Model Checking Lots of Systems

Efficient Verification of Temporal Properties in Software Product Lines

Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, Axel Legay, Jean-François Raskin (2010)

Presented by Laura Walsh

Overview

1. Introduction of Software Product Line Engineering
2. Motivations and Challenges
3. Featured Transition System
4. Model Checking Approach
5. Case Study and Results
6. Strengths and Weaknesses
7. Conclusion
8. Discussion

Software Product Line Engineering

The development of software products in *families* (similar products of a system that share core features and also have independent, variable features).

Main question:
How can we best describe (and verify) all the different products of a family?

Definitions

Software Product Line (SPL): a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way

Software Product Line Engineering (SPLE): promotes reuse of the software lifecycle when developing several similar systems.

- SPLE is beneficial to the development of embedded and critical systems (which makes formal modelling and verification of SPLE very important!)

Running Example

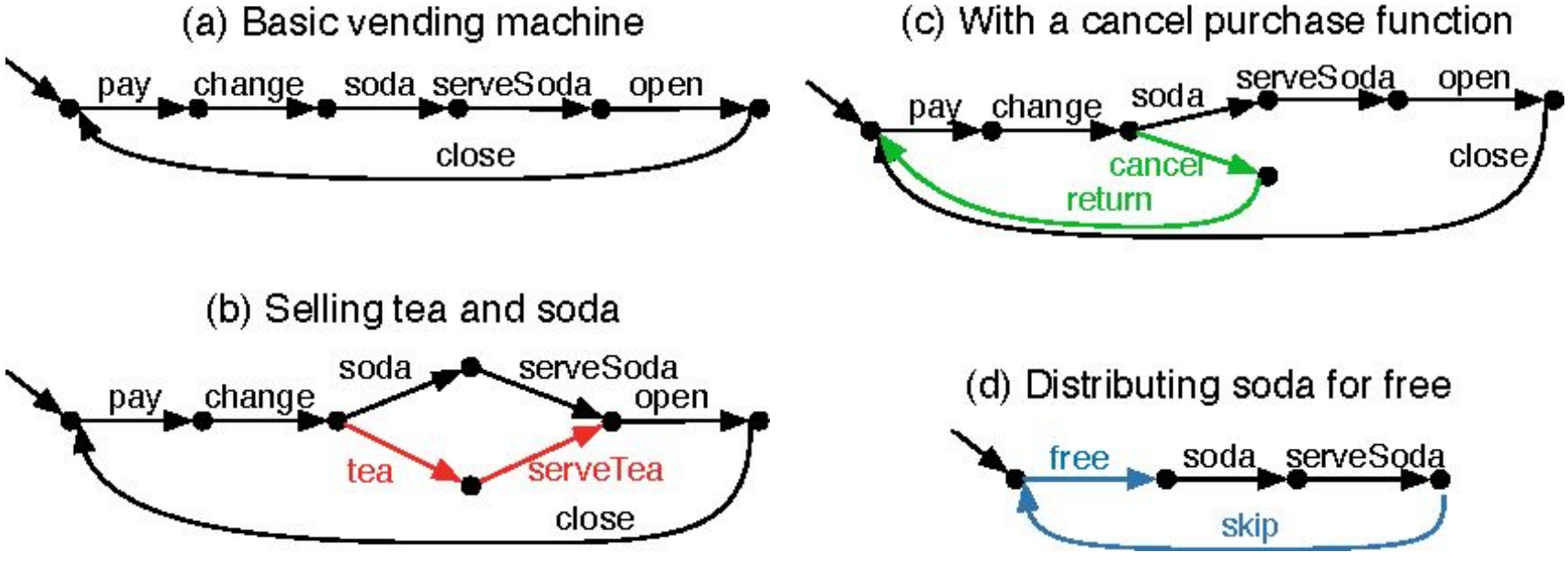
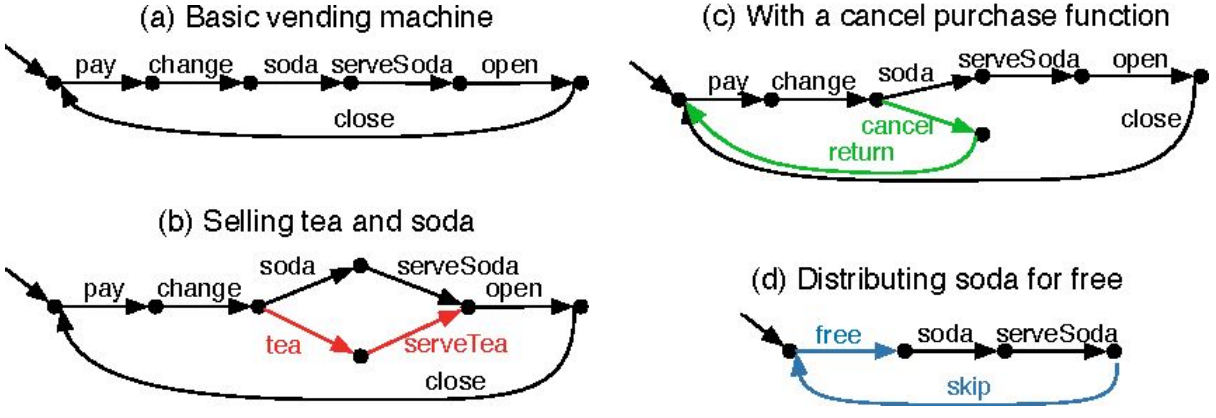


