Feature Link Propagation Across Variability Representations with Isabelle/HOL

Florian Kammüller, Alexander Rein, Mark-Oliver Reiser
Technische Universität Berlin
Fakultät IV - Softwaretechnik - TEL 12-3
Ernst-Reuter-Platz 7, D-10587 Berlin, Germany

ABSTRACT

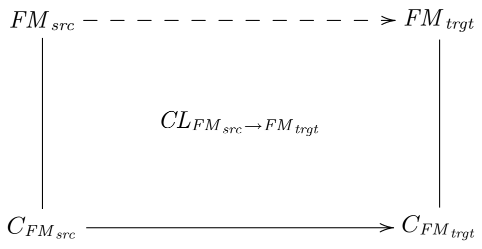
When dealing with highly complex product lines it is usually indispensable to somehow subdivide the overall product line into several smaller, subordinate product lines and to define orthogonal views on the line's variability tailored to particular purposes, such as end-customer configuration. In this article we report on an ongoing research effort for dealing with feature links, i.e. logical constraints between features, in such a setting, by propagating such logical constraints defined in lower-level product lines to a higher level or from one view to another.

1. INTRODUCTION

Product line engineering [CN02, PBvdL05] is still faced with several significant challenges when it comes to dealing with highly complex product ranges, as typically encountered in industrial domains. The product range of an automotive manufacturer, for example, incorporates a multitude of subsystems provided by external suppliers, each with its own variability organized in the suppliers' product lines with diverse scopes and target groups, turning the manufacturer's product line into an aggregate of a multitude of lower-level, subordinate product lines [RTW09]. But even within a single company—manufacturer or supplier—often several different views on variability are required, due to distinct viewpoints of stakeholders and diverse life-cycles of subcomponents or individual development projects.

The concept of configuration links [Rei09], resulting from our earlier research on these issues, provides means to partition a complex product line into several smaller, subordinate product lines and allows to introduce orthogonal, i.e. differently structured views on a product line's variability. It is centered around ordinary cardinality-based feature modeling [CHE05]: subordinate product lines and orthogonal views are each represented by a feature model; then, configuration links are used to relate them.

A *configuration link* in this sense is a directed association from n so-called *source feature models* to m other feature models called *target feature models*, with $n, m \in \mathbb{N}_0$; $m \geq 1$. It defines how to configure the m target feature models, depending on the configuration of the n source feature models (for $n = 0$, it defines an invariant configuration of the target feature models). With the information captured in such a configuration link, it is then possible to deduce configurations of the target feature models whenever configurations of the source feature models are provided. In the remainder of this article we focus on the standard case of a single source and target feature model each, illustrated in Figure 1. The two feature models FM_{src} and FM_{trgt} are supplied with a configuration link $CL_{FM_{src} \rightarrow FM_{trgt}}$ (dashed arrow). Thus, it is possible to derive a configuration $C_{FM_{trgt}}$ of FM_{trgt} from any given configuration $C_{FM_{src}}$ of FM_{src} (solid arrow).



$$\begin{tikzcd}
FM_{src} \arrow[r, dashed] \arrow[d, dash] \arrow[dr, phantom, "CL_{FM_{src} \rightarrow FM_{trgt}}"] & FM_{trgt} \arrow[d, dash] \\
C_{FM_{src}} \arrow[r] & C_{FM_{trgt}}
\end{tikzcd}$$

Figure 1: Relations defined by a configuration link.

While the core concept of configuration links is thoroughly consolidated and stable already, we are currently investigating several advanced topics associated with this approach. In this article we report on our present research on dealing with *feature links* in this context (i.e. logical dependencies/constraints such as an "excludes" link between two incompatible features). We aim for providing techniques to understand and manage how the addition of a novel feature link to the target feature model affects the configuration of the source feature model by automatically transforming the target-side feature link, i.e. a feature link in FM_{trgt}, into a corresponding source-side feature link with the same effect, a process we call *feature link propagation.* The work is based on set theoretical formalizations and implementations in the theorem prover Isabelle/HOL [NPW02].

For illustration purposes and to show its practical use, we first introduce feature link propagation informally with an example (Section 2) before providing a detailed, more formal definition (Section 3). Then, Section 4 elaborates on our current solution before Section 5 concludes with an account of the current status and prospective challenges.

