Introduction to process algebra and the μ-calculus

Luís S. Barbosa

HASLab - INESC TEC
Universidade do Minho
Braga, Portugal

3 April 2014

From LTS to processes

- We already have a semantic model for reactive systems. With which language shall we describe them?
- How to compare and transform such systems?
- How to express and prove their proprieties?

$\rightsquigarrow$ process languages and calculi
cf. CCS (Milner, 80), CSP (Hoare, 85),
ACP (Bergstra & Klop, 82),
π-*calculus* (Milner, 89), among many others

$\rightsquigarrow$ modal (temporal, hybrid) logics

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

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

- 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 $a \in N$,

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

- 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

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

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

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 $p \mathbin{⌊\!\!⌊} q$: 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

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

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.

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

- 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

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

- 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:

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

- 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

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

- 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: $b \rightarrow p \diamond q$

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);
```

Motivation

System's correctness wrt a specification

- equivalence checking (between two designs), through $\sim$ and $=$
- unsuitable to check properties such as

can the system perform action a followed by b?

which are best answered by exploring the process state space

Which logic?

A modal logic with the ability to express enduring (temporal) properties

Motivation

The taxi network example

- $\phi_0 =$ *In a taxi network, a car can collect a passenger or be allocated by the Central to a pending service*
- $\phi_1 =$ *This applies only to cars already on service*
- $\phi_2 =$ *If a car is allocated to a service, it must first collect the passenger and then plan the route*
- $\phi_3 =$ *On detecting an emergence the taxi becomes inactive*
- $\phi_4 =$ *A car on service is not inactive*

Motivation

The taxi network example

- $\phi_0 = \langle rec, alo \rangle \mathrm{true}$
- $\phi_1 = [onservice]\langle rec, alo \rangle \mathrm{true}$ or
  $\phi_1 = [onservice]\phi_0$
- $\phi_2 = [alo]\langle rec \rangle \langle plan \rangle \mathrm{true}$
- $\phi_3 = [sos][\mathrm{true}]\mathrm{false}$
- $\phi_4 = [onservice]\langle \mathrm{true} \rangle \mathrm{true}$

... in mCRL2

The verification problem in mCRL2

- 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

Modal logic

- Modalities: $\langle - \rangle \phi$, $[-]\psi$
- Valuations in non modal logics are based on valuations V : Variables $\longrightarrow$ **2**: propositions are true or false depending on the unique referential provided by V
- Valuations in a modal logic also depends on the current state of computation: V : Variables $\times \mathbb{P} \longrightarrow$ **2** or, equivalently, , V : Variables $\longrightarrow \mathcal{P}\mathbb{P}$: each variable is associated to the set of processes in which its value is fixed as true
- In our case, models for such a logic are defined over the universe of processes $\mathbb{P}$ (*i.e.*, terms of our process language) equipped with relations $\{\stackrel{x}{\longrightarrow} | x \in Act\}$ defined by the operational semantics of the language.
- ... but the topic modal logics has a longer story and a broad spectrum of applications ...

The Hennessy-Milner logic

Syntax

$$\phi \ni \mathsf{true} \mid \mathsf{false} \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \langle\alpha\rangle\phi \mid [\alpha]\phi$$

where α is an action formula

Compare with dynamic logic

Can you spot the difference?

The Hennessy-Milner logic

Some laws

$$
\begin{aligned}
\neg\langle a\rangle\phi &= [a]\neg\phi\\
\neg[a]\phi &= \langle a\rangle\neg\phi\\
\langle a\rangle\mathsf{false} &= \mathsf{false}\\
[a]\mathsf{true} &= \mathsf{true}\\
\langle a\rangle(\phi\vee\psi) &= \langle a\rangle\phi\vee\langle a\rangle\psi\\
[a](\phi\wedge\psi) &= [a]\phi\wedge[a]\psi\\
\langle a\rangle\phi\wedge[a]\psi &\Rightarrow \langle a\rangle(\phi\wedge\psi)
\end{aligned}
$$

The Hennessy-Milner logic

Action formulas

$$\alpha \ni (a_1 \mid \cdots \mid a_n) \mid \mathsf{true} \mid \mathsf{false} \mid -\alpha \mid \alpha \cup \alpha \mid \alpha \cap \alpha$$

where

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

Modalities with action formulas:

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

The language

Semantics: $E \models \phi$

$$
\begin{array}{lll}
E \models \mathrm{true} & & \\
E \not\models \mathrm{false} & & \\
E \models \neg\phi & \text{iff} & E \not\models \phi \\
E \models \phi \wedge \psi & \text{iff} & E \models \phi \;\wedge\; E \models \psi \\
E \models \phi \vee \psi & \text{iff} & E \models \phi \;\vee\; E \models \psi \\
E \models \langle\alpha\rangle\phi & \text{iff} & \exists_{F \in \{E' \mid E \xrightarrow{a} E' \,\wedge\, a \in \alpha\}} \,.\, F \models \phi \\
E \models [\alpha]\phi & \text{iff} & \forall_{F \in \{E' \mid E \xrightarrow{a} E' \,\wedge\, a \in \alpha\}} \,.\, F \models \phi
\end{array}
$$

Notes

- inevitability of a: $\langle \text{true} \rangle \text{true} \wedge [-a]\text{false}$
