Modelling and Analysis of SPL with mCRL2

Modelling and Analysis of Variability in Software Product Lines

ISTI–CNR, Pisa

June 26–30, 2017

These slides are largely based on slides by Erik de Vink
Overview

- FTS: Featured Transition Systems
- The modal $\mu$-calculus $\mu L$ over LTS
- The mCRL2 language and toolset
- A case study: the coffee machine
- Family-based verification
  - A feature $\mu$-calculus $\mu L_f$ over FTS
  - A $\mu$-calculus with data $\mu L_{FO}$ over parametrised LTS
  - From $\mu L_f$ to $\mu L_{FO}$ (and back to $\mu L$)
  - Family-based partitioning algorithm for $\mu L_f$
  - Case study: minepump SPL benchmark
Featured Transition Systems

Modelling and Analysis of Variability in Software Product Lines

ISTI–CNR, Pisa

June 26–30, 2017

These slides are largely based on slides by Erik de Vink
Featured Transition Systems

FTS $F = (S, \theta, s_*)$ over actions $\mathcal{A}$ and features $\mathcal{F}$

- $S$ a finite set of states
- $\theta : S \times \mathcal{A} \times S \rightarrow \mathbb{B}[\mathcal{F}]$ the transition constraint function
- $s_* \in S$ the initial state

LTS $L = (S, \rightarrow, s_*)$ over actions $\mathcal{A}$

- $S$ a finite set of states
- $\rightarrow \subseteq S \times \mathcal{A} \times S$ the transition relation
- $s_* \in S$ the initial state

Classen et al., Model Checking Lots of Systems, ICSE’10
An example FTS

features v, b, s, t, f, c
12 products
Products from Figure 1:

(a) Basic = v, b, s
(b) Tea and soda = v, b, s, t
(c) Cancel function = v, b, s, c
(d) Soda for free = v, b, s, f
An example product LTS

 Featured Transition Systems
FTS $F = (S, \theta, s_\star)$ over actions $\mathcal{A}$ and features $\mathcal{F}$

- $S$ a finite set of states
- $\theta : S \times \mathcal{A} \times S \rightarrow \mathbb{B}[\mathcal{F}]$ the transition constraint function
- $s_\star \in S$ the initial state

Projected LTS $F|_p = (S, \rightarrow, s_\star)$ of product $p$

Transition relation $\rightarrow_{F|_p} \subseteq S \times \mathcal{A} \times S$ such that

$$s \xrightarrow{a}_{F|_p} t \text{ iff } p \models \theta(s, a, t)$$
Example FTS and projection

FTS $VM$

LTS $VM\{v, b, s, t\}$
Vending machine properties

- Is the system deadlock-free?
- Does control always return to the initial state?
- Will a return or open follow a pay?
- Will a free never precede cancel or tea?
Model checking with the $\mu$-calculus

Modelling and Analysis of Variability in Software Product Lines

ISTI–CNR, Pisa

June 26–30, 2017

These slides are largely based on slides by Erik de Vink
The modal $\mu$-calculus

set of actions $A$ and set of variables $X$

$\mu$-calculus $\mu L$ over $A$ and $X$, formula $\varphi \in \mu L$ given by

$$\varphi ::= \bot \mid T \mid$$
$$\neg \varphi \mid \varphi \lor \psi \mid \varphi \land \psi \mid$$
$$\langle a \rangle \varphi \mid [a] \varphi \mid$$
$$X \mid \mu X.\varphi \mid \nu X.\varphi$$

duality $\langle a \rangle \varphi \equiv \neg [a] \neg \varphi$, positive normal form avoids negation

for $\mu X.\varphi$ and $\nu X.\varphi$, free occurrences of $X$ in $\varphi$ in the scope of an even number of negations
Examples of $\mu L$-formulas

$\langle a \rangle ( [b] \bot \land \langle c \rangle \top )$

“it is possible to execute action $a$, after which action $b$ cannot be executed whereas action $c$ can”
Examples of $\mu L$-formulas

- $\langle a \rangle (\lbrack b \rbrack \bot \land \langle c \rangle T)$
  
  “it is possible to execute action $a$, after which action $b$ cannot be executed whereas action $c$ can”

- $\mu X. (\langle a \rangle X \lor \langle b \rangle T)$
  
  “there exists a finite repetition of executing action $a$, followed by an execution of action $b$”

$\mu X$: finite looping
Examples of $\mu L$-formulas

- $\langle a \rangle (\llbracket b \rrbracket \bot \land \langle c \rangle \top)$
  
  “it is possible to execute action $a$, after which action $b$ cannot be executed whereas action $c$ can”

- $\mu X. (\langle a \rangle X \lor \langle b \rangle \top)$
  
  “there exists a finite repetition of executing action $a$, followed by an execution of action $b$”

- $\nu X. (\mu Y. [a] Y \land [b] X)$
  
  “action $b$ is executed infinitely often on all infinite executions containing actions $a$ and $b$”

