Compositional Verification of Software Product Families

Ina Schaefer\(^1\)  Dilian Gurov\(^2\)  Siavash Soleimanifard\(^2\)

\(^1\) Technische Universität Braunschweig, Germany
\(^2\) Kungliga Tekniska Högskolan, Stockholm, Sweden

Deduction at Scale 2011

Schloß Ringberg, 7 March 2011
Motivation

Product Family

Set of products with well-defined commonalities and variabilities
Hierarchical Variability Modelling for Product Families

Product Family

@VP1

Core

Variation Point

@VP2

Variant

V11

V12

V21

V22
Analysis of Product Families

Non-Compositional Analysis

Verification tasks bound by $(\#\text{variants})^{(\#\text{VP})^{ND}}$

Compositional Analysis

Verification tasks bound by $(\#\text{variants} \times \#\text{VP})^{ND}$
Compositional Analysis of Product Families

- Relativize Product Properties towards Variation Points
- Apply Compositional Analysis Technique
Compositional Verification of Control Flow Safety Properties

Hierarchical Variability Modelling

Modular Specification of Core and Variation Point Properties

Compositional Reasoning using Variation Point Properties
Compositional Verification Technique by D. Gurov and M. Huisman

Program Model

- flow graphs (no data)
- method call edges, return nodes
- infinite–state behaviour

Logic

- temporal logic for safety properties
- legal sets of sequences of method invocations

---

1 Dilian Gurov, Marieke Huisman, and Christoph Sprenger: "Compositional Verification of Sequential Programs with Procedures", Journal of Information and Computation, 2008
Inductively defined as

(i) a **ground model** consisting of a **core** set of methods $M_C = (M_{\text{pub}}, M_{\text{priv}})$, partitioned into public and private methods.

(ii) a pair $(M_C, \{VP_1, \ldots, VP_N\})$, where $M_C$ is defined as above and where $\{VP_1, \ldots, VP_N\}$ is a non-empty set of **variation points**.

A **variation point** $VP_i$ is a non-empty set of SHVMs, $VP_i = \{S_{i,j} \mid 1 \leq j \leq k_i\}$. The members of a variation point are called **variants**.
Example: Cash Desk Product Family

- CashDesk
  - @EnterProducts
    - enterProd()
    - useKeyboard()
  - @Payment
    - payment()
    - cardPay()
- Keyboard
  - enterProd()
  - useKeyboard()
- Scanner
  - enterProd()
  - useScanner()
- Cash
  - payment()
  - cashPay()
- Card
  - payment()
  - enterCard()
  - cardPay()
Why **Simple** Hierarchical Variability Model?

- At each variation point, select exactly one variant.
- No dependencies between variants and variation points.
- Same interface for all variants at a variation point. (same set of public provided methods)
We have to provide

▶ a global product property at the top-most SHVM node.
▶ local specifications for every core method.
▶ variation point specifications for every variation point.
▶ each variant inherits the property of its variation point.

Specification Language sLTL

The formulae of sLTL are inductively defined by:

\[
\phi ::= p \mid \neg p \mid \phi_1 \land \phi_2 \mid \phi_1 \lor \phi_2 \mid X \phi \mid G \phi \mid \phi_1 W \phi_2
\]
Global Product Property of Cash Desk

Entering of products must be completed before payment:

\[ \text{sale} \rightarrow (\neg \text{payment} \mathrel{W} (r \land \text{enterProd} \land X \text{ sale})) \]
Local Specification of sale()
sale() only calls payment() after returning from enterProd():

\[ \text{sale} \; \vec{W} \; \text{enterProd} \; \vec{W} \; \text{sale} \; \vec{W} \; \text{payment} \; \vec{W} \; (G \; \text{sale}) \]

where \( \phi \; \vec{W} \; \psi \) abbreviates \( \phi \; \land \; (\phi \; \vec{W} \; \psi) \).
Specification of Example (3)

VP Specification of @EnterProducts
enterProd() never calls payment(): G (¬payment)

VP Specification of @Payment
payment() never calls enterProd(): G (¬enterProd)
For every SHVM \((M_C, \{VP_1, \ldots, VP_N\})\) :

- For each core method \(m \in M_C\), verify local specification.
- For every module, verify SHVM specification under the assumption of core method specifications and variation point specifications.
For every SHVM \((M_C, \{VP_1, \ldots, VP_N\})\) and for every public method \(m \in M_{pub}:\)

- extract the method graph \(G_m\) from the implementation of \(m\)
- inline the already extracted graphs for the private methods
- model check the resulting method graph against the specification \(\psi_m\) of \(m\) to establish \(G_m \models \psi_m\) by standard finite-state model checking
Compositional Verification of SHVM

For every SHVM \((M_C, \{ VP_1, \ldots, VP_N \})\):

- for all public methods \(m \in M_{pub}\) with specification \(\psi_m\), construct the maximal method graphs \(\text{Max}(\psi_m, I_m)\) wrt. interface \(I_m\)

