fmt.isti.cnr.it

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:

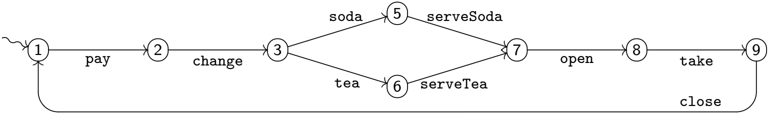
1. dead transitions
2. false optional transitions
3. hidden deadlock states

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

fmt.isti.cnr.it

A Labeled Transition System (LTS) is a quadruple (S, Σ, s_0, δ) with states S, actions Σ, initial state s_0, and transitions $\delta \subseteq S \times \Sigma \times S$



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

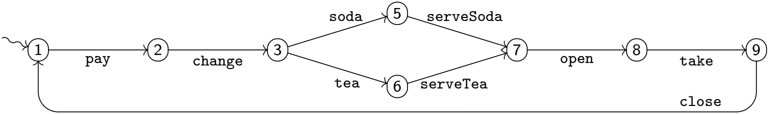
Classen *et al.* @ ICSE'10, *IEEE TSE*, 2013

A Featured Transition System (FTS) is a sextuple $(S, \Sigma, s_0, \delta, F, \Lambda)$ with states S, actions Σ, 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 λ; 2) all unreachable states and their outgoing transitions)

Featured Transition Systems (FTS)

A Labeled Transition System (LTS) is a quadruple (S, Σ, s_0, δ) with states S, actions Σ, initial state s_0, and transitions $\delta \subseteq S \times \Sigma \times S$



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

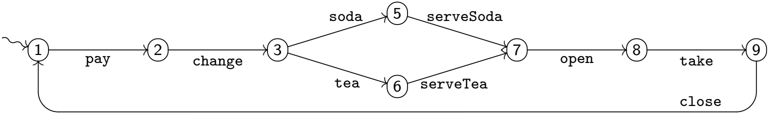
Classen *et al.* @ ICSE'10, ***IEEE TSE***, 2013

A Featured Transition System (FTS) is a sextuple $(S, \Sigma, s_0, \delta, F, \Delta)$ with states S, actions Σ, 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 λ; 2) all unreachable states and their outgoing transitions)

Featured Transition Systems (FTS)

A Labeled Transition System (LTS) is a quadruple (S, Σ, s_0, δ) with states S, actions Σ, initial state s_0, and transitions $\delta \subseteq S \times \Sigma \times S$



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

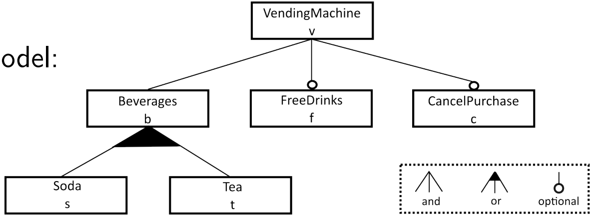
Classen *et al.* @ ICSE'10, *IEEE TSE*, 2013

A Featured Transition System (FTS) is a sextuple $(S, \Sigma, s_0, \delta, F, \Delta)$ with states S, actions Σ, 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 λ; 2) all unreachable states and their outgoing transitions)

FTS of example SPL: a vending machine

Feature model:

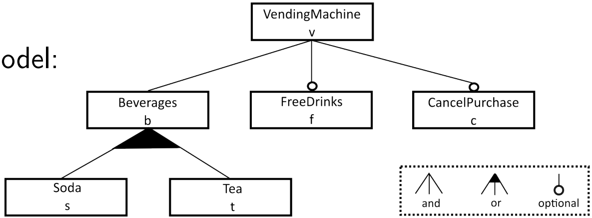


12 valid products

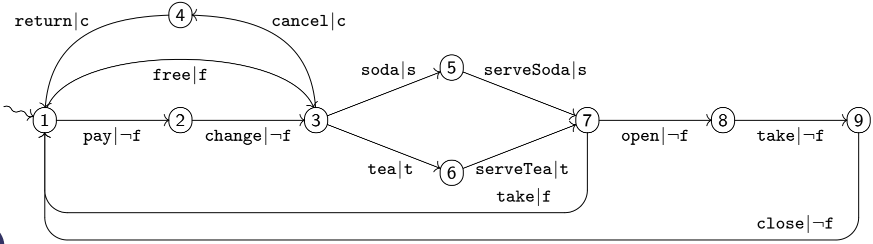
e.g., $\{\mathtt{v,b,s,t}\}$, $\{\mathtt{v,b,s,c}\}$