$\mu X$: finite looping vs. $\nu X$: infinite looping
Modalities: $\langle a \rangle \varphi$ vs. $[a] \varphi$
Modalities: $\langle a \rangle \varphi$ vs. $[a] \varphi$

<table>
<thead>
<tr>
<th></th>
<th>$\langle a \rangle \varphi$</th>
<th>$[a] \varphi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>I, III</td>
<td>III, IV</td>
</tr>
<tr>
<td>true</td>
<td>II, IV</td>
<td>I, II</td>
</tr>
</tbody>
</table>

\[ \mu \text{-calculus} \]
Give a modal formula expressing: in the current state action $a$ can be done, followed by action $b$. Moreover, after this action $a$ no action $c$ is allowed.
Give a modal formula expressing: in the current state action $a$ can be done, followed by action $b$. Moreover, after this action $a$ no action $c$ is allowed.

$$\langle a \rangle (\langle b \rangle \text{true} \land [c]\text{false})$$

---

Formal semantics of $\mu L$

state sets $U \in sSet = 2^S$, state-based environments $\varepsilon \in sEnv = X' \rightarrow sSet$

semantics $\llbracket \cdot \rrbracket_L : \mu L \rightarrow sEnv \rightarrow sSet$

\[
\begin{align*}
\llbracket \bot \rrbracket_L(\varepsilon) &= \emptyset \\
\llbracket \top \rrbracket_L(\varepsilon) &= S \\
\llbracket \neg \varphi \rrbracket_L(\varepsilon) &= S \setminus \llbracket \varphi \rrbracket_L(\varepsilon) \\
\llbracket (\varphi \lor \psi) \rrbracket_L(\varepsilon) &= \llbracket \varphi \rrbracket_L(\varepsilon) \cup \llbracket \psi \rrbracket_L(\varepsilon) \\
\llbracket (\varphi \land \psi) \rrbracket_L(\varepsilon) &= \llbracket \varphi \rrbracket_L(\varepsilon) \cap \llbracket \psi \rrbracket_L(\varepsilon) \\
\llbracket \langle a \rangle \varphi \rrbracket_L(\varepsilon) &= \{ s | \exists t : s \xrightarrow{a} t \land t \in \llbracket \varphi \rrbracket_L(\varepsilon) \} \\
\llbracket [a] \varphi \rrbracket_L(\varepsilon) &= \{ s | \forall t : s \xrightarrow{a} t \rightarrow t \in \llbracket \varphi \rrbracket_L(\varepsilon) \} \\
\llbracket X \rrbracket_L(\varepsilon) &= \varepsilon(X) \\
\llbracket \mu X. \varphi \rrbracket_L(\varepsilon) &= \text{lfp}( U \mapsto \llbracket \varphi \rrbracket_L(\varepsilon[U/X])) \\
\llbracket \nu X. \varphi \rrbracket_L(\varepsilon) &= \text{gfp}( U \mapsto \llbracket \varphi \rrbracket_L(\varepsilon[U/X]))
\end{align*}
\]

variant environment $\varepsilon[U/X]$: $\varepsilon(Y)$ for $Y \neq X$, the set $U$ for $X$
Formal semantics of $\mu L$

state sets $U \in s\text{Set} = 2^S$, state-based environments $\varepsilon \in s\text{Env} = \mathcal{X} \to s\text{Set}$

semantics $\llbracket \cdot \rrbracket_L : \mu L \to s\text{Env} \to s\text{Set}$

\[
\begin{align*}
\llbracket \bot \rrbracket_L(\varepsilon) &= \emptyset \\
\llbracket \top \rrbracket_L(\varepsilon) &= S \\
\llbracket \neg \varphi \rrbracket_L(\varepsilon) &= S \setminus \llbracket \varphi \rrbracket_L(\varepsilon) \\
\llbracket (\varphi \lor \psi) \rrbracket_L(\varepsilon) &= \llbracket \varphi \rrbracket_L(\varepsilon) \cup \llbracket \psi \rrbracket_L(\varepsilon) \\
\llbracket (\varphi \land \psi) \rrbracket_L(\varepsilon) &= \llbracket \varphi \rrbracket_L(\varepsilon) \cap \llbracket \psi \rrbracket_L(\varepsilon)
\end{align*}
\]

variant environment $\varepsilon[U/X]: \varepsilon(Y)$ for $Y \neq X$, the set $U$ for $X$
Formal semantics of $\mu L$

state sets $U \in \text{sSet} = 2^S$, state-based environments $\varepsilon \in \text{sEnv} = \mathcal{X} \to \text{sSet}$

semantics $\llbracket \cdot \rrbracket_L : \mu L \to \text{sEnv} \to \text{sSet}$

\[
\llbracket \langle a \rangle \varphi \rrbracket_L(\varepsilon) = \{ s \mid \exists t : s \xrightarrow{a} t \land t \in \llbracket \varphi \rrbracket_L(\varepsilon) \}
\]

