FTS2MTS

A Tool for Creating, Converting and Comparing FTS and MTS

Mahsa Varshosaz[1], Lars Luthmann[2], Paul Mohr[2], Malte Lochau[2], and Mohammad Reza Mousavi[3]

[1]Halmstad University, Sweden
[2]TU Darmstadt, Germany
[3]University of Leicester, UK

September 28, 2018

To demonstrate the conversion algorithm described in the accompanying paper, this program supports the creation of both Featured Transition Systems (FTS) and Modal Transition Systems (MTS) from scratch. The transition systems are represented both visually and in the form of strings. Using the latter, any previously created FTS or MTS can be re-imported into the program.

Once an FTS has been created, it can be converted into a set of MTS. The FTS and the set of MTS are continuously compared using bisimilarity, and the result of this comparison is displayed.

1 Basic Information

1.1 Key concepts

We begin with some concepts and conventions that will be relevant throughout the interface overview.

configStrings contain all information necessary to build a transition system from scratch and can be used for storage and transfer. All names which are part of the configString must conform to the naming rules.

Naming Rules are imposed on all names. Violating names are automatically rejected. The rule set consists of forbidden symbols and prefixes and is displayed as a tool tip for each relevant text field.

Feature Constraints can be entered in one of three ways:

1. A single feature ID or name, possibly negated using a '-' prefix
2. A format based loosely on the dimacs SAT format: opening parentheses are immediately followed by either '*' or '+' (denoting conjunction and disjunction respectively) and a space separated list of (possibly negated) feature IDs/names or nested expressions.
3. The human readable output of the internal logic engine of this program produces is supported mostly in the interest of copy-pasting. It exclusively uses the CNF format (as the internal logic engine does): Clauses are contained in parentheses and linked together using " A "[1]. Each clause contains a list of literals linked together using " v ", each literal is the name or ID of a feature, with or without a "-" prefix signaling its inversion.

The syntax of these formats is also expressed in the tool tips of all relevant text fields.

[1]Java does not trivially support use of the $\wedge$ or $\vee$ symbols.

1.2 The Main Window

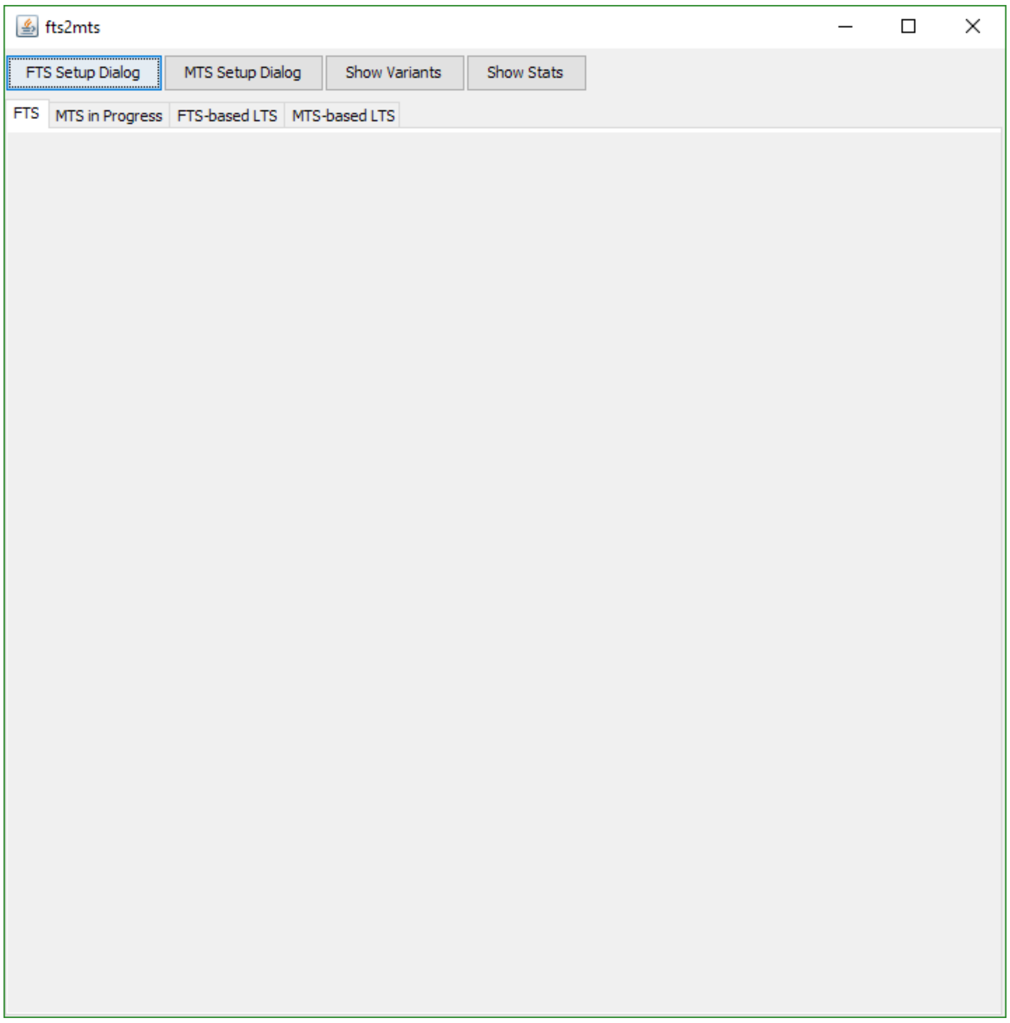
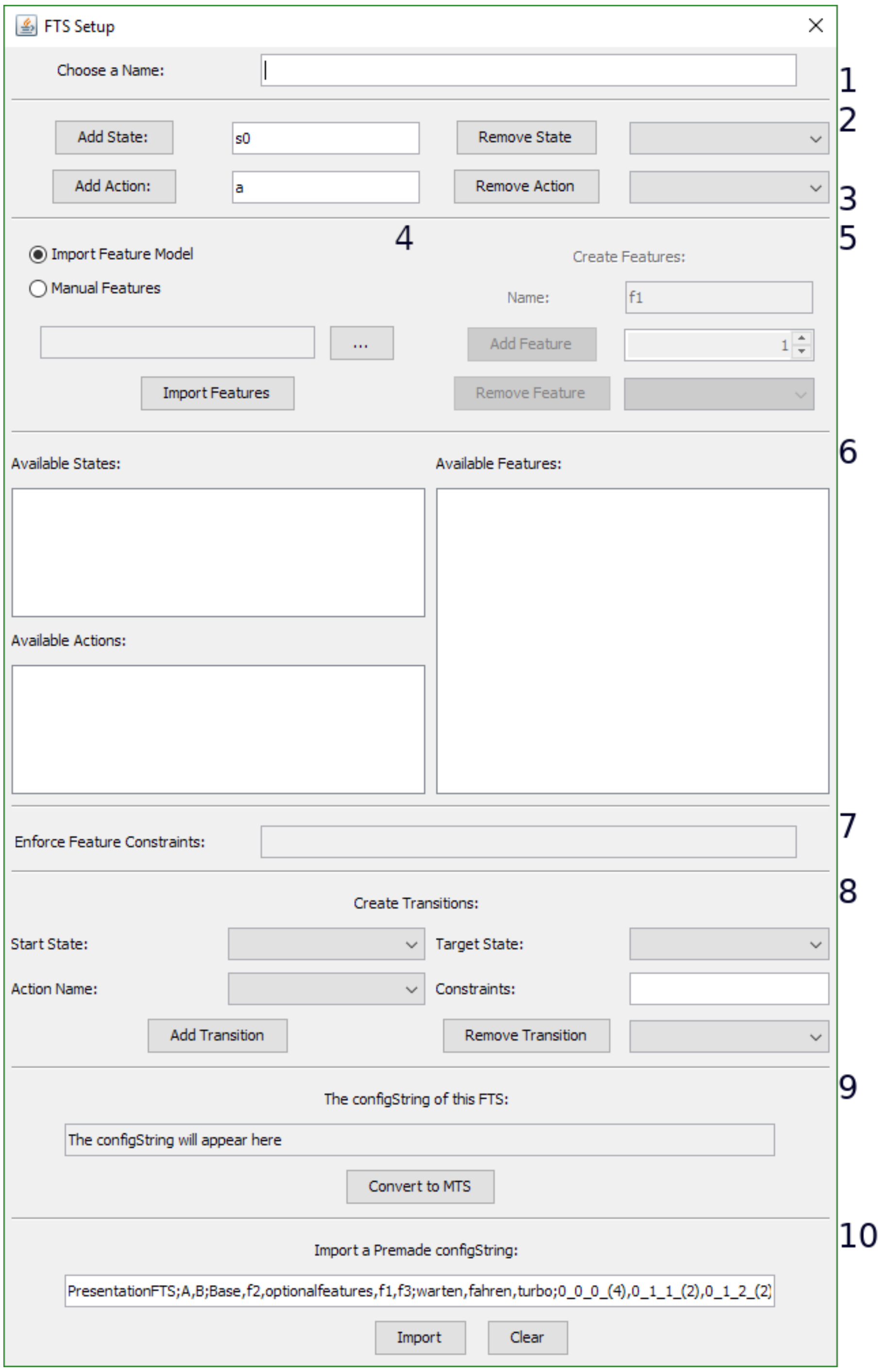


