

Efficient Static Analysis and Verification of Featured Transition Systems

Maurice ter Beek
Franco Mazzanti

Ferruccio Damiani
Luca Paolini

Michael Lienhardt

ISTI-CNR, Pisa

University of Turin

ONERA, France

Quantitative Modelling and Analysis of Variability in Highly
~~Informatics Seminars 2020/21~~

~~Leicester, UK~~

Configurable Systems

~~March 5, 2021~~

PhD in Smart Computing
July 14, 2021



- 1 Background
 - software product lines
 - variability modelling and analysis
- 2 Key idea and aim
- 3 Featured Transition Systems (FTSs)
 - definition and examples
 - ambiguous FTSs and disambiguation
- 4 Efficient static analysis of FTSs
 - algorithm and experiments
- 5 Family-based model checking on FTSs
 - branching-time properties with VMC
 - linear-time properties with SPIN
 - implemented in FTS4VMC
- 6 Conclusion and Outlook



- Configurable (software) system whose variants (products) differ by the provided **features**, i.e. the functionality that is relevant for an end-user

Configure your BMW vehicle

http://www.bmw.com/com/en/general/carconfigurator/content.html

BMW dealer Brochures Corporate/Direct Sales Shop BMW Financial Services Used Vehicles Search

Home 1 2 3 4 5 6 7 X Z4 BMW M BMW i BMW Owners BMW Insights

Configure vehicle

The international BMW website Sheer Driving Pleasure

Configure your BMW vehicle

Are you interested in configuring your ideal BMW? Please select a country to visit the configurator in the Virtual Center or contact your local BMW dealer who will be happy to answer all your questions about the BMW model you are interested in.

Related topics



Request information
Order product catalogues,
brochures and equipment
lists direct from BMW.



Filter

> Reset filter

+ Budget

- Vehicle type

- All
- Petrol
- Diesel
- Hybrid
- Electric Vehicle

- Body type

- Saloon
- Touring
- Convertible
- Coupé
- Gran Turismo
- Sports Hatch
- Roadster
- Sports Activity Coupé
- Sports Activity Vehicle

Number of seats

30 Vehicles **465 Model variants**

1



BMW 1 Series 3-door Sports Hatch (34)
from £ 17,775.00



BMW 1 Series 5-door Sports Hatch (39)
from £ 18,305.00

2



BMW 2 Series Coupé (14)
from £ 24,265.00

3



BMW 3 Series Saloon (56)
from £ 23,550.00



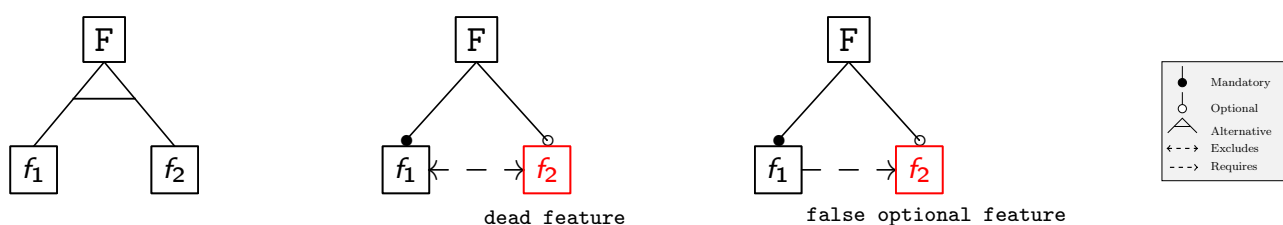
BMW 3 Series Touring (54)
from £ 24,865.00



BMW 3 Series Gran Turismo (39)
from £ 29,200.00

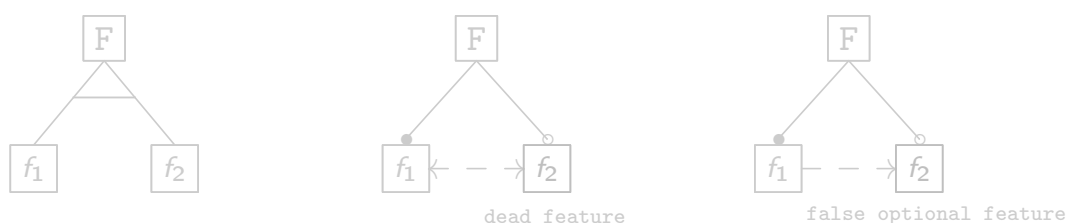


- Configurable (software) system whose variants (products) differ by the provided features, i.e. the functionality that is relevant for an end-user



Benavides *et al.*, Automated Analysis of Feature Models 20 Years Later: a Literature Review. *Inf. Syst.*, 2010

- Configurable (software) system whose variants (products) differ by the provided features, i.e. the functionality that is relevant for an end-user



Benavides *et al.*, Automated Analysis of Feature Models 20 Years Later: a Literature Review. *Inf. Syst.*, 2010

- Popular in embedded and critical systems domain: formal modelling and analysis techniques for proving SPL **behaviour** correct are widely studied
Thüm *et al.*, A classification and survey of analysis strategies for SPLs. *ACM Comput. Surv.*, 2014
- Challenge existing formal methods and tools by potentially high number of different variants, each giving rise to a large state space, in general

33 optional, independent
features



a unique configuration/variant for every

person on this planet

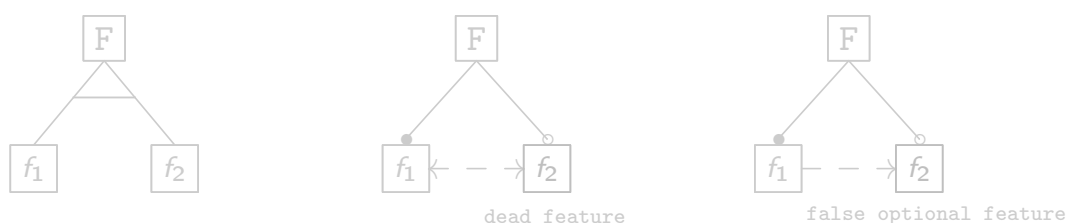


320^{optional, independent} features

more configurations/variants than estimated

atoms in the universe

- Configurable (software) system whose variants (products) differ by the provided features, i.e. the functionality that is relevant for an end-user





Benavides *et al.*, Automated Analysis of Feature Models 20 Years Later: a Literature Review. *Inf. Syst.*, 2010

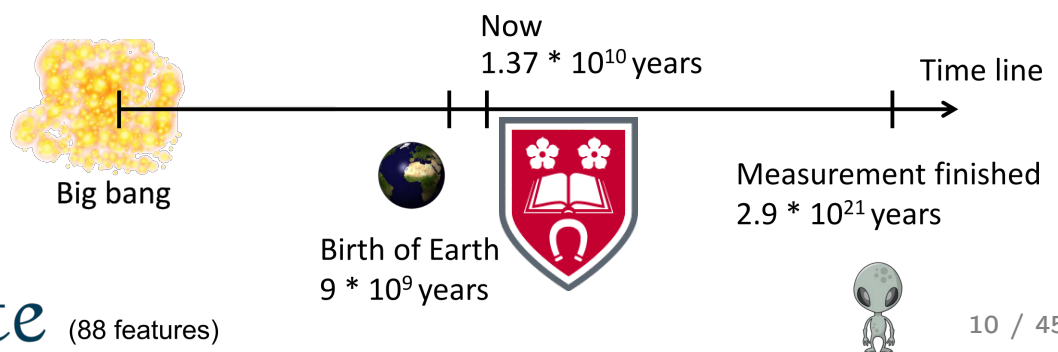
- Popular in embedded and critical systems domain: formal modelling and analysis techniques for proving SPL behaviour correct are widely studied
Thüm *et al.*, A classification and survey of analysis strategies for SPLs. *ACM Comput. Surv.*, 2014
- Challenge existing formal methods and tools by potentially high number of different variants, each giving rise to a large state space, in general

⇒ Lift success stories known for single systems (products) to sets of products (families) by exploiting **variability** modelling and analysis



-  Simple, brute-force approach
-  Make use of standardly available, highly optimised analysis tools







- 👍 Simple, brute-force approach
- 👍 Make use of standardly available, highly optimised analysis tools
- 👎 Number of product variants is exponential in number of features
- 👎 Same piece of behaviour (or code) is verified numerous times, as many times as the number of variants that are able to execute it



- 👍 Beneficial in case of many products with substantial similarities
- 👍 Same piece of behaviour (or code) is verified only once, regardless of how many variants can produce it

- 👍 Beneficial in case of many products with substantial similarities
- 👍 Same piece of behaviour (or code) is verified only once, regardless of how many variants can produce it
- 👉 More complex analysis tasks
- 👉 Requires (compact) family models (superimposed, *150% models*)

- 👍 Beneficial in case of many products with substantial similarities
- 👍 Same piece of behaviour (or code) is verified only once, regardless of how many variants can produce it
- 👍 More complex analysis tasks
- 👍 Requires (compact) family models (superimposed, *150% models*)
- 💡 Dedicated **variability-based** models, logics, and model checkers (e.g. FTSs, Feature Nets, PL-CCS, fLTL, fCTL, v-ACTL, QFLan, SNIP, VMC)