\[
\llbracket [a] \varphi \rrbracket_L(\varepsilon) = \{ s \mid \forall t : s \xrightarrow{a} t \Rightarrow t \in \llbracket \varphi \rrbracket_L(\varepsilon) \}
\]

variant environment $\varepsilon[U/X]$: $\varepsilon(Y)$ for $Y \neq X$, the set $U$ for $X$
Formal semantics of $\mu L$

state sets $U \in sSet = 2^S$, state-based environments $\varepsilon \in sEnv = \mathcal{X} \to sSet$

semantics $\llbracket \cdot \rrbracket_L : \mu L \to sEnv \to sSet$

$$
\begin{align*}
\llbracket X \rrbracket_L(\varepsilon) &= \varepsilon(X) \\
\llbracket \mu X.\varphi \rrbracket_L(\varepsilon) &= \text{lfp}( U \mapsto \llbracket \varphi \rrbracket_L(\varepsilon[U/X]) ) \\
\llbracket \nu X.\varphi \rrbracket_L(\varepsilon) &= \text{gfp}( U \mapsto \llbracket \varphi \rrbracket_L(\varepsilon[U/X]) )
\end{align*}
$$

variant environment $\varepsilon[U/X]$: $\varepsilon(Y)$ for $Y \neq X$, the set $U$ for $X$
Formal semantics of $\mu L$

state sets $U \in sSet = 2^S$, state-based environments $\varepsilon \in sEnv = \mathcal{X} \rightarrow sSet$

semantics $\llbracket \cdot \rrbracket_L : \mu L \rightarrow sEnv \rightarrow sSet$

\[
\begin{align*}
\llbracket \bot \rrbracket_L(\varepsilon) &= \emptyset \\
\llbracket \top \rrbracket_L(\varepsilon) &= S \\
\llbracket \neg \varphi \rrbracket_L(\varepsilon) &= S \setminus \llbracket \varphi \rrbracket_L(\varepsilon) \\
\llbracket (\varphi \lor \psi) \rrbracket_L(\varepsilon) &= \llbracket \varphi \rrbracket_L(\varepsilon) \cup \llbracket \psi \rrbracket_L(\varepsilon) \\
\llbracket (\varphi \land \psi) \rrbracket_L(\varepsilon) &= \llbracket \varphi \rrbracket_L(\varepsilon) \cap \llbracket \psi \rrbracket_L(\varepsilon) \\
\llbracket \langle a \rangle \varphi \rrbracket_L(\varepsilon) &= \{ s \mid \exists t : s \xrightarrow{a} t \land t \in \llbracket \varphi \rrbracket_L(\varepsilon) \} \\
\llbracket [a] \varphi \rrbracket_L(\varepsilon) &= \{ s \mid \forall t : s \xrightarrow{a} t \Rightarrow t \in \llbracket \varphi \rrbracket_L(\varepsilon) \} \\
\llbracket X \rrbracket_L(\varepsilon) &= \varepsilon(X) \\
\llbracket \mu X . \varphi \rrbracket_L(\varepsilon) &= \text{lfp}( U \mapsto \llbracket \varphi \rrbracket_L(\varepsilon[U/X]) ) \\
\llbracket \nu X . \varphi \rrbracket_L(\varepsilon) &= \text{gfp}( U \mapsto \llbracket \varphi \rrbracket_L(\varepsilon[U/X]) )
\end{align*}
\]

variant environment $\varepsilon[U/X]$: $\varepsilon(Y)$ for $Y \neq X$, the set $U$ for $X$
The Knaster-Tarski theorem

complete lattice $(X, \sqsubseteq)$, monotone function $f : X \to X$

- $lfp(f) = \bigcap\{ x \in X \mid f(x) \sqsubseteq x \}$
- $gfp(f) = \bigcup\{ x \in X \mid x \sqsubseteq f(x) \}$
complete lattice \((X, \sqsubseteq)\), monotone function \(f : X \to X\)

- \(lfp(f) = \bigcap \{ x \in X \mid f(x) \sqsubseteq x \}\)
- \(gfp(f) = \bigcup \{ x \in X \mid x \sqsubseteq f(x) \}\)

if \(X\) is finite

\[
lfp(f) = \bigcup_{i=0}^{\infty} f^i(\bot) \quad \text{and} \quad gfp(f) = \bigcap_{i=0}^{\infty} f^i(\top)
\]
The Knaster-Tarski theorem

complete lattice \((X, \sqsubseteq)\), monotone function \(f : X \rightarrow X\)

- \(lfp(f) = \bigsqcap \{ x \in X \mid f(x) \sqsubseteq x \}\)
- \(gfp(f) = \bigsqcup \{ x \in X \mid x \sqsubseteq f(x) \}\)

if \(X\) is finite
\[
\begin{align*}
  lfp(f) &= \bigcup_{i=0}^{\infty} f^i(\bot) \quad \text{and} \quad gfp(f) = \bigcap_{i=0} \ f^i(T)
\end{align*}
\]

favorite application
\((2^S, \subseteq)\) for finite set of states \(S\)
Recursive $\mu L$-formulas

