Feature-based modular verification of software product lines

Maurice ter Beek

ISTI–CNR, Pisa

3rd General Meeting CINA, Torino
February 12, 2015

joint work with Erik de Vink and Tessa Belder
(TU Eindhoven & CWI Amsterdam, The Netherlands)
develop a product line (a.k.a. a family) using a shared architecture or platform (commonalities) and mass customization (variabilities) to serve, e.g., different markets, thus allowing for (software) reuse

aim: maximize commonalities whilst minimizing cost of variations (i.e. of individual products)

variability in terms of features:

- end-user visible pieces of functionality that represent both commonalities (e.g. mandatory, required) and variabilities (e.g. optional, alternative)
- only specific feature combinations concern valid products

“We always have 126,000,000 different bicycles in store! But only the parts for 1,000…”
Configure your 11-inch MacBook Air

**Processor**
Enjoy incredible performance from fourth-generation Intel Core processors. Choose the speed and processor you want.

- 1.3GHz Dual-Core Intel Core i5, Turbo Boost up to 2.6GHz
- 1.7GHz Dual-Core Intel Core i7, Turbo Boost up to 3.3GHz [+ £130.00]

**Memory**
More memory (RAM) increases overall performance and enables your computer to run more applications at the same time.

- 4GB 1600MHz LPDDR3 SDRAM
- 8GB 1600MHz LPDDR3 SDRAM [+ £80.00]

**Storage**
Your MacBook Air comes as standard with flash storage. Flash storage has no moving parts and provides faster responsiveness and enhanced durability.

- 256GB Flash Storage
- 512GB Flash Storage [+ £240.00]
Configure your BMW vehicle

Are you interested in configuring your ideal BMW? Please select a country to visit the configurator in the Virtual Center or contact your local BMW dealer who will be happy to answer all your questions about the BMW model you are interested in.

FIND YOUR BMW.

Filter
- Reset filter
- Budget
- Vehicle type
  - All
  - Petrol
  - Diesel
  - Hybrid
  - Electric Vehicle
- Body type
  - Saloon
  - Touring
  - Convertible
  - Coupé
  - Gran Turismo
  - Sports Hatch
  - Roadster
  - Sports Activity Coupé
  - Sports Activity Vehicle
- Number of seats

30 Vehicles (465 Model variants)

1. BMW 1 Series 3-door Sports Hatch (34)
   from £ 17,775.00

2. BMW 2 Series Coupé (14)
   from £ 24,265.00

3. BMW 3 Series Saloon (56)
   from £ 23,550.00

Related topics
- Request information
- Order product catalogues, brochures and equipment lists direct from BMW.
Attributed feature model: representation of all products

- Machine \( M \)
  - Sweet \( S \): Euro 5
  - Coin \( O \): Dollar 5
  - Ringtone \( R \): Cappuccino 7
  - Beverage \( B \): Coffee 5
  - Change \( X \): Tea 3

Total cost \( \leq 30 \)

\[
\text{cost (product)} = \sum \{ \text{cost (feature)} \mid \text{feature} \in \text{product} \}
\]

(from \( 2^{10} - 1 \) feature diagram \( \rightarrow \) \( 2^5 \) cross-tree constraints \( \rightarrow \) 20 attributes \( \rightarrow \) 16 valid products)
family of 16 valid products (i.e. feature combinations)

‘feature model’
(allowing >16)
Formal modelling and verification in SPLE

computer-aided analysis of variability models

- traditionally focus on modeling/analyzing structural constraints
- but software systems often embedded/distributed/safety-critical
- important to model/verify also behavior (e.g. quality assurance)

goal: rigorously establish critical requirements of (software) systems
lift success stories from single product/system engineering to SPLE

recent approaches to formally model behavioral variability:

- variants of UML diagrams (Haugen et al., Jézéquel et al.)
- extensions of Petri nets (Clarke et al.)
- variety of models with LTS-like semantics: MTS (Fischbein et al., Fantechi et al.), I/O automata (Larsen et al., Lauenroth et al.), CCS/CSP/CCP/SCCP (Gruler et al., Gnesi et al., ter Beek et al., Tribastone), FTS (Classen et al.), FSM (Millo et al.), etc...
Scalability is a major issue! (slide by C. Kästner, CMU)

with 33 optional, independent features

person on this planet
Recommendations for the Usability of Formal Methods for Product Lines (Atlee et alii, FormaliSE’13)

- adopt and extend state-of-the-art analysis tools
- examine[s] only valid product variants
- visualize and (manually or automatically) analyze feature combinations corresponding to products of the product line
- support (feature) modularity

