Transformations of Software Product Lines:
A Generalizing Framework based on Category Theory

Gabriele Taentzer, Rick Salay, Daniel Strüder and Marsha Chechik
Software Product Lines (SPL)

- Manage a large number of similar but different artifact variants (products)

- Washing Machine Co.
SPL Structure

- SPL (annotative) represented by
  - Domain Model – combined parts from all products
  - Feature Model – shows possible features and restrictions for products
SPL Configuration – example

- +Dry product
  - Feature configuration: {Wash, Dry}
SPL of models: washing machine state machine

Domain Model

Locking
- [PRESS_START_WITH_DELAY]/ HeaterOn()
- [PRESS_START]/ Heat
- Heat V Delay
- Exit/HeaterOff()

Waiting
- Heat
- Heat V Delay
- Exit/HeaterOff()

Unlocking
- / QuickCool()
- ~Dry
- Dry

Washing
- / QuickCool()
- ~Heat
- Heat V Delay
- Entry/TempCheck()
- Heat

Dry
- Dry

Presence Conditions

Feature Model

Heat
- Delay
- Wash
- Excludes
- Delay

Dry
- Delay

Delay
- Heat
- Heat
- Heat
- Heat
- Heat
- Heat
- Heat

Transformations of Software Product Lines
Example Product: \{Wash, Dry\}

Domain Model:

- **Locking**
  - \[[PRESS\_START\_WITH\_DELAY]\]
  - \[[PRESS\_START]\] / HeaterOn()
  - Heat V Delay
  - Heat

- **Unlocking**
  - \[/ QuickCool()\]
  - \/-Dry

- **Washing**
  - \entry/TempCheck()\]
  - Heat
  - Heat V Delay

- **Waiting**
  - exit/HeaterOff()
  - Heat
  - Heat V Delay

- **Drying**
  - \[/ QuickCool()\]
  - Dry

Feature Model:

- **Wash**
- **Heat**
- **Delay**
- **Dry**

Heat ∨ Delay

Delay excludes Heat

\[
\neg Heat \\
\neg Dry \\
\neg Heat \\
Heat
\]
Example Product: \{Wash, Dry\}

Feature Model

Product: \{Wash, Dry\}

- Locking
- Unlocking
- Washing
- Drying
- QuickCool()

- Heat
- Delay
- Dry

[PRESS_START]/ wash.Start()
Outline

- Software Product Lines
- Transformations of SPLs
- What is the problem?
- Approach (part I): Category of SPLs
- Approach (part II): SPL Transformations using graph transformation rules
- Summary of results and next steps
Key types of SPL transformations in the Literature

1. **Feature model transformations**
   - Supports reasoning about additions, deletions, and modifications of features
   - e.g., Transformation rules to specify high-level feature editing operations [Bürdek et al.]

2. **Lifted model transformations**
   - Adapts single-product transformation rules to the entire SPL [Salay et al.]
   - Effect of lifting is same as applying the rule to each product separately.

3. **SPL refinement**
   - Supports safe evolution SPL by controlling impact on existing products.
   - e.g., Modifications restricted so that only a subset of products change [Sampaio et al.]
Motivation

- Types (1) & (3) apply only to feature models; type (2) applies only to domain models

  None of these types of transformation apply to entire SPL’s – feature and domain models!

- But, this is needed in practice:
  - Addition or deletion of features usually entails the corresponding changes in the domain model

- Research Objective:
  - A formal characterization of SPL transformations that addresses both feature and domain models.
Approach

- Build on existing formal theories: category theory and theory of Algebraic Graph Transformations (AGT)[1]

- Our strategy: given any suitable (to be defined) category $\mathcal{Mod}$ of models,
  1. Show how to define the category $\mathcal{PPL}_{\mathcal{Mod}}$ of SPLs having $\mathcal{Mod}$ models as domain models.
  2. Use AGT to define transformation rules for SPLs in $\mathcal{PPL}_{\mathcal{Mod}}$

- Benefits:
  - General, systematic and covers both feature and domain model parts of an SPL
  - Gets formal techniques from AGT that support SPL transformation development
    - e.g., conflict and dependency analysis, confluence analysis, etc.

Outline

- Software Product Lines
- Transformations of SPLs
- What is the problem?
- Approach (part I): Category of SPLs
- Approach (part II): SPL Transformations using graph transformation rules
- Summary of results and next steps
Defining category $\mathcal{PL}_{\text{Mod}}$ – objects and morphisms

- An SPL $P = (F_P, \Phi_P, M_P, f_P)$ of $\mathcal{PL}_{\text{Mod}}$ consists of:
  - (feature model) Set $F_P$ of features with set $\Phi_P$ of propositional feature constraints over $F_P$ defining allowable feature configurations
  - (domain model) $\text{Mod}$-model $M_P$
  - (presence conditions) Function $f_P$ assigns a propositional formula over $F_P$ to each submodel of $M_P$ defining for which feature configurations the submodel is present

- An SPL morphism $h: P \rightarrow Q$ is a mapping from SPL $P$ to $Q$ such that
  - (feature mapping) $h$ maps $F_P$ to $F_Q$
  - (domain mapping) $h$ maps $D_P$ to $D_Q$ (using a $\text{Mod}$-morphism)
  - above mappings constrained so that products of $P$ map into products of $Q$
Washing machine SPL $WM$ in $PL_SM$