- for all variation points \(VP_i\) with specification \(\psi_{VP_i}\) construct the maximal flow graphs \(\text{Max}(\psi_{VP_i}, I_{VP_i})\) wrt. interface \(I_{VP_i}\)

- compose the graphs, resulting in flow graph \(G_{\text{Max}}\), and model check the latter against the SHVM property \(\phi\).

\[
\left( \biguplus_{m \in M_{pub}} \text{Max}(\psi_m, I_m) \uplus \biguplus_{VP_i \in \{VP_1, \ldots, VP_N\}} \text{Max}(\psi_{VP_i}, I_{VP_i}) \right) \models \phi
\]
Theorem

Let $S$ be an SHVM with global property $\phi$. If the verification procedure succeeds for $S$, then $p \models \phi$ for all its products $p \in \text{products}(S)$.

Proof.

The proof is by induction on the nesting depth of $S$. 
Tool Support: ProMoVer for ProductFamilies

ProMoVer is partially adapted to verify product families.

Use this page to verify product families. It partially works. Partially because there are still some logic transactors to be done. Also a modular control flow graph extractor should be also added to the tool. But for the moment we have it available for those who can stick these problems away.

Usage

Simply paste your annotated Java class in the Program text area. The annotated Java class has to be compatible with JDK1.6.0, i.e., it should not need any special class path or package. Then write the class name in the textbox Class Name and push the Verify button.

The verification result is shown after some seconds. Notice that the result may take some minutes.
Tool Support: ProMoVer for ProductFamilies

Program:
/**
 * @variant: CashDesk
 * @variant_interface: required nothing
 *                provided sale, enterProd, payment
 * @variant_prop: (! payment U (enterProd && ret))
 * @variation_points: EnterProducts, Payment
 * */

public class CashDesk{
    /**
     * @core: CashDesk
     * @local_interface: required
     *                enterProd, payment, updateStock, writeReceipt
     * @local_prop: [...] 
     */
    public void sale()
    {
        int i = 0;
        while (i < 10)
        {
            enterProd();
            i++;
        }
        payment();
        updateStock();
        writeReceipt();
    }

    /**
     * @variation_point: EnterProducts CashDesk
     * @variation_point_interface: required nothing
     *                provided enterProd
     * @variation_point_prop: [...] 
     */
    private void variants()
    {
        @variant: Keyboard, Scanner
        @variant: Keyboard-EnterProducts
        @variant_interface: required nothing
                provided enterProd
        @variant_prop: true
    }

    Class Name: CashDesk
Variant Annotations:

```java
/**
 * @variant: CashDesk
 *
 * @variant_interface: required
 * provided sale, enterProd, payment
 *
 * @variant_prop:
 * sale --> (!payment \[or\] enterProd \&\& X sale)
 *
 * @variation_points: EnterProducts, Payment
 */

public class CashDesk{ ...
```
Core Annotations:

/**
 * @core: CashDesk
 *
 * @local_interface: required enterProd,payment
 *
 * @local_prop:
 * (sale W enterProd W sale W payment W (G sale))
 */

public void sale(){
    int i = 0;
    while (i < 10){
        enterProd();
        i++;
    }
    payment();
    updateStock();
    writeReceipt();
}
Variation Point Annotations:

/**
 * @variation_point: EnterProducts_CashDesk
 *
 * @variation_point_interface: required
 *                     provided enterProd
 *
 * @variation_point_prop: G !payment
 *
 * @variants: Keyboard, Scanner
 **/
PREPROCESSOR TIME IS: 1.52 seconds

FLOW GRAPH EXTRACTOR TIME IS: 3.12 seconds

the method sale.CashDesk matches its implementation
the method enterProd.Keyboard-EnterProducts matches its implementation
the method enterProd.Scanner-EnterProducts matches its implementation

[...]
We compositionally verified different product families:

- CD - Simple Cash Desks
- CD/CH - Cash Desks with Coupon Handling
- CD/CT - Cash Desks with Credit Cards
- CD/CT/CH - Cash Desks with Credit Cards and Coupon Handling

Analysis Results:

<table>
<thead>
<tr>
<th>Product Line</th>
<th>Depth</th>
<th># Modules</th>
<th># Products</th>
<th>$t_{ind}[s]$</th>
<th>$t_{comp}[s]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>101</td>
<td>26</td>
</tr>
<tr>
<td>CD/CH</td>
<td>1</td>
<td>7</td>
<td>8</td>
<td>206</td>
<td>28</td>
</tr>
<tr>
<td>CD/CT</td>
<td>2</td>
<td>9</td>
<td>11</td>
<td>281</td>
<td>29</td>
</tr>
<tr>
<td>CD/CH/CT</td>
<td>2</td>
<td>11</td>
<td>20</td>
<td>518</td>
<td>30</td>
</tr>
</tbody>
</table>
Conclusion

Summary
- Compositional analysis of product families defined by HVM
- Verification of control flow safety properties for SHVM

Future Work
- Relax restrictions of SHVM
- Improvements of ProMoVer tool
- Use approach with other compositional reasoning techniques