An Introduction to Software Product Lines and Modelling and Analysing Variability

Maurice ter Beek
Formal Methods and Tools lab, ISTI–CNR, Pisa, Italy

Based on joint work with Stefania Gnesi, Franco Mazzanti (ISTI–CNR), Alessandro Fantechi (UNIFI), Erik de Vink, Tim Willemse (TU/e), Axel Legay (UCL/Inria), Alberto Lluch Lafuente, Andrea Vandin (DTU)

IPA Fall Days on Models in Software Engineering
WestCord de Veluwe, Garderen, The Netherlands
October 30, 2018
Outline

1. Introduction to Software Product Lines

2. Modelling Variability
   - Featured Transition Systems (fPROMELA, fSMV, fLTL, fCTL)
   - Modal Transition Systems (with variability constraints, v-ACTL)
   - Flan, PFLan, QFLan, ProFeat

3. Analysing Variability
   - SNIP, ProVeLines, fNuSMV, SPIN, mCRL2
   - VMC
   - QFLan, ProFeat, PRISM

4. Outlook
An introduction to Software Product Lines

- (Software) Product Line Engineering
- Product variability and features
- Variability modelling and analysis

These slides are partly based on slides by Klaus Pohl, Sven Apel and Slinger Jansen
Product line engineering in theory...
...and in practice
Software Product Line Engineering

- Developing a family of products from a common reference model (usually in terms of features) and mass customisation (to serve different markets)
Software Product Line Engineering

- Developing a family of products from a common reference model (usually in terms of features) and mass customisation (to serve different markets)

Two main differences with classical software engineering

- Two distinct development processes

- Variability
Software Product Line Engineering

- Developing a family of products from a common reference model (usually in terms of features) and mass customisation (to serve different markets)

Two main differences with classical software engineering

- Two distinct development processes
  - Domain engineering: develop reusable domain artefacts
  - Application engineering: develop individual products by reusing domain artefacts

- Variability
Software Product Line Engineering

- Developing a family of products from a *common reference model* (usually in terms of features) and *mass customisation* (to serve different markets)

Two main differences with classical software engineering

- Two distinct development processes
  - Domain engineering: develop reusable domain artefacts
  - Application engineering: develop individual products by reusing domain artefacts

- Variability
  - Common features that are part of all products
  - Variable features that can be selected for a product
Variability in requirements

R12: The navigation system must allow the user to make inputs using a control panel or by voice entry

- R12 comprises the following three realisations
Variability in requirements

R12: The navigation system must allow the user to make inputs using a control panel or by voice entry

- R12 comprises the following three realisations
  1. A navigation system that allows the user to make inputs only via the control panel
Variability in requirements

R12: The navigation system must allow the user to make inputs using a control panel or by voice entry

- R12 comprises the following three realisations
  1. A navigation system that allows the user to make inputs only via the control panel
  2. A navigation system that allows the user to make inputs only via voice entry
Variability in requirements

R12: The navigation system must allow the user to make inputs using a control panel or by voice entry

- R12 comprises the following three realisations
  1. A navigation system that allows the user to make inputs only via the control panel
  2. A navigation system that allows the user to make inputs only via voice entry
  3. A navigation system that allows the user to make inputs only via the control panel and by voice entry
Variability in requirements

R12: The navigation system must allow the user to make inputs using a control panel or by voice entry

- R12 comprises the following three realisations
  1. A navigation system that allows the user to make inputs only via the control panel
  2. A navigation system that allows the user to make inputs only via voice entry
  3. A navigation system that allows the user to make inputs only via the control panel and by voice entry

- Conjunction is a logical “or” or an exclusive “or”? 
- Is only one system asked for, or two or three different systems?
Variation points and variants

- A variation point represents an aspect of a product family that varies among the different products
Variation points and variants

- A **variation point** represents an aspect of a product family that varies among the different products
  - R12: “input modality of the user interface”
Variation points and variants

- A **variation point** represents an aspect of a product family that varies among the different products.
- R12: “input modality of the user interface”

- A **variant** represents a specific configuration (i.e. an incarnation) of a variable aspect that a product in a product family can have.
Variation points and variants

- A variation point represents an aspect of a product family that varies among the different products.
  R12: “input modality of the user interface”

- A variant represents a specific configuration (i.e. an incarnation) of a variable aspect that a product in a product family can have.
  R12: “input via control panel”, “input via voice entry”
Variability: prerequisite for success

- **Domain engineering**: explicit documentation of variability supports the identification of possible variable aspects and fosters explicit decisions about which aspects shall be variable in the product family (variation points) and which options shall exist for each variable aspect (variants);
Variability: prerequisite for success

- Domain engineering: explicit documentation of variability supports the identification of possible variable aspects and fosters explicit decisions about which aspects shall be variable in the product family (variation points) and which options shall exist for each variable aspect (variants); it also supports engineers, architects, designers and testers in realising the defined variation points and variants.
Variability: prerequisite for success

- **Domain engineering**: explicit documentation of variability supports the identification of possible variable aspects and fosters explicit decisions about which aspects shall be variable in the product family (variation points) and which options shall exist for each variable aspect (variants); it also supports engineers, architects, designers and testers in realising the defined variation points and variants.

- **Application engineering**: explicit documentation of variability in terms of variation points and variants supports system development by making explicit the necessary decisions and decision options.
Features

■ What is a feature?
Features

- What is a feature?
  - End-user visible behaviour or property of a system...
  - …that may be optional and/or may have alternatives
Features

- What is a feature?
  - End-user visible behaviour or property of a system…
  - …that may be optional and/or may have alternatives

- Features represent commonalities and variabilities of (software) systems
What’s in a *feature*?

<table>
<thead>
<tr>
<th>Reference</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kang et al. [3]</td>
<td>“a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems”</td>
</tr>
<tr>
<td>Kang et al. [8]</td>
<td>“distinctively identifiable functional abstractions that must be implemented, tested, delivered, and maintained”</td>
</tr>
<tr>
<td>Eisenecker and Czarnecki</td>
<td>“anything users or client programs might want to control about a concept”</td>
</tr>
<tr>
<td>Bosch et al. [9]</td>
<td>“A logical unit of behaviour specified by a set of functional and non-functional requirements.”</td>
</tr>
<tr>
<td>Chen et al. [10]</td>
<td>“a product characteristic from user or customer views, which essentially consists of a cohesive set of individual requirements”</td>
</tr>
<tr>
<td>Batory [11]</td>
<td>“an elaboration or augmentation of an entity(s) that introduces a new service, capability or relationship”</td>
</tr>
<tr>
<td>Batory et al. [12]</td>
<td>“an increment in product functionality”</td>
</tr>
<tr>
<td>Apel et al. [13]</td>
<td>“a structure that extends and modifies the structure of a given program in order to satisfy a stakeholder’s requirement, to implement and encapsulate a design decision, and to offer a configuration option.”</td>
</tr>
</tbody>
</table>

(Classsen et al. @ FASE’08)
Feature selection

Typically, only a subset of feature combinations is valid.

Visor $\Rightarrow$ Helmet
Products and product lines

- Product $\Leftrightarrow$ valid feature combination (configuration)
Products and product lines

- Product ⇔ valid feature combination (configuration)
- Product line ⇔ set of valid feature combinations of a domain
Features and software

- Main concept in SPL-E
  - Easy to use in informal models
  - Easily converts into business: product sales
  - Easily converts into product design: variability
  - Enables reuse of features
Features and software

- Main concept in SPL
  - Easy to use in informal models
  - Easily converts into business: product sales
  - Easily converts into product design: variability
  - Enables reuse of features

- In telecommunication, features became popular in the 1960s with the advent of computer-controlled telephone switches; telecommunication software has been conceived in terms of features ever since
Variability models: explicit decisions

- Compact representations of all products of a product family in terms of their features
Variability models: explicit decisions

- Compact representations of all products of a product family in terms of their features

- Feature diagram/model: hierarchical tree structure, with the family as its root, features as its nodes and possibly further cross-tree constraints (Kang et al. ’90)
Variability models: explicit decisions

- Compact representations of all products of a product family in terms of their features
- **Feature diagram/model**: hierarchical tree structure, with the family as its root, features as its nodes and possibly further cross-tree constraints (Kang et al. ’90)
- **Common Variability Language (CVL)**: OMG's effort to standardise variability modelling as a separate and generic language to define variability on base models
- **Orthogonal Variability Model** (Pohl et al. ’05), etc., etc.
Advantages of dedicated conceptual model

- Improved communication with different stakeholders (e.g. communicate to customers which variants can be selected at which variation points)
Advantages of dedicated conceptual model

- **Improved communication** with different stakeholders (e.g. communicate to customers which variants can be selected at which variation points)

- **Transparent decisions** i.e. the originator of a variation point is forced to state the rationale for introducing variability in a specific domain artefact
Advantages of dedicated conceptual model

- **Improved communication** with different stakeholders (e.g. communicate to customers which variants can be selected at which variation points)

- **Transparent decisions** i.e. the originator of a variation point is forced to state the rationale for introducing variability in a specific domain artefact

- **Relationships between requirements and variants** become **traceable** (e.g. stakeholders can document which requirements, design, implementation and test artefacts are influenced by a variant)
Example requirements coffee machine

1. Initially, a coin must be inserted: either a €, exclusively in case of coffee machines for the European market, or a $, exclusively in case of coffee machines for the Canadian market (alternative features)
Example requirements coffee machine

1. Initially, a coin must be inserted: either a €, exclusively in case of coffee machines for the European market, or a $, exclusively in case of coffee machines for the Canadian market (alternative features)

2. After having inserted a coin, the user has to be offered the option to choose whether or not (s)he wants sugar in her/his beverage, after which (s)he has to be offered to select a beverage (mandatory features)
Example requirements coffee machine

1. Initially, a coin must be inserted: either a €, exclusively in case of coffee machines for the European market, or a $, exclusively in case of coffee machines for the Canadian market (alternative features)

2. After having inserted a coin, the user has to be offered the option to choose whether or not (s)he wants sugar in her/his beverage, after which (s)he has to be offered to select a beverage (mandatory features)

3. The choice of beverages offered by a coffee machine may vary (the options being cappuccino, coffee and tea), but every coffee machine must offer at least one beverage (or relation among features);
Example requirements coffee machine