Figure 1: Several variants of a vending machine

Motivation and Challenges

When developing families of products with different features, two major challenges are:

1. Scalable modelling
2. Efficient verification of the system behaviour

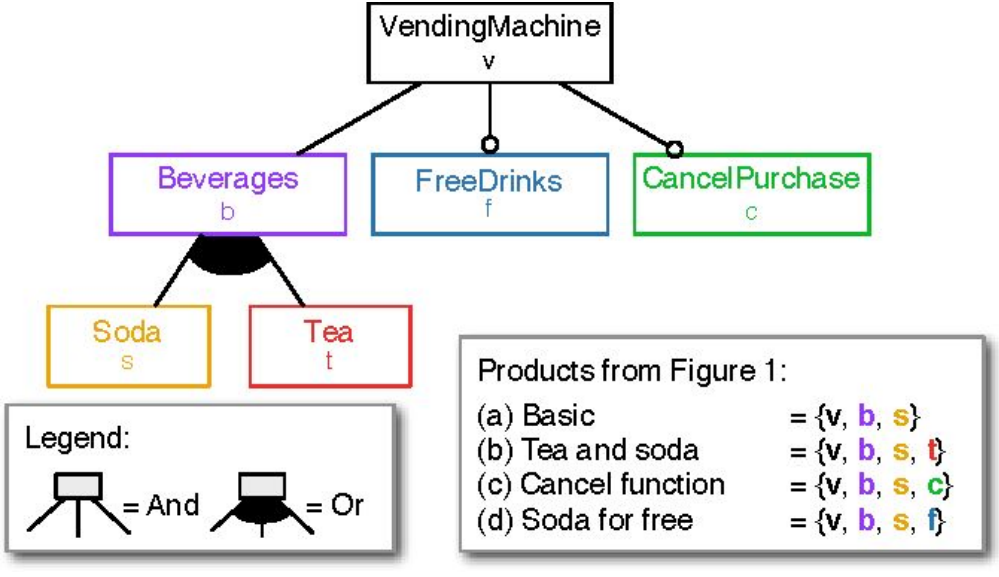


Contribution

1. Featured Transition System
    - Extension of existing Transition System
2. Dedicated Model Checking Technique (supported by proof of concept tool)
    - For verification of desired properties

Base Concepts

“We assume that the reader is familiar with automata theory and has basic knowledge of formal verification”



