Introduction to mCRL2 (modelling)

Luís S. Barbosa

HASLab - INESC TEC
Universidade do Minho
Braga, Portugal

15 April, 2013

mCRL2: A toolset for process algebra

mCRL2 provides:

- a generic process algebra, based on ACP (Bergstra & Klop, 82), in which other calculi can be embedded
- extended with data and (real) time
- with an axiomatic semantics
- the full μ-calculus as a specification logic
- powerful toolset for simulation and verification of reactive systems

`www.mcrl2.org`

Actions

Interaction through multisets of actions

- A multiaction is an elementary unit of interaction that can execute itself atomically in time (no duration), after which it terminates successfully

$$\alpha \quad ::= \quad \tau \;\mid\; a \;\mid\; a(d) \;\mid\; \alpha \mid \alpha$$

- actions may be parametric on data
- the structure $\langle N, |, \tau \rangle$ forms an Abelian monoid

Sequential processes

Sequential, non deterministic behaviour

The set $\mathbb{P}$ of processes is the set of all terms generated by the following BNF, for $a \in N$,

$$p \ ::= \ \alpha \mid \delta \mid p + p \mid p \cdot p \mid \mathsf{P}(d)$$

- atomic process: a for all $a \in N$
- choice: $+$
- sequential composition: $\cdot$
- inaction or deadlock: δ
- process references introduced through definitions of the form $\mathsf{P}(x : D) = p$, parametric on data

Sequential Processes

Exercise

Describe the behaviour of

- $a.b.\delta.c + a$
- $(a + b).\delta.c$
- $(a + b).e + \delta.c$
- $a + (\delta + a)$
- $a.(b + c).d.(b + c)$

Axioms: : $+$, $\cdot$, δ

$A1$	$x + y$	$= y + x$
$A2$	$(x + y) + z$	$= x + (y + z)$
$A3$	$x + x$	$= x$
$A4$	$(x + y).z$	$= x.z + y.z$
$A5$	$(x.y).z$	$= x.(y.z)$
$A6$	$x + \delta$	$= x$
$A7$	$\delta \cdot x$	$= 0$

- the equality relation is sound: if $s = t$ holds for basic process terms, then $s \sim t$
- and complete: if $s \sim t$ holds for basic process terms, then $s = t$
- an axiomatic theory enables equational reasoning

Axioms: : $+$, $\cdot$, δ

$$
\begin{array}{lll}
A1 & x + y & = y + x \\
A2 & (x + y) + z & = x + (y + z) \\
A3 & x + x & = x \\
A4 & (x + y).z & = x.z + y.z \\
A5 & (x.y).z & = x.(y.z) \\
A6 & x + \delta & = x \\
A7 & \delta \cdot x & = 0
\end{array}
$$

- the equality relation is sound: if $s = t$ holds for basic process terms, then $s \sim t$
- and complete: if $s \sim t$ holds for basic process terms, then $s = t$
- an axiomatic theory enables equational reasoning

Axioms: : $+$, $\cdot$, δ

Exercise

- show that $\delta.(a + b) \;=\; \delta \cdot a + \delta \cdot b$
- show that $a + (\delta + a) \;=\; a$
- is it true that $a.(b + c) \;=\; a.b + a.c$?

mCRL2: A toolset for process algebra

Example

```
act    order, receive, keep, refund, return;

proc   Buy = order.OrderedItem

       OrderedItem = receive.ReceivedItem + refund.Buy;
       ReceivedItem = return.OrderedItem + keep;

init   Buy;
```

Deadlock & Termination

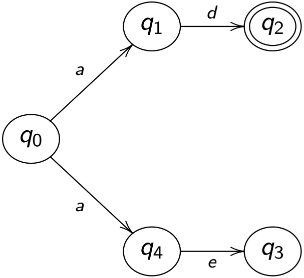
Deadlock state

a reachable state that does not terminate and has no outgoing transitions.

Termination

add a predicate $\downarrow s$ to the definition of a LTS

Termination vs deadlock



Trace equivalence

Trace (from language theory)

A word $\sigma \in N^*$ is a trace of a state $s \in S$ iff there is another state $t \in S$ such that $s \xrightarrow{\sigma}^* t$

Trace (using $\checkmark$ to witness final states)

s, the set of traces of state s, is the minimal set including

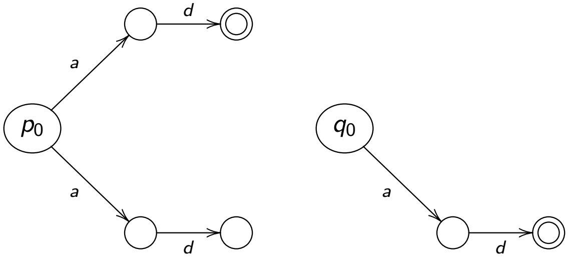
$$
\begin{aligned}
\epsilon &\in s \\
\checkmark &\in s \quad \text{if} \quad \downarrow s \\
a\sigma &\in s \quad \text{if} \quad \exists_t \cdot s \xrightarrow{a} t \wedge \sigma \in t
\end{aligned}
$$

