Behavioural Analysis using mCRL2

Aad Mathijssen Bas Ploeger Frank Stappers
Tim Willemse

Department of Mathematics and Computer Science
Technische Universiteit Eindhoven

IPA Course on Formal Methods
Technische Universiteit Eindhoven

June 26, 2008

Introduction

Analysis techniques

Main analysis techniques used in hardware/software development:

- Structural analysis: what *things* are in the system
  - Class diagrams
  - Component diagrams
  - Package diagrams
- Behavioural analysis: what *happens* in the system
  - State diagrams
  - Message sequence charts
  - Petri nets
  - Process algebra
  - Temporal logic

Introduction

Schedule

10:00 - 11:00	Basic process algebra Parallelism and abstraction Processes with data
11:00 - 11:15	*Break*
11:15 - 12:15	Linear processes Temporal logic Verification
12:15 - 13:15	*Lunch*
13:15 - 13:45	Toolset overview and demo
13:45 - 14:15	Hands-on experience
14:15 - 14:30	*Break*
14:30 - 15:30	Hands-on experience
15:30 - 15:45	*Break*
15:45 - 16:15	Wrap-up Industrial case studies

Outline

1. Basic process algebra
2. Parallelism and abstraction
3. Processes with data
4. Linear processes
5. Temporal Logic
6. Verification
7. Toolset overview and demo
8. Hands-on experience
9. Wrap-up
10. Industrial case studies

Outline

Labelled transition systems

Introduction

A labelled transition system is a basic formalism for describing behaviour.

Also known as labelled directed graphs or state spaces.

Labels represent discrete events, also called actions.

Labelled transition systems

Formal definition

A labelled transition system is a tuple $(S, \mathcal{L}, \rightarrow, s, T)$ where:

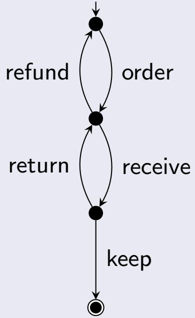
- S is a set of states
- $\mathcal{L}$ is a set of labels
- $\rightarrow \subseteq S \times \mathcal{L} \times S$ is a transition relation
- $s \in S$ is the initial state
- $T \subseteq S$ is the set of terminating states

Labelled transition systems

Example: order items

Example

Ordering items:



Basic process algebra

Motivation

Labelled transition systems are powerful, but low-level.

Basic process algebra allows us to:

- describe labelled transition systems at an abstract level
- reason about labelled transition systems using equations

Basic process algebra

Describe behaviour

Basic processes: $p ::= \mathsf{a} \mid p \cdot p \mid p + p \mid \delta$

- $\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots$ represent actions
- $p \cdot q$ represents sequential composition
- $p + q$ represents non-deterministic choice
- δ represents inaction or deadlock

Operator precedence:

- $\cdot$ binds stronger than $+$
- $\cdot$ and $+$ associate to the right
- Use parentheses to override
- For example: $\mathsf{a} \cdot \mathsf{b} + \mathsf{c} \cdot \mathsf{d} \cdot \mathsf{e}$ stands for $(\mathsf{a} \cdot \mathsf{b}) + (\mathsf{c} \cdot (\mathsf{d} \cdot \mathsf{e}))$

Basic process algebra

Describe behaviour

Exercise: draw LTSs

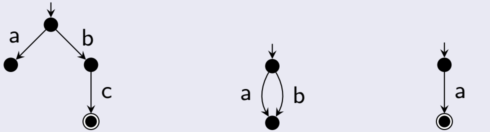
$$a \cdot \delta + b \cdot c \qquad (a + b) \cdot \delta \cdot c \qquad a + \delta$$

Basic process algebra

Describe behaviour

Exercise: draw LTSs

$\mathsf{a} \cdot \delta + \mathsf{b} \cdot \mathsf{c}$ $(\mathsf{a} + \mathsf{b}) \cdot \delta \cdot \mathsf{c}$ $\mathsf{a} + \delta$



Basic process algebra

Describe behaviour

Exercise: draw LTSs

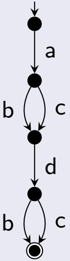
$a \cdot (b + c) \cdot d \cdot (b + c)$ $a \cdot (b \cdot d \cdot (b + c) + c \cdot d \cdot (b + c))$

Basic process algebra

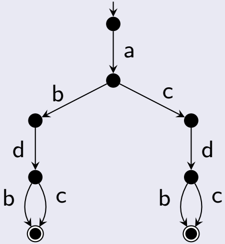
Describe behaviour

Exercise: draw LTSs

$a \cdot (b + c) \cdot d \cdot (b + c)$



$a \cdot (b \cdot d \cdot (b + c) + c \cdot d \cdot (b + c))$

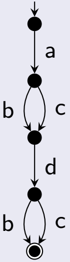


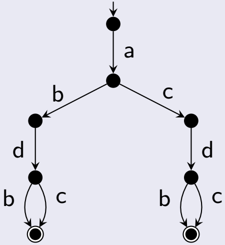
Basic process algebra

Describe behaviour

Exercise: draw LTSs

$a \cdot (b + c) \cdot d \cdot (b + c)$ $a \cdot (b \cdot d \cdot (b + c) + c \cdot d \cdot (b + c))$





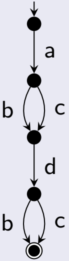
Are the two equivalent?

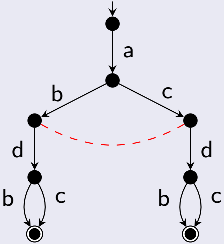
Basic process algebra

Describe behaviour

Exercise: draw LTSs

$a \cdot (b + c) \cdot d \cdot (b + c)$ $\quad$ $a \cdot (b \cdot d \cdot (b + c) + c \cdot d \cdot (b + c))$





Are the two equivalent? Yes!

Basic process algebra

Reason about behaviour: derivation rules

Derivation rules for process equality:

$$\frac{}{p = p} \qquad \frac{p = q}{q = p} \qquad \frac{p = q \qquad q = r}{p = r}$$

$$\frac{p_1 = q_1 \qquad p_2 = q_2}{p_1 \cdot p_2 = q_1 \cdot q_2} \qquad \frac{p_1 = q_1 \qquad p_2 = q_2}{p_1 + p_2 = q_1 + q_2}$$

$$\frac{p = q \ \in \ Ax}{p = q}$$

Basic process algebra

Reason about behaviour: axioms

Axioms for the basic operators:

A1 $p + q = q + p$

A2 $p + (q + r) = (p + q) + r$

A3 $p + p = p$

A4 $(p + q) \cdot r = p \cdot r + q \cdot r$

A5 $(p \cdot q) \cdot r = p \cdot (q \cdot r)$

A6 $\mathsf{a} + \delta = \mathsf{a}$

A7 $\delta \cdot p = \delta$

Basic process algebra

Reason about behaviour: axioms

Axioms for the basic operators:

$$
\begin{array}{lrcl}
\text{A1} & p+q & = & q+p \\
\text{A2} & p+(q+r) & = & (p+q)+r \\
\text{A3} & p+p & = & p \\
\text{A4} & (p+q)\cdot r & = & p\cdot r+q\cdot r \\
\text{A5} & (p\cdot q)\cdot r & = & p\cdot(q\cdot r) \\
\text{A6} & \mathsf{a}+\delta & = & \mathsf{a} \\
\text{A7} & \delta\cdot p & = & \delta
\end{array}
$$

Exercise

1. $\mathsf{a}+(\delta+\mathsf{a})=\mathsf{a}$
2. $\delta\cdot(\mathsf{a}+\mathsf{b})=\delta\cdot\mathsf{a}+\delta\cdot\mathsf{b}$
3. $\mathsf{a}\cdot(\mathsf{b}+\mathsf{c})\cdot\mathsf{d}\cdot(\mathsf{b}+\mathsf{c})=\mathsf{a}\cdot(\mathsf{b}\cdot\mathsf{d}\cdot(\mathsf{b}+\mathsf{c})+\mathsf{c}\cdot\mathsf{d}\cdot(\mathsf{b}+\mathsf{c}))$

Basic process algebra

Reason about behaviour: axioms (2)

Solution to exercise 1

Derivation of $\mathsf{a} + (\delta + \mathsf{a}) = \mathsf{a}$:

$$\begin{array}{ll} & \mathsf{a} + (\delta + \mathsf{a}) \\ = & \quad \{ \text{ Axiom A2: } p + (q + r) = (p + q) + r \ \} \\ & (\mathsf{a} + \delta) + \mathsf{a} \\ = & \quad \{ \text{ Axiom A6: } \mathsf{a} + \delta = \mathsf{a} \ \} \\ & \mathsf{a} + \mathsf{a} \\ = & \quad \{ \text{ Axiom A3: } p + p = p \ \} \\ & \mathsf{a} \end{array}$$

Basic process algebra

Reason about behaviour: axioms (2)

Solution to exercise 2

Derivation of $\delta \cdot (\mathsf{a} + \mathsf{b}) = \delta \cdot \mathsf{a} + \delta \cdot \mathsf{b}$:

$$\begin{array}{ll} & \delta \cdot (\mathsf{a} + \mathsf{b}) \\ = & \quad \{ \text{ Axiom A7: } \delta \cdot p = \delta \ \} \\ & \delta \\ = & \quad \{ \text{ Axiom A3: } p + p = p \ \} \\ & \delta + \delta \\ = & \quad \{ \text{ Axiom A7 (twice) } \} \\ & \delta \cdot \mathsf{a} + \delta \cdot \mathsf{b} \end{array}$$

Basic process algebra

Reason about behaviour: axioms (2)

Solution to exercise 3

Derivation of
$\mathsf{a} \cdot (\mathsf{b} + \mathsf{c}) \cdot \mathsf{d} \cdot (\mathsf{b} + \mathsf{c}) = \mathsf{a} \cdot (\mathsf{b} \cdot \mathsf{d} \cdot (\mathsf{b} + \mathsf{c}) + \mathsf{c} \cdot \mathsf{d} \cdot (\mathsf{b} + \mathsf{c}))$:

$$\begin{array}{ll} & \mathsf{a} \cdot (\mathsf{b} + \mathsf{c}) \cdot \mathsf{d} \cdot (\mathsf{b} + \mathsf{c}) \\ = & \quad \{ \text{ Axiom A4: } (p + q) \cdot r = p \cdot r + q \cdot r \ \} \\ & \mathsf{a} \cdot (\mathsf{b} \cdot \mathsf{d} \cdot (\mathsf{b} + \mathsf{c}) + \mathsf{c} \cdot \mathsf{d} \cdot (\mathsf{b} + \mathsf{c})) \end{array}$$

Basic process algebra

Reason about behaviour: axioms (3)

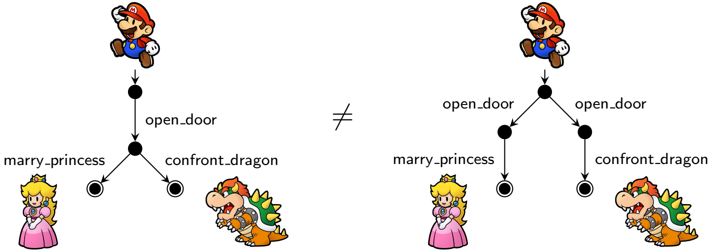
Is the following valid: $p \cdot (q + r) = p \cdot q + p \cdot r$?