Figure 1: The main provides access to the other windows and visually displays the transition systems.

Figure 1 shows a screenshot of the main window.

- The four buttons at the top left of the main window open the other windows.
- At the top right (currently invisible), information regarding the bisimilarity of the FTS and the set of MTS will be displayed.
- Below is the area displaying the different types of transition system. The tabs can be used to choose which transition system to display at any given time. The first four tabs are always available, further tabs will be added and removed to keep track off the variable number of MTS.

1.3 The FTS Setup Dialog



Add State: s0

Add Action: a

(● Import Feature Model) ○ Manual Features

Name: f1

Add Feature: 1

The configString of this FTS: The configString will appear here

Import a Premade configString: PresentationFTS;A,B;Base,f2,optionalfeatures,f1,f3;warten,fahren,turbo;0_0_0_(4),0_1_1_(2),0_1_2_(2)

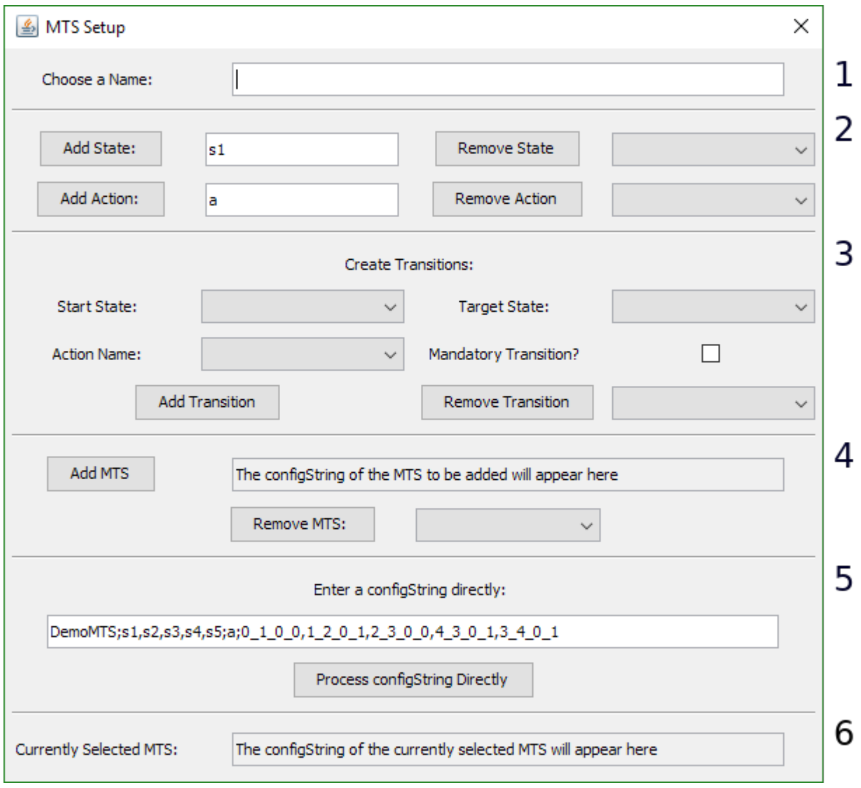
Figure 2: The FTS setup dialog is used to generate or import FTS, and to convert FTS to MTS.