2. MOTIVATION & BASIC IDEA

As an example, consider a car product line's core feature model with a strong technical focus and a customer-oriented feature model providing an orthogonal view on the technical feature model tailored for immediate configuration by the end-customer, as shown in Figure 2. A configuration link from the customer-oriented to the technical feature model, represented by the dashed arrow in the center, then allows to derive a technical configuration from any given customer configuration. For the purpose of this illustrative example we assume a very simple configuration link definition: if the `Comfort` package is selected in the customer feature model, then the rain sensor will be selected in the technical feature model; similarly, if source-side feature `USA` is selected, then feature `Radar` will be selected on the target side, i.e. cars for the U.S. market will always obtain a radar.

Now, let us see what happens if a new technical incompatibility arises, for example between features `Radar` and `RainSensor`, represented in Figure 2 by the "excludes" link between the two features in model *TechnicalFM*. When taking into account the definition of the configuration link, it becomes obvious that this also affects the configuration of the customer feature model: the comfort package—as represented by feature `Comfort`—must no longer be selected together with feature `USA`, because this would result in the selection of the incompatible features `Radar` and `RainSensor` on the target side. Our new technical incompatibility makes the entire comfort package unavailable to the U.S. market!

To make this implicit effect show up explicitly in the customer-oriented feature model we would have to add a feature link of type "excludes" between features `USA` and `Comfort`. This activity of adding feature links to a source feature model in order to explicitly highlight the impact of a particular feature link on the target side is what we call *feature link propagation*. In the example, the "excludes" link between `Radar` and `RainSensor` would be propagated into a source-side "excludes" link between `USA` and `Comfort`.

A theory and tool for automatically conducting such feature link propagations would be of high value to clearly understand how low-level, technical incompatibilities and dependencies affect higher-level representations of a product line's variability. Then, in case the effect is undesired, the product line infrastructure or the mapping of high-level to low-level variability representations, as defined within the configuration link(s), can be adapted accordingly.

3. DEFINITION

Having described the notion of feature link propagation informally from an application perspective in the preceding section, we now provide a detailed definition of such a propagation. This will serve as a terminological framework and will precisely establish what we want to achieve.

Figure 1 above introduced the usual situation when dealing with configuration links: a source feature model FM_{src} is linked to a target feature model FM_{trgt} and, by applying the configuration link to a source configuration $C_{FM_{src}}$ we obtain a corresponding target configuration $C_{FM_{trgt}}$.

However, this slightly simplifies the actual situation by looking only exemplarily at a single source configuration $C_{FM_{src}}$ and its corresponding target configuration. In order to comprehend feature link propagation, we have to think of a configuration link as a function. Let $\mathcal{C}_{FM}$ be the set of all valid configurations of feature model FM. Then, we can define a configuration link from source feature model FM_{src} to target feature model FM_{trgt} as

$$CL_{FM_{src} \to FM_{trgt}} : \mathcal{C}_{FM_{src}} \longrightarrow \mathcal{C}_{FM_{trgt}}$$

i.e. a function from the set of all valid source configurations $\mathcal{C}_{FM_{src}}$ to the set of all valid target configurations $\mathcal{C}_{FM_{trgt}}$.

This way, we can now apply standard terminology to obtain an important set of configurations: given a configuration link CL, this function's image denotes the set of all target configurations that can be obtained by first starting with a valid source configuration and then deriving a target configuration by applying the configuration link. We define

$$\begin{aligned} CL^{\rightarrow}_{FM_{src} \to FM_{trgt}} &:= CL_{FM_{src} \to FM_{trgt}}(\mathcal{C}_{FM_{src}}) \\ &:= \{CL_{FM_{src} \to FM_{trgt}}(c) \mid c \in \mathcal{C}_{FM_{src}}\} \end{aligned}$$