$\mu Y.([a + b] Y \lor \langle c \rangle \top)$

$\mu$-$\text{calculus}$
Recursive $\mu L$-formulas

\[
\mu Y.([a + b] Y \lor \langle c \rangle T)
\]

\[
\begin{align*}
Y_0 &= \emptyset \\
Y_1 &= \{s_5\} \\
Y_2 &= \{s_4, s_5\} \\
Y_3 &= \{s_3, s_4, s_5\} \\
Y_4 &= \{s_3, s_4, s_5\} \\
\ldots
\end{align*}
\]
Recursive $\mu L$-formulas

$$\mu Y.([a + b] Y \lor \langle c \rangle T)$$

$Y_0 = \emptyset$
$Y_1 = \{s_5\}$
$Y_2 = \{s_4, s_5\}$
$Y_3 = \{s_3, s_4, s_5\}$
$Y_4 = \{s_3, s_4, s_5\}$

...$

\{s_3, s_4, s_5\} \models$
$$\mu Y.([a + b] Y \lor \langle c \rangle T)$$

any sequence of $a$ and $b$ shows a $c$
Recursive $\mu L$-formulas (cont.)

$$\nu X. (\mu Y. ([a]Y \land \langle b + c \rangle X))$$
Recursive $\mu L$-formulas (cont.)

$\nu X. (\mu Y. ([a] Y \land \langle b + c \rangle X))$

$\mu Y. [a] Y$

$Y_0 = \emptyset$

$Y_1 = \{s_1, s_2, s_3, s_4, s_5\}$

$Y_2 = \{s_0, s_1, s_2, s_3, s_4, s_5\}$

$Y_3 = \{s_0, s_1, s_2, s_3, s_4, s_5\}$

$\ldots$
Recursive $\mu L$-formulas (cont.)

$\nu X. (\mu Y. ([a] Y \land \langle b + c \rangle X))$

$\mu Y. [a] Y$

$Y_0 = \emptyset$

$Y_1 = \{s_1, s_2, s_3, s_4, s_5\}$

$Y_2 = \{s_0, s_1, s_2, s_3, s_4, s_5\}$

$Y_3 = \{s_0, s_1, s_2, s_3, s_4, s_5\}$

\[\ldots\]

$\{s_0, s_1, s_2, s_3, s_4, s_5\} \models \mu Y. [a] Y$

no infinite sequence of $a$
Recursive $\mu L$-formulas (cont.)

$$\nu X. (\mu Y. ([a] Y \land \langle b + c \rangle X))$$
Recursive $\mu L$-formulas (cont.)

$$\nu X.(\mu Y.([a] Y \land \langle b + c \rangle X))$$

\begin{align*}
X_0 &= \{s_0, s_1, s_2, s_3, s_4, s_5\} \\
X_1 &= \{s_0, s_1, s_2, s_3, s_4, s_5\} \cap \{s_1, s_2, s_3, s_4, s_5\} \\
&= \{s_1, s_2, s_3, s_4, s_5\} \\
X_2 &= \{s_0, s_1, s_2, s_3, s_4, s_5\} \cap \{s_1, s_2, s_3, s_4\} \\
&= \{s_1, s_2, s_3, s_4\} \\
X_3 &= \{s_0, s_1, s_2, s_3, s_4, s_5\} \cap \{s_1, s_2, s_3\} \\
&= \{s_1, s_2, s_3\} \\
X_4 &= \{s_0, s_1, s_2, s_3, s_4, s_5\} \cap \{s_1, s_2\} \\
&= \{s_1, s_2\} \\
X_5 &= \{s_0, s_1, s_2, s_3, s_4, s_5\} \cap \{s_1, s_2\} \\
&= \{s_1, s_2\} \\
\ldots
\end{align*}
Recursive $\mu L$-formulas (cont.)

\[ \nu X. (\mu Y.([a] Y \land \langle b + c\rangle X)) \]

\[ X_0 = \{s_0, s_1, s_2, s_3, s_4, s_5\} \]
\[ X_1 = \{s_0, s_1, s_2, s_3, s_4, s_5\} \cap \{s_1, s_2, s_3, s_4, s_5\} \]
\[ = \{s_1, s_2, s_3, s_4, s_5\} \]
\[ X_2 = \{s_0, s_1, s_2, s_3, s_4, s_5\} \cap \{s_1, s_2, s_3, s_4\} \]
\[ = \{s_1, s_2, s_3, s_4\} \]
\[ X_3 = \{s_0, s_1, s_2, s_3, s_4, s_5\} \cap \{s_1, s_2, s_3\} \]
\[ = \{s_1, s_2, s_3\} \]
\[ X_4 = \{s_0, s_1, s_2, s_3, s_4, s_5\} \cap \{s_1, s_2\} \]
\[ = \{s_1, s_2\} \]
\[ X_5 = \{s_0, s_1, s_2, s_3, s_4, s_5\} \cap \{s_1, s_2\} \]
\[ = \{s_1, s_2\} \]
\[ \ldots \]

\[ \{s_1, s_2\} \models \nu X. (\mu Y.([a] Y \land \langle b + c\rangle X)) \]

no infinitely many $a$’s, but infinitely many $b$’s and $c$’s are possible
The mCRL2 language and toolset