FTS of example SPL: a vending machine

Feature model:

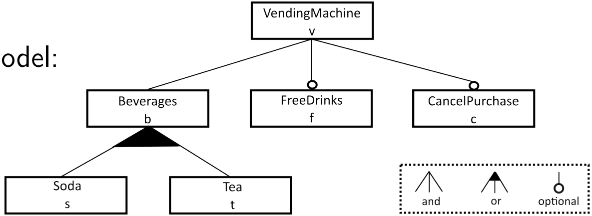


FTS of 12 valid products (LTS) e.g., {v,b,s,t}, {v,b,s,c}

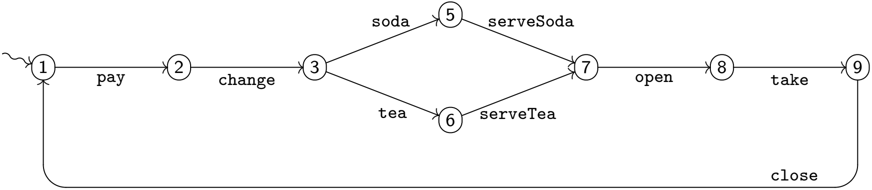


FTS of example SPL: a vending machine

Feature model:

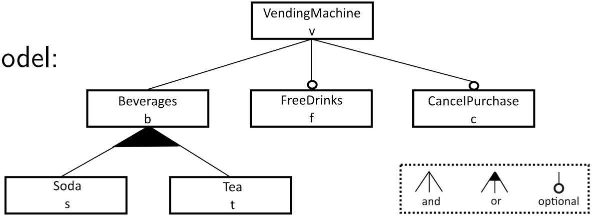


e.g., $\{v,b,s,t\}$, $\{v,b,s,c\}$

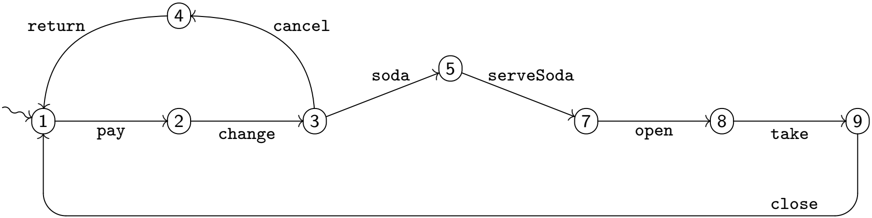


FTS of example SPL: a vending machine

Feature model:



e.g., $\{\mathtt{v},\mathtt{b},\mathtt{s},\mathtt{t}\}$, $\{\mathtt{v},\mathtt{b},\mathtt{s},\mathtt{c}\}$



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

fmt.isti.cnr.it

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)

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

---

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

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 $\longrightarrow$ 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)

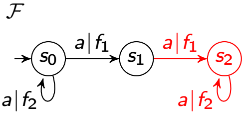
Transformation:

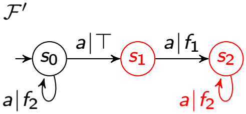
1. remove dead transitions
2. turn false optional transitions into *must* transitions (i.e., labeled $\top$)
3. make hidden deadlock states s explicit:
   3.1 add a deadlock state $s_\dagger \notin Q$
   3.2 $\forall s$: add a *deadlock transition* from s to $s_\dagger$ labeled by $\dagger \notin \Sigma$ 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

