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

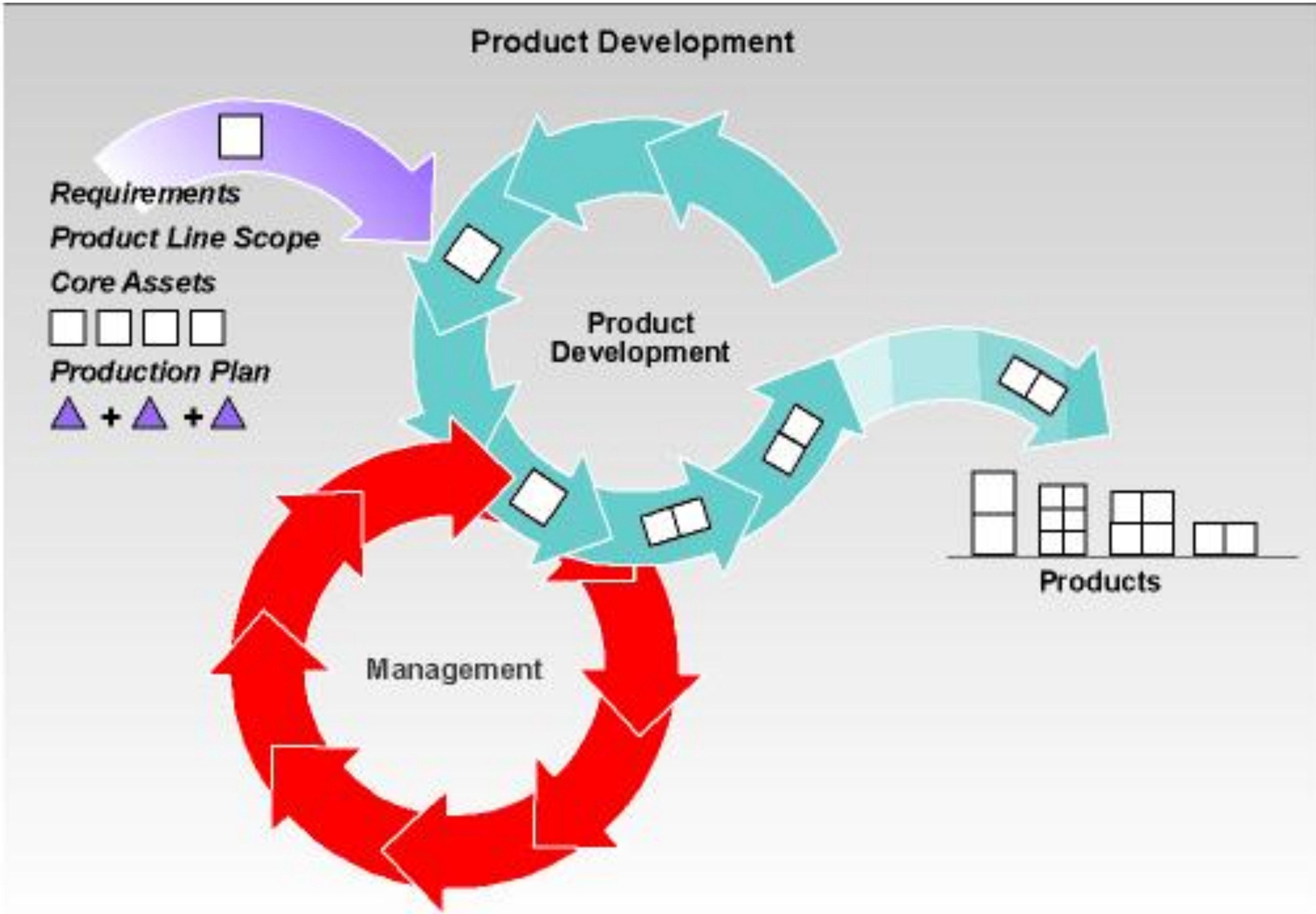
1. Introduction to Software Product Lines
2. Modelling Variability
   - Featured Transition Systems (fPROMELA, fSMV, fLTL, fCTL)
   - Modal Transition Systems (with variability constraints, v-ACTL)
   - Flan, PFLan, QFLan, ProFeat
3. Analysing Variability
   - SNIP, ProVeLines, fNuSMV, SPIN, mCRL2
   - 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

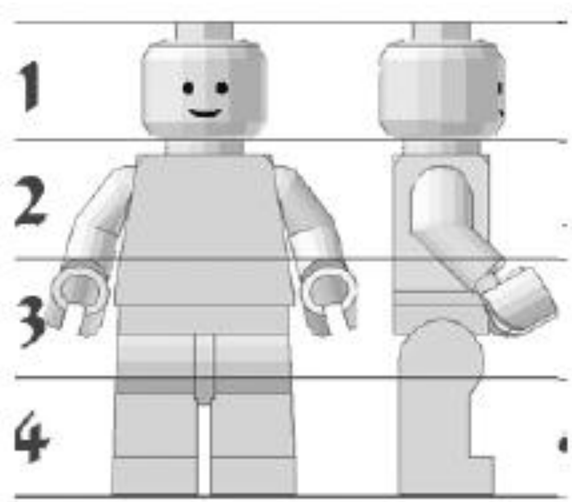
- What is a feature?

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



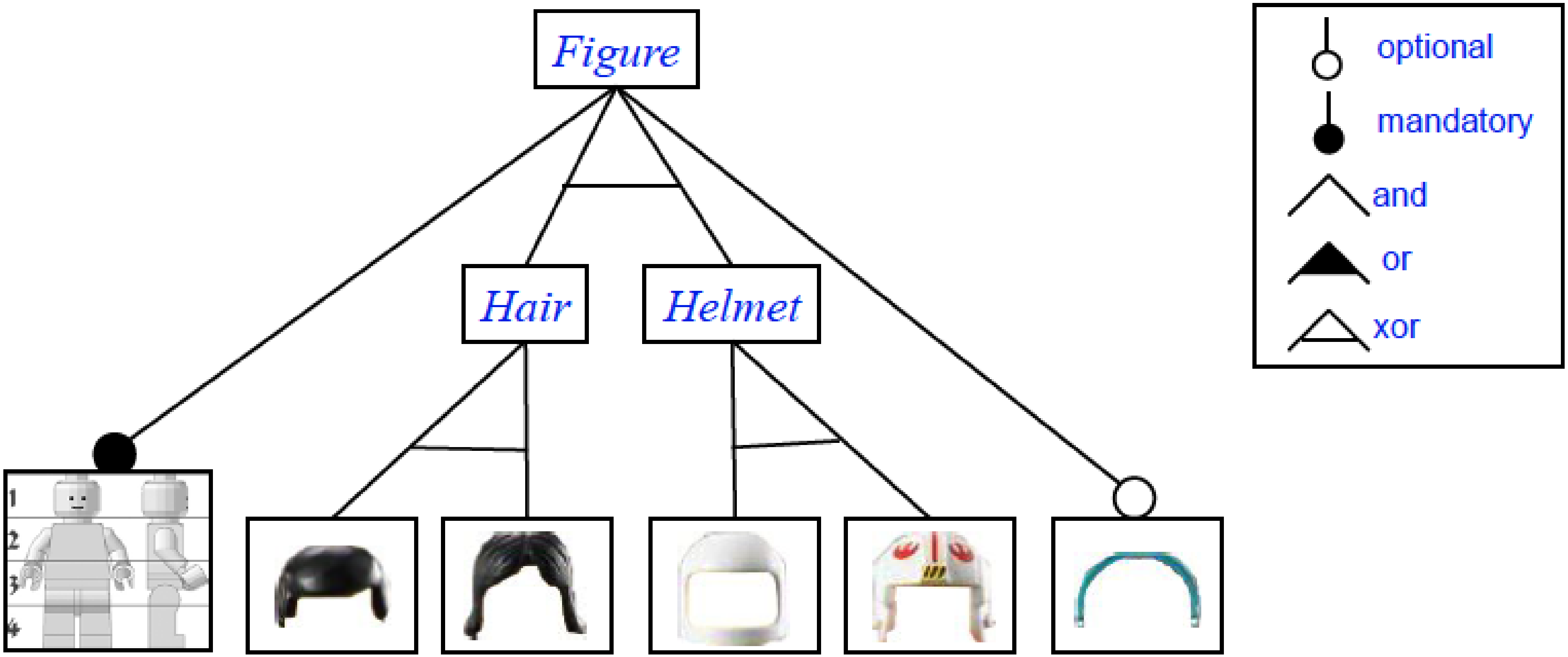
What's in a *feature*?