Modelling and Analysis of Variability in Software Product Lines

ISTI–CNR, Pisa

June 26–30, 2017

These slides are largely based on slides by Erik de Vink
mCRL2: language and toolset

- Formal, process-algebraic specification of distributed and concurrent systems, associated industrial-strength toolset
- Exploration of $10^6$ states/s, state spaces up to $10^{12}$ states

www.mcrl2.org
mCRL2: language and toolset

- Formal, process-algebraic specification of distributed and concurrent systems, associated industrial-strength toolset
- Exploration of $10^6$ states/s, state spaces up to $10^{12}$ states
- Built-in datatypes (e.g. Bool, Int, Real, Sets, Functions) and user-defined abstract datatypes, parametrised actions

www.mcrl2.org
mCRL2: language and toolset

- Formal, process-algebraic specification of distributed and concurrent systems, associated industrial-strength toolset
- Exploration of $10^6$ states/s, state spaces up to $10^{12}$ states
- Built-in datatypes (e.g. Bool, Int, Real, Sets, Functions) and user-defined abstract datatypes, parametrised actions
- Modal $\mu$-calculus with data (subsuming LTL, CTL, etc.)
mCRL2: language and toolset

- Formal, process-algebraic specification of distributed and concurrent systems, associated industrial-strength toolset
- Exploration of $10^6$ states/s, state spaces up to $10^{12}$ states
- Built-in datatypes (e.g. Bool, Int, Real, Sets, Functions) and user-defined abstract datatypes, parametrised actions
- Modal $\mu$-calculus with data (subsuming LTL, CTL, etc.)
- Visualisation,

www.mcrl2.org
mCRL2: language and toolset

- Formal, process-algebraic specification of distributed and concurrent systems, associated industrial-strength toolset
- Exploration of $10^6$ states/s, state spaces up to $10^{12}$ states
- Built-in datatypes (e.g. Bool, Int, Real, Sets, Functions) and user-defined abstract datatypes, parametrised actions
- Modal $\mu$-calculus with data (subsuming LTL, CTL, etc.)
- Visualisation, behavioural reduction,

www.mcrl2.org
mCRL2: language and toolset

- Formal, process-algebraic specification of distributed and concurrent systems, associated **industrial-strength** toolset
- Exploration of $10^6$ states/s, state spaces up to $10^{12}$ states
- Built-in datatypes (e.g. Bool, Int, Real, Sets, Functions) and user-defined abstract datatypes, **parametrised actions**
- Modal $\mu$-calculus with data (subsuming LTL, CTL, etc.)
- Visualisation,
  - behavioural reduction,
  - model checking

www.mcrl2.org
mCRL2: language and toolset

- Formal, process-algebraic specification of distributed and concurrent systems, associated **industrial-strength** toolset
- Exploration of $10^6$ states/s, state spaces up to $10^{12}$ states
- Built-in datatypes (e.g. Bool, Int, Real, Sets, Functions) and user-defined abstract datatypes, **parametrised actions**
- Modal $\mu$-calculus with data (subsuming LTL, CTL, etc.)
- Visualisation, behavioural reduction, model checking
- **Highly optimised, actively maintained**

---

www.mcrl2.org
mCRL2: language and toolset

- Formal, process-algebraic specification of distributed and concurrent systems, associated **industrial-strength** toolset
- Exploration of $10^6$ states/s, state spaces up to $10^{12}$ states
- Built-in datatypes (e.g. Bool, Int, Real, Sets, Functions) and user-defined abstract datatypes, **parametrised actions**
- Modal $\mu$-calculus with data (subsuming LTL, CTL, etc.)
- Visualisation, behavioural reduction, model checking
- Highly optimised, actively maintained
- Intermediate artifacts user-accessible

www.mcrl2.org
proc Aut(st:Int) =
( st==0 ) -> ( b.Aut(1) + a.Aut(2) ) +
( st==1 ) -> ( c.Aut(3) ) +
( st==2 ) -> ( b.Aut(1) + b.Aut(3) + a.Aut(4) ) +
...
proc Aut(st:Int) =
  ( st==0 ) -> ( b.Aut(1) + a.Aut(2) ) +
  ( st==1 ) -> ( c.Aut(3) ) +
  ( st==2 ) -> ( b.Aut(1) + b.Aut(3) + a.Aut(4) ) +
  ... 