Figure 2 shows a screenshot of the FTS setup dialog.

1. A name for the FTS can be set. The name will be displayed as the title of the associated tab and prefix the configString.
2. States can be selectively added and removed. When adding a state causes a naming conflict, the integer suffix of the new state is incremented (or 1 is added as a new suffix), allowing numbered states to be created swiftly. A state currently used in a transition may not be removed. As states are referred to by their IDs, removing a state will leave an unbound ID. Empty IDs are automatically assigned as more states are added.
3. Actions are created and removed similar to states (cf. 2).
4. On the left side, a choice is offered to create features manually (see 5) and to import them from an xml file created by FeatureIDE[2], which will (de)activate the relevant interface components. Once a feature model has been imported, switching into manual mode will allow for the manipulation of the imported features and initial constraints.
5. On the right side, features can be added and removed similarly to states and actions. Features *can* be removed even if they are currently in use by a transition. In this case, the visual display will refer to the feature lacking a name by its ID. To allow deliberate re-naming of such orphaned features, the ID of newly created features can be chosen arbitrarily. Similar to the integer suffix of state and action names, this target ID is incremented automatically should the desired ID be unavailable.
6. This subsection allows the review of all states, actions and features. It displays each name alongside its ID.
7. If no feature model was imported, the effects one would have can be emulated using a logical proposition - only feature combinations allowed by this proposition will be considered when MTS are generated.
8. Each transition requires a start state, an end state, an associated action and a feature constraint governing its presence in any given configuration of the FTS. The relevant syntax is identical to the one discussed in 7.
9. The text field is continually updated with the configString corresponding to the FTS. (As is the visual representation in the main window). Pressing the button generates a bisimilar set of MTS from the given FTS.
10. Lastly, a pre-existing FTS configString can be entered directly, circumventing all other sections of the dialog. Pressing the "Clear" is equivalent to importing an entirely empty configString.

[2]Names used within the feature model may be illegal in the context of this program. In that case, illegal symbols will be stripped, illegal prefixes will be masked by a legal prefix ("a") and the same suffix incrementation used for adding states and actions is applied to resolve any resulting overlaps.

1.4 The MTS Setup Dialog



s1

a

The configString of the MTS to be added will appear here

DemoMTS;s1,s2,s3,s4,s5;a;0_1_0_0,1_2_0_1,2_3_0_0,4_3_0_1,3_4_0_1

The configString of the currently selected MTS will appear here

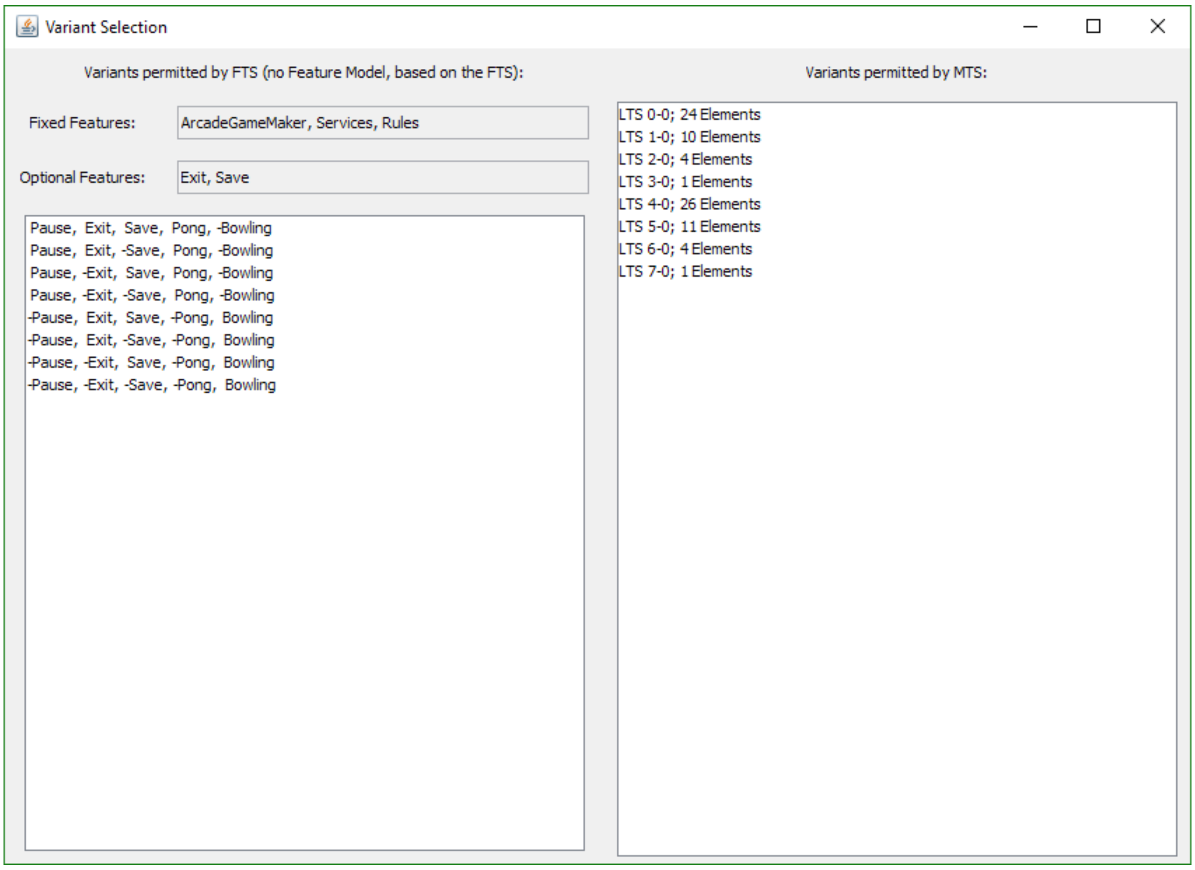
Figure 3: The MTS setup dialog is used to create and remove MTS.

The MTS setup dialog shown in figure 3 is a subset of the FTS Setup Dialog discussed in 1.3 in many ways. Unless otherwise specified, all subsections function like their respective counterparts.