Let us now turn to feature links and their impact on a feature model's valid configurations. As stated in Section 1 above, a feature link imposes a constraint on a feature model's configuration. For example, it might define that one feature may never be selected together with a particular other feature within the same configuration. Hence, by introducing feature links, we reduce the number of valid configurations. More formally: given feature model FM we introduce one or more (additional) feature links and obtain FM'; we then know that

$$\mathcal{C}_{FM'} \subseteq \mathcal{C}_{FM}$$

For convenience, we define abbreviations: for a set L of one or more feature links for feature model FM we define $FM^{|L|}$ as the feature model identical to FM but with these feature links added and $\mathcal{C}^{|L|}_{FM}$ as the set of all valid configurations of FM that comply to the constraints in L. The above proposition can now be rephrased as

$$\mathcal{C}^{|L|}_{FM} \subseteq \mathcal{C}_{FM}$$

We cannot be sure that $\mathcal{C}^{|L|}_{FM}$ is a true subset because a feature link might define some redundant constraint on the feature model's configuration. For example, adding a "needs" link from a child to its parent feature does not introduce any further constraint on the model beyond what is already defined by way of the parent/child relation (i.e. a child feature can only be selected if its parent is selected).

This provides all that is required to explain feature link propagation. When thinking of a set L of one or more feature links for the target(!) feature model of a given configuration link, we would like to see a derived set of feature links L' for its source(!) feature model with

$$CL_{FM_{src} \to FM_{trgt}}(\mathcal{C}^{|L'|}_{FM_{src}}) \;\subseteq\; CL^{\rightarrow}_{FM_{src} \to FM_{trgt}} \cap \mathcal{C}^{|L|}_{FM_{trgt}}$$

Definition 1. Given a configuration link $CL_{FM_{src} \to FM_{trgt}}$ and a set L of one or more feature links for FM_{trgt}, we define the *backward propagation* of the feature links in L as the activity of finding a set L' of feature links for feature model FM_{src} such that the above proposition holds. □

Ideally we would like to see an equivalence instead of $\subseteq$ in the proposition above. Then, the propagated feature links in L' were required to *only* preclude such configurations from $\mathcal{C}_{FM_{trgt}}$ that are actually eliminated by the links in L. At the end of our project, we hope to be able to prove that there is always a set L' even if equivalence were required above, but presently we are using the weaker definition.

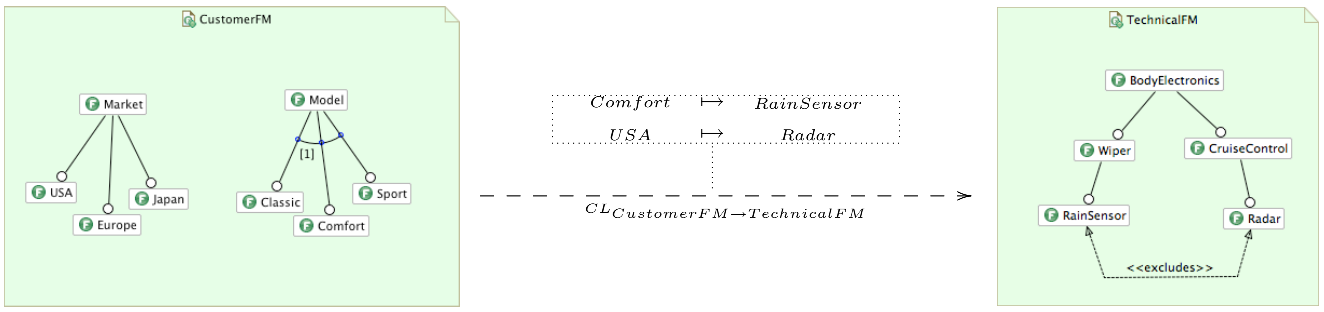


Figure 2: A configuration link from a customer-oriented (left) to a technical feature model (right).

$$\begin{tikzcd}
FM_{src} \arrow[rr, dashed, bend left=10, "CL_{FM_{src} \rightarrow FM_{trgt}}"] \arrow[d, dash] & & FM_{trgt} \arrow[ll, dotted, bend left=10, "rmap_{CL_{FM_{src} \rightarrow FM_{trgt}}}"] \arrow[d, dash] \\
\phi_{FM_{src}} & & \phi_{FM_{trgt}} \arrow[ll, Rightarrow, "trafo_{rmap_{CL_{FM_{src} \rightarrow FM_{trgt}}}}"']
\end{tikzcd}$$

Figure 3: Transformation of a propositional logical formula w.r.t. reverse mapping

4. SOLUTION

In this section, we present a solution for the backward propagation of feature links on a conceptual level. We focus on "needs" and "excludes" feature links. The first step of our approach is that we express feature links by propositional logical formulae over the features of a model. Then, we will introduce a normal form for the definition of a configuration link from which a reverse mapping of the features on source and target side can be obtained. Finally, we will introduce a transformation of logical formulae with respect to such a reverse mapping. This allows transforming a logical formula of the target feature model to a deviated logical formula of the source feature model. This deviated formula then constitutes the propagated feature link.