bullet [true*]<true>true: absence of deadlock
bullet [true*.b.true*.a]false: no a after a b
bullet mu Y.(<true>true ∧ ![c] Y): 
eventually c can be done
bullet mu Y.((nu Z.(<b.d.e> Z)) ∨ [true] Y):
eventually, an infinite sequence of bde is possible
act
a, b, c, d, e;

proc Aut(st:Int) =
( st == 0 ) -> ( b.Aut(1) + a.Aut(2) ) +
( st == 1 ) -> ( c.Aut(3) ) +
( st == 2 ) -> ( b.Aut(1) + b.Aut(3) + a.Aut(4) ) +
( st == 3 ) -> ( d . Aut(5) ) +
( st == 4 ) -> ( b . Aut(3) ) +
( st == 5 ) -> ( e . Aut(4) ) ;

init Aut(0);
Model checking with mCRL2: workflow

- **mcrl22lps** mini-lts.mcrl2 foo.lps -v
  create linear process specification from mcrl2 model code

- **lps2pbes** -f formula.mcf foo.lps foo.pbes -v
  transform formula and lps into
  parametrized boolean equation system

- **pbes2bool** foo.pbes -v
  generate boolean equation system and solve it
Model checking the coffee machine with mCRL2

Modelling and Analysis of Variability in Software Product Lines

ISTI–CNR, Pisa

June 26–30, 2017

These slides are largely based on slides by Erik de Vink
Initially, money must be inserted: either at least one euro’s worth in coins, exclusively for European products, or at least one dollar’s worth in coins, exclusively for Canadian products.

Input of money can be cancelled via a cancel button; optionally, the machine returns change if more than one euro or one dollar was inserted.

Once the machine contains at least one euro or one dollar, the user chooses whether (s)he wants sugar, by pressing one of two buttons, after which (s)he can select a beverage.

The choice of beverage (coffee, tea, cappuccino) varies, but coffee must be offered by all products, whereas cappuccino may be offered solely by European products.

Optionally, a ringtone may be rung after delivering a beverage; a ringtone must be rung by all products offering cappuccino.

After the beverage is taken, the machine returns idle.
The coffee machine: attributed feature model

total cost \leq 30
FTS feature labels omitted for readability
sort
  Feature = struct M | S | O | R | B | X | D | E | P | T | C ;
  FSet = List( Feature ) ;
  Coin = struct dime | quarter | half | dollar |
        ct10 | ct20 | ct50 | euro ;
  Currency = struct Dollar | Euro ;

act
  insert, return : Coin ;
  continue, cancel, sorry, no_change,
  sugar, no_sugar, coffee, tea, cappuccino,
  pour_sugar, pour_milk, pour_coffee, pour_tea,
  ring, skip, take_cup ;
  setS, setO, setR, setB, setX,
  setD, setE, setP, setT, setTP, setC ;
  ctc_tree_ok, attr_ok ;
  dollar_cappo_fault, ring_cappo_fault : FSet ;
  attr_fault : FSet # Int ;
  sel_set, wrong_set, set_ok : FSet ;
  cost : Int ;
map isSorted: FSet -> Bool;
noDuplicates: FSet -> Bool;
isSet: FSet -> Bool;

var ft, ft': Feature;
fset: FSet;

eqn isSorted([]) = true;
isSorted([ft]) = true;
isSorted(ft |> (ft' |> fset)) =
ft <= ft' && isSorted(ft’ |> fset);
noDuplicates([]) = true;
noDuplicates(ft |> fset) =
!(ft in fset) && noDuplicates(fset);
isSet(fset) = isSorted(fset) && noDuplicates(fset);
map insert: Feature # FSet -> FSet;
var ft, ft’: Feature;
  fset: FSet;
eqn insert(ft, []) = [ft];
  (ft < ft’) -> insert(ft, ft’ |> fset) = ft |> ft’ |> fset;
  (ft == ft’) -> insert(ft, ft’ |> fset) = ft’ |> fset;
  (ft > ft’) -> insert(ft, ft’ |> fset) = ft’ |> insert(ft, fset);

map union: FSet # FSet -> FSet;
var ft, ft’: Feature;
  fset, fset’: FSet;
eqn union([], fset) = fset;
  union(fset, []) = fset;
  (ft < ft’) -> union(ft |> fset, ft’ |> fset’) =
               ft |> union(fset, ft’ |> fset’);
  (ft == ft’) -> union(ft |> fset, ft’ |> fset’) =
               ft’ |> union(fset, fset’);
  (ft > ft’) -> union(ft |> fset, ft’ |> fset’) =
               ft’ |> union(ft |> fset, fset’);
map fcost : Feature -> Int ;
eqn fcost(M) = 0 ;
fcost(S) = 5 ;
fcost(O) = 0 ;
fcost(B) = 0 ;
fcost(R) = 5 ;
fcost(D) = 5 ;
fcost(E) = 5 ;
fcost(X) = 10 ;
fcost(C) = 5 ;
fcost(T) = 3 ;
fcost(P) = 7 ;