-  Beneficial in case of many products with substantial similarities
-  Same piece of behaviour (or code) is verified only once, regardless of how many variants can produce it
-  More complex analysis tasks
-  Requires (compact) family models (superimposed, *150% models*)
-  Dedicated **variability-based** models, logics, and model checkers (e.g. FTSs, Feature Nets, PL-CCS, fLTL, fCTL, v-ACTL, QFLan, SNIP, VMC)
-  Dedicated model checkers need to be maintained and optimised

- 👍 Beneficial in case of many products with substantial similarities
- 👍 Same piece of behaviour (or code) is verified only once, regardless of how many variants can produce it
- 👍 More complex analysis tasks
- 👎 Requires (compact) family models (superimposed, *150% models*)
- 💡 Dedicated **variability-based** models, logics, and model checkers (e.g. FTSs, Feature Nets, PL-CCS, fLTL, fCTL, v-ACTL, QFLan, SNIP, VMC)
- 👎 Dedicated model checkers need to be maintained and optimised

Dimovski et al., Family-based model checking without a family-based model checker @ SPIN'15

Chrszon et al., Family-based modeling and analysis for probabilistic systems: featuring ProFeat @ FASE'16

ter Beek et al., Family-Based Model Checking with mCRL2 @ FASE'17

Dimovski et al., Variability-specific Abstraction Refinement for Family-based Model Checking @ FASE'17

Dimovski, Abstract Family-Based Model Checking Using Modal Featured Transition Systems @ FASE'18

ter Beek et al., Family-Based SPL Model Checking Using Parity Games with Variability @ FASE'20

...



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

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



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

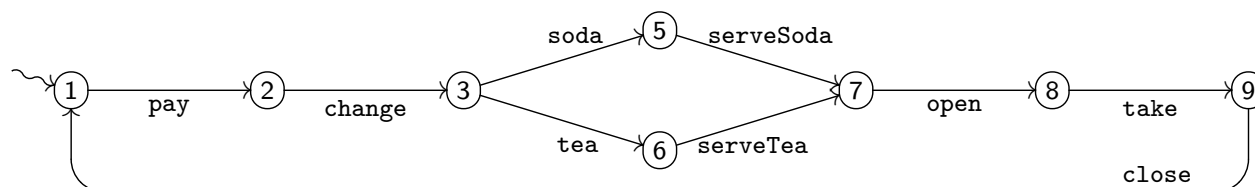
1. dead transitions
2. false optional transitions
3. hidden deadlock states



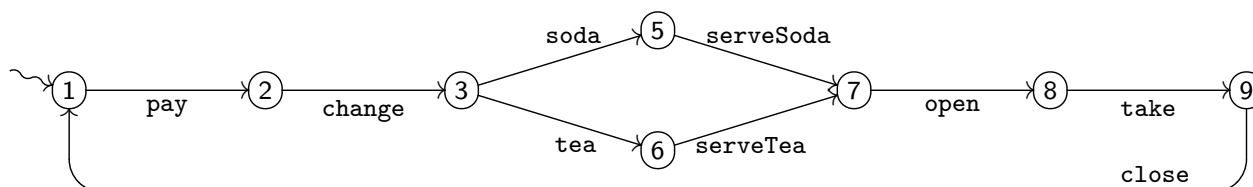
Catch and offer means to remove possible ambiguities in FTSs:

1. Ambiguous FTSs are undesired: they give unclear ideas of the SPLs
2. Unambiguous FTSs pave way to efficient family-based verification

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



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

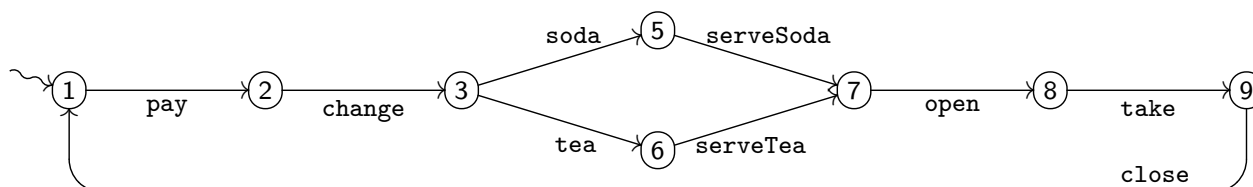


An FTS adds to this a feature model and feature expressions:

Classen *et al.*, Model checking lots of systems @ ICSE'10, FTSs. *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 $\Lambda \subseteq \{ \lambda : F \rightarrow \mathbb{B} \}$

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



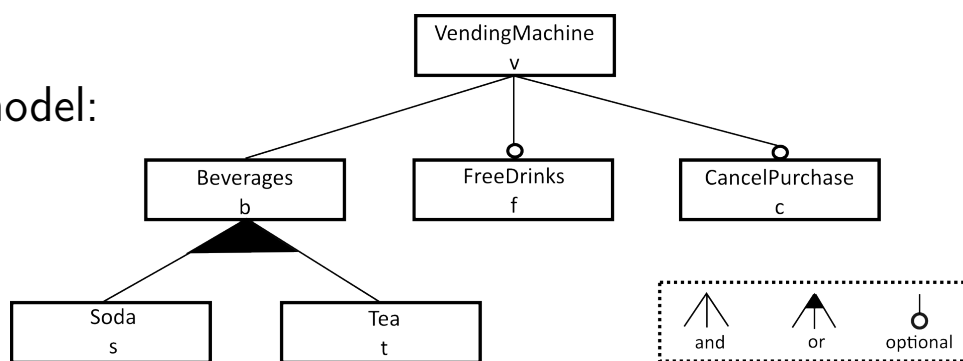
An FTS adds to this a feature model and feature expressions:

Classen *et al.*, Model checking lots of systems @ ICSE'10, FTSS. *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 $\Lambda \subseteq \{ \lambda : F \rightarrow \mathbb{B} \}$

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 λ ; 2) all unreachable states and their outgoing transitions]

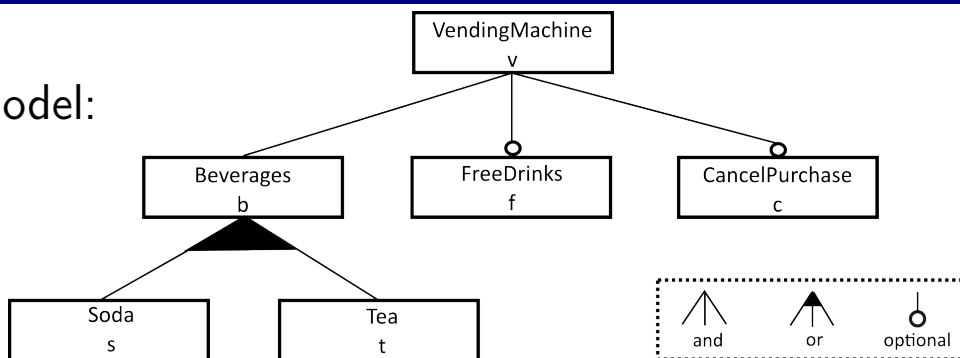
Feature model:



12 valid products

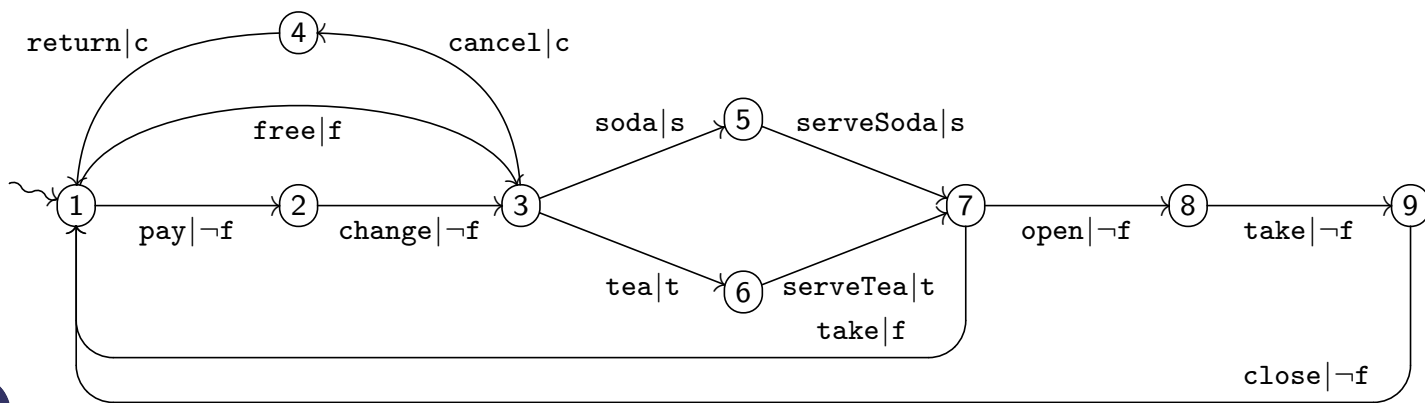
e.g., {v,b,s,t}, {v,b,s,c}

Feature model:

