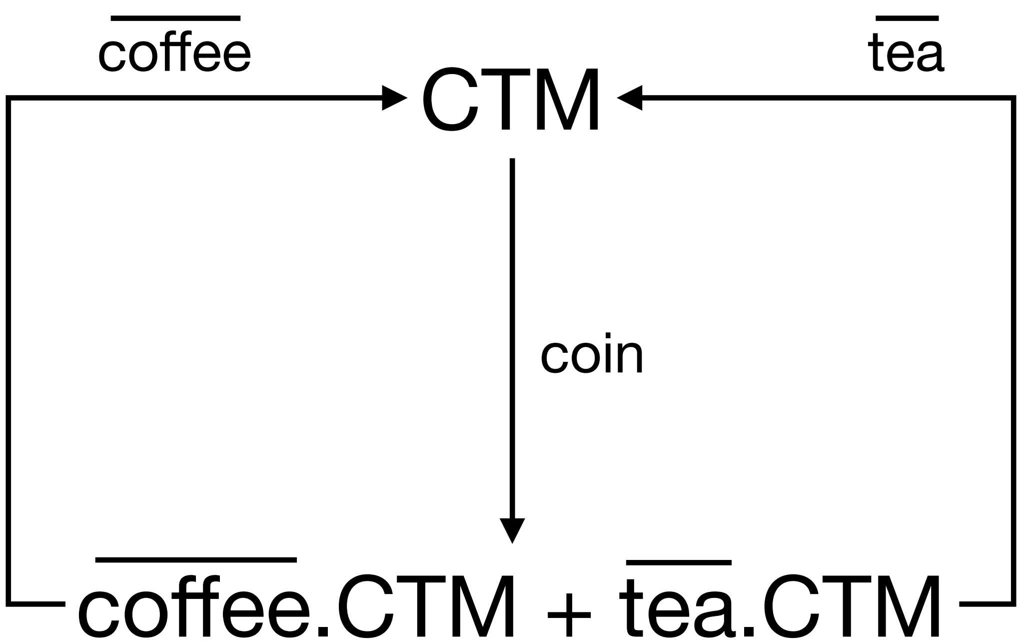
prove it!

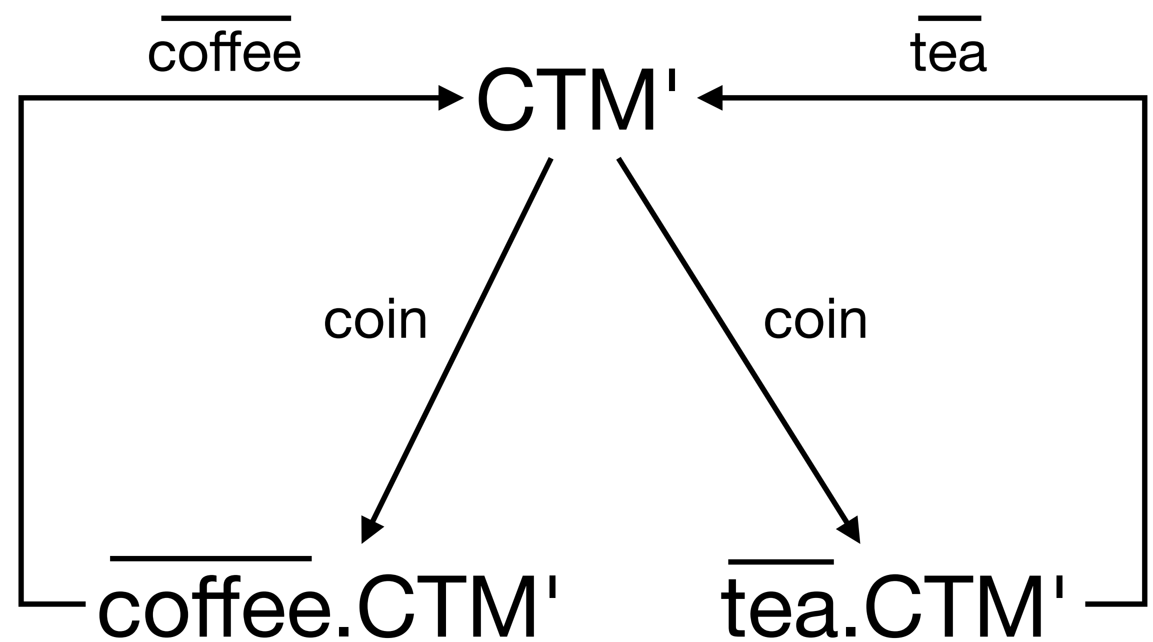
It is easy to see that Trace(CTM) = Trace(CTM'), hence the processes CTM and CTM' are trace equivalent.

Is it enough to say that CTM and CTM' behave the same?

Observable Capabilities

Let's try to perform a black-box experiment
(we are only allowed to interact with observable actions)