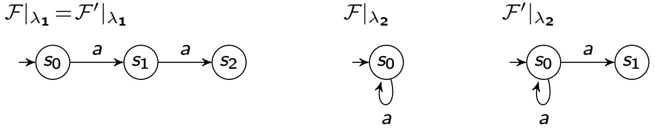
Feature Model: $f_1 \oplus f_2$





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

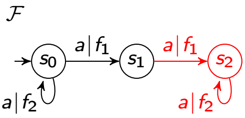
Feature Model: $f_1 \oplus f_2$

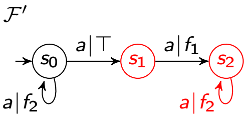


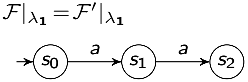
products $\lambda_1 = \{f_1\}$ and $\lambda_2 = \{f_2\}$

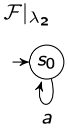
Example transformations

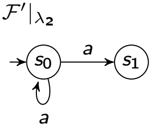
Feature Model: $f_1 \oplus f_2$



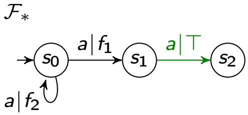


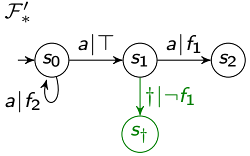






products $\lambda_1 = \{f_1\}$ and $\lambda_2 = \{f_2\}$





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

Static analysis algorithm

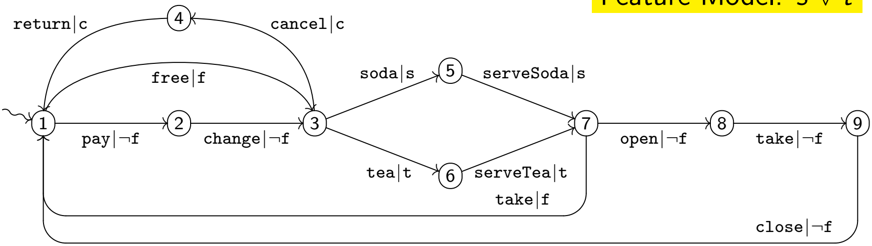
fmt.isti.cnr.it

- 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: $s \vee t$

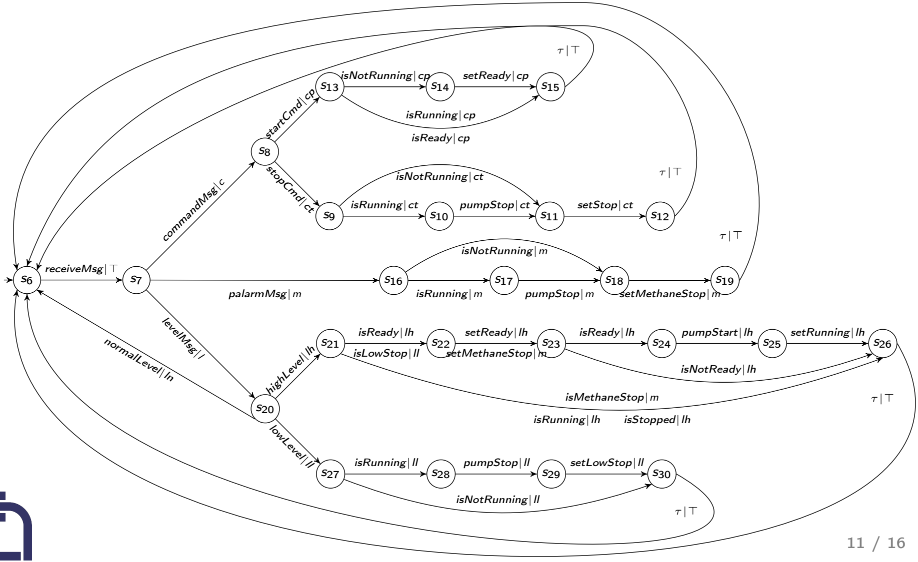


Result of static analysis on FTS

```
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: $(c \leftrightarrow (ct \vee cp)) \wedge l$



Example static analysis: mine pump (2/2)

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

FTS	characteristics			results of static analysis				computational effort	
Model	# states	# transitions	# actions	liveness	# dead transitions	# false optional transitions	# hidden deadlock states	runtime (s)	memory use (Mb)
Vending machine	9	13	12	yes	0	6	0	0.68	41
Coffee machine	14	22	14	yes	0	4	0	1.35	42
Mine pump (system)	25	41	22	no	0	25	1	1.41	44
Mine pump (controller)	77	104	22	no	0	59	4	5.37	48
Mine pump (complete)	417	1255	26	yes	?	?	?	timeout	–

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)

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 ϕ of v-ACTLive$^{\Box}$ 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 ϕ of v-ACTLive$^{\Box}$ 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 $\mathcal{F}$

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 ϕ of v-ACTLive$^{\Box}$ 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 $\mathcal{F}$

Example v-ACTLive$^{\Box}$ formulas that could now be verified with VMC:

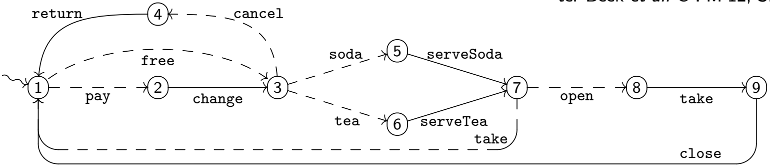
ter Beek *et al.* @ FM'12, SPLC'14

1. $AG\ AF_{pay \vee free}\ \top$: *infinitely often, either action pay or action free is executed*