Feature Diagram

- a type of diagram used to express the variability of the software product line

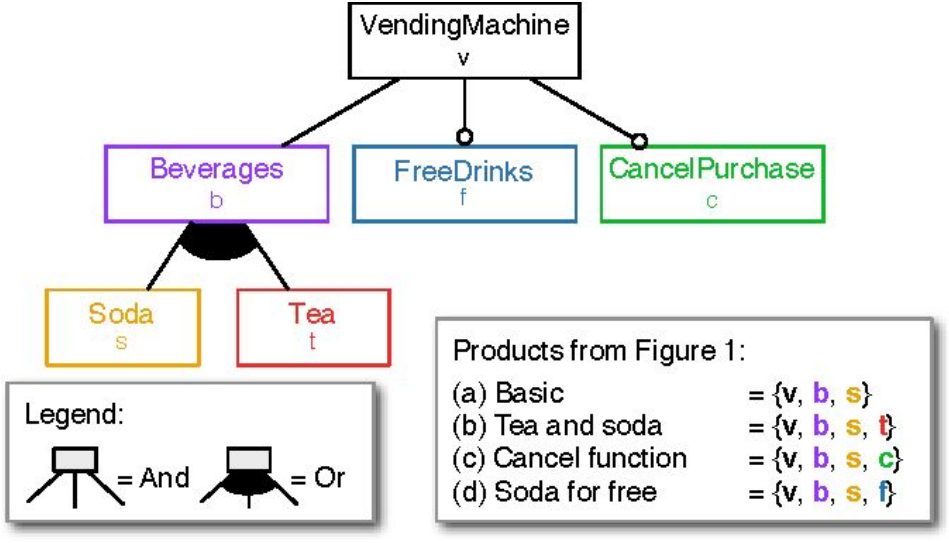


Figure 2: FD for the vending machines of Figure 1.

FD = (N, r, DE)

- N = set of features = {v, b, s, t, f, c}
- r = root node (here, it is **v**), r ∈ N
- DE = set of decomposition edges between features, DE ⊆ N x N = (v,b),(v,f),(v,c),(b,s),(b,t)}

$[[d]]_{FD}$ = semantics of a feature diagram *d* (the set of valid products)

{ {v, b, t}, {v, b, t, f}, {v, b, t, c}, {v, b, t, f, c},
{v, b, s}, {v, b, s, f}, {v, b, s, c}, {v, b, s, f, c},
{v, b, s, t}, {v, b, s, t, f}, {v, b, s, t, c}, {v, b, s, t, f, c}}

Transition System

A transition system is a directed graph with labelled vertices.

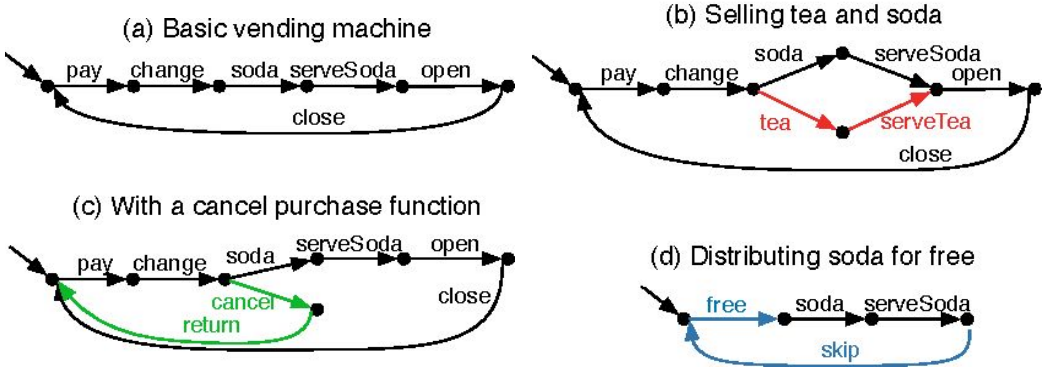


Figure 1: Several variants of a vending machine

An *execution* of transition system *M* is an infinite legal execution paths of the system. The semantics of a TS, $[[t]]_{TS}$ is given by its set of executions (all possible legal paths of execution).

M = (S, Act, trans, I, AP, L)

- S = set of states
- Act = set of actions
- trans = set of transitions between states, encoded with actions
  $\subseteq S \times Act \times S$ $\quad s \xrightarrow{\alpha} s_1$
- I = set of initial states
- AP = set of atomic propositions
- L = labelling function
  $S \rightarrow 2^{AP}$

Featured Transition System

A transition system in which each transition is labelled with the feature it originated from.

