INTRODUCTION TO FEATURE MODELING WITH GLENCOE

Glencoe is an easy to use web application which enables you to specify, visualize and analyze multivariant product lines modeled as feature models.

Components of a Feature Model

This section gives a brief overview on the basics of feature modeling. Readers experienced with feature models can skip to the next section.

Features

Feature models are an industry-approved approach to represent product lines in a standardized way. A product line groups all variants of a product (also known as *buildable configurations* or in short *products*) by describing common features (which are part of every product) as well as individual features (which constitute the variance). A feature is a distinguishable characteristic of a product line that is relevant to some of its stakeholders. This may either be a physical component or a property which a product does or does not have.

Within a feature model the features are arranged hierarchically by linking each feature to exactly one *parent feature*. Each feature is either *mandatory* or *optional*:

- **Mandatory.** A mandatory feature has to be included in every product its parent feature is part of.
- **Optional.** An optional feature may but need not be included in a product its parent feature is part of.

The root feature (i.e. is the uppermost feature representing the complete product) is implicitly mandatory and has no parent feature.

Furthermore, a feature may be of one of the following types:

- **Alternative.** Exactly one of the child features has to be included in every product the parent feature is part of.
- **Selection.** At least one of the child features has to be included in every product the parent feature is part of.
- **Selection with cardinalities *min* and *max*.** At least *min* and at most *max* child features have to be included in every product the parent feature is part of.

Constraints

In addition to the parent-child-hierarchy a feature model can be refined with another type of relationship, the *cross-tree constraints* (or in short *constraints*). In general, constraints provide a possibility to declare properties of the overall product line each buildable product has to fulfill. These properties are expressed as logical connections between arbitrary features regardless of their location within the model.

The most common types of constraints are:

- **Requirement.** A feature f1 requires a feature f2 if f2 has to be included in every product f1 is part of.
- **Mutual Exclusion.** A feature f1 excludes a feature f2 if both features can never be part of the same product.

Besides those *simple constraints* Glencoe supports modeling of more complex constraints between two or even more arbitrary features, called *advanced constraints*. An advanced constraint is expressed by a propositional logic formula following an extended variant of the *SPASS Input Syntax*[1].

A complete coverage of propositional logic is beyond the scope of this tutorial. Therefore, here are just a few examples:

- `"f1"`
  Each product has to include feature f1.
- `and("f1", "f2", "f3")`
  Each product has to include all of the features f1, f2 and f3.
- `or("f1", "f2", "f3")`
  Each product has to include at least one of the features f1, f2 and f3.
- `equiv("f1", "f2")`
  Each product including feature f1 has to include feature f2 and each product including f2 has to include f1.
- `implies("f1", not("f2"))`
  Each product including feature f1 must not include feature f2, but a product including f2 may include f1.
- `implies("f1", xor("f2", "f3"))`
  Each product including feature f1 has to include either feature f2 or feature f3, but never both of them.

The formal definition of allowed formulae is given as follows:

```
TERM ::= "FEATURE_NAME" |
         not(TERM) |
         equiv(TERM, TERM) |
         excludes(TERM, TERM) |
         implies(TERM, TERM) |
         and(TERM [, TERM]+) |
         or(TERM [, TERM]+) |
         xor(TERM [, TERM]+)
```

Conjectures

Like constraints, conjectures describe properties of the overall product line and are expressed

[1] See Section 5 of https://www.spass-prover.org/download/binaries/spass-input-syntax15.pdf

the same way as advanced constraints (i.e. a propositional logic formula).

But unlike constraints, conjectures do not demand these properties from the buildable products and therefore never affect the consistency of the feature model. They are just assumptions that can be confirmed (i.e. each product fulfills the property) or rejected (i.e. some products do not fulfill the property) during analysis.

Configurations

A configuration of a feature model is described by a set of features that a product should contain and another set of features that a product should not contain. Therefore, it may represent a single specific product or a subset of all products.

Each configuration has a status. The possible states are:

- **Incomplete.** Some features may have been decided but there are other features that must be decided too in order to make the configuration buildable. The configuration represents a subset of all products.
- **Buildable.** All features that have to be decided are decided. There may be undecided features but they are optional. The configuration represents a single product.

Analyses on Feature Models

Each of following analyses available in Glencoe takes a feature model (including its features and constraints) as input.

- **Consistency.** This operation returns whether the feature model is consistent or not. A model is consistent if it represents at least one product.
- **Common Features.** This operation returns the set of common features of the model. A feature is common if it is part of every product of the product line.
- **False Optional Features.** This operation returns the set of false optional features of the model. A feature is false optional if it is part of every product of the product line despite being modeled as an optional feature. Therefore, the false optional features are always a subset of the common features.
- **Unique Features.** This operation returns the set of unique features of the model. A feature is unique if it is part of exactly one product of the product line.
- **Dead Features.** This operation returns the set of dead features of the model. A feature is dead if it is not part of any product of the product line.
- **Atomic Sets.** This operation returns disjoint sets of features that always appear together in products of the product line.
- **Conjectures.** This operation returns the subset of all conjectures that are fulfilled by all products.
- **Possible Configurations.** This operation returns the total number of products in the product line.
- **Commonalities.** For each feature this operation returns the ratio between the number of

possible configurations the feature is part of and the total number of possible configurations.

Using the Feature Model Editor

This section describes the basic steps to create a new feature model from scratch using Glencoe's editor. This example introduces a simple product line specifying all variants of a muesli.

First of all, open your web browser and visit https://glencoe.hochschule-trier.de/webapp.

Creating a Feature Model

To create a new feature model, click the + button on the left side of the feature model tab-bar. A dialog in which you can type in the name of the model pops up. Type in *Muesli* and click the *Create* button or press the enter key (see Figure 1).

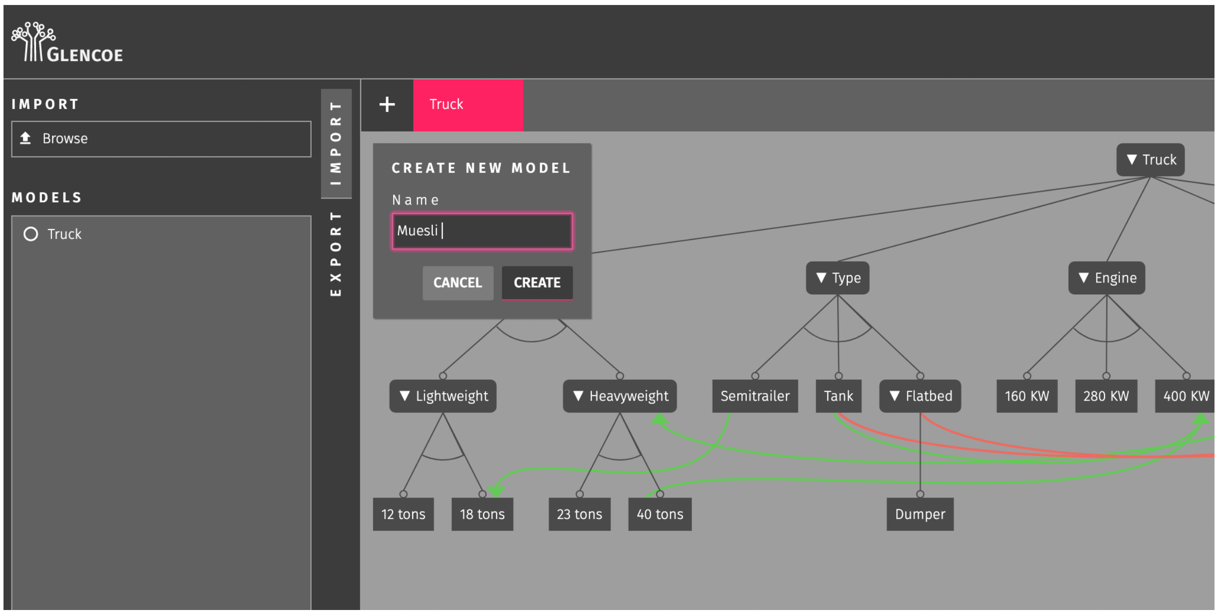


Figure 1: Creating a feature model

The new model initially consisting of only a root feature with the same name as the feature model will be displayed. Please note that the feature model as well as the root feature can be renamed independently from each other later on.

Renaming a Feature Model

Ensure the topmost tab *Import* on the left side-bar is selected and double-click on the name of the model within the list of models (see Figure 2). Type in the new name of the model and press the enter key or click anywhere else within the browser window to confirm.

Adding Child Features

Right-click on the root feature *Muesli* to open the context menu and select *Add child feature* (see Figure 3).

In the following dialog type in the name *Basis*, uncheck the *Optional* checkbox (since every muesli must have a base cereal) and click the *Create* button (see Figure 4).

A new child feature appears below the root feature.

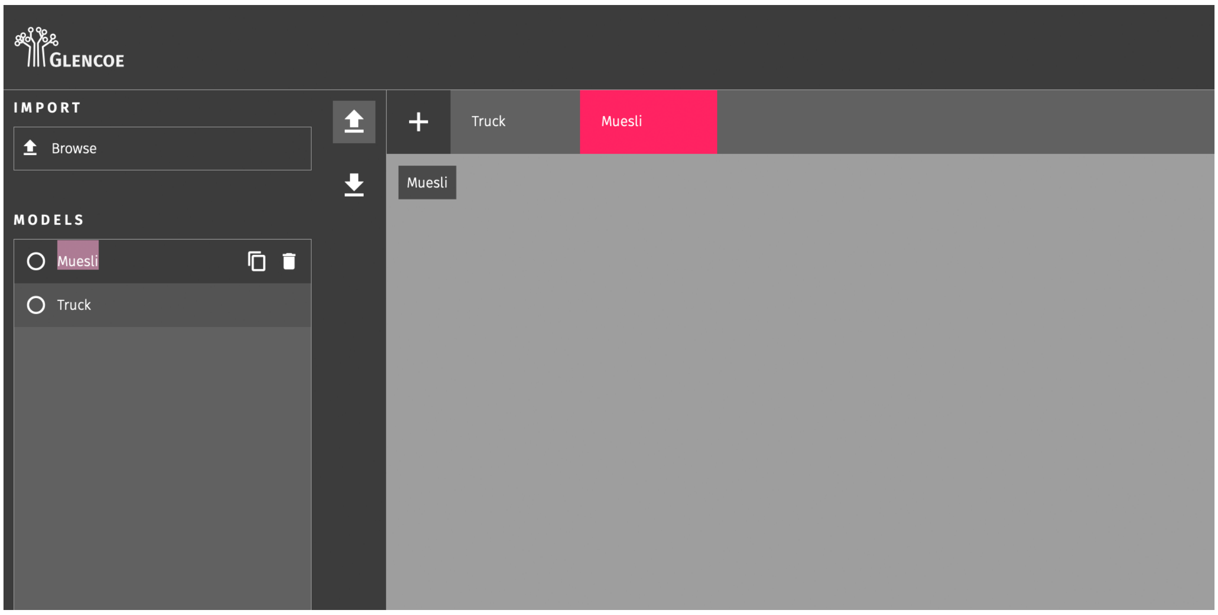


Figure 2: Renaming a feature model

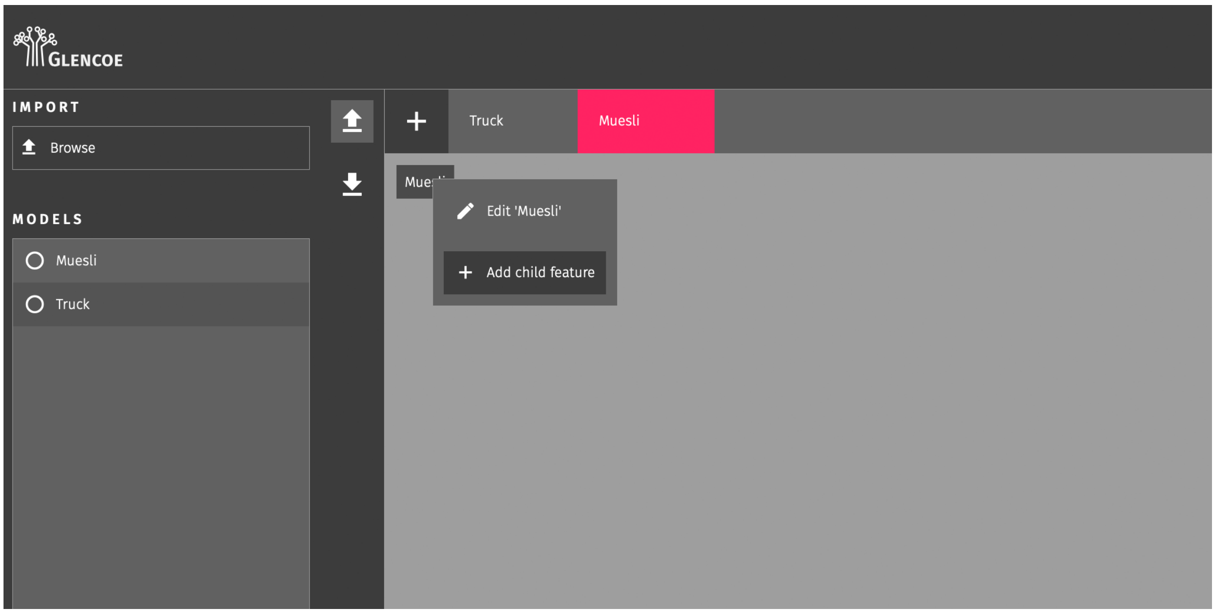


Figure 3: Context menu of the root feature

Adding an Alternative

Add three optional child features to the feature *Basis* with the names *Flakes*, *Oats* and *Quinoa*, representing a set of base cereals.