- progress: $\langle \text{true} \rangle \text{true}$
- deadlock or termination: $[\text{true}]\text{false}$
- what about

$$\langle \text{true} \rangle \text{false} \quad \text{and} \quad [\text{true}]\text{true} \quad ?$$

- satisfaction decided by unfolding the definition of $\models$: no need to compute the transition graph

A denotational semantics

Idea: associate to each formula ϕ the set of processes that make it true

ϕ vs $\|\phi\| = \{E \in \mathbb{P} \mid E \models \phi\}$

$$\begin{aligned}
\|\text{true}\| &= \mathbb{P} \\
\|\text{false}\| &= \emptyset \\
\|\phi \wedge \psi\| &= \|\phi\| \cap \|\psi\| \\
\|\phi \vee \psi\| &= \|\phi\| \cup \|\psi\|
\end{aligned}$$

$$\begin{aligned}
\|[\alpha]\phi\| &= \|[\alpha]\|(\|\phi\|) \\
\|\langle\alpha\rangle\phi\| &= \|\langle\alpha\rangle\|(\|\phi\|)
\end{aligned}$$

$\|[\alpha]\|$ and $\|\langle\alpha\rangle\|$

Just as $\wedge$ corresponds to $\cap$ and $\vee$ to $\cup$, modal logic combinators correspond to unary functions on sets of processes:

$$\|[\alpha]\| \;=\; \lambda_{X \subseteq \mathbb{P}} \,.\, \{F \in \mathbb{P} \mid \text{if } F \xrightarrow{a} F' \;\wedge\; a \in \alpha \;\text{ then }\; F' \in X\}$$

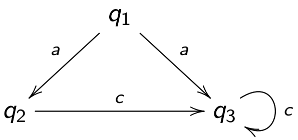
$$\|\langle\alpha\rangle\| \;=\; \lambda_{X \subseteq \mathbb{P}} \,.\, \{F \in \mathbb{P} \mid \exists_{F' \in X, a \in \alpha} \,.\, F \xrightarrow{a} F'\}$$

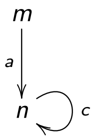
Note

These combinators perform a reduction to the previous state indexed by actions in α

$\|[\alpha]\|$ and $\|\langle\alpha\rangle\|$

Example





$$
\begin{aligned}
\|\langle a\rangle\|\{q_2, n\} &= \{q_1, m\} \\
\|[a]\|\{q_2, n\} &= \{q_2, q_3, m, n\}
\end{aligned}
$$

A denotational semantics

$$\boxed{E \models \phi \ \text{ iff } \ E \in \|\phi\|}$$

Example: $\delta \models [\mathsf{true}]\mathsf{false}$

because

$$\begin{aligned}
\|[\mathsf{true}]\mathsf{false}\| &= \|[\mathsf{true}]\|(\|\mathsf{false}\|) \\
&= \|[\mathsf{true}]\|(\emptyset) \\
&= \{F \in \mathbb{P} \mid \text{if } F \xrightarrow{x} F' \ \wedge \ x \in Act \ \text{ then } \ F' \in \emptyset\} \\
&= \{\delta\}
\end{aligned}$$

A denotational semantics

$$\boxed{E \models \phi \;\text{ iff }\; E \in \|\phi\|}$$

Example: ?? $\models \langle \text{true} \rangle \text{true}$

because

$$\begin{aligned} \|\langle \text{true} \rangle \text{true}\| &= \|\langle \text{true} \rangle\|(\|\text{true}\|) \\ &= \|\langle \text{true} \rangle\|(\mathbb{P}) \\ &= \{F \in \mathbb{P} \mid \exists_{F' \in \mathbb{P}, a \in K} \, . \, F \xrightarrow{a} F'\} \\ &= \mathbb{P} \setminus \{\delta\} \end{aligned}$$

Modal Equivalence

For each (finite or infinite) set Γ of formulae,

$$E \simeq_\Gamma F \quad \Leftrightarrow \quad \forall_{\phi \in \Gamma} \,.\, E \models \phi \Leftrightarrow F \models \phi$$

Examples

$$a.b + a.c \;\simeq_\Gamma\; a.(b + c)$$

for $\Gamma \;=\; \{\langle x_1\rangle\langle x_2\rangle ... \langle x_n\rangle \mathrm{true} \mid x_i \in Act\}$

(what about $\simeq_\Gamma$ for $\Gamma = \{\langle x_1\rangle\langle x_2\rangle\langle x_3\rangle ... \langle x_n\rangle [\mathrm{true}] \mathrm{false} \mid x_i \in Act\}$?)

Modal Equivalence

For each (finite or infinite) set Γ of formulae,

$$E \simeq F \quad \Leftrightarrow \quad E \simeq_{\Gamma} F \text{for every set } \Gamma \text{ of well-formed formulae}$$

Lemma

$$E \sim F \quad \Rightarrow \quad E \simeq F$$

Note

the converse of this lemma does not hold, e.g. let

- $A \triangleq \sum_{i \geq 0} A_i$, where $A_0 \triangleq \mathbf{0}$ and $A_{i+1} \triangleq a.A_i$
- $A' \triangleq A + \underline{fix}\ (X = a.X)$

$$A \nsim A' \quad \text{but} \quad A \simeq A'$$

Modal Equivalence

Theorem [Hennessy-Milner, 1985]

$$E \sim F \quad \Leftrightarrow \quad E \simeq F$$

for image-finite processes.

Image-finite processes

E is image-finite iff $\{F \mid F \xrightarrow{a} E\}$ is finite for every action $a \in Act$

Modal Equivalence

Theorem [Hennessy-Milner, 1985]

$$E \sim F \quad \Leftrightarrow \quad E \simeq F$$

for image-finite processes.

proof

$\Rightarrow$: by induction of the formula structure

$\Leftarrow$: show that $\simeq$ is itself a bisimulation, by contradiction

Regular modalities

Hennessy-Milner logic + regular expressions

ie, add 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

$$\begin{aligned}
\langle \rho_1 + \rho_2 \rangle \phi &= \langle \rho_1 \rangle \phi \vee \langle \rho_2 \rangle \phi \\
[\rho_1 + \rho_2] \phi &= [\rho_1] \phi \wedge [\rho_2] \phi \\
\langle \rho_1 . \rho_2 \rangle \phi &= \langle \rho_1 \rangle \langle \rho_2 \rangle \phi \\
[\rho_1 . \rho_2] \phi &= [\rho_1][\rho_2] \phi
\end{aligned}$$

Regular modalities

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

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

Regular modalities

Examples of properties

Liveness

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

The modal μ-calculus

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

- modalities with regular expressions are not enough in general
- in particular cannot express fairness properties:

  if the system is offered the possibility to perform *a* infinitely often,
  then it will eventually perform *a*

- ... but correspond to a subset of the modal μ-calculus [Kozen83]

The modal μ-calculus

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.

Examples

Translation of regular formulas with closure

$$
\begin{aligned}
\langle R^* \rangle \phi &= \mu X \,.\, \langle R \rangle X \vee \phi \\
[R^*]\phi &= \nu X \,.\, [R]X \wedge \phi \\
\langle R^+ \rangle \phi &= \langle R \rangle \langle R^* \rangle \phi \\
[R^+]\phi &= [R][R^*]\phi
\end{aligned}
$$

Examples

The dining philosophers problem

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

