Communicating automata

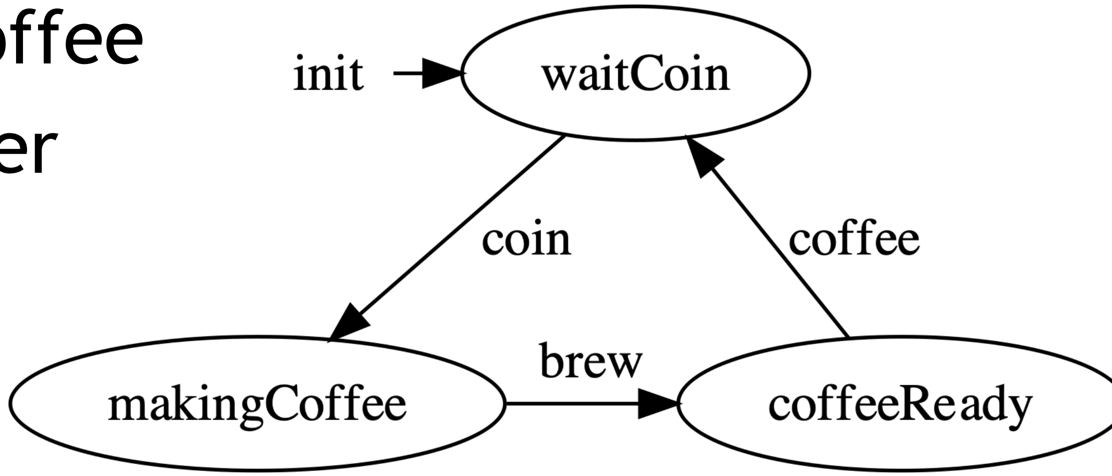
Communicating Automata (CA)

- Simple formalism to describe asynchronous concurrent systems

- Shows some of the basic concepts of this course:
  - System description using an automaton
  - Decomposition into communicating automata
  - automata product (parallelisation)
  - automata synchronisation
  - construction of the state graph

A coffee machine automaton

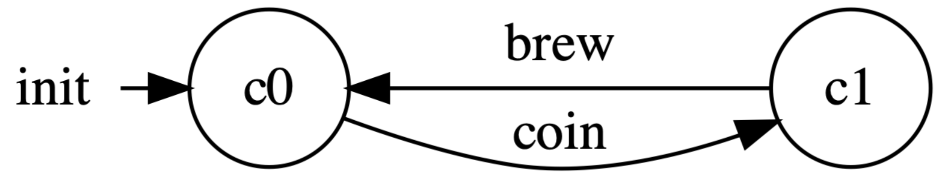
1. Wait for a coin
2. Brew a cup of coffee
3. Give it to the user



Decomposing the coffee machine

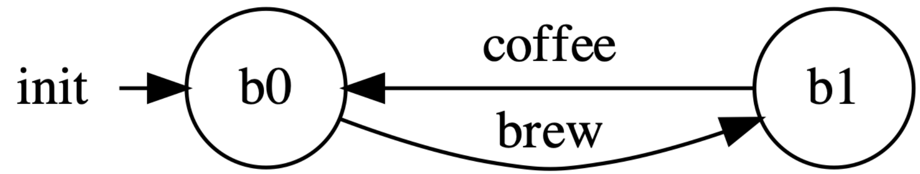
Coin subsystem

- Accepts a coin
- Orders to brew a coffee



Brewing subsystem

- Waits for a command
- Makes coffee



- How do we put these pieces together?
- Will the composition have the same behavior?

Labelled Transition Systems (LTS)

An LTS is a 4-ple $M = \langle S, A, T, s_0 \rangle$

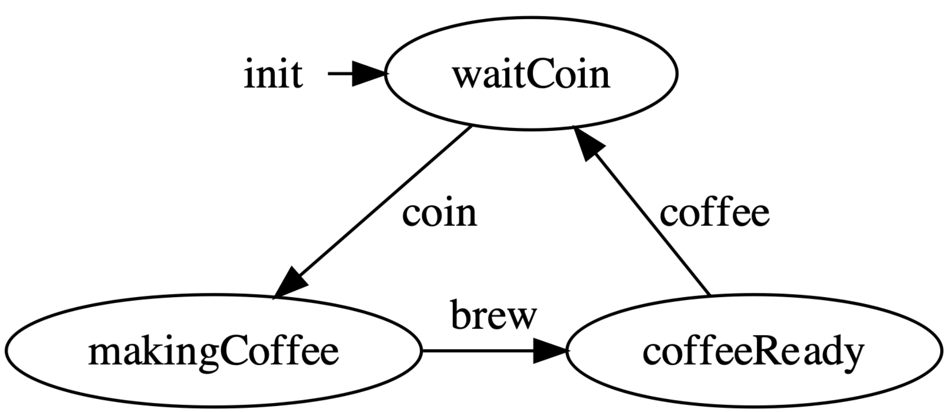
- S: Set of states, A: Set of actions
- $T \subseteq S \times A \times S$: Labelled transition relation
  - If $(s, a, s') \in T$, we simply write $s \xrightarrow{a} s'$
- s_0: Initial state