Reference	Definition
Kang *et al.* [3]	*"a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems"*
Kang *et al.* [8]	*"distinctively identifiable functional abstractions that must be implemented, tested, delivered, and maintained"*
Eisenecker and Czarnecki [6]	*"anything users or client programs might want to control about a concept"*
Bosch *et al.* [9]	*"A logical unit of behaviour specified by a set of functional and non-functional requirements."*
Chen *et al.* [10]	*"a product characteristic from user or customer views, which essentially consists of a cohesive set of individual requirements"*
Batory [11]	*"an elaboration or augmentation of an entity(s) that introduces a new service, capability or relationship"*
Batory *et al.* [12]	*"an increment in product functionality"*
Apel *et al.* [13]	*"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."*

(Classen *et al.* @ FASE'08)

Feature selection

Typically, only a subset of feature combinations is valid



$Visor \Rightarrow Helmet$

Products and product lines

- Product ⇔ 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 SPLE
  - 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 SPLE
  - 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
  - 1992-2009 International Conference on Feature Interactions in Telecommunications and Software Systems

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);

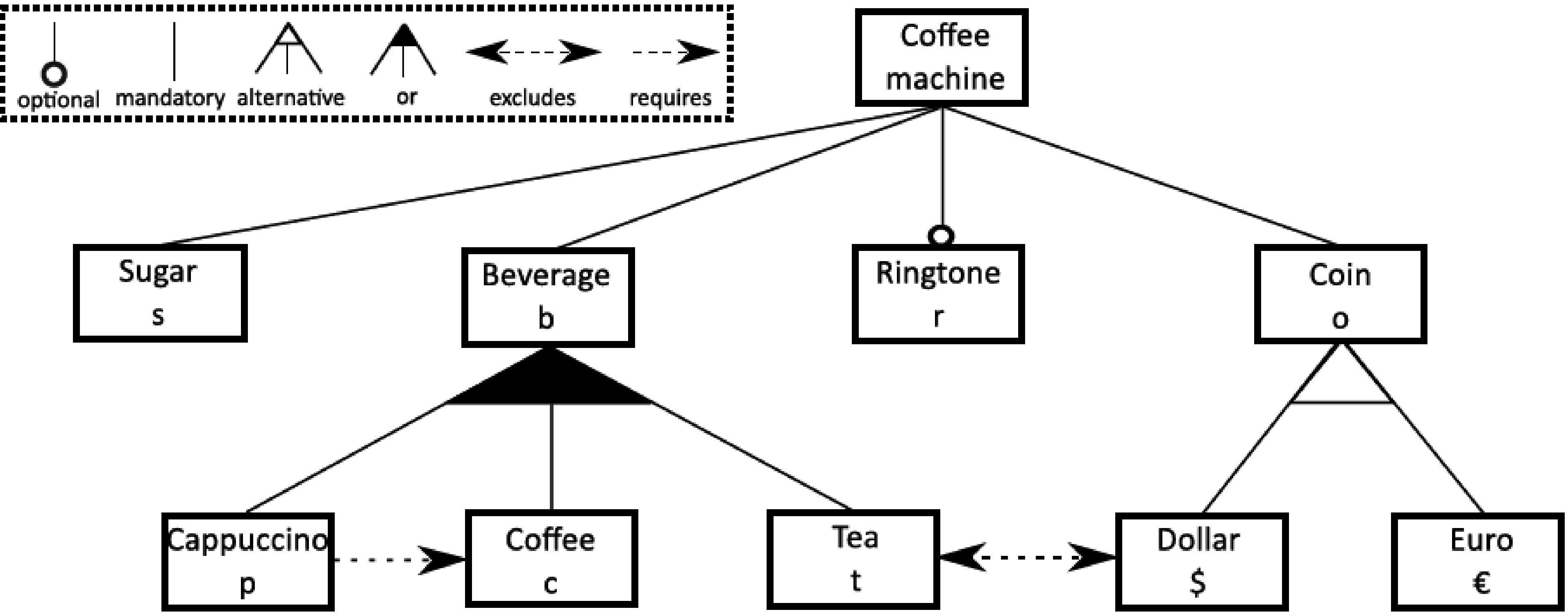
   furthermore, whenever a coffee machine offers cappuccino, then it must offer coffee as well, while tea may only be offered by coffee machines for the European market (requires and excludes relations among features)
4. Optionally, a ringtone may be rung after the coffee machine has delivered the chosen beverage (optional feature)

Example requirements coffee machine

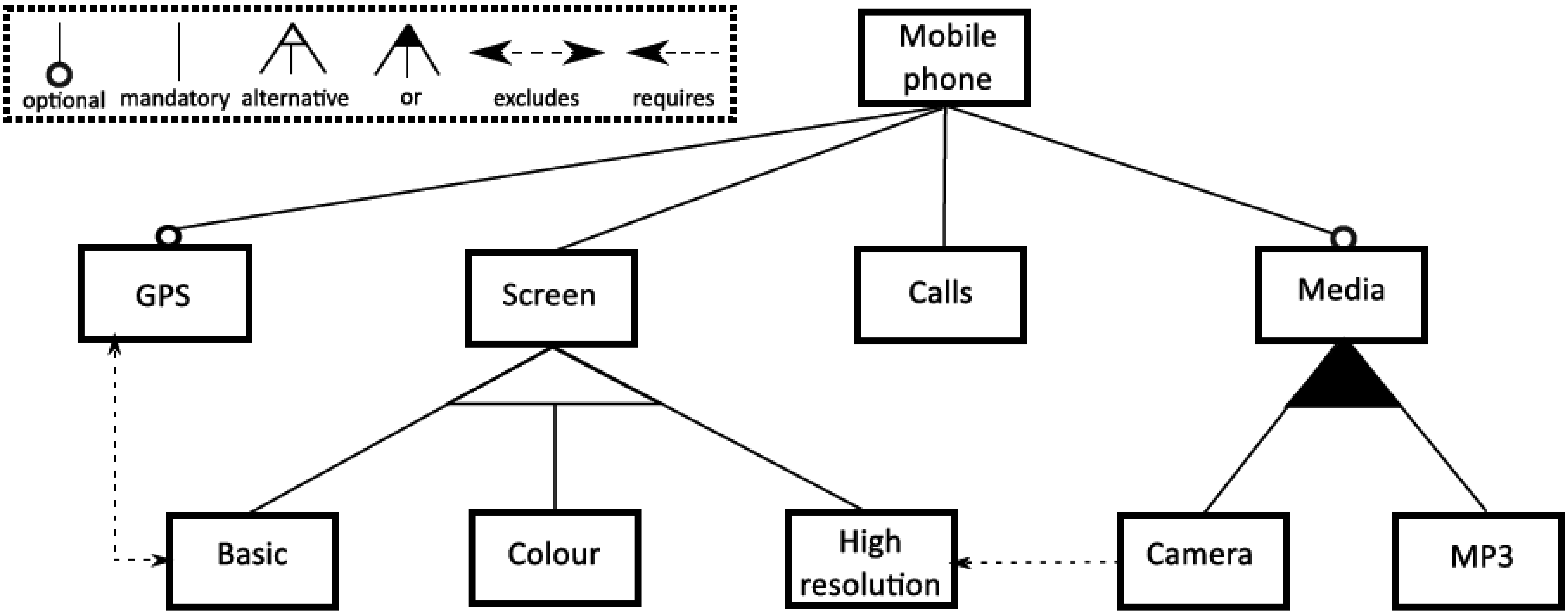
1. Initially, a coin must be inserted: either a €, exclusively in case of coffee machines for the European market, or a $, exclusively in case of coffee machines for the Canadian market (alternative features)
2. After having inserted a coin, the user has to be offered the option to choose whether or 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