Experiment

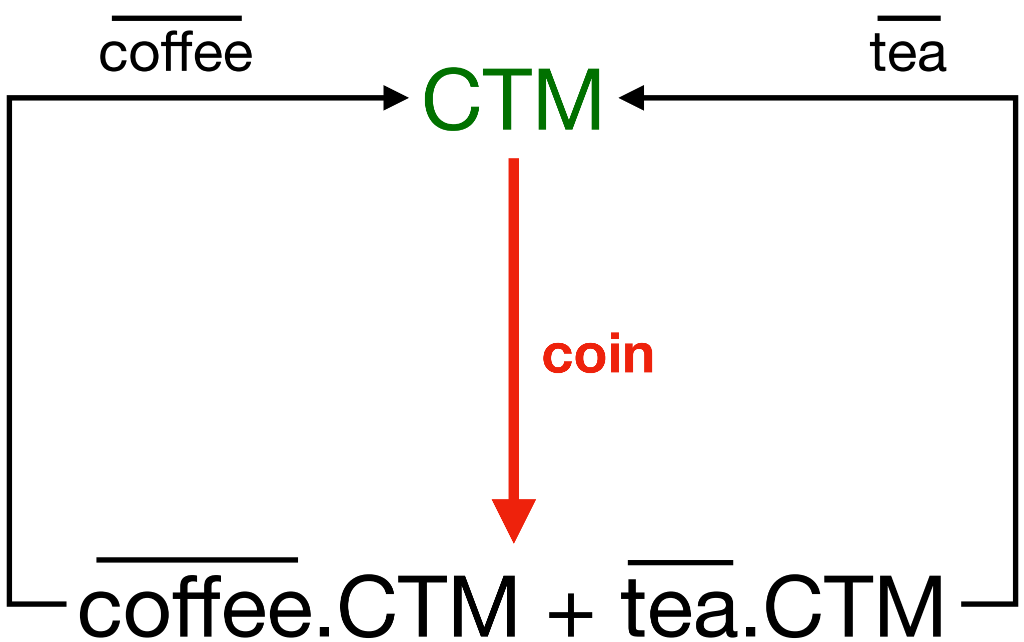
press: ...
observe: ...

Experiment

press: ...
observe: ...

Observable Capabilities

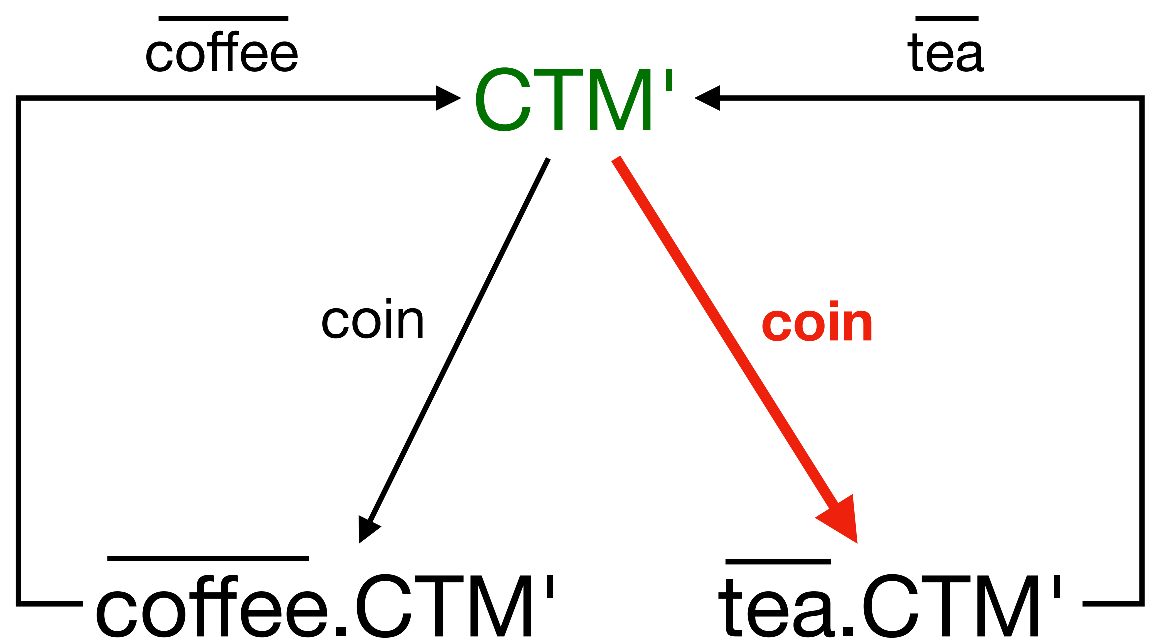
Let's try to perform a black-box experiment
(we are only allowed to interact with observable actions)