map tcost : FSet -> Int ;
var ft : Feature ;
fset : FSet ;
eqn tcost([]) = 0 ;
tcost(ft |> fset) = fcost(ft) + tcost(fset) ;
mCRL2 process Sel

initial call  Sel(0,[M])

proc Sel(st:Int,fs:FSet) =
  ...
  ( st == 1 ) -> (  
    ( M in fs ) -> (  
      setO . Sel(2,insert(0,fs) )  
    )  ) +  
  ( st == 2 ) -> (  
    ( M in fs ) -> (  
      tau . Sel(3,fs) +  
      setR . Sel(3,insert(R,fs) )  
    )  ) + ...
proc Sel(st: Int, fs: FSet) =
  %% feature states
  ( st == 0 ) -> ( ( M in fs ) -> ( setS . Sel(1, insert(S,fs) ) ) ) +
  ( st == 1 ) -> ( ( M in fs ) -> ( setO . Sel(2, insert(O,fs) ) ) ) +
  ( st == 2 ) -> ( ( M in fs ) -> ( tau . Sel(3,fs) + setR . Sel(3, insert(R,fs) ) ) ) +
  ( st == 3 ) -> ( ( M in fs ) -> ( setB . Sel(4, insert(B,fs) ) ) ) +
  ( st == 4 ) -> ( ( M in fs ) -> ( tau . Sel(5,fs) + setX . Sel(5, insert(X,fs) ) ) ) +
  ( st == 5 ) -> ( ( 0 in fs ) -> ( setD . Sel(6, insert(D,fs) ) + setE . Sel(6, insert(E,fs) ) ) ) +
  ( st == 6 ) -> ( ( B in fs ) -> ( tau . Sel(7,fs) + setT . Sel(7, insert(T,fs) ) +
                      setP . Sel(7, insert(P,fs) ) + setTP . Sel(7, union([T,P],fs) ) ) ) +
  ( st == 7 ) -> ( ( B in fs ) -> ( setC . Sel(8, insert(C,fs) ) ) ) +
  ...

Coffee machine
... 

( st == 8 ) -> ( 
  ( ( D in fs ) && ( P in fs ) ) ->
  wrong_set . delta <>
( !( R in fs ) && ( P in fs ) ) ) ->
  wrong_set . delta <>
  ctc_ok . Sel(9,fs)
)
+

( st == 9 ) -> ( 
  ( tcost(fs) <= 30 ) ->
  set_ok(fs) . cost( tcost(fs) ) . Prod(0,fs) <>
  wrong_set . delta );
proc Sel(st:Int,fs:FSet) =
  ...

%% cross-tree constraints
  ( st == 8 ) -> (  
    ( ( D in fs ) && ( P in fs ) ) -> 
      dollar_cappo_fault( fs ). Prod(99,fs) <> skip . Sel(81,fs)  
    ) + 
  ( st == 81 ) -> (  
    ( !( R in fs ) && ( P in fs ) ) -> 
      ring_cappo_fault( fs ). Prod(99,fs) <> ctc_tree_ok . Sel(9,fs)  
    ) + 

%% attribute constraints
  ( st == 9 ) -> (  
    ( tcost(fs) <= 30 ) -> 
      attr_ok . set_ok( fs ). cost( tcost(fs) ) . Prod(fs) <> attr_fault( fs , tcost(fs) ) . Prod(fs) );
proc Prod(st:Int,fs:FSet) =
  ( st == 0 ) -> ( Insert(0,fs) ) + ...
...
  ( st == 2 ) -> ( 
    ( C in fs ) -> coffee . Prod(4,fs) +
    ( T in fs ) -> tea . Prod(5,fs) +
  )
proc Prod(st:Int,fs:FSet) =
  ( st == 0 ) -> (  
    Insert(0,fs) ) +
  ( st == 1 ) -> (  
    ( S in fs ) -> ( sugar . Prod(2,fs) ) +
    ( S in fs ) -> ( no_sugar . Prod(3,fs) ) ) +
  ( st == 2 ) -> (  
    ( C in fs ) -> coffee . Prod(4,fs) +
    ( T in fs ) -> tea . Prod(5,fs) +
    ( P in fs ) -> cappuccino . Prod(6,fs) ) +
  ( st == 3 ) -> (  
    ( C in fs ) -> coffee . Prod(9,fs) +
    ( T in fs ) -> tea . Prod(8,fs) +
    ( P in fs ) -> cappuccino . Prod(7,fs) ) +
  ( st == 4 ) -> (  
    ( M in fs ) -> ( pour_sugar . Prod(9,fs) ) ) +
  ( st == 5 ) -> (  
    ( M in fs ) -> ( pour_sugar . Prod(8,fs) ) ) +
  ...