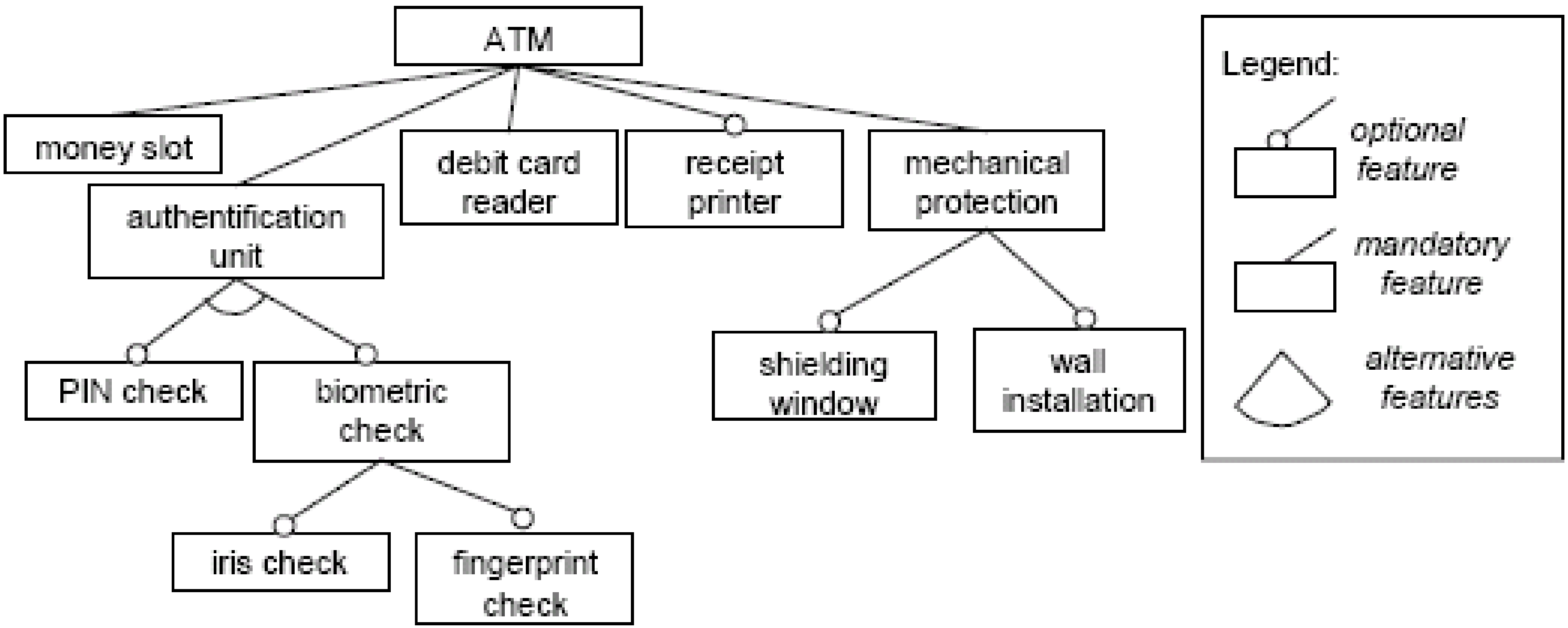
Coffee machine feature diagram



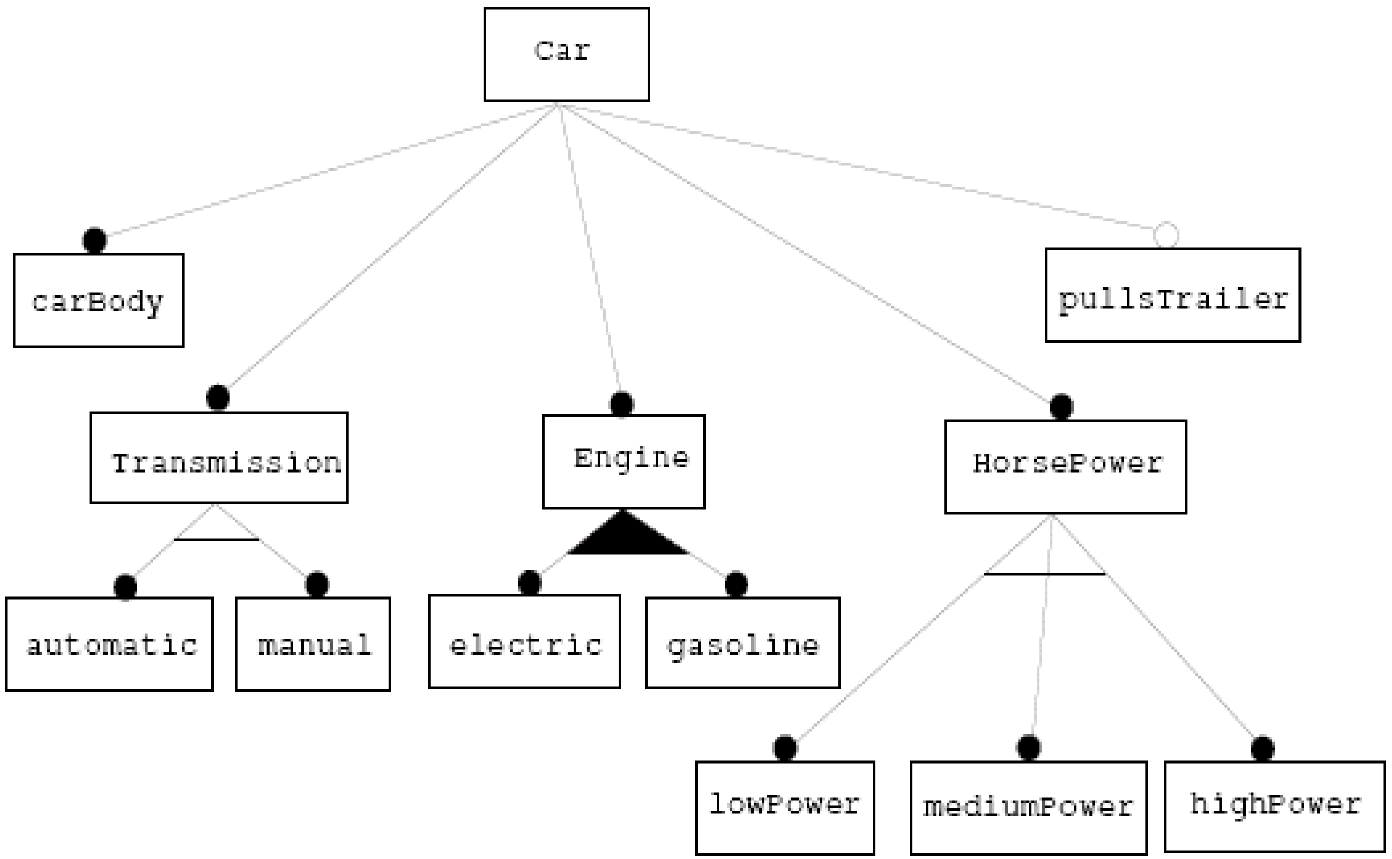
Mobile phone feature diagram



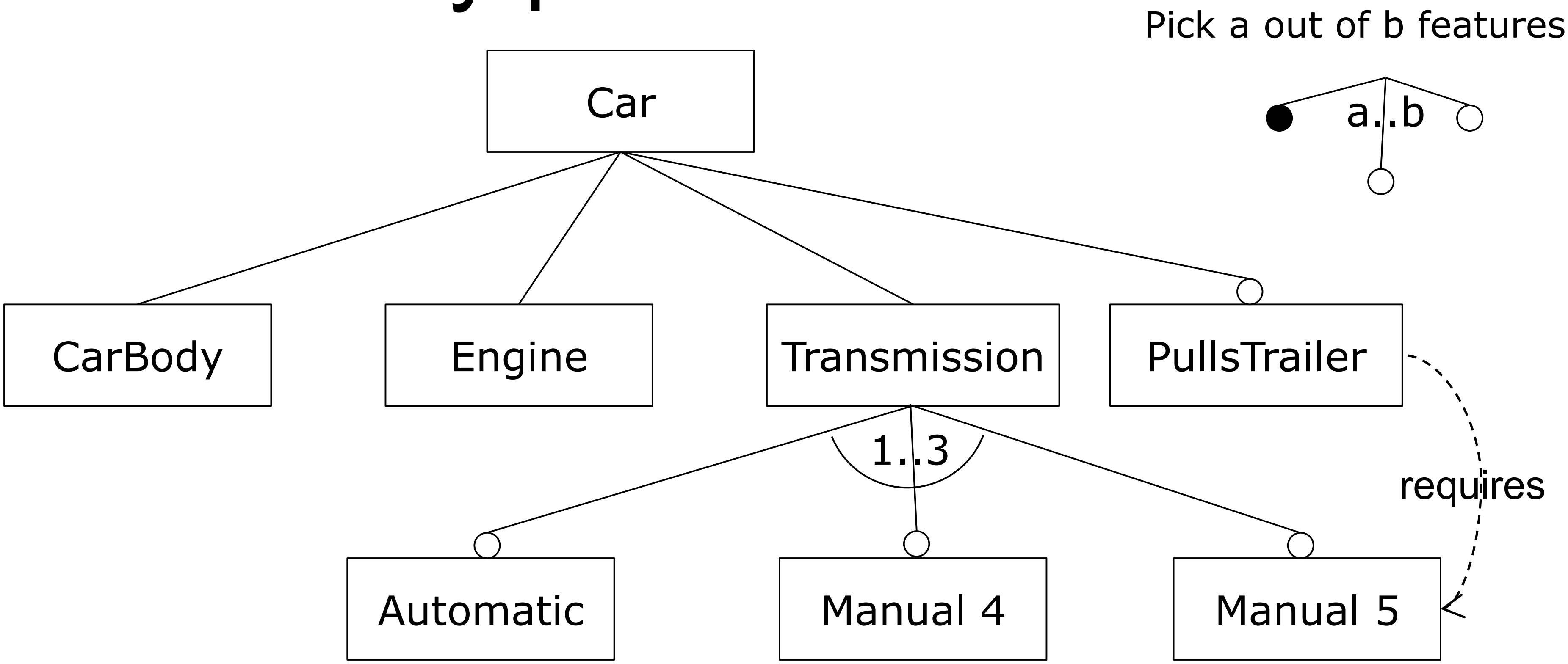
ATM feature diagram



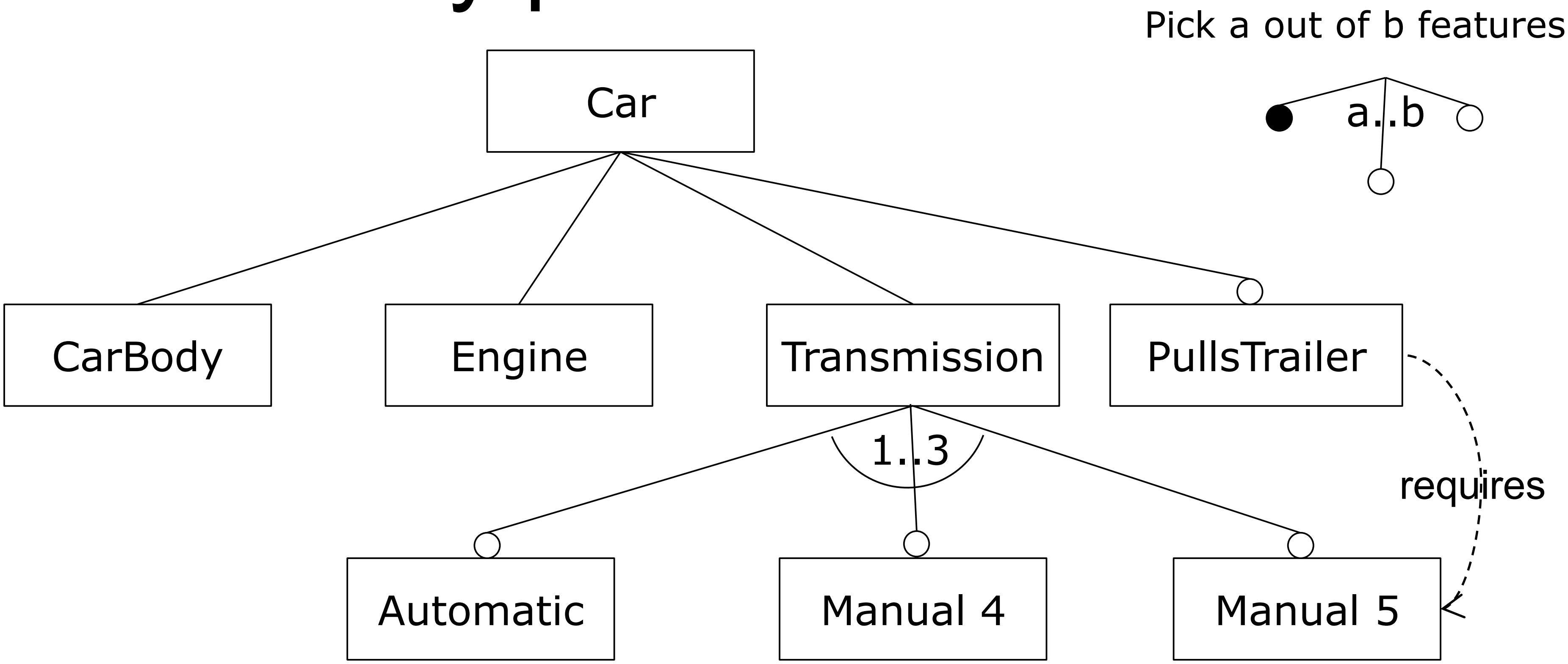
Automotive feature diagram



How many products?