Basic process algebra

Reason about behaviour: axioms (3)

Is the following valid: $p \cdot (q + r) = p \cdot q + p \cdot r$?

The princess, or the dragon?



F. Stockton, "The Lady, or the Tiger?", *An Anthology of Famous American Stories*, New York, Modern Library, 1953, pp. 248-253.

Basic process algebra

Reason about behaviour: axioms (3)

Is the following valid: $p \cdot (q + r) = p \cdot q + p \cdot r$?

It depends on your view:

- Bisimulation equivalence: no
- Trace equivalence: yes

Lots of equivalences inbetween.

Basic process algebra

Process definition

Deal with loops by introducing recursive processes:

- Add process definitions of the form $\mathsf{P} = p$
- P is called a process reference
- Processes: $p \ ::= \ \mathsf{a} \mid p \cdot p \mid p + p \mid \delta \mid \mathsf{P}$

Basic process algebra

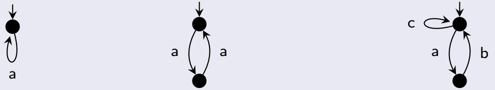
Process definition

Deal with loops by introducing recursive processes:

- Add process definitions of the form $\mathsf{P} = p$
- P is called a process reference
- Processes: $p ::= \mathsf{a} \mid p \cdot p \mid p + p \mid \delta \mid \mathsf{P}$

Example: LTS of P

$\mathsf{P} = \mathsf{a} \cdot \mathsf{P}$ $\mathsf{P} = \mathsf{a} \cdot \mathsf{a} \cdot \mathsf{P}$ $\mathsf{P} = \mathsf{c} \cdot \mathsf{P} + \mathsf{a} \cdot \mathsf{b} \cdot \mathsf{P}$



Basic process algebra

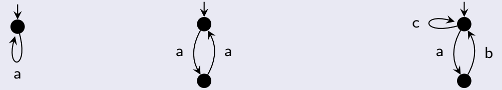
Process definition

Deal with loops by introducing recursive processes:

- Add process definitions of the form $\mathsf{P} = p$
- P is called a process reference
- Processes: $p \ ::= \ \mathsf{a} \mid p \cdot p \mid p + p \mid \delta \mid \mathsf{P}$

Example: LTS of P

$\mathsf{P} = \mathsf{a} \cdot \mathsf{P}$ $\quad$ $\mathsf{P} = \mathsf{a} \cdot \mathsf{a} \cdot \mathsf{P}$ $\quad$ $\mathsf{P} = \mathsf{c} \cdot \mathsf{P} + \mathsf{a} \cdot \mathsf{b} \cdot \mathsf{P}$



$\mathsf{P} = \mathsf{a} \cdot \mathsf{P}$ is equivalent to $\mathsf{P} = \mathsf{a} \cdot \mathsf{a} \cdot \mathsf{P}$.

Basic process algebra

Process specifications

Process specifications:

act $\mathsf{a}_0, \ldots, \mathsf{a}_\mathsf{n}$;
proc $\mathsf{P}_0 = p_0; \;\cdots\; \mathsf{P}_\mathsf{m} = p_m$;
init p;

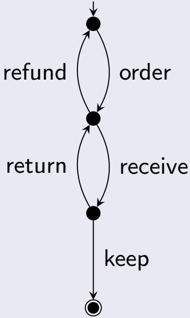
- **act** declares actions used in **proc** and **init**
- **proc** consists of process definitions
- **init** represents the initial process

Basic process algebra

Process specifications (2)

Exercise

Give a process specification of the following LTS:



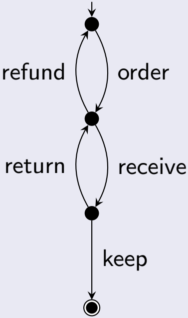
Basic process algebra

Process specifications (2)

Solution

Process specification:

act order, receive, keep, refund, return;
proc Start $=$ order $\cdot$ Ordered;
Ordered $=$ receive $\cdot$ Received
$+$ refund $\cdot$ Start;
Received $=$ return $\cdot$ Ordered
$+$ keep;
init Start;



Outline

Parallelism and abstraction

Motivation

Observation (Robin Milner, 1973):

> *Interaction is a primary concept*
> *in computer science.*

Parallelism and abstraction

Motivation

Observation (Robin Milner, 1973):

> *Interaction is a primary concept*
> *in computer science.*

Key ideas:

- Black box philosophy: focus on the interactions (inputs and outputs) of a system
- Treat distributed systems as communicating black boxes

Parallelism and abstraction

Parallelism

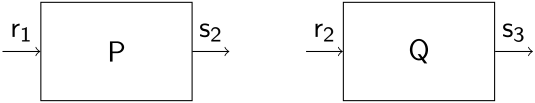
Processes:

$$p ::= \mathsf{a} \mid p \cdot p \mid p + p \mid \delta \mid \mathsf{P} \mid p \parallel p \mid p \,|\, p$$

- $\parallel$ represent parallel composition
- $|$ represents synchronisation
- Processes of the form $\mathsf{a} \,|\, \cdots \,|\, \mathsf{a}$ are called multiactions

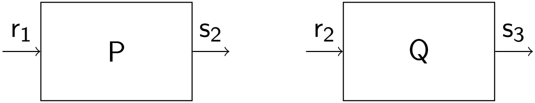
Parallelism and abstraction

Parallelism: example



Parallelism and abstraction

Parallelism: example



Process specification:

act $r_1, s_2, r_2, s_3;$

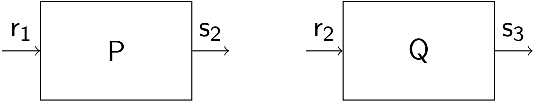
proc $P = r_1 \cdot s_2 \cdot P;$

$Q = r_2 \cdot s_3 \cdot Q;$

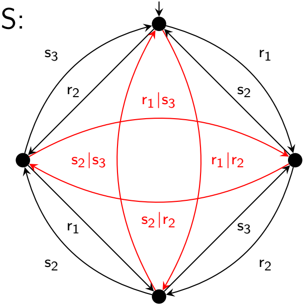
init $P \parallel Q;$

Parallelism and abstraction

Parallelism: example



Corresponding LTS:



Parallelism and abstraction

Communication

Processes:

$$\begin{aligned} p \;&::= \mathsf{a} \mid p\cdot p \mid p+p \mid \delta \mid \mathsf{P} \mid p \parallel p \mid p \,|\, p \\ &\quad\mid \Gamma_{\{\mathsf{a}|ma\to\mathsf{a},\ldots,\mathsf{a}|ma\to\mathsf{a}\}}(p) \mid \partial_{\{\mathsf{a},\ldots,\mathsf{a}\}}(p) \mid \nabla_{\{ma,\ldots,ma\}}(p) \\ ma \;&::= \mathsf{a} \,|\, \cdots \,|\, \mathsf{a} \end{aligned}$$

- $\Gamma_{\{\mathsf{a}|\mathsf{b}\to\mathsf{c}\}}(p)$ renames multiactions a|b to c
- $\partial_S(p)$ blocks (renames to δ) all actions in the set S
- $\nabla_S(p)$ blocks all multiactions different from the ones in S

Parallelism and abstraction

Communication

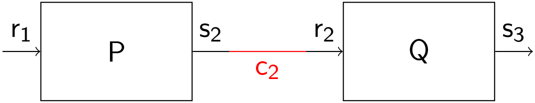
Processes:

$$\begin{aligned} p \;&::= \mathsf{a} \mid p \cdot p \mid p + p \mid \delta \mid \mathsf{P} \mid p \parallel p \mid p \,|\, p \\ &\quad\mid \Gamma_{\{\mathsf{a}|ma\to\mathsf{a},\ldots,\mathsf{a}|ma\to\mathsf{a}\}}(p) \mid \partial_{\{\mathsf{a},\ldots,\mathsf{a}\}}(p) \mid \nabla_{\{ma,\ldots,ma\}}(p) \\ ma &::= \mathsf{a} \,|\, \cdots \,|\, \mathsf{a} \end{aligned}$$

- $\Gamma_{\{\mathsf{a}|\mathsf{b}\to\mathsf{c}\}}(p)$ renames multiactions a|b to c
- $\partial_S(p)$ blocks (renames to δ) all actions in the set S
- $\nabla_S(p)$ blocks all multiactions different from the ones in S
- Enforce communication of a|b to c:
  - $\partial_{\{\mathsf{a},\mathsf{b}\}}(\Gamma_{\{\mathsf{a}|\mathsf{b}\to\mathsf{c}\}}(p))$ by blocking a and b
  - $\nabla_{\{\mathsf{c}\}}(\Gamma_{\{\mathsf{a}|\mathsf{b}\to\mathsf{c}\}}(p))$ by only allowing c

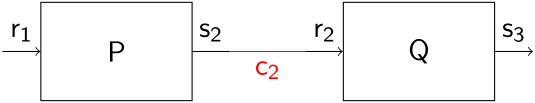
Parallelism and abstraction

Communication: example



Parallelism and abstraction

Communication: example



Process specification:

act $r_1, s_2, r_2, s_3, c_2;$

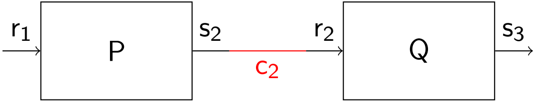
proc $P = r_1 \cdot s_2 \cdot P;$

$Q = r_2 \cdot s_3 \cdot Q;$

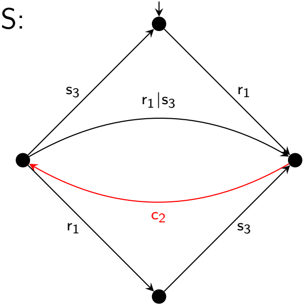
init $\partial_{\{r_2, s_2\}}(\Gamma_{\{s_2 | r_2 \to c_2\}}(P \parallel Q));$

Parallelism and abstraction

Communication: example

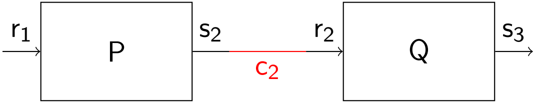


Corresponding LTS:



Parallelism and abstraction

Communication: example



Process specification:

act $r_1, s_2, r_2, s_3, c_2;$

proc $P = r_1 \cdot s_2 \cdot P;$

$Q = r_2 \cdot s_3 \cdot Q;$

init $\nabla_{\{c_2, r_1, s_3, r_1|s_3\}}(\Gamma_{\{s_2|r_2 \rightarrow c_2\}}(P \parallel Q));$

Parallelism and abstraction

Abstraction

Motivation for abstraction:

- Focus on external behaviour:
  abstract from internal behaviour
- Composition of models

Parallelism and abstraction

Abstraction (2)