1. Initially, a coin must be inserted: either a €, exclusively in case of coffee machines for the European market, or a $, exclusively in case of coffee machines for the Canadian market (alternative features)

2. After having inserted a coin, the user has to be offered the option to choose whether or not (s)he wants sugar in her/his beverage, after which (s)he has to be offered to select a beverage (mandatory features)

3. The choice of beverages offered by a coffee machine may vary (the options being cappuccino, coffee and tea), but every coffee machine must offer at least one beverage (or relation among features); furthermore, whenever a coffee machine offers cappuccino, then it must offer coffee as well, while tea may only be offered by coffee machines for the European market (requires and excludes relations among features)
Example requirements coffee machine

1. Initially, a coin must be inserted: either a €, exclusively in case of coffee machines for the European market, or a $, exclusively in case of coffee machines for the Canadian market (alternative features)

2. After having inserted a coin, the user has to be offered the option to choose whether or not (s)he wants sugar in her/his beverage, after which (s)he has to be offered to select a beverage (mandatory features)

3. The choice of beverages offered by a coffee machine may vary (the options being cappuccino, coffee and tea), but every coffee machine must offer at least one beverage (or relation among features); furthermore, whenever a coffee machine offers cappuccino, then it must offer coffee as well, while tea may only be offered by coffee machines for the European market (requires and excludes relations among features)

4. Optionally, a ringtone may be rung after the coffee machine has delivered the chosen beverage (optional feature)
Example requirements coffee machine

1. Initially, a coin must be inserted: either a €, exclusively in case of coffee machines for the European market, or a $, exclusively in case of coffee machines for the Canadian market (alternative features)

2. After having inserted a coin, the user has to be offered the option to choose whether or not (s)he wants sugar in her/his beverage, after which (s)he has to be offered to select a beverage (mandatory features)

3. The choice of beverages offered by a coffee machine may vary (the options being cappuccino, coffee and tea), but every coffee machine must offer at least one beverage (or relation among features); furthermore, whenever a coffee machine offers cappuccino, then it must offer coffee as well, while tea may only be offered by coffee machines for the European market (requires and excludes relations among features)

4. Optionally, a ringtone may be rung after the coffee machine has delivered the chosen beverage (optional feature)

5. As soon as the user has taken her/his beverage, the coffee machine must return in its idle state
Coffee machine feature diagram

- Coffee machine
  - Sugar
  - Beverage
    - Cappuccino
    - Coffee
  - Ringtone
  - Coin
    - Dollar
    - Euro
Mobile phone feature diagram
ATM feature diagram
Automotive feature diagram
How many products?

- Car
  - CarBody
  - Engine
  - Transmission
    - Automatic
    - Manual 4
    - Manual 5
  - PullsTrailer

Pick a out of b features

1..3

requires
How many products?

Transmission contributes one of three options, multiplier = 3
Car contributes two options, multiplier = 2, but…

Transmission contributes one of three options, multiplier = 3
Car contributes two options, multiplier = 2, but…
How many products?

Transmission contributes one of three options, multiplier = 3
Car contributes two options, multiplier = 2, but…
…actually only works in one case; thus: 4 different products
Applications of feature diagrams

- Communicate with stakeholders
- Identify objects for reuse
- Identify objects for sales opportunities
- Identify cross-cutting concerns
- Software composition and deployment
- etc.
... Automated product derivation

User -> Mapping -> Domain artifacts

Generator -> Generated products

[Batory, IEEE TSE 2004]
[Czarnecki et al., 2000]
A product line of Lego minifigures (16 valid feature combinations)
‘Feature diagram’ for configuring the 16 valid products (allowing more…)
Introduction to SPL

VW features

Maak uw keuze uit het kleurenpalet door met de muis een kleur te selecteren.
Configure your 11-inch MacBook Air

Processor
Enjoy incredible performance from fourth-generation Intel Core processors. Choose the speed and processor you want.
Learn more
- 1.3GHz Dual-Core Intel Core i5, Turbo Boost up to 2.6GHz
- 1.7GHz Dual-Core Intel Core i7, Turbo Boost up to 3.3GHz [+ £130.00]

Memory
More memory (RAM) increases overall performance and enables your computer to run more applications at the same time.
Learn more
- 4GB 1600MHz LPDDR3 SDRAM
- 8GB 1600MHz LPDDR3 SDRAM [+ £80.00]

Storage
Your MacBook Air comes as standard with flash storage. Flash storage has no moving parts and provides faster responsiveness and enhanced durability.
Learn more
- 256GB Flash Storage
- 512GB Flash Storage [+ £240.00]
BMW product configurator
## BMW feature packages

### 530d Sedan
- **Total price**: EUR 67,639.48
- **Configuration and pricing details**: View leasing and financing details. Configure and save configuration. Information on prices.

#### Carrosseriekleur, interieur en wielen

<table>
<thead>
<tr>
<th>Model</th>
<th>Carrosseriekleur, interieur en wielen</th>
<th>Lichtmetaal vielen M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpinweiss II</td>
<td>Anthrazit EUR 0,00</td>
<td>Lichtmetaal vielen M</td>
</tr>
<tr>
<td></td>
<td>Stof/Leder Flashlight</td>
<td>Dubbelspaak (172M), 19&quot;</td>
</tr>
<tr>
<td></td>
<td>Anthrazit EUR 0,00</td>
<td>EUR 1,555,62</td>
</tr>
</tbody>
</table>

#### Metallic colors
- Uni
- Metallic

#### Stoffen
- Stoff
- Leder

---

Lichtmetaal vielen M
- Dubbelspaak (172M), 19"
- EUR 1,555,62
- *Only with* M sportpakket (EUR 6,582,78)
Smart constraints (1/2)

smart roadster
roadster BRABUS
74 kW benzinemotor
74 kW benzinemotor Xclusive

Prijs
Adviesprijs van de fabrikant
Basisprijs: 31.450,00 €
tridion &
bodypanels: -- €
Interieur: -- €
Opties: -- €
Accessoires: -- €
Totaalprijs: 31.450,00 €

Verder met...
Interieurkleur
Brandstofverbruik[4] in 1/100 km resp. km/l (gecombineerd): 5,2
Smart constraints (2/2)
Feature binding times

- The moment a choice is made for a variable feature
Feature binding times

- The moment a choice is made for a variable feature.
- For a software product:
  - Compile time
  - Release time
  - Deployment time
  - Start-up time
  - Run-time
Feature binding times

- The moment a choice is made for a variable feature
- For a software product:
  - Compile time
  - Release time
  - Deployment time
  - Start-up time
  - Run-time
- For a car:
  - Spray time
  - Drive-out-of-the-factory time
  - etc.
Feature binding times

- The moment a choice is made for a variable feature
- For a software product:
  - Compile time
  - Release time
  - Deployment time
  - Start-up time
  - Run-time
- For a car:
  - Spray time
  - Drive-out-of-the-factory time
  - etc.
- For lego:
  - Playing time
Variability dependencies (1/2)

Allowed choices of variants at a specific variation point

- **Mandatory variant**: if the variation point is selected, this variant *must* always be selected
Variability dependencies (1/2)

Allowed choices of variants at a specific variation point

- **Mandatory variant**: if the variation point is selected, this variant *must* always be selected
- **Optional variant**: if the variation point is selected, this variant *may* be selected but it does not have to be
Variability dependencies (1/2)

Allowed choices of variants at a specific variation point

- **Mandatory variant**: if the variation point is selected, this variant *must* always be selected
- **Optional variant**: if the variation point is selected, this variant *may* be selected but it does not have to be
- **Alternative choice** i.e. a collection of at least two optional variants together with a [min…max] notation to indicate the permissible number of variants to be selected: if the variation point is selected, at least “min” variants *must* be selected while at most “max” variants *may* be selected
Variability dependencies (2/2)

Cross-tree constraints

- **Requires**: indicates that the presence of one feature requires the presence of another feature
Variability dependencies (2/2)

Cross-tree constraints

- **Requires**: indicates that the presence of one feature requires the presence of another feature
- **Excludes**: indicates that the presence of two features is mutually exclusive
Variability dependencies (2/2)

Cross-tree constraints

- **Requires**: indicates that the presence of one feature requires the presence of another feature
- **Excludes**: indicates that the presence of two features is mutually exclusive

Quantitative constraints

- **Feature attributes (non-functional)**: $\text{cost}(\text{feature}) = 7$, etc.
- $\text{cost}(\text{product}) = \sum \{ \text{cost}(\text{feature}) \mid \text{feature} \in \text{product} \}$
Attributed feature model (1/2)

Quantitative constraint: cost(Coffee Machine) ≤ 30
Attributed feature model (2/2)
Domain Requirement Engineering

- During domain RE, the requirements for the entire SPL are defined
Domain Requirement Engineering

- During domain RE, the requirements for the entire SPL are defined
- Basis for: (i) developing the entire SPL family (ii) defining the requirements of each product
Domain Requirement Engineering

- During domain RE, the requirements for the entire SPL are defined.
- Basis for: (i) developing the entire SPL family (ii) defining the requirements of each product.
- Comprises the same core and cross-sectional RE activities as RE for single systems.
Domain Requirement Engineering

- During domain RE, the requirements for the entire SPL are defined.
- Basis for: (i) developing the entire SPL family (ii) defining the requirements of each product.
- Comprises the same core and cross-sectional RE activities as RE for single systems; these activities have the same goals plus, in addition, the goal to define the SPL variability.
Elicitation of requirements variability

- Requirements stakeholder 1
- Requirements stakeholder 2
- Requirements stakeholder 3
- Requirements stakeholder 4

Commonality and variability analysis

- Common requirements
- Variable requirements
Application RE

During application RE, the requirements for a specific application of the SPL are defined by exploiting the domain requirements artefacts (incl. defined variation points and variants)
Application RE

- During application RE, the requirements for a specific application of the SPL are defined by exploiting the domain requirements artefacts (incl. defined variation points and variants).
- Compared with RE for single systems, two additional tasks must be accomplished:
Application RE

- During application RE, the requirements for a specific application of the SPL are defined by exploiting the domain requirements artefacts (incl. defined variation points and variants).
- Compared with RE for single systems, two additional tasks must be accomplished:
  1. Binding the defined variability
Application RE

- During application RE, the requirements for a specific application of the SPL are defined by exploiting the domain requirements artefacts (incl. defined variation points and variants).

