Introduction to labelled transition systems

Luís Soares Barbosa

HASLab - INESC TEC
Universidade do Minho
Braga, Portugal

February 2019

Reactive systems

Reactive system

system that computes by reacting to stimuli from its environment along its overall computation

- in contrast to sequential systems whose meaning is defined by the results of finite computations, the behaviour of reactive systems is mainly determined by interaction and mobility of non-terminating processes, evolving concurrently.
- observation $\equiv$ interaction
- behaviour $\equiv$ a structured record of interactions

Labelled Transition System

Definition

A LTS over a set N of names is a tuple $\langle S, N, \longrightarrow \rangle$ where

- $S = \{s_0, s_1, s_2, \ldots\}$ is a set of states
- $\longrightarrow \subseteq S \times N \times S$ is the transition relation, often given as an N-indexed family of binary relations

$$s \xrightarrow{a} s' \equiv \langle s, a, s' \rangle \in \longrightarrow$$

Labelled Transition System

System

Given a LTS $\langle S, N, \longrightarrow\rangle$, each state $s \in S$ determines a system over all states reachable from s and the corresponding restriction of $\longrightarrow$.

LTS classification

- deterministic
- non deterministic
- finite
- finitely branching
- image finite
- ...

Reachability

Definition

The reachability relation, $\longrightarrow^* \subseteq S \times N^* \times S$, is defined inductively

- $s \xrightarrow{\epsilon}^* s$ for each $s \in S$, where $\epsilon \in N^*$ denotes the empty word;
- if $s \xrightarrow{a} s''$ and $s'' \xrightarrow{\sigma}^* s'$ then $s \xrightarrow{a\sigma}^* s'$, for $a \in N, \sigma \in N^*$

Reachable state

$t \in S$ is reachable from $s \in S$ iff there is a word $\sigma \in N^*$ st $s \xrightarrow{\sigma}^* t$

An alternative characterisation

Coalgebraic characterization (morphism)

A morphism $h : \langle S, \mathsf{next}\rangle \longrightarrow \langle S', \mathsf{next}'\rangle$ is a function $h : S \longrightarrow S'$ st the following diagram commutes

$$\begin{tikzcd} S \times N \arrow[r, "\mathsf{next}"] \arrow[d, "h \times \mathsf{id}"'] & \mathcal{P}S \arrow[d, "\mathcal{P}h"] \\ S' \times N \arrow[r, "\mathsf{next}'"] & \mathcal{P}S' \end{tikzcd}$$

i.e.,

$$\mathcal{P}h \cdot \mathsf{next} \;=\; \mathsf{next}' \cdot (h \times \mathsf{id})$$

or, going pointwise,

$$\{h\,x \mid x \in \mathsf{next}\ \langle s, a\rangle\} \;=\; \mathsf{next}'\ \langle h\,s, a\rangle$$

An alternative characterisation

Coalgebraic characterization (morphism)

A morphism $h : \langle S, \text{next} \rangle \longrightarrow \langle S', \text{next}' \rangle$

- preseves transitions:

$$s' \in \text{next}\ \langle s, a \rangle \Rightarrow h\ s' \in \text{next}'\ \langle h\ s, a \rangle$$

- reflects transitions:

$$r' \in \text{next}'\ \langle h\ s, a \rangle \Rightarrow \langle \exists\ s' \in S\ :\ s' \in \text{next}\ \langle s, a \rangle :\ r' = h\ s' \rangle$$

(why?)

Comparison

- Both definitions coincide at the object level:

$$\langle s, a, s' \rangle \in T \quad \equiv \quad s' \in \text{next}\ \langle s, a \rangle$$

- Wrt morphisms, the relational definition is more general, corresponding, in coalgebraic terms to

$$\mathcal{P}h \cdot \text{next} \quad \subseteq \quad \text{next}' \cdot (h \times \text{id})$$

How can these notions of morphism be used to compare LTS?

Comparison

- Both definitions coincide at the object level:

$$\langle s, a, s' \rangle \in T \quad \equiv \quad s' \in \text{next}\ \langle s, a \rangle$$

- Wrt morphisms, the relational definition is more general, corresponding, in coalgebraic terms to

$$\mathcal{P}h \cdot \text{next} \quad \subseteq \quad \text{next}' \cdot (h \times \text{id})$$

How can these notions of morphism be used to compare LTS?