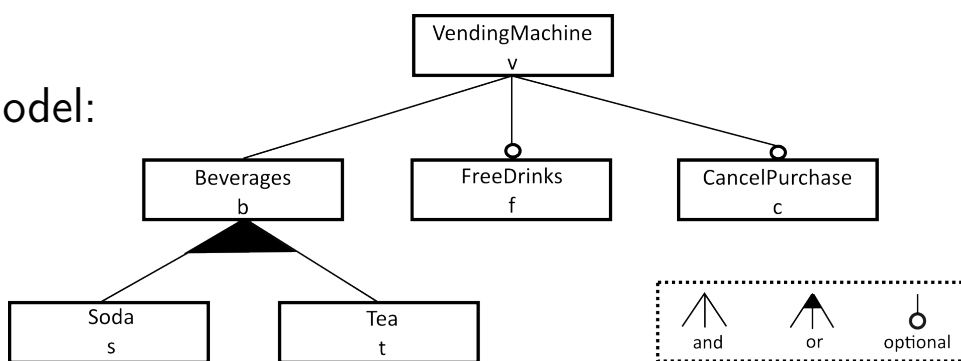


FTS of 12 valid products (LTSs)

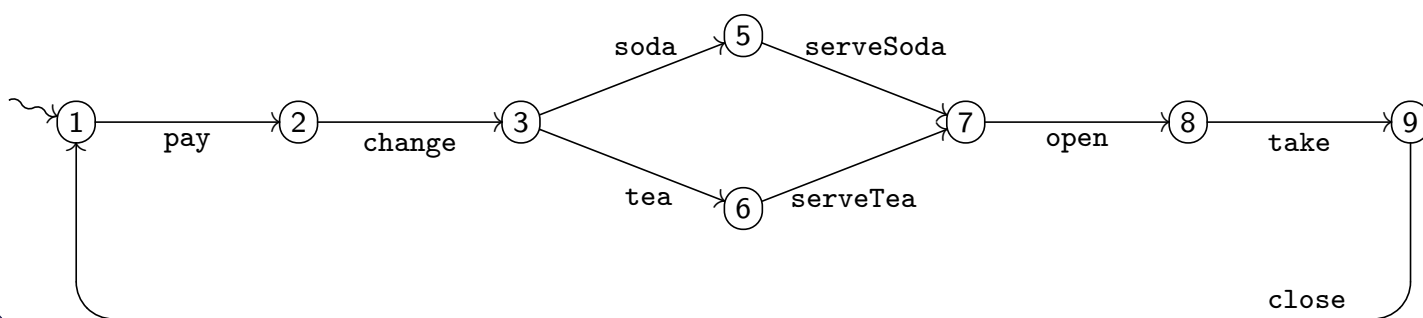
e.g., {v,b,s,t}, {v,b,s,c}



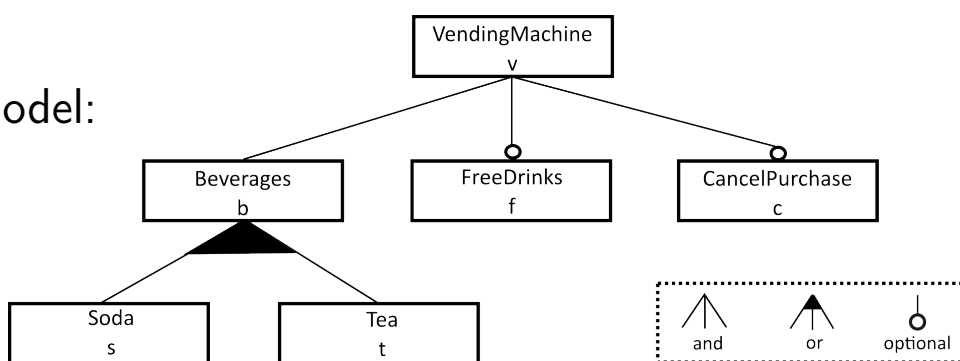
Feature model:



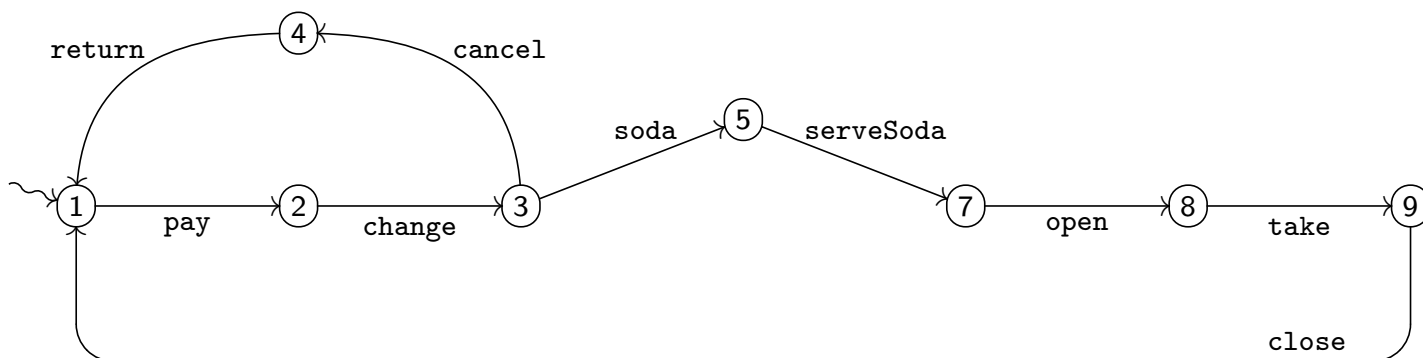
e.g., {v,b,s,t}, {v,b,s,c}



Feature model:



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





Ambiguous FTS



fmt.isti.cnr.it

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



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

false optional transition a featured FTS transition which is

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

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

false optional transition a featured FTS transition which is

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

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 \longrightarrow unambiguous FTS



Transformation:

1. remove dead transitions



Transformation:

1. remove dead transitions
2. turn false optional transitions into *must* transitions (i.e., labelled \top)

Transformation:

1. remove dead transitions
2. turn false optional transitions into *must* transitions (i.e., labelled \top)
3. make hidden deadlock states s explicit:



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., labelled \top)
3. make hidden deadlock states s explicit:
 - 3.1 add a deadlock state $s_{\dagger} \notin S$



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)

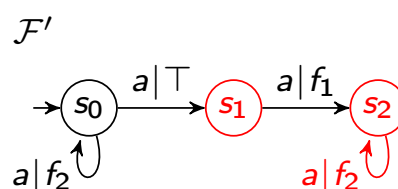
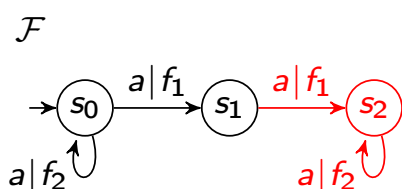
Transformation:

1. remove dead transitions
2. turn false optional transitions into *must* transitions (i.e., labelled \top)
3. make hidden deadlock states s explicit:
 - 3.1 add a deadlock state $s_{\dagger} \notin S$
 - 3.2 $\forall s$: add a *deadlock transition* from s to s_{\dagger} labelled by $\dagger \notin \Sigma$ and by a feature expression that negates the disjunction of the feature expressions of all outgoing transitions of s

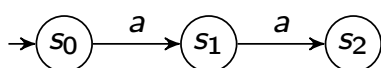


Step (3) must be performed only for the hidden deadlock states that have not yet become explicit deadlock states upon the removal of dead transitions in step (1)

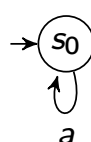
Feature Model: $f_1 \oplus f_2$



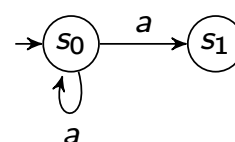
$$\mathcal{F}|_{\lambda_1} = \mathcal{F}'|_{\lambda_1}$$



$$\mathcal{F}|_{\lambda_2}$$

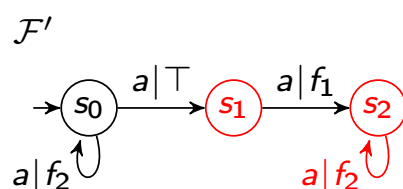
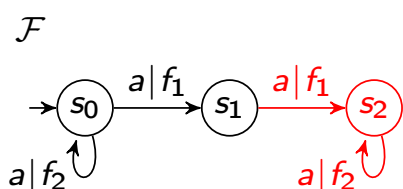


$$\mathcal{F}'|_{\lambda_2}$$

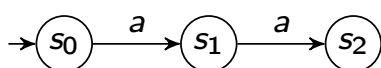


products $\lambda_1 = \{f_1\}$ and $\lambda_2 = \{f_2\}$

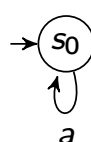
Feature Model: $f_1 \oplus f_2$



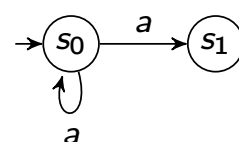
$\mathcal{F}|_{\lambda_1} = \mathcal{F}'|_{\lambda_1}$