- Compared with RE for single systems, two additional tasks must be accomplished:
  1. Binding the defined variability
  2. Documenting the variability binding
So why have variability?

- Flexibility to deliver
  - Different versions of products based on the same trunk of code (Windows XP professional, Windows XP Home edition, etc.)
  - Onto different platforms (a linux, a mac and a windows version)
  - With different components (Windows Media Player + Divx, iTunes, etc.)
  - Onto different plug-in products, etc., etc., etc.
So why have variability?

- Flexibility to deliver
  - Different versions of products based on the same trunk of code (Windows XP professional, Windows XP Home edition, etc.)
  - Onto different platforms (a linux, a mac and a windows version)
  - With different components (Windows Media Player + Divx, iTunes, etc.)
  - Onto different plug-in products, etc., etc., etc.

- Is it an advantage then? Yes, but…
  - Requirements engineering becomes more complex
  - Deployment becomes more complex
  - Sales becomes more complex
  - Updates become WAY more complex
  - Testing becomes more complex (all configurations need to be tested)
So why have variability?

- Flexibility to deliver
  - Different versions of products based on the same trunk of code (Windows XP professional, Windows XP Home edition, etc.)
  - Onto different platforms (a linux, a mac and a windows version)
  - With different components (Windows Media Player + Divx, iTunes, etc.)
  - Onto different plug-in products, etc., etc., etc.

- Is it an advantage then? Yes, but…
  - Requirements engineering becomes more complex
  - Deployment becomes more complex
  - Sales becomes more complex
  - Updates become WAY more complex
  - Testing becomes more complex (all configurations need to be tested)
    “We always have 126,000,000 different bicycles in store!
    (but only the parts for 1,000…)”
Weakness: feature interactions

- Features need to be combined, but have restrictions on each other
Weakness: feature interactions

- Features need to be combined, but have restrictions on each other
- Consider e.g. a phone switching system with the following features
  1. Call forwarding
  2. Do not disturb
  3. 3-way calling
  4. No interaction between the features
Weakness: feature interactions

- Features need to be combined, but have restrictions on each other
- Consider e.g. a phone switching system with the following features
  1. Call forwarding
  2. Do not disturb
  3. 3-way calling
  4. No interaction between the features
- Scenario 1: Bob forwards his calls to Alice; Alice sets “Do not disturb”; Bob receives a call, which is forwarded to Alice; Alice’s phone rings!
Weakness: feature interactions

- Features need to be combined, but have restrictions on each other
- Consider e.g. a phone switching system with the following features
  1. Call forwarding
  2. Do not disturb
  3. 3-way calling
  4. No interaction between the features

- Scenario 1: Bob forwards his calls to Alice; Alice sets “Do not disturb”; Bob receives a call, which is forwarded to Alice; Alice’s phone rings!
- Scenario 2: Bob still forwards his calls to Alice; Bob invites Alice for a 3-way call; Carolina calls Bob to become part of the 3-way call; either
Weakness: feature interactions

- Features need to be combined, but have restrictions on each other
- Consider e.g. a phone switching system with the following features
  1. Call forwarding
  2. Do not disturb
  3. 3-way calling
  4. No interaction between the features
- Scenario 1: Bob forwards his calls to Alice; Alice sets “Do not disturb”; Bob receives a call, which is forwarded to Alice; Alice’s phone rings!
- Scenario 2: Bob still forwards his calls to Alice; Bob invites Alice for a 3-way call; Carolina calls Bob to become part of the 3-way call; either
  1. Carolina is forwarded to Alice, who cannot accept any other calls then; or
Weakness: feature interactions

- Features need to be combined, but have restrictions on each other
- Consider e.g. a phone switching system with the following features
  1. Call forwarding
  2. Do not disturb
  3. 3-way calling
  4. No interaction between the features
- Scenario 1: Bob forwards his calls to Alice; Alice sets “Do not disturb”; Bob receives a call, which is forwarded to Alice; Alice’s phone rings!
- Scenario 2: Bob still forwards his calls to Alice; Bob invites Alice for a 3-way call; Carolina calls Bob to become part of the 3-way call; either
  1. Carolina is forwarded to Alice, who cannot accept any other calls then; or
  2. Carolina is accepted into the 3-way call
Weakness: complexity/scalability

with 33 features

a unique product for every person on this planet
atoms in the universe
(Behavioural) variability analysis

Rigorously establish critical system requirements (for quality assurance) with formal models and automated analysis tools
(Behavioural) variability analysis

Rigorously establish critical system requirements (for quality assurance) with formal models and automated analysis tools

- For decades successful in single product/system engineering
(Behavioural) variability analysis

Rigorously establish critical system requirements (for quality assurance) with formal models and automated analysis tools

- For decades successful in single product/system engineering
- Not exploited broadly in SPLs, while correctness of artifacts for reuse and correctness of developed products is of crucial importance (many massively produced (embedded) systems and safety- or business-critical applications)
(Behavioural) variability analysis

Rigorously establish critical system requirements (for quality assurance) with formal models and automated analysis tools

- For decades successful in single product/system engineering
- Not exploited broadly in SPLE, while correctness of artifacts for reuse and correctness of developed products is of crucial importance (many massively produced (embedded) systems and safety- or business-critical applications)

Traditionally

- Mainstream formal methods do not consider variability directly
(Behavioural) variability analysis

Rigorously establish critical system requirements (for quality assurance) with formal models and automated analysis tools

- For decades successful in single product/system engineering
- Not exploited broadly in SPLE, while correctness of artifacts for reuse and correctness of developed products is of crucial importance (many massively produced (embedded) systems and safety- or business-critical applications)

Traditionally

- Mainstream formal methods do not consider variability directly
- Formal methods that have been applied in SPLE mainly focus on structural rather than behavioural properties (feature model analysis: e.g. dead features, false optional features, etc.)
Variability analysis strategies
(type checking, static analysis, model checking, theorem proving, testing, etc.)

Product-Based Analysis

Family-Based Analysis

Feature-Based Analysis
Product-based analysis

\[ O(2^n) \text{ for } n \text{ features} \]
Product-based analysis

Simple approach
Standard tools available
Infeasible for large product sets
Same behaviour / code verified numerous times

\[ O(2^n) \] for \( n \) features
Family-based analysis
Family-based analysis

Beneficial for many products with substantial similarities
Generates complex analysis tasks
Requires (compact) family metamodels
Feature-based analysis

(classification from Thüm et al. @ ACM Comput. Surv., 2014)
Feature-based analysis

Beneficial for large and cohesive features
Support for open-world scenarios
Incomplete w.r.t. feature interactions

(classification from Thüm et al. @ ACM Comput. Surv., 2014)
Modelling and Analysing Variability

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

⇒ challenges known models and tools by potentially high number of different products, each giving rise to a large state space in general

- Featured Transition Systems (FTS)
  SNIP/fPROMELA/fLTL, fNuSMV/fSMV/fCTL, ProVeLines
  Classen et al. @ ICSE’11, Int. J. Softw. Tools Technol. Transf., 2012, Cordy et al. @ SPLC’13

- Modal Transition Systems (MTS) with variability constraints
  Asirelli et al. @ iFM’10, FMOODS’11, SPLC’11, ter Beek et al. @ J. Logic Algebr. Meth. Program., 2016
  Variability Model Checker VMC/v-ACTL
  ter Beek et al. @ FM’12, SPLC’12, SPLat’14

- UML SPL profile, PL-CCS, variable I/O automata, feature nets, etc.
  Ziadi et al. @ PFE’03, Grüler et al. @ FMOODS’08, SPLC’08, Lauenroth et al. @ ASE’09,
  Muschevici et al. @ SEFM’11, Softw. Syst. Model., 2016
Modelling and Analysing Variability

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

⇒ challenges known models and tools by potentially high number of different products, each giving rise to a large state space in general

- **Featured Transition Systems (FTS)**
  
  SNIP/fPROMELA/fLTL, fNuSMV/fSMV/fCTL, ProVeLines
  Classen et al. @ ICSE'11, *Int. J. Softw. Tools Technol. Transf.*, 2012, Cordy et al. @ SPLC'13

- **Modal Transition Systems (MTS) with variability constraints**
  Asirelli et al. @ iFM’10, FMOODS’11, SPLC’11, ter Beek et al. @ *J. Logic Algebr. Meth. Program.*, 2016

Variability Model Checker VMC/v-ACTL
  ter Beek et al. @ FM’12, SPLC’12, SPLat’14

- **UML SPL profile, PL-CCS, variable I/O automata, feature nets, etc.**
  Ziadi et al. @ PFE’03, Grüler et al. @ FMOODS’08, SPLC’08, Lauenroth et al. @ ASE’09, Muschevici et al. @ SEFM’11, *Softw. Syst. Model.*, 2016
Recall (atomic propositions used for verification purposes):

De Nicola & Vaandrager @ J. ACM, 1995

A **Doubly-Labelled Transition System** ($\text{L}^2\text{TS}$) is a sextuple 
$(Q, A, \bar{q}, \rightarrow, AP, L)$ with states $Q$, actions $A$, initial state $\bar{q}$, 
transitions $\rightarrow \subseteq Q \times A \times Q$, atomic propositions $AP$, and 
labeling function $L : S \rightarrow 2^{AP}$

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


A **Featured Transition System** (FTS) is an octuple 
$(Q, A, \bar{q}, \rightarrow, AP, L, FD, \gamma)$ with *underlying* $\text{L}^2\text{TS} (Q, A, \bar{q}, \rightarrow, AP, L)$, 
feature diagram $FD$ over a set $\mathbb{F}$ of features, and total function 
$\gamma : \rightarrow \rightarrow \mathbb{B}(\mathbb{F})$ labelling each transition with a feature expression, i.e. 
a Boolean expression over the features
FTS for SPLE

Recall (atomic propositions used for verification purposes):

De Nicola & Vaandrager @ *J. ACM*, 1995

A **Doubly-Labelled Transition System** (L$^2$TS) is a sextuple $(Q, A, \overline{q}, \rightarrow, AP, L)$ with states $Q$, actions $A$, initial state $\overline{q}$, transitions $\rightarrow \subseteq Q \times A \times Q$, atomic propositions $AP$, and labeling function $L : S \rightarrow 2^{AP}$