2. $AG\ [open]\ AF_{close}\ \top$: *it is always the case that the execution of action open is eventually followed by that of action close*
3. $AG\ AF_{cancel \vee serveSoda \vee serveTea}\ \top$: *infinitely often, either action cancel or action serveSoda or action serveTea is executed*
4. $\neg E\ [\top_{\neg tea}\ U_{serveTea}\ \top]$: *it is not possible that action serveTea is executed without being preceded by an execution of action tea*
5. $[pay]\ AF_{take \vee cancel}\ \top$: *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/

Example v-ACTLive$^{\square}$ formulas that could now be verified with VMC:

ter Beek *et al.* @ FM'12, SPLC'14

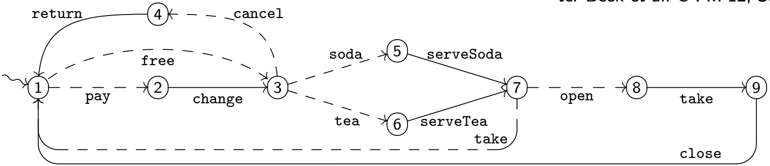


1. $AG\ AF_{pay \vee free}\ \top$: *infinitely often, either action pay or action free is executed*
2. $AG\ [open]\ AF_{close}\ \top$: *it is always the case that the execution of action open is eventually followed by that of action close*
3. $AG\ AF_{cancel \vee serveSoda \vee serveTea}\ \top$: *infinitely often, either action cancel or action serveSoda or action serveTea is executed*
4. $\neg E\ [\top_{\neg tea} U_{serveTea}\ \top]$: *it is not possible that action serveTea is executed without being preceded by an execution of action tea*
5. $[pay]\ AF_{take \vee cancel}\ \top$: *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/

Example v-ACTLive$^\Box$ formulas that could now be verified with VMC:

ter Beek *et al.* @ FM'12, SPLC'14



1. $AG\ AF_{pay \vee free}\ \top$: *infinitely often, either action pay or action free is executed*
2. $AG\ [open]\ AF_{close}\ \top$: *it is always the case that the execution of action open is eventually followed by that of action close*
3. $AG\ AF_{cancel \vee serveSoda \vee serveTea}\ \top$: *infinitely often, either action cancel or action serveSoda or action serveTea is executed*
4. $\neg E\ [\top_{\neg tea} U_{serveTea}\ \top]$: *it is not possible that action serveTea is executed without being preceded by an execution of action tea*
5. $[pay]\ AF_{take \vee cancel}\ \top$: *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$^{\square}$ 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$^{\square}$ model checking