Experiment

press: $\overline{\text{coin}}$

observe: τ



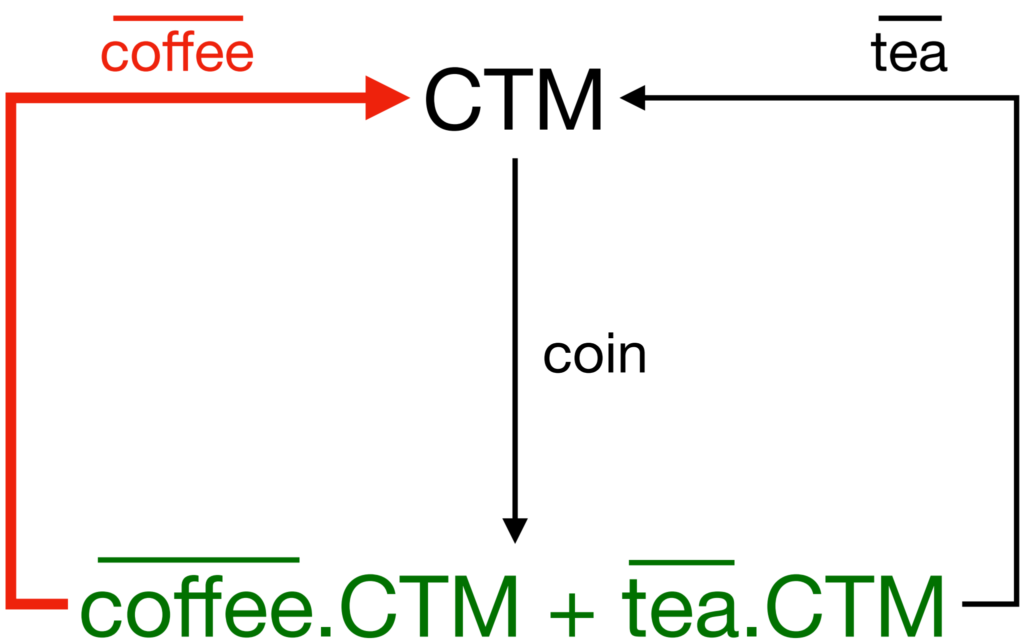
Experiment

press: $\overline{\text{coin}}$

observe: τ

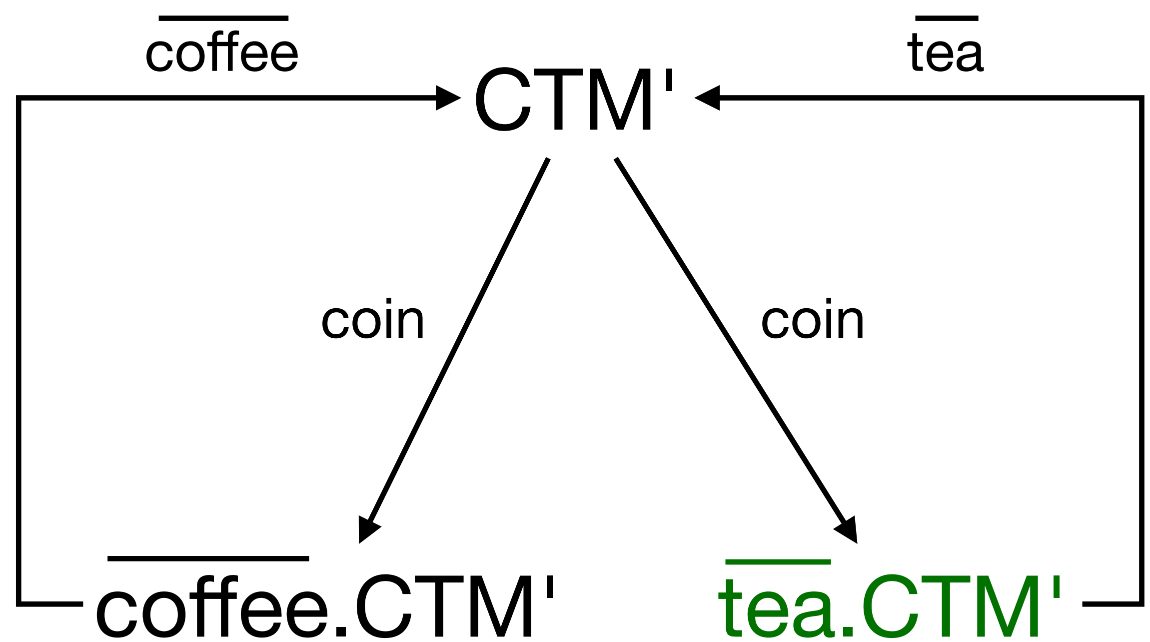
Observable Capabilities

Let's try to perform a black-box experiment
(we are only allowed to interact with observable actions)



Experiment

press: coffee
observe: τ



Experiment

press: coffee
observe: ?

Strong Bisimilarity

idea: if no external observer can tell two processes apart break?

Strong Bisimilarity

Let (Proc, Act, $\{\xrightarrow{\alpha} \mid \alpha \in Act\}$) be an LTS.

Definition (Strong Bisimulation)

A binary relation $R \subseteq Proc \times Proc$ is a *strong bisimulation* iff whenever s R t, for each $\alpha \in Act$

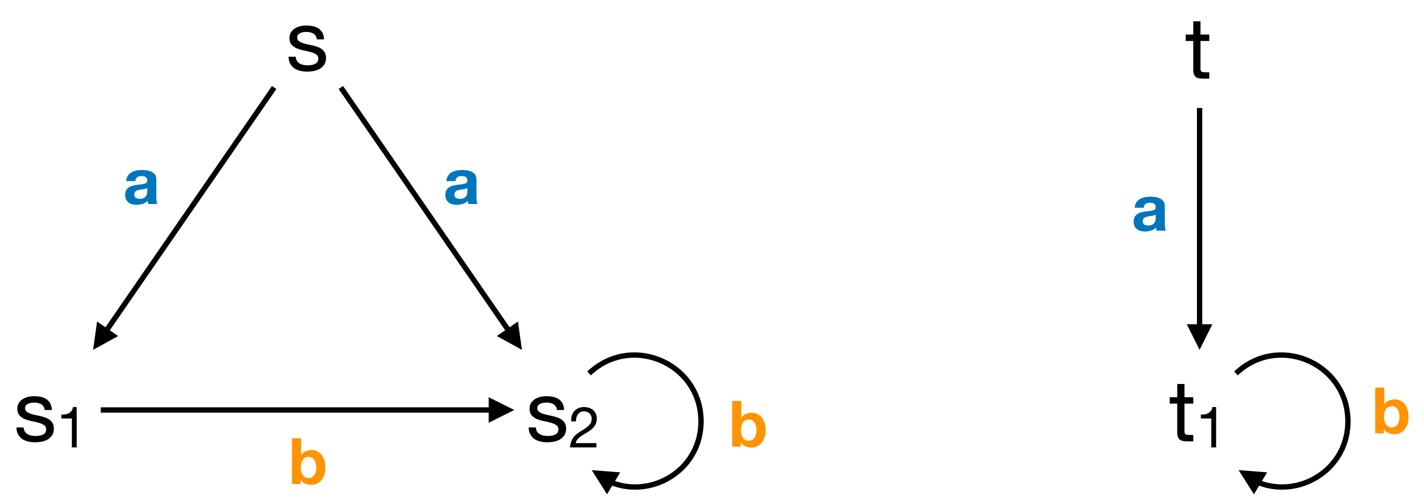
- if $s \xrightarrow{\alpha} s'$, then $t \xrightarrow{\alpha} t'$, for some t' such that s'R t'
- if $t \xrightarrow{\alpha} t'$, then $s \xrightarrow{\alpha} s'$, for some s' such that s'R t'

Definition (Strong Bisimilarity)

Two states $s,t \in Proc$ are *strongly bisimilar* ($s \sim t$) iff there exists a strong bisimulation R such that s R t.