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


A **Featured Transition System** (FTS) is an octuple $(Q, A, \overline{q}, \rightarrow, AP, L, FD, \gamma)$ with underlying L$^2$TS $(Q, A, \overline{q}, \rightarrow, AP, L)$, feature diagram $FD$ over a set $\mathbb{F}$ of features, and total function $\gamma : \rightarrow \rightarrow \mathbb{B}(\mathbb{F})$ labelling each transition with a feature expression, i.e. a Boolean expression over the features
FTS of example SPL: a vending machine

Feature model:

12 valid products  

e.g. \{v, b, s, t\},  \{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:

\[
\text{VendingMachine} \\
\text{Beverages} \quad \text{FreeDrinks} \quad \text{CancelPurchase} \\
\text{Soda} \quad \text{Tea} \\
\]

\[
\{v,b,s,t\}, \{v,b,s,c\}
\]
FTS of example SPL: a vending machine

Feature model:

VendingMachine

Beverages b

FreeDrinks f

CancelPurchase c

Soda s

Tea t

e.g. \{v, b, s, t\}, \{v, b, s, c\}
Dedicated FTS model checker SNIP (now ProVeLines)

fPROMELA:

```cpp
typedef features {
    bool v;
    bool f;
    ...
};
features F;
```

gd :: F.v && !F.f;
pay :: F.f;
free :: else;
skip;
gd;

```
return/c
```
```
cancel/c
```
```
free/f
```
```
serveSoda/s
```
```
selected
```
```
soda/s
```
```
serveTea/t
```
```
take/f
```
```
take/v
```
```
close/v
```
```
serveTea/t
```
```
selected
```
```
tea/t
```
```
selected
```
```
open/v^¬f
```
```
open
```
```
[¬f] □ (selected ⇒ ◇ open)
```
Similarly fNuSMV with fSMV/fCTL
Main ingredient: Modal Transition Systems (MTS)

- LTS distinguishing possible (may) and required (must) transitions
  Larsen & Thomsen @ LICS’88

- Recognised as a useful model to describe in a compact way the possible behaviour of all the products (LTS) of a product family
  Fischbein et al. @ ROSATEA’06, Fantechi & Gnesi @ ESEC/FSE’07, SPLC’08

MTS cannot model variability constraints regarding alternative features, nor regarding requires/excludes inter-feature relations, resulting in several variants and extensions
  Larsen et al. @ ESOP’07, Lauenroth et al. @ ASE’09

Our solution: add a set of variability constraints to the MTS to be able to decide which derivable products (LTS) are valid ones
  Asirelli et al. @ SPLC’11, ter Beek et al. @ J. Logic Algebr. Meth. Program., 2016
MTSv for SPLE

Main ingredient: Modal Transition Systems (MTS)

- LTS distinguishing possible (may) and required (must) transitions
  Larsen & Thomsen @ LICS’88

- Recognised as a useful model to describe in a compact way the possible behaviour of all the products (LTS) of a product family
  Fischbein et al. @ ROSATEA’06, Fantechi & Gnesi @ ESEC/FSE’07, SPLC’08

춧 MTS cannot model variability constraints regarding alternative features, nor regarding requires/excludes inter-feature relations, resulting in several variants and extensions
  Larsen et al. @ ESOP’07, Lauenroth et al. @ ASE’09

👍 Our solution: add a set of variability constraints to the MTS to be able to decide which derivable products (LTS) are valid ones
  Asirelli et al. @ SPLC’11, ter Beek et al. @ J. Logic Algebr. Meth. Program., 2016
MTS for SPLE

Main ingredient: Modal Transition Systems (MTS)

- LTS distinguishing possible (may) and required (must) transitions
  Larsen & Thomsen @ LICS’88

- Recognised as a useful model to describe in a compact way the possible behaviour of all the products (LTS) of a product family
  Fischbein et al. @ ROSATEA’06, Fantechi & Gnesi @ ESEC/FSE’07, SPLC’08

👉 MTS cannot model variability constraints regarding alternative features, nor regarding requires/excludes inter-feature relations, resulting in several variants and extensions
  Larsen et al. @ ESOP’07, Lauenroth et al. @ ASE’09

👍 Our solution: add a set of variability constraints to the MTS to be able to decide which derivable products (LTS) are valid ones
  Asirelli et al. @ SPLC’11, ter Beek et al. @ J. Logic Algebr. Meth. Program., 2016
Recall:

A **Labelled Transition System** (LTS) is a quadruple \((Q, A, \overline{q}, \rightarrow)\) with states \(Q\), actions \(A\), initial state \(\overline{q}\) and transitions \(\rightarrow \subseteq Q \times A \times Q\)

Next we define MTS with variability constraints:

A **Modal Transition System** (MTS) is a quintuple \((Q, A, \overline{q}, \rightarrow\square, \rightarrow\lozenge)\) such that \((Q, A, \overline{q}, \rightarrow\square \cup \rightarrow\lozenge)\) is an LTS, called its underlying LTS

An MTS has two distinct transition relations

1. may transition relation \(\rightarrow\lozenge \subseteq Q \times A \times Q\): possible transitions
2. must transition relation \(\rightarrow\square \subseteq Q \times A \times Q\): required transitions

By definition, any required transition is also possible: \(\rightarrow\square \subseteq \rightarrow\lozenge\)

(denote \(\rightarrow\equiv \rightarrow\lozenge \setminus \rightarrow\square\): optional transitions)
Recall:

A **Labelled Transition System (LTS)** is a quadruple \((Q, A, \bar{q}, \to)\) with states \(Q\), actions \(A\), initial state \(\bar{q}\) and transitions \(\to \subseteq Q \times A \times Q\).

Next we define MTS with variability constraints:

A **Modal Transition System (MTS)** is a quintuple \((Q, A, \bar{q}, \to^\square, \to^\lozenge)\) such that \((Q, A, \bar{q}, \to^\square \cup \to^\lozenge)\) is an LTS, called its underlying LTS.

An MTS has two distinct transition relations:

1. **may** transition relation \(\to^\lozenge \subseteq Q \times A \times Q\): possible transitions
2. **must** transition relation \(\to^\square \subseteq Q \times A \times Q\): required transitions

By definition, any required transition is also possible: \(\to^\square \subseteq \to^\lozenge\) (denote \(-\to \equiv \to^\lozenge \setminus \to^\square\): **optional** transitions)
Variability constraints of form \texttt{ALTernative}, \texttt{EXCludes}, \texttt{REQuires}, etc.

\begin{itemize}
    \item \(a_1 \texttt{ ALT} \cdots \texttt{ ALT} a_n\): precisely one among the \(n \geq 2\) actions \(a_1, \ldots, a_n\) is reachable in \(L\) (i.e. is the label of a reachable transition)
    \item \(b_1 \texttt{ OR} \cdots \texttt{ OR} b_n\), where \(b_i\) is either \(a_i\) or \(\neg a_i\): at least one among the conditions on \(n \geq 2\) actions \(b_1, \ldots, b_n\) holds, i.e. \(b_i = a_i\) is reachable in \(L\) or \(b_i = \neg a_i\) is not reachable in \(L\)
    \item \(a_1 \texttt{ EXC} a_2\): at most one of the actions \(a_1\) and \(a_2\) is reachable in \(L\)
    \item \(a_1 \texttt{ REQ} a_2\): action \(a_2\) is reachable in \(L\) whenever \(a_1\) is reachable in \(L\)
    \item \(a_1 \texttt{ REQ} (a_2 \texttt{ ALT} \cdots \texttt{ ALT} a_n)\): precisely one among the \(n \geq 2\) actions \(a_2, \ldots, a_n\) is reachable in \(L\) if \(a_1\) is reachable in \(L\)
    \item \(a_1 \texttt{ REQ} (a_2 \texttt{ OR} \cdots \texttt{ OR} a_n)\): at least one among the \(n \geq 2\) actions \(a_2, \ldots, a_n\) is reachable in \(L\) if \(a_1\) is reachable in \(L\)
\end{itemize}
**Derive products (implemented in VMC)**

A **product LTS** is obtained from a family MTS 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 \((Q, A, \overline{q}, \delta^\bigcirc, \delta^\Box, \Upsilon)\) be a coherent MTS, i.e. \(\exists \xrightarrow{a} \quad \not\exists \xrightarrow{a}\)

The set \(\{\mathcal{P}_i = (Q_i, A, \overline{q}, \delta_i) \mid i > 0\}\) of product LTS is obtained by considering each pair of \(Q_i \subseteq Q\) and \(\delta_i \subseteq \delta^\bigcirc \cup \delta^\Box\) to be defined s.t.

1. every \(q \in Q_i\) is reachable in \(\mathcal{P}_i\) from \(\overline{q}\) via transitions from \(\delta_i\)
2. there exists no \((q, a, q') \in \delta^\Box \setminus \delta_i\) such that \(q \in Q_i\)
3. LTS is consistent: both \(\xrightarrow{a} \not\sim \xrightarrow{a}\) and \(\not\xrightarrow{a} \not\sim \xrightarrow{a}\) not allowed
4. \(\mathcal{P}_i\) satisfies all variability constraints in \(\Upsilon\)
Derive products (implemented in VMC)

A product LTS is obtained from a family MTS 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 \((Q, A, \overline{q}, \delta^{\Diamond}, \delta^{\Box}, \Upsilon)\) be a coherent MTS, i.e. \(\exists \rightarrow a \quad \Rightarrow \quad \not\exists \rightarrow a\)

The set \(\{ \mathcal{P}_i = (Q_i, A, \overline{q}, \delta_i) \mid i > 0 \}\) of product LTS is obtained by considering each pair of \(Q_i \subseteq Q\) and \(\delta_i \subseteq \delta^{\Diamond} \cup \delta^{\Box}\) to be defined s.t.