We decide that every muesli must have exactly one base cereal. Mixed cereals are not allowed.

Therefore, right-click on the feature *Basis* to open the context menu and select *Edit 'Basis'*. Change the type to *Alternative* and click the *Update* button (see Figure 5).

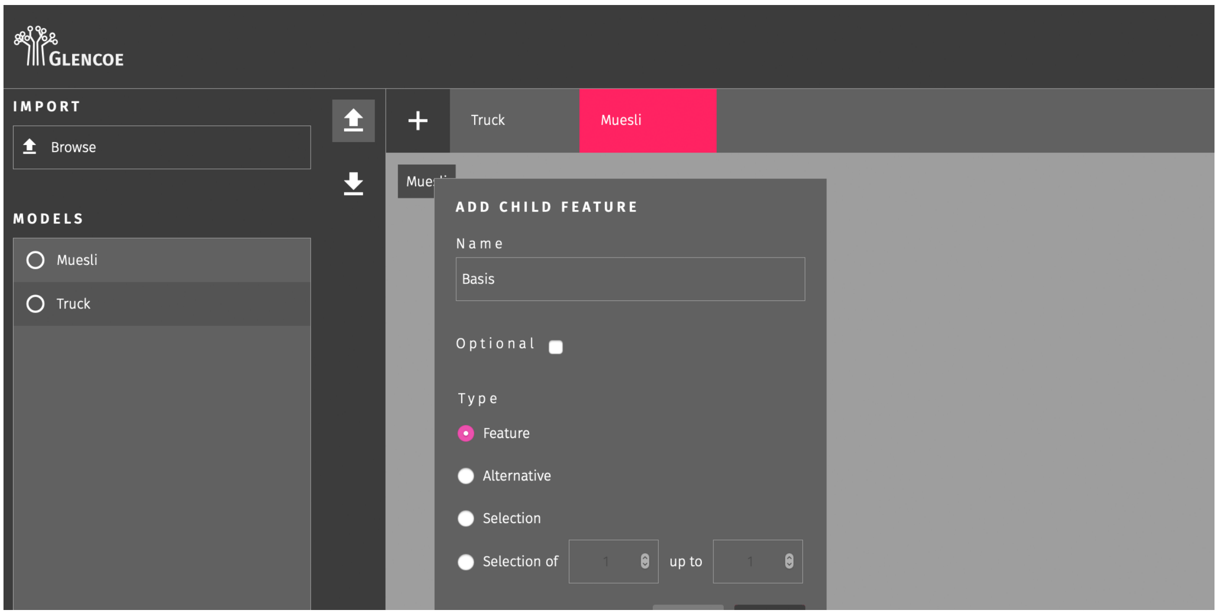


Figure 4: Adding a first child feature

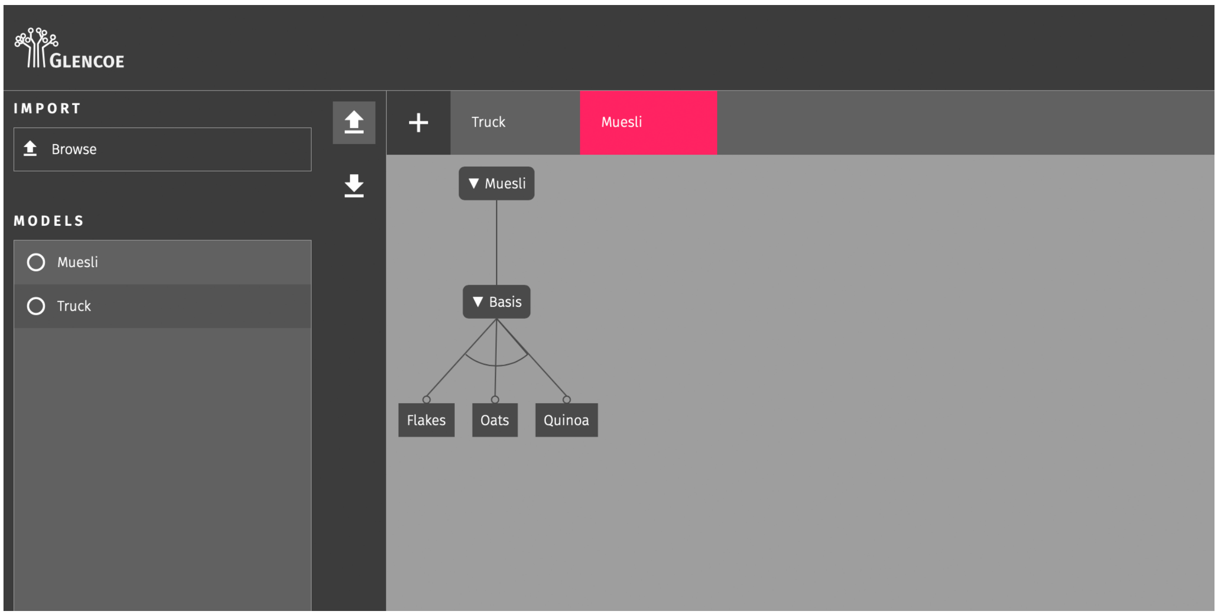


Figure 5: Adding an alternative

Adding a Selection

A muesli may but need not contain one or more types of fruits.

Therefore, add a second child feature *Fruits* to the root feature *Muesli*. Unlike *Basis* this feature is optional (since a muesli does not need to contain fruits at all). Set its type to *Selection* (since it is allowed to add one or more fruits).

Add three optional child features to the feature *Fruits* with the names *Apple*, *Banana* and *Strawberry* (see Figure 6).

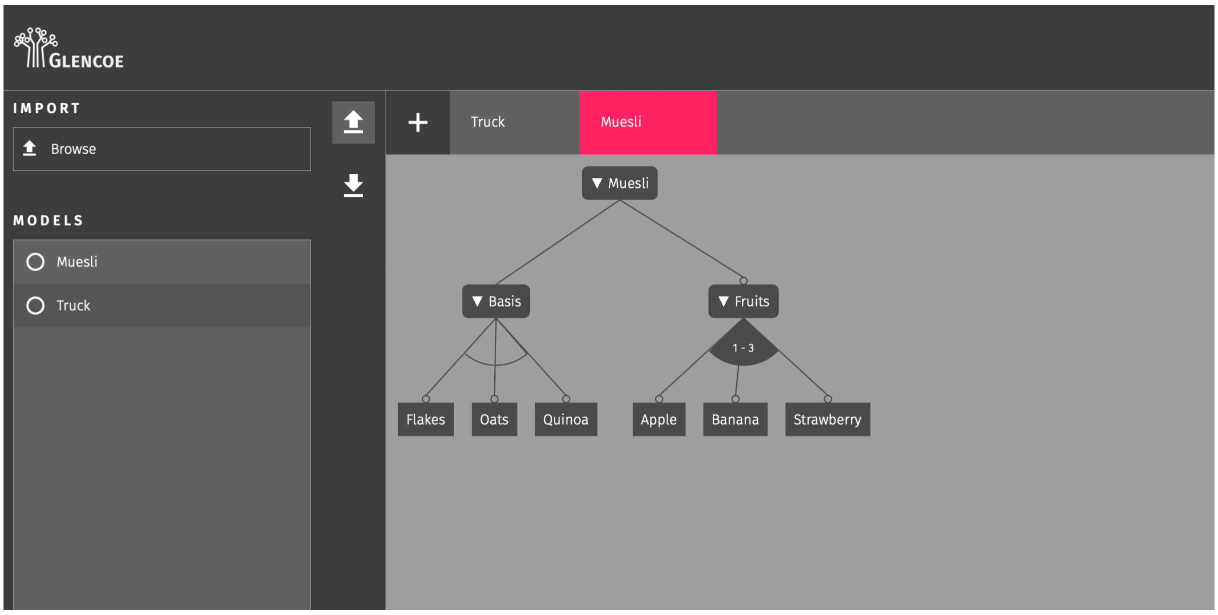


Figure 6: Adding a selection

Adding a Mutual Exclusion

Add two more optional child features to the root feature with the names *Chocolate rasps* and *Lactose-free*.

While the feature *Chocolate rasps* represents a physical component of the product, the feature *Lactose-free* represents a property which a product has or has not.

Right-click on the feature *Chocolate rasps* to open the context menu and select *Add mutual exclusion*.

Select the feature *Lactose-free* using the second combo-box (since chocolate rasps contain milk and therefore cannot be lactose-free) and click the *Create* button (see Figure 7).

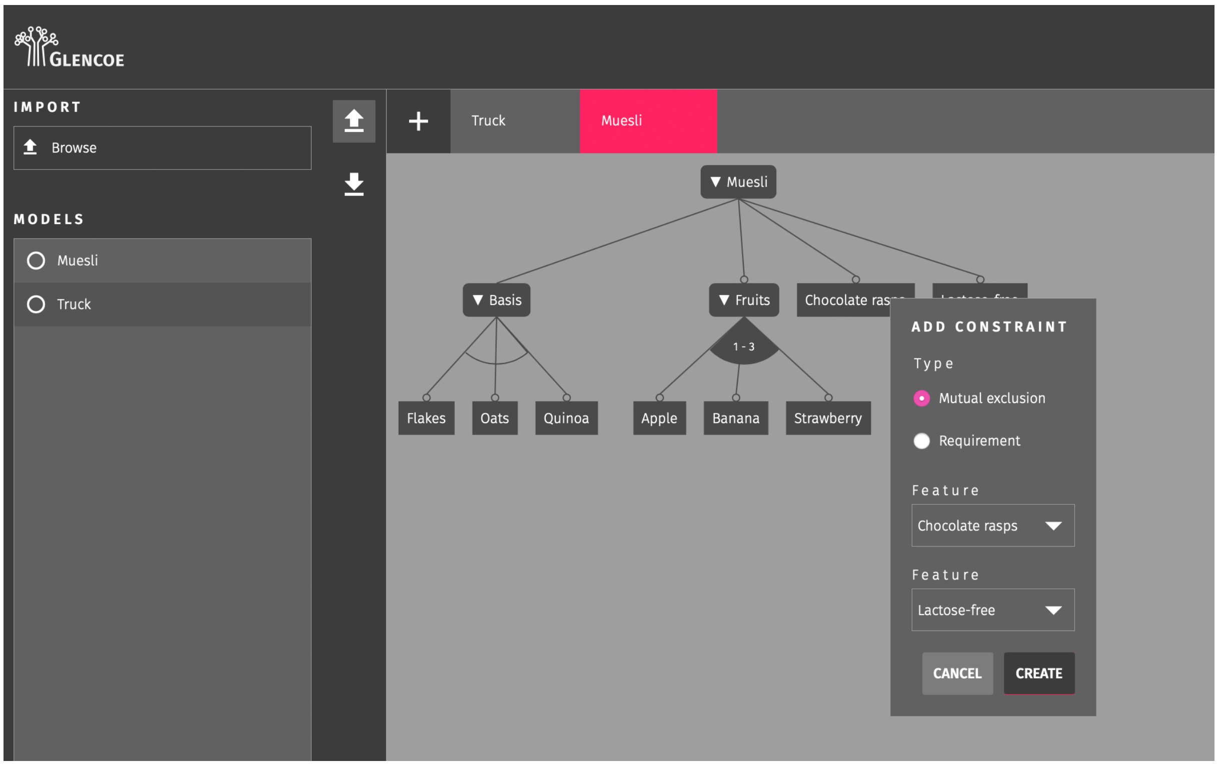
Adding a Requirement

Within the set of base cereals only oats are lactose-free. This could be modeled using mutual exclusions between every other base cereal and the feature *Lactose-free* or much easier by a single requirement.

Right-click on the feature *Lactose-free* to open the context menu and select *Add requirement*. Alternatively, you can click the *Create simple constraint* button on the tab *Constraints* on the right side-bar.

Select the feature *Oats* using the second combo-box and click the *Create* button.

Please note the direction of this constraint visualized by an arrow tip (see Figure 8). Here, the muesli can only be lactose-free if the base cereal are oats. But there still can be mueslis which contain oats but are not lactose-free. Unlike in a mutual exclusion (where the order of the two involved features is irrelevant) a requirement's direction determines different meanings.



