# Mutation testing using Featured Mutants Model: technical report

Xavier Devroey Gilles Perrouin

August 28, 2015

## 1 Featured Mutants Model

A featured mutant model is a couple: featured transition system (FTS) and feature diagram (FD). A FD represents all the possible valid products of a product line. It is usually (but not always) represented as a boolean expression over a set of variables called feature. A set of variable (set to true) satisfying this boolean expression is a valid product (also called valid configuration) of the product line. The semantic of a FD ($d$) is the set of valid products of the product line and is noted $[\![d]\!]$. A FTS is (basically) a transition system where transitions have been tagged with the set of products able to execute it. To compactly represent this set, we use feature expressions (boolean expressions over features).

**Definition 1** (Featured Transition System (FTS)). *A Featured Transition System (FTS) is a tuple* ($S$, $Act$, $trans$, $init$, $d$, $\gamma$)*, where:*

- *$S$ is a set of states;*
- *$Act$ is a set of actions;*
- *$trans \subseteq S \times Act \times$ is a transition relation such as the FTS is deterministic (with $(s_1, \alpha, s_2) \in trans$ sometimes noted $s_1 \xrightarrow{\alpha} s_2$);*
- *$d$ is a FD;*
- *$\gamma : trans \mapsto [\![d]\!] \mapsto \top, \bot$ is a function labelling specifying for each transition which valid product may execute it, this function is represented as a boolean expression over the features of $d$;*
- *$init : S \mapsto ([\![d]\!] \mapsto \top, \bot)$ a total function that indicates if a state $i \in S$ is an initial state for a product $p \in [\![d]\!]$, which allows one to model mutants that change the initial state of the system.*

**Definition 2** (Featured Mutant Model (FMM)). *A Featured Mutant Model is a couple ($fd_{fmm}$, $fts_{fmm}$) where $fd_{fmm}$ is a feature diagram representing a set of mutation operators applications and $fts_{fmm}$ is a FTS where the feature diagram ($d$) is $fd_{fmm}$.*

## 2 Mutation operators

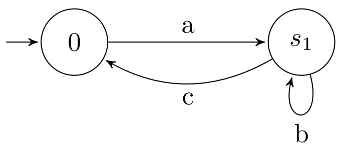
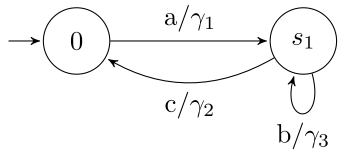
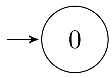
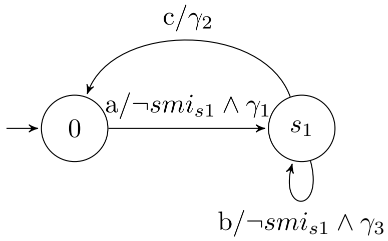
**Definition 3** (Mutation operator). *A mutation operator is a function: $op : ts \rightarrow ts_m$, where $ts$ and $ts'$ are transition systems. The operator $op$ performs a certain model transformation on $ts$ and produces $ts'$.*

**Definition 4** (FMM Mutation operator). *A FMM mutation operator is a function: $op_{fmm} : fmm \rightarrow fmm'$, where $fmm$ and $fmm'$ are FMM'. The operator $op$ performs a certain model transformation on $fmm$ and produces a $fmm'$ representing the modifications made by previously used operators and the modification made by $op_{fmm}$.*

Practically, a FMM operator will modify the FMM's FTS and add a feature in the FMM's FD representing the added transformation. In the following sections, we define mutation operators and FMM mutation operators, inspired from Fabbri et al. [Fabbri et al., 1999].

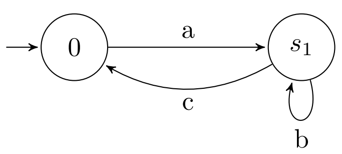
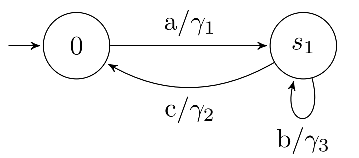
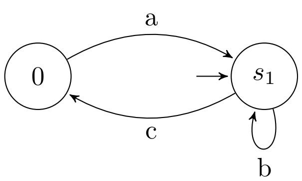
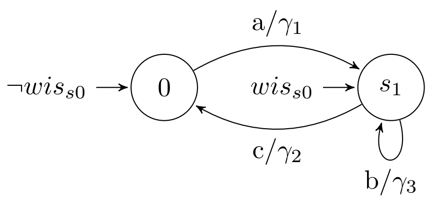
### 2.1 State Missing (SMI)

Removes a random state $s_i \neq i$ (except initial state). Let $ts$ a TS and $(fts_{fmm}, fd_{fmm})$ a FMM representing the previous mutations on $ts$. The result of the application $smi_{fmm}$ $(fts_{fmm}, fd_{fmm})$ is a FMM where $fts_{fmm}$ has all the feature expression of the input transitions of the state $s_i$ (the state removed in the mutant) conjunct with $\neg smi$.