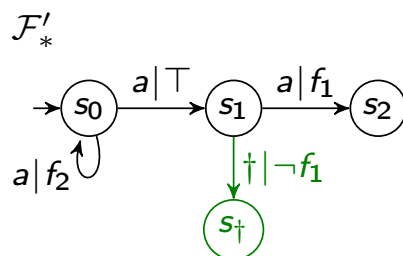
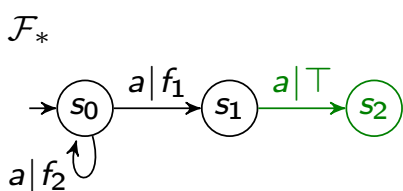
$\mathcal{F}|_{\lambda_2}$



$\mathcal{F}'|_{\lambda_2}$



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



- Based on expressing criteria for ambiguities as propositional formulae, thus reducing ambiguity detection to **SAT solving**

- Based on expressing criteria for ambiguities as propositional formulae, thus reducing ambiguity detection to **SAT solving**

```
# fts contains the input FTS
for s in fts.states:
    if(s.out_trs = ∅):
        s.hdead ← False
    else:
        s.hdead ← check(exist_deadlock(s))
# for all states s, it holds that:
# (s.hdead ≡ "s is a hidden deadlock")
```

- Based on expressing criteria for ambiguities as propositional formulae, thus reducing ambiguity detection to **SAT solving**

```

# fts contains the input FTS

for s in fts.states:
    if(s.out_trs = ∅):
        s.hdead ← False
    else:
        s.hdead ← check(exist_deadlock(s))
# for all states s, it holds that:
# (s.hdead ≡ "s is a hidden deadlock")

for s in fts.states:
    for t in s.in_trs:
        t.dead ← not check(is_not_dead_transition(t))
        if(t.dead or t.bx = ⊥):
            t.false_opt ← False
        else:
            t.false_opt ← not check(may_be_opt_transition(t))
# for all transitions t, it holds that: (t.dead ≡ "t is dead")
# and (t.false_opt ≡ "t is false optional")

```

- Based on expressing criteria for ambiguities as propositional formulae, thus reducing ambiguity detection to **SAT solving**

```

# fts contains the input FTS
for s in fts.states:
    if(s.out_trs = ∅):
        s.hdead ← False
    else:
        s.hdead ← check(exist_deadlock(s))
# for all states s, it holds that:
# (s.hdead ≡ "s is a hidden deadlock")

for s in fts.states:
    for t in s.in_trs:
        t.dead ← not check(is_not_dead_transition(t))
        if(t.dead or t.bx = ⊥):
            t.false_opt ← False
        else:
            t.false_opt ← not check(may_be_opt_transition(t))
# for all transitions t, it holds that: (t.dead ≡ "t is dead")
# and (t.false_opt ≡ "t is false optional")
    
```

- We prove its correctness

- Based on expressing criteria for ambiguities as propositional formulae, thus reducing ambiguity detection to SAT solving

```

# fts contains the input FTS
for s in fts.states:
    if(s.out_trs == ∅):
        s.hdead ← False
    else:
        s.hdead ← check(exist_deadlock(s))
# for all states s, it holds that:
# (s.hdead ≡ "s is a hidden deadlock")

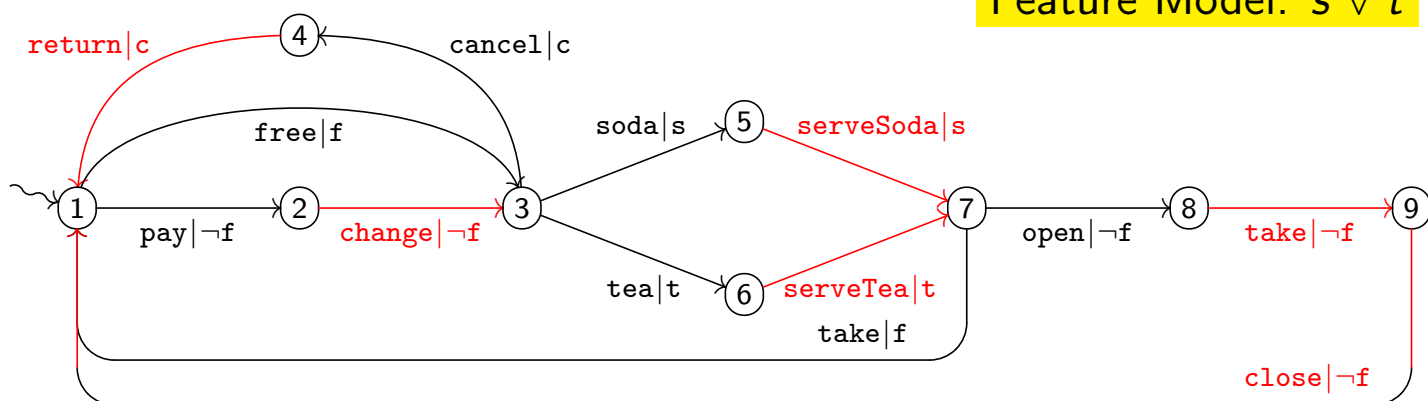
for s in fts.states:
    for t in s.in_trs:
        t.dead ← not check(is_not_dead_transition(t))
        if(t.dead or t.bx == ⊥):
            t.false_opt ← False
        else:
            t.false_opt ← not check(may_be_opt_transition(t))
# for all transitions t, it holds that: (t.dead ≡ "t is dead")
# and (t.false_opt ≡ "t is false optional")

```

- We prove its correctness
- Our implementation uses Z3



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]

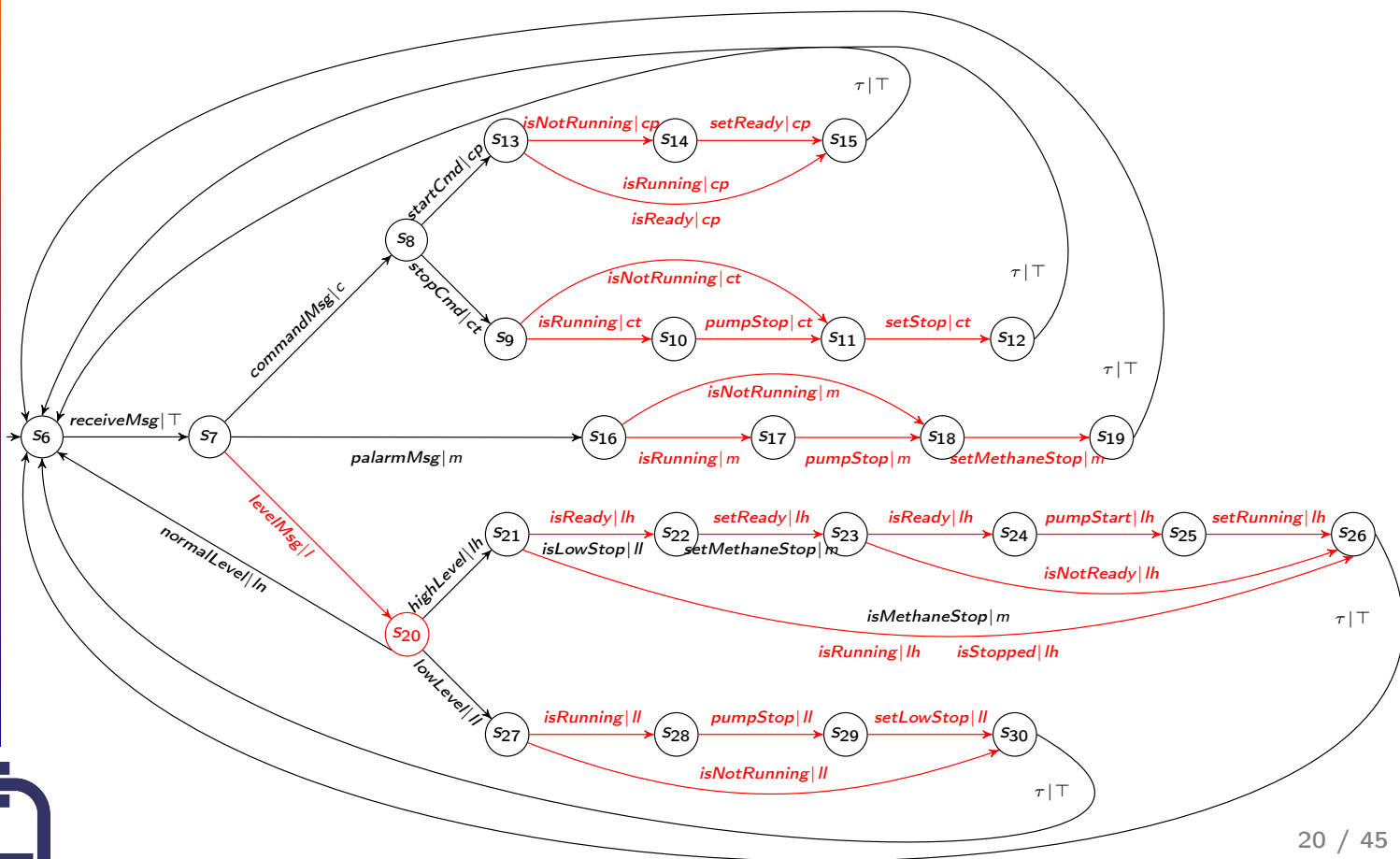
DEAD TRANSITIONS = []

