Static Analysis of Featured Transition Systems

Maurice ter Beek  Ferruccio Damiani  Michael Lienhardt
Franco  Mazzanti  Luca Paolini

ISTI–CNR, Pisa  University of Turin  ONERA, France

SPLC 2019
Paris, France
09/11, 2019
Outline

1. Key idea and aim

2. Featured Transition Systems (FTSs)
   - definition and examples
   - ambiguous FTSs and transformation into unambiguous FTSs

3. Static analysis of FTSs
   - algorithm and experiments

4. Towards family-based model checking

5. Conclusion and Outlook
Key idea and aim

💡 Mimick anomaly detection known from feature model analysis in behavioral SPL models (FTSs) by automated static analysis:

1. dead transitions
2. false optional transitions
3. hidden deadlock states

Catch and offer means to remove possible ambiguities in FTSs:
1. Ambiguous FTSs undesired, as it gives an unclear idea of the SPL
2. Unambiguous FTSs pave way to efficient family-based verification
Key idea and aim

Mimick anomaly detection known from feature model analysis in behavioral SPL models (FTSs) by automated static analysis:

1. dead transitions
2. false optional transitions
3. hidden deadlock states

Catch and offer means to remove possible ambiguities in FTSs:

1. Ambiguous FTSs undesired, as it gives an unclear idea of the SPL
2. Unambiguous FTSs pave way to efficient family-based verification
A **Labeled Transition System (LTS)** is a quadruple \((S, \Sigma, s_0, \delta)\) with states \(S\), actions \(\Sigma\), initial state \(s_0\), and transitions \(\delta \subseteq S \times \Sigma \times S\).

![Diagram of a LTS]

An FTS adds to this a feature model and feature expressions:

A **Featured Transition System (FTS)** is a sextuple \((S, \Sigma, s_0, \delta, F, \Delta)\) with states \(S\), actions \(\Sigma\), initial state \(s_0\), (featured) transitions \(\delta \subseteq S \times \Sigma \times \mathbb{B}(F) \times S\), with Boolean (feature) expressions \(\mathbb{B}(F)\) over features \(F\), and (product) configurations \(\Lambda \subseteq \{ \lambda : F \to \mathbb{B} \}\).

LTS \(\mathcal{F}|_\lambda\) specified by configuration \(\lambda \in \Lambda\) is called a product of \(\mathcal{F}\) (remove: 1) all featured transitions whose feature expressions are not satisfied by \(\lambda\); 2) all unreachable states and their outgoing transitions).

Classen et al. @ ICSE'10, *IEEE TSE*, 2013
A **Labeled Transition System** (LTS) is a quadruple \((S, \Sigma, s_0, \delta)\) with states \(S\), actions \(\Sigma\), initial state \(s_0\), and transitions \(\delta \subseteq S \times \Sigma \times S\).

An FTS adds to this a feature model and feature expressions:

Classen *et al.* @ ICSE'10, *IEEE TSE*, 2013

A **Featured Transition System** (FTS) is a sextuple \((S, \Sigma, s_0, \delta, F, \Delta)\) with states \(S\), actions \(\Sigma\), initial state \(s_0\), (featured) transitions \(\delta \subseteq S \times \Sigma \times \mathbb{B}(F) \times S\), with Boolean (feature) expressions \(\mathbb{B}(F)\) over features \(F\), and (product) configurations \(\Lambda \subseteq \{ \lambda : F \rightarrow \mathbb{B} \}\).

LTS \(\mathcal{F}|\lambda\) specified by configuration \(\lambda \in \Lambda\) is called a product of \(\mathcal{F}\) (remove: 1) all featured transitions whose feature expressions are not satisfied by \(\lambda\); 2) all unreachable states and their outgoing transitions)
A Labeled Transition System (LTS) is a quadruple \((S, \Sigma, s_0, \delta)\) with states \(S\), actions \(\Sigma\), initial state \(s_0\), and transitions \(\delta \subseteq S \times \Sigma \times S\).