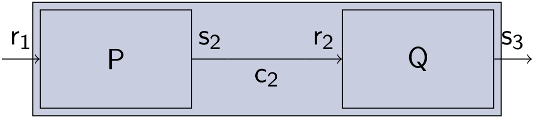
Processes:

$$\begin{array}{lll} p & ::= & \mathsf{a} \mid p \cdot p \mid p + p \mid \delta \mid \mathsf{P} \mid p \parallel p \mid p \,|\, p \\ & & \mid \Gamma_{\{\mathsf{a}|ma \rightarrow \mathsf{a}, \ldots, \mathsf{a}|ma \rightarrow \mathsf{a}\}}(p) \mid \partial_{\{\mathsf{a}, \ldots, \mathsf{a}\}}(p) \mid \nabla_{\{ma, \ldots, ma\}}(p) \\ & & \mid \tau \mid \tau_{\{\mathsf{a}, \ldots, \mathsf{a}\}}(p) \\ ma & ::= & \mathsf{a} \,|\, \cdots \,|\, \mathsf{a} \end{array}$$

- τ represents an internal action
- $\tau_S(p)$ hides (renames to τ) all actions from S in p

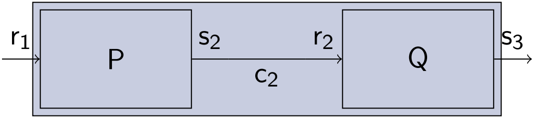
Parallelism and abstraction

Abstraction: example



Parallelism and abstraction

Abstraction: example



Process specification:

act $\mathsf{r_1, s_2, r_2, s_3, c_2;}$

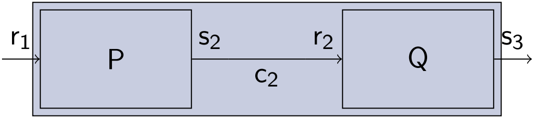
proc $\mathsf{P = r_1 \cdot s_2 \cdot P;}$

$\mathsf{Q = r_2 \cdot s_3 \cdot Q;}$

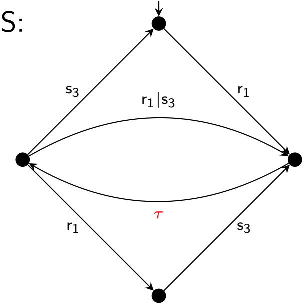
init $\tau_{\{c_2\}}(\partial_{\{r_2,s_2\}}(\Gamma_{\{s_2|r_2\rightarrow c_2\}}(P \parallel Q)));$

Parallelism and abstraction

Abstraction: example

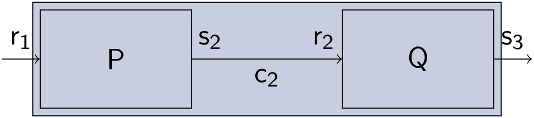


Corresponding LTS:

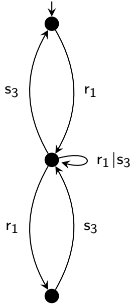


Parallelism and abstraction

Abstraction: example



Corresponding LTS:



Parallelism and abstraction

Branching bisimulation

Consequences of adding τ transitions:

- Only external actions are observable
- The effects of an internal action can only be observed if it determines a choice
- Weaker notion of bisimulation: branching bisimulation

Parallelism and abstraction

Branching bisimulation: example

Example

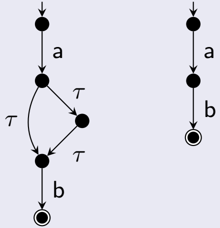
The following are equivalent: $\mathsf{a} \cdot (\tau + \tau \cdot \tau) \cdot \mathsf{b}$ and $\mathsf{a} \cdot \mathsf{b}$

Parallelism and abstraction

Branching bisimulation: example

Example

The following are equivalent: $a \cdot (\tau + \tau \cdot \tau) \cdot b$ and $a \cdot b$

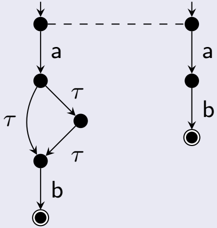


Parallelism and abstraction

Branching bisimulation: example

Example

The following are equivalent: $\mathsf{a} \cdot (\tau + \tau \cdot \tau) \cdot \mathsf{b}$ and $\mathsf{a} \cdot \mathsf{b}$

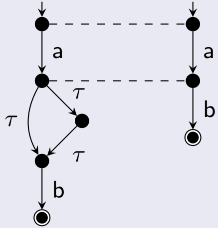


Parallelism and abstraction

Branching bisimulation: example

Example

The following are equivalent: $\mathsf{a} \cdot (\tau + \tau \cdot \tau) \cdot \mathsf{b}$ and $\mathsf{a} \cdot \mathsf{b}$

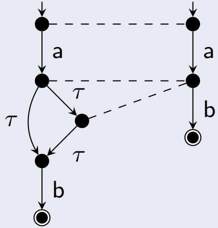


Parallelism and abstraction

Branching bisimulation: example

Example

The following are equivalent: $\mathsf{a} \cdot (\tau + \tau \cdot \tau) \cdot \mathsf{b}$ and $\mathsf{a} \cdot \mathsf{b}$

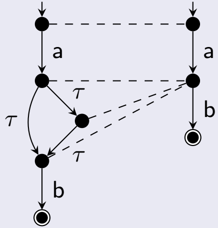


Parallelism and abstraction

Branching bisimulation: example

Example

The following are equivalent: $\mathsf{a} \cdot (\tau + \tau \cdot \tau) \cdot \mathsf{b}$ and $\mathsf{a} \cdot \mathsf{b}$

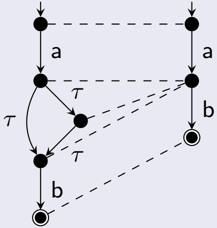


Parallelism and abstraction

Branching bisimulation: example

Example

The following are equivalent: $a \cdot (\tau + \tau \cdot \tau) \cdot b$ and $a \cdot b$



Parallelism and abstraction

Branching bisimulation: axioms

Axioms for the basic operators and τ:

A1	$p+q$	$=$	$q+p$
A2	$p+(q+r)$	$=$	$(p+q)+r$
A3	$p+p$	$=$	p
A4	$(p+q)\cdot r$	$=$	$p\cdot r+q\cdot r$
A5	$(p\cdot q)\cdot r$	$=$	$p\cdot(q\cdot r)$
A6	$\mathsf{a}+\delta$	$=$	a
A7	$\delta\cdot p$	$=$	δ
T1	$p\cdot\tau$	$=$	p
T2	$p\cdot(\tau\cdot(q+r)+q)$	$=$	$p\cdot(q+r)$

Parallelism and abstraction

Branching bisimulation: axioms

Axioms for the basic operators and τ:

A1 $\quad p + q = q + p$

A2 $\quad p + (q + r) = (p + q) + r$

A3 $\quad p + p = p$

A4 $\quad (p + q) \cdot r = p \cdot r + q \cdot r$

A5 $\quad (p \cdot q) \cdot r = p \cdot (q \cdot r)$

A6 $\quad \mathsf{a} + \delta = \mathsf{a}$

A7 $\quad \delta \cdot p = \delta$

T1 $\quad p \cdot \tau = p$

T2 $\quad p \cdot (\tau \cdot (q + r) + q) = p \cdot (q + r)$

Exercise

Show the following: $\mathsf{a} \cdot ((\tau + \tau \cdot \tau) \cdot \mathsf{b}) = \mathsf{a} \cdot \mathsf{b}$

Parallelism and abstraction

Branching bisimulation: axioms

Axioms for the basic operators and τ:

A1	$p + q$	$=$	$q + p$
A2	$p + (q + r)$	$=$	$(p + q) + r$
A3	$p + p$	$=$	p
A4	$(p + q) \cdot r$	$=$	$p \cdot r + q \cdot r$
A5	$(p \cdot q) \cdot r$	$=$	$p \cdot (q \cdot r)$
A6	$\mathsf{a} + \delta$	$=$	a
A7	$\delta \cdot p$	$=$	δ
T1	$p \cdot \tau$	$=$	p
T2	$p \cdot (\tau \cdot (q + r) + q)$	$=$	$p \cdot (q + r)$

Exercise

$$\mathsf{a} \cdot ((\tau + \tau \cdot \tau) \cdot \mathsf{b}) \overset{\text{T1}}{=} \mathsf{a} \cdot ((\tau + \tau) \cdot \mathsf{b}) \overset{\text{A3,A5}}{=} (\mathsf{a} \cdot \tau) \cdot \mathsf{b} \overset{\text{T1}}{=} \mathsf{a} \cdot \mathsf{b}$$

Outline

1. Basic process algebra
2. Parallelism and abstraction
3. Processes with data
4. Linear processes
5. Temporal Logic
6. Verification
7. Toolset overview and demo
8. Hands-on experience
9. Wrap-up
10. Industrial case studies

Processes with data

Why add data?

- In real-life systems data is essential
- Data allows for finite specifications of infinite systems

Processes with data

Why add data?

- In real-life systems data is essential
- Data allows for finite specifications of infinite systems

Example

A specification of a buffer that repeatedly receives a natural number and then sends it to the outside world:

act send_0, receive_0, send_1, receive_1, . . .

proc $\mathsf{Buffer} = \mathsf{receive\_0} \cdot \mathsf{send\_0} \cdot \mathsf{Buffer}$

$+\ \mathsf{receive\_1} \cdot \mathsf{send\_1} \cdot \mathsf{Buffer}$

$+\ \ldots$

init Buffer;

Processes with data

Data types

- All types: equality, inequality and if
  $\approx$, $\not\approx$, $\mathit{if}(\_, \_, \_)$
- Basic types: $\mathbb{B}$, $\mathbb{N}^+$, $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$
  $\neg$, $\wedge$, $\vee$, $\forall$, $\exists$, $<$, $\leq$, $+$, $-$, $*$, $\mathbf{div}$, $\mathbf{mod}$, $\max$, $\min$, $\ldots$
- Lists, sets and bags
  $[1, 3, 4]$, $\triangleright$, $\triangleleft$, $+\!\!+$, $\cup$, $\cap$, $\setminus$, $\in$, $\subseteq$, $\subset$, $\ldots$
- Functions
  $\lambda x{:}\mathbb{N} \; . \; x * x$
- Structured types
  sort $\quad \mathit{State} = \mathbf{struct}\ \mathit{idle} \mid \mathit{running} \mid \mathit{defect};$
  sort $\quad \mathit{Tree} = \mathbf{struct}\ \mathit{leaf}(\mathbb{N}) \mid \mathit{node}(\mathit{Tree}, \mathit{Tree});$

Processes with data

Data specifications

Example: flatten a tree using pattern matching

sort $\mathit{Tree} = \textbf{struct } \mathit{leaf}(\mathbb{N})$
$\qquad\qquad \mid \mathit{node}(\mathit{Tree}, \mathit{Tree});$

map $\mathit{flatten}\colon \mathit{Tree} \to \mathit{List}(\mathbb{N});$

var $n{:}\mathbb{N}; t, u\colon \mathit{Tree};$

eqn $\mathit{flatten}(\mathit{leaf}(n)) = [n];$
$\mathit{flatten}(\mathit{node}(t, u)) = t \mathbin{+\!\!+} u;$