We also implemented this approach in Isabelle/HOL. Our implementation automatically applies our transformation to formalized feature links of the target model and delivers propagated feature links of the source model.

Figure 3 provides an illustrative overview of our approach. In the first step, we express one or more feature links of FM_{trgt} by a propositional logical formula $\phi_{FM_{trgt}}$ and then compute the reverse mapping $rmap_{CL_{FM_{src} \rightarrow FM_{trgt}}}$ with respect to the configuration link. Then, by applying the above transformation *trafo* on $\phi_{FM_{trgt}}$, we obtain $\phi_{FM_{src}}$, which represents the propagated feature link in the source model.

4.1 Formalizing Feature Links

First we require a formalization of feature links. An "excludes" feature link between two features f_1 and f_2 means that two features are optional alternative w.r.t. configuration. We can express this by the formula $\neg f_1 \vee \neg f_2$, which is equivalent to $\neg(f_1 \wedge f_2)$. A "needs" feature link (feature f_1 requires feature f_2) can be denoted as $f_1 \Rightarrow f_2$, which is equivalent to $\neg f_1 \vee f_2$. Obviously, more complex constraints and feature links can be constructed by composing several formulae. Note that the set of logical connectives $\{\neg, \wedge, \vee\}$ is functional complete. So we can express every propositional logic formulae with these three operators.

4.2 Normalized Configuration Links

As mentioned above, we require a normalized form of a configuration link's definition. We consider a configuration link as a set of configuration decisions, i.e. individual rules stating how to configure the target feature model depending on a given configuration of the source feature model.

Definition 2. Given feature models FM_{src} and FM_{trgt} with sets of features F_{src} and F_{trgt} then a *(normalized) configuration link* is defined as follows: $CL_{FM_{src} \rightarrow FM_{trgt}} \subseteq \{f_s \mapsto f_t \mid f_s \in F_{src}, f_t \in F_{trgt} \cup \{\neg f \mid f \in F_{trgt}\}\}$. Each element in the configuration link is called *(normalized) configuration decision.* □

Note that we only allow atomic formulae (of the source model) on the left-hand sides of configuration decisions, but we allow negated features (of the target model) on the right-hand sides.

4.3 Reverse Mapping

Here, we formalize a reverse mapping with respect to a configuration link. This reverse mapping turns around the configuration decisions of a configuration link and is the base for our transformation.

Definition 3. Let $CL_{FM_{src} \rightarrow FM_{trgt}}$ (short CL) be a configuration link as defined in Def. 2. Then a reverse mapping w.r.t. CL is a partial function $rmap_{CL} : F_{trgt} \rightharpoonup \Phi_{src}$ as defined in the following, where Φ_{src} is the set of propositional logic formulae over F_{src} with the reduced set $\{\neg, \wedge, \vee\}$ of logical connectives.

(1) For all features $f_t \in F_{trgt}$ with $\nexists f \in F_{src} : (f \mapsto \neg f_t) \in CL$ we define

$$rmap_{CL}(f_t) = f_{s_1} \vee f_{s_2} \vee ...$$

for all $f_{s_1}, f_{s_2}, ... \in \{f \in F_{src} | (f \mapsto f_t) \in CL\}$.

(2) For all features $f_t \in F_{trgt}$ with $\exists f, f' \in F_{src} : (f \mapsto f_t) \in CL \wedge (f' \mapsto \neg f_t) \in CL$ we define

$$rmap_{CL}(f_t) = (f_{s_1} \vee f_{s_2} \vee ...) \wedge (\neg f'_{s_1} \wedge \neg f'_{s_2} \wedge ...)$$

for all $f_{s_1}, f_{s_2}, ... \in \{f \in F_{src} | (f \mapsto f_t) \in CL$ and $f'_{s_1}, f'_{s_2}, ... \in \{f \in F_{src} | (f \mapsto \neg f_t) \in CL\}$.