Domain Model $D_{WM}$

- **Locking**
  - [PRESS_START_WITH_DELAY]
  - [PRESS_START] / HeaterOn()

- **Unlocking**
  - / QuickCool()

- **Washing**
  - entry / TempCheck()

- **Drying**
  - / QuickCool()

Feature Model

$F_{WM}: \{\text{Wash, Heat, Delay, Dry}\}$

$\Phi_{WM}: \{\text{Wash, } \neg (\text{Delay } \land \text{ Heat)}\}$

Presence condition (example)

$f_{WM}$

$\text{f_{WM}}$

Dry $\land$ Heat
Example Product: \{Wash, Dry\}

Product: \{Wash, Dry\}

- **Locking**
  - [PRESS_START]/ wash.Start()
- **Unlocking**
- **Washing**
- **Drying**
  - / QuickCool()

Presence condition (example)

$\text{Wash} \land \text{Dry} \land \neg \text{Heat} \land \neg \text{Delay}$

Products are the **maximal submodels** with presence condition for a configuration.
Morphisms in $PL_{SM}$

Feature Model $F_B$: {Base}  
$\Phi_B$: {}

Domain Model $D_B$

- $x_1$
- $x_2$

$\tau_i / a$

Feature Model $F_{WM}$: {Wash, Heat, Delay, Dry}  
$\Phi_{WM}$: {Wash, $\neg(Heat \land Delay)$}

Domain Model $D_{WM}$

- Locking
  - [PRESS_START_WITH_DELAY]
  - [PRESS_START] / HeaterOn()
  - exit/HeaterOff()
- Washing
  - entry/TempCheck()
- Drying
  - /QuickCool()
- Unlocking
  - /QuickCool()
Morphisms map products

Product: \{Base\}
- x1
- x2

Product: \{Wash, Dry\}
- Locking
- Unlocking
- Washing
- Drying

\[ \text{PRESS\_START}/ \text{wash\_Start()} \]

\[ \text{PRESS\_START}/ \text{wash\_Start()} \]

\[ h \]
Key result: Pushout construction in $PL_{Mod}$

- In $PL_{Mod}$ we can use the standard category theory pushout construction to combine two SPLs that are related by a common SPL.

See Paper for Details!
Outline

- Software Product Lines
- Transformations of SPLs
- What is the problem?
- Approach (part I): Category of SPLs
- Approach (part II): SPL Transformations using graph transformation rules
- Summary of results and next steps
Transformations in $PL_{Mod}$

- Since we can construct pushouts in $PL_{Mod}$, we can use the double pushout approach from AGT to define transformation rules.

Key result: We show the existence and uniqueness of rule applications for $PL_{Mod}$. See details in the paper.
Washing machine SPL $WM$

**Feature Model**

$F_{WM}: \{\text{Wash, Heat, Delay, Dry}\}$

$\Phi_{WM}: \{\text{Wash, } \neg(Delay \land Heat)\}$

**Domain Model $D_{WM}$**

- **Locking**
  - [PRESS_START_WITH_DELAY] / HeaterOn()
  - [PRESS_START] / HeaterOn()
  - [PRESS_START] / wash.Start()

- **Unlocking**
  - / QuickCool()

- **Waiting**
  - exit/HeaterOff()
  - /wash.Start()

- **Washing**
  - entry/TempCheck()

- **Drying**
  - / QuickCool()
  - / QuickCool()

Transformations of Software Product Lines
Applying AddBeepFeature: Match LHS

Domain Model $D_{WM}$

- Locking
  - [PRESS_START_WITH_DELAY] HeaterOn()
  - [PRESS_START] Heat
  - Heat V Delay

- Waiting
  - exit/HeaterOff()
  - Heat
  - Heat V Delay

- Washing
  - entry/TempCheck()
  - Heat
  - Heat V Delay
  - /wash.Start()

- Drying
  - / QuickCool()
  - Dry
  - Dry

Feature Model

$F_{WM}: \{\text{Wash, Heat, Delay, Dry}\}$

$\Phi_{WM}: \{\text{Wash, } \neg(\text{Delay } \land \text{Heat})\}$

Feature Model $F_B: \{\text{Base}\}$

$\Phi_B: \{\}$

Domain Model $D_B$

- x1
  - [t]: a

- x2

Transformations of Software Product Lines
Applying AddBeepFeature: Result SPL $WMB$ of double pushout

Domain Model $D_{WMB}$

Rule application affects domain model, feature model and presence conditions!
Summary of Results

- Given any suitable category $\text{Mod}$ of models,
  1. Showed how to define the category $PL_{\text{Mod}}$ of SPLs having $\text{Mod}$ models as domain models.
  2. Defined the pushout construction for $PL_{\text{Mod}}$
  3. Showed how to define transformation rules for $PL_{\text{Mod}}$ using double pushout
  4. Proved the existence and uniqueness of rule application.

- Illustrated how an SPL rule can affect both feature and domain model parts of an SPL
  - i.e., we have exceeded these limitations in the literature
Next Steps

- Have only partially proven that $PL_{Mod}$ satisfies the formal requirements for AGT
  - We are completing this task
- Plan to implement formal analysis techniques from AGT for $PL_{Mod}$
  - e.g., conflict and dependency analysis, confluence analysis, etc.
  - Henshin is likely the platform
- Want to explore the scope of SPL transformations expressible using our approach.
- Want to explore the kinds of SPLs obtained by using different model categories for $Mod$