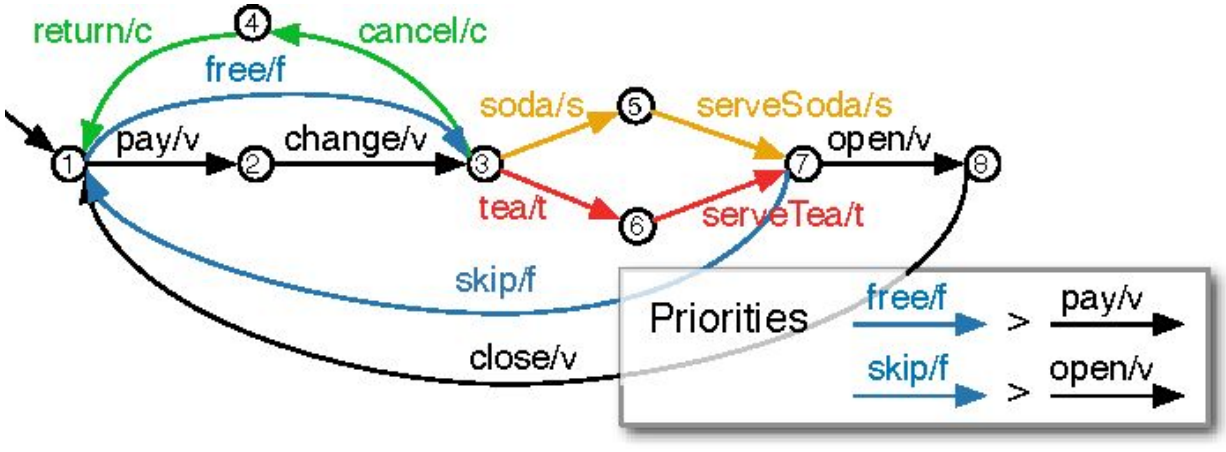


Figure 3: FTS of the vending machine

Same components as a TS (S, Act, trans, I, AP, L), plus:

- d = (N, r, DE) is a feature diagram
- γ = labels transitions with features (trans -> N)
- > = priority relation (⊆ trans x trans)

Featured Transition System

Projection: obtaining the behaviour of one particular product given the overall featured transition system