(3) For all features $f_t \in F_{trgt}$ with $\nexists f \in F_{src} : (f \mapsto f_t) \in CL$ the reverse mapping $rmap_{CL}(f_t)$ is not defined — roughly spoken, all features without a preimage in F_{src} cannot be mapped. □

According to [Rei09] *an exclude has priority over an include* within a configuration link definition, i.e. if the two configuration decisions $(f_1 \mapsto f)$ and $(f_2 \mapsto \neg f)$ were defined and f_1 and f_2 were both selected in the source configuration, then $(... \mapsto \neg f)$ has priority and target-side feature f will be excluded in the resulting target configuration. To account for this prioritization we split our reverse mapping into two cases. If, for a feature $f_t \in F_{trgt}$, there is both a configuration decision $(f_{s_1} \mapsto f_t)$ (include) and another $(f_{s_2} \mapsto \neg f_t)$ (exclude), we can combine both decisions and obtain $((f_{s_1} \wedge \neg f_{s_2}) \mapsto f_t)$ and $(f_{s_2} \mapsto \neg f_t)$. So, if feature f_t is selected, we can conclude that $(f_{s_1} \wedge \neg f_{s_2}) \vee \neg f_{s_2}$ holds in the source feature model. This formula can be simplified to $\neg f_{s_2}$, corresponding to our reverse mapping. If more source features $f_{s_1}, f_{s_2}, ... \in F_{src}$ exclude a feature $f_t \in F_{trgt}$ and f_t is selected on the target side, we can conclude that the features $f_{s_1}, f_{s_2}, ...$ were deselected in the source configuration. Therefore, we defined our reverse mapping as conjunction $\neg f_{s_1} \wedge \neg f_{s_2} \wedge ...$ (in this case). For the target features which are not excluded by a source feature (all $f_t \in F_{trgt}$ with $\nexists f \in F_{src} : (f, \neg f_t) \in CL$), we defined the reverse mapping as disjunction of all source features $f_{s_1}, f_{s_2}, ... \in F_{src}$ which are mapped to f_t ($(f_{s_1} \mapsto f_t)$, $(f_{s_2} \mapsto f_t)$, ...) since we can only conclude that at least one of the source features was selected if the target feature f_t is selected.

4.4 Transformation of Logical Formulae

Now we focus on the transformation of logical formulae with respect to configuration links.

Definition 4. Given a transformation link CL and a reverse mapping $rmap_{CL}$ as defined in Def. 2 and 3. Then a transformation w.r.t. a reverse mapping $rmap_{CL}$ is recursively defined as $trafo_{rmap_{CL}} : \Phi_{trgt} \rightarrow \Phi^{\perp}_{src}$ with

$$trafo_{rmap_{CL}}(\phi_t) = \begin{cases} rmap_{CL}(f_t) & \text{if } \phi_t = f_t \in F_{trgt} \text{ and } rmap_{CL}(f_t) \text{ is defined} \\ \perp & \text{if } \phi_t = f_t \in F_{trgt} \text{ and } rmap_{CL}(f_t) \text{ is not defined} \\ \neg trafo_{rmap_{CL}}(\phi'_t) & \text{if } \phi_t = \neg\phi'_t \text{ and } \phi'_t \in \Phi_{trgt} \\ trafo_{rmap_{CL}}(\phi'_t) \wedge trafo_{rmap_{CL}}(\phi''_t) & \text{if } \phi_t = \phi'_t \wedge \phi''_t \text{ and } \phi'_t, \phi''_t \in \Phi_{trgt} \\ trafo_{rmap_{CL}}(\phi'_t) \vee trafo_{rmap_{CL}}(\phi''_t) & \text{if } \phi_t = \phi'_t \vee \phi''_t \text{ and } \phi'_t, \phi''_t \in \Phi_{trgt} \end{cases}$$

where $\Phi^{\perp}_{src}$ and Φ_{trgt} are the sets of propositional logic formulae over F_{src} resp. F_{trgt} with the reduced set $\{\neg, \wedge, \vee\}$ of logical connectives. Note that $\Phi^{\perp}_{src}$ additionally contains formulae with undefined features ($\perp$). Undefined features are required since the transformation is total, although the reverse mapping is partial. $\Box$

With this definition, we obtain a formula $\phi_s \in \Phi^{\perp}_{src}$ over the features F_{src} of the source model FM_{src}. This formula represents the propagated feature link for the source model.