$$S = \{waitCoin, makingCoffee, coffeeReady\}$$

$$A = \{coin, brew, coffee\}$$

$$T = \{(w, coin, m), (m, brew, c), (c, coffee, w)\}$$

$$s_0 = waitCoin$$



LTS product (1/3)

- We have the LTSs of two systems
  - $M_1 = \langle S_1, A_1, T_1, s_{10} \rangle$
  - $M_2 = \langle S_2, A_2, T_2, s_{20} \rangle$
- We want the LTS of a system M that is composed of M_1 and M_2
- M_1 and M_2 evolve independently, in parallel

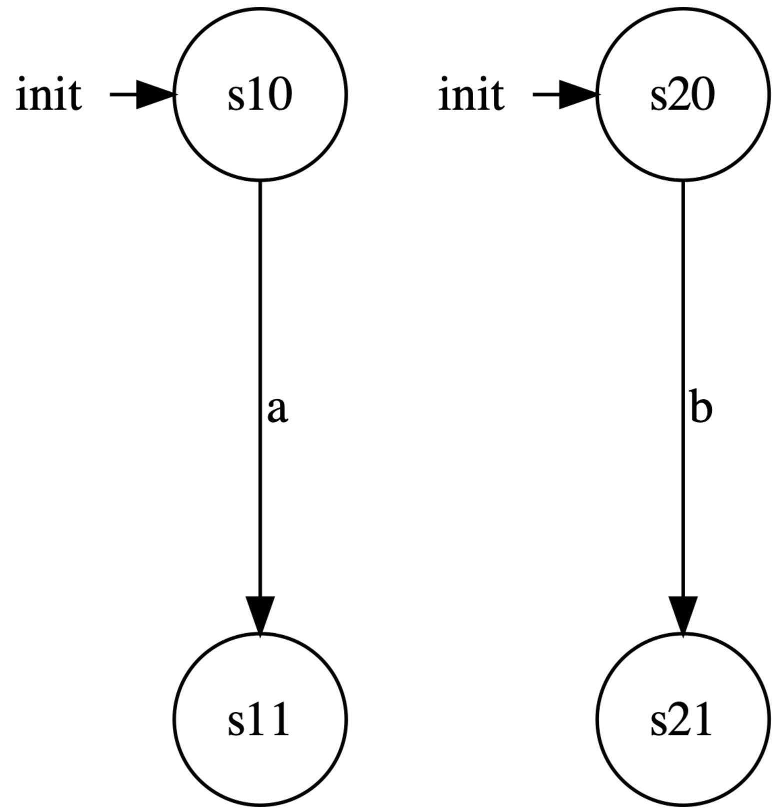
We can define a product of LTSs: $M = M_1 \otimes M_2$

LTS product (2/3)

Example: one system can only do *a*, the other can only do *b*.

The product should be able to do:

- *a*, then *b*
- *b*, then *a*



LTS product (3/3)

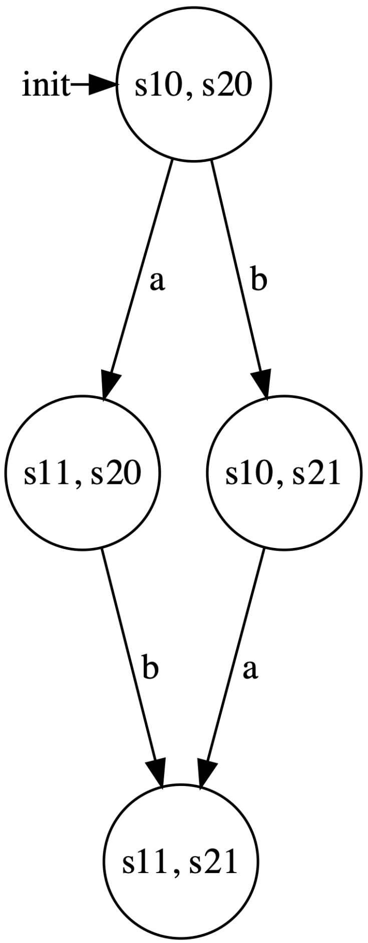
- States: Product of S_1, S_2
- Actions: Union of A_1, A_2
- Initial state: (s_{10}, s_{20})
- Transitions?