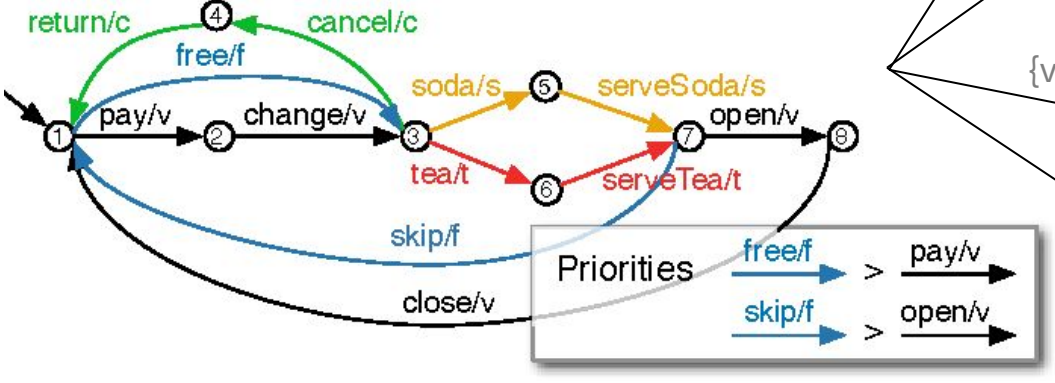


Figure 3: FTS of the vending machine

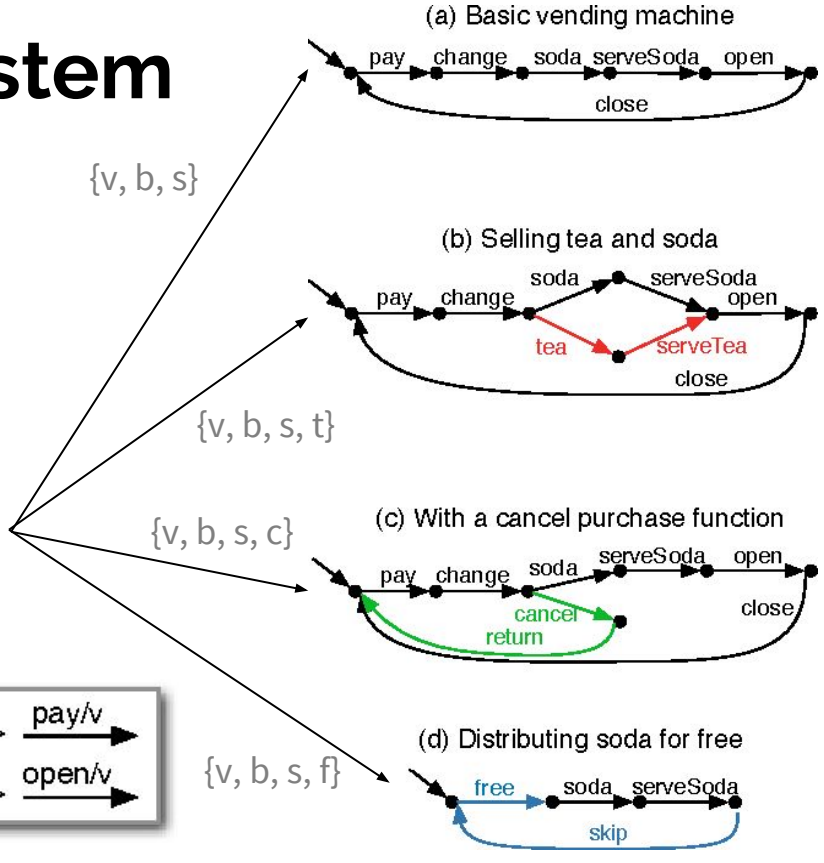


Figure 1: Transition systems for four products

Featured Transition System

FTS Semantics vs. TS Semantics

$$\forall fts \bullet [fts]_{FTS} \subseteq [\![TS(fts)]\!]_{TS}$$

- FTS semantics are not equivalent to TS semantics
  - TS's do not account for the priority relations which are very important in the FTS
  - Using TS model checking techniques would generate false positives
- FTS-specific model checking algorithm is required

Reachability in Featured Transition Systems

- Model checker is meant to perform a search in the state space of the Featured Transition System and thus needs an execution model that is faithful to the FTS semantics
- We want our model checker to indicate the products for which a property does (or does not hold)

EXPLORING THE STATE SPACE OF A FEATURED TRANSITION SYSTEM

- We need a proper execution model that keeps track of products and respects transition priorities

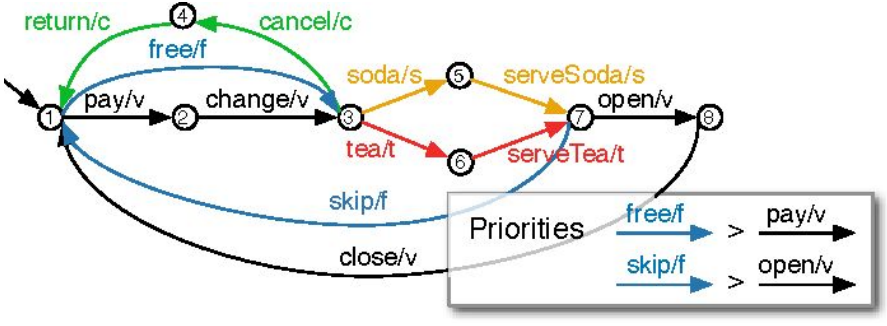
Reachability in Featured Transition Systems

Solution: construct a reachability relation R_0 as you explore the state space

-> Which states are reachable by which products?

- The initial states of the FTS are reachable for all products.
- From a reachable state s0, another state s1 is reachable if there exists a transition (encoded in *trans*) from s0 to s1 which runs action α, AND there is no higher priority relation between s0 and another state (s2) on action α'.

Reachability in Featured Transition Systems



Set of States	States	Reachable By
Initial states	{s1}	All products
Reachable from s1	{s2, s3}	**s2:** reachable by products which contain the feature v (for pay) but do NOT contain the feature f (for free, which has priority over the pay/v transition) **s3:** reachable by products which contain the feature f
Reachable from s2	{s3}	Same products which could reach s2

Model Checking Featured Transition Systems

Goal:

- Verify regular and ω-regular properties
  - If a property is satisfied by the Featured Transition System, then it also must be satisfied by every product of the Software Product Line
  - If a property is violated, the algorithm should report a counter-example as well as the products of the Software Product line that violate the property
- A model satisfies a temporal property if all its projections satisfy the property

Aside: classical model checking algorithms only return a counter-example if there's a violation of a property. Here, we would like both a counter example and a list of the products which violate the property, to help the engineer correct the model.

Model Checking Featured Transition Systems

Problem: If there's a violation, a counter example and list of offending products is returned. But in this case, we don't know which (if any) products DID satisfy the property! (We would like to have this information!)