Processes with data

Data specifications

Example: flatten a tree without pattern matching

$$
\begin{array}{ll}
\textbf{sort} & \mathit{Tree} = \textbf{struct}\ \mathit{leaf}(\mathit{val}{:}\mathbb{N})?\mathit{is\_leaf} \\
 & \qquad\qquad\quad \mid \mathit{node}(\mathit{left}{:}\mathit{Tree}, \mathit{right}{:}\mathit{Tree})?\mathit{is\_node}; \\
\textbf{map} & \mathit{flatten}{:}\mathit{Tree} \rightarrow \mathit{List}(\mathbb{N}); \\
\textbf{var} & t{:}\mathit{Tree}; \\
\textbf{eqn} & \mathit{is\_leaf}(t) \ \rightarrow \mathit{flatten}(t) = [\mathit{val}(t)]; \\
 & \mathit{is\_node}(t) \rightarrow \mathit{flatten}(t) = \\
 & \qquad\qquad \mathit{flatten}(\mathit{left}(t)) \mathbin{+\!\!+} \mathit{flatten}(\mathit{right}(t));
\end{array}
$$

Processes with data

Adding data to processes

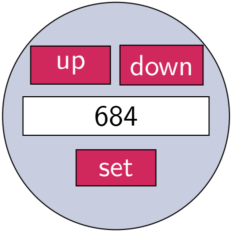
Processes:

$$
\begin{array}{lll}
p & ::= & \mathsf{a} \mid p \cdot p \mid p + p \mid \delta \mid \mathsf{P} \mid p \parallel p \mid p \,|\, p \\
 & & \mid \Gamma_{\{\mathsf{a}|ma \to \mathsf{a},\ldots,\mathsf{a}|ma \to \mathsf{a}\}}(p) \mid \partial_{\{\mathsf{a},\ldots,\mathsf{a}\}}(p) \mid \nabla_{\{ma,\ldots,ma\}}(p) \\
 & & \mid \tau \mid \tau_{\{\mathsf{a},\ldots,\mathsf{a}\}}(p) \\
 & & \mid \mathsf{a}(d,\ldots,d) \mid \mathsf{P}(d,\ldots,d) \mid b \to p \diamond p \mid \sum_{x:s} p \\
ma & ::= & \mathsf{a} \,|\, \cdots \,|\, \mathsf{a}
\end{array}
$$

- Action and processes can be parameterised: $\mathsf{a}(25)$, $\mathsf{P}(true)$
  Declarations of actions and processes: $\mathsf{a}{:}\mathbb{N}$, $\mathsf{P}(b{:}\mathbb{B}) = \ldots$
- Conditions influence process behaviour: $b \to \mathsf{a} \diamond \mathsf{b}$
  $b \to p$ is an abbrevation of $b \to p \diamond \delta$
- Summation over data types: $\sum_{n:\mathbb{N}} \mathsf{a}(n)$

Processes with data

Adding data to processes: example



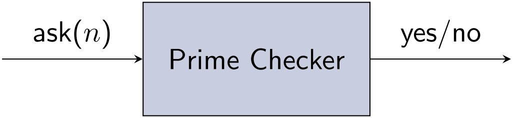
act up, down;
set:$\mathbb{N}$;

proc Counter$(n{:}\mathbb{N})$ $= \text{up} \cdot \text{Counter}(n+1)$
$+ (n > 0) \rightarrow \text{down} \cdot \text{Counter}(n-1)$
$+ \sum_{m:\mathbb{N}} \text{set}(m) \cdot \text{Counter}(m)$;

init Counter(684);

Processes with data

Adding data to processes: example (2)



ask(n)

Prime Checker

yes/no

map $primes : Set(\mathbb{N});$

eqn $primes = \{n{:}\mathbb{N} \mid \forall_{p,q:\mathbb{N}}\; p > 1 \land q > 1 \;\Rightarrow\; p * q \neq n\};$

act $\mathsf{ask} : \mathbb{N};$

$\mathsf{yes}, \mathsf{no};$

proc $\mathsf{PC} = \sum_{n:\mathbb{N}} \mathsf{ask}(n) \cdot ((n \in primes) \rightarrow \mathsf{yes} \diamond \mathsf{no}) \cdot \mathsf{PC};$

init $\mathsf{PC};$

Outline

1. Basic process algebra
2. Parallelism and abstraction
3. Processes with data
4. **Linear processes**
5. Temporal Logic
6. Verification
7. Toolset overview and demo
8. Hands-on experience
9. Wrap-up
10. Industrial case studies

Linear processes

Linear process definitions

A linear process definition is a process of the form:

$$\mathsf{P}(d:D) = \sum_{i\in I}\sum_{e:E_i} c_i(d,e) \rightarrow \mathsf{a}_i(f_i(d,e)) \cdot \mathsf{P}(g_i(d,e))$$

Idea: a series of *condition – action – effect* rules:

Linear processes

Linear process definitions

A linear process definition is a process of the form:

$$\mathsf{P}(d : D) = \sum_{i \in I} \sum_{e:E_i} c_i(d, e) \rightarrow \mathsf{a}_i(f_i(d, e)) \cdot \mathsf{P}(g_i(d, e))$$

Idea: a series of *condition* – *action* – *effect* rules:

- Given the current state

Linear processes

Linear process definitions

A linear process definition is a process of the form:

$$\mathsf{P}(d:D) = \sum_{i\in I}\sum_{e:E_i} c_i(d,e) \rightarrow \mathsf{a}_i(f_i(d,e)) \cdot \mathsf{P}(g_i(d,e))$$

Idea: a series of *condition* – *action* – *effect* rules:

- Given the current state
- If the condition holds

Linear processes

Linear process definitions

A linear process definition is a process of the form:

$$\mathsf{P}(d : D) = \sum_{i \in I} \sum_{e : E_i} c_i(d, e) \rightarrow \mathsf{a}_i(f_i(d, e)) \cdot \mathsf{P}(g_i(d, e))$$

Idea: a series of *condition* – *action* – *effect* rules:

- Given the current state
- If the condition holds
- The action can be executed

Linear processes

Linear process definitions

A linear process definition is a process of the form:

$$\mathsf{P}(d:D) = \sum_{i \in I} \sum_{e:E_i} c_i(d,e) \rightarrow \mathsf{a}_i(f_i(d,e)) \cdot \mathsf{P}(g_i(d,e))$$

Idea: a series of *condition* – *action* – *effect* rules:

- Given the current state
- If the condition holds
- The action can be executed
- Resulting in the next state (optional)

Linear processes

Linear process specifications

A linear process specification (LPS) is a restricted form of an mCRL2 process specification:

- a data type specification;
- an action specification;
- a single, linear process definition;
- an initial process reference.

An LPS is a symbolic representation of a labelled transition system.

An mCRL2 specification can be linearised to an LPS if it is a *parallel composition of parallel-free processes*.

Linear processes

Linearisation

Example

mCRL2 specification *before* linearisation:

$$
\begin{array}{lll}
\textbf{act} & \multicolumn{2}{l}{\text{order}, \text{receive}, \text{keep}, \text{refund}, \text{return};} \\
\textbf{proc} & \text{Start} & = \text{order} \cdot \text{Ordered}; \\
 & \text{Ordered} & = \text{receive} \cdot \text{Received} + \text{refund} \cdot \text{Start}; \\
 & \text{Received} & = \text{return} \cdot \text{Ordered} + \text{keep}; \\
\textbf{init} & \multicolumn{2}{l}{\text{Start};}
\end{array}
$$

Linear processes

Linearisation

Example

mCRL2 specification *after* linearisation:

sort $State = \textbf{struct}\ start \mid ordered \mid received;$

act order, receive, keep, refund, return;

proc $\mathsf{P}(s : State) =$

$$\begin{array}{rl}
(s \approx start) & \rightarrow \mathsf{order} \cdot \mathsf{P}(ordered) \\
+\ (s \approx ordered) & \rightarrow \mathsf{receive} \cdot \mathsf{P}(received) \\
+\ (s \approx ordered) & \rightarrow \mathsf{refund} \cdot \mathsf{P}(start) \\
+\ (s \approx received) & \rightarrow \mathsf{return} \cdot \mathsf{P}(ordered) \\
+\ (s \approx received) & \rightarrow \mathsf{keep};
\end{array}$$

init $\mathsf{P}(start);$

Linear processes

Linearisation

Exercise: linearise the following mCRL2 specification

act receive, send : $\mathbb{N}$;

proc Buffer $= \sum_{n:\mathbb{N}}$ receive$(n) \cdot$ send$(n) \cdot$ Buffer;

init Buffer;

Linear processes

Linearisation

Exercise: linearise the following mCRL2 specification

act receive, send : $\mathbb{N}$;

proc Buffer $= \sum_{n:\mathbb{N}}$ receive$(n) \cdot$ send$(n) \cdot$ Buffer;

init Buffer;

Parameter $b : \mathbb{B}$:

Linear processes

Linearisation

Exercise: linearise the following mCRL2 specification

act receive, send : $\mathbb{N}$;

proc $\text{Buffer} = \sum_{n:\mathbb{N}} \underset{\uparrow}{}\text{receive}(n) \underset{\uparrow}{\cdot} \text{send}(n) \underset{\uparrow}{\cdot} \text{Buffer};$

init Buffer;

Parameter $b : \mathbb{B}$: false true false

Linear processes

Linearisation

Exercise: linearise the following mCRL2 specification

act receive, send : $\mathbb{N}$;

proc $\text{Buffer} = \sum_{n:\mathbb{N}} \underset{\uparrow}{} \text{receive}(n) \underset{\uparrow}{\cdot} \text{send}(n) \underset{\uparrow}{\cdot} \text{Buffer};$

init Buffer;

Parameter $b : \mathbb{B}$: false $\quad$ true $\quad$ false

$$
\begin{aligned}
\textbf{proc}\quad \mathsf{P}(b{:}\mathbb{B}, m{:}\mathbb{N}) &= \textstyle\sum_{n:\mathbb{N}} \neg b \rightarrow \text{receive}(n) \cdot \mathsf{P}(\mathit{true}, n) \\
&+ \qquad\quad b \rightarrow \text{send}(m) \cdot \mathsf{P}(\mathit{false}, 0);
\end{aligned}
$$

init $\mathsf{P}(\mathit{false}, 0)$;

Linear processes

Summary

Linear process specification:

- Simple mCRL2 specification:
    - no parallelism
    - single process
    - restricted format (condition – action – effect)
- Symbolic representation of LTS, hence:
    - compact
    - finite, even if LTS is infinite
- Very suitable for automated manipulation and analysis
- Most mCRL2 specifications can be easily linearised
- Central notion in mCRL2 toolset

Outline

1. Basic process algebra
2. Parallelism and abstraction
3. Processes with data
4. Linear processes
5. Temporal Logic
6. Verification
7. Toolset overview and demo
8. Hands-on experience
9. Wrap-up
10. Industrial case studies

Temporal Logic

Model checking is an automated verification method. It can be used to check functional requirements against a model.

