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

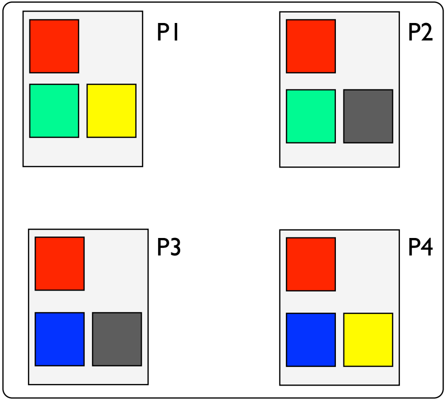
HATS
Highly Adaptable and Trustworthy Software using Formal Models

SEVENTH FRAMEWORK PROGRAMME

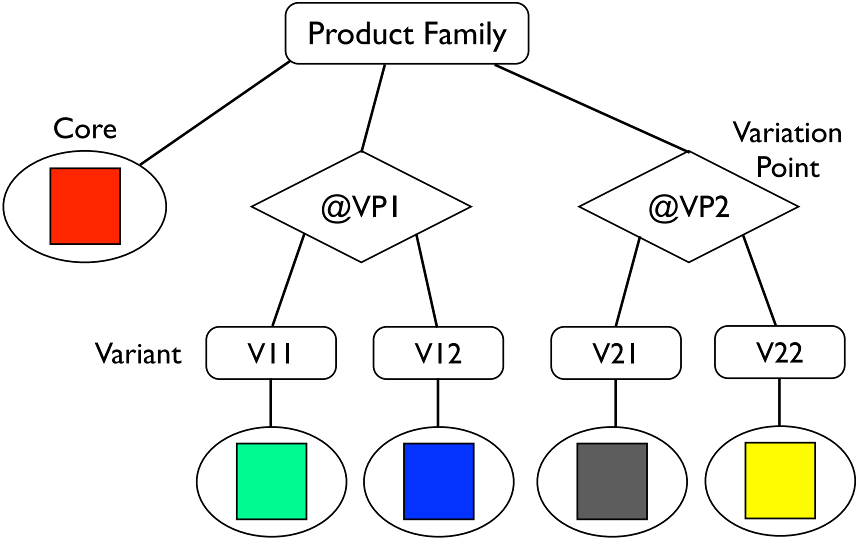
Motivation

Product Family

Set of products with well–defined commonalities and variabilities



Hierarchical Variability Modelling for Product Families



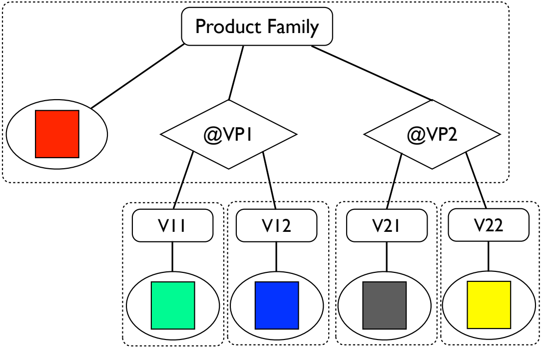
Analysis of Product Families

Non-Compositional Analysis

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

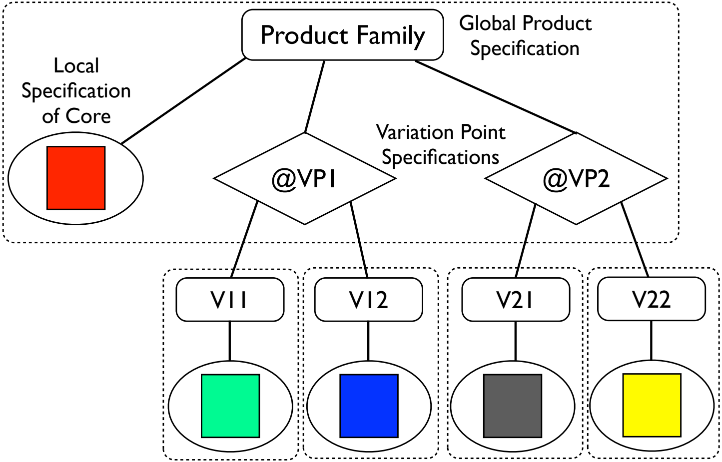
Compositional Analysis

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



Compositional Analysis of Product Families

- Relativize Product Properties towards Variation Points
- Apply Compositional Analysis Technique



Outline

- 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 of Control Flow Safety Prop.