```
[true*]<true>true
```

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

```
forall p:Phil. [true*.!eat(p)*] <!eat(p)*.eat(p)>true
```

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

```
forall p:Phil. nu X. mu Y. [eat(p)]Y && [!eat(p)]X
```

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

```
[true*](forall p:Phil. mu Y. ([!eat(p)]Y && <true>true))
```

Semantics

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

cf the Knaster-Tarski theorem (1928)

Laws

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

and self-duals:

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

A denotational semantics

$\boxed{\rho : X \longrightarrow \mathcal{P}\mathbb{P}\text{: predicate environment}}$

$$\begin{aligned}
\|\mathsf{true}\|_\rho &= \mathbb{P} \\
\|\mathsf{false}\|_\rho &= \emptyset \\
\|X\|_\rho &= \rho(X) \\
\|\phi \wedge \psi\|_\rho &= \|\phi\|_\rho \cap \|\psi\|_\rho \\
\|\phi \vee \psi\|_\rho &= \|\phi\|_\rho \cup \|\psi\|_\rho \\
\|[\alpha]\phi\|_\rho &= \|[\alpha]\|(\|\phi\|_\rho) \\
\|\langle\alpha\rangle\phi\|_\rho &= \|\langle\alpha\rangle\|(\|\phi\|_\rho)
\end{aligned}$$

$$\|\mu X \,.\, \phi\|_\rho = \bigcap\{V \in \mathbb{P} \mid \|\phi\|_{\rho\{X \mapsto V\}} \subseteq V\}$$

$$\|\nu X \,.\, \phi\|_\rho = \bigcup\{V \in \mathbb{P} \mid V \subseteq \|\phi\|_{\rho\{X \mapsto V\}}\}$$

computing by Kleene approximation

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