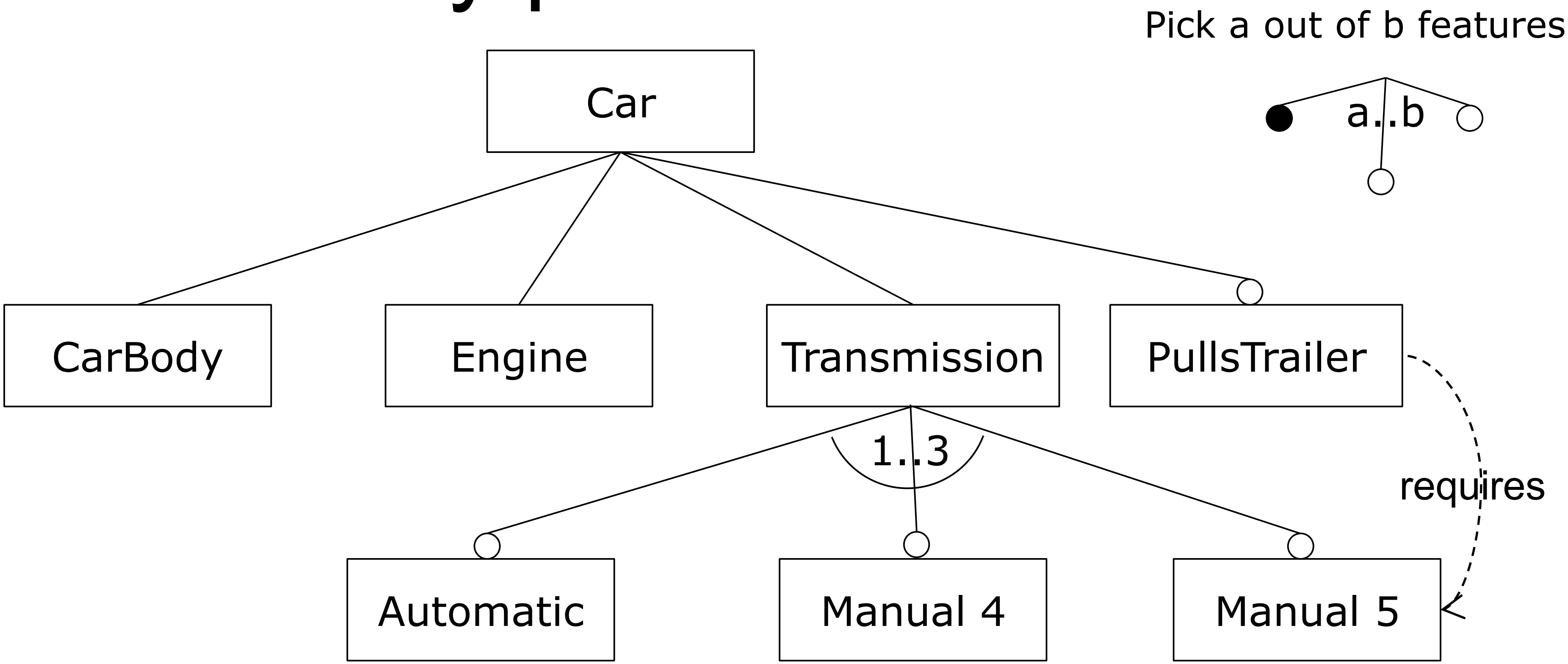


How many products?



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

How many products?

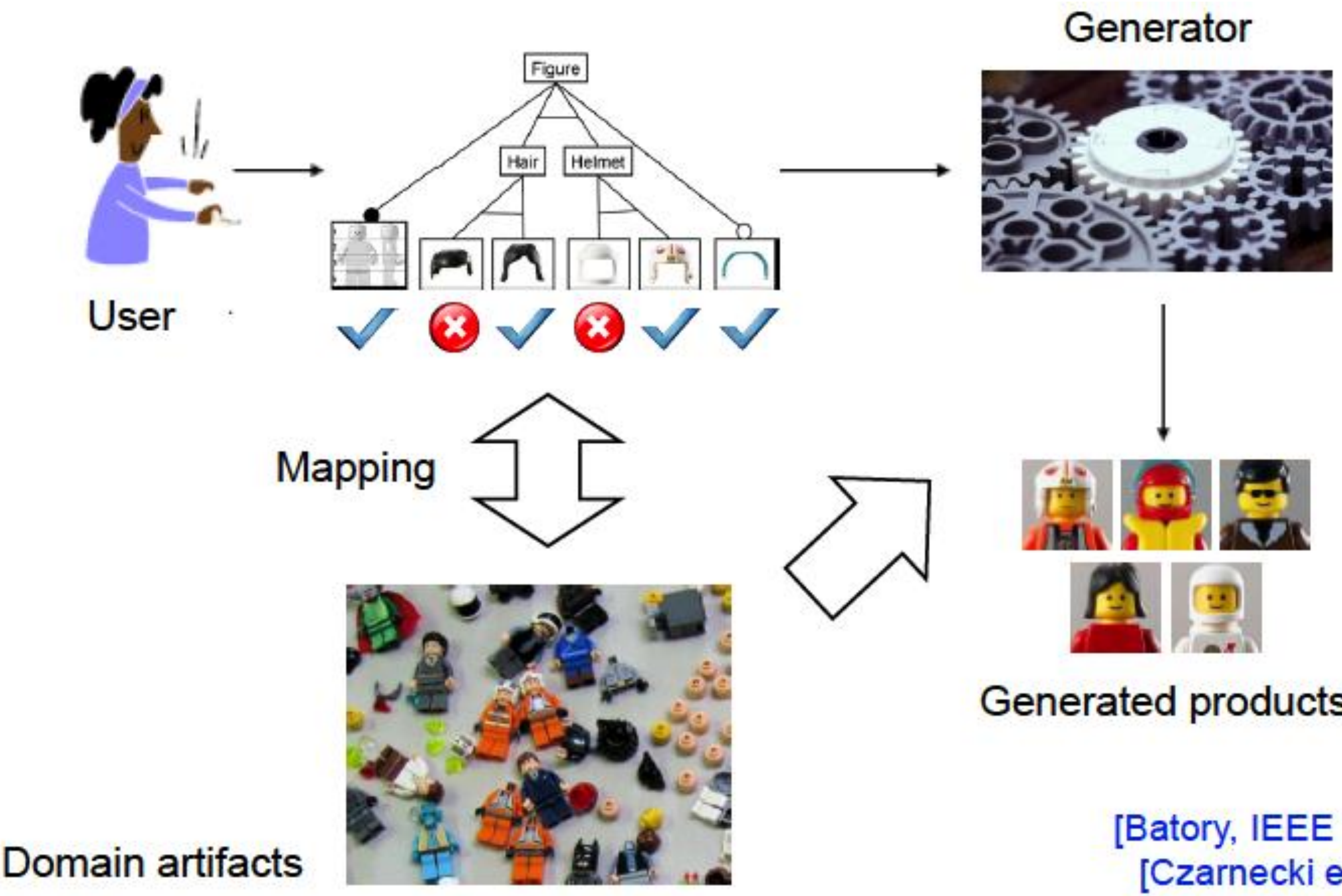


Transmission contributes one of three options, multiplier = 3
Car contributes two options, multiplier = 2, but…
…actually only works in one case; thus: 4 different products

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.