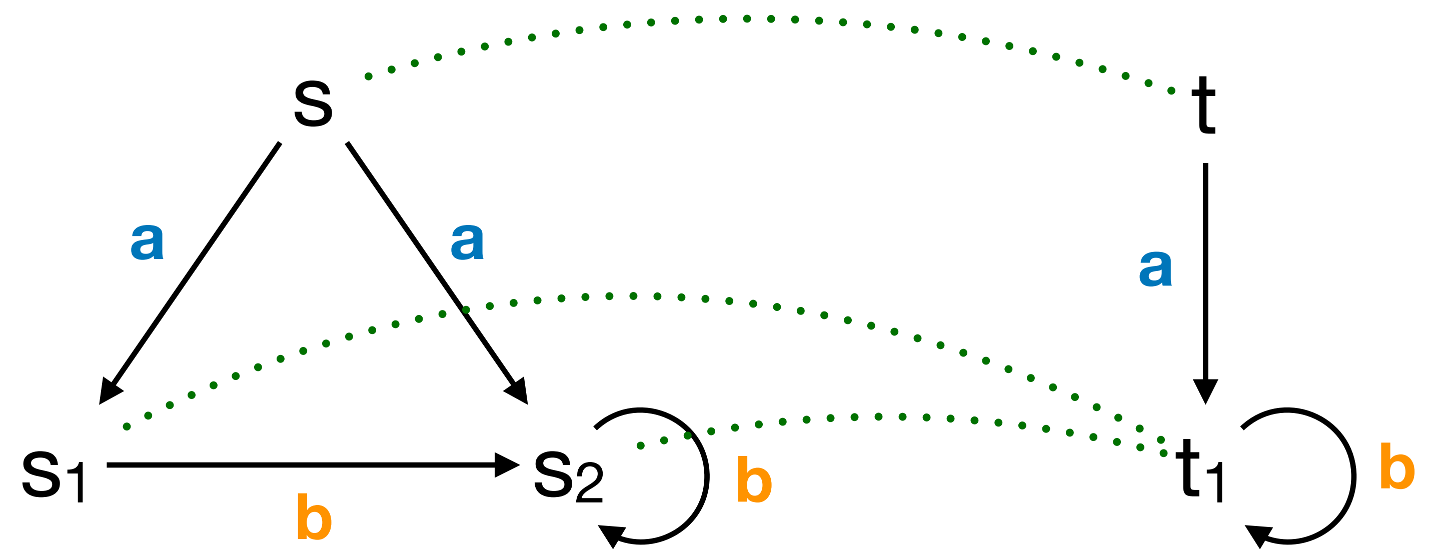
$$\sim \; = \bigcup \{ R \mid R \text{ is a strong bisimulation} \}$$

Example



To show that s ~ t, we just need to exhibit a strong bisimulation R such that s R t.

Example



To show that s ~ t, we need to find a strong bisimulation R such that s R t.

$$R = \{(s,t), (s_1,t_1), (s_2, t_1)\}$$

one needs to show that it satisfies the conditions of strong bisimulation

Basic Properties

Theorem

$\sim$ is an *equivalence* (reflexive, symmetric, and transitive)

Theorem

$\sim$ is the *largest strong bisimulation*, that is, $s \sim t$, if and only if, for each $\alpha \in Act$

- if $s \xrightarrow{\alpha} s'$, then $t \xrightarrow{\alpha} t'$, for some t' such that $s' \sim t'$
- if $t \xrightarrow{\alpha} t'$, then $s \xrightarrow{\alpha} s'$, for some s' such that $s' \sim t'$

Bisimilarity & CCS

Clearly the definition of bisimilarity applies also to CCS processes by means of its LTS semantics

Theorem

Let P and Q be CCS processes such that $P \sim Q$. Then

- $\alpha.P \sim \alpha.Q$, for each $\alpha \in Act$
- $P+R \sim Q+R$ and $R+P \sim R+Q$, for each CCS process R
- $P|R \sim Q|R$ and $R|P \sim R|Q$, for each CCS process R
- $P[f] \sim Q[f]$, for each relabelling function f
- $P\backslash L \sim Q\backslash L$, for each set of labels $L \subseteq \mathbb{A}$

hence, bisimilarity is a congruence for CCS operations

Bisimilarity & CCS

Theorem

For any P, Q, and R CCS processes, the following hold

- $P+Q \sim Q+P$ (*nondeterministic choice is symmetric*)
- $P|Q \sim Q|P$ (*parallel composition is symmetric*)
- $P+0 \sim P$ (*0 is null element for nondeterministic choice*)
- $P|0 \sim P$ (*0 is null element for parallel composition*)
- $(P+Q)+R \sim P+(Q+R)$ (*nondet. choice is associative*)
- $(P|Q)|R \sim P|(Q|R)$ (*parallel composition is associative*)

Game characterisation of Bisimilarity

What about non-bisimilarity?