ADD CONSTRAINT

Type

[✓ Mutual exclusion]

Requirement

Feature: Chocolate rasps

Feature: Lactose-free

CANCEL CREATE

Figure 7: Adding a mutual exclusion

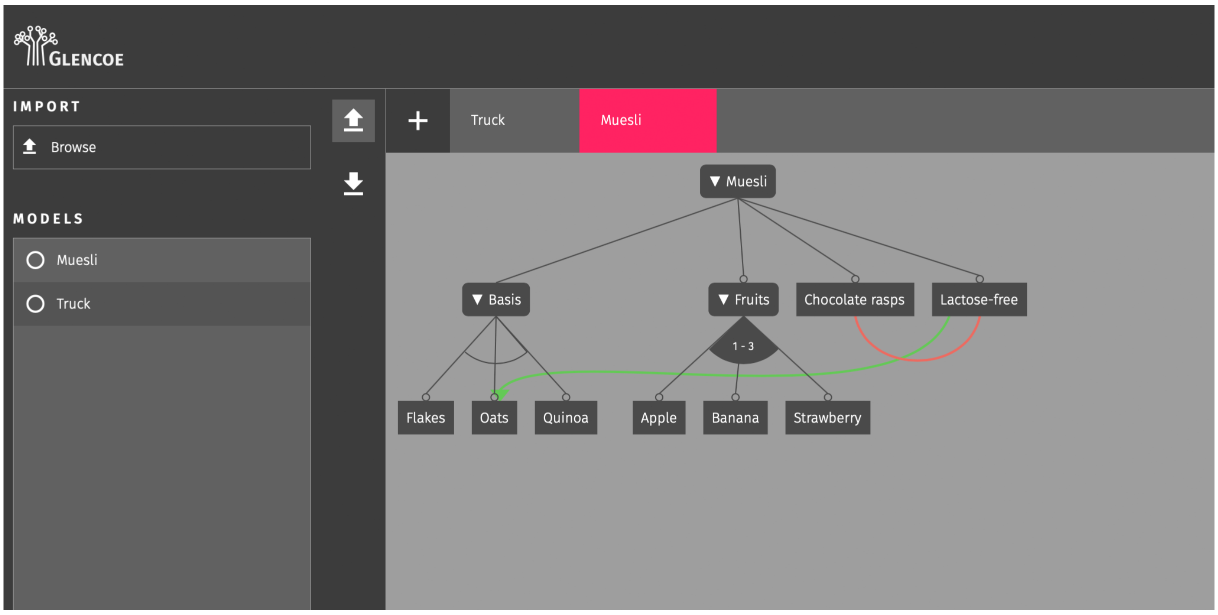


Figure 8: Adding a requirement

Exporting a Feature Model

Select the tab *Export* on the left side-bar and click the *GFM* button (see Figure 9).

GFM is Glencoe's default format for exporting feature models. It is based on the popular and open JSON data format. Please note that only files in *GFM* and *v.control* formats can be re-imported.

The feature model will be saved in your web browser's download directory.

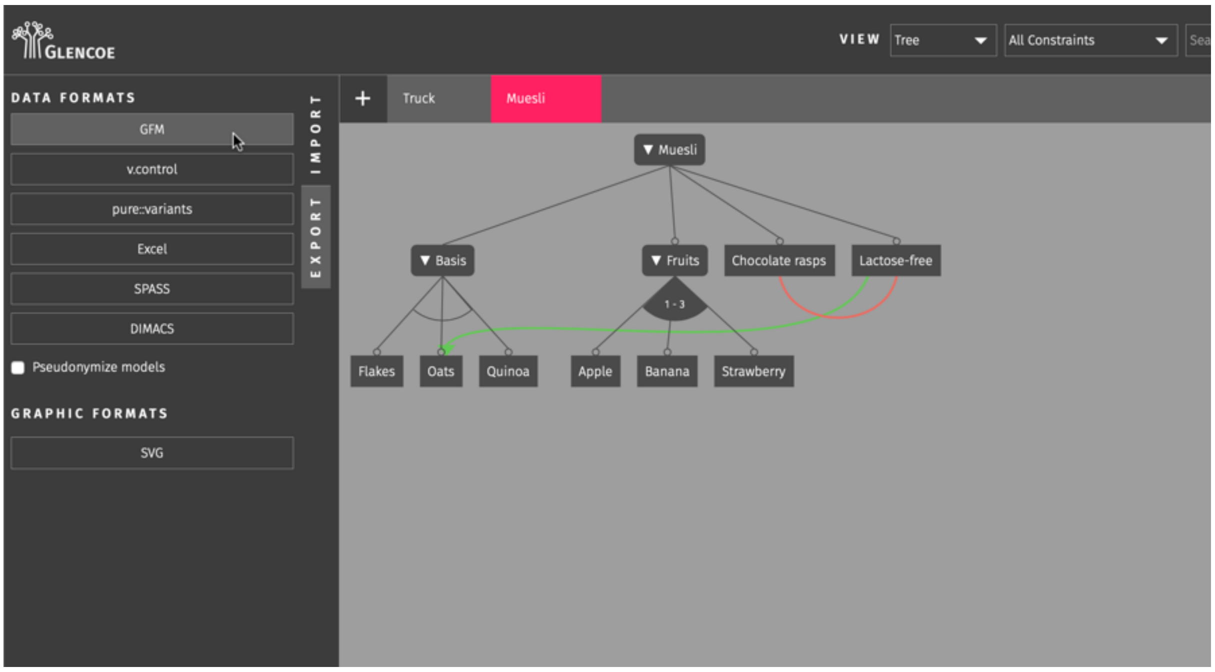
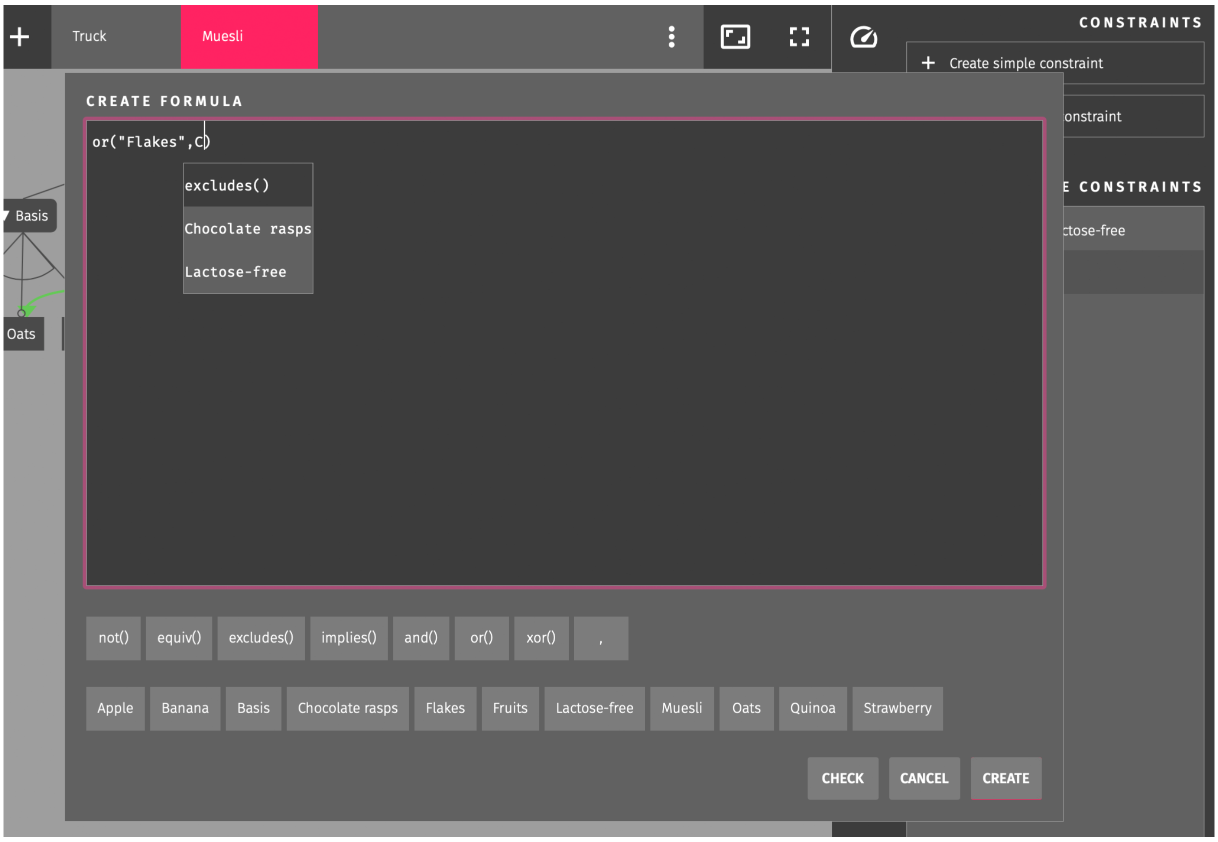


Figure 9: Exporting a feature model

Adding an Advanced Constraint

To add an advanced constraint, right-click any feature to open the context menu and select *Add advanced constraint*. Alternatively, you can click the *Create advanced constraint* button on the tab *Constraints* on the right side-bar. The following dialog displays an editor for logic formulae (see Figure 10).



```
or("Flakes",C)
```

Figure 10: Adding an advanced constraint

Enter a propositional logic formula describing the advanced constraint as explained in section *Constraints*. While typing a context-sensitive auto-completion will guide you.

Alternatively, clicking the buttons below the input area will insert the selected term or feature into the formula.

Click the *Create* button to validate and add the constraint. In case of a syntax error a toast message on the lower right corner of the browser window will display a hint to the erroneous part of the formula.

Editing a Constraint

Right-click one of the features which are involved in the mutual exclusion to open the context menu and select *Edit 'Chocolate rasps ↔ Lactose-free'* (see Figure 11).

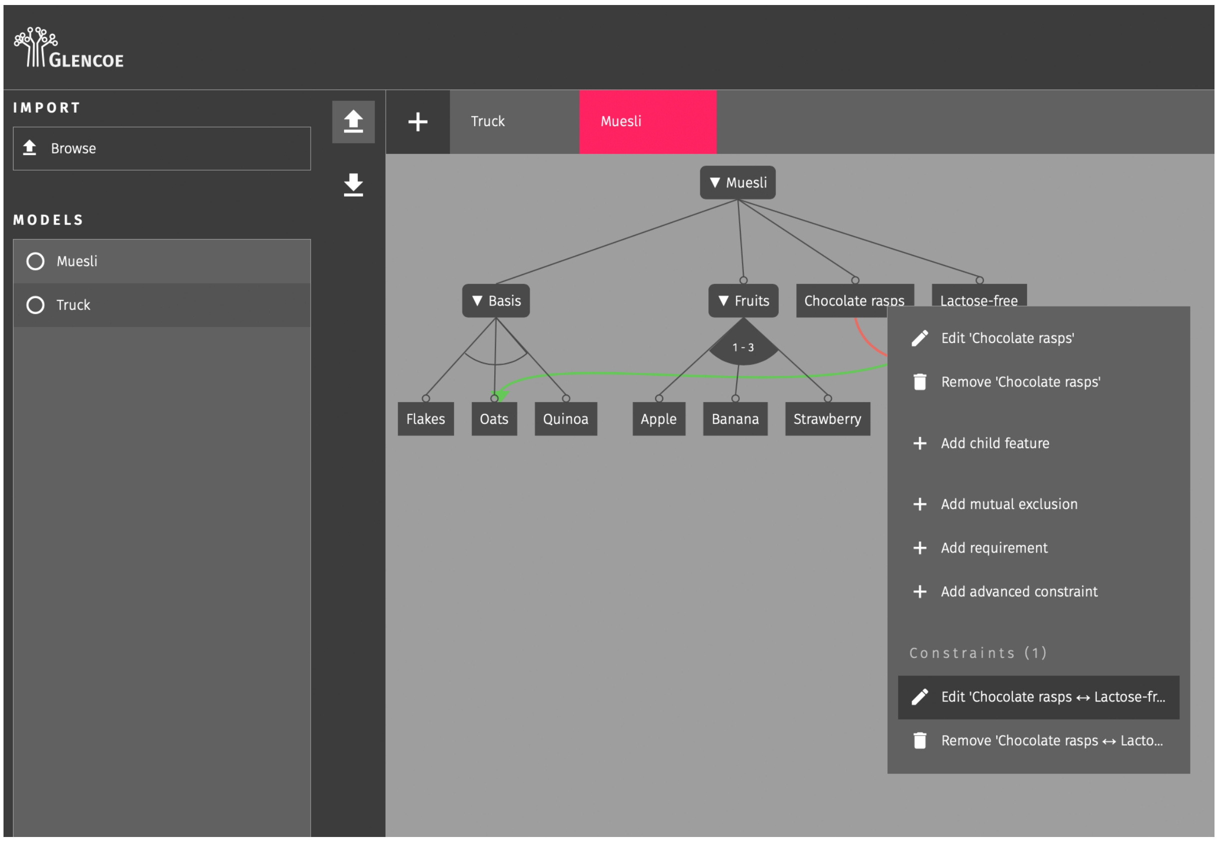


Figure 11: Editing a constraint

The dialog from Figure 7 allows you to change the constraint's type or the involved features. In case the selected constraint is an advanced one, the dialog from Figure 10 allows you to edit and update the formula.

Alternatively, it is possible to edit any constraint (simple constraints as well as advanced ones) using the tab *Constraints* on the right side-bar by double-clicking on the constraint within the list of constraints.

Removing a Constraint

Right-click one of the features which are involved in the mutual exclusion to open the context menu, select *Remove constraint* and then *Remove 'Chocolate rasps ↔ Lactose-free'* (see Figure

12).

Alternatively, it is possible to remove a constraint using the tab *Constraints* on the right side-bar. Hover the mouse over the constraint within the list of constraints and click the appearing trash can icon.