Trace equivalence

Two states are trace equivalent if $s = s'$

Trace equivalence

In any case, fails to preserve deadlock



although preserving sequencing

e.g. before every c an a action b must be done

Language equivalence

Language (from language theory)

A word $\sigma \in N^*$ is a run (or a complete trace) of a state $s \in S$ iff there is another state $t \in S$, such that $s \xrightarrow{\sigma}^* t$ and $\downarrow t$. The language recognized by a state $s \in S$ is the set of runs of s

Language (using ✓ to witness final states)

s, the language recognized by a state s, is the minimal set including

$$
\begin{aligned}
\epsilon \in s & \quad \text{if } s \text{ is a deadlock state} \\
\checkmark \in s & \quad \text{if } \downarrow s \\
a\sigma \in s & \quad \text{if } \exists_t \cdot s \xrightarrow{a} t \wedge \sigma \in t
\end{aligned}
$$

Language equivalence

Two states are language equivalent if $s = s'$, i.e., if both recognize the same language.

... need more general models and theories:

- Several interaction points
- Need to distinguish normal from anomolous termination
- Non determinisim should be taken seriously: the notion of equivalence based on accepted language is blind wrt non determinism
- Moreover: the reactive character of systems entail that not only the generated language is important, but also the states traversed during an execution of the automata.

Notes

The Van Glabbeek linear - branching time spectrum

$$\begin{tikzcd}
 & \textit{Bisimilary} \arrow[dl] \arrow[dr] & \\
\dots \arrow[dr] & & \dots \arrow[dl] \\
 & \textit{Language Eq} \arrow[d] & \\
 & \textit{Trace Eq} &
\end{tikzcd}$$

... collapses for deterministic transition systems: why?

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

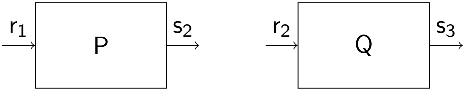
$\|$ = interleaving + synchronization

- 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)

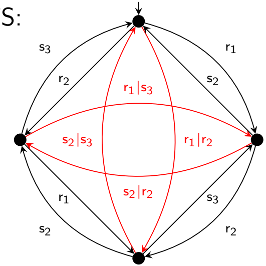
$$p ::= \cdots \mid p \parallel p \mid p \,|\, p \mid p \mathbin{⌊\!\!⌊} p$$

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 $\parallel$ modulo bisimilarity [F. Moller, 1990].

Solution: combine two auxiliar operators:

- left merge: $\lfloor\!\lfloor$
- synchronous product: $|$

such that

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

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.

$$\Gamma_{\{a|b\rightarrow c\}}(a(8) \mid b(8)) \;=\; c(8)$$
$$\Gamma_{\{a|b\rightarrow c\}}(a(12) \mid b(8)) \;=\; a(12) \mid b(8)$$
$$\Gamma_{\{a|b\rightarrow c\}}(a(8) \mid a(12) \mid b(8)) \;=\; a(12) \mid c(8)$$

- 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

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

- 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`)

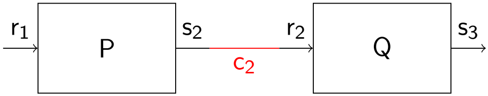
- specifies which multiactions from a set of action names are not allowed to occur
- disregards the data parameters of the multiactions

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

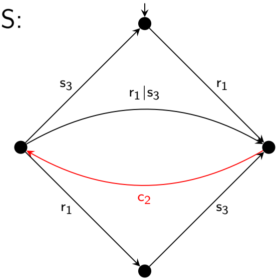
- the effect is that of renaming to δ
- τ cannot be blocked

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:

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

- τ 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

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

- τ and δ cannot be renamed

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   BTree = struct leaf(Pos) | node(BTree, BTree);

map    flatten:  BTree -> List(Pos);

var    n:Pos, t,r:BTree;

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: $b \to p \diamond q$

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);
```

Mini-project: Part 1

Aim: becoming proficient in process modelling

- Choose examples from the exercises sheets
- Model and simulate in mCRL2

To follow

- Specify relevant properties in a process logic
- Verify them in mCRL2
- Investigate other features of mCRL2 (e.g., time, semantics, ...)