Coffee machine
proc Prod(st: Int, fs: FSet) =

... (st == 6) -> ( (M in fs) -> (pour_sugar . Prod(7, fs)) ) +
(st == 7) -> ( (M in fs) -> (pour_milk . Prod(10, fs)) +
(M in fs) -> (pour_coffee . Prod(11, fs)) ) +
(st == 8) -> ( (M in fs) -> (pour_tea . Prod(12, fs)) ) +
(st == 9) -> ( (M in fs) -> (pour_coffee . Prod(12, fs)) ) +
(st == 10) -> ( (M in fs) -> (pour_coffee . Prod(12, fs)) ) +
(st == 11) -> ( (M in fs) -> (pour_milk . Prod(12, fs)) ) +
(st == 12) -> ( (R in fs) -> (ring . Prod(13, fs)) +
(!(R in fs)) -> (skip . Prod(13, fs)) ) +
(st == 13) -> ( (M in fs) -> (take_cup . Prod(0, fs)) );
mCRL2 code: money handling

proc Insert(bal:Nat,fs:FSet) =
( bal < 100 ) -> (  
  ( D in fs ) -> (  
    insert(dime) . Insert(bal+10,fs) +  
    insert(quarter) . Insert(bal+25,fs) +  
    insert(half) . Insert(bal+50,fs) +  
    insert(dollar) . Insert(bal+100,fs) ) ) +  
( E in fs ) -> (  
    insert(ct10) . Insert(bal+10,fs) +  
    insert(ct20) . Insert(bal+20,fs) +  
    insert(ct50) . Insert(bal+50,fs) +  
    insert(euro) . Insert(bal+100,fs) ) ) +  
( ( bal > 0 ) && ( bal < 100 ) ) ->  
  Return(bal,fs) . cancel . Prod(0,fs) +  
( bal >= 100 ) -> (  
  ( ( !(X in fs) ) ->  
    no_change . continue . Prod(1,fs) <>  
  Return(Int2Nat(bal-100),fs) .  
    continue . Prod(1,fs) ) );
proc Return(bal:Nat,fs:FeatSet) =
  ( bal == 0 ) -> tau +
  ( D in fs ) -> ( 
    ( bal >= 50 ) ->
      return(half) . Return(Int2Nat(bal-50),fs) +
    ( ( bal < 50 ) && ( bal >= 25 ) ) ->
      return(quarter) . Return(Int2Nat(bal-25),fs) +
    ( ( bal < 25 ) && ( bal >= 10 ) ) ->
      return(dime) . Return(Int2Nat(bal-10),fs) +
    ( ( bal < 10 ) && ( bal > 0 ) ) ->
      sorry . Return(0,fs) ) +
  ( E in fs ) -> ( 
    ( bal >= 50 ) ->
      return(ct50) . Return(Int2Nat(bal-50),fs) +
    ( ( bal < 50 ) && ( bal >= 20 ) ) ->
      return(ct20) . Return(Int2Nat(bal-20),fs) +
    ( ( bal < 20 ) && ( bal > 0 ) ) ->
      return(ct10) . Return(Int2Nat(bal-10),fs) +
    ( ( bal < 10 ) && ( bal > 0 ) ) ->
      sorry . Return(0,fs)
  );
Visual inspection of the state space

Coffee machine
Visual inspection of the state space
Example properties

- \[(!\text{continue}).\text{take\_cup}] \text{false}\]
  if payment is not settled, no beverage is delivered
- \[[\text{true}.\text{setX}.\text{true}.\text{no\_change}] \text{false}\]
  once feature X is selected, no_change will not occur
- \[[\text{true}] \forall \text{fs:}\text{FSet.}\]
  \[<\text{set\_ok}(\text{fs})> \text{(val}(\text{D in fs}) \Rightarrow !\text{(P in fs)}))]\]
  if a product is configured successfully, then
  it cannot accept dollars and also provide cappuccino
- \[\mu Y. (<\exists \text{fs:}\text{FSet.}\text{set\_ok}(\text{fs})> \text{true} ||
  <\exists \text{fs:}\text{FSet.}\text{wrong\_set}(\text{fs})> \text{true} ||
  [\text{true}] Y)\]
  eventually either set\_ok or wrong\_set can occur
- \[\forall c: \text{Coin.}[\text{true}.\text{insert}(c)]\]
  \[\mu Y. (<\text{cancel} || \text{take\_cup}> \text{true} || [\text{true}] Y)\]
  after money is inserted, eventually a beverage is given
  or money insertion is canceled
- \[[\text{true}] \forall \text{n:}\text{Nat.}[\text{cost}(\text{n})](\text{val}(\text{n} \leq 30))\]
  all valid products have a total cost at most 30
Next: family-based model checking with a feature $\mu$-calculus

Modelling and Analysis of Variability in Software Product Lines

ISTI–CNR, Pisa

June 26–30, 2017

These slides are largely based on slides by Erik de Vink