1. every \(q \in Q_i\) is reachable in \(\mathcal{P}_i\) from \(\overline{q}\) via transitions from \(\delta_i\)
2. there exists no \((q, a, q') \in \delta^{\Box} \setminus \delta_i\) such that \(q \in Q_i\)
3. LTS is consistent: both \(\not\rightarrow a\not\rightarrow a\) and \(\not\rightarrow a\not\rightarrow a\) not allowed
4. \(\mathcal{P}_i\) satisfies all variability constraints in \(\Upsilon\)
Theorem (FTS2MTS\(\nu\) transformation is sound and complete)

Let \( F \) be an FTS and let \( M \) be the MTS\(\nu\) generated from \( F \) according to the FTS2MTS\(\nu\) model transformation algorithm. Then the sets of derived variants \( \text{lts}(F) \) and \( \text{lts}(M) \) coincide, up to dummy transitions and action relabelling.
MTS\(\nu\) equally expressive as FTS

ter Beek et al. @ SEFM’15, Sci. Comput. Program., 2019

Theorem (FTS2MTS\(\nu\) transformation is sound and complete)

Let \(\mathcal{F}\) be an FTS and let \(\mathcal{M}\) be the MTS\(\nu\) generated from \(\mathcal{F}\) according to the FTS2MTS\(\nu\) 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.

\[
\begin{align*}
\gamma = & \{(\text{pay}, v \land \neg f) \leftrightarrow (v \land \neg f), (\text{free}, f) \leftrightarrow f, (\text{change}, v) \leftrightarrow v, (\text{cancel}, c) \leftrightarrow c, (\text{return}, c) \leftrightarrow c, (\text{soda}, s) \leftrightarrow s, (\text{tea}, t) \leftrightarrow t, \\
& (\text{serveSoda}, s) \leftrightarrow s, (\text{serveTea}, t) \leftrightarrow t, (\text{take}, f) \leftrightarrow f, (\text{open}, v \land \neg f) \leftrightarrow (v \land \neg f), (\text{take}, v) \leftrightarrow v, (\text{close}, v) \leftrightarrow v\}
\end{align*}
\]
**Theorem (FTS2MTS\(_\nu\) transformation is sound and complete)**

Let \( \mathcal{F} \) be an FTS and let \( \mathcal{M} \) be the MTS\(_\nu\) generated from \( \mathcal{F} \) according to the FTS2MTS\(_\nu\) 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.

**Theorem (MTS\(_\nu\)2FTS transformation is sound and complete)**

Let \( \mathcal{M} \) be an MTS\(_\nu\) and let \( \mathcal{F} \) be the FTS generated from \( \mathcal{M} \) according to the MTS\(_\nu\)2FTS model transformation algorithm. Then \( \text{lts}(\mathcal{M}) = \text{lts}(\mathcal{F}) \).
VMC builds on optimization of UMC (input: UML state machines)

V. ter Beek et al. @ Sci. Comput. Program., 2011

VMC: bounded, on-the-fly model checking, providing explanations

B. Biere et al. @ TACAS'99

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

- interactively explore the model (MTS)
- derive and explore (all) the model’s valid products (LTS)
- visualize the model/products graphically as MTS/LTS
- verify v-ACTL properties over MTS/LTS
- interactively explain why a property is (not) satisfied

Model checking of v-ACTL formulae on MTS can be achieved in a complexity that is linear w.r.t. the state space size.
Dedicated MTS v Variability Model Checker VMC

VMC builds on optimization of UMC (input: UML state machines)  
  ter Beek et al. @ Sci. Comput. Program., 2011

VMC: bounded, on-the-fly model checking, providing explanations
  Biere et al. @ TACAS'99

VMC accepts as input a specification in (value-passing) modal process algebra, possibly with additional variability constraints
  - interactively explore the model (MTS)
  - derive and explore (all) the model’s valid products (LTS)
  - visualize the model/products graphically as MTS/LTS
  - verify v-ACTL properties over MTS/LTS
  - interactively explain why a property is (not) satisfied

Model checking of v-ACTL formulae on MTS can be achieved in a complexity that is linear w.r.t. the state space size
Variability analysis strategies supported by VMC

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 LTS can all be verified against one and the same logic property (expressed in Action-based CTL)

   De Nicola, Vaandrager @ J. ACM, 1995

2. A logic property (expressed in variability-aware ACTL) can directly be verified against the MTS, relying on the fact that under certain syntactic conditions validity over the MTS implies validity of the same property for all its derived products (more later)

   ter Beek et al. @ J. Logic Algebr. Meth. Program., 2016

VMC v6.4 is freely usable online:

http://fmt.isti.cnr.it/vmc/
Variability analysis strategies supported by VMC

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 LTS can all be verified against one and the same logic property (expressed in Action-based CTL)
   De Nicola, Vaandrager @ J. ACM, 1995

2. A logic property (expressed in variability-aware ACTL) can directly be verified against the MTS, relying on the fact that under certain syntactic conditions validity over the MTS implies validity of the same property for all its derived products (more later)
   ter Beek et al. @ J. Logic Algebr. Meth. Program., 2016

VMC v6.4 is freely usable online:

http://fmt.isti.cnr.it/vmc/
Variability analysis strategies supported by VMC

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 LTS can all be verified against one and the same logic property (expressed in Action-based CTL)
   De Nicola, Vaandrager @ J. ACM, 1995

2. A logic property (expressed in variability-aware ACTL) can directly be verified against the MTS, relying on the fact that under certain syntactic conditions validity over the MTS implies validity of the same property for all its derived products (more later)
   ter Beek et al. @ J. Logic Algebr. Meth. Program., 2016

VMC v6.4 is freely usable online:
http://fmt.isti.cnr.it/vmc/
Variability analysis strategies supported by VMC

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 LTS can all be verified against one and the same logic property (expressed in Action-based CTL)
   De Nicola, Vaandrager @ J. ACM, 1995

2. A logic property (expressed in variability-aware ACTL) can directly be verified against the MTS, relying on the fact that under certain syntactic conditions validity over the MTS implies validity of the same property for all its derived products (more later)
   ter Beek et al. @ J. Logic Algebr. Meth. Program., 2016

VMC v6.4 is freely usable online:

http://fmt.isti.cnr.it/vmc/
Vending Machine: family-based analysis

VMC notifies whenever preservation of an analysis result is applicable

It is not possible that serveTea occurs without being preceded by tea
Vending Machine: product-based analysis

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

Whenever pay occurs, eventually takePaid occurs.
Recall: Action-based CTL

Action formulae (Boolean compositions of actions, denoted by \( \chi \)), state formulae (\( \phi \)) and path formulae (\( \pi \))

\[
\phi ::= \text{true} \mid \neg \phi \mid \phi \land \phi \mid [\chi] \phi \mid (\chi) \phi \mid E \pi \mid A \pi \mid \\
\mu \ Y.\phi(Y) \mid \upsilon \ Y.\phi(Y)
\]

\[
\pi ::= X \{\chi\} \phi \mid [\phi \{\chi\} U \{\chi'\} \phi'] \mid [\phi \{\chi\} W \{\chi'\} \phi'] \mid \\
[\phi \{\chi\} U \phi'] \mid [\phi \{\chi\} W \phi'] \mid F \phi \mid F \{\chi\} \phi \mid G \phi \mid G \{\chi\} \phi
\]

(\( \phi(Y) \) is syntactically monotone in \( Y \))

\( \mu \) and \( \upsilon \): recursion (greatest fixed point and least fixed point)

\( X, U, W, F, G \): action-based versions of neXt, Until, Weak until, Future (“eventually”), Globally (“always”)
Recall: Action-based CTL

Action formulae (Boolean compositions of actions, denoted by $\chi$), state formulae ($\phi$) and path formulae ($\pi$)

$\phi ::= \text{true} | \neg \phi | \phi \land \phi | [\chi] \phi | \langle \chi \rangle \phi | E \pi | A \pi |

\[ \mu Y.\phi(Y) | \nu Y.\phi(Y) \]

$\pi ::= X \{\chi\} \phi | [\phi \{\chi\} U \{\chi'\} \phi'] | [\phi \{\chi\} W \{\chi'\} \phi'] |

[\phi \{\chi\} U \phi'] | [\phi \{\chi\} W \phi'] | F \phi | F \{\chi\} \phi | G \phi | G \{\chi\} \phi$

($\gamma$ is a propositional variable, $\phi(\gamma)$ is syntactically monotone in $\gamma$)

$\mu$ and $\nu$: recursion (greatest fixed point and least fixed point)

$X, U, W, F, G$: action-based versions of neXt, Until, Weak until, Future ("eventually"), Globally ("always")
Recall: Action-based CTL

Action formulae (Boolean compositions of actions, denoted by $\chi$), state formulae ($\phi$) and path formulae ($\pi$)

$$
\phi ::= \text{true} \mid \neg \phi \mid \phi \land \phi \mid [\chi] \phi \mid \langle \chi \rangle \phi \mid E \pi \mid A \pi \mid \\
\mu Y. \phi(Y) \mid \nu Y. \phi(Y)
$$

$$
\pi ::= X \{\chi\} \phi \mid [\phi \{\chi\} U \{\chi\} \phi'] \mid [\phi \{\chi\} W \{\chi\} \phi'] \mid \\
[\phi \{\chi\} U \phi'] \mid [\phi \{\chi\} W \phi'] \mid F \phi \mid F \{\chi\} \phi \mid G \phi \mid G \{\chi\} \phi
$$

($Y$ is a propositional variable, $\phi(Y)$ is syntactically monotone in $Y$)

$\mu$ and $\nu$: recursion (greatest fixed point and least fixed point)

$X, U, W, F, G$: action-based versions of $\text{neXt}$, Until, Weak until, Future (“eventually”), Globally (“always”)
v-ACTL: variability-aware, action-based CTL

Additional variability-aware (action-based) versions of Box, Diamond, neXt, Future, Globally

\[ \langle \chi \rangle^\Box \phi \] a next state exists, reachable by a must transition, executing an action satisfying \( \chi \), in which \( \phi \) holds

\[ [\chi]^\Box \phi \] in all next states reachable by a must transition, executing an action satisfying \( \chi \), \( \phi \) holds

\[ X^\Box \phi \] in the next state in the path, reached by a must transition

\[ (X^\Box \{ \chi \} \phi) \] (and executing an action satisfying \( \chi \)), \( \phi \) holds

\[ F^\Box \phi \] there exists a future state in the path in which \( \phi \) holds

\[ (F^\Box \{ \chi \} \phi) \] (reached by executing an action satisfying \( \chi \))

\[ G^\Box \phi \equiv \] in all states in the path, \( \phi \) holds, and all transitions

\[ \neg F^\Box \neg \phi \] are must transitions
v-ACTL: variability-aware, action-based CTL