1. The MTS can be named.
2. States and actions can be created.
3. Transitions can be created. MTS transitions do not have feature constraints, but can be either optional- or mandatory-transitions.
4. The MTS can be added to the active set, causing it to be displayed in a new tab on the far right. Furthermore, LTS are generated using the active set and those LTS are used to check for bisimilarity with the FTS. Since the MTS currently being built is *not* part of the active set, it is displayed in a dedicated tab[3]. MTS can also be removed from the active set.
5. Import an MTS from an MTS configString. The configStrings for FTS and MTS are not interchangeable, as the latter lack features and initial constraints entirely and use a different format for their transitions.
6. The field in 4. is not sufficient to obtain configStrings as MTS created during FTS conversion do not pass through the MTS setup dialog. To circumvent this problem, whenever the main window is displaying an MTS, its configString can be retrieved here.

[3]"MTS in Progress"

1.5 The Variant Selection Window


Variant Selection
Variants permitted by FTS (no Feature Model, based on the FTS):
Variants permitted by MTS:

Fixed Features: ArcadeGameMaker, Services, Rules

Optional Features: Exit, Save

```
Pause, Exit, Save, Pong, -Bowling
Pause, Exit, -Save, Pong, -Bowling
Pause, -Exit, Save, Pong, -Bowling
Pause, -Exit, -Save, Pong, -Bowling
-Pause, Exit, Save, -Pong, Bowling
-Pause, Exit, -Save, -Pong, Bowling
-Pause, -Exit, Save, -Pong, Bowling
-Pause, -Exit, -Save, -Pong, Bowling
```

```
LTS 0-0; 24 Elements
LTS 1-0; 10 Elements
LTS 2-0; 4 Elements
LTS 3-0; 1 Elements
LTS 4-0; 26 Elements
LTS 5-0; 11 Elements
LTS 6-0; 4 Elements
LTS 7-0; 1 Elements
```

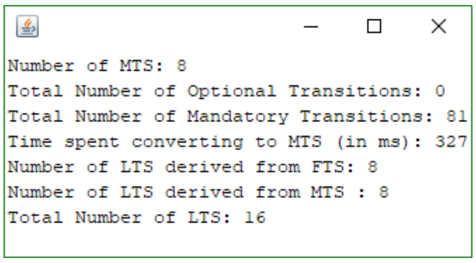
Figure 4: The variant selection window allows comparisons between LTS based on either the FTS and its feature model or the generated set of MTS.

For the sake of presentation, figure 4 shows the variant selection window not as it would appear by default, but after an FTS has already been converted. By default, all fields and lists are completely empty.

- The left side is concerned with LTS variants derived from the FTS and its associated feature model.
  - At the top, there are two fields meant to increase readability: In the fixed features field, all features which take on the same value in all configurations are listed. If that value is *false* , the name of the feature will be preceded by a "-". These features will not appear in the list below to reduce clutter.
  - The field directly below points out optional features, which can take on both values regardless of all other features. A feature being optional has no effect on the list below.
  - At the bottom, the list of generated variants will appear. Entries on this list will be compared against one another, all entries recognized to be bisimilar to an entry higher up on the list will be marked with the suffix " (redundant)".
- On the right, there is only the list of generated MTS variants. For each MTS, each possible combination of currently present and currently missing optional transitions is generated and displayed here. The LTS are numbered according to their parent MTS and the order of their generation. As on the left, the list is checked for internal redundancy.
- Selecting an entry from either list will draw the associated LTS in the appropriate tab[4] in the main window and grant focus to that tab.

[4]"FTS-based LTS" and "MTS-based LTS", respectively.

1.6 The Stats Window



```
Number of MTS: 8
Total Number of Optional Transitions: 0
Total Number of Mandatory Transitions: 81
Time spent converting to MTS (in ms): 327
Number of LTS derived from FTS: 8
Number of LTS derived from MTS : 8
Total Number of LTS: 16
```

Figure 5: The stats window displays some key figures about the FTS, the active MTS set and the sets of generated LTS.

For the sake of presentation, figure 5 shows the stats window not as it would appear by default, but after an FTS has already been converted. By default, all values are zero.
The content of this window should be self-explanatory.

2 Step-by-step Creation of an FTS

Let's assume that we want to convert an FTS into MTS, and that we do not already know that the relevant configString is

```
manualFTS;state0,state1,state2,state3;Root,f2,f3,f4,f5;a,b,c,d,e;
0_1_0_(2),1_2_1_(3),1_2_2_(-3),2_1_3_(4),0_3_4_(5);(1) A (2 v 5) A (2 v -3) A
(2 v -4) A  (-2 v -5) A (-3 v -5) A (-4 v -5) A (3 v 4 v 5) A (-2 v 3 v 4)
```

(These line breaks will be tolerated. Nevertheless, they constitute a deviation from the format specification and ought to be removed before processing.)

