Modelling and Validation of Concurrent System

António Ravara

May 7, 2024

Labelled Transition Systems

Summary and Plan

Last lecture

1. A new model of computation
   Captures the key aspects of concurrent reactive systems, centred on interactive behaviour
2. A new language
   The Calculus of Communicating Systems (CCS)
3. Syntax and Operational Semantics of CCS

Summary and Plan

Last lecture

1. A new model of computation
   Captures the key aspects of concurrent reactive systems, centred on interactive behaviour
2. A new language
   The Calculus of Communicating Systems (CCS)
3. Syntax and Operational Semantics of CCS

This lecture

1. Denotational semantics of CCS
2. A behavioural congruence relation

Labelled Transition Systems: the notion

Definition

A *Labelled Transition System* (LTS) is a triple

$$(\text{Proc}, \text{Act}, \xrightarrow{a})$$

1. Proc is a set (of *states* or *processes*)
2. Act is a set (of *labels* or *actions*)
3. $\xrightarrow{a} \subseteq \text{Proc} \times \text{Act} \times \text{Proc}$ is a relation
   (the *transition relation*)

For clarity sake, given a transition relation T, we write

$p \xrightarrow{a} q \in T$ instead of $(p, a, q) \in T$

Labelled Transition Systems: auxiliary notions

A *pointed* LTS is a quadruple

$$(\mathrm{Proc}, \mathrm{Act}, \xrightarrow{a}, s)$$

where $(\mathrm{Proc}, \mathrm{Act}, \xrightarrow{a})$ is an LTS and a state $s \in \mathrm{Proc}$ is distinguished as the initial state.

The extended transition (or traces) relation

$$T^* \subseteq \mathrm{Proc} \times \mathrm{Act}^* \times \mathrm{Proc}$$

of a transition relation T is inductively defined by the rules below

$$(p, \varepsilon, p) \in T^*$$

$$(p, at, q) \in T^* \text{ if } \exists r.((p, a, r) \in T \wedge (r, t, q) \in T^*)$$

Accessible LTS

A pointed LTS $(\mathrm{Proc}, \mathrm{Act}, \xrightarrow{a}, s)$ is *accessible* if

$$\forall p \in \mathrm{Proc}\,.\,\exists t \in \mathrm{Act}^*.\, s \xrightarrow{t} p$$

We consider henceforth accessible LTSs only to represent concurrent systems.

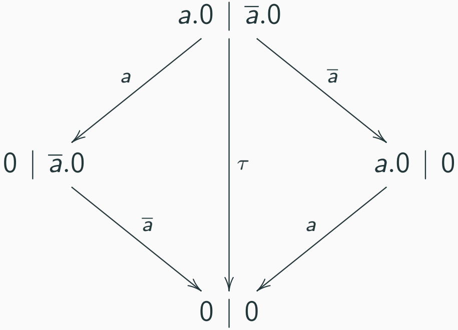
An accessible LTS $(\mathrm{Proc}, \mathrm{Act}, \xrightarrow{a}, s)$ **corresponds to a machine:**

1. with initial state s
2. able of executing actions in Act
3. whose states are elements of Proc, all accessible by sequences of actions from the initial state

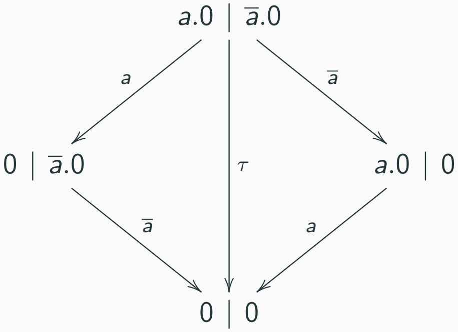
Labelled Transition Systems and CCS

One defines easily an isomorphic function from CCS to LTS (similar to the translation of Finite Automata into Regular Expressions).