Additional variability-aware (action-based) versions of Box, Diamond, neXt, Future, Globally

\[\langle \chi \rangle \Box \phi \text{ a next state exists, reachable by a must transition, executing an action satisfying } \chi, \text{ in which } \phi \text{ holds}\]

\[[\chi] \Box \phi \text{ in all next states reachable by a must transition, executing an action satisfying } \chi, \phi \text{ holds}\]

\[X \Box \phi \text{ in the next state in the path, reached by a must transition}\]

\[(X \Box \{\chi\} \phi) \text{ (and executing an action satisfying } \chi), \phi \text{ holds}\]

\[F \Box \phi \text{ there exists a future state in the path in which } \phi \text{ holds}\]

\[(F \Box \{\chi\} \phi) \text{ (reached by executing an action satisfying } \chi)\]

\[\text{and all transitions until that state are must transitions}\]

\[G \Box \phi \equiv \text{ in all states in the path, } \phi \text{ holds, and all transitions}\]

\[\neg F \Box \neg \phi \text{ are must transitions}\]
v-ACTL: variability-aware, action-based CTL

Additional variability-aware (action-based) versions of Box, Diamond, neXt, Future, Globally

\(\langle \chi \rangle \Box \phi\) a next state exists, reachable by a must transition, executing an action satisfying \(\chi\), in which \(\phi\) holds

\([\chi]\Box \phi\) in all next states reachable by a must transition, executing an action satisfying \(\chi\), \(\phi\) holds

\(X \Box \phi\) in the next state in the path, reached by a must transition (and executing an action satisfying \(\chi\)), \(\phi\) holds

\((X \Box \{\chi\} \phi)\) there exists a future state in the path in which \(\phi\) holds (reached by executing an action satisfying \(\chi\)) and all transitions until that state are must transitions

\(G \Box \phi \equiv \) in all states in the path, \(\phi\) holds, and all transitions

\((\neg F \Box \neg \phi)\) are must transitions
v-ACTL: variability-aware, action-based CTL

Additional variability-aware (action-based) versions of Box, Diamond, neXt, Future, Globally

\(\langle \chi \rangle \Box \phi\) a next state exists, reachable by a must transition, executing an action satisfying \(\chi\), in which \(\phi\) holds

\([\chi] \Box \phi\) in all next states reachable by a must transition, executing an action satisfying \(\chi\), \(\phi\) holds

\(X \Box \phi\) in the next state in the path, reached by a must transition (and executing an action satisfying \(\chi\)), \(\phi\) holds

\((X^{\Box} \{\chi\} \phi)\) there exists a future state in the path in which \(\phi\) holds (reached by executing an action satisfying \(\chi\)) and all transitions until that state are must transitions

\(G^{\Box} \phi \equiv \) in all states in the path, \(\phi\) holds, and all transitions

\(\neg F^{\Box} \neg \phi\) are must transitions
v-ACTL: variability-aware, action-based CTL

Additional variability-aware (action-based) versions of Box, Diamond, neXt, Future, Globally

\[
\langle \chi \rangle \Box \phi \quad \text{a next state exists, reachable by a must transition,}
\]
\[
\text{executing an action satisfying } \chi, \text{ in which } \phi \text{ holds}
\]
\[
[\chi] \Box \phi \quad \text{in all next states reachable by a must transition,}
\]
\[
\text{executing an action satisfying } \chi, \ \phi \text{ holds}
\]
\[
X \Box \phi \quad \text{in the next state in the path, reached by a must transition}
\]
\[
(X \Box \{\chi\} \phi) \quad \text{(and executing an action satisfying } \chi), \ \phi \text{ holds}
\]
\[
F \Box \phi \quad \text{there exists a future state in the path in which } \phi \text{ holds}
\]
\[
(F \Box \{\chi\} \phi) \quad \text{(reached by executing an action satisfying } \chi)\text{, and all transitions until that state are must transitions}
\]
\[
G \Box \phi \equiv \quad \text{in all states in the path, } \phi \text{ holds, and all transitions}
\]
\[
\neg F \Box \neg \phi \quad \text{are must transitions}
\]
Preservation of formulae in $\nu$-ACTL$^\square$/\nu-ACTLive$^\square$

$\nu$-ACTL$^\square$/\nu-ACTLive$^\square$:

\[
\phi ::= \text{false} \mid \text{true} \mid \phi \land \phi \mid \phi \lor \phi \mid [\chi] \phi \mid \langle \chi \rangle \Box \phi \\
EX \Box \phi \mid EX \Box \{ \chi \} \phi \mid EF \Box \phi \mid EF \Box \{ \chi \} \phi \\
AF \Box \phi \mid AF \Box \{ \chi \} \phi \mid AG \phi \mid AF \phi \mid AF \{ \chi \} \phi
\]

any formula that is true for MTS, is also true for all products (LTS)

$\nu$-ACTL$^\neg$:

\[
\psi ::= \text{false} \mid \text{true} \mid \psi \land \psi \mid \psi \lor \psi \mid [\chi] \Box \psi \mid \langle \chi \rangle \psi \\
EX \psi \mid EX \{ \chi \} \psi \mid EF \psi \mid EF \{ \chi \} \psi \mid AF \psi \mid AF \{ \chi \} \psi
\]

any formula that is false for MTS, is also false for all products (LTS)
Preservation of formulae in $v$-ACTL$^k$/$v$-ACTLIVE$^k$:

$v$-ACTL$^k$/$v$-ACTLIVE$^k$: 

$$
\phi ::= \text{false} \mid \text{true} \mid \phi \land \phi \mid \phi \lor \phi \mid [\chi] \phi \mid \langle \chi \rangle \Box \phi \mid \\
EX \phi \mid EX \{\chi\} \phi \mid EF \phi \mid EF \{\chi\} \phi \mid \\
AF \phi \mid AF \{\chi\} \phi \mid AG \phi \mid AF \phi \mid AF \{\chi\} \phi
$$

any formula that is true for MTS, is also true for all products (LTS)

$v$-ACTL$^-$:

$$
\psi ::= \text{false} \mid \text{true} \mid \psi \land \psi \mid \psi \lor \psi \mid [\chi] \Box \psi \mid \langle \chi \rangle \psi \mid \\
EX \phi \mid EX \{\chi\} \phi \mid EF \psi \mid EF \{\chi\} \psi \mid AF \phi \mid AF \{\chi\} \phi
$$

any formula that is false for MTS, is also false for all products (LTS)
Live states use SPL-specific information

\[ S \models \phi \Rightarrow S_p \models \phi \quad \forall \text{ product LTS } S_p \text{ of MTS } S \]

Recall: all (reachable) must transitions are preserved \((\langle \rangle, [], X, F)\)

Live action sets define live states (not occur as final in any product)

In any product in which \(p\) occurs, \(p\) has at least one outgoing transition

\(\Rightarrow p\) is a live state, since \(a\ OR b\) gives rise to a live action set \(\{a, b\}\)
Live states use SPL-specific information

\[ S \models \phi \Rightarrow S_p \models \phi \quad \forall \text{ product LTS } S_p \text{ of MTS } S \]

Recall: all (reachable) must transitions are preserved \((\langle \rangle, [], X, F)\)

Live action sets define live states (not occur as final in any product)

In any product in which \(p\) occurs, \(p\) has at least one outgoing transition

\[ p \rightarrow \text{a OR b} \]

\(\Rightarrow p\) is a live state, since \(a\) OR \(b\) gives rise to a live action set \(\{a, b\}\)
Live states use SPL-specific information

\[ S \models \phi \Rightarrow S_p \models \phi \quad \forall \text{ product LTS } S_p \text{ of MTS } S \]

Recall: all (reachable) must transitions are preserved \((\langle \rangle \boxdot, [], X \boxdot, F \boxdot)\)

Live action sets define live states (not occur as final in any product)

In any product in which \( p \) occurs, \( p \) has at least one outgoing transition

\( \Rightarrow p \) is a live state, since \( a \) OR \( b \) gives rise to a live action set \( \{a, b\} \)
Live states use SPL-specific information

\[ S \models \phi \Rightarrow S_p \models \phi \quad \forall \text{ product LTS } S_p \text{ of MTS } S \]

Recall: all (reachable) must transitions are preserved \((\langle\rangle^\square, [[\square], X^\square, F^\square)\)

Live action sets define live states (not occur as final in any product)

MTS

\[ \sim \quad p \quad a \rightarrow q \quad b \rightarrow r \quad \]

Assume

\[ a \text{ OR } b \quad \]

\[
\sim \quad p \quad a \rightarrow q \quad b \rightarrow r
\]

\[ \sim \quad p \quad b \rightarrow r \quad \]

In any product in which \(p\) occurs, \(p\) has at least one outgoing transition

\[ \Rightarrow p \text{ is a live state, since } a \text{ OR } b \text{ gives rise to a live action set } \{a, b\} \]
Live states use SPL-specific information

\[ S \models \phi \implies S_p \models \phi \quad \forall \text{ product LTS } S_p \text{ of MTS } S \]

Recall: all (reachable) must transitions are preserved \((\langle \rangle, [], X, F)\)

**Live action sets** define live states (not occur as final in any product)

\[ \text{MTS} \quad \sim \begin{array}{c} p \quad \xrightarrow{a} \quad q \\ \quad \xrightarrow{b} \quad r \end{array} \quad \text{Assume} \quad \begin{array}{c} a \quad \text{OR} \quad b \end{array} \quad \text{LTS} \quad \sim \begin{array}{c} p \quad \xrightarrow{a} \quad q \\ \quad \xrightarrow{b} \quad r \end{array} \]