5. CONCLUSION

Based on the concept of configuration links, we have presented in this paper a logic-based approach to derive feature links on source feature models implicitly induced by corresponding feature links on target feature models. We have introduced the theoretical foundation for such a derivation by defining *backward propagation* that consistently transforms feature links across orthogonal variability representations. Beyond a theoretical characterization, we propose a constructive solution to derive feature links: a reverse map presently enables the backward propagation of "needs" and "excludes" feature links.

In addition to the related work cited above, several other approaches and publications are dealing with a hierarchical decomposition of large-scale product lines and their variability representations [vO04, BN09, Kru06]. To the best of our knowledge, none of these provide support for automatically transforming logic constraints from one variability representation to another.

As a prototypical support engine for managing feature link propagation, we are using Isabelle/HOL. We have argued in Section 4 that the logical transformation *trafo* correctly realizes backward propagation. In our experiments, we already validated our solution concept on several examples. In the future, we plan to enrich our Isabelle/HOL model to be able to reliably prove such soundness properties. Another important future project is the completion of reverse maps to general feature constraints other than "needs" and "excludes" links.

Finally, as the name suggests, the notion of backward propagation might be complemented with a propagation in the opposite direction from source to target feature model, a *forward propagation*.

6. REFERENCES

[BN09] Felix Bachmann and Linda Northrop. Structured variation management in software product lines. In *Proceedings of the 42nd Hawaii International Conference on System Sciences (HICSS-42)*. IEEE Computer Society Press, 2009.

[CHE05] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practices*, 10(1):7–29, 2005.

[CN02] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns.* Addison-Wesley, 2002.

[Kru06] Charles W. Krueger. New methods in software product line development. In *Proceedings of the 10th International Software Product Line Conference (SPLC 2006)*, pages 95–102, 2006.

[NPW02] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer-Verlag, 2002.

[PBvdL05] Klaus Pohl, Günter Böckle, and Frank van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer, Heidelberg, 2005.

[Rei09] Mark-Oliver Reiser. Core concepts of the Compositional Variability Management framework (CVM). Technical Report, no. 2009-16, Technische Universität Berlin, 2009.

[RTW09] Mark-Oliver Reiser, Ramin Tavakoli, and Matthias Weber. Compositional variability. In *Proceedings of the 42nd Hawaii International Conference on System Sciences (HICSS-42).* IEEE Computer Society Press, 2009.

[vO04] Rob van Ommering. *Building Product Populations with Software Components.* PhD thesis, University of Groningen, 2004.

APPENDIX

Implementation in Isabelle

This appendix presents the implementation of our approach in Isabelle/HOL. For our formalization, we used Isabelle 2009 with the X-Symbol package for a prettier rendering of mathematical symbols.

In our approach, we decide between application dependent and application independent parts. In this context, application (in)dependency means that this part has (not) to be redefined for different applications, e.g. different feature models, configuration links or logical formulae to be transformed. Application dependent parts are the definition of concrete features of a model, the (reverse) mapping of the features and the logical formula to be transformed. Many fragments of these parts could be generated fully automated by a corresponding feature modeling tool, for example CVM (see [Rei09]) – although this functionality is not yet implemented. Application independent parts are the definition of a datatype for logical formulae, axioms for simplification and the definition of the transformation of logical formulae. We decided to use Isabelle's simplifier for the execution of the transformation and for simplifying the result. We specified our own logic to be able to define explicit operation for the transformation of formulae. Although we first considered using directly the higher order logic of Isabelle/HOL. This turned out to be not practical, because automatic simplification routines obstruct the control of formula transformations necessary for modelling configuration links. Therefore, we defined datatypes for features and for logical formulae. Note that we did not formalize feature models in Isabelle. We only implemented our transformation of logical formulae similar to the description in Section 4.

First of all, we defined the set of features (of the source and the target model together) as datatype `Feature = A|B|...`, where `A,B,...` are the concrete features. This datatype has to be redefined for every pair of feature models. Then we added the application independent datatype named `Formula` as base for all propositional logical formulae over the reduced set $\{\neg, \wedge, \vee\}$ of logical connectives.

```
datatype Formula =
  Cons Feature ("%")
  | Neg Formula ("¬ _" [500] 500)
  | And Formula Formula (infixr "∧" 200)
  | Or Formula Formula (infixr "∨" 100)
  | TRUE | FALSE
```

This definition is quite intuitive: A formula can be a feature, the constant `TRUE`, the constant `FALSE` or several features connected by the defined logical connectives.