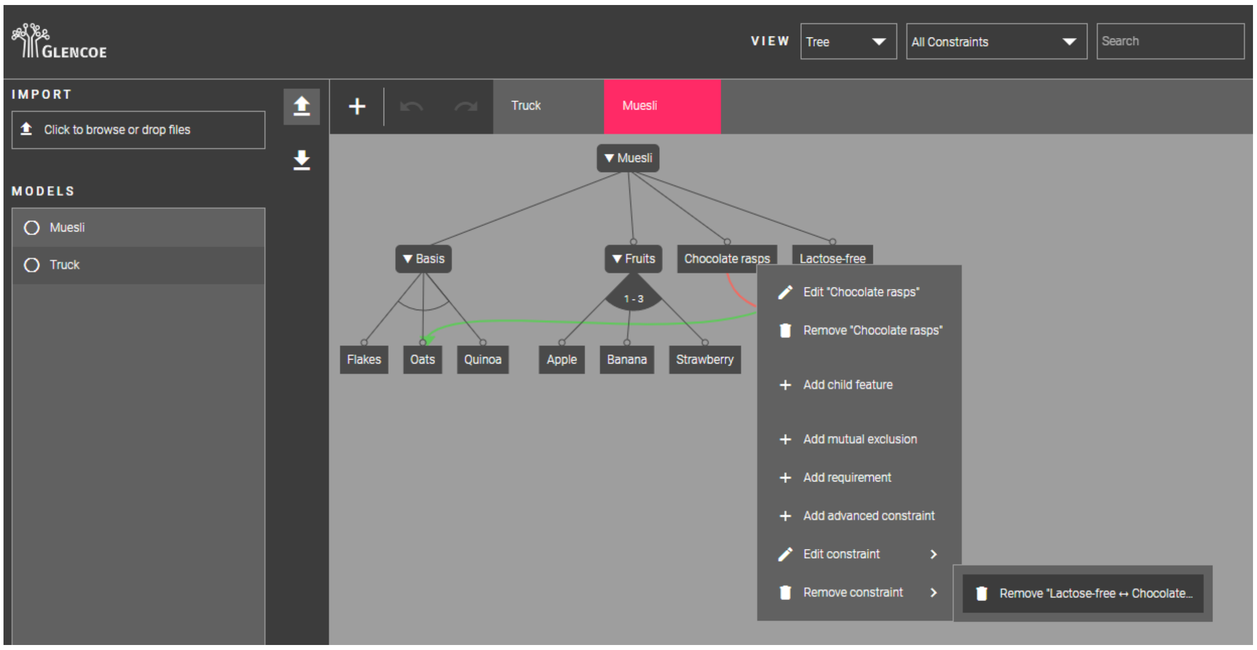


Figure 12: Removing a constraint

Removing a Feature

Right-click the feature *Basis* to open the context menu and select *Remove 'Basis'* (see Figure 13).

When a feature gets removed all child features are removed recursively as well as all constraints in which one of the removed features is involved. Please note that the root feature cannot be deleted.

Closing a Feature Model

Hover the mouse over the tab *Muesli* within the feature model tab-bar and click the appearing × icon (see Figure 14).

To re-open the feature model, ensure the tab *Import* on the left side-bar is selected and click the feature model within the list of models on the left side.

Removing a Feature Model

Ensure the tab *Import* on the left side-bar is selected. Hover the mouse over the model *Muesli* within the list of models on the left side and click the appearing trash can icon (see Figure 15).

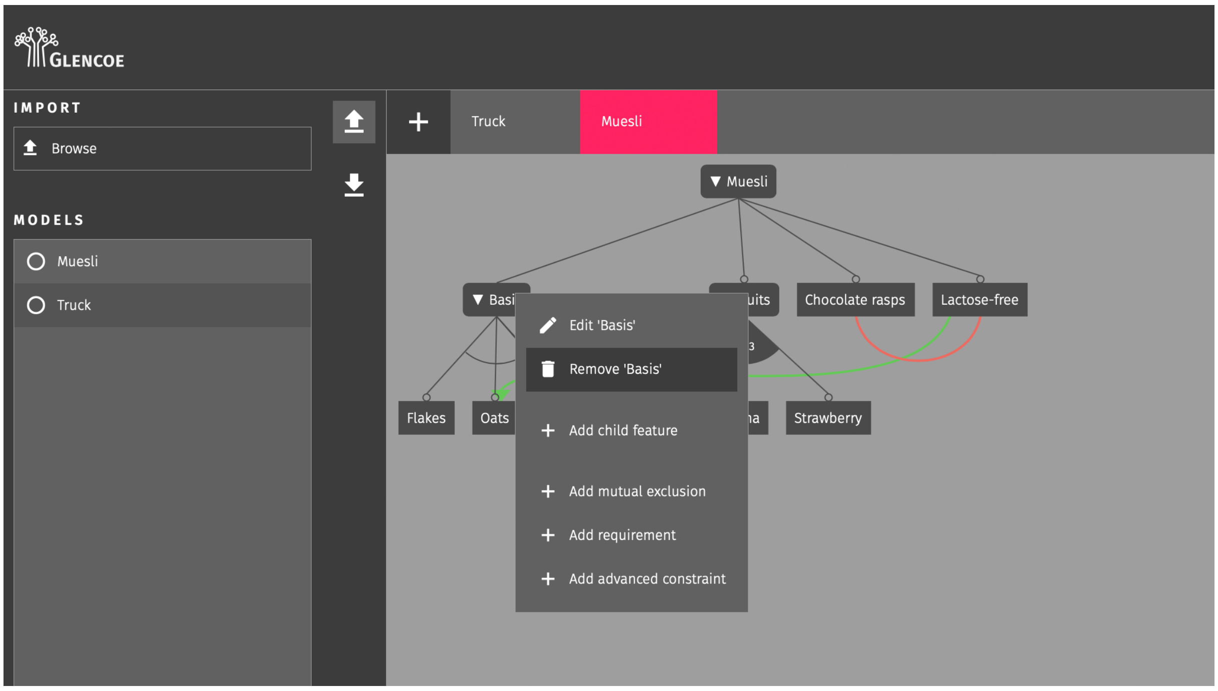


Figure 13: Removing a feature

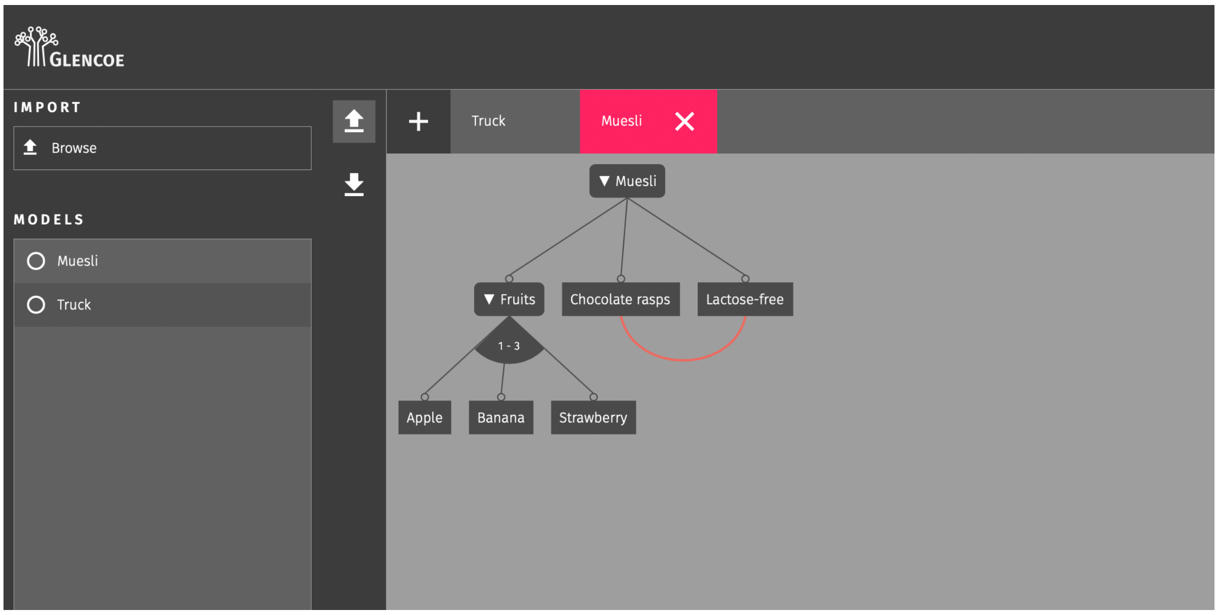


Figure 14: Closing a feature model

Importing a Feature Model

Besides creating a new feature model, it is possible to import a model from a (formerly exported) file.

Reset the application by refreshing the site within your web browser.

Ensure the tab *Import* on the left side-bar is selected and click the *Browse* button.

Navigate to the directory the exported feature model *Muesli* has been downloaded to, select the file *Muesli.gfm.json* and confirm with the *Open* button.

Click the imported feature model within the list of models on the left side (see Figure 16).

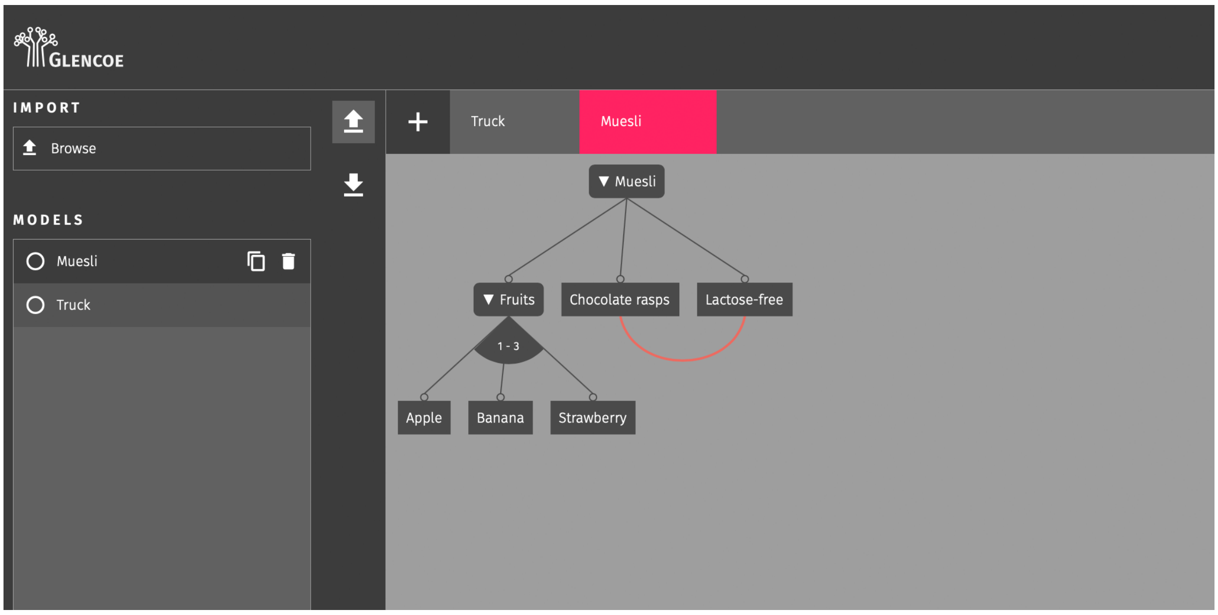


Figure 15: Removing a feature model

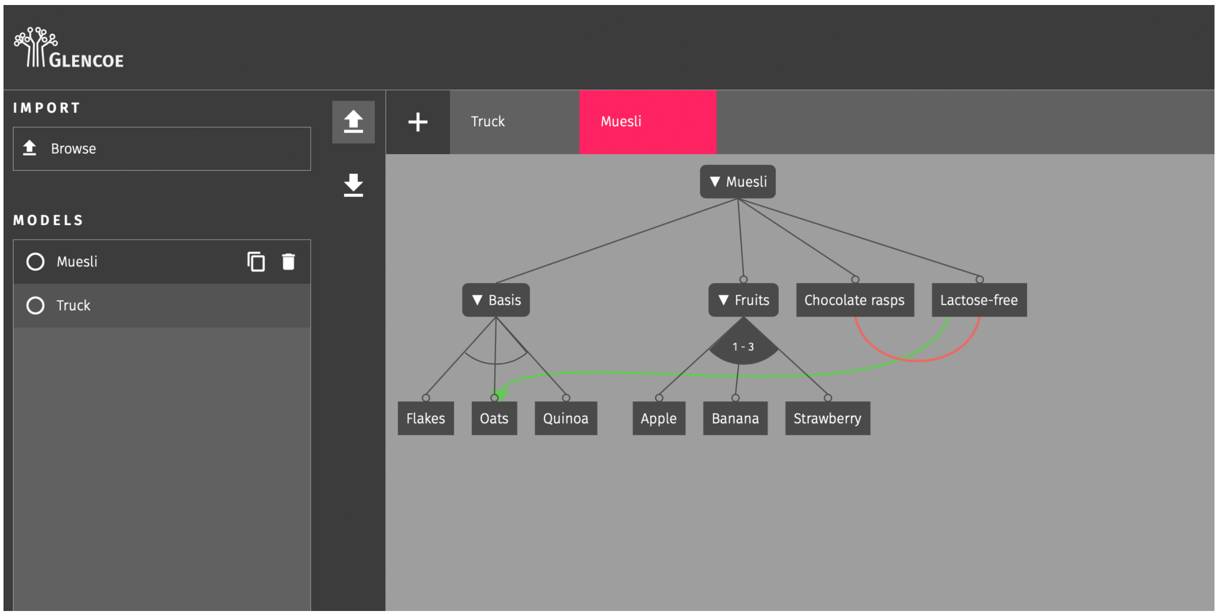


Figure 16: Importing a feature model

Adding a Conjecture

Since *Chocolate rasps* and *Lactose-free* are mutually exclusive and *Lactose-free* requires *Oats* some may be tempted to conclude that a muesli has to contain *Chocolate rasps* if it contains one of the other base cereals (*Flakes* or *Quinoa*).[2]

To formulate that assumption, select the tab *Conjecture* on the right side-bar and click the *Create conjecture* button (see Figure 18). The following dialog displays the editor for logic formulae (see Figure 10).