---

An FTS adds to this a feature model and feature expressions:

A Featured Transition System (FTS) is a sextuple \((S, \Sigma, s_0, \delta, F, \Delta)\) with states \(S\), actions \(\Sigma\), initial state \(s_0\), (featured) transitions \(\delta \subseteq S \times \Sigma \times \mathbb{B}(F) \times S\), with Boolean (feature) expressions \(\mathbb{B}(F)\) over features \(F\), and (product) configurations \(\Lambda \subseteq \{ \lambda : F \to \mathbb{B} \}\).

LTS \(\mathcal{F}|_{\lambda}\) specified by configuration \(\lambda \in \Lambda\) is called a product of \(\mathcal{F}\) (remove: 1) all featured transitions whose feature expressions are not satisfied by \(\lambda\); 2) all unreachable states and their outgoing transitions).
FTS of example SPL: a vending machine

Feature model:

```
VendingMachine
  ^
 / \                                    
Beverages         FreeDrinks              CancelPurchase
    \                                   /                                    
      \                               /                                    
Soda     Tea                         
  s     t
```

12 valid products

- \{v, b, s, t\}, \{v, b, s, c\}

12 valid products
FTS of example SPL: a vending machine

Feature model:

FTS of 12 valid products (LTS) e.g., \(\{v, b, s, t\}\), \(\{v, b, s, c\}\)
FTS of example SPL: a vending machine

Feature model:

```
VendingMachine

<table>
<thead>
<tr>
<th>Beverages</th>
<th>FreeDrinks</th>
<th>CancelPurchase</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>b</td>
<td>f</td>
</tr>
<tr>
<td>Soda</td>
<td>Tea</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>t</td>
<td></td>
</tr>
</tbody>
</table>
```

e.g., \{v,b,s,t\}, \{v,b,s,c\}
FTS of example SPL: a vending machine

Feature model:

```
VendingMachine
  ▲
 /  \
|  |
|  |
Beverages b
  ▲
 /  \
|  |
|  |
Soda s
  ▲
 /  \
|  |
|  |
Tea t
  ▲
 /  \
|  |
|  |
FreeDrinks f
  ▲
 /  \
|  |
|  |
CancelPurchase c
  ▲
 /  \
|  |
|  |
```

e.g., \{v,b,s,t\}, \{v,b,s,c\}
Ambiguous FTS

dead transition an FTS transition not reachable in any product (LTS)

false optional transition a featured FTS transition which is

1. not annotated with feature expression $\top$ (true, i.e., selected)
2. present in every FTS product in which its source state is reachable

hidden deadlock state an FTS state which is

1. not a deadlock (i.e., it has outgoing transitions) in the FTS
2. a deadlock (i.e., no outgoing transitions) in some FTS product(s)

Important safety property: deadlock freedom, i.e., system should not reach a state in which no further action is possible, thus guaranteeing progress or liveness; for configurable systems, this notion is extended to guaranteeing liveness for each system variant (product)
Ambiguous FTS

dead transition an FTS transition not reachable in any product (LTS)

false optional transition a featured FTS transition which is

1. not annotated with feature expression $\top$ (true, i.e., selected)
2. present in every FTS product in which its source state is reachable

hidden deadlock state an FTS state which is

1. not a deadlock (i.e., it has outgoing transitions) in the FTS
2. a deadlock (i.e., no outgoing transitions) in some FTS product(s)

Important safety property: deadlock freedom, i.e., system should not reach a state in which no further action is possible, thus guaranteeing progress or liveness; for configurable systems, this notion is extended to guaranteeing liveness for each system variant (product)
Ambiguous FTS

dead transition  an FTS transition not reachable in any product (LTS)

false optional transition  a featured FTS transition which is

1. not annotated with feature expression \( \top \) (true, i.e., selected)
2. present in every FTS product in which its source state is reachable

hidden deadlock state  an FTS state which is

1. not a deadlock (i.e., it has outgoing transitions) in the FTS
2. a deadlock (i.e., no outgoing transitions) in some FTS product(s)