$$\frac{s_1 \xrightarrow{\mu} s_1'}{(s_1, s_2) \xrightarrow{\mu} (s_1', s_2)}$$

$$\frac{s_2 \xrightarrow{\mu} s_2'}{(s_1, s_2) \xrightarrow{\mu} (s_1, s_2')}$$

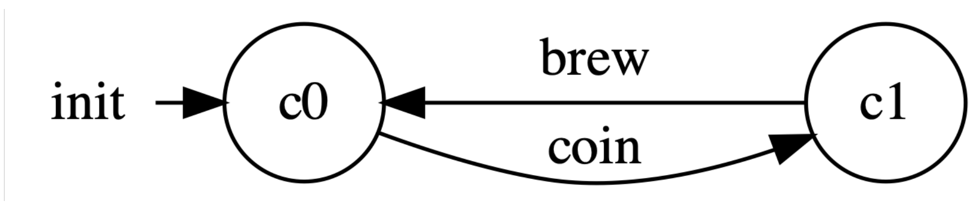
Inference rules

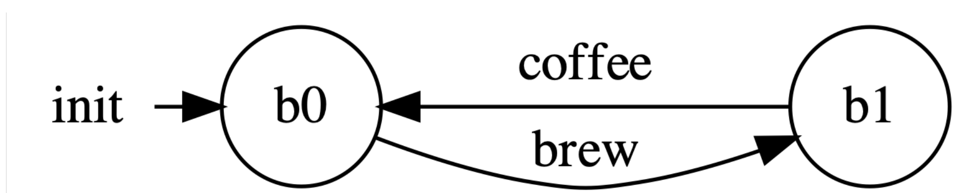
if all premises (above the line) are true, the conclusion (below the line) must also be true



What about communication?

- What we have seen is pure interleaving
  - The two components just alternate their execution
- But components often communicate
  - Let's go back at the coffee machine example
  - *brew* represents a communication between the two automata: they should do *brew* together





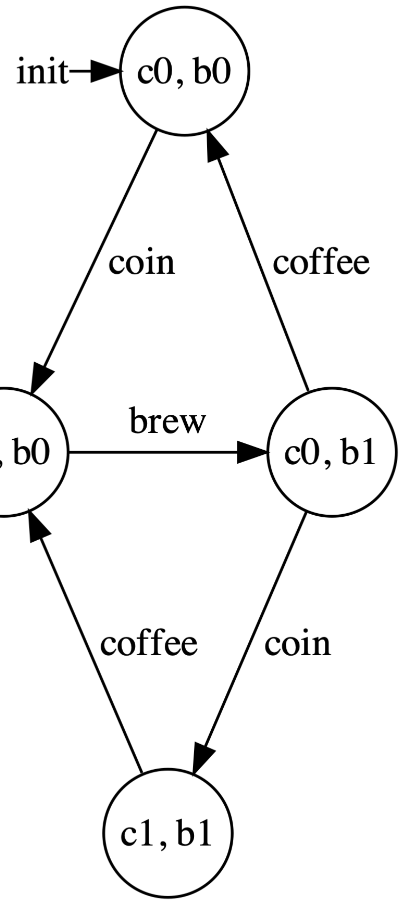
Synchronisation

- Let us introduce a set L of synchronising actions
- Extend $\otimes$ to $\otimes_L$
  - E.g., for our coffee machine, $L = \{ brew \}$
  - When M_1, M_2 can both do $\mu \in L$, they evolve together
  - When $L = \emptyset$: pure interleaving

$$\frac{s_1 \xrightarrow{\mu} s_1' \quad \mu \notin L}{(s_1, s_2) \xrightarrow{\mu} (s_1', s_2)} \qquad \frac{s_2 \xrightarrow{\mu} s_2' \quad \mu \notin L}{(s_1, s_2) \xrightarrow{\mu} (s_1, s_2')}$$

$$\frac{s_1 \xrightarrow{\mu} s_1' \quad s_2 \xrightarrow{\mu} s_2' \quad \mu \in L}{(s_1, s_2) \xrightarrow{\mu} (s_1', s_2')}$$