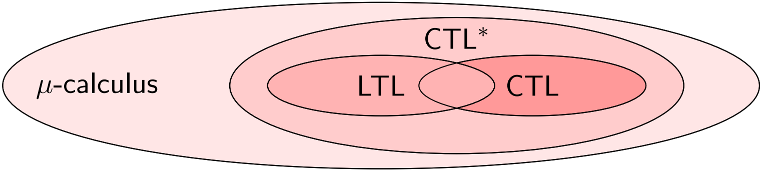
- A (software or hardware) system is modelled in mCRL2
- The requirements are specified as properties in a temporal logic
- A model checking algorithm decides whether the property holds for the model: the property can be verified or refuted. Sometimes, witnesses or counter examples can be provided

Temporal logic of choice in mCRL2:
μ-calculus with data, time and regular expressions.

Temporal Logic

Idea of μ-calculus: add fixed point operators (i.e. recursion) as primitives to standard *Hennessy-Milner* logic.

- μ-calculus is very expressive (subsumes e.g. CTL*).
- μ-calculus is very pure.
- drawback: lack of intuition.
- Today: *alternation-free* μ-calculus using regular expressions and data.



Temporal Logic

Hennessy-Milner logic: propositional logic with *modalities*:

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

Notation

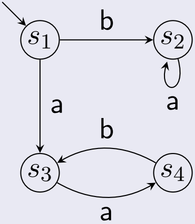
$s \models \phi$: state s of a transition system satisfies formula ϕ

- for all states s: $s \models \mathit{true}$; for no state s: $s \models \mathit{false}$;
- $s \models [\mathsf{a}]\phi$ iff all a-labelled transitions starting in s and leading to a state t satisfy $t \models \phi$;
- $s \models \langle \mathsf{a} \rangle \phi$ iff there is at least one a-labelled transition starting in s and leading to a state t satisfying $t \models \phi$.

Temporal Logic

Exercise

Determine the largest subset $S \subseteq \{s_1, s_2, s_3, s_4\}$ in the following satisfaction problems:

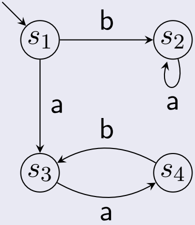


- $S \models [\mathsf{b}]\mathit{false}$
- $S \models [\mathsf{a}][\mathsf{b}][\mathsf{c}]\mathit{true}$
- $S \models \langle\mathsf{a}\rangle\mathit{true}$

Temporal Logic

Exercise

Determine the largest subset $S \subseteq \{s_1, s_2, s_3, s_4\}$ in the following satisfaction problems:



- $S \models [\mathsf{b}]\mathit{false}$ $S = \{s_2, s_3\}$
- $S \models [\mathsf{a}][\mathsf{b}][\mathsf{c}]\mathit{true}$ $S = \{s_1, s_2, s_3, s_4\}$
- $S \models \langle \mathsf{a} \rangle \mathit{true}$ $S = \{s_1, s_2, s_3\}$

Temporal Logic

HM-logic + *basic regular expressions*:

$$\begin{aligned}\phi &::= \mathit{true} \mid \mathit{false} \mid \phi \wedge \phi \mid \phi \vee \phi \mid [\rho]\phi \mid \langle\rho\rangle\phi \\ \rho &::= \epsilon \mid \mathsf{a} \mid \rho \cdot \rho \mid \rho + \rho\end{aligned}$$

- ϵ is the empty word;
- a is an action;
- $\rho \cdot \rho$ is concatenation;
- $\rho + \rho$ is *choice*.

Temporal Logic

HM-logic + *basic regular expressions*:

$$\begin{aligned}\phi &::= \mathit{true} \mid \mathit{false} \mid \phi \wedge \phi \mid \phi \vee \phi \mid [\rho]\phi \mid \langle\rho\rangle\phi \\ \rho &::= \epsilon \mid \mathsf{a} \mid \rho \cdot \rho \mid \rho + \rho\end{aligned}$$

- ϵ is the empty word;
- a is an action;
- $\rho \cdot \rho$ is concatenation;
- $\rho + \rho$ is *choice*.

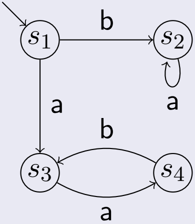
Combined with the modalities $[\_]\_$ and $\langle\_\rangle\_$:

$$\begin{array}{lcl} s \models [\rho_1 \cdot \rho_2]\phi & \text{iff} & s \models [\rho_1][\rho_2]\phi \\ s \models [\rho_1 + \rho_2]\phi & \text{iff} & s \models [\rho_1]\phi \wedge [\rho_2]\phi \\ \\ s \models \langle\rho_1 \cdot \rho_2\rangle\phi & \text{iff} & s \models \langle\rho_1\rangle\langle\rho_2\rangle\phi \\ s \models \langle\rho_1 + \rho_2\rangle\phi & \text{iff} & s \models \langle\rho_1\rangle\phi \vee \langle\rho_2\rangle\phi \end{array}$$

Temporal Logic

Exercise

Determine the largest subset $S \subseteq \{s_1, s_2, s_3, s_4\}$ in the following satisfaction problems:

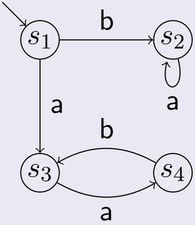


- $S \models [\mathsf{b} + \mathsf{a}]\mathit{false}$
- $S \models [\mathsf{a} \cdot \mathsf{b} \cdot \mathsf{c}]\mathit{false}$
- $S \models \langle \mathsf{a} \cdot \mathsf{a} \cdot \mathsf{b} + \mathsf{a} \cdot \mathsf{a} \cdot \mathsf{a} \rangle \mathit{true}$

Temporal Logic

Exercise

Determine the largest subset $S \subseteq \{s_1, s_2, s_3, s_4\}$ in the following satisfaction problems:



- $S \models [\mathsf{b} + \mathsf{a}]\mathit{false}$ $S = \emptyset$
- $S \models [\mathsf{a} \cdot \mathsf{b} \cdot \mathsf{c}]\mathit{false}$ $S = \{s_1, s_2, s_3, s_4\}$
- $S \models \langle \mathsf{a} \cdot \mathsf{a} \cdot \mathsf{b} + \mathsf{a} \cdot \mathsf{a} \cdot \mathsf{a} \rangle \mathit{true}$ $S = \{s_1, s_2\}$

Temporal Logic

HM-logic + *iteration* + *regular expressions*:

$$\begin{aligned} \phi &::= \mathit{true} \mid \mathit{false} \mid \phi \wedge \phi \mid \phi \vee \phi \mid [\rho]\phi \mid \langle\rho\rangle\phi \\ \rho &::= \epsilon \mid \mathsf{a} \mid \rho \cdot \rho \mid \rho + \rho \mid \rho^* \mid \rho^+ \end{aligned}$$

- $\rho^* := \epsilon + \rho \cdot \rho^*$: transitive, reflexive closure of ρ;
- $\rho^+ := \rho \cdot \rho^*$: transitive closure of ρ.

- Iteration operators + modalities = recursion.
- recursion is coded using fixed points in the μ-calculus.

Temporal Logic

HM-logic + *iteration* + *regular expressions*:

$$\begin{array}{l} \phi ::= \mathit{true} \mid \mathit{false} \mid \phi \wedge \phi \mid \phi \vee \phi \mid [\rho]\phi \mid \langle\rho\rangle\phi \\ \rho ::= \epsilon \mid \mathsf{a} \mid \rho \cdot \rho \mid \rho + \rho \mid \rho^* \mid \rho^+ \end{array}$$

- $\rho^* := \epsilon + \rho \cdot \rho^*$: transitive, reflexive closure of ρ;
- $\rho^+ := \rho \cdot \rho^*$: transitive closure of ρ.

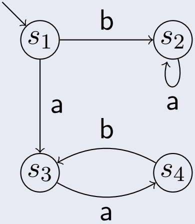
- Iteration operators + modalities = recursion.
- recursion is coded using fixed points in the μ-calculus.

- $[\rho^*]\phi := \nu X.\ [\rho]X \wedge \phi$; ν expresses *looping*;
- $\langle\rho^*\rangle\phi := \mu X.\ \langle\rho\rangle X \vee \phi$; μ expresses *finite looping*.

Temporal Logic

Exercise

Determine the largest subset $S \subseteq \{s_1, s_2, s_3, s_4\}$ in the following satisfaction problems:

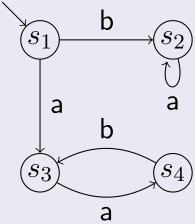


- $S \models \langle \mathsf{a}^* \rangle \mathit{true}$
- $S \models \langle \mathsf{a}^+ \rangle \mathit{true}$
- $S \models [\mathsf{a}^* \cdot \mathsf{b}] \mathit{false}$
- How to phrase *absence of deadlock*?

Temporal Logic

Exercise

Determine the largest subset $S \subseteq \{s_1, s_2, s_3, s_4\}$ in the following satisfaction problems:



- $S \models \langle \mathsf{a}^* \rangle \mathit{true}$ $S = \{s_1, s_2, s_3, s_4\}$
- $S \models \langle \mathsf{a}^+ \rangle \mathit{true}$ $S = \{s_1, s_2, s_3\}$
- $S \models [\mathsf{a}^* \cdot \mathsf{b}] \mathit{false}$ $S = \{s_2\}$
- How to phrase *absence of deadlock*?
  $[(\mathsf{a} + \mathsf{b})^*] \langle \mathsf{a} + \mathsf{b} \rangle \mathit{true}$

Temporal Logic

Consider the following definition of a lossy channel:

$$\textbf{proc}\quad \mathsf{C}(b{:}\mathbb{B}, m{:}M) = \textstyle\sum_{k:M}\ b \rightarrow \mathsf{read}(k) \cdot \mathsf{C}(\mathit{false}, k) \\ + \neg b \rightarrow \mathsf{send}(m) \cdot \mathsf{C}(\mathit{true}, m) \\ + \neg b \rightarrow \mathsf{lose} \cdot \mathsf{C}(\mathit{true}, m);$$

Temporal Logic

Consider the following definition of a lossy channel:

$$\textbf{proc}\quad \mathsf{C}(b{:}\mathbb{B}, m{:}M) = \textstyle\sum_{k:M}\ b \rightarrow \mathsf{read}(k)\cdot \mathsf{C}(\mathit{false}, k) + \neg b \rightarrow \mathsf{send}(m)\cdot \mathsf{C}(\mathit{true}, m) + \neg b \rightarrow \mathsf{lose}\cdot \mathsf{C}(\mathit{true}, m);$$

Problem

$|M| = \infty \implies$ infinitely many read and send actions;

- How to specify deadlock freedom as a finite expression?
- How to verify that no miracles happen? (e.g. *message creation*, *duplication*, *etc.*)

Temporal Logic

Extended HM-logic + *action abstraction*:

$$
\begin{aligned}
\phi &::= \mathit{true} \mid \mathit{false} \mid \phi \wedge \phi \mid \phi \vee \phi \mid [\rho]\phi \mid \langle\rho\rangle\phi \\
\rho &::= \epsilon \mid \alpha \mid \rho \cdot \rho \mid \rho + \rho \mid \rho^{*} \mid \rho^{+} \\
\alpha &::= \mathsf{a} \mid \mathsf{a}(d, \ldots, d) \mid b \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \neg\alpha \mid \forall_{x:D}\, \alpha \mid \exists_{x:D}\, \alpha
\end{aligned}
$$

Changes regular formulae (ρ):

- Actions have been replaced by parameterised actions.
- Logic is used to describe a possibly infinite set of actions.