Fisler & Krishnamurthi (ESEC/FSE’01) first recognized that most properties of interest naturally decompose around features
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, user-defined abstract datatypes, parametrized actions
- modal $\mu$-calculus with data
- visualization, behavioral reduction, model checking
- highly optimized, actively maintained
- intermediate artifacts user-accessable

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

- \([\text{true*}]<\text{true}>\text{true}\): absence of deadlock
- \([\text{true*}.b.\text{true*}.a]\) false: no a after a b
- \(\mu Y.(<\text{true}>\text{true} \&\& [!c] Y)\): eventually c can be done (or deadlock occurs earlier)
- \(\mu Y.( (\nu Z.(<b.d.e> Z)) || [\text{true}] Y)\): eventually, an infinite sequence of bde is possible
A product line or family of coffee machines

- 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 canceled 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 $\leq 30$
mCRL2 process Sel

initial call  \text{Sel}(0,[M])

proc \text{Sel}(st:\text{Int},fs:\text{FSet}) =
  \ldots
  (\text{st} == 1) \rightarrow (\text{M in fs}) \rightarrow (\text{setO . Sel}(2,\text{insert}(0,fs)))

) +

(\text{st} == 2) \rightarrow (\text{M in fs}) \rightarrow (\tau . \text{Sel}(3,fs) + \text{setR . Sel}(3,\text{insert}(R,fs)))

) + \ldots
mCRL2 process Sel (continued)

\[
\begin{align*}
\ldots
\quad ( \text{st} == 8 ) & \rightarrow ( \\
\quad ( \text{D in fs} ) \&\& ( \text{P in fs} ) ) & \rightarrow \\
\quad \text{wrong_set . delta <> } \\
\quad ( !( \text{R in fs} ) \&\& ( \text{P in fs} ) ) & \rightarrow \\
\quad \text{wrong_set . delta <> } \\
\quad \text{ctc_ok . Sel(9,fs) } \\
\end{align*}
\]

\[
\begin{align*}
\ldots
\quad + \\
\quad ( \text{st} == 9 ) & \rightarrow ( \\
\quad ( \text{tcost(fs}) \leq 30 ) & \rightarrow \\
\quad \text{set_ok(fs) . cost( tcost(fs) ) . Prod(0,fs) <> } \\
\quad \text{wrong_set . delta }); \\
\end{align*}
\]
The coffee machine: family behaviour

FTS feature labels omitted for readability
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) + 
      ( P in fs ) -> cappuccino . Prod(6, fs) 
   ) + ...
mCRL2 process Insert

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) ) );
Example properties

- [(!continue)*.take_cup] false
  if payment is not settled, no beverage is delivered

- [true*.setX.true*.no_change] false
  once feature \( X \) is selected, \( \text{no_change} \) will not occur

- [true*] forall \( fs: \text{FSet} \).
  
  <set_ok(fs)> (val((\( D \) in \( fs \)) => !(\( P \) in \( fs \))))
  if a product is configured successfully, then
  it cannot accept dollars and also provide cappuccino

- \( \mu Y. ( < \text{exists} \text{ fs: FSet.set_ok(fs)> true} \)
  || <\text{wrong_set}> true || [true] Y) \)
  eventually either \( \text{set_ok} \) or \( \text{wrong_set} \) can occur

- forall \( c: \text{Coin} \). [true*.insert(c)]
  \( \mu Y. ( <\text{cancel}|\text{take_cup}> \text{true} \ || [true] Y) \)
  after money is inserted, eventually a beverage is given
  or money insertion is canceled
Decomposition of product behaviour

five main components

- Machine (m): take_cup
- Sweet (s): sugar, no_sugar, pour_sugar
- Coin/Change (o,d,e,x): insert, cancel, no_change, sorry, continue
- Ringtone (r): ring, skip
- Beverage (b,c,t,p): coffee, tea, cappuccino, pour_coffee, pour_tea, pour_milk
Decomposition of product behaviour (continued)
Isolated Beverage component
proc Driver(st:Int) =
    drv_start(st) . sum st':Int . catch(st’) . Driver(st’);

proc Beverage(fs:FSet) =
    cmp_start(2) . ( ( C in fs ) -> coffee . raise(4) . Beverage() + 
    ( T in fs ) -> tea . raise(5) . Beverage() + 
    ( P in fs ) -> cappuccino . raise(6) . Beverage() ) + 
    ...
    cmp_start(10) .
    pour_coffee . raise(12) . Beverage() +
    cmp_start(11) .
    pour_milk . raise(12) . Beverage();
Driver, component(s) and stub process