Composing the coffee machine



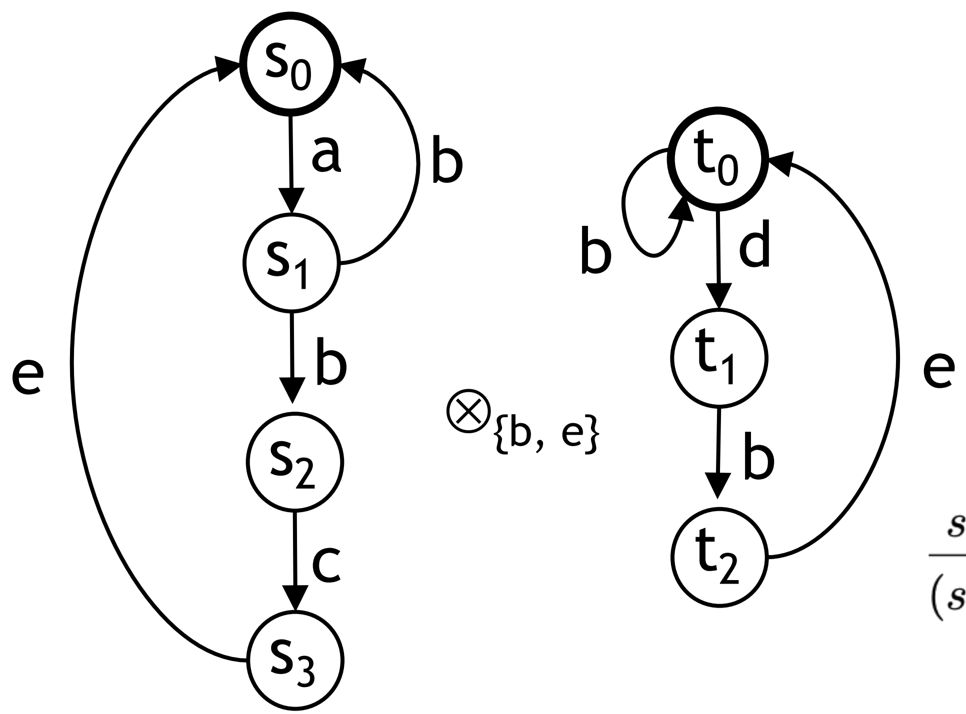
Wait a minute…

This automaton does not have the same behaviour as the original

- After brewing a coffee, it can accept a coin before giving the coffee

Exercise

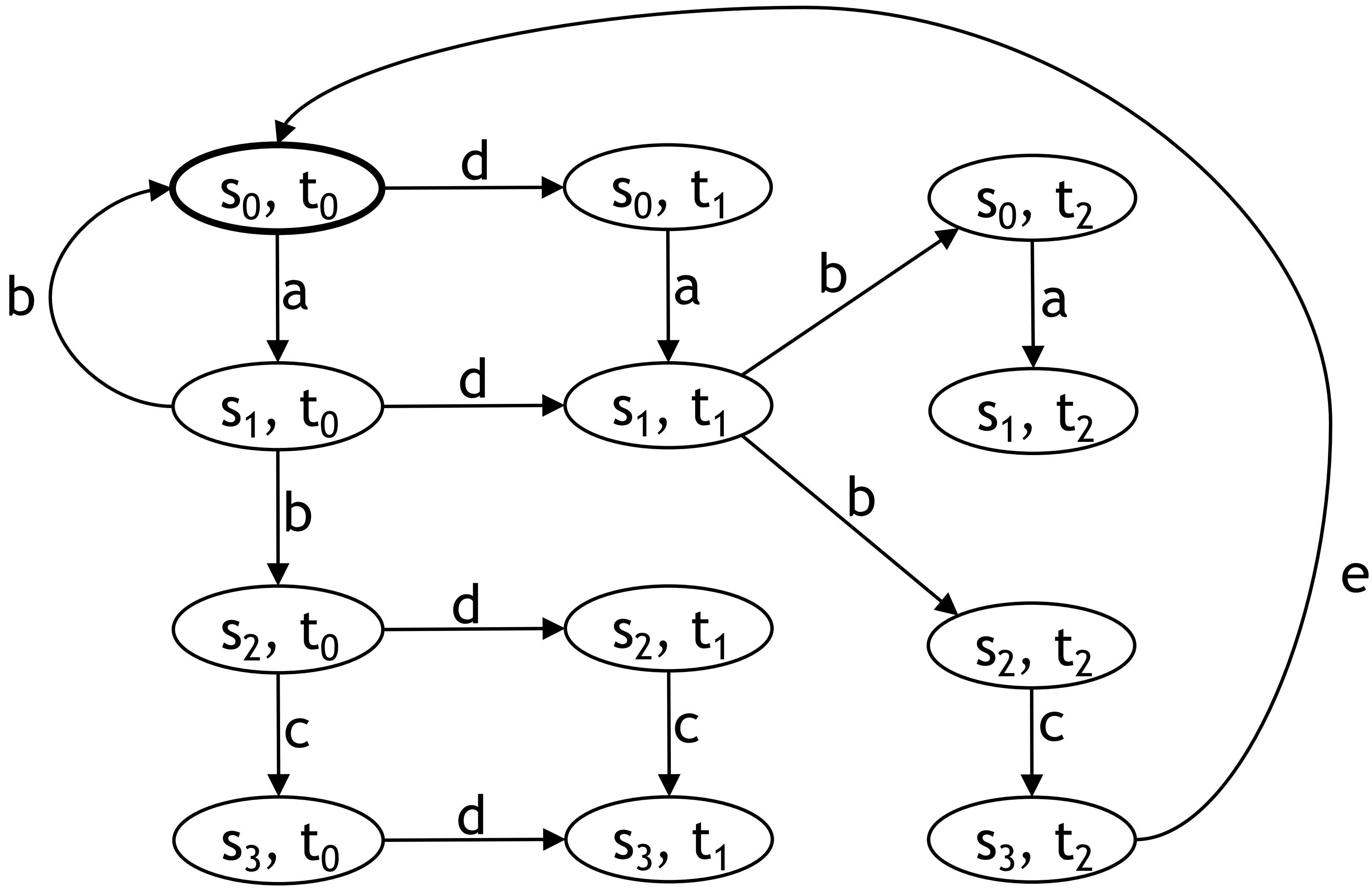
- Compute the following product



$$\frac{s_1 \xrightarrow{\mu} s_1' \quad \mu \notin L}{(s_1, s_2) \xrightarrow{\mu} (s_1', s_2)} \qquad \frac{s_2 \xrightarrow{\mu} s_2' \quad \mu \notin L}{(s_1, s_2) \xrightarrow{\mu} (s_1, s_2')}$$

$$\frac{s_1 \xrightarrow{\mu} s_1' \quad s_2 \xrightarrow{\mu} s_2' \quad \mu \in L}{(s_1, s_2) \xrightarrow{\mu} (s_1', s_2')}$$

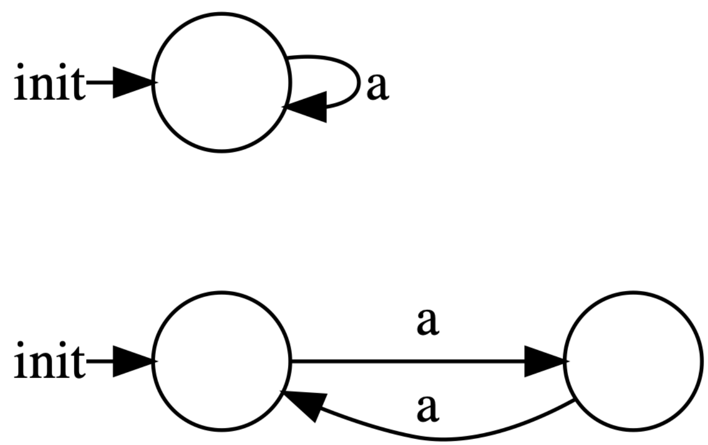
Solution


s_0, t_0
d
s_0, t_1
s_0, t_2
b
a
a
b
a
s_1, t_0
d
s_1, t_1
s_1, t_2
b
b
e
s_2, t_0
d
s_2, t_1
s_2, t_2
c
c
c
s_3, t_0