<table>
<tr><td>Input $ts$<br></td><td>Input $(fts_{fmm}, fd_{fmm})$<br></td></tr>
<tr><td>$smi : ts \to ts'$</td><td>$smi_{fmm} : (fts_{fmm}, fd_{fmm}) \to (fts'_{fmm}, fd'_{fmm})$</td></tr>
<tr><td>Output $ts'$<br></td><td>Output $(fts'_m, fd_{fmm} \veebar smi_{s1})$<br></td></tr>
</table>

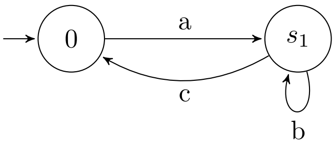
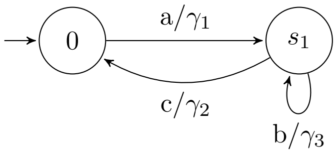
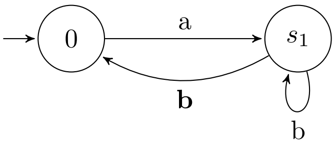
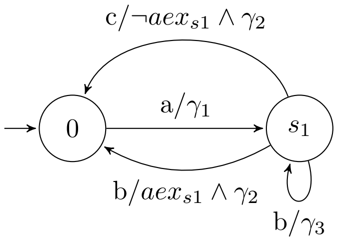
## 2.2 Wrong Initial State (WIS)

Modifies the start state of the transition system to another random state.

<table>
<tr><td>Input $ts$<br></td><td>Input $(fts_{fmm}, fd_{fmm})$<br></td></tr>
<tr><td>$wis : ts \to ts'$</td><td>$wis_{fmm} : (fts_{fmm}, fd_{fmm}) \to (fts'_{fmm}, fd'_{fmm})$</td></tr>
<tr><td>Output $ts'$<br></td><td>Output $(fts_{fmm}, fd_{fmm} \veebar wis_{s0})$<br></td></tr>
</table>

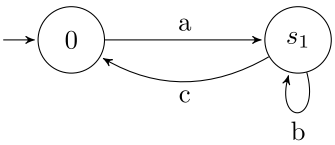
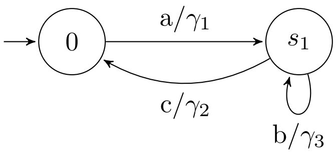
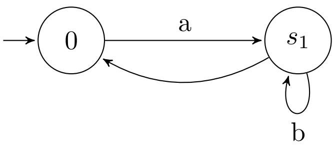
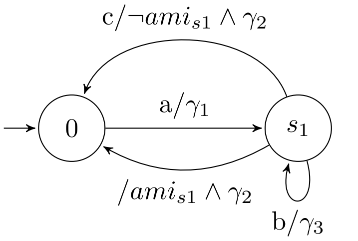
## 2.3 Action Exchange (AEX)

Modifies an action on a transition and replace it by another action.

<table>
<tr><td>Input $ts$<br>
</td><td>Input $fts_m$<br>
</td></tr>
<tr><td>$aex : ts \to ts'$</td><td>$aex_{fmm} : (fts_{fmm}, fd_{fmm}) \to (fts'_{fmm}, fd'_{fmm})$</td></tr>
<tr><td>Output $ts'$<br>
</td><td>Output $(fts_{fmm}, fd_{fmm} \veebar aex_{s1})$<br>
</td></tr>
</table>

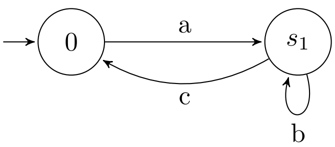
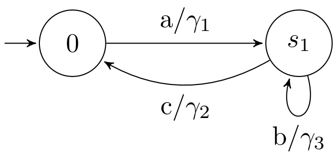
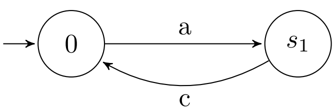
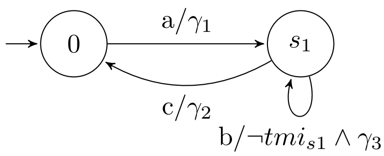
## 2.4 Action Missing (AMI)

Removes the actions from a transition.