Nota Bene:

- d stands for a data expression;
- b stands for a data expression of sort $\mathbb{B}$.

Temporal Logic

Logic for describing sets of actions:

- $true$ acts as wildcard (i.e. the *entire set* of actions);
- $\forall$ acts as intersection; $\exists$ is dual;
- $\neg$ acts as set complement.

Examples:

- Any parameterised action $a{:}\mathbb{N}$: . . . . $\langle \exists_{n:\mathbb{N}}\ \mathsf{a}(n) \rangle true$
- Any action (but not $a{:}\mathbb{N}$): . . . . $\langle \forall_{n:\mathbb{N}}\ \neg \mathsf{a}(n) \rangle true$
- Absence of deadlock: . . . . $[true^*]\langle true \rangle true$

Temporal Logic

Logic for describing sets of actions:

- $true$ acts as wildcard (i.e. the *entire set* of actions);
- $\forall$ acts as intersection; $\exists$ is dual;
- $\neg$ acts as set complement.

Examples:

- Any parameterised action $a{:}\mathbb{N}$: . . . . $\langle \exists_{n:\mathbb{N}}\, \mathsf{a}(n)\rangle true$
- Any action (but not $a{:}\mathbb{N}$): . . . . $\langle \forall_{n:\mathbb{N}}\, \neg \mathsf{a}(n)\rangle true$
- Absence of deadlock: . . . . $[true^*]\langle true\rangle true$

Abstraction enables finite description of infinite set of actions.
It does not provide full support for *data-dependence*.

Temporal Logic

Extended HM-logic + *action abstraction* + *data*:

$$\begin{array}{l}\phi ::= \phi \land \phi \mid \phi \lor \phi \mid [\rho]\phi \mid \langle\rho\rangle\phi \mid b \mid \forall_{x:D}\,\phi \mid \exists_{x:D}\,\phi \\ \rho ::= \epsilon \mid \alpha \mid \rho\cdot\rho \mid \rho+\rho \mid \rho^{*} \mid \rho^{+} \\ \alpha ::= \mathsf{a} \mid \mathsf{a}(d,\ldots,d) \mid b \mid \alpha\land\alpha \mid \alpha\lor\alpha \mid \neg\alpha \mid \forall_{x:D}\,\alpha \mid \exists_{x:D}\,\alpha\end{array}$$

Example

- No $\mathsf{a}(n)$ action with $n < 10$ is allowed to occur:

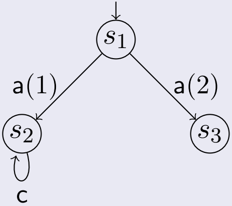
$$\forall_{n:\mathbb{N}}(n < 10) \implies [\mathit{true}^{*}\cdot\mathsf{a}(n)]\mathit{false}$$

- All $\mathsf{a}(n)$ actions can be followed by $\mathsf{a}(n{+}1)$ actions:

$$\forall_{n:\mathbb{N}}[\mathit{true}^{*}\cdot\mathsf{a}(n)]\langle\mathit{true}^{*}\cdot\mathsf{a}(n{+}1)\rangle\mathit{true}$$

Temporal Logic

Exercise

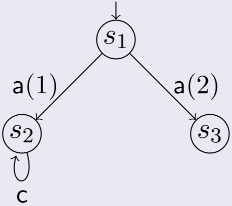


Which of the following holds:

- $s_1 \models \exists_{n:\mathbb{N}}\, [\mathsf{a}(n)]\langle \mathsf{c}\rangle \mathit{true}$
- $s_1 \models [\exists_{n:\mathbb{N}}\, \mathsf{a}(n)]\langle \mathsf{c}\rangle \mathit{true}$

Temporal Logic

Exercise



Which of the following holds:

- $s_1 \models \exists_{n:\mathbb{N}}\, [\mathsf{a}(n)]\langle \mathsf{c}\rangle \mathit{true}$ Yes.
- $s_1 \models [\exists_{n:\mathbb{N}}\, \mathsf{a}(n)]\langle \mathsf{c}\rangle \mathit{true}$ No.

Temporal Logic

Patterns coding for functional properties:

- *Invariance:* . . . . $[true^*]\psi$
- *Safety:* . . . . $[\rho]false$
- *Attainability:* . . . . $\langle\rho\rangle true$
- *Fair reachability:* . . . . $[\rho \cdot (\neg \mathsf{a})^*]\langle(\neg \mathsf{a})^* \cdot \mathsf{a}\rangle true$

Outside regular formulae (but still valid μ-calculus formulae):

- *Inevitability of* a: . . . . $\mu X.\ [\neg \mathsf{a}]X \wedge \langle true\rangle true$
- *Finitely many* a *actions*: . . . . $\mu X.\ \nu Y.\ [\mathsf{a}]X \wedge [\neg \mathsf{a}]Y$
- *Infinitely often action* a: . . . . $\nu X.\ \mu Y.\ \langle \mathsf{a}\rangle X \vee \langle\neg \mathsf{a}\rangle Y$
- ψ *holds along* ρ*-paths while* ϕ *fails*: . . $\nu X.\ \phi \vee (\psi \wedge [\rho]X)$

Outline

1. Basic process algebra
2. Parallelism and abstraction
3. Processes with data
4. Linear processes
5. Temporal Logic
6. Verification
7. Toolset overview and demo
8. Hands-on experience
9. Wrap-up
10. Industrial case studies

Verification

Model Checking Problem

Given a model with initial state s and a formula ϕ,
decide (compute) whether $s \models \phi$ holds or not.

- infinity in specifications . . . . $\mathsf{C}(n{:}\mathbb{N}) = \mathsf{a}(n) \cdot \mathsf{C}(n{+}1)$
- infinity in μ-calculus . . . . . . $\nu X(n{:}\mathbb{N} = 0).\ \langle \mathsf{a}(n) \rangle X(n{+}1)$

Verification

Model Checking Problem

Given a model with initial state s and a formula ϕ,
decide (compute) whether $s \models \phi$ holds or not.

- infinity in specifications .... $\mathsf{C}(n{:}\mathbb{N}) = \mathsf{a}(n) \cdot \mathsf{C}(n{+}1)$
- infinity in μ-calculus ...... $\nu X(n{:}\mathbb{N} = 0).\ \langle \mathsf{a}(n) \rangle X(n{+}1)$

mCRL2 Model Checking Rationale

The two sources of infinity require symbolic techniques to make model checking tractable in practice .... PBESs

Verification

Equation Systems

Sequences of equations of the following form:

$$(\mu X(x_1{:}D_1, \ldots, x_n{:}D_n) = \phi)$$

or

$$(\nu X(x_1{:}D_1, \ldots, x_n{:}D_n) = \phi)$$

- X is a (sorted) predicate variable;
- ϕ is a predicate in which predicate variables occur.

Verification

Equation Systems

Sequences of equations of the following form:

$$(\mu X(x_1{:}D_1, \ldots, x_n{:}D_n) = \phi)$$

or

$$(\nu X(x_1{:}D_1, \ldots, x_n{:}D_n) = \phi)$$

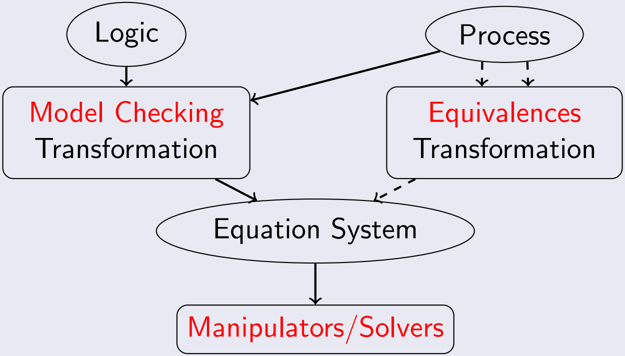
- X is a (sorted) predicate variable;
- ϕ is a predicate in which predicate variables occur.

Example

$$\begin{array}{l}\big(\nu X(n{:}\mathbb{N}) = \forall m{:}\mathbb{N}.\ m \leq 10 \implies Y(n{+}m)\big) \\ \big(\mu Y(n{:}\mathbb{N}) = X(n+1)\big)\end{array}$$

Verification

Methodology



Logic

Process

Model Checking
Transformation

Equivalences
Transformation

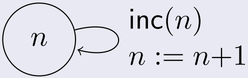
Equation System

Manipulators/Solvers

Verification

Example (Infinite State Counter System)

act inc:$\mathbb{N}$;
proc $\mathsf{C}(n{:}\mathbb{N}) = \mathsf{inc}(n) \cdot \mathsf{C}(n{+}1)$;
init $\mathsf{C}(0)$;



- Absence of deadlock: . . . . $\mathsf{C}(0) \overset{?}{\models} [\mathit{true}^*]\langle \mathit{true}\rangle \mathit{true}$
- Equation system encoding absence of deadlock:
  . . . . $(\nu X(n{:}\mathbb{N}) = X(n+1))$
- Note: $X(0) = \mathit{true}$ iff $\mathsf{C}(0)$ is deadlock-free.

Verification

- Solving equation systems is generally undecidable;
- Decidable fragment: Boolean Equation Systems;
- PBES manipulations:
  - logic rewriting, e.g.:

$$\phi = \psi \implies (\nu X(d{:}D) = \phi) \equiv (\nu X(d{:}D) = \psi)$$

  - strengthen/weaken equations, e.g.;

$$\phi \sqsubseteq \psi \implies (\nu X(d{:}D) = \phi) \leq (\nu X(d{:}D) = \psi)$$

  - Gauß elimination + symbolic approximation;
  - invariants;
  - instantiation to BES.

Verification

Example (Symbolic approximation)

Equation coding absence of deadlock for the counter:

$$\big(\nu X(n{:}\mathbb{N}) = X(n+1)\big)$$

Computing the solution to X using symbolic approximation:

Denote the i^{th} approximant of X by X^i:

- $X^0 \quad = \mathit{true}$
- $X^1 \quad = X(n+1)[X := \mathit{true}]$
  $\quad\quad = \mathit{true}$

Solution to X is true, since $X^0 = X^1$;
Conclusion: the counter system is deadlock-free

Verification

- Tools for Gauß Elimination + Symbolic Approximation:
  - MUCHECK (μCRL), and
  - PBESSOLVE (mCRL2, still under development);
- Successful case studies with MUCHECK:
  - ABP with infinitely large data domain (instead of the usual 2 elements);
  - Bakery Protocol infinite state (natural numbers);
  - EUV Wafer Handler Controller;
  - FireWire;
- Slow when complex data is involved;
- On finite state-spaces, symbolic approximation is often (not always!) outperformed by explicit state techniques.