After defining these datatypes, we teached Isabelle's simplifier through the definition of axioms, e.g. we formalized idempotence, absorption, commutativity, associativity, distributivity, De Morgan's law, complements and some other known axioms and laws of propositional logic. With this implementation, we can now express every propositional logical formulae in our own logic and Isabelle is able to simplify these formulae automatically by the command `apply simp`. This is very important because our transformation sometimes produces very long and confusing formulae, which should be simplified for a better readability.

Now we focus on our transformation, which we implemented as recursive function. The input of this function are the application dependent reverse mapping `rmap` (cf. Def. 3), which is (here) a partial function from `Formula` to `Formula`, and the formula to be transformed. Roughly spoken, the transformation applies the given reverse mapping to all features of the input formula. It delivers the transformed formula as output.

```
primrec trafo:: "[Formula ⇀ Formula,Formula] ⇒ Formula"
where
 trafo rmap (Cons x) =
   (if ((rmap (Cons x)) = None) then
     (Cons x)
   else
     (the (rmap (Cons x))))
 | (trafo rmap (¬ x)) = (¬ (trafo rmap x))
 | (trafo rmap (x ∧ y)) =
              ((trafo rmap x) ∧ (trafo rmap y))
 | (trafo rmap (x ∨ y)) =
              ((trafo rmap x) ∨ (trafo rmap y))
```

If the transformation is applied to a feature, then the result is the reverse mapping of the feature (if it exists) or the feature itself (if there is no reverse mapping for this feature). Note this is a small difference to our formal definition in Section 4. The application of the transformation to complex formulae is defined recursively – roughly spoken, the mapping is pulled into the formula (cf. Def. 4).

The application dependent reverse mapping (cf. Def. 3) between the concrete features is defined as partial function from `Formula` to `Formula`. Even though, we only allow atomic features (and not complex formulae) in its domain, according to Definition 3. This definition has technical reasons.

In order to accomplish a transformation to a given logical formula `<FORM>` (over the features of a feature model) w.r.t. a reverse mapping `<RMAP>`, we only have to apply our transformation to the formula. Therefore, we define a lemma in Isabelle with the formula to be translated and apply Isabelle's simplifier.

```
lemma "trafo <RMAP> <FORM> = ?Y";
  apply (unfold <RMAP>_def);
  apply simp
done
```

The result of this simplification is the derived propositional logical formula for the source feature model, which formalizes propagated feature links.

Implementation of Example 1

In this Section, we revisit our example shown in Figure 2. Of course we only have to implement the application dependent parts of this example.

First of all, we define the features of the example in Isabelle.

```
datatype Feature =
    Market | USA | Europe | Japan | Model | Classic |
    Comfort | Sport | BodyElectronics | Wiper |
    RainSensor | CruiseControl | AdaptiveCC | Radar
```

Then we define the reverse mapping. Therefore, we apply our algorithm defined in Def. 3 to the given mapping ($Comfort \mapsto RainSensor, USA \mapsto Radar$) and obtain the following reverse mapping.

```
constdefs rmap_1 :: "Formula ⇀ Formula"
  "rmap_1 ==
   [% RainSensor ↦ % Comfort,
    % Radar  ↦ % USA]"
```

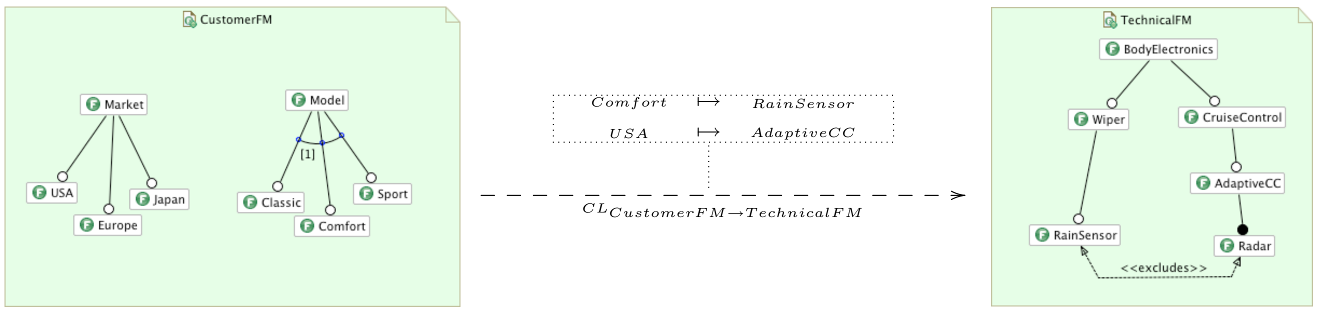


Figure 4: Second example of feature link propagation, modified from Figure 2.