1. Any CCS term can be described by an accessible LTS
2. Any accessible LTS can be described by a CCS term



Labelled Transition Systems as infinite sets/trees



is the graphical representation of a set of transitions:

$$\{(a.0 \mid \overline{a}.0, a, 0 \mid \overline{a}.0), \ldots\}$$

Labelled Transition Systems as infinite sets/trees

Process $A = a.A$

is represented as an LTS with a single state and a self-loop with label a (a single transition)

What about $AStar = a.(0 \mid AStar)$**?**

Labelled Transition Systems as infinite sets/trees

Process $A = a.A$

is represented as an LTS with a single state and a self-loop with label a (a single transition)

What about $AStar = a.(0 \mid AStar)$**?**

$AStar \xrightarrow{a} (0 \mid AStar) \xrightarrow{a} (0 \mid (0 \mid AStar)) \ \cdots$

The LTS has

- an infinite number of states, and
- an infinite number of triples in the transition relation

How can one mathematically define *infinite trees* and relations on them?

On coinduction

Basic concepts

Check: Dexter Kozen, Alexandra Silva. "Practical Coinduction". DOI 10.1.1.252.3961

Coinduction defines infinite datastructures

streams or infinite trees are not finitely representable

Polymorphic streams

$$\texttt{Stream}\ a = \mathbf{S}\ a\ (\texttt{Stream}\ a)$$

where

- a is a generic datatype, and
- **S** is a (data) constructor

Polymorphic streams

Definition

$$\texttt{Stream}\ a = \mathbf{S}\ a\ (\texttt{Stream}\ a)$$

To manipulate coinductive datatypes one needs "destructors" (operations to observe the values)

Stream destructors

```
head (S a astream) = a

tail (S a astream) = astream
```

Lexicographic order on streams

Consider an ordered alphabet $(A, \leq)$

The ordering $\leq_{lex}$ on A-streams is maximum relation $\mathcal{R} \subseteq A^\omega \times A^\omega$ satisfying the following property.

If $\sigma_1 \,\mathcal{R}\, \sigma_2$ **then**

$$\texttt{head}(\sigma_1) \leq \texttt{head}(\sigma_2) \text{ and}$$

$$\texttt{head}(\sigma_1) = \texttt{head}(\sigma_2) \text{ implies } \texttt{tail}(\sigma_1) \,\mathcal{R}\, \texttt{tail}(\sigma_2)$$

Lexicographic order on streams

Consider an ordered alphabet $(A, \leq)$

The ordering $\leq_{lex}$ on A-streams is maximum relation $\mathcal{R} \subseteq A^\omega \times A^\omega$ satisfying the following property.

If $\sigma_1 \, \mathcal{R} \, \sigma_2$ **then**

$$\texttt{head}(\sigma_1) \leq \texttt{head}(\sigma_2) \text{ and}$$

$$\texttt{head}(\sigma_1) = \texttt{head}(\sigma_2) \text{ implies } \texttt{tail}(\sigma_1) \, \mathcal{R} \, \texttt{tail}(\sigma_2)$$

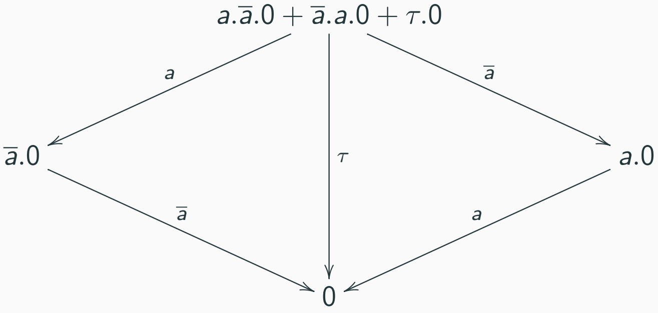
Theorem: $\leq_{lex}$ **is**

1. reflexive and transitive
2. canonic (exists and is unique)
3. the union of all relations satisfying the property

Behavioural Equivalence

Why is an equivalence notion needed?