<table>
<tr><td>Input $ts$<br>
</td><td>Input $(fts_{fmm}, fd_{fmm})$<br>
</td></tr>
<tr><td>$ami : ts \to ts'$</td><td>$ami_{fmm} : (fts_{fmm}, fd_{fmm}) \to (fts'_{fmm}, fd'_{fmm})$</td></tr>
<tr><td>Output $ts'$<br>
</td><td>Output $(fts_{fmm}, fd_{fmm} \veebar ami_{s1})$<br>
</td></tr>
</table>

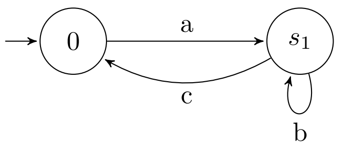
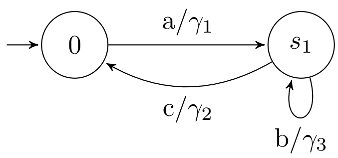
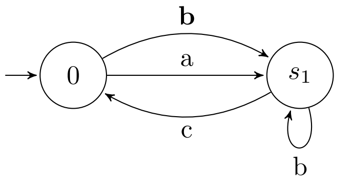
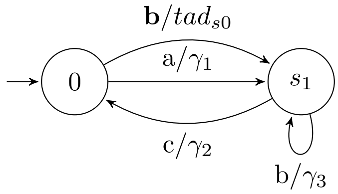
## 2.5 Transition Missing (TMI)

Removes a transition from the system.

<table>
<tr><td>Input $ts$<br>
</td><td>Input $(fts_{fmm}, fd_{fmm})$<br>
</td></tr>
<tr><td>$tmi : ts \to ts'$</td><td>$tmi_{fmm} : (fts_{fmm}, fd_{fmm}) \to (fts'_{fmm}, fd'_{fmm})$</td></tr>
<tr><td>Output $ts'$<br>
</td><td>Output $(fts_{fmm}, fd_{fmm} \veebar tmi_{s1})$<br>
</td></tr>
</table>

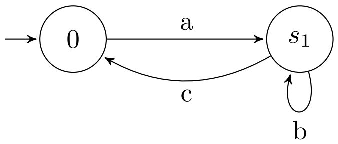
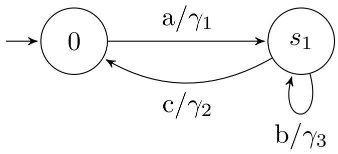
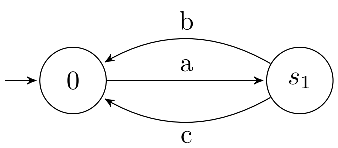
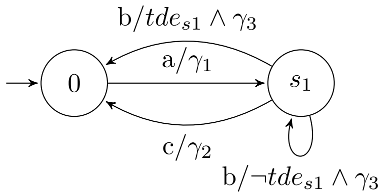
## 2.6 Transition Add (TAD)

Adds a transition to the system by randomly picking up two states and an action. Note: this corresponds to the event extra operator in [Fabbri et al., 1999]. Adding an action without adding a transition with this action has no sense since it can not be detected without being fired.

| | |
|---|---|
| **Input** $ts$<br><br> | **Input** $(fts_{fmm}, fd_{fmm})$<br><br> |
| $tad : ts \to ts'$ | $tad_{fmm} : (fts_{fmm}, fd_{fmm}) \to (fts'_{fmm}, fd'_{fmm})$ |
| **Output** $ts'$<br><br> | **Output** $(fts_{fmm}, fd_{fmm} \veebar tad_{s0})$<br><br> |

## 2.7 Transition Destination Exchange (TDE)

Changes the destination of a transition to the system by randomly picking up another state in the system.

| | |
|---|---|
| **Input** $ts$<br><br> | **Input** $(fts_{fmm}, fd_{fmm})$<br><br> |
| $tde : ts \to ts'$ | $tde_{fmm} : (fts_{fmm}, fd_{fmm}) \to (fts'_{fmm}, fd'_{fmm})$ |
| **Output** $ts'$<br><br> | **Output** $(fts_{fmm}, fd_{fmm} \veebar tde_{s1})$<br><br> |

# 3 Executing test case

Let $(fts_{fmm}, fd_{fmm})$ be a FMM of the original TS $ts$ and $tc = (\alpha_1, \alpha_2, ..., \alpha_n)$ be a test case over $ts$ starting from $i$ (the initial state of $ts$) and ending in $i$. The question is which mutants are not killed by $tc$ when executing it on $fts_{fmm}$? Let us consider:

- $trs = \{(t_k, ..., t_l), ...\}$ the set possible sequences of transitions fired when executing $tc$ on $fts_{fmm}$, where each sequence ends in $i_{ts}$, the initial state $i$ in $ts$ ;

Sequences in $trs$ return to the initial state, meaning that the original system *and* mutants with their feature expression activated by transitions in $trs$ may execute $tc$ (and are not killed by $trs$). The set of mutants not killed by $tc$ is defined using the feature expression :

$$alive = \left( \bigvee_{(t_k,...,t_l) \in trs} \left( \bigwedge_{t \in (t_k,...,t_l)} \gamma t \right) \right)$$

The alive feature expression is conjunct with the feature diagram $fd_{fmm}$ (representing all the possible mutants in $ts$) to compute the set of mutants not killed by a test case ($tc$).

## References

[Fabbri et al., 1999] Fabbri, S., Maldonado, J. C., and Delamaro, M. E. (1999). Proteum/FSM: a tool to support finite state machine validation based on mutation testing. In *Computer Science Society, 1999. Proceedings. SCCC '99. XIX International Conference of the Chilean*, pages 96–104.