The target feature model of the example excludes all configurations with a rain sensor and a radar. Expressing this feature link in propositional logic leads to $\neg(RainSensor \wedge Radar)$. Now we define a lemma in Isabelle and use the simplifier for applying our transformation.

```
lemma "trafo rmap_1
       (Neg (And (% RainSensor) (% Radar)))
       = ?Y";
apply (unfold rmap_1_def);
apply simp
done
```

The result of this transformation delivers the estimated result $\neg Comfort \vee \neg USA$. For a better understandability, this can be rearranged to $\neg(Comfort \wedge USA)$. This step can be done manuelly or it can also be implemented in Isabelle in the following way:

```
lemma "trafo rmap_1
       (Neg (And (% RainSensor) (% Radar)))
       = ?Y";
apply (unfold rmap_1_def);
apply (rule trans)
apply simp
apply (rule sym)
apply (rule de_Morgan_conj)
done
```

Finally, we can construct a propagated *exclude* feature link in the source model: *The comfort package is not suitable in the USA.*

In practice, the application independent parts should be automatically generated by a feature modeling tool. Thereby we see no problem since the reverse mapping can be easily generated by configuration links. Note that the feature modeling tool only has to derive a naive formula as reverse mapping since the interpretation (resp. simplification) of the logical formula is alredy covered by our Isabelle implementation. Analogously, formalized feature links (i.e. the logical formula to be transformed) can be easily generated by a feature modeling tool. The third and last application dependent part that should be automatically generated are the concrete features, which is obviously trivial. Up to now, our implementation is only a prototypical implementation and the simplification of logical formulae is not perfect. Therefore, importing the propagated feature links in a feature modeling tool (without manual influence) could sometimes be problematic. Up to now, we did not define an interface between out implementation and common feature modeling tools.

Implementation of Example 2

This mechanism can also be applied to more intricate situations in which the structure—i.e. the parent/child relations—of the target feature model have to be taken into account. To illustrate this, let us consider another small example, given in Figure 4.

In this example, the selection of feature USA includes feature $AdaptiveCC$ in the target feature model. If $AdaptiveCC$ is selected, then the mandatory feature $Radar$ has to be selected also. Up to now, we do not include the formalization of mandatory features in our approach since we did not implement the whole feature models in Isabelle, but only the transformation of logical formulae. Therefore, we have to add additional mappings for such constraints. In this example, we add $(AdaptiveCC \mapsto Radar)$ (in reverse) to our mapping function.

```
constdefs rmap_2 :: "Formula ⇀ Formula"
  "rmap_2
   [% RainSensor ↦ % Comfort,
    % AdaptiveCC ↦ % USA,
    % Radar ↦ % AdaptiveCC]"
```

Applying our transformation in the same way as described in the first example now leads to the constraint $\neg AdaptiveCC \wedge Comfort$. This is, of course, correct but we could apply our mapping again and would obtain $\neg USA \wedge Comfort$. For automating this, we formulated another transformation, that applies the mapping recursively as often as possible. As termination condition, we inserted a numeric value as additional parameter.

```
primrec trafoX::
     "[nat, Formula ⇀ Formula,Formula] ⇒ Formula"
where
    "trafoX 0 rmap form = form"
  | "trafoX (Suc n) rmap form =
       trafo rmap (trafoX n rmap form)"
```

Now we obtain the desired result $\neg USA \wedge Comfort$ by running the following code.

```
lemma "trafoX (Suc 1) rmap_2
       (Neg (And (% RainSensor) (% Radar)))
       = ?Y";
apply (unfold rmap_2_def);
apply (rule trans)
apply simp
apply (rule sym)
apply (rule de_Morgan_conj)
done
```