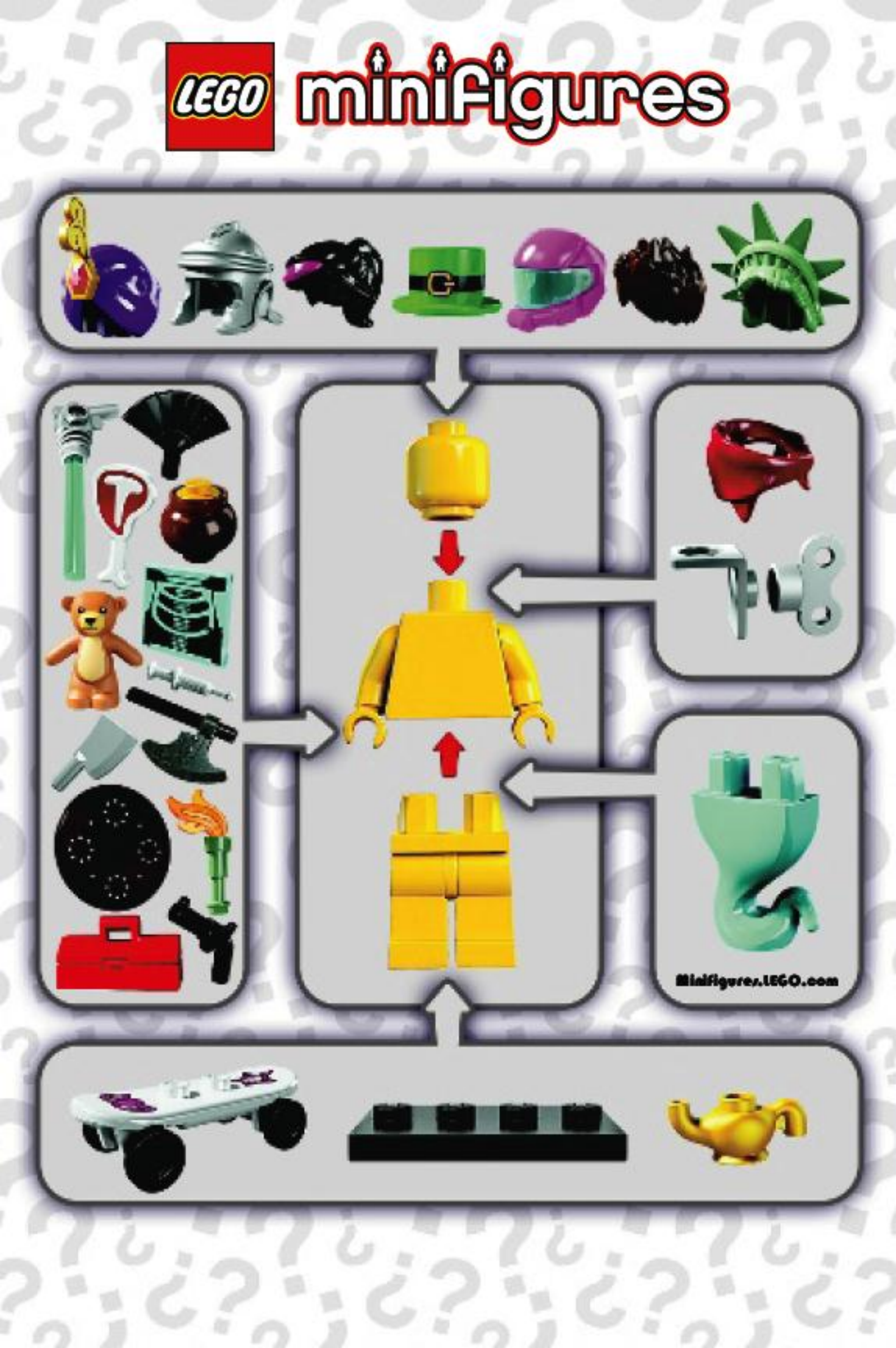
… Automated product derivation



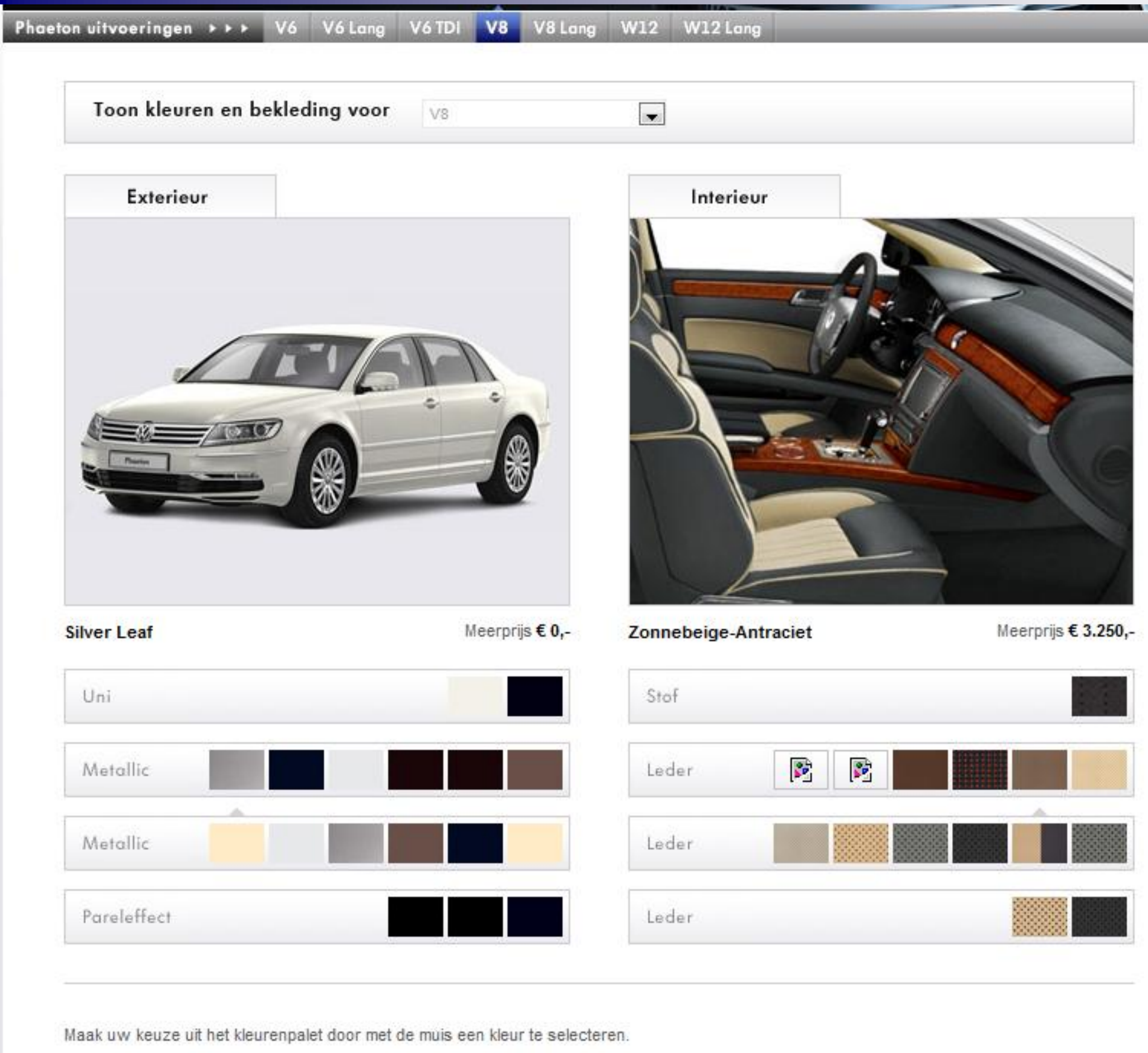
[Batory, IEEE TSE 2004]
[Czarnecki et al., 2000]

A product line of Lego minifigures (16 valid feature combinations)

‘Feature diagram’ for configuring the 16 valid products (allowing more…)