In any product in which \( p \) occurs, \( p \) has at least one outgoing transition

\[ \Rightarrow p \text{ is a live state, since } a \text{ OR } b \text{ gives rise to a live action set } \{a, b\} \]
QFLan: a framework for quantitative modelling and analysis of highly (re)configurable systems

ter Beek et al. © IEEE TSE, 2018, FM’18

https://github.com/qflanTeam/QFLan/

These slides on QFLan are largely based on slides by Andrea Vandin
A simple coffee vending machine product line: attributed feature model with quantitative constraints

begin abstract features
  Machine Beverage CoffeeBased
end abstract features

begin feature predicates
  price = { Cappuccino = 7, Coffee = 5, Cocoa = 2, Tea = 5 }
end feature predicates

begin quantitative constraints
  ; price(Machine) <= 10 ;
end quantitative constraints

begin feature diagram
  Machine -> {?Cocoa, Beverage}
  Beverage -XOR- {CoffeeBased, Tea}
  CoffeeBased -OR- {Cappuccino, Coffee}
end feature diagram

begin concrete features
  Cocoa Tea Cappuccino Coffee
end concrete features

begin cross-tree constraints
  Cappuccino requires Coffee
  Tea excludes Cocoa
end cross-tree constraints
A simple coffee vending machine product line: probabilistic behaviour with action constraints

begin processes diagram
begin process dynamics

states = factory, deposit, operating, prepareCoffee, prepareCappuccino, prepareTea, prepareChocaccino

transitions =
//Operating
//Coffee
operating -(Coffee,3) -> prepareCoffee,
prepareCoffee -(serveCoffee,1) -> operating,
//Cappuccino
operating -(Cappuccino,3) -> prepareCappuccino,
prepareCappuccino -(serveCappuccino,1) -> operating,
//Chocaccino
operating -(chocaccino,2) -> prepareChocaccino,
prepareChocaccino -(serveChocaccino,1) -> operating,
//Tea
operating -(Tea,3) -> prepareTea,
prepareCappuccino -(serveTea,1) -> operating,

begin actions
sell deploy reconfigure chocaccino
serveCoffee serveCappuccino
serveChocaccino serveTea
end actions

begin action constraints
do(chocaccino) -> (has(cappuccino)
and has(Cocoa))
end action constraints

begin init
installedFeatures = { Coffee }
initialProcesses = dynamics
end init

begin variables
sold = 0
deploys = 0
end variables
Pinpoint the precise moment in which a coffee machine is sold

begin analysis
query = eval when {sold == 1.0} :
{ price(Machine) [delta=0.5],
  Coffee, Tea, Cappuccino, Cocoa
}
default delta=0.05
alpha = 0.05
parallelism = 1
end analysis

begin processes diagram
begin process dynamics
states = factory, deposit, operating, prepareCoffee,
        prepareCappuccino, prepareTea, prepareChocaccino

transitions =
//Factory
factory - (replace(Coffee, Tea), 2.0) -> factory,
factory - (install(Cocoa), 1.0) -> factory,
factory - (install(Cappuccino), 1.0) -> factory,
factory - (sell, 1; {sold=1}) -> deposit,

//Deposit
deposit - (install(Cappuccino), 2.0) -> deposit,
deposit - (uninstall(Cappuccino), 2.0) -> deposit,
deposit - (install(Cocoa), 2.0) -> deposit,
deposit - (uninstall(Cocoa), 2.0) -> deposit,
deposit - (deploy, 2; {deployes=deployes+1}) -> operating,
end processes diagram
Average price and probability of features of sold coffee machines

<table>
<thead>
<tr>
<th>Price (machine)</th>
<th>Coffee</th>
<th>Tea</th>
<th>Cappuccino</th>
<th>Cocoa</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;= 10</td>
<td>5.68</td>
<td>0.36</td>
<td>0.64</td>
<td>0.34</td>
</tr>
<tr>
<td>&lt;= 15</td>
<td>9.07</td>
<td>0.49</td>
<td>0.51</td>
<td>0.45</td>
</tr>
</tbody>
</table>
A simple coffee vending machine product line: statistical model checking with MultiVeSta

Average price and probability of features at varying of time

begin analysis
query = eval from 0 to 60 by 1 ;
{ price(Machine) [delta=0.5],
  Coffee, Tea, Cappuccino, Cocoa }
default delta=0.05
alpha = 0.05
parallelism = 1
end analysis
Dedicated vs. off-the-shelf model checkers

Recall: dedicated SPL model checkers like SNIP, VMC, QFLan, etc.

Dedicated model checkers need to be maintained and optimised

- Dimovski et al., Family-based model checking without a family-based model checker @ SPIN'15
- Chrzczen et al., Family-based modeling and analysis for probabilistic systems – featuring ProFeat @ FASE’16
- Dimovski et al., Variability-specific abstraction refinement for family-based model checking @ FASE’17
- Chrzczen et al., ProFeat: feature-oriented engineering for family-based probabilistic model checking @ FAC, 2018

Our approach towards family-based model checking with mCRL2:

- Product-based model checking of FTSs with mCRL2
  ter Beek & de Vink @ FormaliSE’14, SPLC’14

- Branching feature bisimulation for FTSs
  Belder, ter Beek & de Vink @ FMSPLE’15

- Feature-oriented modal $\mu$-calculi for reasoning on FTSs
  ter Beek, de Vink & Willemse @ FMSPLE’16

- Family-based model checking of FTSs with mCRL2 as is
  ter Beek, de Vink & Willemse @ FASE’17
Dedicated vs. off-the-shelf model checkers

Recall: dedicated SPL model checkers like SNIP, VMC, QFLan, etc.

Dedicated model checkers need to be maintained and optimised

- Dimovski et al., Family-based model checking without a family-based model checker @ SPIN’15
- Chrzson et al., Family-based modeling and analysis for probabilistic systems – featuring ProFeat @ FASE’16
- Dimovski et al., Variability-specific abstraction refinement for family-based model checking @ FASE’17
- Chrzson et al., ProFeat: feature-oriented engineering for family-based probabilistic model checking @ FAC, 2018

Our approach towards family-based model checking with mCRL2:

- Product-based model checking of FTSs with mCRL2
  ter Beek & de Vink @ FormaliSE’14, SPLC’14
- Branching feature bisimulation for FTSs
  Belder, ter Beek & de Vink @ FMSPLE’15
- Feature-oriented modal $\mu$-calculi for reasoning on FTSs
  ter Beek, de Vink & Willemse @ FMSPLE’16
- Family-based model checking of FTSs with mCRL2 as is
  ter Beek, de Vink & Willemse @ FASE’17
Dedicated vs. off-the-shelf model checkers

Recall: dedicated SPL model checkers like SNIP, VMC, QFLan, etc.

Dedicated model checkers need to be maintained and optimised

- Dimovski et al., Family-based model checking without a family-based model checker @ SPIN’15
- Chrzan et al., Family-based modeling and analysis for probabilistic systems – featuring ProFeat @ FASE’16
- Dimovski et al., Variability-specific abstraction refinement for family-based model checking @ FASE’17
- Chrzan et al., ProFeat: feature-oriented engineering for family-based probabilistic model checking @ FAC, 2018

Our approach towards family-based model checking with mCRL2:

- **Product-based model checking** of FTSs with mCRL2
  ter Beek & de Vink @ FormaliSE’14, SPLC’14

- **Branching feature bisimulation** for FTSs
  Belder, ter Beek & de Vink @ FMSPLE’15

- **Feature-oriented modal $\mu$-calculi** for reasoning on FTSs
  ter Beek, de Vink & Willemse @ FMSPLE’16

- **Family-based model checking** of FTSs with mCRL2 as is
  ter Beek, de Vink & Willemse @ FASE’17
Recall: a product line or family of coffee machines

- **Initially**, money must be inserted: either at least one euro’s worth in coins, *exclusively* for European products, or at least one dollar’s worth in coins, *exclusively* for Canadian products

- Input of money can be cancelled via a cancel button; *optionally*, the coffee machine returns change if more than one euro or one dollar was inserted

- Once the coffee machine contains at least one euro or one dollar, the user chooses whether (s)he wants sugar, by pressing one out of two buttons, after which (s)he can select a beverage

- The choice of beverage (coffee, tea, cappuccino) varies, but coffee *must* be offered by all products, whereas cappuccino *may* be offered *solely* by European products

- *Optionally*, a ringtone *may* be rung after delivering a beverage; a ringtone *must* be rung by *all* products offering cappuccino

- After the beverage is taken, the coffee machine returns idle

These slides on mCRL2 are largely based on slides by Erik de Vink
The coffee machine: attributed feature model

Recall:
Non-functional attributes: $\text{cost (product)} = \sum \{ \text{cost (feature)} | \text{feature} \in \text{product} \}$

From $2^{10} - 1 \xrightarrow{\text{feature diagram}} 2^5 \xrightarrow{\text{cross-tree constraints}} 20 \xrightarrow{\text{attributes}} 16$ valid products

total cost $\leq 30$
The coffee machine: family behaviour

FTS feature expression labels omitted for readability
mCRL2 code: declarations

sort
   Feature = struct M | S | O | R | B | X | D | E | P | T | C ;
   FSet = List( Feature );
Coin = struct dime | quarter | half | dollar |
      ct10 | ct20 | ct50 | euro ;
Currency = struct Dollar | Euro ;

act
   insert, return : Coin ;
   continue, cancel, sorry, no_change,
   sugar, no_sugar, coffee, tea, cappuccino,
   pour_sugar, pour_milk, pour_coffee, pour_tea,
   ring, skip, take_cup ;
   setS, setO, setR, setB, setX,
   setD, setE, setP, setT, setTP, setC ;
   set_ko, ctc_ok ;
   set_ok : FSet ;
   cost : Int ;
map isSorted: FSet -> Bool;
  noDuplicates: FSet -> Bool;
  setIs: FSet -> Bool;

var ft, ft': Feature;
  fset: FSet;

eqn isSorted([]) = true;
  isSorted([ft]) = true;
  isSorted(ft |> (ft' |> fset)) =
    ft <= ft' & isSorted(ft' |> fset);
noDuplicates([]) = true;
noDuplicates(ft |> fset) =
  !(ft in fset) & noDuplicates(fset);
isSet(fset) = isSorted(fset) & noDuplicates(fset);
map insert: Feature # FSet -> FSet;
var ft, ft’: Feature;
  fset: FSet;
eqn insert(ft, []) = [ft];
  (ft < ft’) -> insert(ft, ft’ |> fset) = ft |> ft’ |> fset;
  (ft == ft’) -> insert(ft, ft’ |> fset) = ft’ |> fset;
  (ft > ft’) -> insert(ft, ft’ |> fset) = ft’ |> insert(ft, fset);

map union: FSet # FSet -> FSet;
var ft, ft’: Feature;
  fset, fset’: FSet;
eqn union([], fset) = fset;
  union(fset, []) = fset;
  (ft < ft’) -> union(ft |> fset, ft’ |> fset’) =
    ft |> union(fset, ft’ |> fset’);
  (ft == ft’) -> union(ft |> fset, ft’ |> fset’) =
    ft’ |> union(fset, fset’);
  (ft > ft’) -> union(ft |> fset, ft’ |> fset’) =
    ft’ |> union(ft |> fset, fset’);
map fcost : Feature -> Int ;
eqn fcost(M) = 0 ;
    fcost(S) = 5 ;
    fcost(O) = 0 ;
    fcost(B) = 0 ;
    fcost(R) = 5 ;
    fcost(D) = 5 ;
    fcost(E) = 5 ;
    fcost(X) = 10 ;
    fcost(C) = 5 ;
    fcost(T) = 3 ;
    fcost(P) = 7 ;