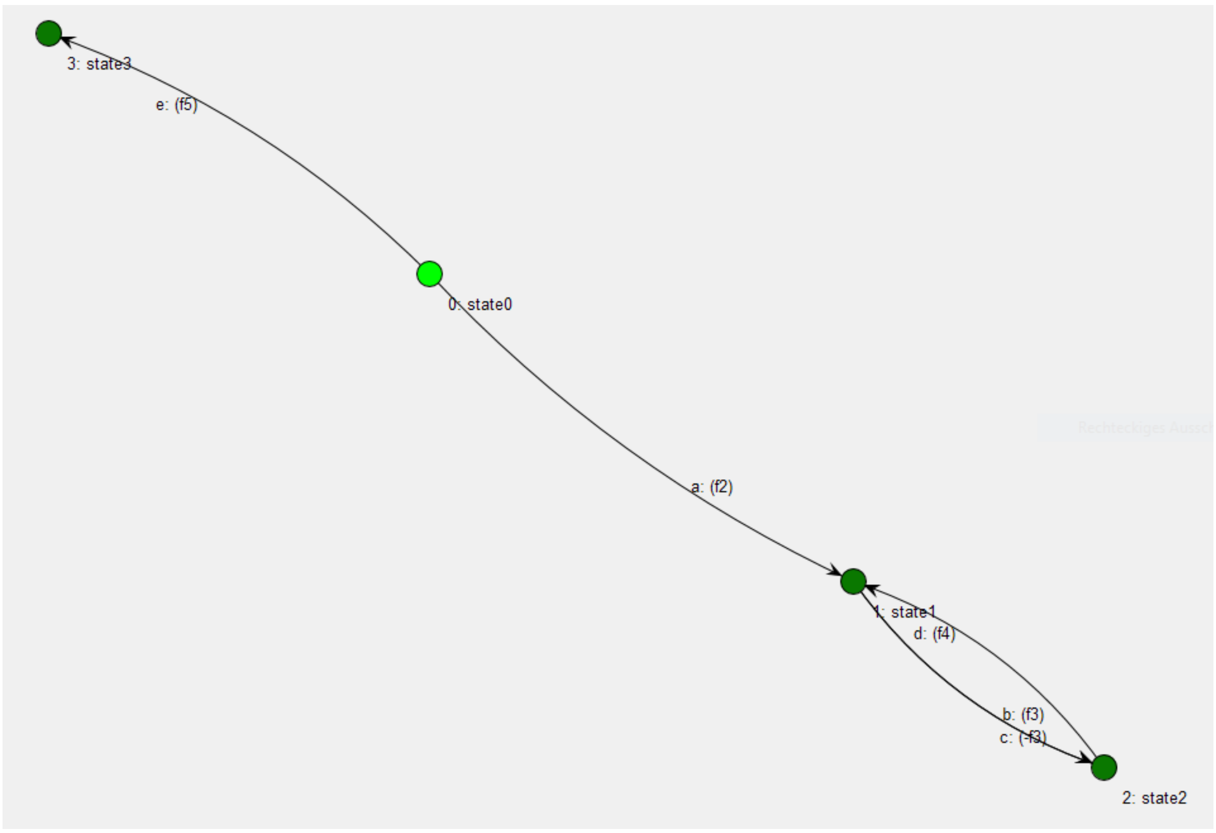
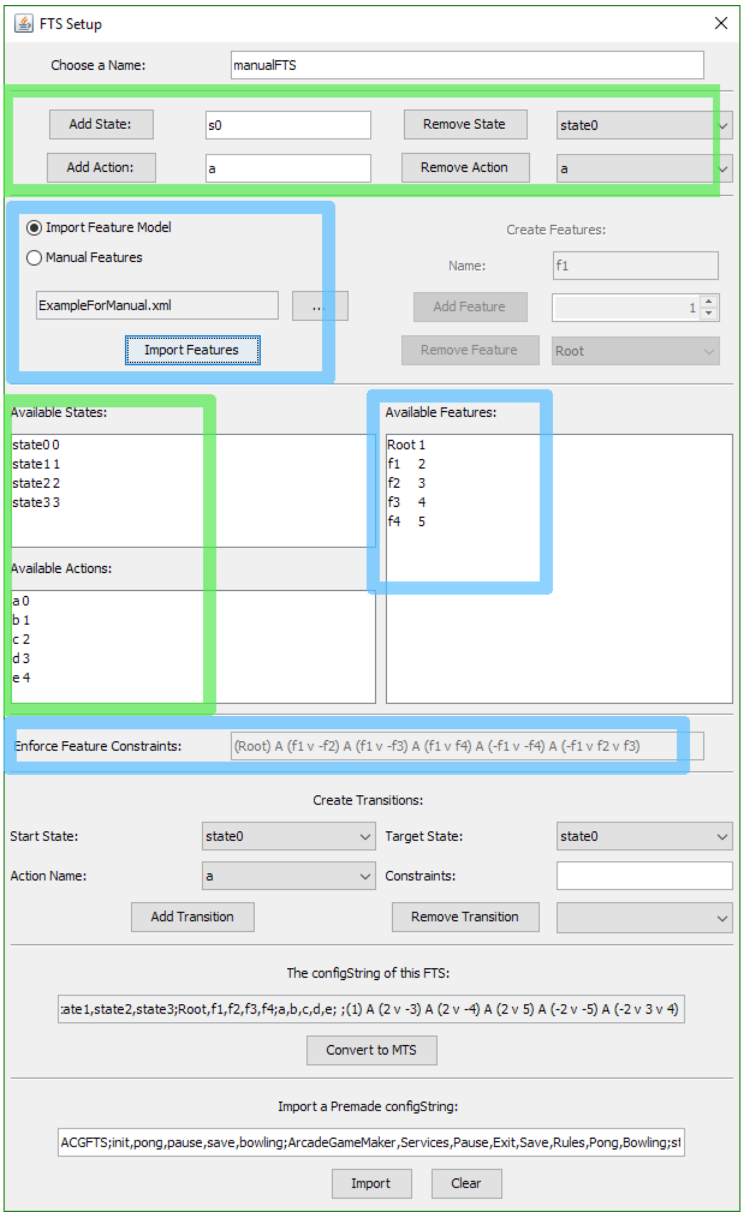


Figure 6: The FTS we want to reproduce.

2.1 States, Actions, Features



Choose a Name: manualFTS

Add State: s0 | Remove State: state0

Add Action: a | Remove Action: a

(●) Import Feature Model
( ) Manual Features

ExampleForManual.xml

Name: f1 | 1 | Root

Available States:

state0 0
state1 1
state2 2
state3 3

Available Actions:

a 0
b 1
c 2
d 3
e 4

Available Features:

Root 1
f1 2
f2 3
f3 4
f4 5

Enforce Feature Constraints: (Root) A (f1 v -f2) A (f1 v -f3) A (f1 v f4) A (-f1 v -f4) A (-f1 v f2 v f3)

Start State: state0 | Target State: state0

Action Name: a | Constraints:

The configString of this FTS:

```
:ate1,state2,state3;Root,f1,f2,f3,f4;a,b,c,d,e; ;(1) A (2 v -3) A (2 v -4) A (2 v 5) A (-2 v -5) A (-2 v 3 v 4)
```