It is natural to asks if there are techniques to show that two states are *not* bisimilar

How to prove $s \not\sim t$

- Enumerate all binary relations and show that none of them is a strong bisimulation containing (s,t). (**Expansive**: 2^{n^2} relations, for an LTS with n states)
- Make certain observations which will enable to disqualify many bisimulation candidates in one step
- Use **game characterisation** of strong bisimilarity.

General and economic

Strong Bisimulation Game

Let (Proc, Act, $\{\xrightarrow{\alpha} \mid \alpha \in \text{Act}\}$) be an LTS, and s,t∈Proc.

We define a two-player game of an **'attacker'** and a **'defender'** starting from s and t

- The game is played in *rounds* and the *configurations* of the game are pairs of states from Proc×Proc.
- In every round, exactly one configuration is called *current*.
- Initially, the game has (s,t) as *starting (current) configuration*.

Intuition

The **defender** wants to show that s and t are strongly bisimilar, while the **attacker** aims to prove the opposite.

The Rules of the Game

Rounds of Bisimulation Games

Each round goes as follows:

1. The **attacker** chooses one of the processes in the current configuration and makes a $\xrightarrow{\alpha}$-move for some $\alpha \in Act$;
2. the **defender** must respond with a matching $\xrightarrow{\alpha}$-move with same action α in the other process.

The reached pair of processes becomes the next current configuration. The game then continues by another round.

Winning Conditions

- If one player cannot move, the other player wins;
- If the game is infinite, the **defender** wins.

The Game Characterisation

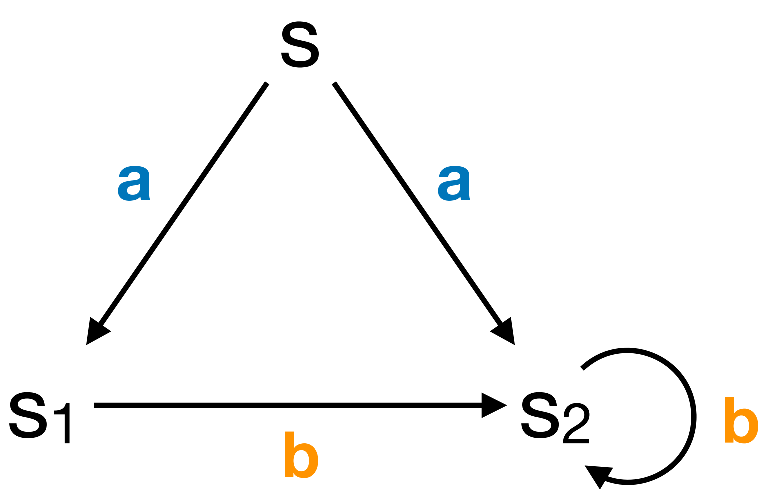
A *universal strategy* is a set of moves (possibly conditional) describing what a player does in each configuration.

Theorem

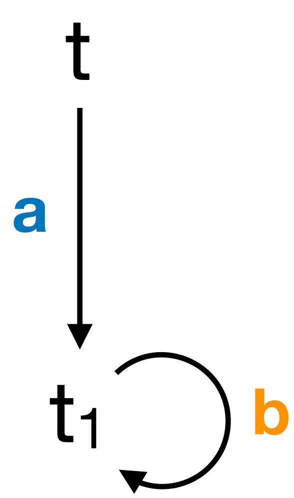
- The states s and t are strongly bisimilar iff the **defender** has a universal winning strategy starting from the configuration (s,t).
- The states s and t are *not* strongly bisimilar iff the **attacker** has a universal winning strategy starting from the configuration (s,t).