Syntactically different processes may denote the same LTS



Apart from the state names, the LTS (structure and arrows) is the same of that of $a.0 \mid \overline{a}.0$

Interactive behaviour

1. Can 0 do anything different from $0 \mid 0$?
   None have immediate actions available.
2. Can $\overline{a}.0$ do anything different from $0 \mid \overline{a}.0$?
   Both can only do the action $\overline{a}$ and fall in the previous situation.
3. Can $a.0$ do anything different from $a.0 \mid 0$?
   Both can only do the action a and fall in the initial situation.
4. Can $a.0 \mid \overline{a}.0$ do anything different from $a.\overline{a}.0 + \overline{a}.a.0 + \tau.0$?
   Both can either do the action $\overline{a}$ and fall in the second situation or do the action a and fall in the previous situation.

Requirement to equate interactive behaviour

Key ideas

1. An external observer should not tell apart equivalent systems
   All possible sequences of interactive behaviour of one system should be possible in the other.
2. The notion should be a congruence relation
   To allow to substitute equals for equals.

Two systems should be considered **behaviourally equivalent** if:

1. one can be used instead of the other, and
2. an external entity, interacting with them, cannot notice the difference.

Simulation

First idea: let one system **imitate** the other.

Definition

A binary relation $\mathcal{R} \subseteq \mathrm{Proc} \times \mathrm{Proc}$ is a *simulation*, if

whenever $(p, q) \in \mathcal{R}$ then, for each $a \in \mathrm{Act}$,

if $p \xrightarrow{a} p'$ then $q \xrightarrow{a} q'$ and $(p', q') \in \mathcal{R}$

Simulation

First idea: let one system **imitate** the other.

Definition

A binary relation $\mathcal{R} \subseteq \text{Proc} \times \text{Proc}$ is a *simulation*, if

whenever $(p, q) \in \mathcal{R}$ then, for each $a \in \text{Act}$,

if $p \xrightarrow{a} p'$ then $q \xrightarrow{a} q'$ and $(p', q') \in \mathcal{R}$

Notice that

1. the definition above is coinductive

Simulation

First idea: let one system **imitate** the other.

Definition

A binary relation $\mathcal{R} \subseteq \text{Proc} \times \text{Proc}$ is a *simulation*, if

whenever $(p, q) \in \mathcal{R}$ then, for each $a \in \text{Act}$,

if $p \xrightarrow{a} p'$ then $q \xrightarrow{a} q'$ and $(p', q') \in \mathcal{R}$

Notice that

1. the definition above is coinductive
   Proc has infinite terms

Simulation

First idea: let one system **imitate** the other.

Definition

A binary relation $\mathcal{R} \subseteq \text{Proc} \times \text{Proc}$ is a *simulation*, if

whenever $(p, q) \in \mathcal{R}$ then, for each $a \in \text{Act}$,

if $p \xrightarrow{a} p'$ then $q \xrightarrow{a} q'$ and $(p', q') \in \mathcal{R}$

Notice that

1. the definition above is coinductive
   Proc has infinite terms
2. who are then the destructors of Proc?

Simulation

First idea: let one system **imitate** the other.

Definition

A binary relation $\mathcal{R} \subseteq \text{Proc} \times \text{Proc}$ is a *simulation*, if

whenever $(p, q) \in \mathcal{R}$ then, for each $a \in \text{Act}$,

if $p \xrightarrow{a} p'$ then $q \xrightarrow{a} q'$ and $(p', q') \in \mathcal{R}$

Notice that

1. the definition above is coinductive
   Proc has infinite terms
2. who are then the destructors of Proc?
   the transition rules

Example

Consider the processes $P = coin.tea.pick.0$ and
$Q = coin.(tea.pick.0 + coffee.pick.0)$

It is simple to see that P is simulated by Q:

Example

Consider the processes $P = \mathit{coin}.\mathit{tea}.\mathit{pick}.0$ and $Q = \mathit{coin}.(\mathit{tea}.\mathit{pick}.0 + \mathit{coffee}.\mathit{pick}.0)$