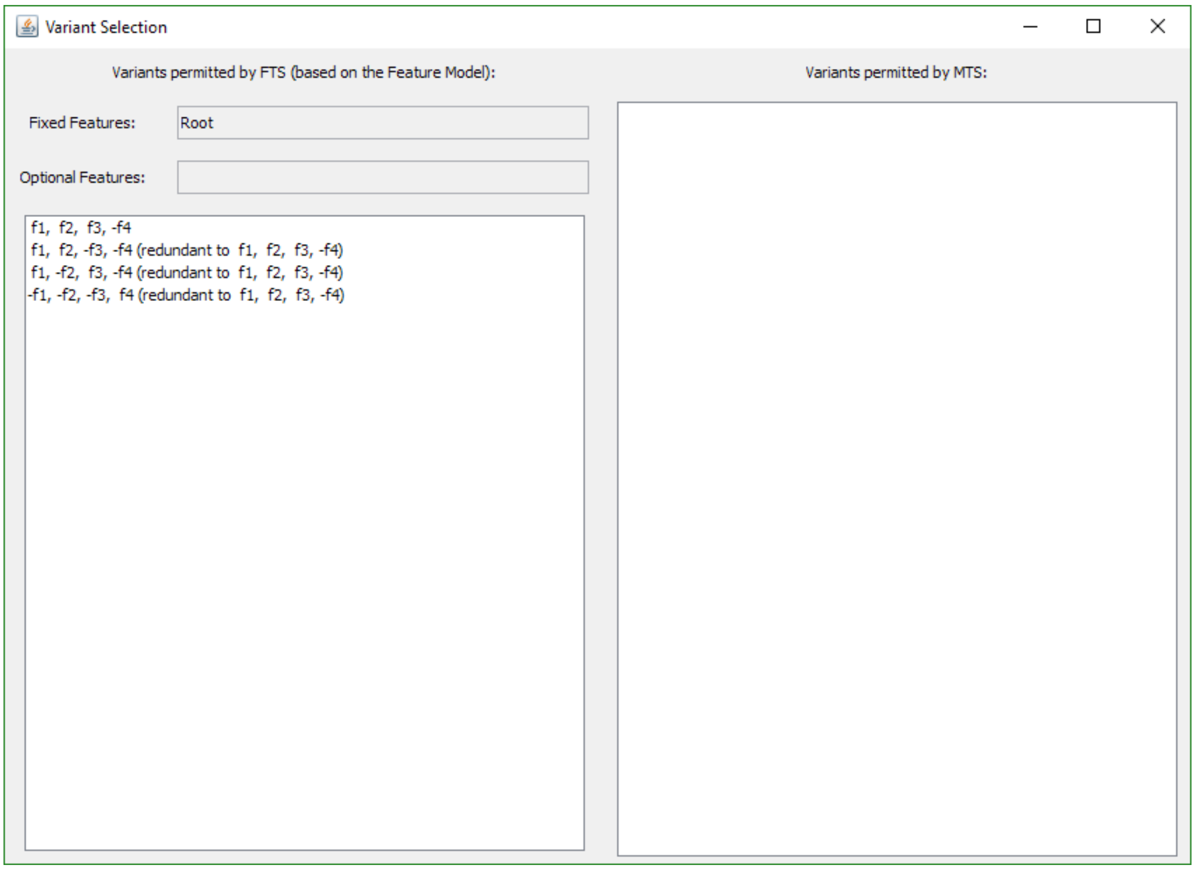
Import a Premade configString:

```
ACGFTS;init,pong,pause,save,bowling;ArcadeGameMaker,Services,Pause,Exit,Save,Rules,Pong,Bowling;st
```

Figure 7: The FTS Setup Dialog, after entering states, actions and features.

We begin by creating all necessary states and actions and importing the feature model[5]. States and actions can be created at any time prior to the creation of transitions relying on those states or transitions; features can be created at entirely arbitrary times. However, by creating them first we avoid going back and forth during transition creation.
As the FTS's states are numbered, we can simply enter "state0" in the relevant field and confirm four times using the enter key. Since the actions are not numbered, we will have to enter each name separately. Lastly, we import the feature model. We open the file chooser using the "..." button, retrieve our file, then confirm our choice by pressing "Import Features". The xml file used in this example can be found in the appendix. Note that LTS based on the possible feature configurations are already computed. As there is no FTS yet, all LTS are empty and universally redundant to one another.

[5]Note: When using a preexisting configString, the feature model must be imported last, as importing the configString will override its effects.


Variant Selection
Variants permitted by FTS (based on the Feature Model):
Variants permitted by MTS:
Fixed Features:
Optional Features:

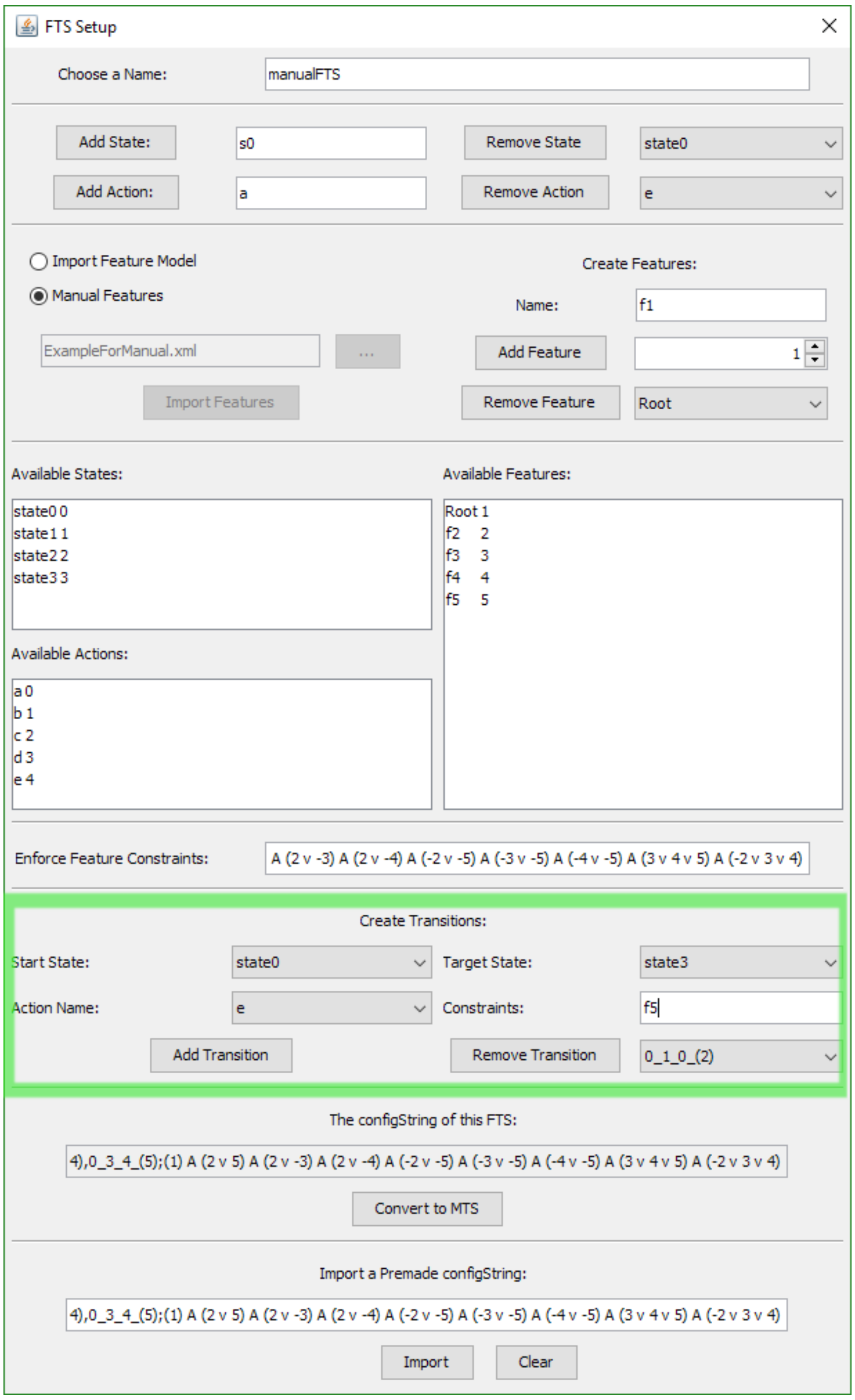
Root

```
f1,  f2,  f3, -f4
f1,  f2, -f3, -f4 (redundant to  f1,  f2,  f3, -f4)
f1, -f2,  f3, -f4 (redundant to  f1,  f2,  f3, -f4)
-f1, -f2, -f3,  f4 (redundant to  f1,  f2,  f3, -f4)
```

Figure 8: The generated feature combinations, without an FTS to contextualize them.

2.2 The Transitions

Next, we will enter the transitions. Start state, end state and action are selected using the combo boxes, the feature constraints are entered into their field. For the first transition, we choose state0, state1, a and "(*f1 -f4)", respectively. For the next transition, we choose state1, state2, b and "f2", and so on. We suggest using the mouse wheel to select states and actions, as this saves clicks and does not take focus away from the feature constraint field[6]. Feature names and their IDs (displayed in the list above) can be exchanged for one another arbitrarily, which may be helpful in the case of long or cumbersome feature names. Note that the view in the main window is updated each time a transition is added (or removed). Once all transitions are added, the configString is complete and may be copied into a separate file.



Choose a Name: manualFTS

Add State: s0 — Remove State: state0

Add Action: a — Remove Action: e

( ) Import Feature Model
(●) Manual Features

ExampleForManual.xml

Name: f1 — 1 — Remove Feature: Root

Available States:

```
state0 0
state1 1
state2 2
state3 3
```

Available Actions:

```
a 0
b 1
c 2
d 3
e 4
```

Available Features:

```
Root 1
f2   2
f3   3
f4   4
f5   5
```

Enforce Feature Constraints: A (2 v -3) A (2 v -4) A (-2 v -5) A (-3 v -5) A (-4 v -5) A (3 v 4 v 5) A (-2 v 3 v 4)

Start State: state0 — Target State: state3

Action Name: e — Constraints: f5

Remove Transition: 0_1_0_(2)

The configString of this FTS:

```
4),0_3_4_(5);(1) A (2 v 5) A (2 v -3) A (2 v -4) A (-2 v -5) A (-3 v -5) A (-4 v -5) A (3 v 4 v 5) A (-2 v 3 v 4)
```

Import a Premade configString:

```
4),0_3_4_(5);(1) A (2 v 5) A (2 v -3) A (2 v -4) A (-2 v -5) A (-3 v -5) A (-4 v -5) A (3 v 4 v 5) A (-2 v 3 v 4)
```

Figure 9: The FTS Setup Dialog, after the transitions were added.

[6]Unless it does on your system.

2.3 Generating and Evaluating MTS

Finally, we can confirm our FTS by clicking ”Convert to MTS”. Four new MTS are added to the main window, the right-hand list in the variant selection window is updated and the program successfully compares both LTS sets using bisimilarity and declares itself correct.