Important safety property: deadlock freedom, i.e., system should not reach a state in which no further action is possible, thus guaranteeing progress or liveness; for configurable systems, this notion is extended to guaranteeing liveness for each system variant (product)
Ambiguous FTS $\rightarrow$ unambiguous FTS

Transformation:

1. remove dead transitions

2. turn false optional transitions into must transitions (i.e., labeled $\top$)

3. make hidden deadlock states $s$ explicit:
   3.1 add a deadlock state $s_\dagger \notin Q$
   3.2 $\forall s$: add a deadlock transition from $s$ to $s_\dagger$ labeled by $\dagger \notin \Sigma$ and by a feature expression that negates the disjunction of the feature expressions of all outgoing transitions of $s$
Transformation:

1. remove dead transitions

2. turn false optional transitions into must transitions (i.e., labeled $\top$)

3. make hidden deadlock states $s$ explicit:
   3.1 add a deadlock state $s \dagger \notin Q$
   3.2 $\forall s$: add a deadlock transition from $s$ to $s \dagger$ labeled by $\dagger \notin \Sigma$ and
      by a feature expression that negates the disjunction of the
      feature expressions of all outgoing transitions of $s$

Step (3) must be performed only for the hidden deadlock states that have not
yet become explicit deadlock states upon the removal of dead transitions in step (1)
Transformation:

1. remove dead transitions

2. turn false optional transitions into must transitions (i.e., labeled $\top$)

3. make hidden deadlock states $s$ explicit:
   
   3.1 add a deadlock state $s_\dagger \notin Q$

   3.2 $\forall s$: add a *deadlock transition* from $s$ to $s_\dagger$ labeled by $\dagger \notin \Sigma$ and by a feature expression that negates the disjunction of the feature expressions of all outgoing transitions of $s$
Ambiguous FTS $\rightarrow$ unambiguous FTS

Transformation:

1. remove dead transitions

2. turn false optional transitions into *must* transitions (i.e., labeled $\top$)

3. make hidden deadlock states $s$ explicit:
   3.1 add a deadlock state $s_\dagger \notin Q$
   3.2 $\forall s$: add a deadlock transition from $s$ to $s_\dagger$ labeled by $\dagger \notin \Sigma$ and by a feature expression that negates the disjunction of the feature expressions of all outgoing transitions of $s$

Step (3) must be performed only for the hidden deadlock states that have not yet become explicit deadlock states upon the removal of dead transitions in step (1)
Ambiguous FTS $\rightarrow$ unambiguous FTS

Transformation:

1. remove dead transitions

2. turn false optional transitions into must transitions (i.e., labeled $\top$)

3. make hidden deadlock states $s$ explicit:
   
   3.1 add a deadlock state $s_{\dagger} \notin Q$

   3.2 $\forall s$: add a deadlock transition from $s$ to $s_{\dagger}$ labeled by $\dagger \notin \Sigma$ and by a feature expression that negates the disjunction of the feature expressions of all outgoing transitions of $s$

---

Step (3) must be performed only for the hidden deadlock states that have not yet become explicit deadlock states upon the removal of dead transitions in step (1)
Example transformations

Feature Model: $f_1 \oplus f_2$

$$\mathcal{F}$$

$$\mathcal{F}'$$

$F|_{\lambda_1} = F'|_{\lambda_1}$

products $\lambda_1 = \{f_1\}$ and $\lambda_2 = \{f_2\}$
Example transformations

**Feature Model:** $f_1 \oplus f_2$

\[
\begin{align*}
\mathcal{F} &\xrightarrow{a|f_1} s_1 \xrightarrow{a|f_1} s_2 \\
&\xrightarrow{a|f_2} s_0
\end{align*}
\]