Verification

Example (Instantiation)

$$\big(\nu X(n{:}\mathbb{N}) = n \leq 2 \wedge Y(n)\big)\ \big(\mu Y(n{:}\mathbb{N}) = \mathsf{odd}(n) \vee X(n{+}1)\big)$$

Instantiation to BES for solution of $X(0)$:

1. $X(0) = 0 \leq 2 \wedge Y(0) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots = Y(0)$
2. $Y(0) = \mathsf{odd}(0) \vee X(1) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots = X(1)$
3. $X(1) = 1 \leq 2 \wedge Y(1) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots = Y(1)$
4. $Y(1) = \mathsf{odd}(1) \vee X(2) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots = \mathit{true}$

$X(0) \mapsto X_0 \qquad X(1) \mapsto X_1 \qquad Y(0) \mapsto Y_0 \qquad Y(1) \mapsto Y_1$

BES: $(\nu X^0 = Y^0)\ (\nu X^1 = Y^1)\ (\mu Y_0 = X^1)\ (\mu Y_1 = \mathit{true})$

Verification

- Instantiation is akin to state-space exploration;
- Algorithms for solving BESs:
  - Gauß Elimination (no symbolic approximation needed!);
  - Small Progress Measures;
  - . . .
- Linear time algorithms for alternation-free BESs exist;
- Tool implementing instantiation and BES solving: PBES2BOOL (mCRL2);
- Applicable to all finite state systems and formulae;
- Remarkable: instantiation and solving can outperform state space exploration.

Verification

Instantiation may not terminate: $\big(\nu X(n{:}\mathbb{N}) = X(n+1)\big)$

- Instantiation starting at e.g. $X(2)$
- $X(3)$ occurs in $(X(n+1)[n := 2])$
- $X(4)$ occurs in $(X(n+1)[n := 3])$
- *etcetera*

Observe: parameter n is non-influential and can be removed (tool: $\textsc{pbesparelm}$):

$$\big(\nu X(n{:}\mathbb{N}) = X(n+1)\big) \approx \big(\nu X = X\big)$$

Note: n cannot be removed in:
proc $\mathsf{C}(n{:}\mathbb{N}) = \mathsf{inc}(n) \cdot \mathsf{C}(n{+}1)$;

Verification

Open Ends

- Develop tooling to support invariants;
- Exploit confluence and symmetry for PBESs;
- Conduct timed verifications using PBESs;
- Transfer regions techniques from Timed Automata;
- Develop (and implement) new patterns;
- Connect to theorem proving technology.

Verification

Some References

1. A. Mader, *Verification of Modal Properties Using Boolean Equation Systems*, 1997.
2. R. Mateescu, *Local model-checking of an alternation-free value-based modal mu-calculus*, 1998.
3. J.F. Groote and T.A.C. Willemse, *Verification of temporal properties of processes in a setting with data*, 2005.
4. J.F. Groote and T.A.C. Willemse, *Parameterised Boolean Equation Systems*, 2005.
5. M.M. Gallardo, C. Joubert, and P. Merino *Implementing influence analysis using parameterised boolean equation systems*, 2006.
6. T. Chen, B. Ploeger, J. van de Pol, and T.A.C. Willemse, *Equivalence checking for infinite systems using parameterized boolean equation systems*, 2007.
7. S.M. Orzan and T.A.C. Willemse, *Invariants for parameterised boolean equation systems*, 2008.

Outline

1. Basic process algebra
2. Parallelism and abstraction
3. Processes with data
4. Linear processes
5. Temporal Logic
6. Verification
7. Toolset overview and demo
8. Hands-on experience
9. Wrap-up
10. Industrial case studies

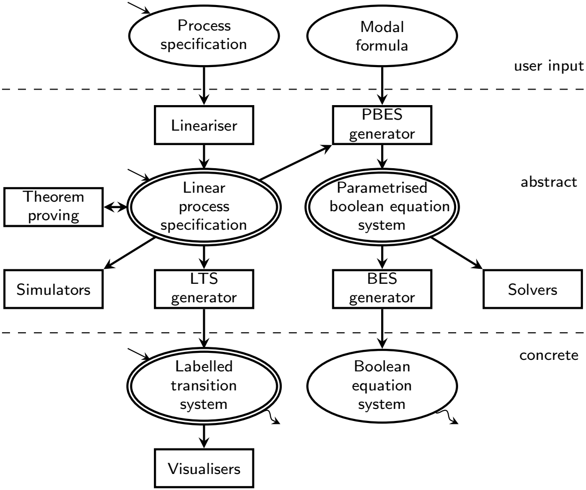
Toolset overview

Introduction

- The mCRL2 toolset can be used for modelling, validation and verification of concurrent systems and protocols.
- Developed at the department of Mathematics and Computer Science of the Technische Universiteit Eindhoven, in collaboration with LaQuSo and CWI.
- The mCRL2 toolset is available for the following platforms:
    - Microsoft Windows
    - Linux
    - Mac OS X
    - FreeBSD
    - Solaris
- Available at http://mcrl2.org

Toolset overview

Tool categories



Toolset overview

Linear process specifications

LPS tools:

- Generation:
    - mcrl22lps: Linearise a process specification
- Information:
    - lpsinfo: Information about an LPS
    - lpspp: Pretty prints an LPS
- Simulation:
    - sim: Text based simulation of an LPS
    - xsim: Graphical simulation of an LPS

Toolset overview

Linear process specifications (2)

LPS tools:

- Optimisation:
    - lpsconstelm: Removes constant process parameters
    - lpsparelm: Removes irrelevant process parameters
    - lpssuminst: Instantiate sum operators
    - lpssumelm: Removes superfluous sum operators
    - lpsactionrename: Renaming of actions
    - lpsconfcheck: Marks confluent tau summands
    - lpsinvelm: Removes violating summands on invariants
    - lpsbinary: Replaces finite sort variables by vectors of boolean variables
    - lpsrewr: Rewrites data expressions of an LPS
    - lpsuntime: Removes time from an LPS

Toolset overview

Labelled transition systems

LTS tools:

- Generation:
  - lps2lts: Generates an LTS from an LPS
- Information and visualisation:
  - ltsinfo: Information about an LTS
  - tracepp: View traces generated by sim/xsim or lps2lts
  - ltsgraph: 2D LTS graph based visualisation
  - ltsview: 3D LTS state based clustered visualisation
  - diagraphica: Multivariate state visualisation and simulation analysis for LTSs
- Comparison, conversion and minimisation:
  - ltscompare: Compares two LTSs with respect to an equivalence or preorder
  - ltsconvert: Converts and minimises an LTS

Toolset overview

Parameterised boolean equation systems

PBES tools:

- Generation:
    - lps2pbes: Generates a PBES from an LPS and a temporal formula
    - txt2pbes: Parses a textual description of a PBES
- Information:
    - pbesinfo: Information about a PBES
    - pbes2pp: Pretty prints a PBES
- Solving:
    - pbes2bool: Solves a PBES
- Optimisation:
    - pbesrewr: Rewrite data expressions in a PBES

Toolset overview

Import and export

Import and export tools:

- chi2mcrl2: Translates a χ specification to an mCRL2 specification
- pnml2mcrl2: Translates a Petri net to an mCRL2 specification
- tbf2lps: Translates a μCRL LPE to an mCRL2 LPS
- formcheck : Checks whether a boolean data expression holds
- lps2torx: Provide TorX explorer interface to an LPS

Toolset demo: dining philosophers

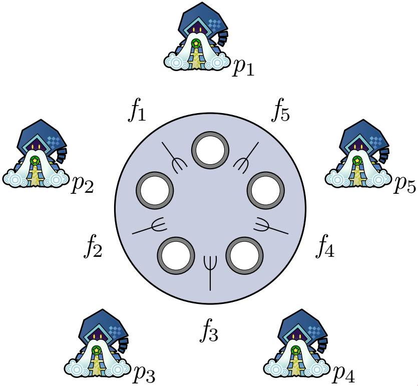
Dining philosophers:

1. Problem description
2. Model the problem
3. Verify the problem
4. A solution
5. Verify the solution

Toolset demo: dining philosophers

Problem description

- Illustrative example of a common computing problem in concurrency
- 5 hungry philosophers
- 5 forks in-between the philosophers
- Rules:
    - Philosophers cannot communicate
    - Two forks are needed for eating



Toolset demo: dining philosophers

Problem description (2)

- Deadlock: Every philosopher holds a left fork and waits for a right fork (or vice versa).
- Starvation: If a philosopher cannot acquire two forks he will starve.

The dining philosophers problem is a generic and abstract problem used for explaining various issues which arise in concurrency theory.

- The forks resemble shared resources.
- The philosophers resemble concurrent processes.

Toolset demo: dining philosophers

Modelling the problem: data types

Data type for representing the philosophers and the forks:

$$\begin{array}{lll}\textsf{\textbf{sort}} & \mathit{PhilId} & = \mathbf{struct}\ p_1 \mid p_2 \mid p_3 \mid p_4 \mid p_5; \\ & \mathit{ForkId} & = \mathbf{struct}\ f_1 \mid f_2 \mid f_3 \mid f_4 \mid f_5; \end{array}$$

Function for representing the positions of the forks relative to the philosophers (the left and right fork):

$$\begin{array}{ll}\textsf{\textbf{map}} & \mathit{lf}, \mathit{rf} : \mathit{PhilId} \to \mathit{ForkId}; \\ \textsf{\textbf{eqn}} & \mathit{lf}(p_1) = f_1;\ \mathit{lf}(p_2) = f_2;\ \mathit{lf}(p_3) = f_3; \\ & \mathit{lf}(p_4) = f_4;\ \mathit{lf}(p_5) = f_5; \\ & \mathit{rf}(p_1) = f_5;\ \mathit{rf}(p_2) = f_1;\ \mathit{rf}(p_3) = f_2; \\ & \mathit{rf}(p_4) = f_3;\ \mathit{rf}(p_5) = f_4; \end{array}$$

Toolset demo: dining philosophers

Modelling the problem: individual processes

Modelling the behaviour of the philosophers:

- eat(p): philosopher p eats
- get(p, f): philosopher p takes up fork f
- put(p, f): philosopher p puts down fork f

$$
\begin{array}{ll}
\textbf{act} & \mathsf{get}, \mathsf{put} : \mathit{PhilId} \times \mathit{ForkId}; \\
 & \mathsf{eat} : \mathit{PhilId}; \\
\textbf{proc} & \mathsf{Phil}(p : \mathit{PhilId}) = \\
 & \quad (\mathsf{get}(p, \mathit{lf}(p)) \cdot \mathsf{get}(p, \mathit{rf}(p)) + \mathsf{get}(p, \mathit{rf}(p)) \cdot \mathsf{get}(p, \mathit{lf}(p))) \\
 & \quad \cdot\, \mathsf{eat}(p) \\
 & \quad \cdot\, (\mathsf{put}(p, \mathit{lf}(p)) \cdot \mathsf{put}(p, \mathit{rf}(p)) + \mathsf{put}(p, \mathit{rf}(p)) \cdot \mathsf{put}(p, \mathit{lf}(p))) \\
 & \quad \cdot\, \mathsf{Phil}(p);
\end{array}
$$