It is simple to see that P is simulated by Q:

$$P \xrightarrow{\mathit{coin}} \mathit{tea}.\mathit{pick}.0 \text{ and } Q \xrightarrow{\mathit{coin}} (\mathit{tea}.\mathit{pick}.0 + \mathit{coffee}.\mathit{pick}.0)$$

$$\mathit{tea}.\mathit{pick}.0 \xrightarrow{\mathit{tea}} \mathit{pick}.0 \text{ and } (\mathit{tea}.\mathit{pick}.0 + \mathit{coffee}.\mathit{pick}.0) \xrightarrow{\mathit{tea}} \mathit{pick}.0$$

$$\mathit{pick}.0 \xrightarrow{\mathit{pick}} 0 \text{ and } \mathit{pick}.0 \xrightarrow{\mathit{pick}} 0$$

Simulation

Recall that

$$P = \mathit{coin}.\mathit{tea}.\mathit{pick}.0 \text{ and } Q = \mathit{coin}.(\mathit{tea}.\mathit{pick}.0 + \mathit{coffee}.\mathit{pick}.0)$$

P however, does not simulate Q:

Simulation

Recall that

$$P = \mathit{coin}.\mathit{tea}.\mathit{pick}.0 \text{ and } Q = \mathit{coin}.(\mathit{tea}.\mathit{pick}.0 + \mathit{coffee}.\mathit{pick}.0)$$

P however, does not simulate Q:

$$Q \xrightarrow{\mathit{coin}} (\mathit{tea}.\mathit{pick}.0 + \mathit{coffee}.\mathit{pick}.0) \text{ and } P \xrightarrow{\mathit{coin}} \mathit{tea}.\mathit{pick}.0$$

$$(\mathit{tea}.\mathit{pick}.0 + \mathit{coffee}.\mathit{pick}.0) \xrightarrow{\mathit{coffee}} \mathit{pick}.0 \text{ but } \mathit{tea}.\mathit{pick}.0 \not\xrightarrow{\mathit{coffee}}$$

Therefore, equivalent systems should simulate each other.

Mutual simulations

Consider the systems

$$P \;= coin.tea.pick.0 + coin.tea.pick.0 + Q$$
$$Q \;= coin.(tea.pick.0 + coffee.pick.0)$$

1. It is easy to see that P simulates Q and Q simulates P
2. P however, has more “non-determinism” than Q, what may lead to undesired situations:

$$P \;\xrightarrow{coin} P' = tea.pick.0 \text{ and}$$
$$Q \;\xrightarrow{coin} Q' = (tea.pick.0 + coffee.pick.0)$$

Now P' and Q' no longer simulate each other!

Bisimulation: a co-inductive behavioural equivalence

A candidate for behavioural equivalence

Symmetric simulation.

Definition

A binary relation $\mathcal{R} \subseteq \text{Proc} \times \text{Proc}$ is a *strong bisimulation*, if

whenever $(p, q) \in \mathcal{R}$ then, for each $a \in \text{Act}$,

1. if $p \xrightarrow{a} p'$ then $q \xrightarrow{a} q'$ and $(p', q') \in \mathcal{R}$
2. if $q \xrightarrow{a} q'$ then $p \xrightarrow{a} p'$ and $(p', q') \in \mathcal{R}$

Canonicity: does such a relation always exists? Is it unique?

