
Draft pre-submission version (with author’s formatting)

This version is available at: https://eprints.mdx.ac.uk/6849/

Copyright:

Middlesex University Research Repository makes the University's research available electronically. Copyright and moral rights to this work are retained by the author and/or other copyright owners unless otherwise stated. The work is supplied on the understanding that any use for commercial gain is strictly forbidden. A copy may be downloaded for personal, non-commercial, research or study without prior permission and without charge.

Works, including theses and research projects, may not be reproduced in any format or medium, or extensive quotations taken from them, or their content changed in any way, without first obtaining permission in writing from the copyright holder(s). They may not be sold or exploited commercially in any format or medium without the prior written permission of the copyright holder(s).

Full bibliographic details must be given when referring to, or quoting from full items including the author’s name, the title of the work, publication details where relevant (place, publisher, date), pagination, and for theses or dissertations the awarding institution, the degree type awarded, and the date of the award.

If you believe that any material held in the repository infringes copyright law, please contact the Repository Team at Middlesex University via the following email address:

eprints@mdx.ac.uk

The item will be removed from the repository while any claim is being investigated.

See also repository copyright: re-use policy: http://eprints.mdx.ac.uk/policies.html#copy
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 ∈ N₀ ; m ≥ 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→FM_TRGT (dashed arrow). Thus, it is possible to derive a configuration CF_FM_TRGT of FM_TRGT from any given configuration CF_FM_SRC of FM_SRC (solid arrow).

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.

Figure 1: Relations defined by a configuration link.
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 $C_{FM}$ 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

\[ CL_{FM_{src} \rightarrow FM_{trgt}} : C_{FM_{src}} \rightarrow C_{FM_{trgt}} \]

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

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

\[ CL_{FM_{src} \rightarrow FM_{trgt}} := \{ CL_{FM_{src} \rightarrow FM_{trgt}}(c) \mid c \in C_{FM_{src}} \} \]

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

\[ C_{FM'} \subseteq C_{FM} \]

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 $C_{FM}^{L}$ as the set of all valid configurations of $FM$ that comply to the constraints in $L$. The above proposition can now be rephrased as

\[ C_{FM}^{L} \subseteq C_{FM} \]

We cannot be sure that $C_{FM}^{L}$ 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

\[ CL_{FM_{src} \rightarrow FM_{trgt}}(C_{FM_{src}}^{L'}) \subseteq CL_{FM_{src} \rightarrow FM_{trgt}} \cap C_{FM_{trgt}}^{L'} \]

Definition 1. Given a configuration link $CL_{FM_{src} \rightarrow FM_{trgt}}$ 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 $C_{FM_{src}}$ 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.
In the first step, we express one or more feature links of the source feature model. This deviated formula then constitutes the propagated feature link. 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 into another logical formula over the features of a model. Then, we will introduce transformation on “needs” and “excludes” feature links. The first step of our approach is that we express feature links by propositional logical formulae. Let formula \( \phi_{FM_{src}} \) be 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.

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: \( CL_{FM_{src} \rightarrow FM_{trgt}} \subseteq \{ \{ f_1 \rightarrow f_2 \mid f_1 \in F_{src}, f_2 \in F_{trgt} \cup \{ \neg f \mid f \in F_{trgt} \} \} \}. \) 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 \( rmap_{CL} : F_{trgt} \rightarrow F_{src} \) as defined in the following, where \( \Phi_{src} \) is the set of propositional logical formulae over \( F_{src} \) with the reduced set \( \{ \land, \lor, \neg \} \) of logical connectives.

(1) For all features \( f \in F_{trgt} \) with \( \exists f' \in F_{src} : (f \leftrightarrow \neg f') \in CL \) we define \( rmap_{CL}(f) = f_1 \lor f_2 \lor \ldots \)

for all \( f_1, f_2, \ldots \in \{ f \in F_{src} \mid (f \leftrightarrow f) \in CL \} \).

(2) For all features \( f_1 \in F_{trgt} \) with \( \exists f, f' \in F_{src} : (f \leftrightarrow f) \in CL \land (f' \leftrightarrow \neg f) \in CL \) we define \( rmap_{CL}(f) = (f_1 \lor f_2 \lor \ldots) \land (\neg f_1 \land \neg f_2 \land \ldots) \)

for all \( f_1, f_2, \ldots \in \{ f \in F_{src} \mid (f \leftrightarrow f_1) \in CL \) and \( f_1, f_2, \ldots \in \{ f \in F_{src} \mid (f \leftrightarrow f) \in CL \} \).

(3) For all features \( f \in F_{trgt} \) with \( \exists f' \in F_{src} : (f \leftrightarrow f') \in CL \) the reverse mapping \( rmap_{CL}(f) \) 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 \((\ldots \mapsto \neg f)\) 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 \(f_1 \in F_{\text{src}}\), there is both a configuration decision \((f_1 \mapsto f_1)\) (include) and another \((f_2 \mapsto \neg f_2)\) (exclude), we can combine both decisions and obtain \(((f_1 \wedge \neg f_2) \mapsto f_1)\) and \((f_2 \mapsto \neg f_2)\). So, if feature \(f_1\) is selected, we can conclude that \((f_1 \wedge \neg f_2) \vee \neg f_2\) holds in the source feature model. This formula can be simplified to \(\neg f_2\), corresponding to our reverse mapping. If more source features \(f_1, f_2, \ldots \in F_{\text{src}}\) exclude a feature \(f_1 \in F_{\text{trgt}}\) and \(f_1\) is selected on the target side, we can conclude that the features \(f_1, f_2, \ldots \) were deselected in the source configuration.