map tcost : FSet -> Int ;
var ft : Feature ;
    fset : FSet ;
eqn tcost([]) = 0 ;
    tcost(ft |> fset) = fcost(ft) + tcost(fset) ;
init
  Sel(0,[M]) ;

proc Sel(st:Int,fs:FSet) =
  \% feature states
  ( st == 0 ) -> ( ( M in fs ) -> ( setS . Sel(1, insert(S,fs) ) ) ) +
  ( st == 1 ) -> ( ( M in fs ) -> ( setO . Sel(2, insert(0,fs) ) ) ) +
  ( st == 2 ) -> ( ( M in fs ) ->
    ( tau . Sel(3, fs ) + setR . Sel(3, insert(R,fs) ) ) ) +
  ( st == 3 ) -> ( ( M in fs ) -> ( setB . Sel(4, insert(B,fs) ) ) ) +
  ( st == 4 ) -> ( ( M in fs ) ->
    tau . Sel(5, fs ) + setX . Sel(5, insert(X,fs) ) ) +
  ( st == 5 ) -> ( ( 0 in fs ) -> ( setD . Sel(6, insert(D,fs) ) + setE . Sel(6, insert(E,fs) ) ) ) +
  ( st == 6 ) -> ( ( B in fs ) -> ( tau . Sel(7, fs ) + setT . Sel(7, insert(T,fs) ) +
    setP . Sel(7, insert(P,fs) ) + setTP . Sel(7, union([T,P],fs) ) ) ) ) +
...
... (st == 7) -> ( (B in fs) -> (setC . Sel(8, insert(C,fs))) )+
%%% cross-tree constraints
(st == 8) -> ( ((D in fs) && (P in fs)) -> 
    set_ko(fs) . deadlock <>
    (! (R in fs) && (P in fs)) ->
    set_ko(fs) . deadlock <> ctc_ok . Sel(9,fs)
)+
%%% attribute constraints
(st == 9) -> ( (tcost(fs) <= 30) ->
    set_ok(fs) . cost(tcost(fs)) . Prod(0,fs) <> set_ko . deadlock
);
mCRL2 code: product behaviour process Prod (1/2)

proc Prod(st: Int, fs: FSet) =
    (st == 0) -> (Insert(0, fs)) +
    (st == 1) -> ((S in fs) -> (sugar . Prod(2, fs)) +
                  (S in fs) -> (no_sugar . Prod(3, fs))) +
    (st == 2) -> ((C in fs) -> coffee . Prod(4, fs) +
                   (T in fs) -> tea . Prod(5, fs) +
                   (P in fs) -> cappuccino . Prod(6, fs)) +
    (st == 3) -> ((C in fs) -> coffee . Prod(9, fs) +
                   (T in fs) -> tea . Prod(8, fs) +
                   (P in fs) -> cappuccino . Prod(7, fs)) +
    (st == 4) -> ((M in fs) -> (pour_sugar . Prod(9, fs)) +
                   (st == 5) -> ((M in fs) -> (pour_sugar . Prod(8, fs)) +
                                  (st == 6) -> ((M in fs) -> (pour_sugar . Prod(7, fs)) +
                                  ...
mCRL2 code: product behaviour process Prod (2/2)

... 

( st == 7 ) -> ( 
( M in fs ) -> ( pour_milk . Prod(10,fs) ) + 
( M in fs ) -> ( pour_coffee . Prod(11,fs) ) ) + 
( st == 8 ) -> ( 
( M in fs ) -> ( pour_teal . Prod(12,fs) ) ) + 
( st == 9 ) -> ( 
( M in fs ) -> ( pour_coffee . Prod(12,fs) ) ) + 
( st == 10 ) -> ( 
( M in fs ) -> ( pour_coffee . Prod(12,fs) ) ) + 
( st == 11 ) -> ( 
( M in fs ) -> ( pour_milk . Prod(12,fs) ) ) + 
( st == 12 ) -> ( 
( R in fs ) -> ( ring . Prod(13,fs) ) + 
( !(R in fs) ) -> ( skip . Prod(13,fs) ) ) ) + 
( st == 13 ) -> ( 
( M in fs ) -> ( take_cup . Prod(0,fs) ) ) ;
proc Insert(bal:Nat,fs:FSet) =
  ( bal < 100 ) -> (  
    ( D in fs ) -> (  
      insert(dime) . Insert(bal+10,fs) +  
      insert(quarter) . Insert(bal+25,fs) +  
      insert(half) . Insert(bal+50,fs) +  
      insert(dollar) . Insert(bal+100,fs) ) +  
    ( E in fs ) -> (  
      insert(ct10) . Insert(bal+10,fs) +  
      insert(ct20) . Insert(bal+20,fs) +  
      insert(ct50) . Insert(bal+50,fs) +  
      insert(euro) . Insert(bal+100,fs) ) ) +  
  ( ( bal > 0 ) && ( bal < 100 ) ) ->  
  Return(bal,fs) . cancel . Prod(0,fs) +  
  ( bal >= 100 ) -> (  
    ( ( !(X in fs) ) ->  
      no_change . continue . Prod(1,fs) <>  
    Return(Int2Nat(bal-100),fs) .  
      continue . Prod(1,fs) )  
  );
mCRL2 code: money handling process Return

```ml
proc Return(bal:Nat, fs:FeatSet) =
  ( bal == 0 ) -> tau +
  ( D in fs ) -> ( 
    ( bal >= 50 ) ->
      return(half) . Return(Int2Nat(bal-50),fs) +
      ( ( bal < 50 ) && ( bal >= 25 ) ) ->
      return(quarter) . Return(Int2Nat(bal-25),fs) +
      ( ( bal < 25 ) && ( bal >= 10 ) ) ->
      return(dime) . Return(Int2Nat(bal-10),fs) +
      ( ( bal < 10 ) && ( bal > 0 ) ) ->
      sorry . Return(0,fs) ) +
  ( E in fs ) -> ( 
    ( bal >= 50 ) ->
      return(ct50) . Return(Int2Nat(bal-50),fs) +
      ( ( bal < 50 ) && ( bal >= 20 ) ) ->
      return(ct20) . Return(Int2Nat(bal-20),fs) +
      ( ( bal < 20 ) && ( bal > 0 ) ) ->
      return(ct10) . Return(Int2Nat(bal-10),fs) +
      ( ( bal < 10 ) && ( bal > 0 ) ) ->
      sorry . Return(0,fs)
  )
```

Visual inspection of the state space
Visual inspection of the state space: behavioural reduction
Product behaviour abstracting from . . .
configuration and payment
Product-based analysis with mCRL2: example properties

- **[(!continue)*.take_cup] false**
  if payment is not settled, no beverage is delivered
- **[true*.setX.true*.no_change] false**
  once feature X is selected, no_change will not occur
- **[true*] forall fs:FSet.**
  \( <\text{set\_ok}(fs)> \) (val((D in fs) => !(P in fs)))
  if a product is configured successfully,
  then it cannot accept dollars and also provide cappuccino
- **mu Y.\(<\exists\text{fs:FSet.set\_ok}(fs)\>) true ||**
  \(<\exists\text{fs:FSet.set\_ko}(fs)\>) true || [true] Y)
  eventually either set\_ok or set\_ko can occur
- **forall c:Coin.[true*.insert(c)]**
  mu Y.\(<\text{cancel}||\text{take\_cup}>\) true || [true] Y)
  after money is inserted, eventually a beverage is given
  or money insertion is cancelled
- **[true*]forall n:Nat.[cost(n)](val(n <= 30))**
  all valid products have a total cost at most 30
Outlook

Next talk: Family-Based Model Checking with mCRL2 (Erik de Vink)

Current hot topic: Quantitative Variability Modelling and Analysis

- Quantitative specification and verification techniques for systems with variability
- Modelling and analysis of real-time, hybrid or probabilistic systems with variability
- Analysis of safety, security or dependability properties of systems with variability
- Modelling and analysis of dynamic, adaptive and (runtime) reconfigurable systems

- Variable UML Sequence Diagrams, Performance-Annotated UML Activity Diagrams
- Featured Timed Automata, Configurable Parametric Timed Automata, Featured Modal Contract Automata, Featured Weighted Automata, Priced Featured Automata,
- Probabilistic/Statistical Model Checking, SMT solving
  - SNIP-Z3, ProFeat, QFLan
Outlook

Next talk: Family-Based Model Checking with mCRL2 (Erik de Vink)

👉 Current hot topic: Quantitative Variability Modelling and Analysis
  ter Beek & Legay @ QSPL special section, LNCS FoMaC, 2018

- Quantitative specification and verification techniques for systems with variability
- Modelling and analysis of real-time, hybrid or probabilistic systems with variability
- Analysis of safety, security or dependability properties of systems with variability
- Modelling and analysis of dynamic, adaptive and (runtime) reconfigurable systems

- Variable UML Sequence Diagrams, Performance-Annotated UML Activity Diagrams
- Featured Timed Automata, Configurable Parametric Timed Automata, Featured Modal Contract Automata, Featured Weighted Automata, Priced Featured Automata,
- Probabilistic/Statistical Model Checking, SMT solving
  - SNIP-Z3, ProFeat, QFLan
Outlook

Next talk: Family-Based Model Checking with mCRL2 (Erik de Vink)

🔥 Current hot topic: **Quantitative Variability Modelling and Analysis**

- ter Beek & Legay @ QSPL special section, *LNCS FoMaC, 2018*
- Quantitative specification and verification techniques for systems with variability
- Modelling and analysis of real-time, hybrid or probabilistic systems with variability
- Analysis of safety, security or dependability properties of systems with variability
- Modelling and analysis of dynamic, adaptive and (runtime) reconfigurable systems

- Variable UML Sequence Diagrams, Performance-Annotated UML Activity Diagrams
- Featured Timed Automata, Configurable Parametric Timed Automata, Featured Modal Contract Automata, Featured Weighted Automata, Priced Featured Automata,
- Probabilistic/Statistical Model Checking, SMT solving
  - SNIP-Z3, ProFeat, QFLan