FALSE OPTIONAL TRANSITIONS = [(2,3), (4,1), (5,7), (6,7), (8,9), (9,1)]

HIDDEN DEADLOCK STATES = []



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



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]

FTS	characteristics			results of static analysis				computational effort	
	Model	$ S $	$ \delta $	$ \Sigma $	live-ness	# dead transitions	# false optional transitions	# hidden deadlock states	run-time (s)
Vending machine <small>Classen, PhD thesis, 2011</small>	9	13	12	yes	0	6	0	0.26	29.765
Coffee machine <small>Asirelli et al. @ SPLC'11</small>	14	23	15	yes	0	14	0	0.29	30.305
Soup component <small>Belder et al. @ FMSPLE'15</small>	13	28	18	yes	0	7	0	0.316	30.85
Mine pump (system) <small>Classen, PhD thesis, 2011</small>	25	41	22	no	0	25	1	0.344	31.704
Mine pump (controller) <small>Classen, PhD thesis, 2011</small>	77	104	22	no	0	59	4	0.548	36.295
Coffee/Soup machine <small>Belder et al. @ FMSPLE'15</small>	182	691	33	yes	8	284	0	37.766	119.427
Mine pump (complete) <small>Classen, PhD thesis, 2011</small>	418	1,255	26	yes	0	308	0	98.994	119.127
Claroline <small>Devroey et al. @ VaMoS'14</small>	107	11,236	106	yes	0	259	0	2413.8	2010.229



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)

FTS	characteristics			computational effort				results
				implementation [SPLC19]		implementation [EMSE21]		
	Model	S	$ \delta $	$ \Sigma $	runtime (s)	memory usage (Mb)	runtime (s)	memory usage (Mb)
Vending machine <small>Classen, PhD thesis, 2011</small>	9	13	12	0.92	38.230	0.26	29.765	3.54x
Coffee machine <small>Asirelli et al. @ SPLC'11</small>	14	23	15	2.822	40.140	0.29	30.305	9.72x
Soup component <small>Belder et al. @ FMSPLE'15</small>	13	28	18	2.544	40.870	0.316	30.85	8.05x
Mine pump (system) <small>Classen, PhD thesis, 2011</small>	25	41	22	2.192	41.899	0.344	31.704	6.37x
Mine pump (controller) <small>Classen, PhD thesis, 2011</small>	77	104	22	8.12	49.091	0.548	36.295	14.82x
Coffee/Soup machine <small>Belder et al. @ FMSPLE'15</small>	182	691	33	timeout	–	37.766	119.427	>7200.00x
Mine pump (complete) <small>Classen, PhD thesis, 2011</small>	418	1,255	26	timeout	–	98.994	119.127	>7200.00x
Claroline <small>Devroey et al. @ VaMoS'14</small>	107	11,236	106	timeout	–	2413.8	2010.229	>7200.00x



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)

FTS	characteristics			computational effort				results
				full-fledged implementation		specialised implementation		
	Model	S	δ	Σ	runtime (s)	memory usage (Mb)	runtime (s)	memory usage (Mb)
Coffee/Soup machine <small>Belder <i>et al.</i> © FMSPLE'15</small>					182	691	33	37.766
Mine pump (complete) <small>Classen, PhD thesis, 2011</small>	417	1,255	26	98.994	119.127	2.948	68.969	2.97%
Claroline <small>Devroey <i>et al.</i> © VaMoS'14</small>	107	11,236	106	2413.8	2010.229	86.752	551.888	3.59%



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)

Main ingredient: Modal Transition System (MTS)

- LTS distinguishing admissible (**may**) and necessary (**must**) transitions

Larsen & Thomsen, A Modal Process Logic @ LICS'88

- Recognised as a useful formalism to describe in a **compact** way the possible **behaviour** of all the products (LTSs) of a product family

Fischbein *et al.*, A foundation for behavioural conformance in SPL architectures @ ROSATEA'06

Fantechi & Gnesi, A behavioural model for product families @ ESEC/FSE'07

Main ingredient: Modal Transition System (MTS)

- LTS distinguishing admissible (**may**) and necessary (**must**) transitions

Larsen & Thomsen, A Modal Process Logic @ LICS'88

- Recognised as a useful formalism to describe in a **compact** way the possible **behaviour** of all the products (LTSs) of a product family

Fischbein *et al.*, A foundation for behavioural conformance in SPL architectures @ ROSATEA'06

Fantechi & Gnesi, A behavioural model for product families @ ESEC/FSE'07

- 👉 MTSs cannot model variability constraints regarding **alternative** features, nor regarding **requires** / **excludes** inter-feature relations, resulting in several variants and extensions

Larsen *et al.*, Modal I/O Automata for Interface and Product Line Theories @ ESOP'07

Lauenroth *et al.*, Model Checking of Domain Artifacts in Product Line Engineering @ ASE'09

Main ingredient: Modal Transition System (MTS)

- LTS distinguishing admissible (**may**) and necessary (**must**) transitions

Larsen & Thomsen, A Modal Process Logic @ LICS'88

- Recognised as a useful formalism to describe in a **compact** way the possible **behaviour** of all the products (LTSs) of a product family

Fischbein *et al.*, A foundation for behavioural conformance in SPL architectures @ ROSATEA'06

Fantechi & Gnesi, A behavioural model for product families @ ESEC/FSE'07

- ✋ MTSs cannot model variability constraints regarding **alternative** features, nor regarding **requires** / **excludes** inter-feature relations, resulting in several variants and extensions

Larsen *et al.*, Modal I/O Automata for Interface and Product Line Theories @ ESOP'07

Lauenroth *et al.*, Model Checking of Domain Artifacts in Product Line Engineering @ ASE'09

- 👍 Our solution: add a set of **variability constraints** to the MTSs to be able to decide which derivable products (LTSs) are valid ones

Asirelli *et al.*, Formal Description of Variability in Product Families @ SPLC'11

ter Beek *et al.*, Modelling and analysing variability in product families. *JLAMP*, 2016



Recall:

A **Labelled Transition System** (LTS) is a quadruple $(Q, \Sigma, s_0, \rightarrow)$ with states S , actions A , initial state s_0 , and transitions $\rightarrow \subseteq S \times \Sigma \times S$

Next we define MTSs with variability constraints:

Recall:

A **Labelled Transition System** (LTS) is a quadruple $(Q, \Sigma, s_0, \rightarrow)$ with states S , actions A , initial state s_0 , and transitions $\rightarrow \subseteq S \times \Sigma \times S$

Next we define MTSs with variability constraints:

A **Modal Transition System** (MTS) is a quintuple $(S, \Sigma, s_0, \rightarrow_{\square}, \rightarrow_{\diamond})$ such that $(S, \Sigma, s_0, \rightarrow_{\square} \cup \rightarrow_{\diamond})$ is an LTS, called its underlying LTS

An MTS has two distinct transition relations

1. **may** transition relation $\rightarrow_{\diamond} \subseteq S \times \Sigma \times S$: **admissible** transitions
2. **must** transition relation $\rightarrow_{\square} \subseteq S \times \Sigma \times S$: **necessary** transitions

By definition, any necessary transition is also admissible: $\rightarrow_{\square} \subseteq \rightarrow_{\diamond}$
 (denote $\dashrightarrow \equiv \rightarrow_{\diamond} \setminus \rightarrow_{\square}$: **optional** transitions)

Variability constraints of form **ALT**ernative, **EXC**ludes, **REQ**uires, etc.

a_1 **ALT** \dots **ALT** a_n : precisely one among the $n \geq 2$ actions a_1, \dots, a_n is reachable in \mathcal{L} (i.e. is the label of a reachable transition)

b_1 **OR** \dots **OR** b_n , where b_i is either a_i or $\neg a_i$: at least one among the conditions on $n \geq 2$ actions b_1, \dots, b_n holds, i.e. $b_i = a_i$ is reachable in \mathcal{L} or $b_i = \neg a_i$ is not reachable in \mathcal{L}

a_1 **EXC** a_2 : at most one of the actions a_1 and a_2 is reachable in \mathcal{L}

a_1 **REQ** a_2 : action a_2 is reachable in \mathcal{L} whenever a_1 is reachable in \mathcal{L}

a_1 **REQ** (a_2 **ALT** \dots **ALT** a_n) : precisely one among the $n \geq 2$ actions a_2, \dots, a_n is reachable in \mathcal{L} if a_1 is reachable in \mathcal{L}

a_1 **REQ** (a_2 **OR** \dots **OR** a_n) : at least one among the $n \geq 2$ actions a_2, \dots, a_n is reachable in \mathcal{L} if a_1 is reachable in \mathcal{L}



A **product LTS** is obtained from a family MTS_v in the following way:

1. include **all** (reachable) must transitions and
 2. include **subset** of the (reachable) optional transitions, remove rest
 3. satisfy assumptions of **coherence** and **consistency**
 4. satisfy **variability constraints**
- ⇒ Each selection gives rise to a different variant

A **product LTS** is obtained from a family MTS_v in the following way:

1. include **all** (reachable) must transitions and
 2. include **subset** of the (reachable) optional transitions, remove rest
 3. satisfy assumptions of **coherence** and **consistency**
 4. satisfy **variability constraints**
- ⇒ Each selection gives rise to a different variant

Let $(S, \Sigma, s_0, \delta^\diamond, \delta^\square, \Upsilon)$ be a coherent MTS_v , i.e. $\exists \xrightarrow{a} \implies \nexists \xrightarrow{a}$

A **product LTS** is obtained from a family MTS_v in the following way:

1. include **all** (reachable) must transitions and
 2. include **subset** of the (reachable) optional transitions, remove rest
 3. satisfy assumptions of **coherence** and **consistency**
 4. satisfy **variability constraints**
- ⇒ Each selection gives rise to a different variant

Let $(S, \Sigma, s_0, \delta^\diamond, \delta^\square, \Upsilon)$ be a coherent MTS_v , i.e. $\exists \xrightarrow{-a} \implies \nexists \xrightarrow{a}$

The set $\{\mathcal{P}_i = (S_i, \Sigma, s_0, \delta_i) \mid i > 0\}$ of **product LTSs** is obtained by considering each pair of $S_i \subseteq S$ and $\delta_i \subseteq \delta^\diamond \cup \delta^\square$ to be defined s.t.