Enter the propositional logic formula describing the conjecture:

[2] This conjecture will be rejected because *Chocolate rasps* and *Lactose-free* are both optional. Not a single product has to contain one of them. Even a muesli with oats does not have to be lactose-free since the requirement is defined with opposite direction.

```
implies(or("Flakes","Quinoa"),"Chocolate rasps")
```

Click the *Create* button to validate and add the conjecture.

Repeat the steps above in order to create a second conjecture:

```
implies("Flakes",not(and("Oats","Quinoa")))
```

See section *Analyzing a Feature Model* to run an analysis that confirms or rejects these conjectures and to display proofs for these statements.

Editing a Conjecture

To edit any conjecture, open the tab *Conjectures* on the right side-bar and double-click on the conjecture. The dialog from Figure 10 allows you to edit and update the conjecture's formula.

Removing a Conjecture

To remove any conjecture, ensure the tab *Conjectures* on the right side-bar is selected. Hover the mouse over the conjecture to remove and click on the appearing trash can icon.

Analyzing a Feature Model

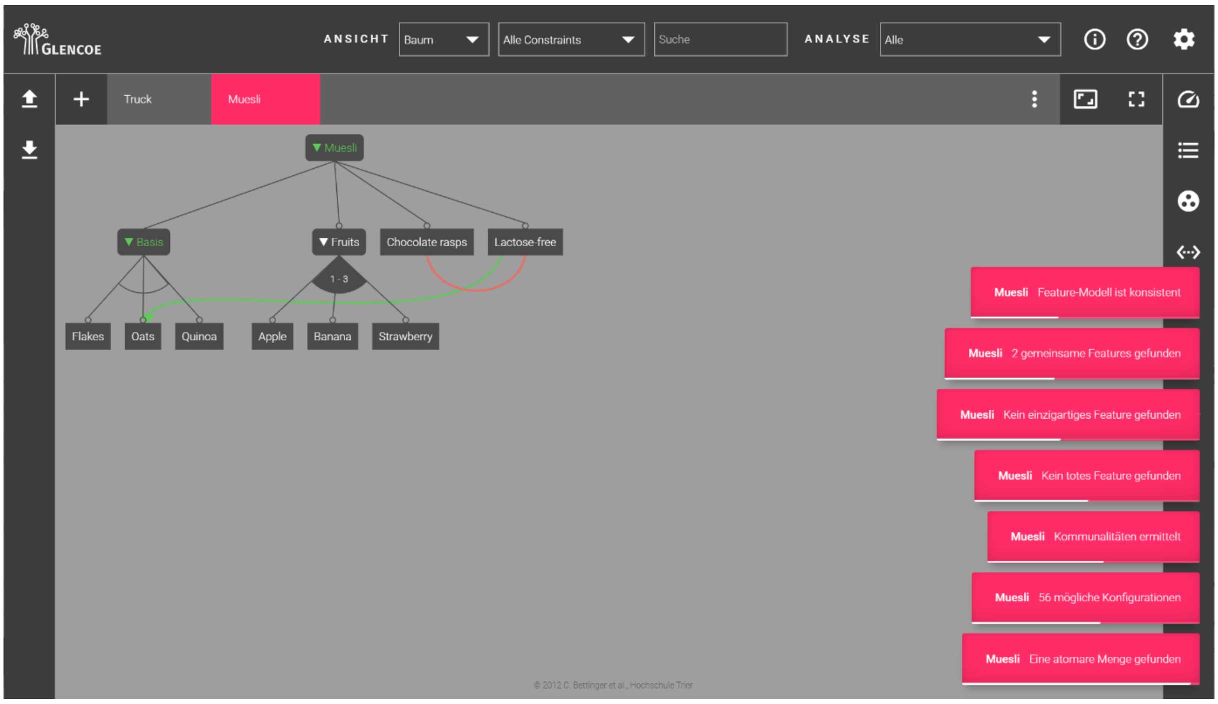
This section describes how to analyze a feature model using Glencoe's solver farm.

Running an Analysis

By opening the analysis combo-box within the top-bar you can see the set of provided analysis. Selecting an item sets the default analysis and runs it immediately. To run the default analysis again just click the combo-box.

Since the false optional features are always a subset of the common features the analysis *False Optional Features* is executed too whenever the common features are determined.

Select the item *All* to run all analyses in sequence with a single click (see Figure 17).


ANSICHT Baum · Alle Constraints · ANALYSE Alle

Muesli Feature-Modell ist konsistent

Muesli 2 gemeinsame Features gefunden

Muesli Kein einzigartiges Feature gefunden

Muesli Kein totes Feature gefunden

Muesli Kommunalitäten ermittelt

Muesli 56 mögliche Konfigurationen

Muesli Eine atomare Menge gefunden

Figure 17: Running all analyses

Visualization of Analysis Results

The result of every analysis is visualized in multiple ways:

First, the results are displayed in a toast message on the lower right corner of the browser window (see Figure 17). The toast message automatically disappears after a short while, but hovering the mouse over the message prevents it from disappearing.

The result of the analysis *Consistency* is also visualized by a circle in front of the model's name within the list of models on the left side:

- Green circle: The model is consistent.
- Red circle: The model is not consistent.
- Grey circle: The consistency has not been determined yet.

The result of the analysis *Conjectures* is also visualized by a circle in front of the conjecture's formula within the tab *Conjectures* on the right side (see Figure 18):

- Green circle: The conjecture is confirmed, i.e. each product fulfills this property.
- Red circle: The conjecture is rejected, i.e. some products do not fulfill this property.
- Grey circle: The conjecture has not been checked yet.

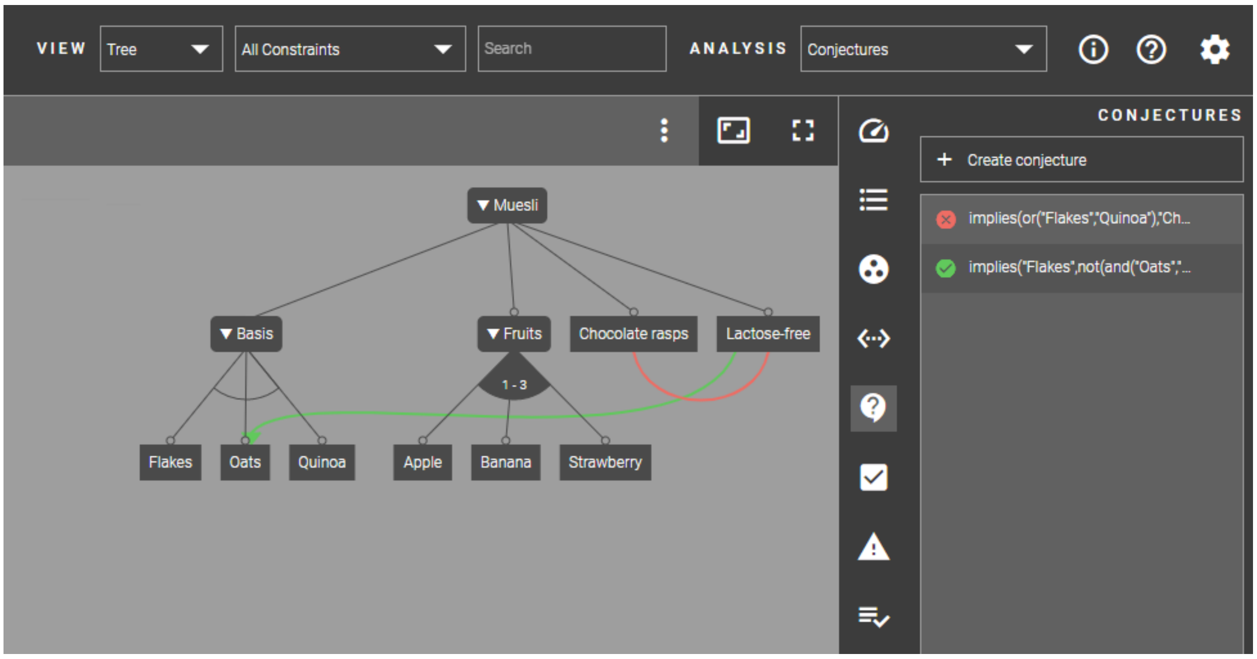


Figure 18: Results of conjecture analysis

After running the analyses *Common Features*, *Unique Features*, *Dead Features*, *Possible Configurations* or *Conjectures*, the number of common/false optional/unique/dead features, possible configurations or rejected conjectures, is displayed within the tab *Metrics* on the right side-bar (see Figure 22).

Within the main visualization of a feature model

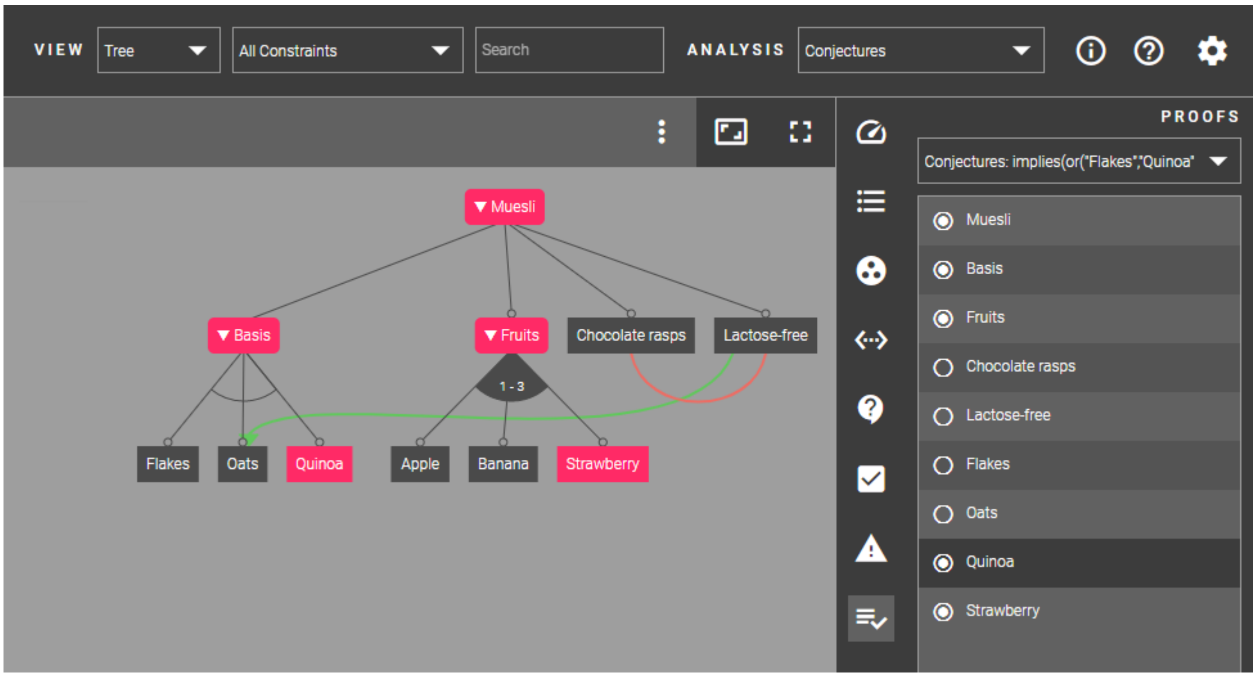
- all common features are highlighted by a green name,
- all unique features are highlighted by a yellow name,
- the circle on top of an optional feature turns red if it is a false optional feature and
- each dead feature is crossed out in red.

Proofs

After running the corresponding analyses, a logical proof for each conjecture and for each dead feature can be inspected. Therefore, select the tab *Proofs* on the right side-bar and the statement to be proved using the combo-box at the top of the tab.

If the statement to be proved is a rejected conjecture, the list below will contain a minimal buildable configuration that violates the conjecture. Hover the mouse over any list item to highlight the features that are part of this configuration within the feature model view.

Figure 19 shows a muesli that contains *Quinoa* but no *Chocolate rasps*. The first conjecture from section *Adding a Conjecture* (`implies(or("Flakes","Quinoa"),"Chocolate rasps"))` is therefore rejected.



Proofs list (Conjectures: implies(or("Flakes","Quinoa"):
- ◉ Muesli
- ◉ Basis
- ◉ Fruits
- ○ Chocolate rasps
- ○ Lactose-free
- ○ Flakes
- ○ Oats
- ◉ Quinoa
- ◉ Strawberry

Figure 19: Buildable product as a proof for a rejected conjecture

If the statement to be proved is a confirmed conjecture or a dead feature, the list below will contain the steps of a proof by contradiction: The negation of the statement to be proved will be assumed and led to a contradiction on the basis of the logical connections within the feature model. Hover the mouse over any step to highlight the features that are related to that step within the feature model view (see Figure 20).

Since *Basis* is an alternative, *Quinoa* and *Oats* are mutually exclusive. If the second conjecture from section *Adding a Conjecture* (`implies("Flakes",not(and("Oats","Quinoa")))`) is assumed negated, *Oats* as well as *Quinoa* must be part of the muesli[3]. Those three statements are contradictive. Therefore the assumption must be rejected and the conjecture is confirmed.