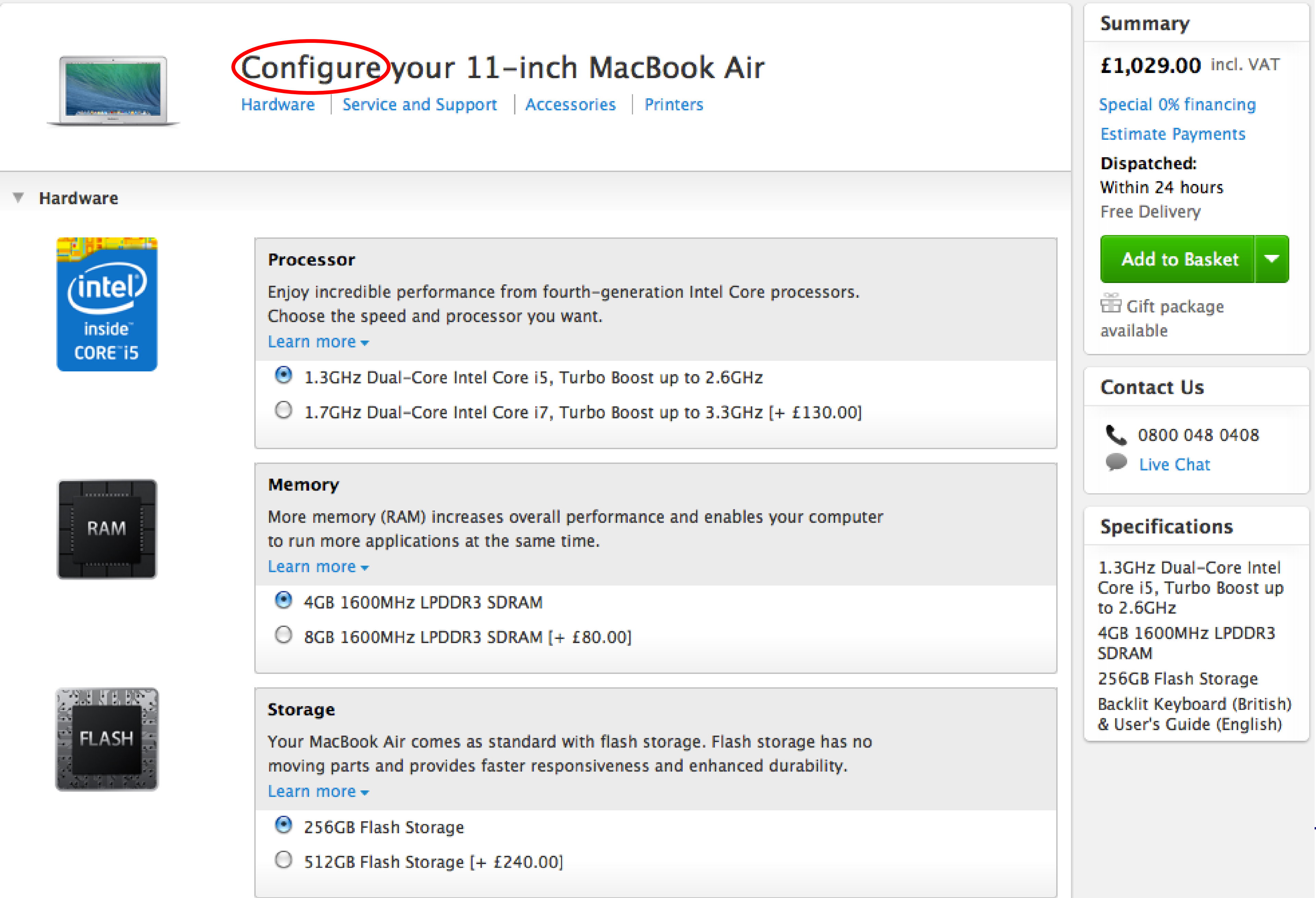


VW features



Phaeton uitvoeringen
V6
V6 Lang
V6 TDI
V8
V8 Lang
W12
W12 Lang
Toon kleuren en bekleding voor
V8
Exterieur
Interieur
Silver Leaf
Meerprijs € 0,-
Zonnebeige-Antraciet
Meerprijs € 3.250,-
Uni
Metallic
Metallic
Pareleffect
Stof
Leder
Leder
Leder
Maak uw keuze uit het kleurenpalet door met de muis een kleur te selecteren.

Introduction to SPL



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

The international BMW website

Sheer Driving Pleasure

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

Configure vehicle

Configure your BMW vehicle

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

United Kingdom

Related topics

Request information

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

Arrange a test drive

Make a date for a test drive with your BMW dealer.

FIND YOUR BMW.

Filter

> Reset filter

Budget

Vehicle type

- All
- Petrol
- Diesel
- Hybrid
- Electric Vehicle

Body type

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

Number of seats

30 Vehicles (465 Model variants)

1

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

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

2

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

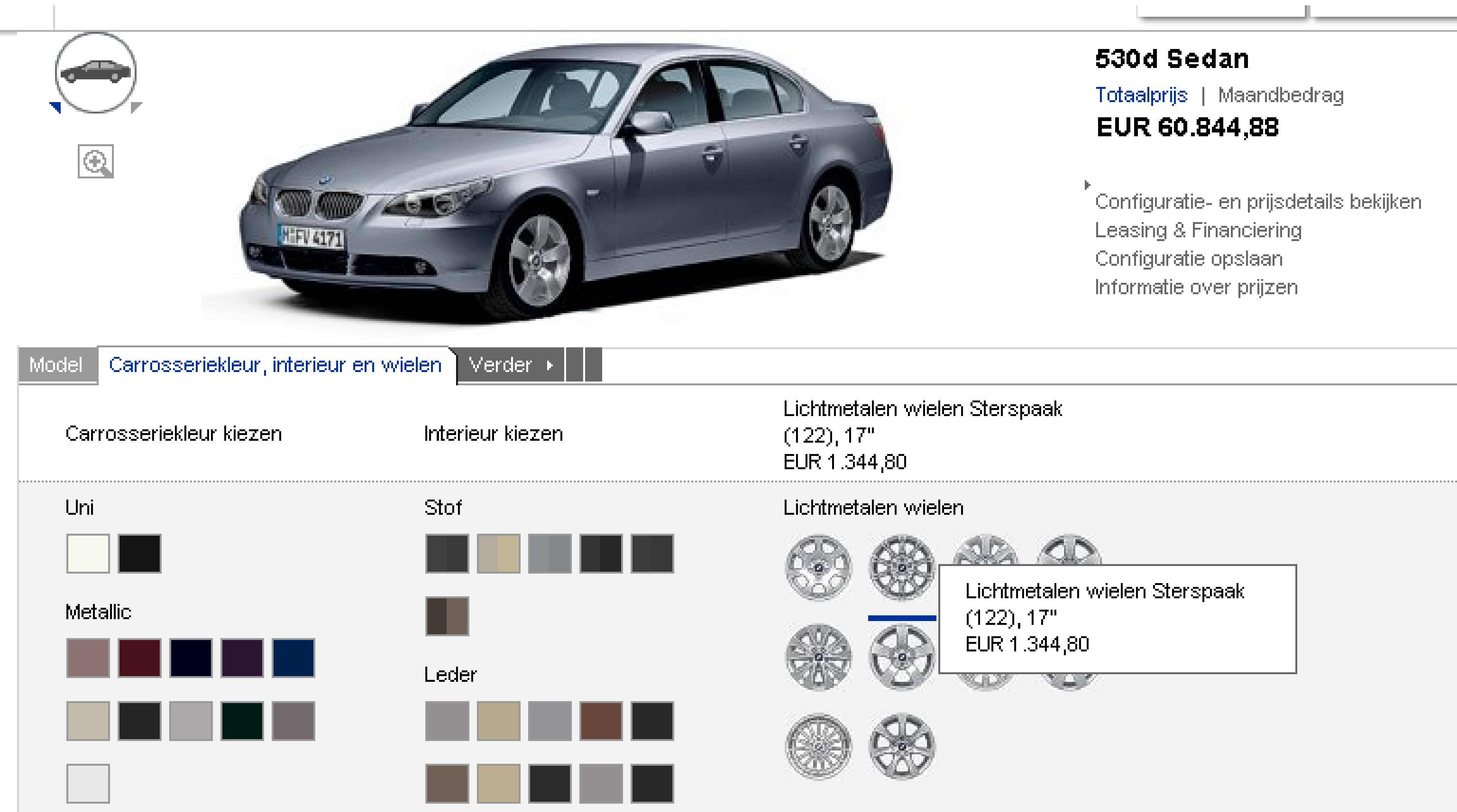
3

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

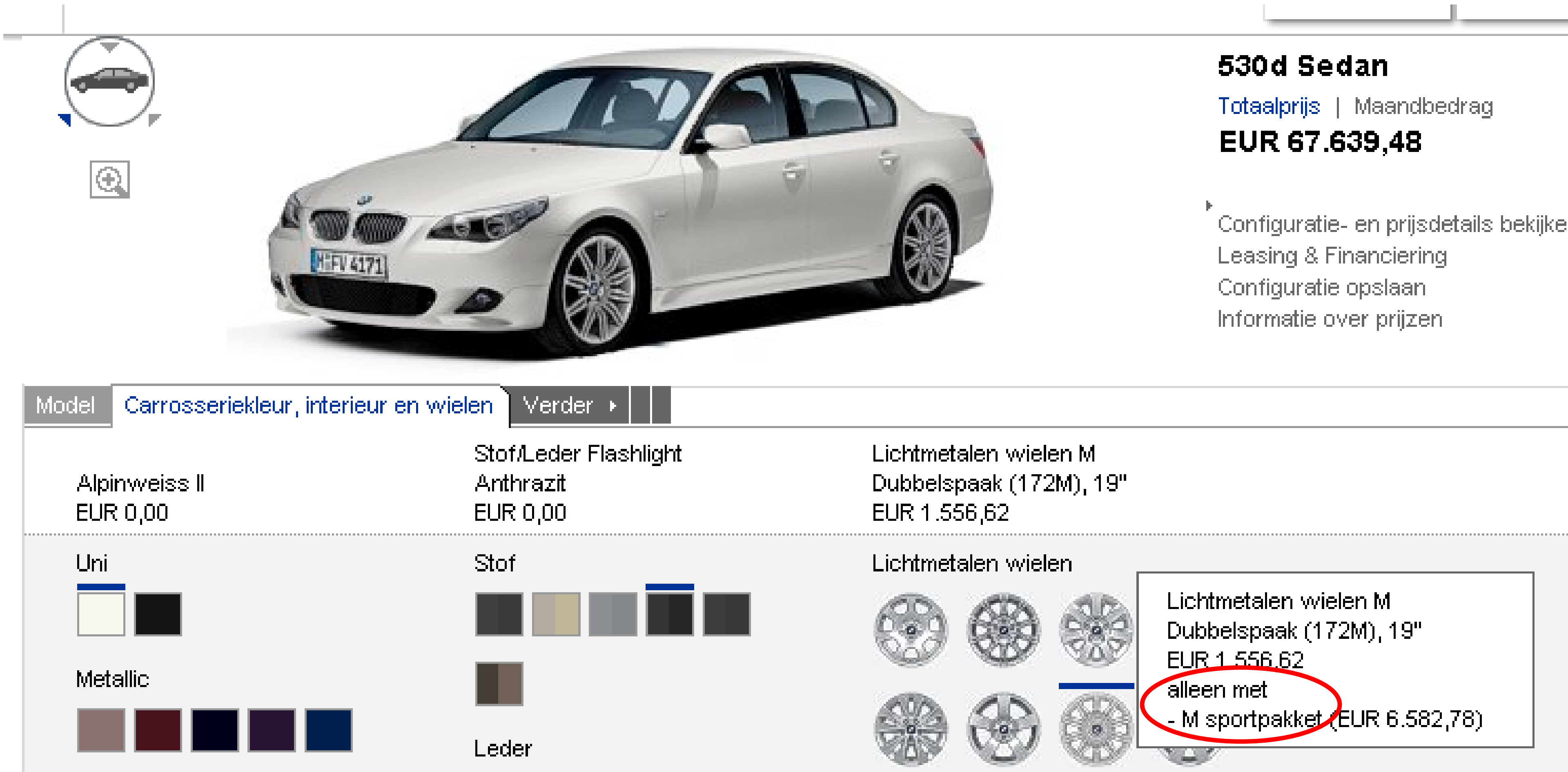
BMW 3 Series Touring (54)
from £ 24,865.00