Process algebras

CCS - Syntax

$$\mathcal{P} \ni P, Q ::= K \mid \alpha.P \mid \sum_{i \in I} P_i \mid P[f] \mid P|Q \mid P \backslash L$$

where

- $\alpha \in N \cup \overline{N} \cup \{\tau\}$ is an action
- K s a collection of process names or process contants
- I is an indexing set
- $L \subseteq N \cup \overline{N}$ is a set of labels
- f is a function that renames actions s.t. $f(\tau) = \tau$ and $f(\bar{a}) = \overline{f(a)}$
- notation:

  $\mathbf{0} = \sum_{i \in \emptyset} P_i$
  $P_1 + P_2 = \sum_{i \in \{1,2\}} P_i$
  $[f] = [b_1/a_1, \ldots, b_n/a_n]$

Process algebras

Syntax

$$\mathcal{P} \ni P, Q ::= K \mid \alpha.P \mid \sum_{i \in I} P_i \mid P[f] \mid P|Q \mid P \backslash L$$

Exercise: Which are syntactically correct?

$$a.b.A + B \quad (1)$$
$$(a.\mathbf{0} + \overline{a}.A) \backslash \{a, b\} \quad (2)$$
$$(a.\mathbf{0} + \overline{a}.A) \backslash \{a, \tau\} \quad (3)$$
$$a.B + [a/b] \quad (4)$$
$$\tau.\tau.B + \mathbf{0} \quad (5)$$
$$(a.B + b.B)[a/a, b/\tau] \quad (6)$$
$$(a.B + \tau.B)[a/b, a/a] \quad (7)$$
$$(a.b.A + \overline{a}.\mathbf{0})|B \quad (8)$$
$$(a.b.A + \overline{a}.\mathbf{0}).B \quad (9)$$
$$(a.b.A + \overline{a}.\mathbf{0}) + B \quad (10)$$
$$(\mathbf{0}|\mathbf{0}) + \mathbf{0} \quad (11)$$

CCS semantics - building an LTS

(act)
$$\frac{}{\alpha.P \xrightarrow{\alpha} P}$$

(sum-j)
$$\frac{P_j \xrightarrow{\alpha} P_j'}{\sum_{i\in I} P_i \xrightarrow{\alpha} P_j'} \; j \in I$$

(com1)
$$\frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}$$

(com2)
$$\frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}$$

(com3)
$$\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{\overline{a}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

(res)
$$\frac{P \xrightarrow{\alpha} P'}{P\backslash L \xrightarrow{\alpha} P'\backslash L} \; \alpha, \overline{\alpha} \notin L$$

(rel)
$$\frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$$

Exercise: Draw the LTS's

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

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

$$SmUni = (CM|CS)\backslash\{\text{coin}, \text{coffee}\}$$

CCS semantics - building an LTS

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}\ \text{(act)}$$

$$\frac{P_j \xrightarrow{\alpha} P_j'}{\sum_{i \in I} P_i \xrightarrow{\alpha} P_j'}\ j \in I \quad \text{(sum-j)}$$

$$\frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}\ \text{(com1)}$$

$$\frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}\ \text{(com2)}$$

$$\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{\overline{a}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}\ \text{(com3)}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \backslash L \xrightarrow{\alpha} P' \backslash L}\ \alpha, \overline{\alpha} \notin L \quad \text{(res)}$$

$$\frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}\ \text{(rel)}$$

Exercise: Draw the LTS's

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

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

$$SmUni = (CM|CS)\backslash\{\text{coin}, \text{coffee}\}$$

mCRL2

http://mcrl2.org

- Formal specification language with an associated toolset
- Used for modelling, validating and verifying concurrent systems and protocols

mCRL2

Syntax (by example)

$$a.P \rightarrow$$ `a.P`

$$P_1 + P_2 \rightarrow$$ `P1 + P2`

$$P \backslash L \rightarrow$$ `block(L,P)`

$$P[f] \rightarrow$$ `rename(f,P)`

$$a.P|\overline{a}.Q \rightarrow$$ `hide({a},comm({a1|a2->a},a1.P||a2.P))`

$$(a.P|\overline{a}.Q)\backslash\{a\} \rightarrow$$ `hide({a},block({a1,a2},comm({a1|a2->a},`
`a1.P||a2.Q)))`

mCRL2

```
act
  coin, coin', coinCom,
  coffee, coffee', coffeeCom, pub';

proc
  CM = coin.coffee'.CM;
  CS = pub'.coin'.coffee.CS;
  CMCS = CM || CS;
  SmUni = hide({coffeeCom,coinCom},
          block({coffee,coffee',coin,coin'},
          comm({coffee|coffee' → coffeeCom,
                coin|coin'     → coinCom},
          CMCS )));
init
  SmUni;
```

Example

Clock

```
act    set, alarm, reset;

proc   P = set.R
       R = reset.P + alarm.R

init   P
```