Compositional Verification Technique by D. Gurov and M. Huisman[1]

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

Simple Hierarchical Variability Model

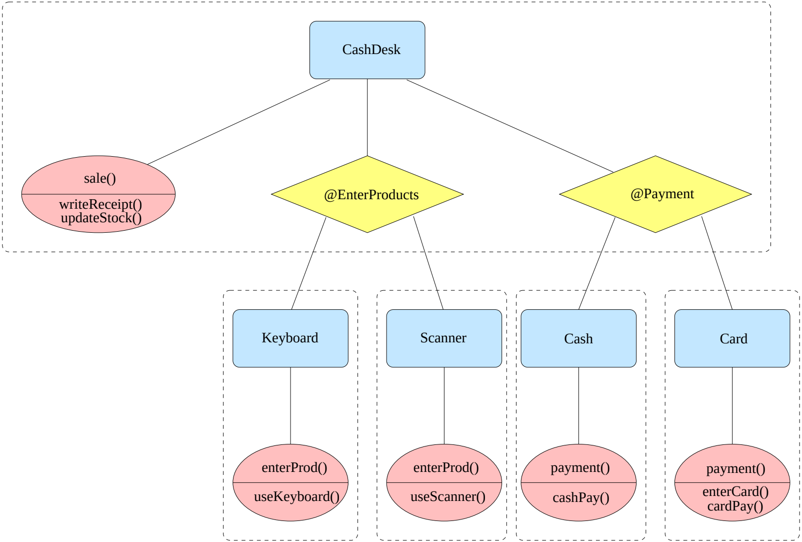
Inductively defined as

(i) a ground model consisting of a core set of methods $M_C = (M_{pub}, M_{priv})$, 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, $VP_i = \{\mathcal{S}_{i,j} \mid 1 \leq j \leq k_i\}$. The members of a variation point are called variants.

Example: Cash Desk Product Family



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)

Specification for Compositional Reasoning

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:

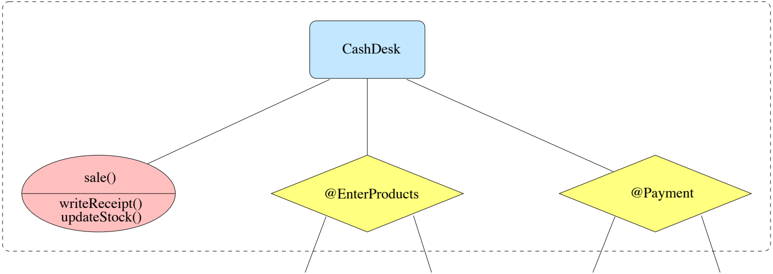
$$\phi ::= p \mid \neg p \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \mathtt{X}\,\phi \mid \mathtt{G}\,\phi \mid \phi_1\,\mathtt{W}\,\phi_2$$

Specification of Example

Global Product Property of `Cash Desk`

Entering of products must be completed before payment:

$$sale \longrightarrow (\neg payment \text{ W } (r \wedge enterProd \wedge \text{X } sale))$$



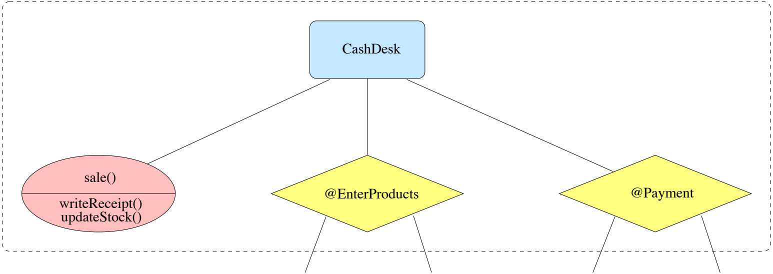
Specification of Example (2)

Local Specification of `sale()`

`sale()` only calls `payment()` after returning from `enterProd()`:

$$\mathit{sale}\ \mathtt{W}'\ \mathit{enterProd}\ \mathtt{W}'\ \mathit{sale}\ \mathtt{W}'\ \mathit{payment}\ \mathtt{W}'\ (\mathtt{G}\ \mathit{sale})$$

where $\phi\ \mathtt{W}'\ \psi$ abbreviates $\phi \wedge (\phi\ \mathtt{W}\ \psi)$.



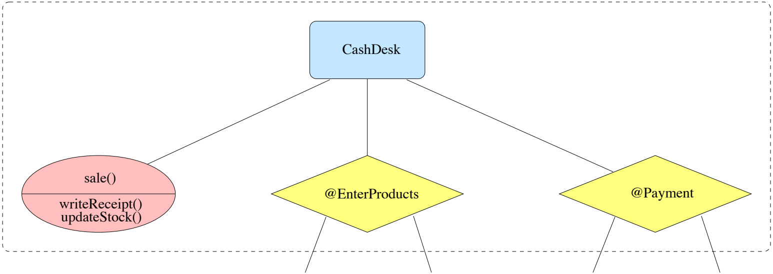
Specification of Example(3)

VP Specification of @EnterProducts

`enterProd()` never calls `payment()`: $\mathsf{G}\ (\neg payment)$

VP Specification of @Payment

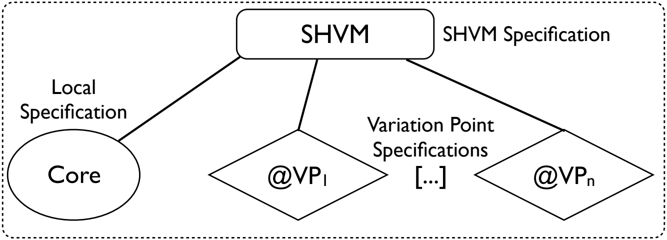
`payment()` never calls `enterProd()`: $\mathsf{G}\ (\neg enterProd)$



Compositional Verification Procedure

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.