Therefore, we defined our reverse mapping as conjunction \(\neg f_1 \wedge \neg f_2 \wedge \ldots\) (in this case). For the target features which are not excluded by a source feature (all \(f_1 \in F_{\text{src}}\) with \(\exists f \in F_{\text{src}} : (f_1, f) \in \text{CL}\)) we defined the reverse mapping as disjunction of all source features \(f_1, f_2, \ldots \in F_{\text{src}}\) which are mapped to \(f_1\) \(((f_1 \mapsto f_1), (f_2 \mapsto f_1), \ldots\) since we can only conclude that at least one of the source features was selected if the target feature \(f_1\) 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 \(\text{CL}\) and a reverse mapping \(\text{rmap}_{\text{CL}}\) as defined in Def. 2 and 3. Then a transformation w.r.t. a reverse mapping \(\text{rmap}_{\text{CL}}\) is recursively defined as \(\text{trafo}_{\text{rmap}_{\text{CL}}} : \Phi_{\text{trgt}} \rightarrow \Phi_{\text{src}}\) with

\[
\text{trafo}_{\text{rmap}_{\text{CL}}} (\phi_t) = \begin{cases} \\
\text{rmap}_{\text{CL}} (\phi_s) & \text{if } \phi_s \in F_{\text{src}} \text{ and } \text{rmap}_{\text{CL}} (\phi_s) \text{ is defined} \\
\bot & \text{if } \phi_s \in F_{\text{src}} \text{ and } \text{rmap}_{\text{CL}} (\phi_s) \text{ is not defined} \\
\top & \text{if } \phi_s \equiv \neg \psi_t \text{ and } \text{rmap}_{\text{CL}} (\psi_t) \text{ is not defined} \\
\text{rmap}_{\text{CL}} (\phi_t') \wedge \text{trafo}_{\text{rmap}_{\text{CL}}} (\phi_t') & \text{if } \phi_s \equiv \phi_t' \wedge \phi_t'' \text{ and } \phi_t', \phi_t'' \in F_{\text{trgt}} \\
\text{rmap}_{\text{CL}} (\phi_t') \vee \text{trafo}_{\text{rmap}_{\text{CL}}} (\phi_t') & \text{if } \phi_s \equiv \phi_t' \wedge \phi_t'' \text{ and } \phi_t', \phi_t'' \in F_{\text{trgt}} 
\end{cases}
\]

where \(\Phi_{\text{src}}\) and \(\Phi_{\text{trgt}}\) are the sets of propositional logic formulae over \(F_{\text{src}}\) resp. \(F_{\text{trgt}}\), with the reduced set \(\{-, \wedge, \vee\}\) of logical connectives. Note that \(\Phi_{\text{src}}^+\) additionally contains formulae with undefined features \((\bot)\). Undefined features are required since the transformation is total, although the reverse mapping is partial.

With this definition, we obtain a formula \(\phi_s \in \Phi_{\text{src}}^+\) over the features \(F_{\text{src}}\) of the source model \(FM_{\text{src}}\). 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 hierarchically decomposed 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 \(\text{trafo}\) 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


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 formulae to be transformed. Many fragments of these parts could be generated fully automated by a corresponding feature modeling tool, for example CVM (see [Re09]) – 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. 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> = ??";
  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 → RainSensor, USA → Radar) and obtain the following reverse mapping.

```
constdefs rmap_1 :: "Formula → Formula"
  "rmap_1 = [\% RainSensor → % Comfort,
             % Radar → % USA]"
```

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) ∧ (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).

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 formulae to be transformed. Many fragments of these parts could be generated fully automated by a corresponding feature modeling tool, for example CVM (see [Re09]) – 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. 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> = ??";
  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 → RainSensor, USA → Radar) and obtain the following reverse mapping.

```
constdefs rmap_1 :: "Formula → Formula"
  "rmap_1 = [\% RainSensor → % Comfort,
             % Radar → % USA]"
```

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) ∧ (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> = ??";
  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.
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 \( \neg (\text{RainSensor} \land \text{Radar}) \). Now we define a lemma in Isabelle and use the simplifier for applying our transformation.

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

The result of this transformation delivers the estimated result \( \neg \text{Comfort} \lor \neg \text{USA} \). For a better understandability, this can be rearranged to \( \neg (\text{Comfort} \land \text{USA}) \). This step can be done manually or it can also be implemented in Isabelle in the following way:

```isabelle
lemma "trafo rmap_1
  (Neg (And (% RainSensor) (% Radar)))
= ??";
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 already 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, we did not define an interface between our 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 \( (\text{AdaptiveCC} \rightarrow \rightarrow \text{Radar}) \) (in reverse) to our mapping function.

```isabelle
constdefs rmap_2 :: "Formula => Formula"
"rmap_2 [\%
  RainSensor \rightarrow \rightarrow%
  Comfort,
  \%
  AdaptiveCC \rightarrow \rightarrow%
  USA,
  \%
  Radar \rightarrow \rightarrow%
  AdaptiveCC]"
```

Applying our transformation in the same way as described in the first example now leads to the constraint \( \neg \text{AdaptiveCC} \land \text{Comfort} \). This is, of course, correct but we could apply our mapping again and would obtain \( \neg \text{USA} \land \text{Comfort} \). For automating this, we formulated another transformation, that applies the mapping recursively as often as possible.

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

Now we obtain the desired result \( \neg \text{USA} \land \text{Comfort} \) by running the following code.

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