1. every $s \in S_i$ is reachable in \mathcal{P}_i from s_0 via transitions from δ_i
2. there exists no $(s, a, s') \in \delta^\square \setminus \delta_i$ such that $s \in S_i$
3. LTS is consistent: both $\xrightarrow{-a} \rightsquigarrow \xrightarrow{a}$ and ~~$\xrightarrow{-a}$~~ not allowed
4. \mathcal{P}_i satisfies all variability constraints in Υ



Theorem (FTS2MTS_v transformation is sound and complete)

Let \mathcal{F} be an FTS and let \mathcal{M} be the MTS_v generated from \mathcal{F} according to the FTS2MTS_v model transformation algorithm. Then the sets of derived variants $\text{lts}(\mathcal{F})$ and $\text{lts}(\mathcal{M})$ coincide, up to dummy transitions and action relabelling.

ter Beek *et al.*, From FTSs to MTSs with Variability Constraints © SEFM'15

Theorem (FTS₂MTS_v transformation is sound and complete)

Let \mathcal{F} be an FTS and let \mathcal{M} be the MTS_v generated from \mathcal{F} according to the FTS₂MTS_v model transformation algorithm. Then the sets of derived variants $\text{Its}(\mathcal{F})$ and $\text{Its}(\mathcal{M})$ coincide, up to dummy transitions and action relabelling.

ter Beek et al., From FTSs to MTSs with Variability Constraints © SEFM'15

Theorem (MTS_v₂FTS transformation is sound and complete)

Let \mathcal{M} be an MTS_v and let \mathcal{F} be the FTS generated from \mathcal{M} according to the MTS_v₂FTS model transformation algorithm. Then $\text{Its}(\mathcal{M}) = \text{Its}(\mathcal{F})$.

ter Beek et al., On the Expressiveness of MTSs with Variability Constraints. SCP, 2019



Dedicated Variability Model Checker VMC

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



fmt.isti.cnr.it

VMC profits from optimisations of KandISTI family of model checkers: bounded, on-the-fly model checking with **linear complexity**, providing clear and easily understandable **explanations** when the validation fails

ter Beek *et al.*, A state/event-based model-checking approach for the analysis of abstract system properties
© SCP, 2011



VMC profits from optimisations of KandISTI family of model checkers: bounded, on-the-fly model checking with **linear complexity**, providing clear and easily understandable **explanations** when the validation fails

ter Beek *et al.*, A state/event-based model-checking approach for the analysis of abstract system properties
© SCP, 2011

VMC accepts as input a specification in (value-passing) modal process algebra, possibly with additional variability constraints:

- interactively explore the model (MTS_v)
- derive and explore (all) the model's valid products (LTSs)
- visualise the model/products graphically as MTS_v /LTSs
- verify branching-time v-ACTL properties over MTS_v /LTSs
- interactively explain why a property is (not) satisfied



Variability analysis strategies supported by VMC



fmt.isti.cnr.it

VMC offers both product-based and family-based variability analyses:



VMC offers both product-based and family-based variability analyses:

1. The actual set of all valid product behaviour can explicitly be generated and the resulting LTSs can all be verified against one and the same logic property (expressed in Action-based CTL)

De Nicola & Vaandrager, Three Logics for Branching Bisimulation. *J. ACM*, 1995

VMC offers both product-based and family-based variability analyses:

1. The actual set of all valid product behaviour can explicitly be generated and the resulting LTSs can all be verified against one and the same logic property (expressed in Action-based CTL)
2. A logic property (expressed in **variability-aware** ACTL) can directly be verified against the MTS_v , relying on the fact that under certain syntactic conditions validity over the MTS_v implies validity of the same property for all its derived products (more later)

De Nicola & Vaandrager, Three Logics for Branching Bisimulation. *J. ACM*, 1995

ter Beek *et al.*, Modelling and analysing variability in product families. *JLAMP*, 2016

VMC offers both product-based and family-based variability analyses:

1. The actual set of all valid product behaviour can explicitly be generated and the resulting LTSs can all be verified against one and the same logic property (expressed in Action-based CTL)
De Nicola & Vaandrager, Three Logics for Branching Bisimulation. *J. ACM*, 1995
2. A logic property (expressed in **variability-aware** ACTL) can directly be verified against the MTS_v , relying on the fact that under certain syntactic conditions validity over the MTS_v implies validity of the same property for all its derived products (more later)
ter Beek *et al.*, Modelling and analysing variability in product families. *JLAMP*, 2016

VMC is freely usable online:

<http://fmt.isti.cnr.it/vmc/>



VMC notifies whenever preservation of an analysis result is applicable

VMC V6.5
(2019)

● ● ● ● ●

Edit Model

View Current Model


Explore the MTS

Draw Family MTS

Generate Products

Welcome

Quit



Kandinsky 1908

The Formula: `not E[true {not tea} U {serveTea>true}]`
is **TRUE**

The formula holds for ALL the MTS variants

(evaluation time= 0.067 sec.)

(total states generated= 9, computations fragments generated= 17, total evaluation time= 0.067 sec.)

ACTL-UCTL-SocL-vACTL

`not E [true {not tea} U {serveTea} true]`

Check
The
Formula

Explain
the
Result



It is not possible that serveTea occurs without being preceded by tea

VMC lists for each product the action labels of all may transitions that have been preserved (as must transitions) in that product

VMC V6.5
(2019)

● ● ● ● ●

New Model ...

Edit Current Model

Explore the MTS


View Current Model

Draw Family MTS

Generate Products

Welcome

Quit



Kandinsky 1908

Evaluation of the formula "[pay] AF {takePaid} true" on all family products

product+CancelPurchase+FreeDrinks+Soda+Tea_12	Formula evaluates	TRUE
product+CancelPurchase+FreeDrinks+Soda_08	Formula evaluates	TRUE
product+CancelPurchase+FreeDrinks+Tea_10	Formula evaluates	TRUE
product+CancelPurchase+Soda+Tea_06	Formula evaluates	FALSE
product+CancelPurchase+Soda_02	Formula evaluates	FALSE
product+CancelPurchase+Tea_04	Formula evaluates	FALSE
product+FreeDrinks+Soda+Tea_11	Formula evaluates	TRUE
product+FreeDrinks+Soda_07	Formula evaluates	TRUE
product+FreeDrinks+Tea_09	Formula evaluates	TRUE
product+Soda+Tea_05	Formula evaluates	TRUE
product+Soda_01	Formula evaluates	TRUE
product+Tea_03	Formula evaluates	TRUE

Logic Formula for all Products

[pay] AF {takePaid} true

Check The Formula

Explain the Result



Whenever pay occurs, eventually take(Paid) occurs

Note

1. Any FTS \mathcal{F} can trivially be transformed into an MTS \mathcal{F}_{MTS} (must \rightarrow necessary transitions, featured \rightarrow optional transitions, all transitions admissible, remove all feature expressions)
2. If the FTS is unambiguous, then the corresponding MTS is **live** (as it has no hidden deadlocks, and all must transitions are necessary)

Note

1. Any FTS \mathcal{F} can trivially be transformed into an MTS \mathcal{F}_{MTS} (must \rightarrow necessary transitions, featured \rightarrow optional transitions, all transitions admissible, remove all feature expressions)
2. If the FTS is unambiguous, then the corresponding MTS is **live** (as it has no hidden deadlocks, and all must transitions are necessary)

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

ter Beek *et al.*, Modelling and analysing variability in product families. *JLAMP*, 2016

Note

1. Any FTS \mathcal{F} can trivially be transformed into an MTS \mathcal{F}_{MTS} (must \rightarrow necessary transitions, featured \rightarrow optional transitions, all transitions admissible, remove all feature expressions)
2. If the FTS is unambiguous, then the corresponding MTS is **live** (as it has no hidden deadlocks, and all must transitions are necessary)

This allows us to carry over a result for MTS v s to unambiguous FTSs:

ter Beek *et al.*, Modelling and analysing variability in product families. *JLAMP*, 2016