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

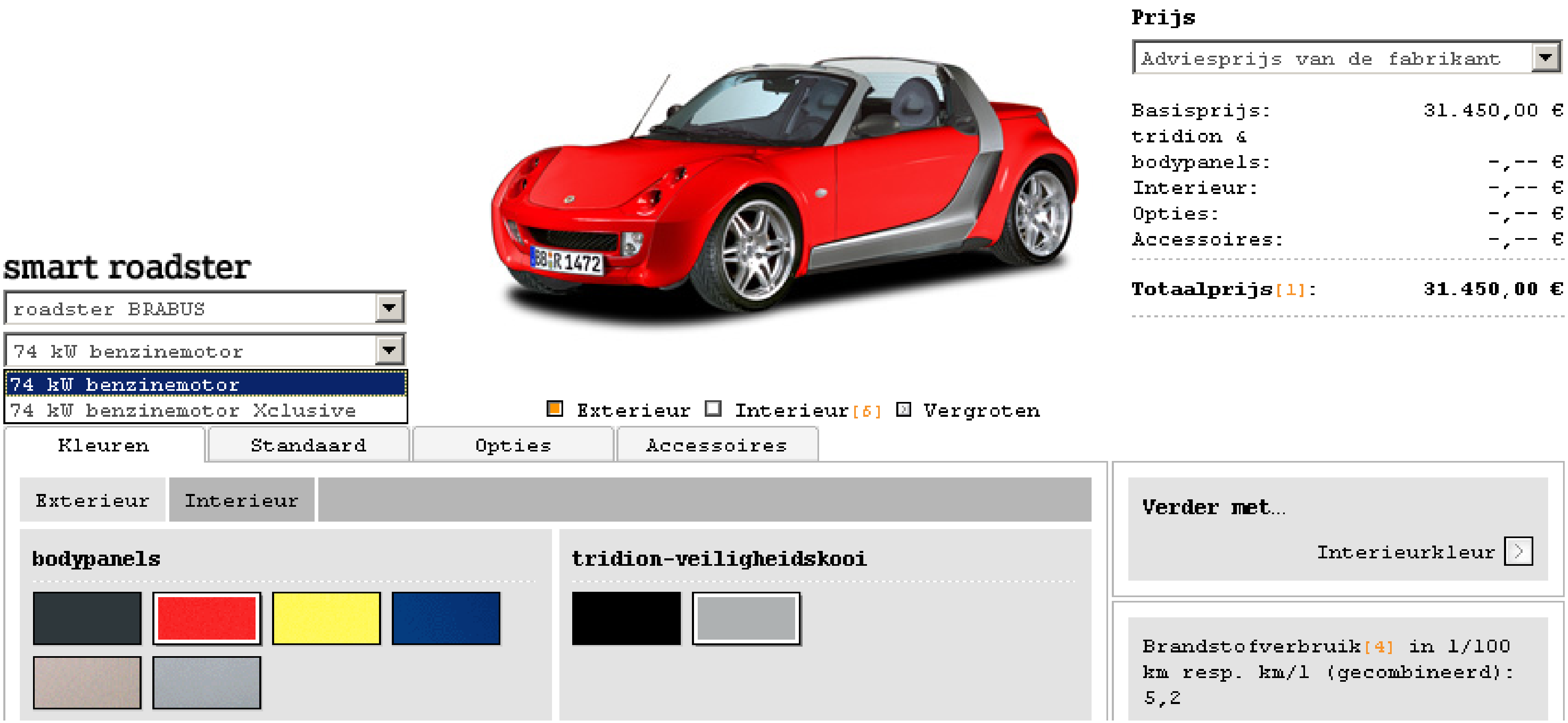
BMW product configurator



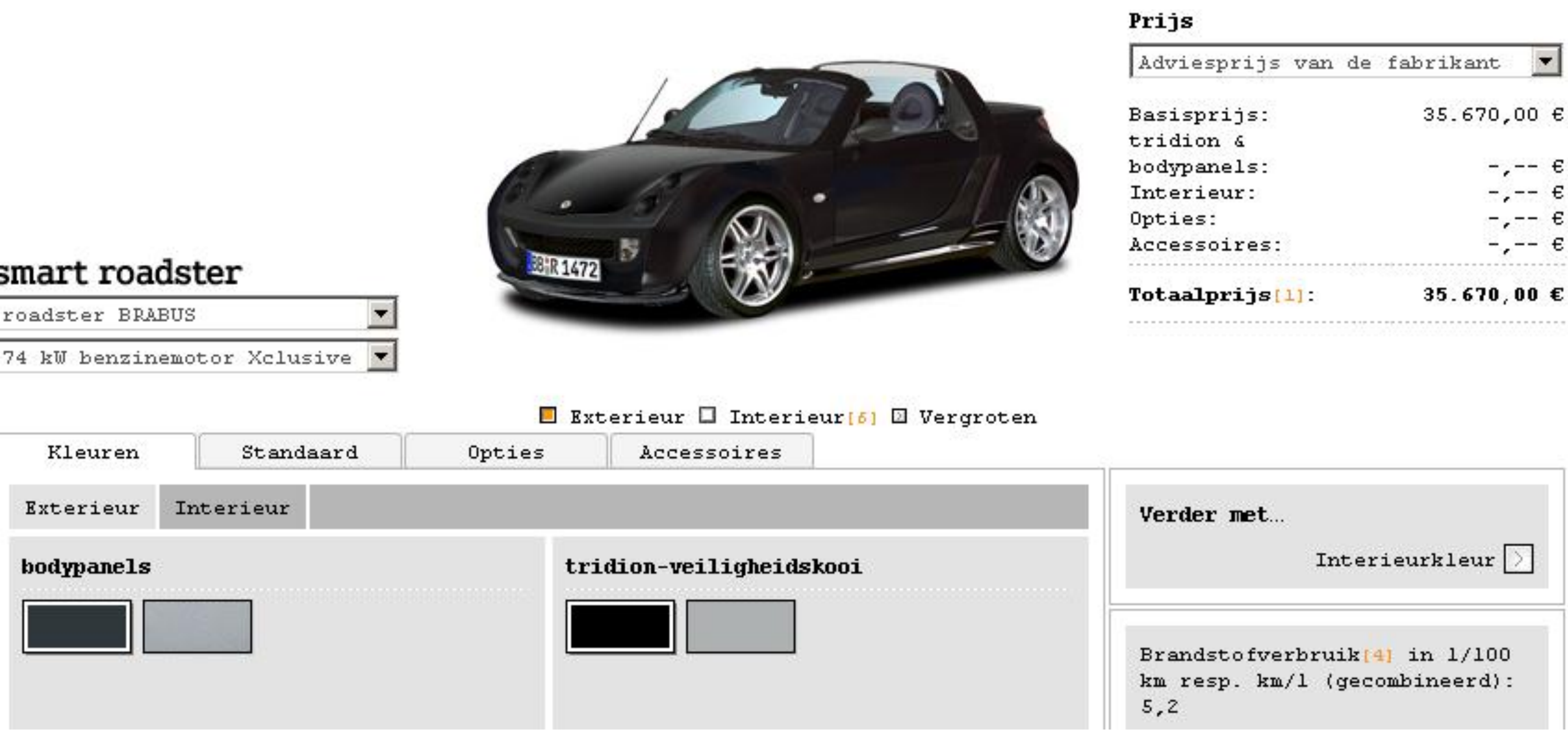
BMW feature packages



Smart constraints (1/2)



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)