[3] Here is the equivalent transformation:

```
  not(implies("Flakes",not(and("Oats","Quinoa"))))
⇔ not(or(not("Flakes"),not(and("Oats","Quinoa"))))
⇔ and(not(not("Flakes)),not(not(and("Oats","Quinoa"))))
⇔ and("Flakes",and("Oats","Quinoa"))
⇔ and("Flakes","Oats","Quinoa")
```

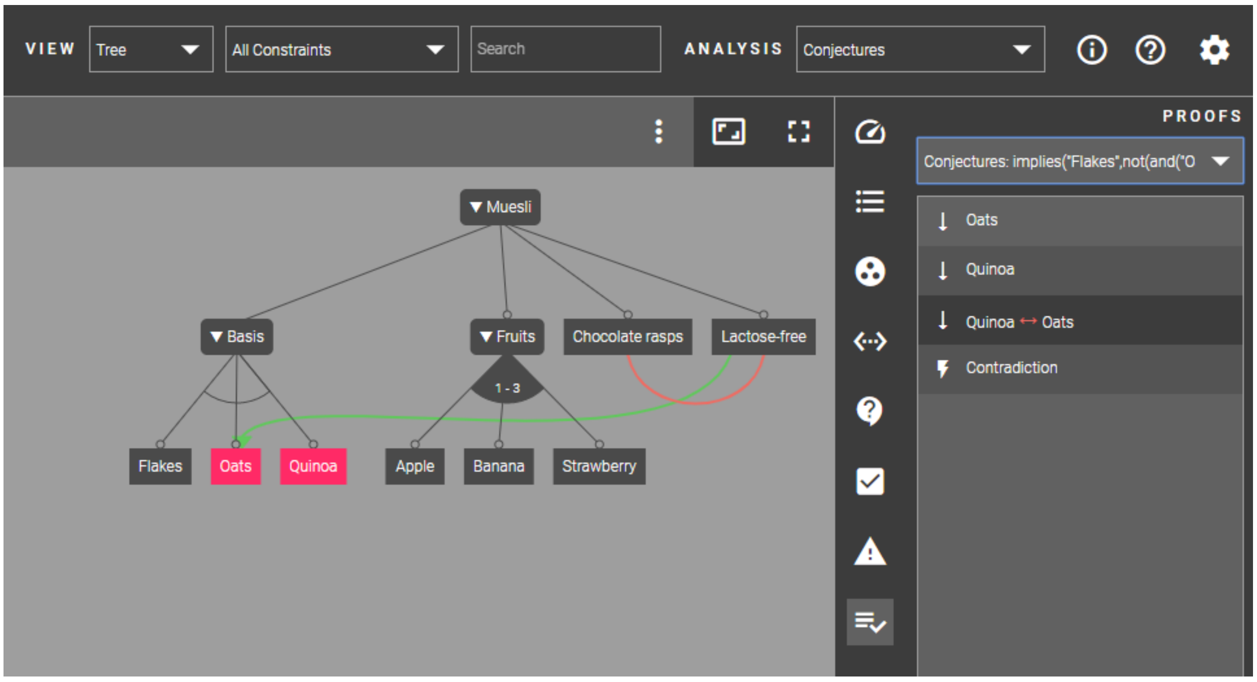


Figure 20: *Proof by contradiction for a confirmed conjecture*

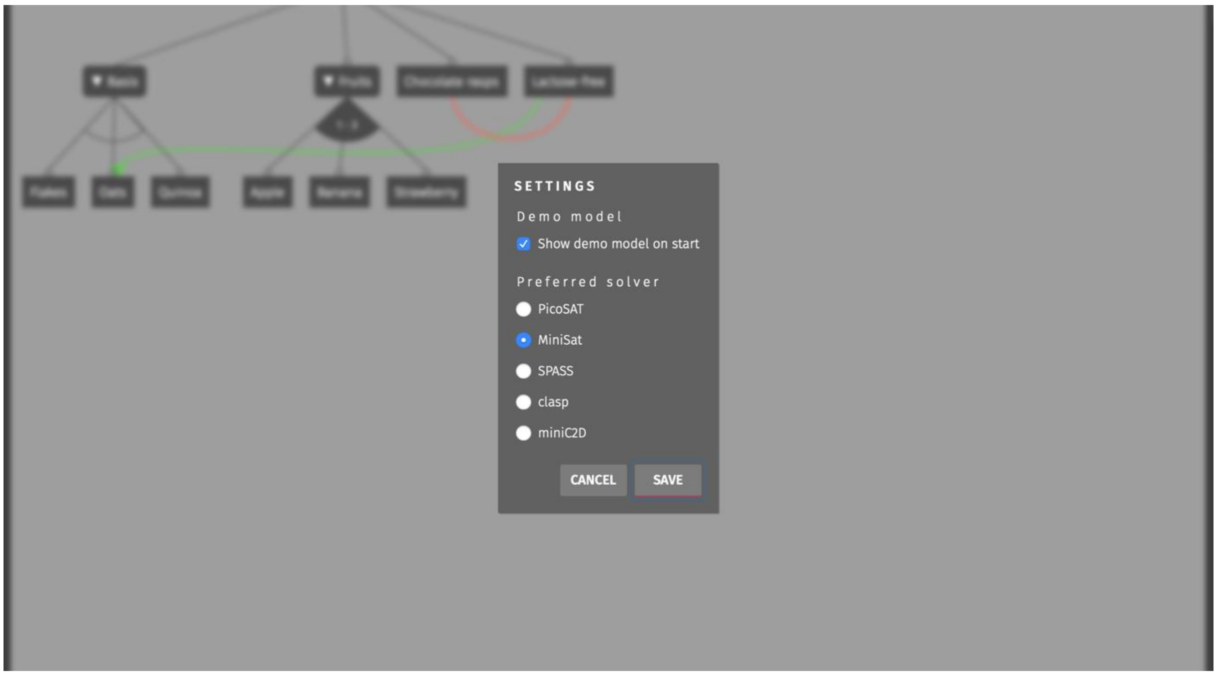
Selecting the Preferred Solver

Glencoe utilizes a set of freely available and well-known applications called *solvers* in order to execute the analyses. It is possible to change the preferred solver.

Regardless of that choice, Glencoe may select a more appropriate one in terms of runtime performance depending on the requested analysis. The results of an analysis never depend on that choice.

To choose the preferred solver, click on *Sign in* within the top-bar and select *Settings*.

A dialog with a list of all available solvers pops up. Select the preferred solver and confirm with *Save* (see Figure 21).



SETTINGS

Demo model

[✓ Show demo model on start]

Preferred solver

PicoSAT [✓ MiniSat] SPASS clasp miniC2D

Figure 21: Selecting the preferred solver

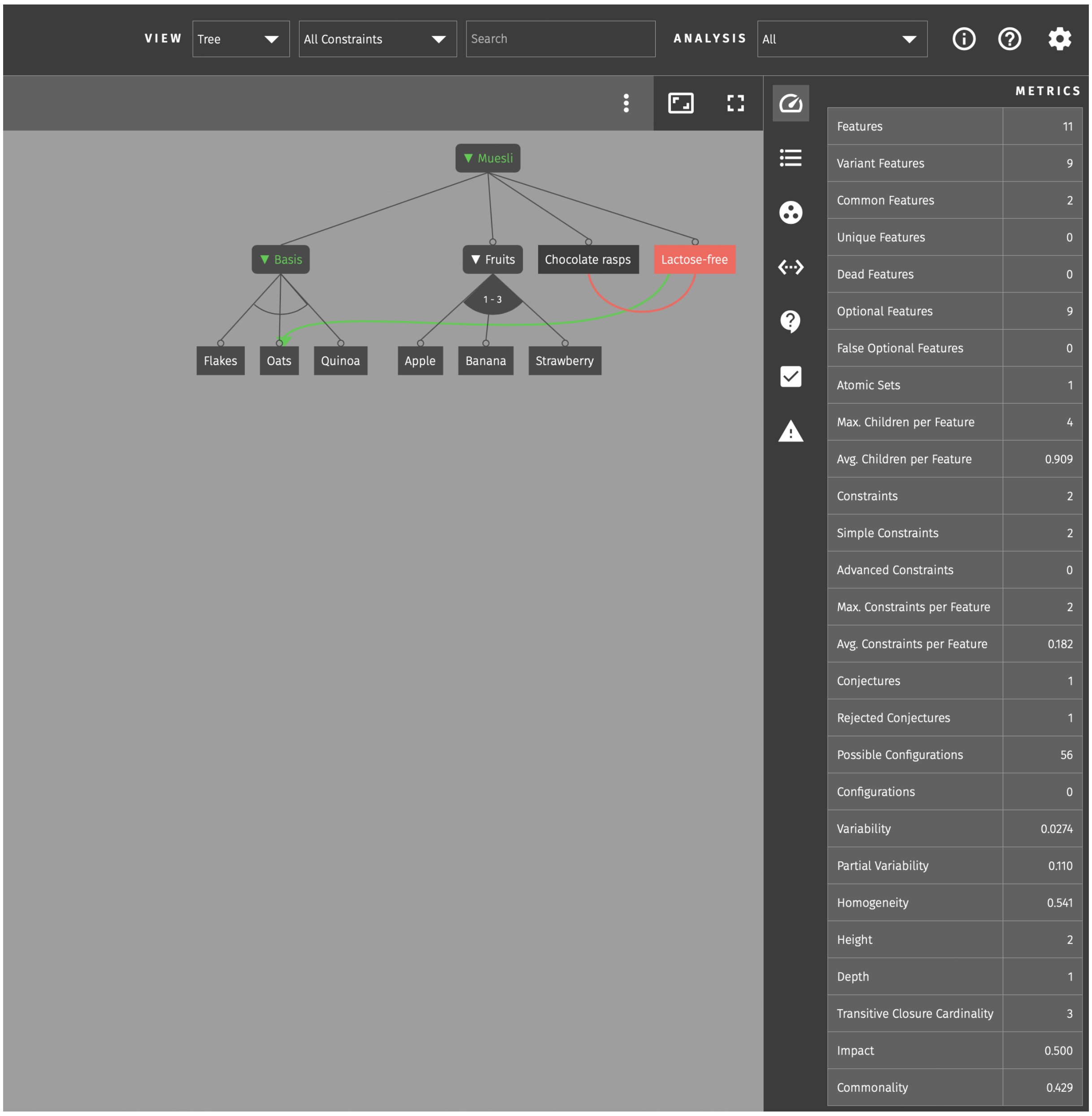
Metrics

The metrics provided by Glencoe are displayed within the tab *Metrics* on the right side-bar (see Figure 22).

While most metrics refer to the overall feature model some refer to a specific feature. To evaluate and display their values hover the mouse over any feature.

The following list gives a short explanation of the provided metrics:

- **Features.** The total number of features.
- **Variant Features.** The number of features that are neither common nor dead.
- **Common Features.** The number of common features.
- **Unique Features.** The number of unique features.
- **Dead Features.** The number of dead features.
- **Optional Features.** The number of optional features.
- **False Optional Features.** The number of false optional features.
- **Atomic Sets.** The number of atomic sets.
- **Max. Children per Feature.** The maximal number of child features per feature.
- **Avg. Children per Feature.** The average number of child features per feature.
- **Constraints.** The total number of constraints.
- **Simple Constraints.** The number of constraints which are of the type requirement or mutual exclusion (also known as simple constraints).



Features	11
Variant Features	9
Common Features	2
Unique Features	0
Dead Features	0
Optional Features	9
False Optional Features	0
Atomic Sets	1
Max. Children per Feature	4
Avg. Children per Feature	0.909
Constraints	2
Simple Constraints	2
Advanced Constraints	0
Max. Constraints per Feature	2
Avg. Constraints per Feature	0.182
Conjectures	1
Rejected Conjectures	1
Possible Configurations	56
Configurations	0
Variability	0.0274
Partial Variability	0.110
Homogeneity	0.541
Height	2
Depth	1
Transitive Closure Cardinality	3
Impact	0.500
Commonality	0.429

Figure 22: Metrics

- Advanced Constraints. The number of constraints which are not classified as simple constraint (also known as advanced constraints).
- Max. Constraints per Feature. The maximal number of constraints per feature.
- Avg. Constraints per Feature. The average number of constraints per feature.
- Conjectures. The total number of conjectures.
- Rejected conjectures. The number of rejected conjectures.
- Possible Configurations. The number of possible configurations.
- Configurations. The number of created configurations.
- Variability. The ratio between the number of possible configurations and $2^n - 1$ where n is the number of features.

- **Partial Variability.** The ratio between the number of possible configurations and $2^n - 1$ where n is the number of variant features.
- **Height.** The maximal feature depth.
- **Depth.** The number of features within the path from the root feature to the selected feature.
- **Transitive Closure Cardinality.** The number of features that are directly or indirectly linked to the selected feature by any constraint.
- **Impact.** A value between 0 and 1 indicating the potential impact of changes made to the selected feature.
- **Commonality.** The ratio between the number of possible configurations the selected feature is part of and the total number of possible configurations.
- **Homogeneity.** The ratio between the sum of commonalities and the number of features.

Modeling approaches

By now it is possible to further extend the variance within the feature model. In order to demonstrate two different modeling approaches we will add a new feature *Gluten-free* to the product line.

The first approach is to add another optional child feature to the root feature with the name *Gluten-free*. Further a requirement from the new feature *Gluten-free* to *Quinoa* (since this is the only gluten-free base cereal) has to be added (see Figure 23).