Toolset demo: dining philosophers

Modelling the problem: individual processes

Modelling the behaviour of the forks:

- up(p, f): fork f is picked up by philosopher p
- down(p, f): fork f is put down by philosopher p

$$\begin{array}{ll}
\textbf{act} & \mathsf{up}, \mathsf{down} : \mathit{PhilId} \times \mathit{ForkId}; \\
\textbf{proc} & \mathsf{Fork}(f : \mathit{ForkId}) = \\
 & \qquad \sum_{p:\mathit{Phil}} \mathsf{up}(p, f) \cdot \mathsf{down}(p, f) \cdot \mathsf{Fork}(f);
\end{array}$$

Toolset demo: dining philosophers

Modelling the problem: communication and initialisation

Complete specification:

- put all forks and philosophers in parallel
- synchronise on actions get and up,
  and on actions put and down

$$
\begin{array}{ll}
\textbf{act} & \mathsf{lock}, \mathsf{free} : \mathit{PhilId} \times \mathit{ForkId}; \\
\textbf{init} & \nabla(\{\mathsf{lock}, \mathsf{free}, \mathsf{eat}\}, \\
 & \quad \Gamma(\{\mathsf{get}|\mathsf{up} \rightarrow \mathsf{lock}, \mathsf{put}|\mathsf{down} \rightarrow \mathsf{free}\}, \\
 & \qquad \mathsf{Phil}(p_1) \parallel \mathsf{Phil}(p_2) \parallel \mathsf{Phil}(p_3) \parallel \mathsf{Phil}(p_4) \parallel \mathsf{Phil}(p_5) \parallel \\
 & \qquad \mathsf{Fork}(f_1) \parallel \mathsf{Fork}(f_2) \parallel \mathsf{Fork}(f_3))) \parallel \mathsf{Fork}(f_4) \parallel \mathsf{Fork}(f_5) \\
 & \qquad));
\end{array}
$$

Toolset demo: dining philosophers

Analysing the model

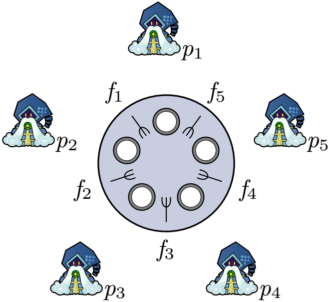
- Linearisation:
```
mcrl22lps -vD dining5.mcrl2 dining5.lps
```
- Sum instantation:
```
lpssuminst -v dining5.lps dining5.sum.lps
```
- Constant elimination:
```
lpsconstelm -v dining5.sum.lps dining5.sum.const.lps
```
- Parameter elimination:
```
lpsparelm -v dining5.sum.const.lps
    dining5.sum.const.par.lps
```
- Generate state space:
```
lps2lts -vD dining5.sum.const.lps dining5.sum.const.lts
```
- Deadlock detected!

Toolset demo: dining philosophers

A Possible solution: the waiter

Waiter:

- Decides whether a philosopher may pick up two forks
- Only allowed when less than four forks are in use



Toolset demo: dining philosophers

Modelling the solution: actions

New actions:

- ack(p): philosopher p takes the opportunity to pick up two forks and eat
- done(p): philosopher p signal the waither that he is done eating and has put down both forks

act r_ack, s_ack, ack : $Phil$;
r_done, s_done, done : $Phil$;

Toolset demo: dining philosophers

Modelling the solution: the waiter

Modelling the behaviour of the waiter:

$$
\begin{array}{ll}
\textbf{proc} & \mathsf{Waiter}(n:\mathbb{N}) = \\
& \quad (n < 4) \rightarrow \sum_{p:Phil} \mathsf{s\_ack}(p) \cdot \mathsf{Waiter}(n{+}2) \\
& + \ (n > 1) \rightarrow \sum_{p:Phil} \mathsf{r\_done}(p) \cdot \mathsf{Waiter}(Int2Nat(n{-}2));
\end{array}
$$

Toolset demo: dining philosophers

Modelling the solution: the philosophers

Extend the philosopher process:

$$
\begin{array}{ll}
\textbf{proc} & \mathsf{Phil}(p : \mathit{PhilId}) = \\
& \quad \mathsf{r\_ack}(p) \\
& \cdot\ (\mathsf{get}(p, \mathit{lf}(p)) \cdot \mathsf{get}(p, \mathit{rf}(p)) + \mathsf{get}(p, \mathit{rf}(p)) \cdot \mathsf{get}(p, \mathit{lf}(p))) \\
& \cdot\ \mathsf{eat}(p) \\
& \cdot\ (\mathsf{put}(p, \mathit{lf}(p)) \cdot \mathsf{put}(p, \mathit{rf}(p)) + \mathsf{put}(p, \mathit{rf}(p)) \cdot \mathsf{put}(p, \mathit{lf}(p))) \\
& \cdot\ \mathsf{s\_done}(p) \\
& \cdot\ \mathsf{Phil}(p);
\end{array}
$$

Toolset demo: dining philosophers

Modelling the solution: communication and initialisation

Complete specification:

init $\nabla(\{\mathsf{lock}, \mathsf{free}, \mathsf{eat}, \mathsf{ack}, \mathsf{done}\},$

$\quad\Gamma(\{\mathsf{get}|\mathsf{up} \rightarrow \mathsf{lock}, \mathsf{put}|\mathsf{down} \rightarrow \mathsf{free}$

$\qquad\mathsf{r\_ack}|\mathsf{s\_ack} \rightarrow \mathsf{ack}, \mathsf{r\_done}|\mathsf{s\_done} \rightarrow \mathsf{done},$

$\qquad\mathsf{Phil}(p_1) \parallel \mathsf{Phil}(p_2) \parallel \mathsf{Phil}(p_3) \parallel \mathsf{Phil}(p_4) \parallel \mathsf{Phil}(p_5) \parallel$

$\qquad\mathsf{Fork}(f_1) \parallel \mathsf{Fork}(f_2) \parallel \mathsf{Fork}(f_3) \parallel \mathsf{Fork}(f_4) \parallel \mathsf{Fork}(f_5) \parallel$

$\qquad\mathsf{Waiter}(0)$

$\qquad));$

Toolset demo: dining philosophers

Verifying the solution

- Deadlock freedom: Yes

$$[true^*]\,\langle true\rangle\, true$$

1. lps2pbes --formula=nodeadlock.mcf dining5_waiter.lps dining5_waiter_nd.pbes
2. pbes2bool dining5_waiter_nd.pbes

- Starvation freedom: Yes

$$\forall_{p:Phil}\,[true^* \cdot (\neg \mathsf{eat}(p))^*]\,\langle(\neg \mathsf{eat}(p))^* \cdot \mathsf{eat}(p)\rangle\, true$$

1. lps2pbes --formula=nostarvation.mcf dining5_waiter.lps dining5_waiter_ns.pbes
2. pbes2bool dining5_waiter_ns.pbes

Outline

1. Basic process algebra
2. Parallelism and abstraction
3. Processes with data
4. Linear processes
5. Temporal Logic
6. Verification
7. Toolset overview and demo
8. **Hands-on experience**
9. Wrap-up
10. Industrial case studies

Hands-on experience

Start up:

- Boot the laptop into **Ubuntu**!
- Log in as usual (local).
- Start a terminal window and go to directory:
  - `~/Desktop/VendingMachine` for the vending machine
  - `~/Desktop/RopeBridge` for the rope bridge

  Directories are also visible on your desktop.

Information on mCRL2 language/tools can be found:

- in your handouts
- on the website: `http://mcrl2.org`

Good luck!

Outline

Outline

Industrial case studies

Overview

Some industrial case studies:

- Océ: automated document feeder
- Add-controls: distributed system for lifting trucks
- CVSS: automated parking garage
- Vitatron: pacemaker
- AIA: ITP load-balancer

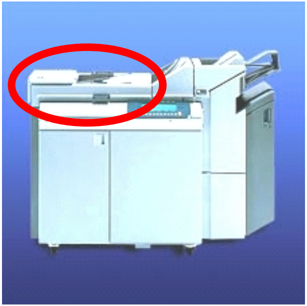
Industrial case studies

Océ: automatic document feeder

- Feed documents to the scanner automatically
- One sheet at a time
- Prototype implementation

Analysis:

- Model: μCRL
- Verification: CADP
  μ-calculus model checking
- Size: 350,000 states
  and 1,100,000 transitions
- Actual errors found: 2



Industrial case studies

Add-controls: distributed system for lifting trucks

- Each lift has a controller
- Controls are connected in a circular network
- 3 errors found after testing by the developers

Analysis:

- Model: μCRL
- Verification: μ-calculus
- Actual errors found: 4

Lifts	States	Transitions
2	383	716
3	7,282	18,957
4	128,901	419,108
5	2,155,576	8,676,815

Industrial case studies

CVSS: automated parking garage

An automated parking garage:

Industrial case studies

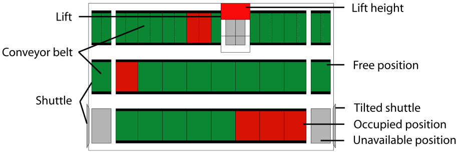
CVSS: automated parking garage (2)

Verified design:

- Design of the control software
- Verified the safety layer of this design

Analysis:

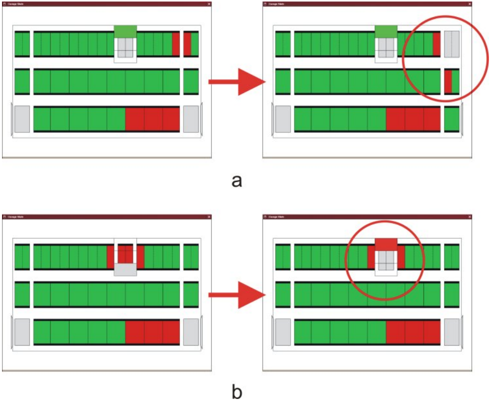
- Design: 991 lines of mCRL2
- Verification: 217 lines of mCRL2
- Size: 3.3 million states and 98 million transitions
- Simulation using custom built visualisation plugin



Industrial case studies

CVSS: automated parking garage (3)

Design flaws detected using the visualisation plugin:



Industrial case studies

Vitatron: pacemaker

- Controlled by firmware
- Must deal with all possible rates and arrhythmias
- Firmware design

Analysis:

- Model: mCRL2 (and Uppaal)
- Verification: mCRL2 state space generation and μ-calculus model checking
- Size:
    - full model: 500 million states
    - suspicious part: 714.464 states
- Actual errors found: 1 (known)

Industrial case studies

AIA: ITP load-balancer

- ITP: Intelligent Text Processing
- Print job distribution over document processors
- 7,500 lines of C code

Analysis:

- Load balancing part
- Model: mCRL2
- Verification: mCRL2 state space generation
- Actual errors found: 6
- Size: 1.9 billion states and 38.9 billion transitions
- LaQuSo certification