\[
\begin{align*}
\mathcal{F}' &\xrightarrow{a|T} s_1 \xrightarrow{a|f_1} s_2 \\
&\xrightarrow{a|f_2} s_0
\end{align*}
\]

products $\lambda_1 = \{f_1\}$ and $\lambda_2 = \{f_2\}$
Example transformations

Feature Model: $f_1 \oplus f_2$

$\mathcal{F}$

$\mathcal{F}'$

$\mathcal{F}|_{\lambda_1} = \mathcal{F}'|_{\lambda_1}$

products $\lambda_1 = \{f_1\}$ and $\lambda_2 = \{f_2\}$

$\mathcal{F}_*$

$\mathcal{F}'_*$
Static analysis algorithm

- Algorithm visits all cycle-free paths (starting in initial state) of FTS in depth-first order and in one unique FTS traversal identifies all ambiguities

- Based on formalization of criteria for ambiguities

- Criteria for ambiguities defined as deciding for a given Boolean formula if it is a tautology or not satisfiable, i.e., checking the criteria are variations of SAT solving

- We use Z3

- We prove its correctness in the paper
Static analysis algorithm

- Algorithm visits all cycle-free paths (starting in initial state) of FTS in depth-first order and in one unique FTS traversal identifies all ambiguities

- Based on formalization of criteria for ambiguities

- Criteria for ambiguities defined as deciding for a given Boolean formula if it is a tautology or not satisfiable, i.e., checking the criteria are variations of SAT solving

- We use Z3

- We prove its correctness in the paper

Python code publicly available, declared as reusable artifact
Static analysis algorithm

- Algorithm visits all cycle-free paths (starting in initial state) of FTS in depth-first order and in one unique FTS traversal identifies all ambiguities.

- Based on formalization of criteria for ambiguities.

- Criteria for ambiguities defined as deciding for a given Boolean formula if it is a tautology or not satisfiable, i.e., checking the criteria are variations of SAT solving.

- We use Z3.

- We prove its correctness in the paper.

Python code publicly available, declared as reusable artifact.
Example static analysis: vending machine

Feature Model: $s \lor t$

Result of static analysis on FTS

Vending Machine: live
LIVE STATES = [1, 2, 3, 4, 5, 6, 7, 8, 9]
DEAD TRANSITIONS = []
FALSE OPTIONAL TRANSITIONS = [(2, 3), (4, 1), (5, 7), (6, 7), (8, 9), (9, 1)]
HIDDEN DEADLOCK STATES = []
Example static analysis: mine pump (1/2)

Feature Model: \((c \leftrightarrow (ct \lor cp)) \land l\)
Result of static analysis on FTS

Mine Pump: not live
LIVE STATES = [S6, S7, S8, S9, S10, S11, S12, S13, S14, S15, S16, S17, S18, S19, S21, S22, S23, S24, S25, S26, S27, S28, S29, S30]
DEAD TRANSITIONS = []
FALSE OPTIONAL TRANSITIONS = [(S10, S11), (S11, S12), (S13, S14), (S13, S15, isReady), (S13, S15, isRunning), (S14, S15), (S16, S17), (S16, S18), (S17, S18), (S18, S19), (S21, S22, isReady), (S21, S26, isRunning), (S21, S26, isStopped), (S22, S23, setReady), (S23, S24), (S23, S26), (S24, S25), (S25, S26), (S27, S28), (S27, S30), (S28, S29), (S29, S30), (S7, S20), (S9, S10), (S9, S11)]
HIDDEN DEADLOCK STATES = [S20]
## Experiments

The experiments were performed on a virtual machine Gentoo 201905, CLI Version VirtualBox (VDI) 64 bit, with 2048 Mb of allocated memory on a Windows 10 Pro 64 bit with 16 Gb of RAM and CPU AMD Ryzen 7 1700X (8 core, 16 threads, 3.4 Ghz)