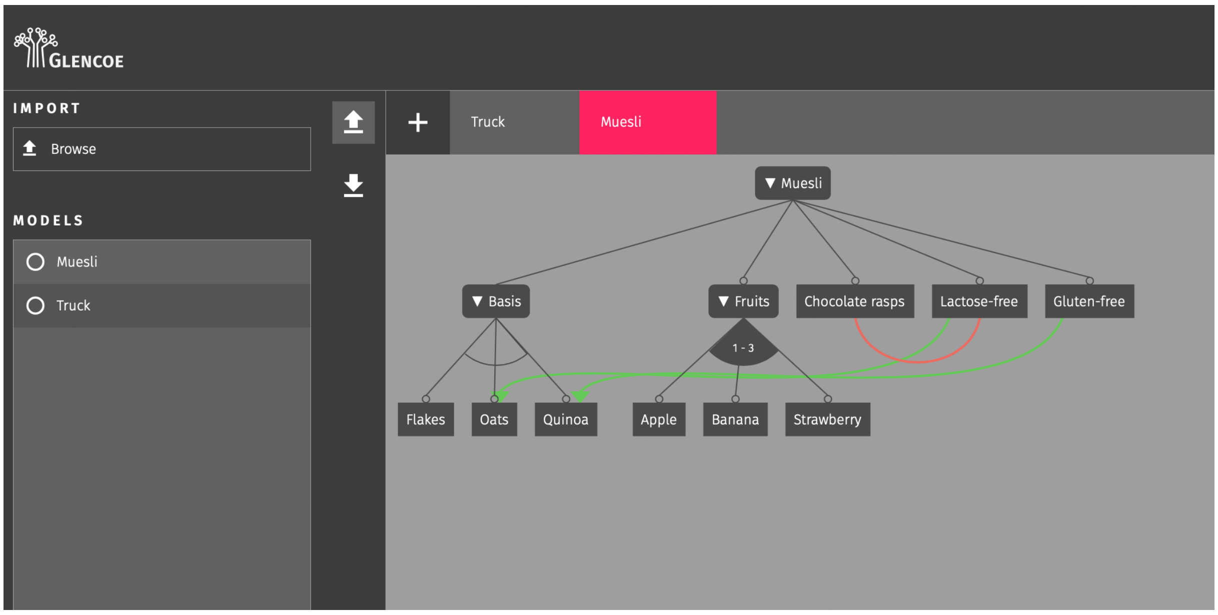


Figure 23: Modeling approach 1

An alternative approach is to introduce a new distinction below the feature *Basis* because this is the only feature affected by the product characteristic *Gluten-free*:

First delete all child features of *Basis*.

Instead, add two optional child features *Gluten-free* and *Not gluten-free* with type *Alternative* to *Basis*. Now, classify each base cereal by adding the features *Flakes*, *Oats* and *Quinoa* to the appropriate parent feature.

Re-add the requirement from *Lactose-free* to *Oats* that was deleted with the feature *Oats* in the first step (see Figure 24).

It is up to the user to decide which approach fits best. While the first approach increases the width of the feature model as well as the number of cross-tree constraints, the second one increases the depth of the model.

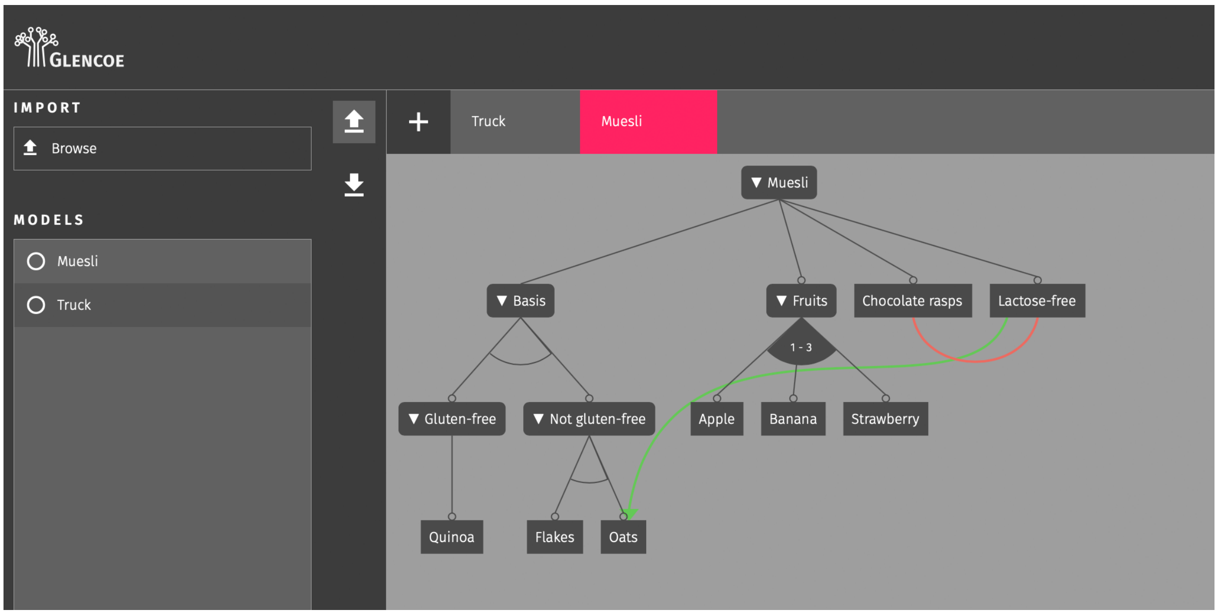


Figure 24: Modeling approach 2

Issue Handling

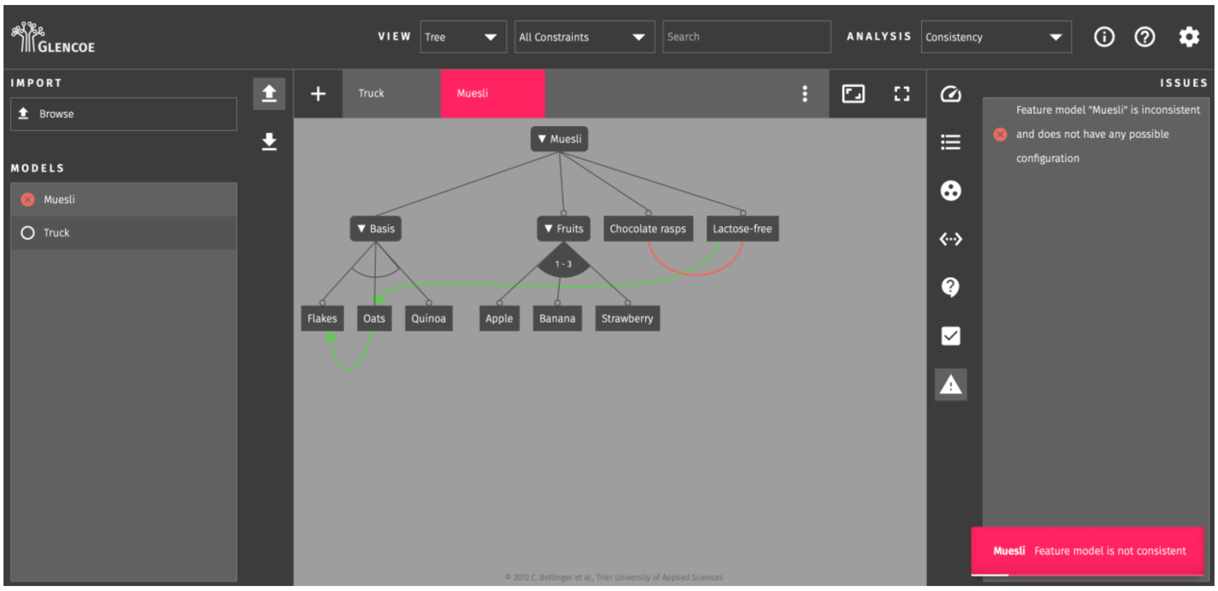
This section describes how to handle some typical issues within a feature model. These issues are usually introduced unintentionally as side-effects while making design decisions in an iterative process.

We start with the version of the feature model *Muesli* that we exported earlier.

We decide every muesli which contains oats has to contain flakes, too: Right-click on the feature *Oats* to open the context menu and select *Add requirement*. Select the feature *Flakes* using the second combo-box and click the *Create* button.

Furthermore, we decide just every muesli has to contain oats: Right-click on the feature *Oats* to open the context menu and select *Edit 'Oats'*. Uncheck the *Optional* checkbox and click the *Update* button.

Running all analyses reports the model is no longer consistent (see Figure 25).



VIEW Tree | All Constraints | Search | ANALYSIS Consistency

Feature model "Muesli" is inconsistent and does not have any possible configuration

Muesli Feature model is not consistent

Figure 25: Inconsistent feature model

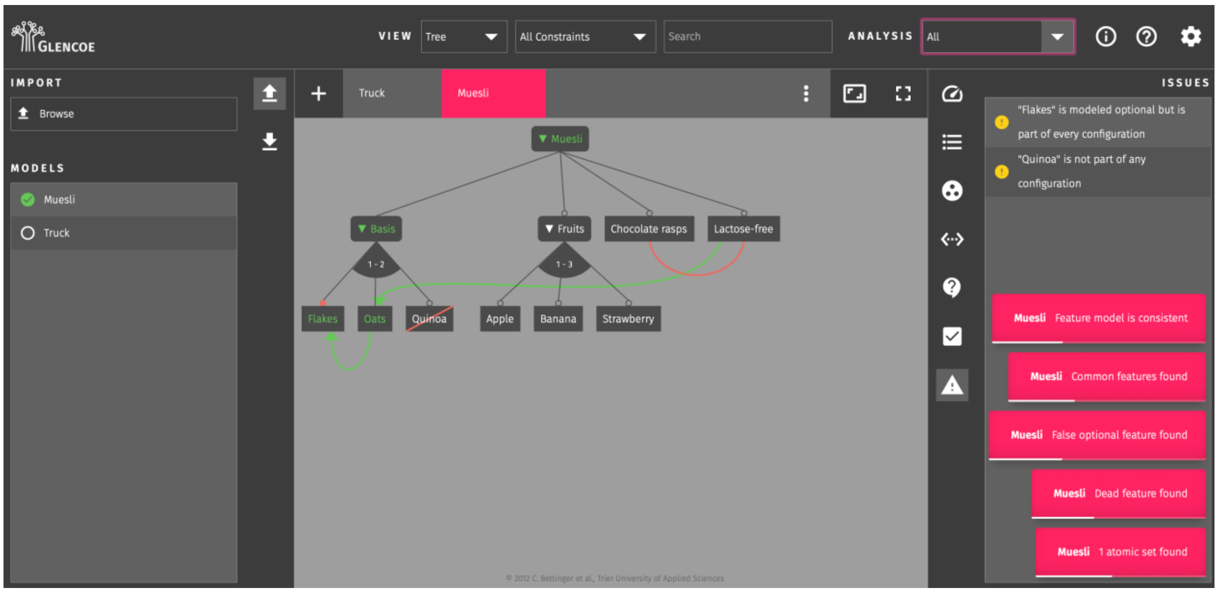
While it may be a reasonable decision that oats always come with flakes it contradicts the former decision that *Basis* is an alternative: Exactly one (not two or more) of the cereals has to be included in every muesli.

This issue can be remedied by changing the type of the feature *Basis* to a selection with at most two child features to select.

Therefore, right-click on *Basis* to open the context menu and select *Edit 'Basis'*. Set its type to *Selection of*, change the upper cardinality to 2 and click the *Update* button to confirm.

Re-running the analyses reports the model is consistent again (see Figure 26). However, the feature model now contains a false optional feature (*Flakes*) as well as a dead feature (*Quinoa*).

If every muesli contains oats and every muesli that contains oats has to contain flakes, consequently every muesli contains flakes. As a result, modeling *Flakes* as an optional feature pretends having a choice that you actually don't have.



"Flakes" is modeled optional but is part of every configuration

"Quinoa" is not part of any configuration

Muesli Feature model is consistent

Muesli Common features found

Muesli False optional feature found

Muesli Dead feature found

Muesli 1 atomic set found

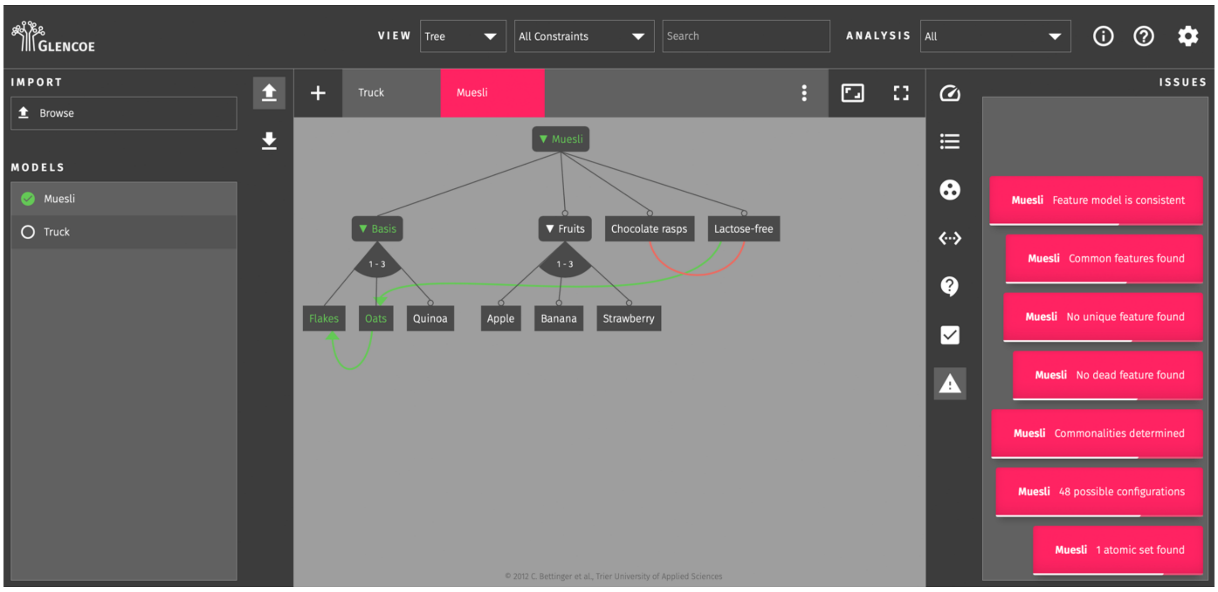
Figure 26: Feature model with a false optional and a dead feature

To solve this issue right-click on *Flakes* to open the context menu and select *Edit 'Flakes'*. Uncheck the *Optional* checkbox within the appearing dialog and click the *Update* button to confirm.