Strong bisimilarity

Two processes p and q are *strongly bisimilar* ($p \sim q$), if there exists a strong bisimulation $\mathcal{R}$ such that $(p, q) \in \mathcal{R}$

Properties of Bisimilarity

Let bisimilarity be such that

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

Theorem

Bisimilarity is the largest bisimulation

Theorem

Bisimilarity is a congruence relation – it is:

1. an equivalence relation (reflexive, symmetric, and transitive)
2. substitutive: preserved by the operators of the language

Properties of Bisimilarity

Substitutivity / Preservation by the operators

$P \sim Q$ implies

- $\alpha.P \sim \alpha.Q$
- $P \mid R \sim Q \mid R$ and $P + R \sim Q + R$
- $A\langle a_1, \ldots, a_n \rangle \sim B\langle a_1, \ldots, a_n \rangle$, if
  $A(x_1, \ldots, x_n) = P$ and $B(y_1, \ldots, y_n) = Q$

Properties of Bisimilarity

Substitutivity / Preservation by the operators

$P \sim Q$ implies

- $\alpha.P \sim \alpha.Q$
- $P \mid R \sim Q \mid R$ and $P + R \sim Q + R$
- $A\langle a_1, \ldots, a_n\rangle \sim B\langle a_1, \ldots, a_n\rangle$, if
  $A(x_1, \ldots, x_n) = P$ and $B(y_1, \ldots, y_n) = Q$

Theorem

$(\text{Proc}, 0, \sim)$ is a commutative monoid.

Recall the definition of capture avoiding substitution and of α-convertion.

Substitution

Let $P\{\vec{a} \leftarrow \vec{b}\}$ denote the simultaneous substitution of the free occurrences of the actions $\vec{a}$ in P for $\vec{b}$.

α-convertion

The binary relation $=_\alpha$ on processes is inductively defined by the rule $(\mathbf{new}\, a)P =_\alpha (\mathbf{new}\, b)P\{a \leftarrow b\}$ if $b \notin \mathrm{bn}(P)$, and homomorphic rules on the remaining process constructs.

Theorem

α-convertion is a congruence relation and a bisimulation.

How to prove systems bisimilar?

One only needs to present a bisimulation.

Example

$$
\begin{aligned}
P &= \mathit{coin}.P' \text{ and } P' = \mathit{tea}.P'' \text{ and } P'' = \mathit{pick}.0 \\
Q &= \mathit{coin}.Q' \text{ and } Q' = (\mathit{tea}.Q'' + \mathit{tea}.Q''') \text{ where} \\
&\quad Q'' = \mathit{pick}.0 \text{ and } Q''' = \mathit{pick}.(0 \mid 0)
\end{aligned}
$$

The binary relation

$$\{(P, Q), (P', Q'), (P'', Q''), (P'', Q'''), (0, 0), (0, 0 \mid 0)\}$$

is a bisimulation.

How to prove systems not bisimilar?

One needs to show that no binary relation between them including the initial states is a bisimulation.

How can one do this?

- Enumerating all relations can be very inefficient
  There are $2^{|\mathrm{Proc}|^2}$ relations!
- Use a characterisation: the strong bisimulation game:
  - consider a "board" with two LTSs, each with a pin
  - consider two players, a defender, aiming at proving the systems bisimilar, and an attacker, aiming at the oposite.

  The game is played in rounds:
  - the attacker moves the pin in one LTS following an a-arrow (for some $a \in \mathrm{Act}$
  - the defender must respond moving the other pin in the other LTS following an a-arrow with the same $a \in \mathrm{Act}$

How to prove systems not bisimilar?

Winning conditions of the bisimulation game

- if a player cannot move, the other wins
- if the game is infinite, the defender wins

Theorem

- Processes P and Q are strongly bisimilar, if and only if, the defender has a universal winning strategy starting the game from (P, Q)
- Processes P and Q are NOT strongly bisimilar, if and only if, the attacker has a universal winning strategy starting the game from (P, Q)

How to prove systems not bisimilar?

Example
Let $P = a.(b.c.0 + b.d.0)$ and $Q = a.b.c.0 + a.b.d.0$

1. The attacker starts and moves P with a to $b.c.0 + b.d.0$
2. The defender responds moving Q with a to $b.c.0$
3. The attacker now moves $b.c.0 + b.d.0$ with b to $d.0$
4. The defender responds moving $b.c.0$ with b to $c.0$
5. The attacker now moves $d.0$ with d to 0
6. The defender cannot respond and looses the game