Introduction to mCRL2 (modelling)

Luís Soares Barbosa

Universidade do Minho

HASLab HIGH-ASSURANCE SOFTWARE LABORATORY

UNITED NATIONS UNIVERSITY UNU-EGOV

Interaction & Concurrency Course Unit (Lcc)

Universidade do Minho, 16.III.2018

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 (it cannot even to terminate!)
- process references introduced through definitions of the form $\mathsf{P}(x : D) = p$, parametric on data

Sequential Processes

Exercise

Describe the behaviour of

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

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;
```

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

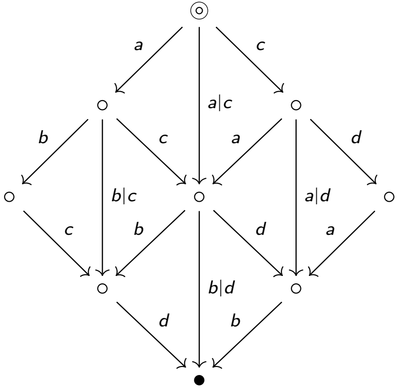
$\| =$ interleaving $+$ synchronization

- 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)

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

Parallel composition

An example



$$\boxed{a \cdot b \parallel c \cdot d}$$

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 $p \mathbin{⌊\!⌊} q$: executes a first action of p and thereafter combines the remainder of p with q with $\parallel$.

Parallel composition

A semantic parenthesis

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

Solution: combine two auxiliar operators:

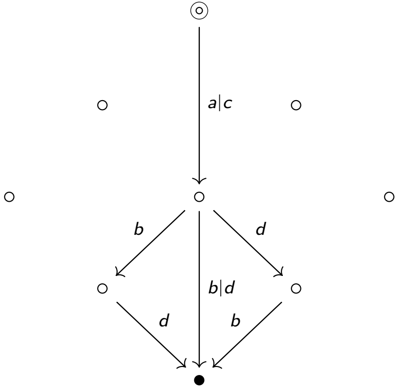
- left merge: $\lfloor\!\lfloor$
- synchronous product: $|$

such that

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

Parallel composition

An example



$$\boxed{a \cdot b \mid c \cdot d}$$

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 actions are allowed to occur
- disregards the data parameters of actions

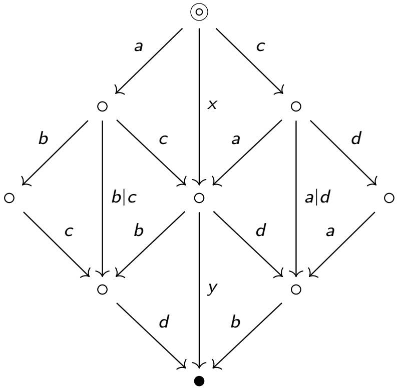
$$\nabla_{\{d,b|c\}}(d(12) + a(8) + (b(false,4) \mid c)) \;=\; d(12) + (b(false,4) \mid c)$$

- τ is always allowed to occur

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

Interface control

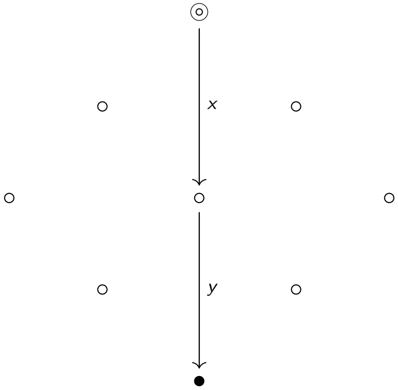
An example



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

Interface control

An example



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

Interface control

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

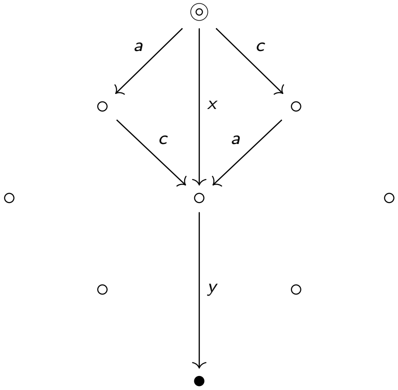
- specifies which actions are not allowed to occur
- disregards the data parameters of actions

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

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

Interface control

An example



$$\boxed{\partial_{\{b,d\}}(\Gamma_{\{b|d->y\}}(a.b \parallel c.d))}$$

Interface control

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 values of data parameters are retained:

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

- τ cannot be renamed

Interface control

Hiding $\tau_H(p)$ (hide)

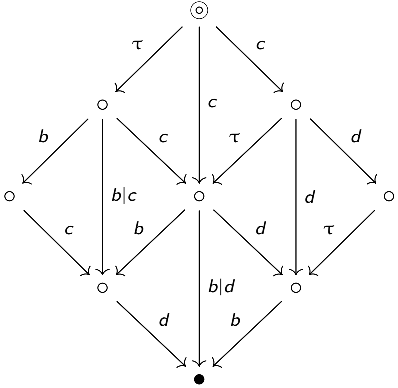
- hides (or renames to τ) all actions in H in all multiactions of p.
- disregards the data parameters

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

- τ and δ cannot be renamed

Interface control

An example



$$\boxed{\tau_{\{a\}}(\Gamma_{\{b|d->y\}}(a.b \parallel c.d))}$$

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: $b \rightarrow p \diamond q$

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);
```