Introduction to mCRL2: Process modelling

Luís S. Barbosa

HASLab - INESC TEC
Universidade do Minho
Braga, Portugal

11 April 2014

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
- 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

$$\alpha \quad \ni \quad \tau \mid a(d) \mid (\alpha \mid \alpha)$$

- 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 $a \in N$,

$$p \ni \alpha \mid \delta \mid p + p \mid p \cdot p \mid \mathrm{P}(d)$$

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

Example

Buffers

```
act    in, out, t; inn, outt :  Bool;

proc   Buffer1 = in.out;

       Buffer2 = in.out.Buffer2;

       Buffer3 = in.(out.Buffer3 + t.Buffer3);

       Buffer4 = sum n:  Bool.inn(n).outt(n).Buffer4;
```

Sequential Processes

Exercise

Describe the behaviour of

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

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 a flexible synchronization discipline

$$p ::= \cdots \mid p \parallel p \mid p \mid p \mid p \between p$$

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 $p \mathbin{\lfloor\!\lfloor} q$: 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: $\lfloor\!\lfloor$
- synchronous product: $|$

such that

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

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

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

- τ is always allowed to occur

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

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

- τ cannot be blocked

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:

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

- 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 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);
```

Exercise

Composing buffers with acknowledges

```
act    inn, outt, r, t, ia, ib, oa, ob, ta, tb, ra, rb, c, a;

proc   BufferS = inn.outt.r.t.BufferS;

       BufferA =
         rename({inn -> ia, outt -> oa, r -> ra, t -> ta}, BufferS);
       BufferB =
         rename({inn -> ib, outt -> ob, r -> rb, t -> tb}, BufferS);

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

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

Exercise

Composing buffers with acknowledges (corrected)

```
act    inn, outt, r, t, ia, ib, oa, ob, ta, tb, ra, rb, c, a;

proc   BufferS = inn.t.outt.r.BufferS;

       BufferA =
         rename({inn -> ia, outt -> oa, r -> ra, t -> ta}, BufferS);
       BufferB =
         rename({inn -> ib, outt -> ob, r -> rb, t -> tb}, BufferS);

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

init   hide({c,a}, 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)) = t++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);
```

Overview

Strategies to deal with infinite models and specifications

- A specification of the system's behaviour is written in mCRL2 (`x.mcrl2`)
- The specification is converted to a stricter format called Linear Process Specification (`x.lps`)
- In this format the specification can be transformed and simulated
- In particular a Labelled Transition System (`x.lts`) can be generated, simulated and analysed through symbolic model checking (boolean equation solvers)

Architecture