Any formula ϕ of $v\text{-ACTLive}^\square$ is preserved by live FTSs: given a live FTS \mathcal{F} , whenever $\mathcal{F}_{\text{MTS}} \models \phi$, then $\mathcal{F}|_\lambda \models \phi$ for all products $\mathcal{F}|_\lambda$ of \mathcal{F}





Family-based model checking with VMC

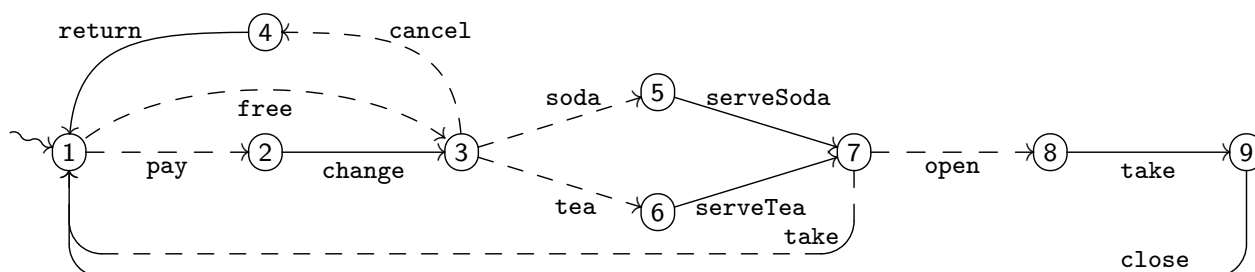


fmt.isti.cnr.it

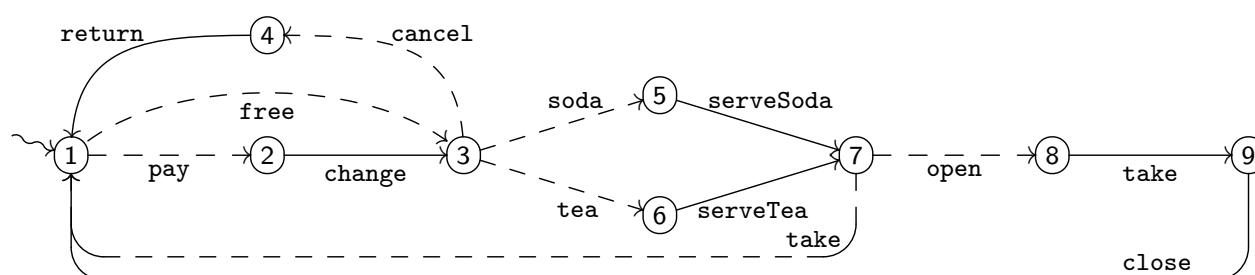
Example v-ACTLive[□] formulae that could now be verified with VMC:
(with a **linear complexity**)



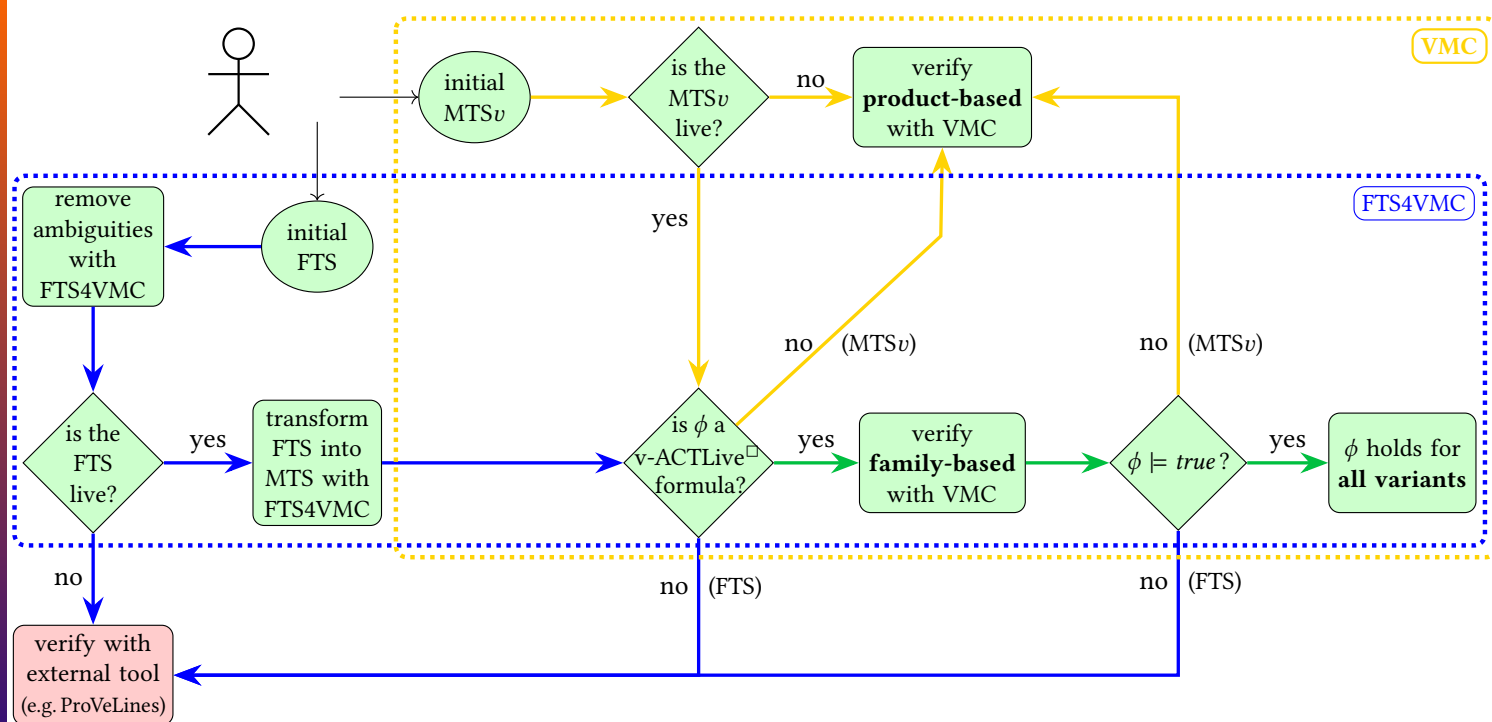
Example v-ACTLive[□] formulae that could now be verified with VMC:
 (with a **linear complexity**)



Example v-ACTLive[□] formulae that could now be verified with VMC:
(with a **linear complexity**)



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



The **green** blocks are automated by the toolchain (FTS4VMC+VMC)

The **blue** and **green** steps (applied to FTSs) are realised by FTS4VMC



Select FTS Model

Upload model

Delete model and close

Download displayed result

Full ambiguities analysis

Liveness analysis

Stop processing

Remove all ambiguities

Remove false optional transitions

Remove dead transitions and hidden deadlock states

Apply transformation

View modal transition system

Verify property

Insert an ACTL formula to verify

Show explanation

Legend

dead transition

false optional transition

hidden deadlock

Analysis of vendingnew.dot

FTS

Console Source Graph Summary Counterexample graph

FM = s or t



Select FTS Model

Upload model

Delete model and close

Download displayed result

Full ambiguities analysis

Liveness analysis

Stop processing

Remove all ambiguities

Remove false optional transitions

Remove dead transitions and hidden deadlock states

Apply transformation

View modal transition system

Verify property

Insert an ACTL formula to verify

Show explanation

Analysis of vendingnew.dot

FTS

Console Source Graph Summary Counterexample graph

Number of states: 9

Number of transitions: 13

Ambiguities found

Dead transitions

False optional transitions

- (2, 3): change | Not(f)
- (4, 1): return | c
- (5, 7): serveSoda | s
- (6, 7): serveTea | t
- (8, 9): take | Not(f)
- (9, 1): close | Not(f)

Hidden deadlock states



Select FTS Model

Upload model

Delete model and close

Download displayed result

Full ambiguities analysis

Liveness analysis

Stop processing

Remove all ambiguities

Remove false optional transitions

Remove dead transitions and hidden deadlock states

Apply transformation

View featured transition system

Verify property

Insert an ACTL formula to verify

Show explanation

Legend

dead transition

false optional transition

hidden deadlock

Analysis of vendingnew.dot

MTS

Console Source Graph Summary Counterexample graph



Select FTS Model

Upload model

Delete model and close

Download displayed result

Full ambiguities analysis

Liveness analysis

Stop processing

Remove all ambiguities

Remove false optional transitions

Remove dead transitions and hidden deadlock states

Apply transformation

View modal transition system

Verify property

Insert an ACTL formula to verify

Show explanation

Legend

dead transition

false optional transition

hidden deadlock

Analysis of mine_pump_new.dot

FTS

Console Source Graph Summary Counterexample graph

FM = I and (c => (ct or cp))



A path in an LTS is *maximal* if it cannot be extended further
Model checking LTL formulae on an LTS reduces to analysing its maximal paths: an LTL formula is valid if it holds for all maximal paths

A path in an LTS is *maximal* if it cannot be extended further
Model checking LTL formulae on an LTS reduces to analysing its maximal paths: an LTL formula is valid if it holds for all maximal paths

Note

1. Trivially carries over to FTSs by ignoring the feature expressions
2. If the FTS is live, then the set of maximal paths of any product is a subset of the set of maximal paths of the FTS