The upper cardinality of *Basis* has been set to 2, so no more than two cereals can be included in a muesli. From the three available cereals oats and flakes are part of every muesli. Therefore, no muesli can contain quinoa at all – *Quinoa* is a dead feature.

If this is an expected or acceptable consequence, you can simply delete the dead feature *Quinoa*.

Otherwise, this issue can be remedied by changing the type of *Basis* to *Selection*. This increases the upper cardinality from 2 to the number of child features (3). For that reason, every muesli now contains oats and flakes but may or may not contain quinoa (see Figure 27).



Muesli Feature model is consistent

Muesli Common features found

Muesli No unique feature found

Muesli No dead feature found

Muesli Commonalities determined

Muesli 48 possible configurations

Muesli 1 atomic set found

Figure 27: Repaired feature model

Using the Interactive Configurator

This section describes how to create configurations and use the interactive configurator to deduce specific products of your product line.

To create a configuration and add it to the feature model, select the tab *Configurations* on the right side-bar and click the *Create configuration* button (see Figure 28).

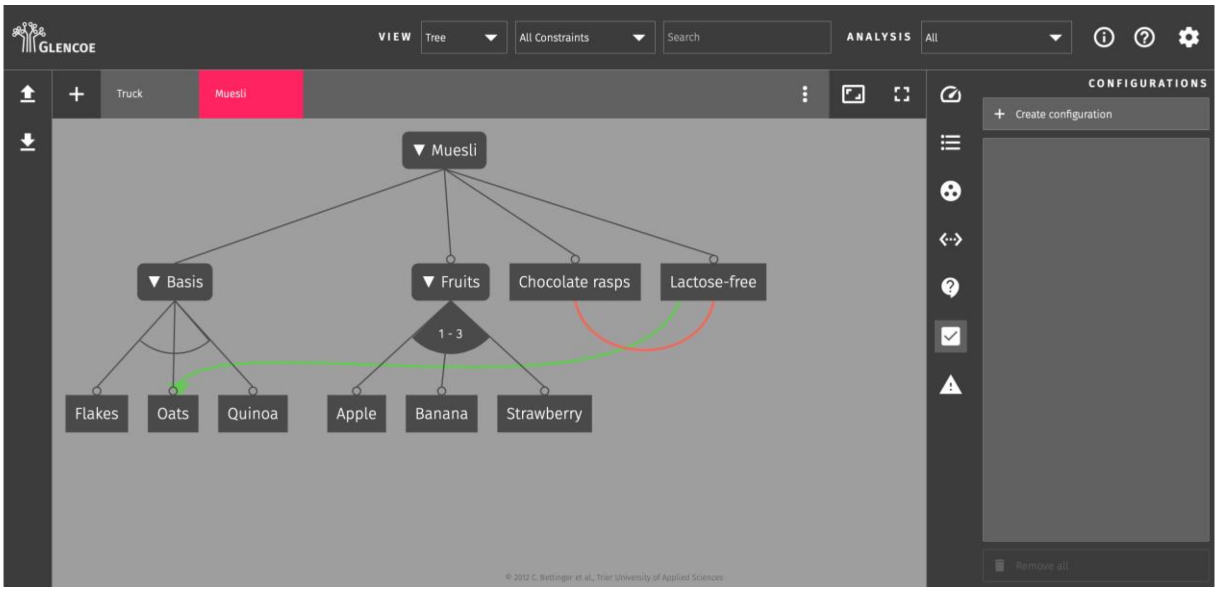
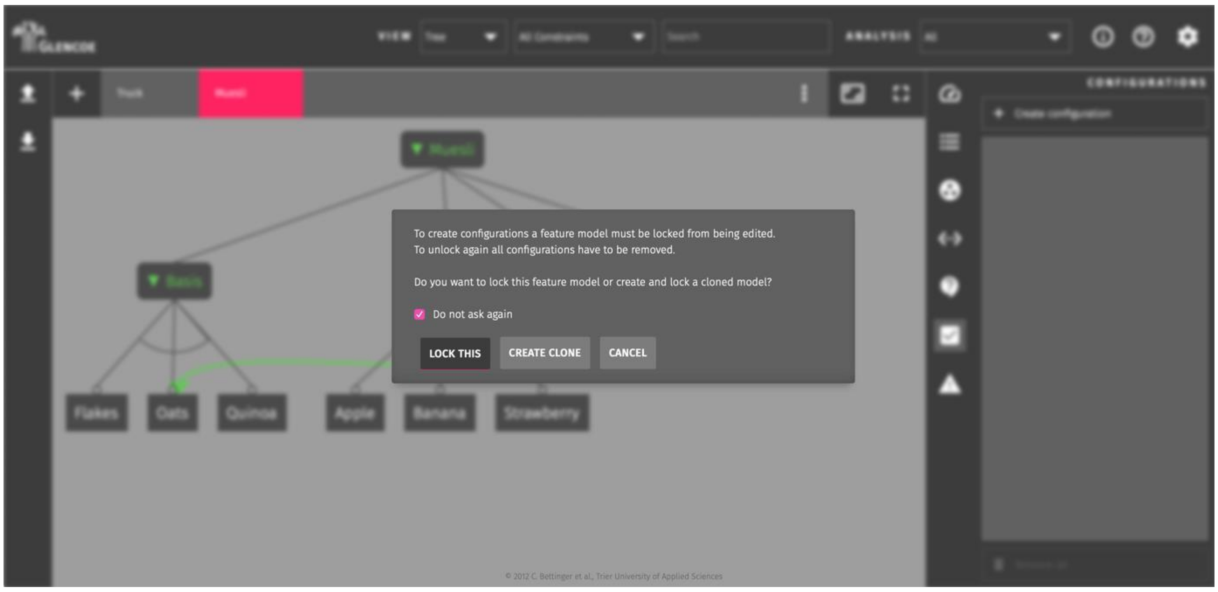


Figure 28: Adding a configuration

While a feature model contains some configuration, it must be locked from being edited. You will be asked whether you want to lock that feature model or create and lock a clone of the feature model (see Figure 29) when creating a first configuration. To unlock a feature model later on, all configurations have to be removed first. Therefore, click the *Remove all* button below the list of configurations.


To create configurations a feature model must be locked from being edited.
To unlock again all configurations have to be removed.

Do you want to lock this feature model or create and lock a cloned model?

Do not ask again

Figure 29: Locking a feature model

The newly created configuration will be added to the list of configurations and selected automatically. To show another configuration, click on that configuration within the list. To hide a selected configuration, click on that configuration again.

Now you can start to decide which features you want to be part of the product by clicking these features. A user-selected feature will be highlighted by applying a darker green background color.

On each single decision Glencoe determines and executes all consequences arising from the product line specification. Features which in consequence must be part of the product too will have a light green background color, features which must not be part of the product line will have a light red background color.

In Figure 30 you can see that selecting the feature *Lactose-free* results in deselecting *Chocolate rasps* (since there is a mutual exclusion). Additionally, *Oats* are selected (since there is a requirement) and the other options within the alternative *Basis* are deselected.

If the configuration represents a buildable product the icon in front of the configuration's name within the list of configurations will turn green (see Figure 30). Since all features that have not been decided yet are optional this Muesli configuration is already buildable.

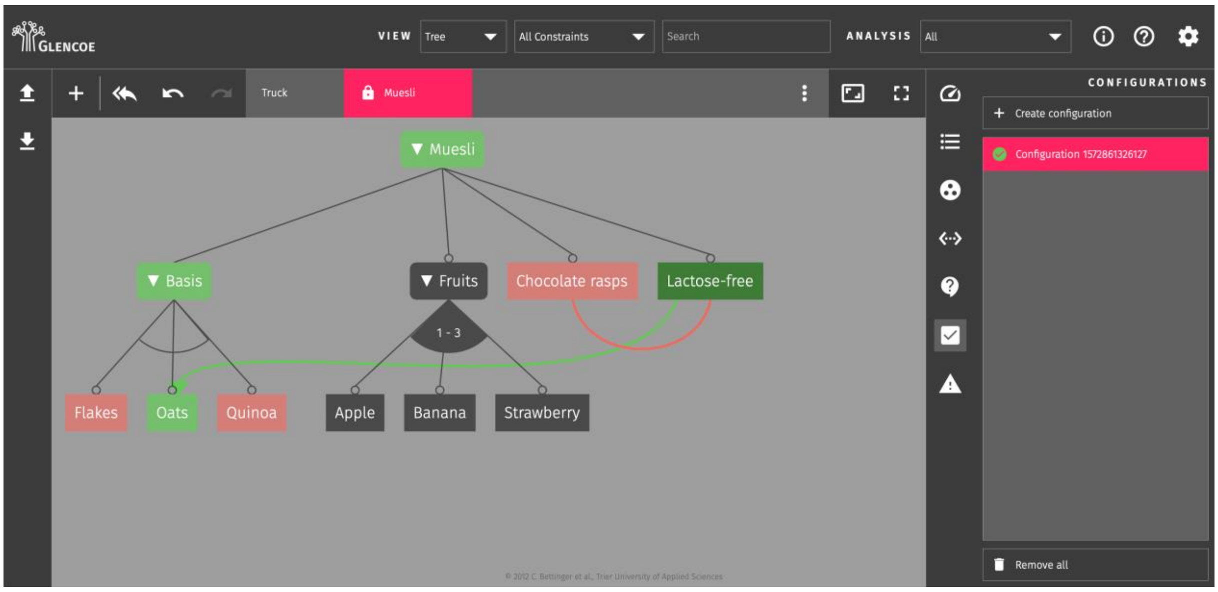


Figure 30: A buildable configuration by selecting the feature Lactose-free

To revert a user decision, click on that feature again. Alternatively, you can use the *Undo*/*Redo* buttons on the left of the feature model tab-bar.

Besides selecting a feature, the user can explicitly deselect a feature by right-clicking on it. A user-deselected feature will be highlighted by applying a darker red background color.

In Figure 31 you can see that deselecting the feature *Oats* results in deselecting *Lactose-free*, since there is a requirement. This configuration is incomplete and not buildable yet, which is indicated by a white filled icon in front of the configuration's name within the list of configurations.

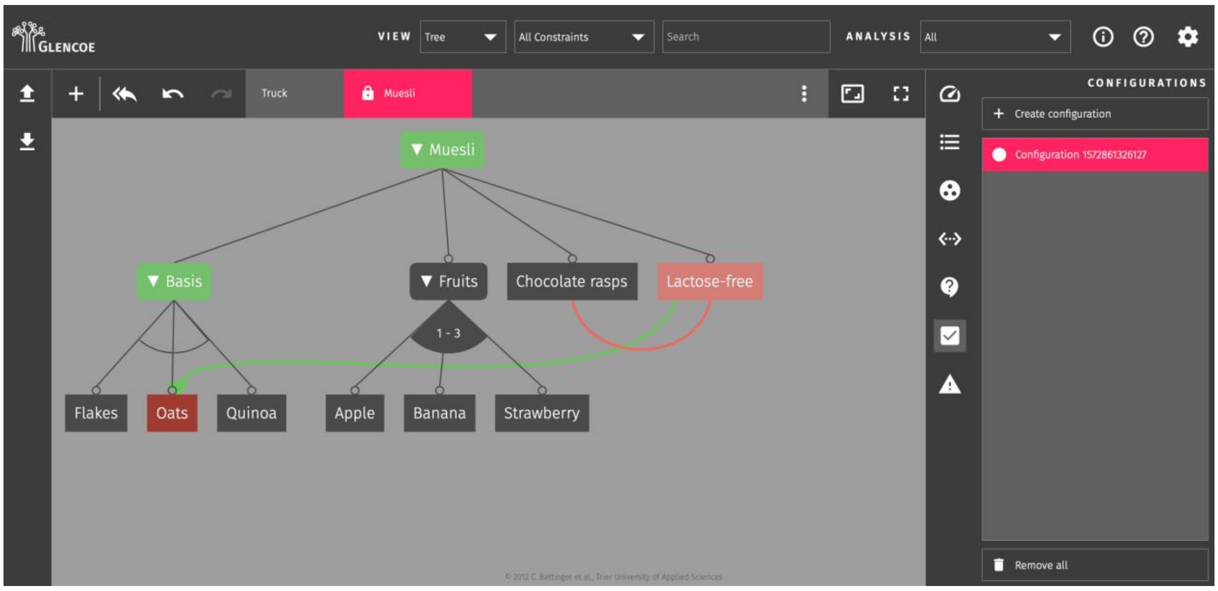


Figure 31: An incomplete configuration by deselecting the feature Oats