transforming

\[
\text{Driver}(0) \ || \ \text{Sweet}(fs) \ || \ \text{Coin}(fs) \ || \\
\ \ \ \text{Ringtone}(fs) \ || \ \text{Beverage}(fs) \ || \ \text{Machine}
\]

into

\[
\text{Driver}(0) \ || \ \text{Beverage}(fs) \ || \ \text{Stub}
\]

proc Stub =

cmp_start(0) . other . ( raise(2) + raise(3) ) . Stub +
cmp_start(4) . other . raise(9) . Stub +
cmp_start(5) . other . raise(8) . Stub +
cmp_start(6) . other . raise(7) . Stub +
cmp_start(12) . other . ( raise(2) + raise(3) ) . Stub;

Isolated Beverage component (revisited)

(re)entry and exit points
Model checking with stubs

- example property (modal μ-formula in CTL* – next operator)
  \[
  \text{[ true*.coffee]} \text{( mu X. [ !pour\_coffee ] X )}
  \]
  after a coffee action, a pour\_coffee action happens within a finite number of steps

- model check property against reduced model
  \[
  \text{Driver(0) || Beverage(fs) || Stub}
  \]

- verify branching bisimilarity of reduced model
  \[
  \text{Driver(0) || Beverage(fs) || Stub}
  \]
  and full model (with ltscompare tool in the mCRL2 toolset)
  \[
  \text{Driver(0) || Sweet(fs) || Coin(fs) || Ringtone(fs) || Beverage(fs) || Machine}
  \]

previous example property

\[ [ \text{true}.\text{coffee} ] ( \mu X. [ !\text{pour\_coffee} ] X ) \]

new example property

\[ [\text{true}.\text{coffee}]( \mu X. [ !(\text{pour\_coffee} || \text{return}) ] X ) \]

action \textit{return} does not belong to component \textit{Beverage}

will check adapted property

\[ [\text{true}.\text{coffee}]( \mu X. [ !(\text{pour\_coffee} || \text{escape}) ] X ) \]

action \textit{escape} represents pre-emption of component
proc Stub2 =
    cmp_start(0) . other . (raise(2) + raise(3)) . Stub2 +
    cmp_start(4) . (escape . (raise(2) + raise(3)) . Stub2 +
        other . raise(9) . Stub2 ) +
    cmp_start(5) . ( ... ) +
    cmp_start(6) . ( ... ) +
    cmp_start(12) . other . (raise(2) + raise(3)) . Stub2 ;
verification of adapted property \texttt{adapted\_prop} \\
\[ \text{true}^*\text{.coffee}(\mu X. [ \neg (\text{pour\_coffee} \lor \text{escape}) ] X ) \]
against new reduced model \texttt{RedSys} \\
\texttt{Driver(0) || Beverage(fs) || Stub2}
conditional to equivalence of satisfaction \\
\texttt{FullSys} \models \texttt{new\_prop} \quad \text{vs.} \quad \texttt{RedSys} \models \texttt{adapted\_prop}
$\mathcal{F}$: finite non-empty set of features, $\mathcal{P} \subseteq 2^{\mathcal{F}}$ subset of products

$\mathbb{B}(\mathcal{F})$: set of boolean expressions over $\mathcal{F}$ (feature expressions)

$(S, \theta, s_0)$: featured transition system with set $S$ of states, transition constraint function $\theta : S \times A \cup \{\tau\} \times S \to \mathbb{B}(\mathcal{F})$, initial state $s_0 \in S$

we write $s \xrightarrow{\alpha|\psi} s'$ if $\theta(s, \alpha, s') = \psi$

product $P \in \mathcal{P}$ satisfies feature expression $\varphi \in \mathbb{B}(\mathcal{F})$, denoted $P \models \varphi$, if $\varphi$ is valid when the Boolean variables corresponding to the features of $P$ are assigned value true and those not in $P$ value false

equivalence relation $\sim_P$ on $\mathbb{B}(\mathcal{F})$ given by $\varphi \sim_P \psi$ iff $\forall P \in \mathcal{P}$: $P \models \varphi \iff P \models \psi$; we denote $\hat{\mathbb{B}}(\mathcal{F}) = \mathbb{B}(\mathcal{F})/\sim_P$
Branching feature bisimulation $R \subseteq S \times \hat{\mathcal{B}}(\mathcal{F}) \times S$

for $s, s' \in S$ and satisfiable $\eta \in \mathcal{B}(\mathcal{F})$ we write $s \xrightarrow{\eta} s'$ if