Solution: The model checker will return an exhaustive list of the violating products. So, we will implicitly know which products satisfy the property (all products that are not mentioned on the violation list)

Model Checking Featured Transition Systems

- Automata-based model checking
- Verifies regular and ω-regular properties (expressed by finite automata and Büchi automata, respectively)

Check whether the *synchronous product* of the system (an FTS) with the automaton (FA or BA) representing the *negation* of the property has an empty language.

- If the language is empty: your property is satisfied
- If the language is not empty: your property is *not* satisfied

Synchronous product = FTS⊗ a

Case Study

Mine Pump Controller

System is made up of:

- Water pump
- Sensor measuring the water level
- Sensor measuring the level of methane in the mine

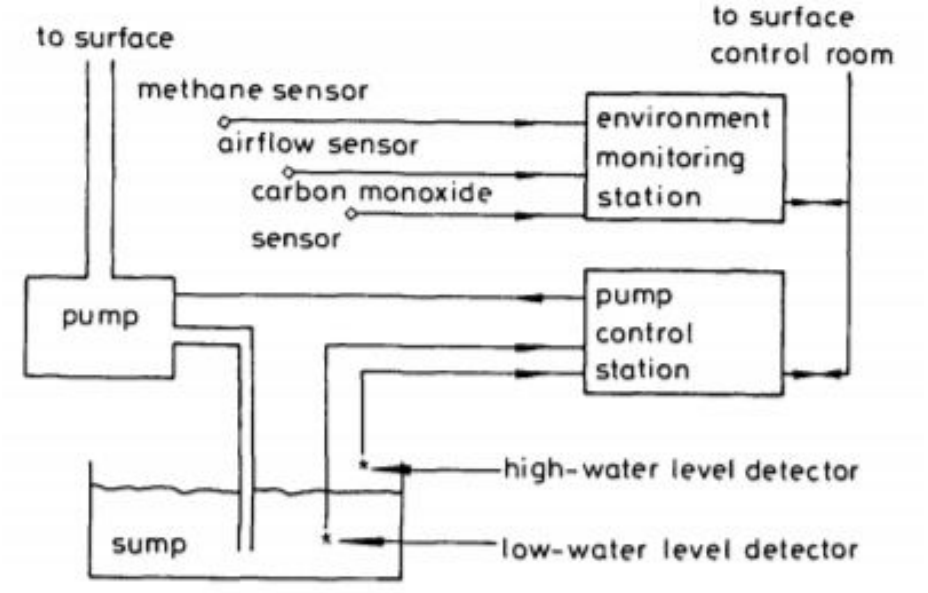


Fig. 1 *Control of mainpump for mine drainage*

The system should activate the pump once the water level reaches a set threshold, but only if the methane is below a critical limit.

Case Study

Five separate Featured Transition Systems were used to model the min pump system and its environment:

1. Control structure of the program
2. Changes to the system state
3. Water level
4. Methane Level
5. State of the pump

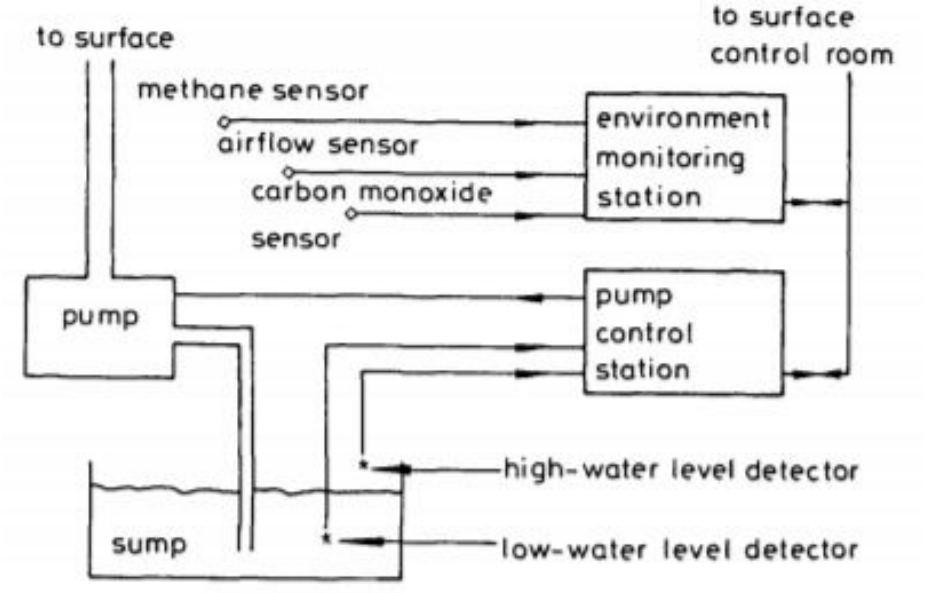


Fig. 1 *Control of mainpump for mine drainage*

These separate FTSs were combined into two overall system FTS: One with 1828 states and 4612 transitions, the other with 29760 states and 69856 transitions

Results

Table 1: Benchmark results for exhaustive counter example search $ExtMC(\mathbf{FTS}, \phi)$.