d
s_3, t_1
s_3, t_2

Behavioural Equivalences

Weaker than LTS equivalence: some automata are different, but have the same behaviour

- Example:



- Both can only do an ∞ sequence of a
- We need to formalise this notion

Strong bisimulation (1/2)

A binary relation between states

Binary relation = Set of pairs

When do states p and q have the same behaviour?

- They can do the same actions
- When they do an action, they must reach states with the same behaviour (recursive!)

Strong bisimulation

R is a strong bisimulation if $\forall\ (p, q) \in R$:

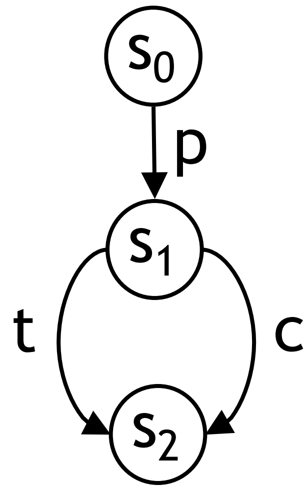
1. If $p \xrightarrow{a} p'$ then $\exists q'$ s.t. $q \xrightarrow{a} q'$ and $(p', q') \in \mathrm{R}$
2. If $q \xrightarrow{a} q'$ then $\exists p'$ s.t $p \xrightarrow{a} p'$ and $(p', q') \in \mathrm{R}$