It often provides elegant arguments for proving the negative case

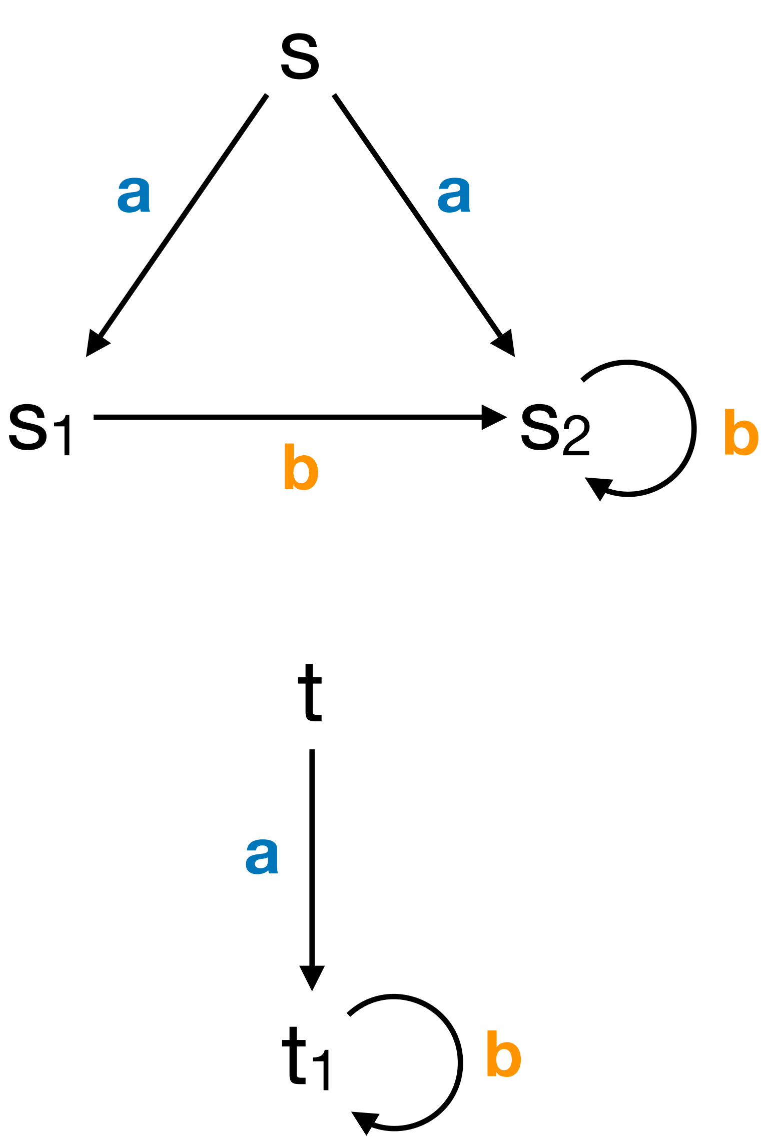
Example 1



To show that s ~ t, we provide a universal winning strategy for the **defender**, starting from the configuration (s,t)



Example 1

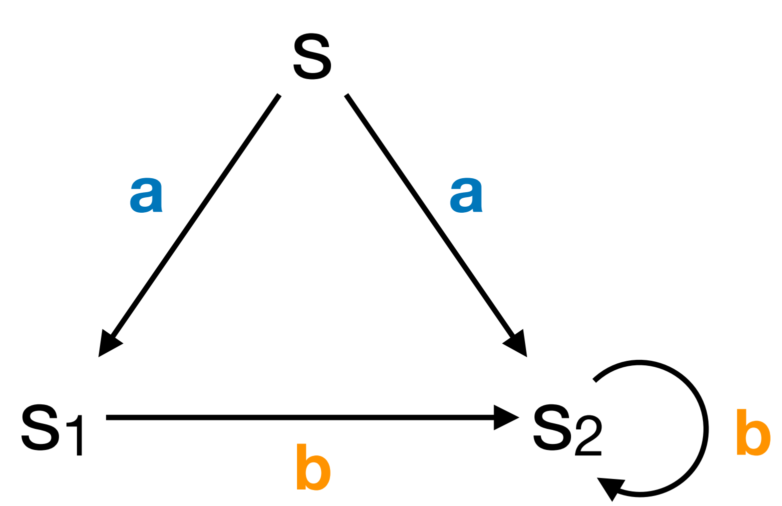


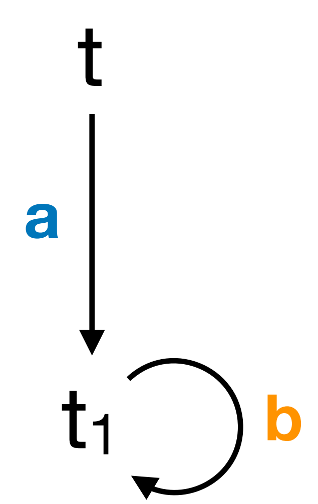
From (s,t)

1. if **attacker** makes the move $t \xrightarrow{a} t_1$, then **defender** reacts with $s \xrightarrow{a} s_2$
2. if **attacker** makes the move $s \xrightarrow{a} s_1$, then **defender** reacts with $t \xrightarrow{a} t_1$
3. if **attacker** makes the move $s \xrightarrow{a} s_2$, then **defender** reacts with $t \xrightarrow{a} t_1$

Hence from (s,t) there are two possible next configurations: (s_1,t_1) and (s_2, t_1).

Example 1





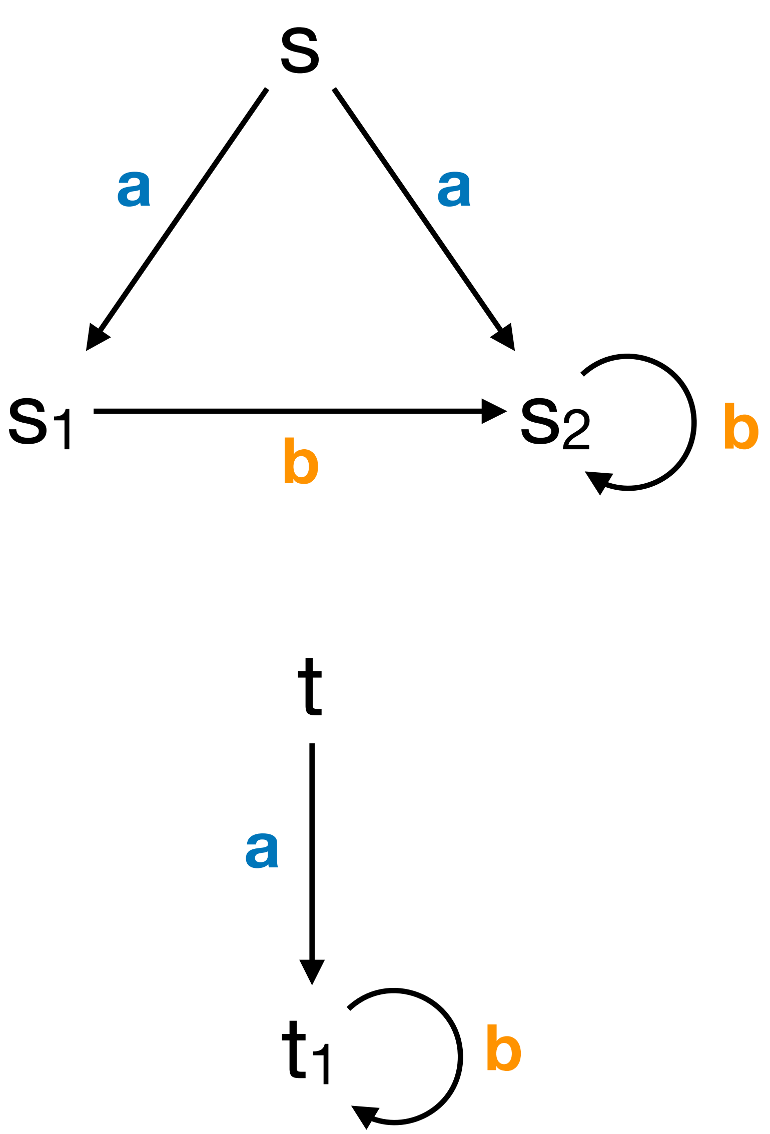
From (s_2,t_1)