Formula ϕ		4 features, 4 products Cur.	Our	Diff.	9 features, 64 products Cur.	Our	Diff.
(1.1) $\Box\Diamond(start \wedge \bigcirc msg \wedge (methane \Rightarrow palarm)) \Rightarrow (\Box\Diamond(methane \Rightarrow \Diamond pumpoff)$	✓	9.389 s	5.563 s	1.69	57.706 s	8.162 s	7.07
(1.2) $\neg\Box\Diamond(start \wedge \bigcirc msg \wedge (methane \Rightarrow \Diamond palarm))$	✗	25.741 s	37.663 s	0.68	138.970 s	102.716 s	1.35
(1.3) $\Box\Diamond(start \wedge \bigcirc msg) \Rightarrow \Box(pumpon \Rightarrow \Diamond running)$	✓	5.084 s	4.308 s	1.18	13.716 s	5.317 s	2.58
(1.4) $\Box\Diamond(msg \wedge \bigcirc level) \Rightarrow \Box\Diamond(lowwater \Rightarrow \Diamond pumpoff)$	✓	4.970 s	4.156 s	1.20	16.450 s	4.926 s	3.34
(1.5) $\Box\Diamond(msg \wedge \bigcirc level \wedge ready) \Rightarrow \Box((highwater \wedge \Box!methane) \Rightarrow \Diamond pumpon)$	✓	5.172 s	4.462 s	1.16	14.981 s	5.033 s	2.98
(1.6) $\Box\Diamond(msg \wedge \bigcirc level \wedge ready) \Rightarrow \Box((highwater \wedge !methane) \Rightarrow \Diamond pumpon)$	✗	5.437 s	4.405 s	1.23	17.741 s	4.914 s	3.61

Classical model checking algorithm vs. author's proposed model checking algorithm (implemented in Haskell - functional programming language)

Future Work

- Further optimize their approach
- Incorporate the use of Boolean expressions in the labelling of transitions
  - This would allow for more robust representations, such as the possibility to express a situation in which one transition is associated with more than one feature
- Define translations from high-level modelling languages (e.g. StateCharts, Promela) to FTS (to allow for usability in an industrial setting)
- Develop merging techniques which create a single FTS from multiple TS’s

Conclusion

- Challenge: to efficiently model and verify the variety of products which contain different features within a software product line
- Contributions: Featured Transition System representation of an SPL & Dedicated model checking technique using an automata-based model checking approach
- Case study benchmark results showed their method was on average 3.5x faster than the classical model checking algorithm

Strengths

- Good explanation of the motivations for this work, the challenges associated, and their improvements upon existing work
- Quite a computational improvement compared to classical approaches!
- Great ideas for improvements through future work

Weaknesses

- A lot of prior knowledge assumed (and minimal re-explanation of these concepts)
- Not many concrete examples given of the concepts proposed/explained
- More figures would have been helpful (there are only three, and they are in the first three pages)

Discussion: Does anyone have strengths and weaknesses to add?

Weaknesses

mal definition for FA and BA is given below. The language of an FA (resp. BA) consists of finite (resp. infinite) words and can be *empty* (accepts no word).

$$[\![d]\!]_{FD} \subseteq \mathcal{P}(N).$$

Discussion

1. How did you find the technical depth of this paper?
    - Was the paper totally understandable / difficult to understand?
2. What could the authors have done to make their paper more accessible and understandable?
3. What do you think the Related Work section adds to the paper?

Questions

1. What kind of audience was this paper written for?
    a. Part of the International Conference on Software Engineering
2. Did the authors end up doing the future work they proposed?
    a. Seems like yes!
3. Was this a highly original approach at the time (2010)?
    a. What was typically done until this approach was proposed?