Example

A refined clock

```
act    set:N, alarm, reset, tick;

proc   P = (sum n:N . set(n).R(n)) + tick.P
       R(n:N) = reset.P + ((n == 0) -> alarm.R(0) <> tick.R(n-1))

init   P
```

Parallel composition

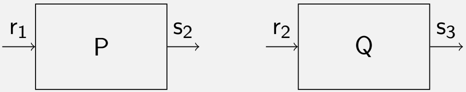
$\|$ = interleaving + synchronization

- modelling principle: interaction is the key element in software design
- modelling principle: (distributed, reactive) architectures are configurations of communicating black boxes
- mCRL2: supports flexible synchronization discipline ($\neq$ CCS)

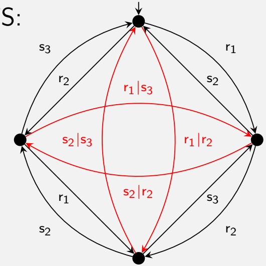
$$p \ ::= \ \cdots \ | \ p \parallel p \ | \ p \,|\, p \ | \ p \,⌊\!\!⌊\, p$$

Parallel composition

Example $P \parallel Q$



Corresponding LTS:



Parallel composition

- parallel $p \parallel q$: interleaves and synchronises the actions of both processes.
- synchronisation $p \mid q$: synchronises the first actions of p and q and combines the remainder of p with q with $\parallel$, cf axiom:

$$(a.p) \mid (b.q) \sim (a \mid b) . (p \parallel q)$$

- left merge $p \mathbin{\|\!\!\lfloor} q$: executes a first action of p and thereafter combines the remainder of p with q with $\parallel$.

Parallel composition

A semantic parentesis

Lemma: There is no sound and complete finite axiomatisation for this process algebra with $\|$ modulo bisimilarity [F. Moller, 1990].

Solution: combine two auxiliar operators:

- left merge: $\llfloor$
- synchronous product: $|$

such that

$$\boxed{p \parallel t \;\sim\; (p \llfloor t + t \llfloor p) + p \mid t}$$

Interaction

Communication $\Gamma_C(p)$ (com)

- applies a communication function C forcing action synchronization and renaming to a new action:

$$a_1 \mid \cdots \mid a_n \;\rightarrow\; c$$

- data parameters are retained in action c, e.g.

$$\begin{aligned}
\Gamma_{\{a|b\rightarrow c\}}(a(8) \mid b(8)) &= c(8) \\
\Gamma_{\{a|b\rightarrow c\}}(a(12) \mid b(8)) &= a(12) \mid b(8) \\
\Gamma_{\{a|b\rightarrow c\}}(a(8) \mid a(12) \mid b(8)) &= a(12) \mid c(8)
\end{aligned}$$

- left hand-sides in C must be disjoint: e.g., $\{a \mid b \rightarrow c, a \mid d \rightarrow j\}$ is not allowed

Interface control

Restriction: $\nabla_B(p)$ (`allow`)

- specifies which multiactions from a non-empty multiset of action names are allowed to occur
- disregards the data parameters of the multiactions

$$\nabla_{\{d,b|c\}}(d(12)+a(8)+(b(\mathit{false},4)\mid c)) \;=\; d(12)+(b(\mathit{false},4)\mid c)$$

- τ is always allowed to occur

Discuss: $\nabla_{\{x,y\}}(\Gamma_{\{a|c->x,b|d->y\}}(a.b \parallel c.d))$

Interface control

Block: $\partial_B(p)$ (`block`)

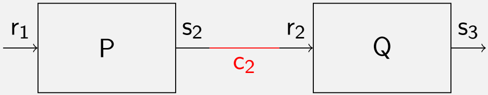
- specifies which multiactions from a set of action names are not allowed to occur
- disregards the data parameters of the multiactions

$$\partial_{\{b\}}(d(12) + a(8) + (b(\mathit{false}, 4) \mid c)) \;=\; d(12) + a(8)$$

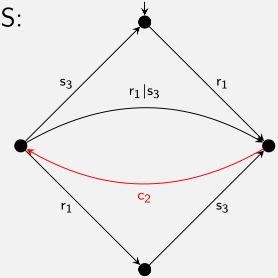
- the effect is that of renaming to δ
- τ cannot be blocked

Interaction

Example $\partial_{r_2,s_2}((\Gamma_{\{s_2|r_2\to c_2\}}(P \parallel Q))$



Corresponding LTS:



Interaction

Enforce communication

- $\nabla_{\{c\}}(\Gamma_{\{a|b \to c\}}(p))$
- $\partial_{\{a,b\}}(\Gamma_{\{a|b \to c\}}(p))$

Interface control

Renaming $\rho_M(p)$ (rename)

- renames actions in p according to a mapping M
- also disregards the data parameters, but when a renaming is applied the data parameters are retained:

$$\begin{aligned}&\partial_{\{d\to h\}}(d(12) + s(8) \mid d(\mathit{false}) + d.a.d(7))\\ &\quad = h(12) + s(8) \mid h(\mathit{false}) + h.a.h(7)\end{aligned}$$

- τ and δ cannot be renamed

Interface control

Hiding $\tau_H(p)$ (hide)

- hides (or renames to τ) all actions with an action name in H in all multiactions of p. renames actions in p according to a mapping M
- disregards the data parameters

$$\tau_{\{d\}}(d(12) + s(8) \mid d(false) + h.a.d(7))$$
$$= \tau + s(8) \mid \tau + h.a.\tau = \tau + s(8) + h.a.\tau$$

Example

New buffers from old

```
act    inn,outt,ia,ib,oa,ob,c :  Bool;

proc   BufferS = sum n:  Bool.inn(n).outt(n).BufferS;

       BufferA = rename({inn -> ia, outt -> oa}, BufferS);
       BufferB = rename({inn -> ib, outt -> ob}, BufferS);

       S = allow({ia,ob,c}, comm({oa|ib -> c}, BufferA || BufferB));

init   hide({c}, S);
```

Data types

- Equalities: equality, inequality, conditional (`if(-,-,-)`)
- Basic types: booleans, naturals, reals, integers, ... with the usual operators
- Sets, multisets, sequences ... with the usual operators
- Function definition, including the λ-notation
- Inductive types: as in

```
sort   BTree = struct leaf(Pos) | node(BTree, BTree)
```

Signatures and definitions

Sorts, functions, constants, variables ...

```
sort   S, A;

cons   s,t:S, b:set(A);

map    f:  S x S -> A;
       c:  A;

var    x:S;

eqn    f(x,s) = s;
```

Signatures and definitions

A full functional language ...

```
sort   LTree = struct leaf(Pos) | node(LTree, LTree);

map    flatten:  LTree -> List(Pos);

var    n:Pos, t,r:LTree;

eqn    flatten(leaf(n)) = [n];
       flatten(node(t,r)) = flatten(t) ++ flatten(r);
```

Processes with data

Why?

- Precise modeling of real-life systems
- Data allows for finite specifications of infinite systems

How?

- data and processes parametrized
- summation over data types: $\sum_{n:N} s(n)$
- processes conditional on data: $b \rightarrow p \diamond q$

Examples

A counter

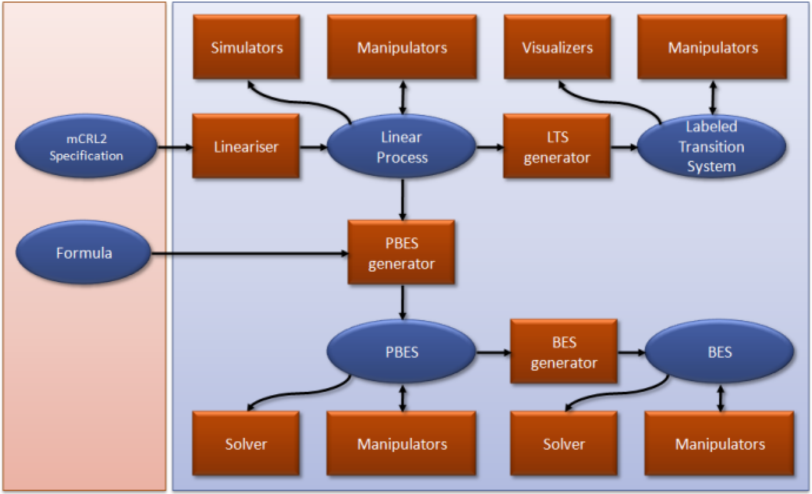
```
act    up, down;
       setcounter:Pos;

proc   Ctr(x:Pos) = up.Ctr(x+1)
              + (x>0) -> down.Ctr(x-1)
              + sum m:Pos.(setcounter(m).Ctr(m))

init   Ctr(345);
```

Examples

A dynamic binary tree

```
act   left,right;
map   N:Pos;
eqn   N = 512;
proc  X(n:Pos)=(n<=N)->(left.X(2*n)+right.X(2*n+1))<>delta;
init  X(1);
```

mCRL2 toolset overview



– mCRL2 tutorial: Modelling part –