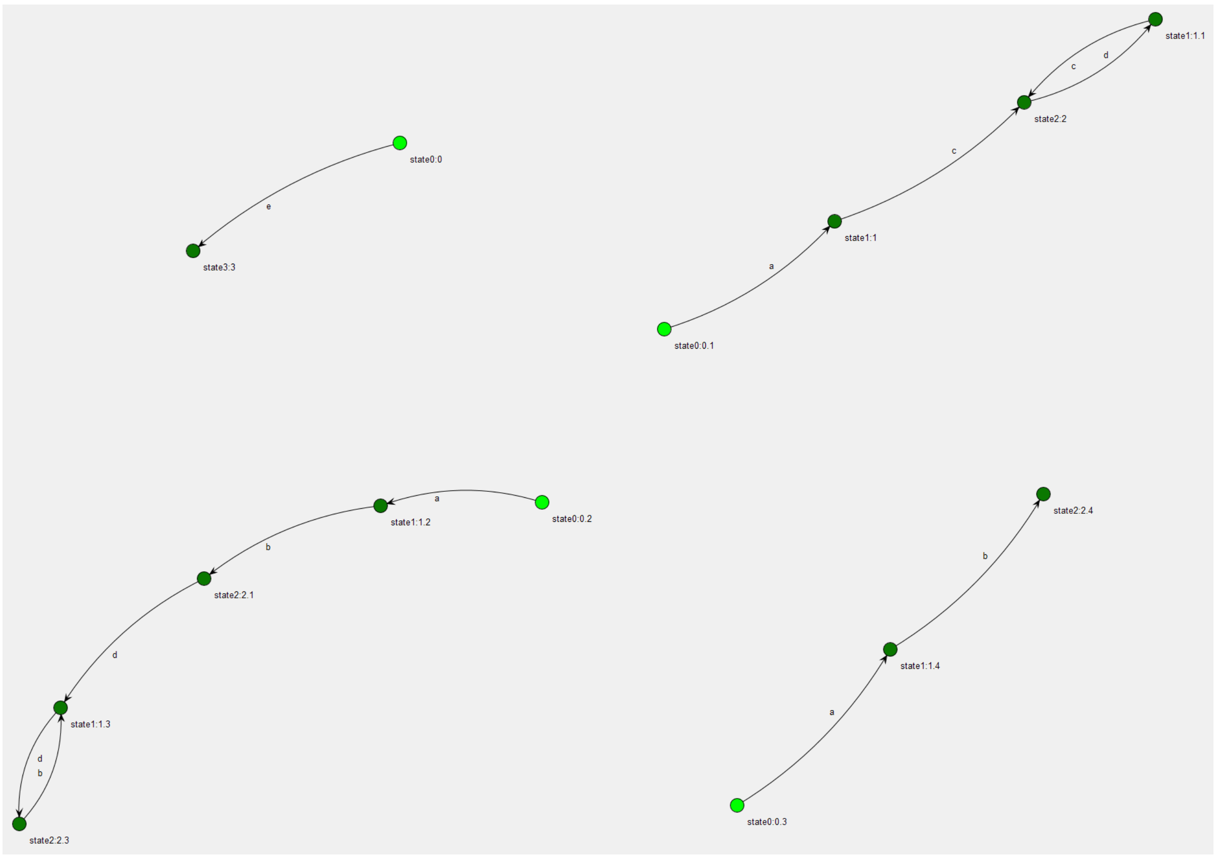
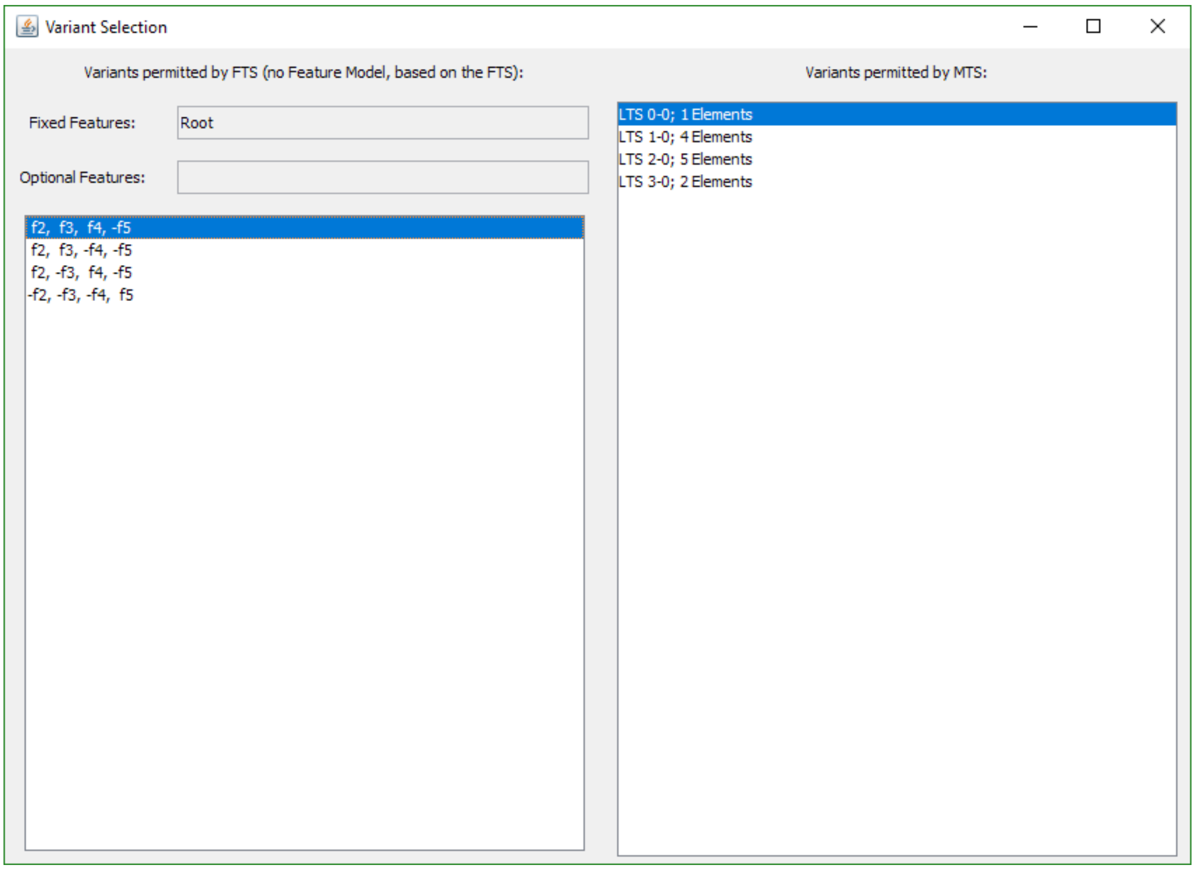


Figure 10: The generated MTS.


Variant Selection

Variants permitted by FTS (no Feature Model, based on the FTS):

Fixed Features: Root

Optional Features:

```
f2, f3, f4, -f5
f2, f3, -f4, -f5
f2, -f3, f4, -f5
-f2, -f3, -f4, f5
```

Variants permitted by MTS:

```
LTS 0-0; 1 Elements
LTS 1-0; 4 Elements
LTS 2-0; 5 Elements
LTS 3-0; 2 Elements
```

Figure 11: The variant selection window after the fact.

3 Appendix

3.1 The Exemplary Feature Model

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
        <featureModel>
                <properties/>
                <struct>
                        <alt abstract="true" mandatory="true" name="Root">
                                <or name="f1">
                                        <feature name="f2"/>
                                        <feature name="f3"/>
                                </or>
                                <feature name="f4"/>
                        </alt>
                </struct>
                <constraints/>
                <calculations Auto="true" Constraints="true" Features="true"
                        Redundant="true" Tautology="true"/>
                <comments/>
        <featureOrder userDefined="false"/>
</featureModel>
```