<table>
<thead>
<tr>
<th>FTS Model</th>
<th># states</th>
<th># transitions</th>
<th># actions</th>
<th>live-ness</th>
<th># dead transitions</th>
<th># false optional transitions</th>
<th># hidden deadlock states</th>
<th>computational effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vending machine</td>
<td>9</td>
<td>13</td>
<td>12</td>
<td>yes</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0.68 41</td>
</tr>
<tr>
<td>Coffee machine</td>
<td>14</td>
<td>22</td>
<td>14</td>
<td>yes</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>1.35 42</td>
</tr>
<tr>
<td>Mine pump (system)</td>
<td>25</td>
<td>41</td>
<td>22</td>
<td>no</td>
<td>0</td>
<td>25</td>
<td>1</td>
<td>1.41 44</td>
</tr>
<tr>
<td>Mine pump (controller)</td>
<td>77</td>
<td>104</td>
<td>22</td>
<td>no</td>
<td>0</td>
<td>59</td>
<td>4</td>
<td>5.37 48</td>
</tr>
<tr>
<td>Mine pump (complete)</td>
<td>417</td>
<td>1255</td>
<td>26</td>
<td>yes</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>timeout</td>
</tr>
</tbody>
</table>

The computational effort includes run-time (s) and memory use (Mb).
Note

1. any FTS $\mathcal{F}$ can trivially be transformed into an MTS

2. if the FTS is unambiguous, then the corresponding MTS is live

This allows us to carry over a result for MTSs to unambiguous FTSs:

Any formula $\phi$ of $\nu$-ACTLive is preserved by unambiguous FTSs: given an unambiguous FTS $\mathcal{F}$, whenever $\mathcal{F} \models \phi$, then $\mathcal{F}|_{\lambda} \models \phi$ for all products $\mathcal{F}|_{\lambda}$ of $\mathcal{F}$
Family-based model checking

Note

1. any FTS $\mathcal{F}$ can trivially be transformed into an MTS

2. if the FTS is unambiguous, then the corresponding MTS is live

This allows us to carry over a result for MTSs to unambiguous FTSs:

ter Beek et al. *JLAMP*, 2015

Any formula $\phi$ of $\nu$-ACTLive is preserved by unambiguous FTSs: given an unambiguous FTS $\mathcal{F}$, whenever $\mathcal{F} \models \phi$, then $\mathcal{F}|_\lambda \models \phi$ for all products $\mathcal{F}|_\lambda$ of $\mathcal{F}$
Family-based model checking

Note

1. any FTS $\mathcal{F}$ can trivially be transformed into an MTS

2. if the FTS is unambiguous, then the corresponding MTS is live

This allows us to carry over a result for MTSs to unambiguous FTSs:

ter Beek et al. JLAMP, 2015

Any formula $\phi$ of v-ACTLive$\square$ is preserved by unambiguous FTSs:
given an unambiguous FTS $\mathcal{F}$, whenever $\mathcal{F} \models \phi$, then $\mathcal{F}\mid_\lambda \models \phi$
for all products $\mathcal{F}\mid_\lambda$ of $\mathcal{F}$
Example v-ACTLive formulas that could now be verified with VMC:

1. \( \text{AG AF}_{\text{pay} \lor \text{free}} \top \): infinitely often, either action pay or action free is executed.
2. \( \text{AG [open] AF}_{\text{close}} \top \): it is always the case that the execution of action open is eventually followed by that of action close.
3. \( \text{AG AF}_{\text{cancel} \lor \text{serveSoda} \lor \text{serveTea}} \top \): infinitely often, either action cancel or action serveSoda or action serveTea is executed.
4. \( \neg \text{E } [\top \neg_{\text{tea}} \text{U }_{\text{serveTea}} \top] \): it is not possible that action serveTea is executed without being preceded by an execution of action tea.
5. \( [\text{pay}] \text{AF}_{\text{take} \lor \text{cancel}} \top \): whenever action pay is executed, eventually also either action take or action cancel is executed.