$\exists n \exists s_0, \ldots, s_n \exists \eta_1, \ldots, \eta_n :$

$s = s_0 \land \forall i, 1 \leq i \leq n : s_{i-1} \xrightarrow{\tau|\eta_i} s_i \land s' = s_n \land \eta = \bigwedge_{1 \leq i \leq n} \eta_i$

we write $s \xrightarrow{(\alpha|\psi)} s'$ in case $s \xrightarrow{\alpha|\psi} s'$ or $\alpha = \tau \land s = s' \land \psi = true$
FTS vs. LTS bisimulation

bisimilar FTS assuming $\varphi_1 \land \varphi_2 \Rightarrow (\psi_1 \iff \psi_2)$

non-bisimilar LTS assuming $\varphi_1 \land \varphi_2 \land \psi_1 \land \neg \psi_2$

**Theorem**

*Let $s, t$ be states of FTS $\mathcal{S}$. Then $s \simeq_{bf} t$ iff $s \simeq_P t$ for all $P \in \mathcal{P}$.***
$\mathcal{T}$ and $\mathcal{U}$ are both branching feature bisimilar to $S$ and both have the minimal number of states, but $\mathcal{U}$ has twice as many transitions as $\mathcal{T}$.

**Theorem**

*Constructing a minimal branching feature bisimilar FTS is NP-complete.*

**Theorem**

*Our polynomial-time sub-optimal algorithm reduces an FTS $S$ to the smallest FTS $S_{\text{min}}$ for which there exists a coherent branching feature bisimulation relation for $S$ and $S_{\text{min}}$.*

Adaptation of efficient Groote & Vaandrager algorithm (ICALP’90)
Experimental evaluation (beverage + soup component)

(for a total of 182 states and 691 transitions in parallel composition)
mCRL2 results of experimental evaluation (time in seconds)

<table>
<thead>
<tr>
<th>PROPERTIES</th>
<th>PRODUCT-BY-PRODUCT</th>
<th></th>
<th>FTS-BASED FAMILY APPROACH</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WITHOUT BISIMULATION</td>
<td>WITH BISIMULATION</td>
<td>WITHOUT BISIMULATION</td>
<td>WITH BISIMULATION</td>
</tr>
<tr>
<td>TIME (s)</td>
<td>RESULT</td>
<td>TIME (s)</td>
<td>RESULT</td>
<td>TIME (s)</td>
</tr>
<tr>
<td>1</td>
<td>42.04</td>
<td>false</td>
<td>38.18</td>
<td>true</td>
</tr>
<tr>
<td>2</td>
<td>41.78</td>
<td>true</td>
<td>41.65</td>
<td>true</td>
</tr>
<tr>
<td>3a</td>
<td>42.32</td>
<td>true</td>
<td>37.76</td>
<td>true</td>
</tr>
<tr>
<td>3b</td>
<td>42.01</td>
<td>true</td>
<td>37.78</td>
<td>true</td>
</tr>
<tr>
<td>4a</td>
<td>40.62</td>
<td>true</td>
<td>38.00</td>
<td>true</td>
</tr>
<tr>
<td>4b</td>
<td>40.20</td>
<td>true</td>
<td>37.88</td>
<td>true</td>
</tr>
<tr>
<td>5a</td>
<td>42.38</td>
<td>false</td>
<td>38.51</td>
<td>true</td>
</tr>
<tr>
<td>5b</td>
<td>42.34</td>
<td>false</td>
<td>38.09</td>
<td>true</td>
</tr>
<tr>
<td>6</td>
<td>43.63</td>
<td>true</td>
<td>39.17</td>
<td>true</td>
</tr>
<tr>
<td>7a</td>
<td>42.45</td>
<td>true</td>
<td>38.19</td>
<td>true</td>
</tr>
<tr>
<td>7b</td>
<td>42.35</td>
<td>true</td>
<td>38.04</td>
<td>true</td>
</tr>
<tr>
<td>8</td>
<td>42.82</td>
<td>true</td>
<td>39.09</td>
<td>true</td>
</tr>
</tbody>
</table>

| TOT        | 504.94             | 462.34           | 754.50                   | 209.32           |

1. if a coffee is ordered, then eventually coffee is poured:

\[ \text{true}* . \text{coffee} \] ( \mu X . [ ! \text{pour\_coffee} ] X )

2. the SPL is deadlock-free: \[ \text{true}* ] < \text{true} > \text{true} etc.
Conclusions and future work

- mCRL2 for software product line analysis
- model reduction before model checking
- automate property and stub adaptation?
- establish subset of modal $\mu$-calculus preserved by branching feature bisimulation
- reduce complexity of minimization algorithm
- does modular verification approach scale to realistic SPL?