1. if **attacker** makes the move $s_2 \xrightarrow{b} s_2$, then **defender** reacts with $t_1 \xrightarrow{b} t_1$
2. if **attacker** makes the move $t_1 \xrightarrow{b} t_1$, then **defender** reacts with $s_2 \xrightarrow{b} s_2$

Hence from (s_2,t_1) we can only reach the configuration (s_2,t_1).

defender wins with an infinite game

Example 1



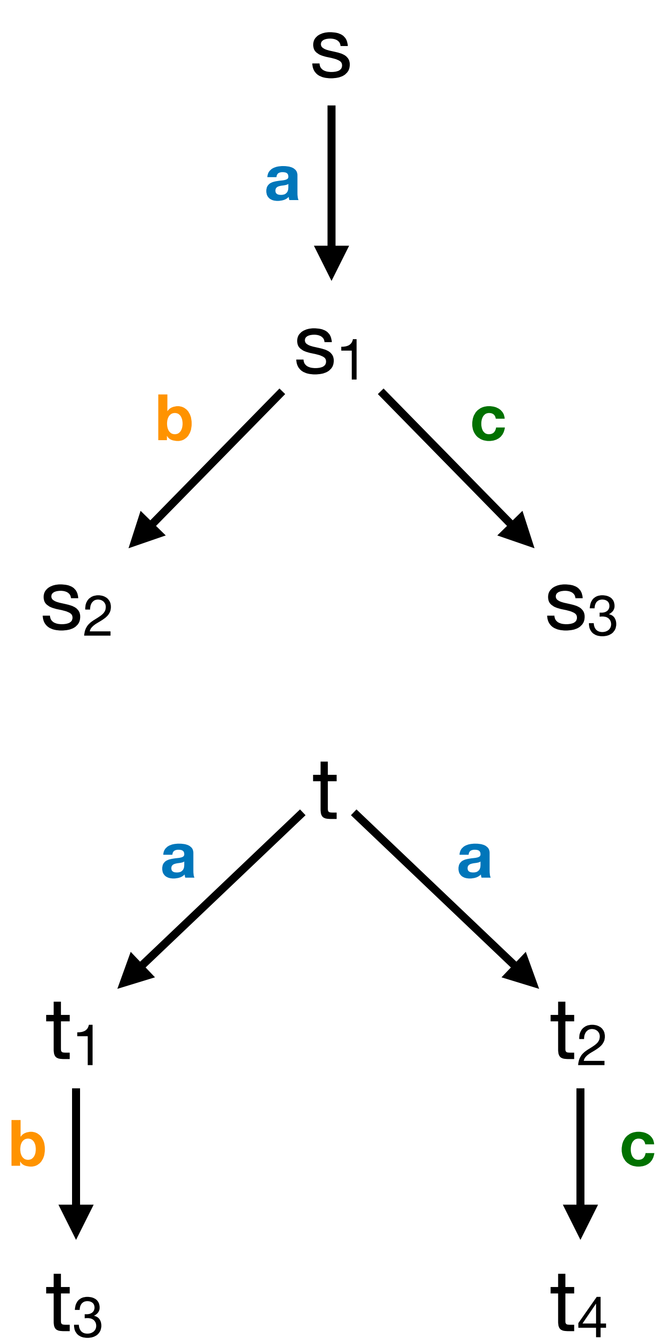
From (s_1,t_1)

1. if **attacker** makes the move $s_1 \xrightarrow{b} s_2$, then **defender** reacts with $t_1 \xrightarrow{b} t_1$
2. if **attacker** makes the move $t_1 \xrightarrow{b} t_1$, then **defender** reacts with $s_1 \xrightarrow{b} s_2$

Hence from (s_1,t_1) we can only reach the configuration (s_2,t_1).