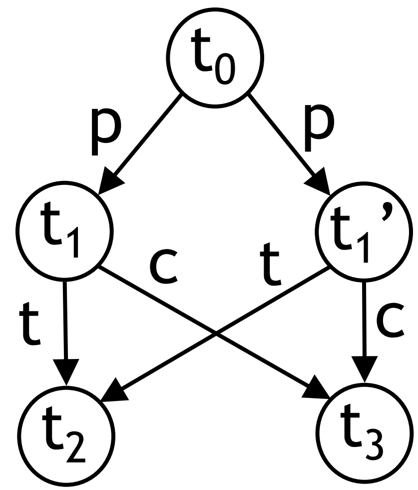
p, q are strongly bisimilar $(p \sim q)$ if there exists a strong bisimulation R such that $(p, q) \in R$

Two LTSs with initial states s_{10}, s_{20} are strongly bisimilar if $s_{10} \sim s_{20}$

Example (1/3)

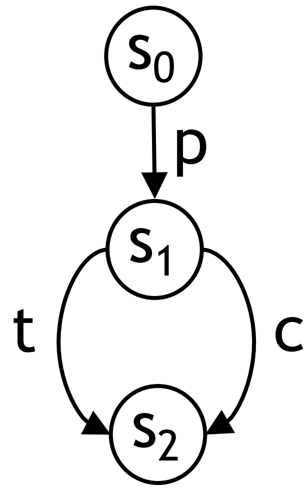
- Prove that $s_0 \sim t_0$

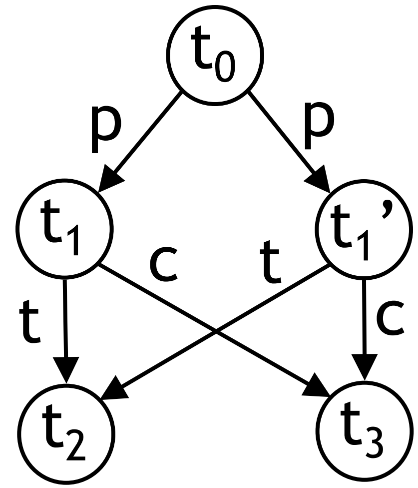




Example (2/3)

- Prove that $s_0 \sim t_0$

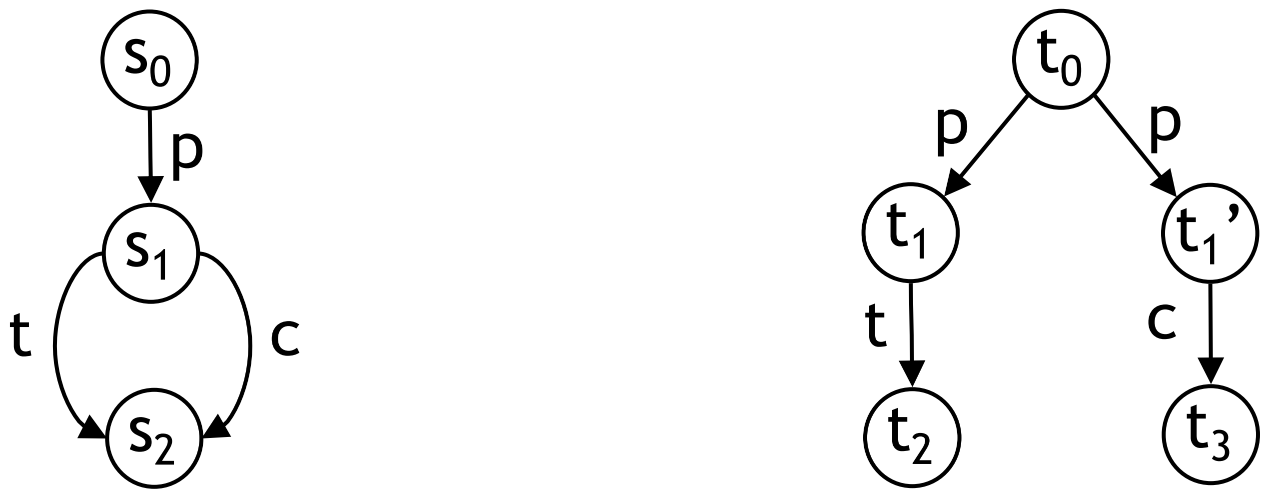




- Deadlocked states are bisimilar: $s_2 \sim t_2 , s_2 \sim t_3$
- $s_1 \sim t_1 , s_1 \sim t'_1$
- $s_0 \sim t_0$
- $R = \{ (s_0,t_0),(s_1,t_1), (s_1,t'_1), (s_2,t_2), (s_2,t_3) \}$

Example (3/3)