VMC v6.4 is freely usable online: http://fmt.isti.cnr.it/vmc/
Family-based model checking with VMC

Example v-ACTLive\(\square\) formulas that could now be verified with VMC:

\[\begin{align*}
1. \quad & \text{AG AF}_{\text{pay} \lor \text{free}} \top: \text{infinitely often, either action pay or action free is executed} \\
2. \quad & \text{AG [open] AF}_{\text{close}} \top: \text{it is always the case that the execution of action open is eventually followed by that of action close} \\
3. \quad & \text{AG AF}_{\text{cancel} \lor \text{serveSoda} \lor \text{serveTea}} \top: \text{infinitely often, either action cancel or action serveSoda or action serveTea is executed} \\
4. \quad & \neg E [\top \neg \text{tea} U_{\text{serveTea}} \top]: \text{it is not possible that action serveTea is executed without being preceded by an execution of action tea} \\
5. \quad & [\text{pay}] AF_{\text{take} \lor \text{cancel}} \top: \text{whenever action pay is executed, eventually also either action take or action cancel is executed}
\end{align*}\]

1. AG AF\textit{pay} \lor \textit{free} \top: infinitely often, either action \textit{pay} or action \textit{free} is executed
2. AG [\textit{open}] AF\textit{close} \top: it is always the case that the execution of action \textit{open} is eventually followed by that of action \textit{close}
3. AG AF\textit{cancel} \lor \textit{serveSoda} \lor \textit{serveTea} \top: infinitely often, either action \textit{cancel} or action serve\textit{Soda} or action serve\textit{Tea} is executed
4. \neg E [\top \neg \textit{tea} U_{\textit{serveTea}} \top]: it is not possible that action serve\textit{Tea} is executed without being preceded by an execution of action \textit{tea}
5. [\textit{pay}] AF_{\textit{take} \lor \textit{cancel}} \top: whenever action \textit{pay} is executed, eventually also either action \textit{take} or action \textit{cancel} is executed

VMC v6.4 is freely usable online: \url{http://fmt.isti.cnr.it/vmc/}
Example \( \nu\text{-ACTLive} \) formulas that could now be verified with VMC:

1. \( \text{AG AF}_{\text{pay} \lor \text{free}} \top \): infinitely often, either action pay or action free is executed
2. \( \text{AG} [\text{open}] \text{AF}_{\text{close}} \top \): it is always the case that the execution of action open is eventually followed by that of action close
3. \( \text{AG AF}_{\text{cancel} \lor \text{serveSoda} \lor \text{serveTea}} \top \): infinitely often, either action cancel or action serveSoda or action serveTea is executed
4. \( \neg \text{E} [\top \neg_{\text{tea}} U_{\text{serveTea}} \top] \): it is not possible that action serveTea is executed without being preceded by an execution of action tea
5. \( [\text{pay}] \text{AF}_{\text{take} \lor \text{cancel}} \top \): whenever action pay is executed, eventually also either action take or action cancel is executed

VMC v6.4 is freely usable online: http://fmt.isti.cnr.it/vmc/
Conclusion and Outlook

Static analysis of FTSs:
1. effective algorithm
2. proof of correctness
3. example applications
4. python code available

Future work:
1. scalability concerns: investigate optimizations of algorithm (e.g., modular analysis, heuristics)
2. evolve current prototype into tool for static analysis of FTSs
   2.1 input standard specification format
   2.2 automatically disambiguate them
   2.3 apply v-ACTLive model checking
Conclusion and Outlook

Static analysis of FTSs:
1. effective algorithm
2. proof of correctness
3. example applications
4. python code available

Future work:
1. scalability concerns: investigate optimizations of algorithm (e.g., modular analysis, heuristics)
2. evolve current prototype into tool for static analysis of FTSs
   2.1 input standard specification format
   2.2 automatically disambiguate them
   2.3 apply v-ACTLive\textsuperscript{□} model checking