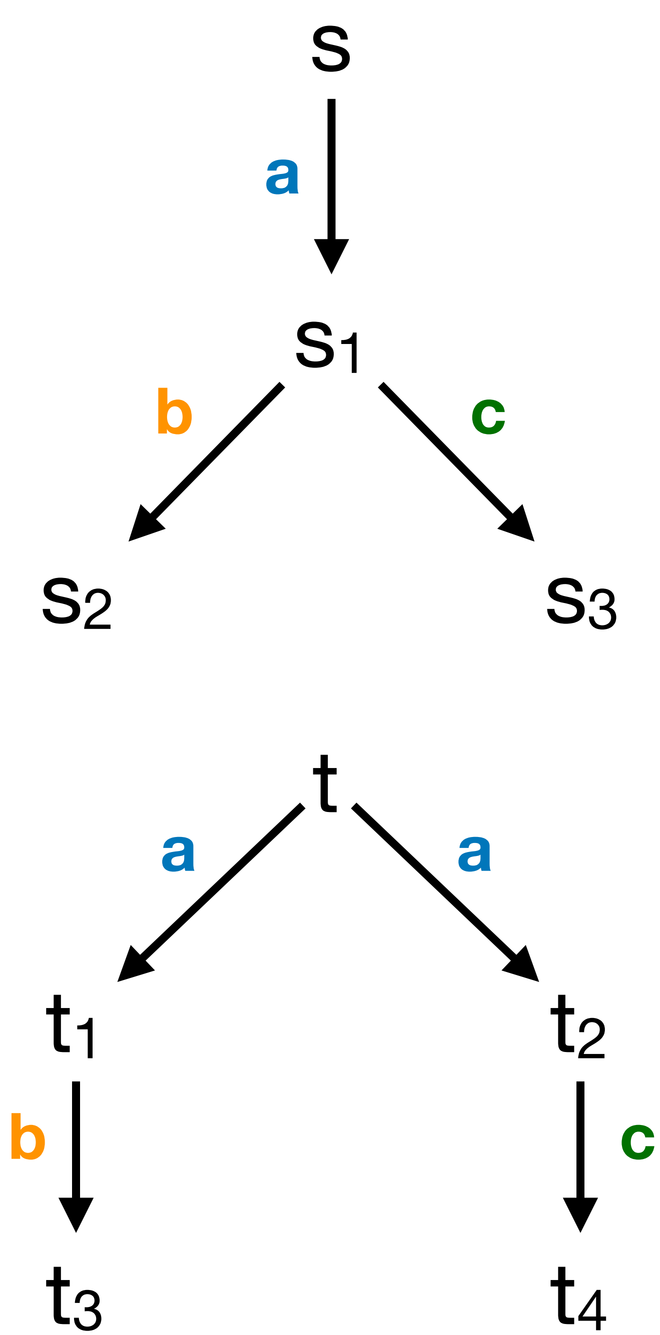
defender wins with an infinite game

Example 2



To show that s $\not\sim$ t, we provide a universal winning strategy for the **attacker**, starting from the configuration (s,t)

Example 2



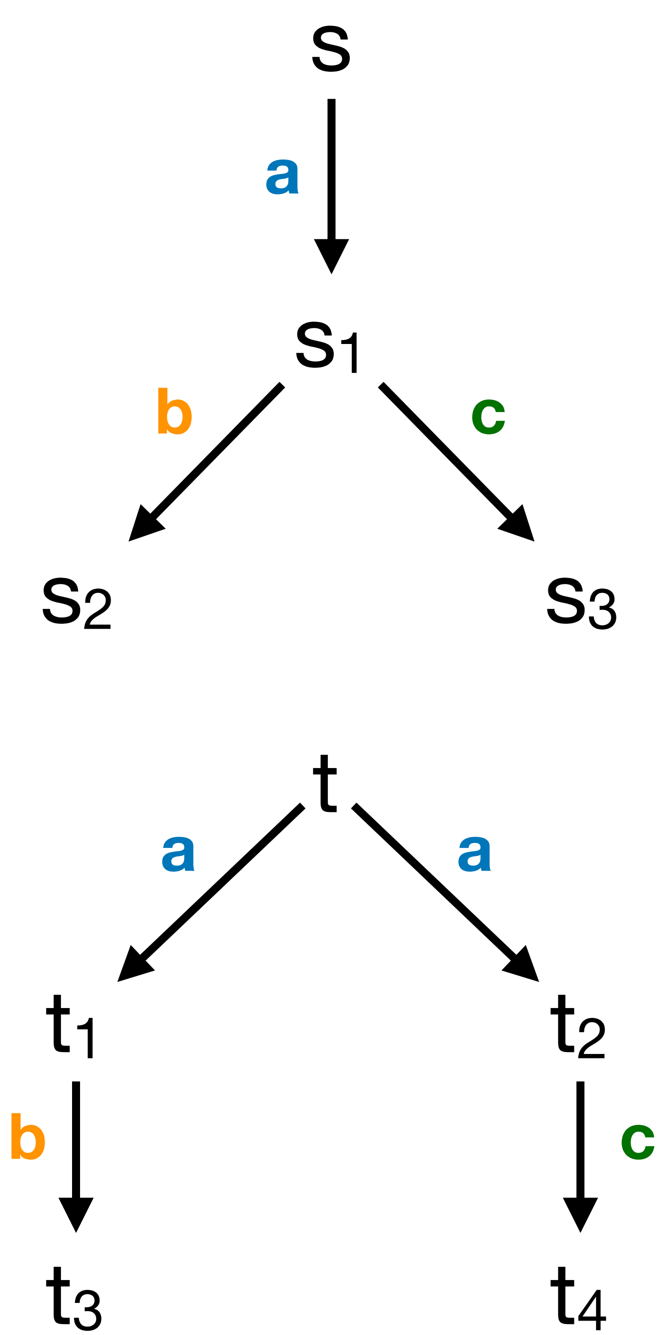
From (s,t)

attacker makes the move $s \xrightarrow{a} s_1$.
Then **defender** can react as

1. $t \xrightarrow{a} t_1$
2. $t \xrightarrow{a} t_2$

Hence from (s,t) there are two possible next configurations: (s_1,t_1) and (s_1, t_2).

Example 2



From (s_1,t_1)

attacker makes the move $s_1 \xrightarrow{c} s_3$. Then **defender** cannot move with matching **c** label.

From (s_1,t_2)

attacker makes the move $s_1 \xrightarrow{b} s_2$. Then **defender** cannot move with matching **b** label.

attacker wins with a finite game