Verification of Core Specifications

For every SHVM $(M_C, \{VP_1, \ldots, VP_N\})$ and for every public method $m \in M_{pub}$:

- extract the method graph $\mathcal{G}_m$ from the implementation of m
- inline the already extracted graphs for the private methods
- model check the resulting method graph against the specification ψ_m of m to establish $\mathcal{G}_m \models \psi_m$ 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 ψ_m, construct the maximal method graphs $\mathcal{M}ax(\psi_m, I_m)$ wrt. interface I_m
- for all variation points VP_i with specification ψ_{VP_i} construct the maximal flow graphs $\mathcal{M}ax(\psi_{VP_i}, I_{VP_i})$ wrt. interface I_{VP_i}
- compose the graphs, resulting in flow graph $\mathcal{G}_{\mathcal{M}ax}$, and model check the latter against the SHVM property ϕ.

$$\left(\biguplus_{m \in M_{pub}} \mathcal{M}ax(\psi_m, I_m) \uplus \biguplus_{VP_i \in \{VP_1, \ldots, VP_N\}} \mathcal{M}ax(\psi_{VP_i}, I_{VP_i}) \right) \models \phi$$

Correctness

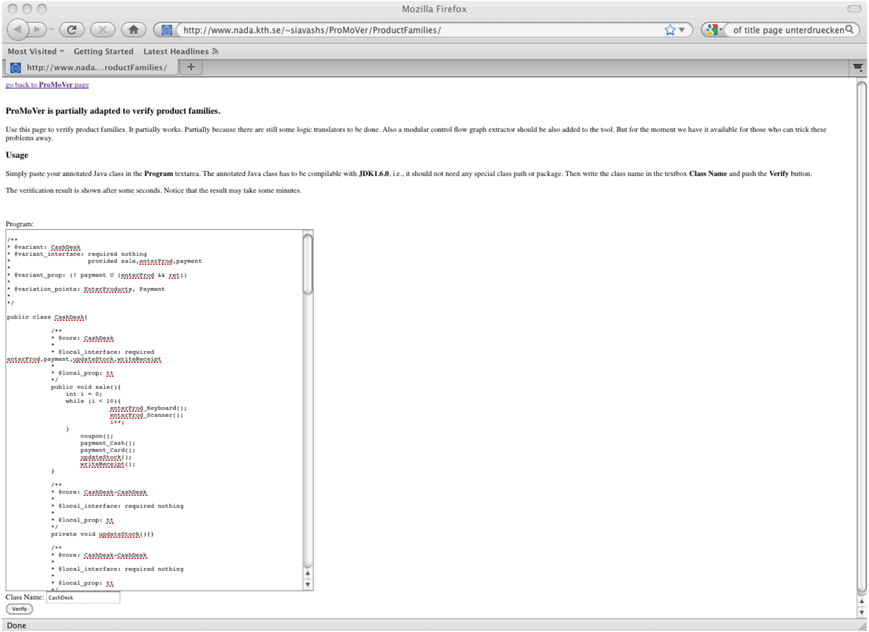
Theorem

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

Proof.

The proof is by induction on the nesting depth of $\mathcal{S}$. □

Tool Support: ProMoVer for ProductFamilies



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: [...]
            *
            * @variants: Keyboard,Scanner
            *
            *
            *
            * @variant: Keyboard-EnterProducts
            *
            * @variant_interface:   required nothing
            *                    provided enterProd
            *
            * @variant_prop: true
```

Class Name: CashDesk

Verify

Input for Cash Desk Example

Variant Annotations:

```
/**
* @variant: CashDesk
*
* @variant_interface: required
*                     provided sale,enterProd,payment
*
* @variant_prop:
*    sale --> ( !payment W (r && enterProd && X sale))
*
* @variation_points: EnterProducts, Payment
*/
public class CashDesk{ ...
```

Input for Cash Desk Example (2)

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();
}
```

Input for Cash Desk Example (3)

Variation Point Annotations:

```
/**
* @variation_point: EnterProducts_CashDesk
*
* @variation_point_interface: required
*                             provided enterProd
*
* @variation_point_prop: G !payment
*
* @variants: Keyboard,Scanner
**/
```

Analysis Result for Cash Desk Example

```
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
[...]

FIRST TASK TIME IS: 3.58 seconds // for verification of local specifications


Verifying variant Keybord-EnterProducts
THE VERIFICATION RESULT IS: YES.

Verifying variant Scanner-EnterProducts
THE VERIFICATION RESULT IS: YES.

[...]

Verifying variant CashDesk
THE VERIFICATION RESULT IS: YES.


THE WHOLE VERIFICATION TIME IS: 25.37 seconds
```

Evaluation

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:

Product Line	Depth	# Modules	# Products	t_{ind}[s]	t_{comp}[s]
`CD`	1	5	4	101	26
`CD/CH`	1	7	8	206	28
`CD/CT`	2	9	11	281	29
`CD/CH/CT`	2	11	20	518	30

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