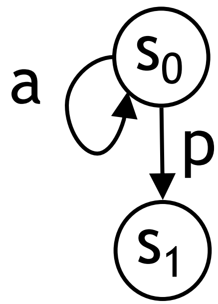
- These automata are not bisimilar

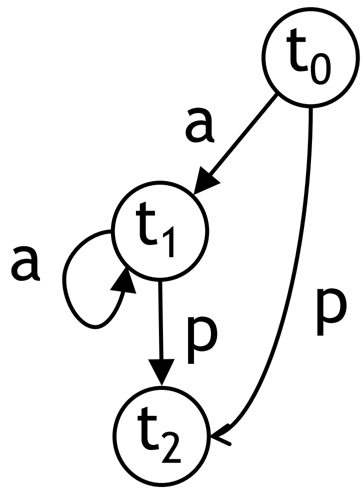


- Look at s_1, t_1, and t_1'
- This is equivalence checking, can be automated

Exercise

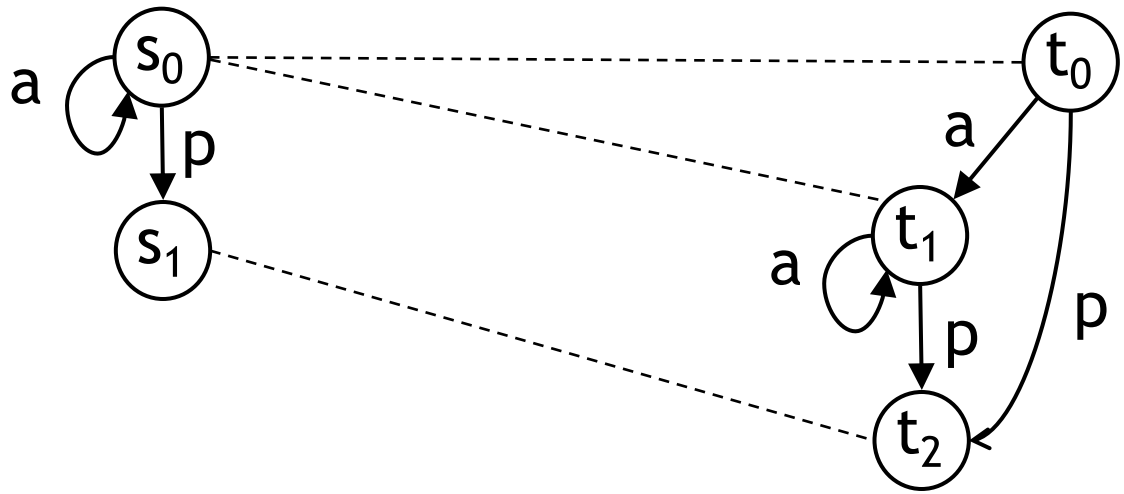
- Are these LTSs bisimilar? ($s_0 \sim t_0$?)





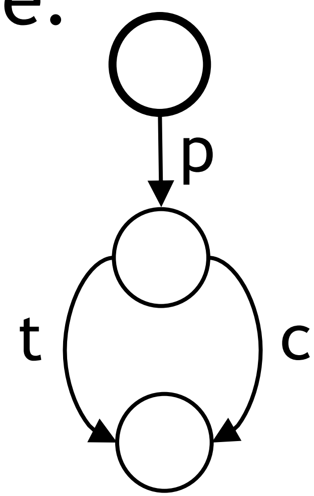
Solution

- Are these LTSs bisimilar? ($s_0 \sim t_0$?) Yes
  - Dashed lines represent the bisimulation relation
  - Self-transitions may be confusing…

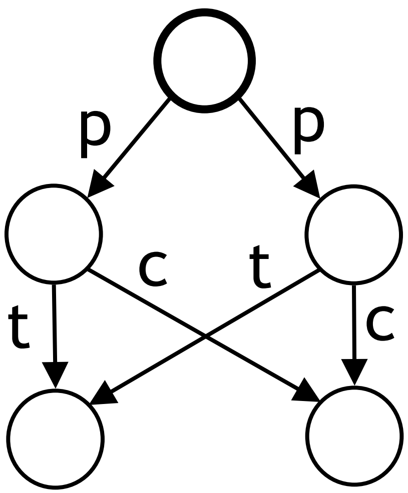


LTS minimization

- For every LTS M we can construct an M' that
  - Is strongly bisimilar to M
  - Has a minimal number of states/transitions
- M' is known as the minimal representative of M
- M' can be computed automatically, given M
- Example:



is the minimal representative of

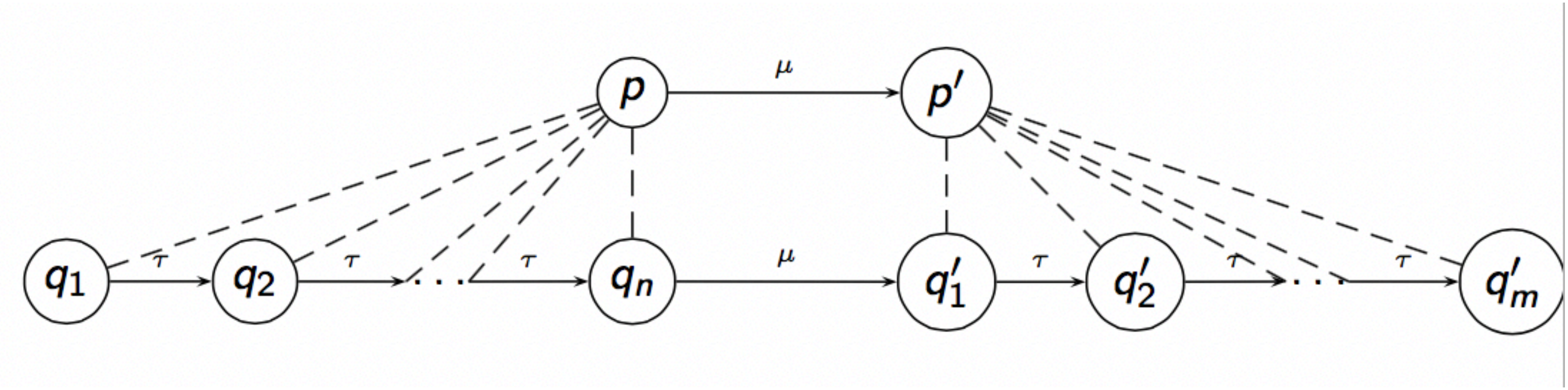


Internal actions and hiding

- Internal (or invisible) action
  - traditionally written i, or τ (tau)
  - Automata cannot synchronise on it
- Often, we want to check the equivalence of a specification S and an implementation P
  - But P may contain actions that are irrelevant to S.
  - Strong bisimulation does not work
- Solution:
  - In P, rename those irrelevant actions to i (hiding)
  - Define an equivalence that "ignores" internal actions

Branching bisimulation

- For non-τ actions, same as strong bisimulation
- "Collapse" sequences of τ actions



- You can also minimize an LTS up to branching bisimulation

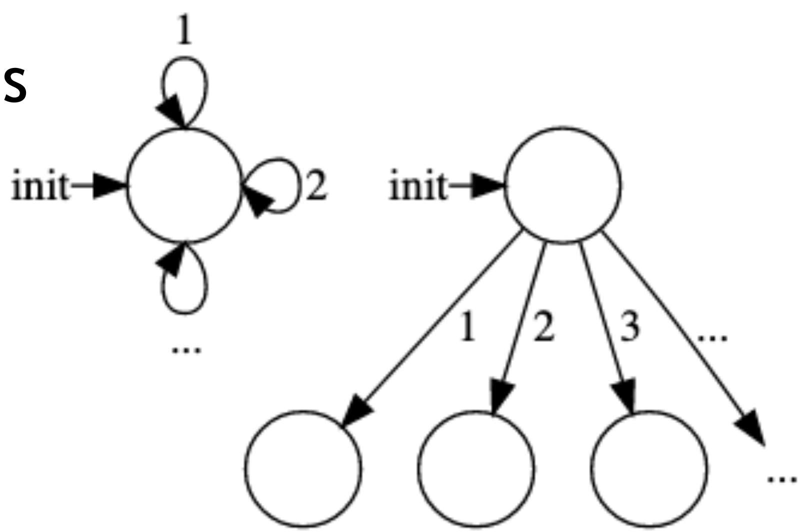
Rendezvous

When two (or more!) CA synchronize, we say that they perform a rendezvous. Two “styles”:

- Symmetrical (shown earlier)
  - No such distinction
  - Rendezvous on the same action
  - Easy to extend to many CA (multi-party rendezvous)
- Asymmetrical
  - Distinguish between input and output actions
  - Rendezvous on input/output pairs
  - Typically results in an internal action
  - Typical syntax: *’a — a* or *?a — !a* or *a? — a!*

Drawbacks

- Risk of state space explosion with $\otimes$
  - Size of $S_1 \times S_2$ = (Size of S_1) × (Size of S_2)
  - Minimization can help with that
- No modelling of data
  - Scenario: an automaton sends an `int` to another
  - Automata need a different action for each `int`
  - Receiver needs a different state for each int



- Too low-level for human use