

Introduction to mCRL2

Luís S. Barbosa

DI-CCTC
Universidade do Minho
Braga, Portugal

May, 2011

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.mcr12.org

mCRL2: A toolset for process algebra

Our aim

- To use mCRL2 to animate CCS models and verify modal and temporal properties
- To introduce a method and a language to describe [software architectures](#) on top of mCRL2

This lecture provides an [overview](#) and a [demo](#)
Refer to recommended reading for [semantics](#)
and the toolset [algorithms](#)

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

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

- actions may be parametric on **data**
- the structure $\langle \mathcal{N}, |, \tau \rangle$ 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 $a \in \mathcal{N}$,

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

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

Sequential Processes

Exercise

Describe the behaviour of

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

Parallel composition

\parallel = 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)

$$p ::= \dots \mid p \parallel p \mid p \mid p \mid p \parallel p$$

Parallel composition

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

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

- **left merge** $p \ll q$: 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 \parallel modulo bisimilarity [F. Moller, 1990].

Solution: combine two auxiliary operators:

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

such that

$$p \parallel t \sim (p \ll t + t \ll p) + p | t$$

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

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

- τ is always allowed to occur

Interface control

Block: $\partial_B(\rho)$ (block)

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

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

- the effect is that of renaming to δ
- 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:

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

- 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

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

- 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}, comm({oa|ib -> c}, BufferA || BufferB));

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

Data types

- **Equalities:** equality, inequality, conditional ($\text{if}(-,-,-)$)
- **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: $b \rightarrow p \diamond q$