A path in an LTS is *maximal* if it cannot be extended further
Model checking LTL formulae on an LTS reduces to analysing its maximal paths: an LTL formula is valid if it holds for all maximal paths

Note

1. Trivially carries over to FTSs by ignoring the feature expressions
2. If the FTS is live, then the set of maximal paths of any product is a subset of the set of maximal paths of the FTS

Thus:

Any formula ϕ of LTL is preserved by live FTSs: given a live FTS \mathcal{F} ,
whenever $\mathcal{F}_{LTS} \models \phi$, then $\mathcal{F}|_{\lambda} \models \phi$ for all products $\mathcal{F}|_{\lambda}$ of \mathcal{F}





Family-based model checking with SPIN



fmt.isti.cnr.it

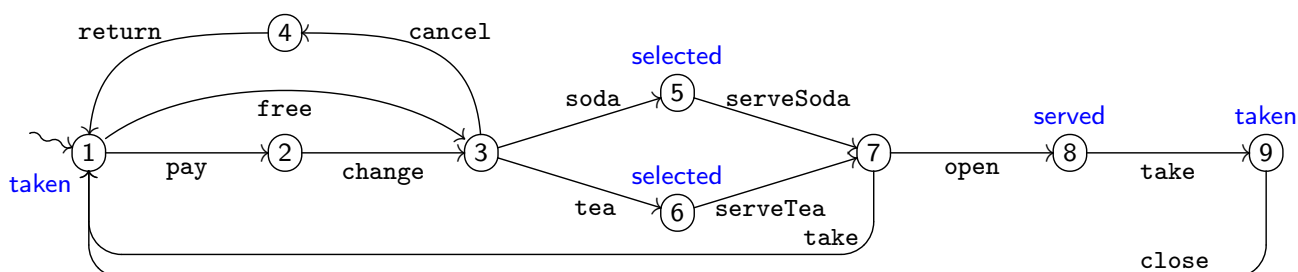
Example LTL formulae that could now be verified with SPIN (for instance):

<http://spinroot.com/>



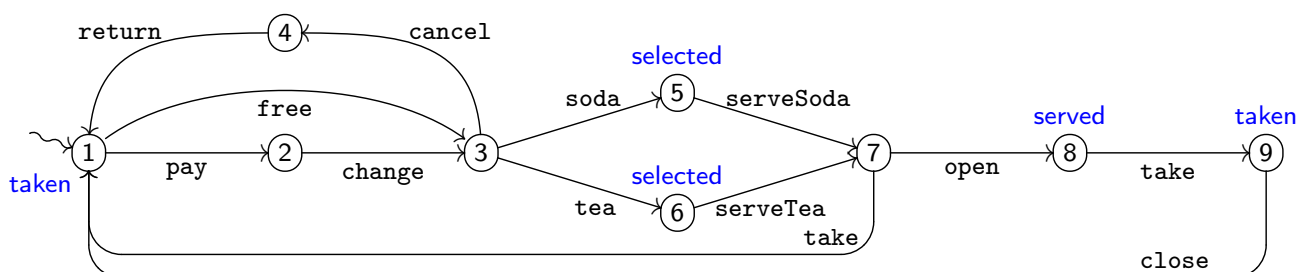
Example LTL formulae that could now be verified with SPIN (for instance):

<http://spinroot.com/>



Example LTL formulae that could now be verified with SPIN (for instance):

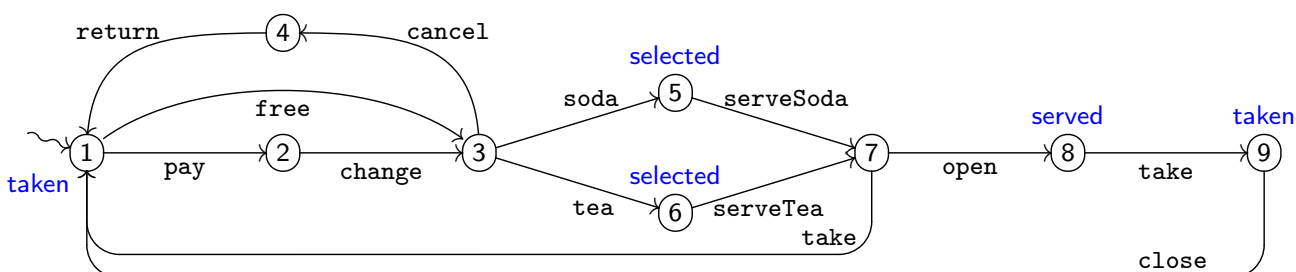
<http://spinroot.com/>



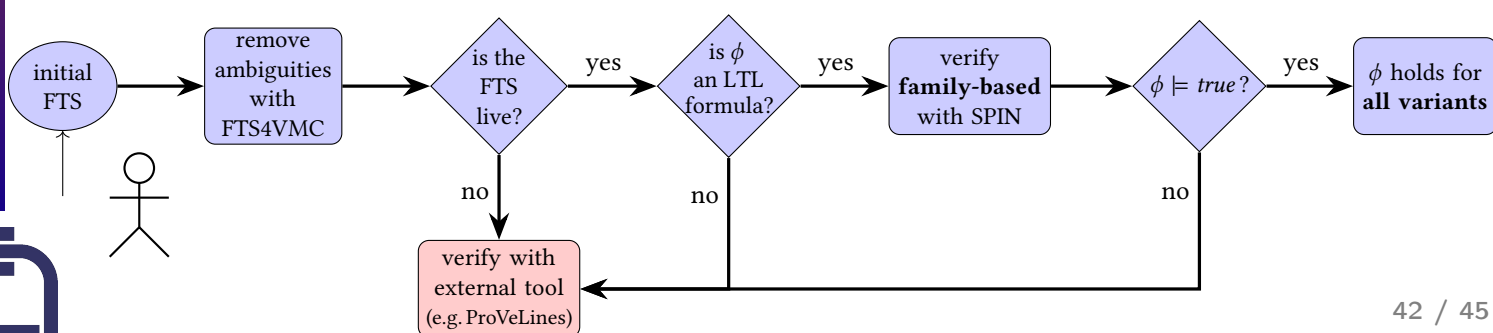
1. $\square (selected \Rightarrow \diamond served)$: after a beverage is selected, the vending machine will always eventually serve a beverage
2. $\square (served \Rightarrow \diamond taken)$: after a beverage is served, a customer will always eventually take the beverage

Example LTL formulae that could now be verified with SPIN (for instance):

<http://spinroot.com/>



1. $\square (selected \Rightarrow \diamond served)$: after a beverage is selected, the vending machine will always eventually serve a beverage
2. $\square (served \Rightarrow \diamond taken)$: after a beverage is served, a customer will always eventually take the beverage



1. Efficient static analysis of FTSs:

- scalable algorithm
- proof of correctness
- benchmark experiments

1. Efficient static analysis of FTSs:

- scalable algorithm
- proof of correctness
- benchmark experiments

2. Efficient verification of FTSs:

- a kind of family-based model checking
- both linear- and branching-time properties

1. Efficient static analysis of FTSs:
 - scalable algorithm
 - proof of correctness
 - benchmark experiments

2. Efficient verification of FTSs:
 - a kind of family-based model checking
 - both linear- and branching-time properties

3. Automated by a toolchain:
 - front-end tool FTS4VMC



1. Efficient static analysis of FTSs:
 - scalable algorithm
 - proof of correctness
 - benchmark experiments
2. Efficient verification of FTSs:
 - a kind of family-based model checking
 - both linear- and branching-time properties
3. Automated by a toolchain:
 - front-end tool FTS4VMC
4. *Ongoing* ~~Future~~ work:
 - integrate FTS2PROMELA transformation

- SPLC19 M.H. ter Beek, F. Damiani, M. Lienhardt, F. Mazzanti, and L. Paolini, Static Analysis of Featured Transition Systems. In Proceedings of the 23rd International Systems and Software Product Line Conference (SPLC'19), Paris, France, ACM Press, New York, 2019, 39–51 ([best paper](#))
- EMSE21 M.H. ter Beek, F. Damiani, M. Lienhardt, F. Mazzanti, and L. Paolini, Efficient Static Analysis and Verification of Featured Transition Systems. *Empirical Software Engineering* (2021) ([to appear](#))
- SPLC21 M.H. ter Beek, F. Damiani, F. Mazzanti, G. Scarso, and M. Valfrè, Static Analysis and Family-based Model Checking of Featured Transition Systems with VMC. Zenodo (February 2021) (~~to be submitted~~)
<http://doi.org/10.5281/zenodo.4497888> *accepted at SPLC'21*
(tool demo + tutorial)



Thanks for your attention!



Hope to see you all at SPLC 2021 'here' in Leicester:

<https://splc2021.net/>

25th Systems and Software Product Line Conference

Submission deadline research papers: April 23, 2021

PC chairs:

- Maurice H. ter Beek, ISTI–CNR, Pisa, Italy
- Ina Schaefer, TU Braunschweig, Germany

Many more tracks (industry, challenges, demos & tools, journal first, doctoral symposium, hall of fame) plus workshops and tutorials

Organised by Mohammad Mousavi and his team