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);
```

Overview

The verification problem

- Given a specification of the system's behaviour is in mCRL2
- and the system's requirements are specified as properties in a temporal logic,
- a model checking algorithm decides whether the property holds for the model: the property can be verified or refuted;
- sometimes, witnesses or counter examples can be provided

Which logic?

μ -calculus with data, time and regular expressions

From modal logic ...

Hennessy-Milner logic

... propositional logic with **action** modalities

$$\phi ::= \text{true} \mid \text{false} \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \langle a \rangle \phi \mid [a]\phi$$

Laws

$$\neg \langle a \rangle \phi = [a] \neg \phi$$

$$\neg [a] \phi = \langle a \rangle \neg \phi$$

$$\langle a \rangle \text{false} = \text{false}$$

$$[a] \text{true} = \text{true}$$

$$\langle a \rangle (\phi \vee \psi) = \langle a \rangle \phi \vee \langle a \rangle \psi$$

$$[a] (\phi \wedge \psi) = [a] \phi \wedge [a] \psi$$

$$\langle a \rangle \phi \wedge [a] \psi \Rightarrow \langle a \rangle (\phi \wedge \psi)$$

From modal logic ...

Hennessy-Milner logic + regular expressions

ie, with regular expressions within modalities

$$\rho ::= \epsilon \mid \alpha \mid \rho.\rho \mid \rho + \rho \mid \rho^* \mid \rho^+$$

where

- α is an **action formula** and ϵ is the **empty word**
- **concatenation** $\rho.\rho$, **choice** $\rho + \rho$ and **closures** ρ^* and ρ^+

Laws

$$\langle \rho_1 + \rho_2 \rangle \phi = \langle \rho_1 \rangle \phi \vee \langle \rho_2 \rangle \phi$$

$$[\rho_1 + \rho_2] \phi = [\rho_1] \phi \wedge [\rho_2] \phi$$

$$\langle \rho_1.\rho_2 \rangle \phi = \langle \rho_1 \rangle \langle \rho_2 \rangle \phi$$

$$[\rho_1.\rho_2] \phi = [\rho_1][\rho_2] \phi$$

From modal logic ...

Action formulas

$$\alpha ::= a_1 \mid \cdots \mid a_n \mid \text{true} \mid \text{false} \mid \neg\alpha \mid \alpha \cup \alpha \mid \alpha \cap \alpha$$

where

- $a_1 \mid \cdots \mid a_n$ is a set with this single multiaction
- true (universe), false (empty set)
- $\neg\alpha$ is the set complement

Modalities with action formulas:

$$\langle \alpha \rangle \phi = \bigvee_{a \in \alpha} \langle a \rangle \phi \quad [\alpha] \phi = \bigwedge_{a \in \alpha} [a] \phi$$

... to temporal logic

Examples of properties

- $\langle \epsilon \rangle \phi = [\epsilon] \phi = \phi$
- $\langle a.a.b \rangle \phi = \langle a \rangle \langle a \rangle \langle b \rangle \phi$
- $\langle a.b + g.d \rangle \phi$

Safety

- $[\text{true}^*] \phi$
- it is impossible to do two consecutive enter actions without a leave action in between:
 $[\text{true}^*.enter. - leave^*.enter] \text{false}$
- absence of **deadlock**:
 $[\text{true}^*] \langle \text{true} \rangle \text{true}$

... to temporal logic

Examples of properties

Liveness

- $\langle \text{true}^* \rangle \phi$
- after sending a message, it can eventually be received:
 $[\text{send}] \langle \text{true}^* . \text{receive} \rangle \text{true}$
- after a send a receive is possible as long as it has not happened:
 $[\text{send} . - \text{receive}^*] \langle \text{true}^* . \text{receive} \rangle \text{true}$

... to temporal logic

The modal μ -calculus

- modalities with regular expressions are not enough in general
- ... but correspond to a subset of the modal μ -calculus [Kozen83]

Add explicit **minimal/maximal fixed point operators** to Hennessy- Milner logic

$$\phi ::= X \mid \text{true} \mid \text{false} \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \Rightarrow \phi \mid \langle a \rangle \phi \mid [a] \phi \mid \mu X . \phi \mid \nu X . \phi$$

... to temporal logic

The modal μ -calculus (intuition)

- $\mu X . \phi$ is valid for all those states in the **smallest** set X that satisfies the equation $X = \phi$ (finite paths, **liveness**)
- $\nu X . \phi$ is valid for the states in the **largest** set X that satisfies the equation $X = \phi$ (infinite paths, **safety**)

Warning

In order to be sure that a fixed point exists, X must occur positively in the formula, ie **preceded by an even number of negations**.

... to temporal logic

Laws & Notes (but see the μ -calculus slides!)

$$\mu X . \phi \Rightarrow \nu X . \phi$$

and self-duals:

$$\neg \mu X . \phi = \nu X . \neg \phi$$

$$\neg \nu X . \phi = \mu X . \neg \phi$$

Translation of regular formulas with closure

$$\langle R^* \rangle \phi = \mu X . \langle R \rangle X \vee \phi$$

$$[R^*] \phi = \nu X . [R] X \wedge \phi$$

$$\langle R^+ \rangle \phi = \langle R \rangle \langle R^* \rangle \phi$$

$$[R^+] \phi = [R][R^*] \phi$$

Example: The dining philosophers problem

Formulas to verify Demo

- No deadlock (every philosopher holds a left fork and waits for a right fork (or vice versa):

$$[\text{true}^*] \langle \text{true} \rangle \text{true}$$

- No starvation (a philosopher cannot acquire 2 forks):

$$\text{forall } p:\text{Phil}. [\text{true}^*. !\text{eat}(p)^*] \langle !\text{eat}(p)^*. \text{eat}(p) \rangle \text{true}$$

- A philosopher can only eat for a finite consecutive amount of time:

$$\text{forall } p:\text{Phil}. \nu X. \mu Y. [\text{eat}(p)]Y \ \&\& \ [!\text{eat}(p)]X$$

- there is no starvation: for all reachable states it should be possible to eventually perform an $\text{eat}(p)$ for each possible value of $p:\text{Phil}$.

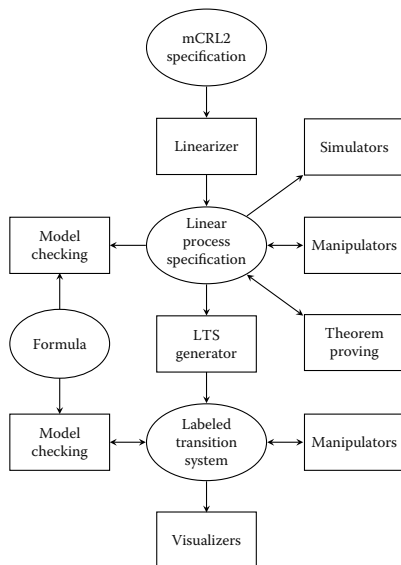
$$[\text{true}^*](\text{forall } p:\text{Phil}. \mu Y. ([!\text{eat}(p)]Y \ \&\& \ \langle \text{true} \rangle \text{true}))$$

Overview

Strategies to deal with infinite models and specifications

- A specification of the system's behaviour is written in mCRL2 (`x.mcr12`)
- 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



Mini-project

Aim: becoming proficient in mCRL2

- Choose examples from the exercises sheets
- Model and simulate in mCRL2
- Specify relevant properties and test them
- ... within 2 weeks