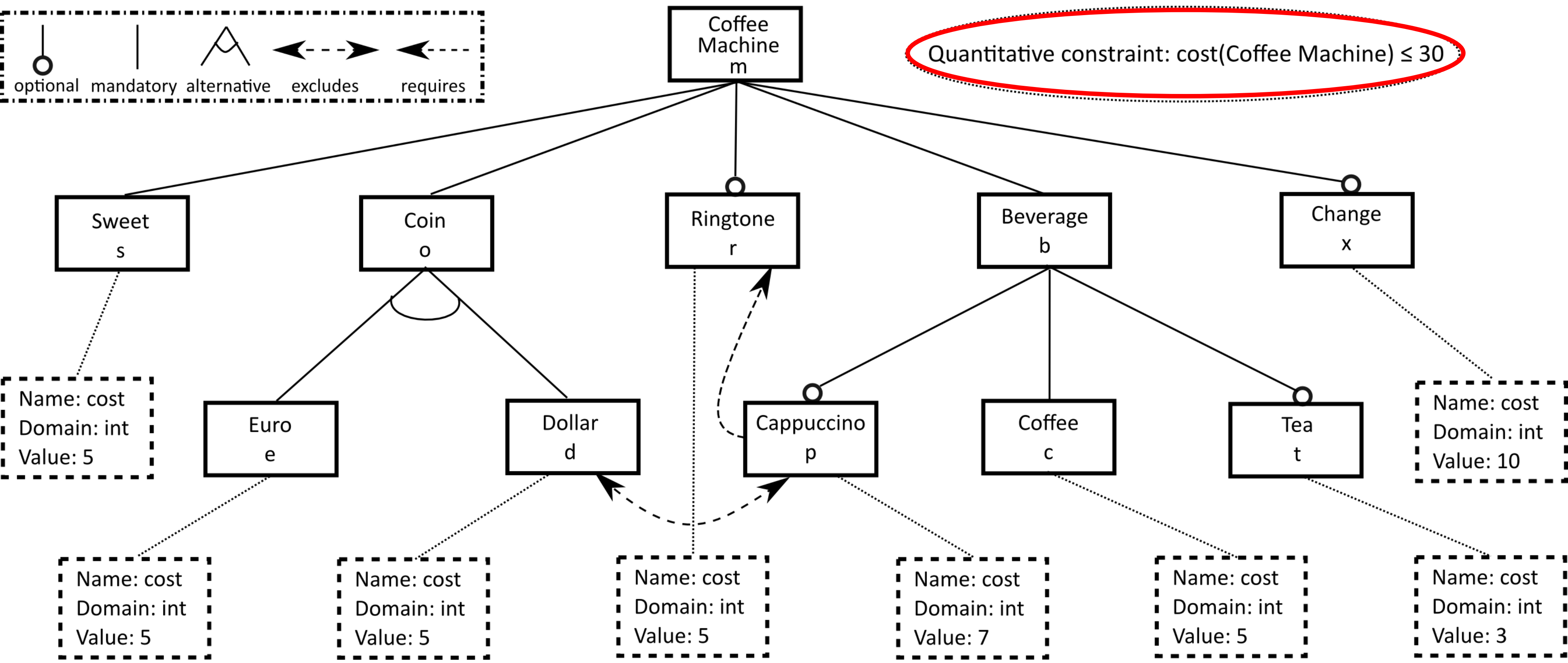
Cross-tree constraints

- 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

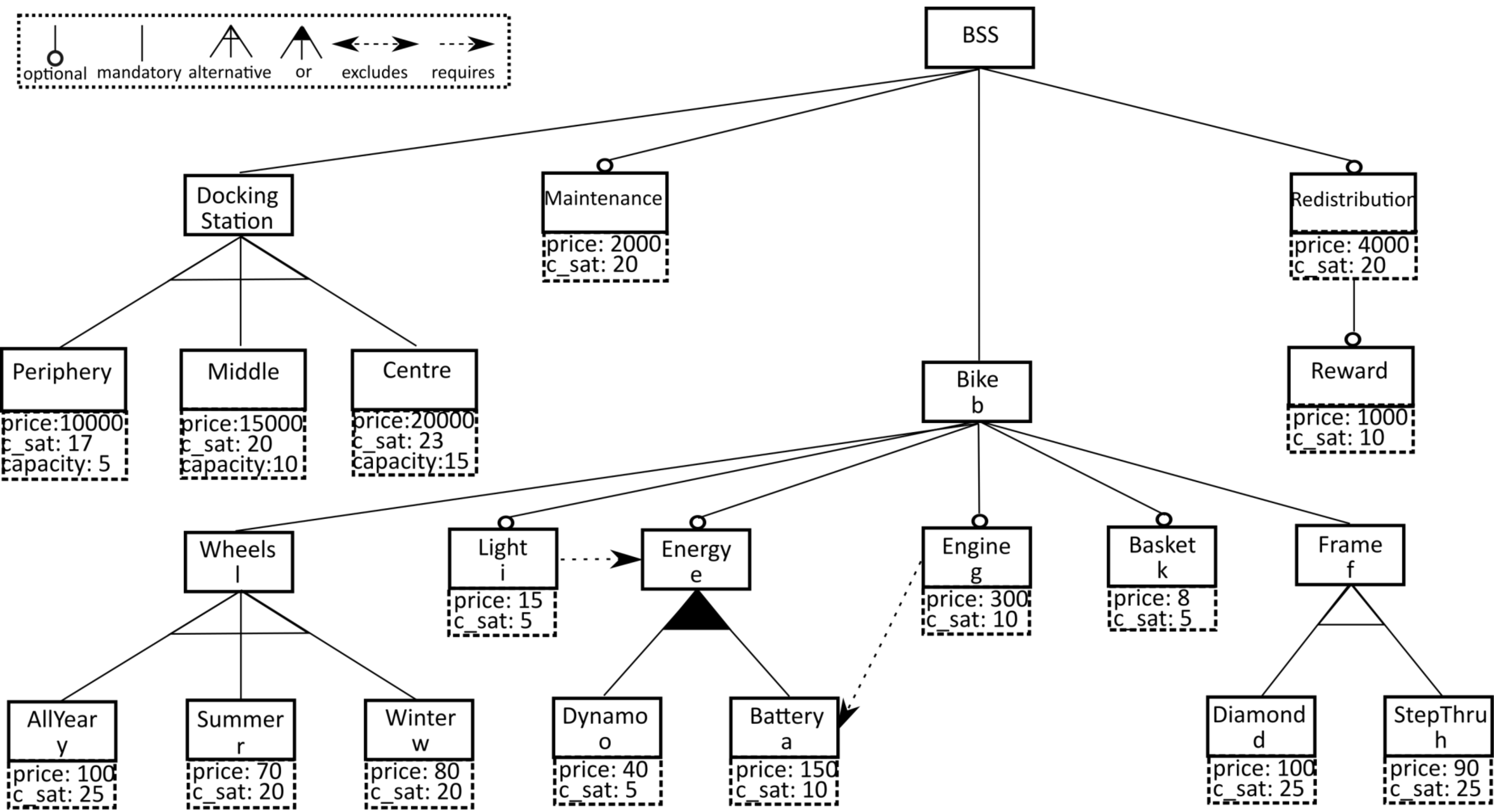
Quantitative constraints

- Feature attributes (non-functional): *cost*(feature) = 7, etc.
- *cost*(product) = ∑{*cost*(feature) | feature ∈ product }

Attributed feature model (1/2)



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

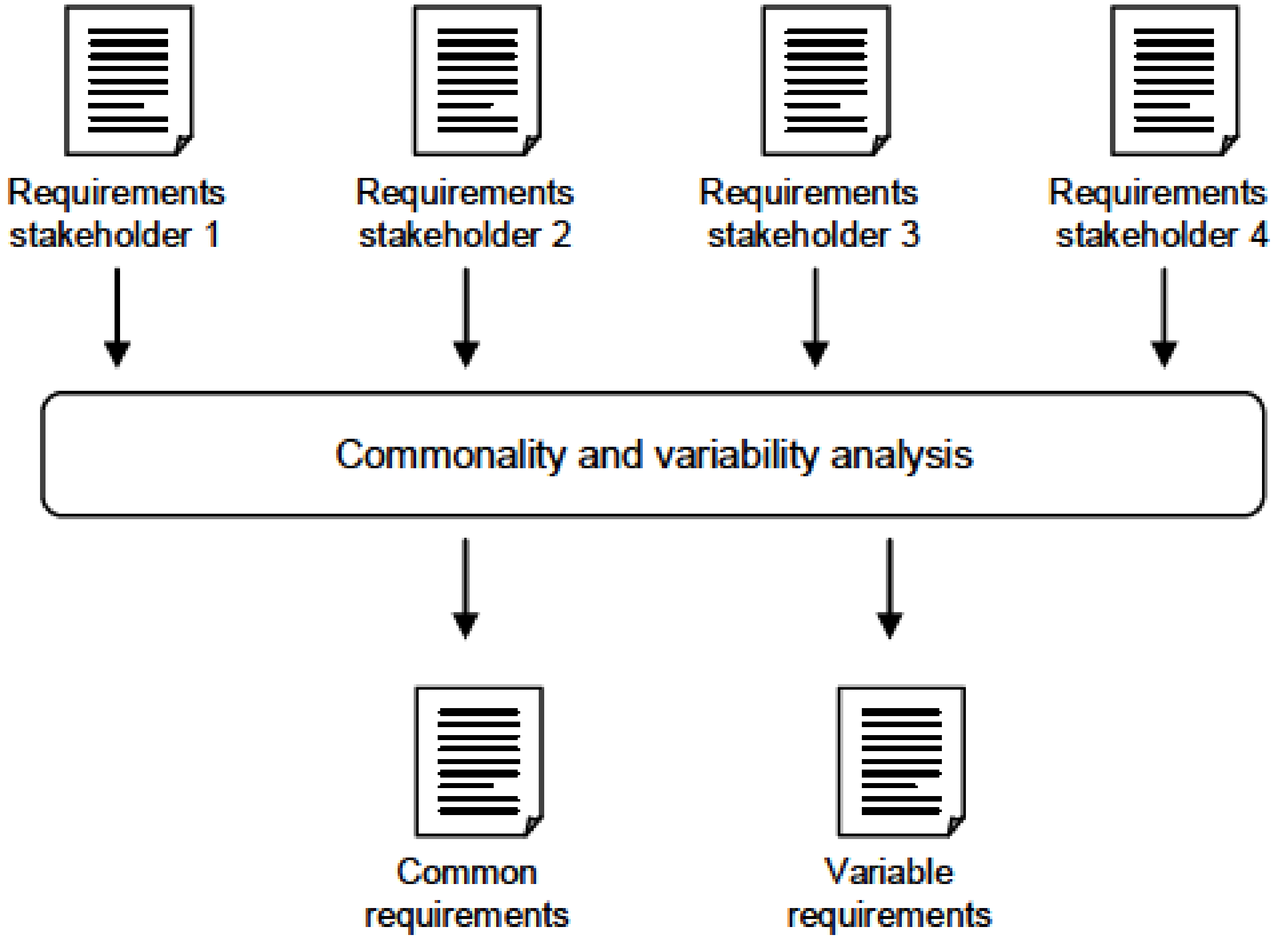
- During domain RE, the requirements for the entire SPL are defined
- Basis for: (i) developing the entire SPL family (ii) defining the requirements of each product
- Comprises the same core and cross-sectional RE activities as RE for single systems;

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



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 optional, independent features

a unique product for every

person on this planet

Examples by C. Kästner (CMU, Pittsburgh, USA)


320
optional, independent
features
more possible products than estimated
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 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)

(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

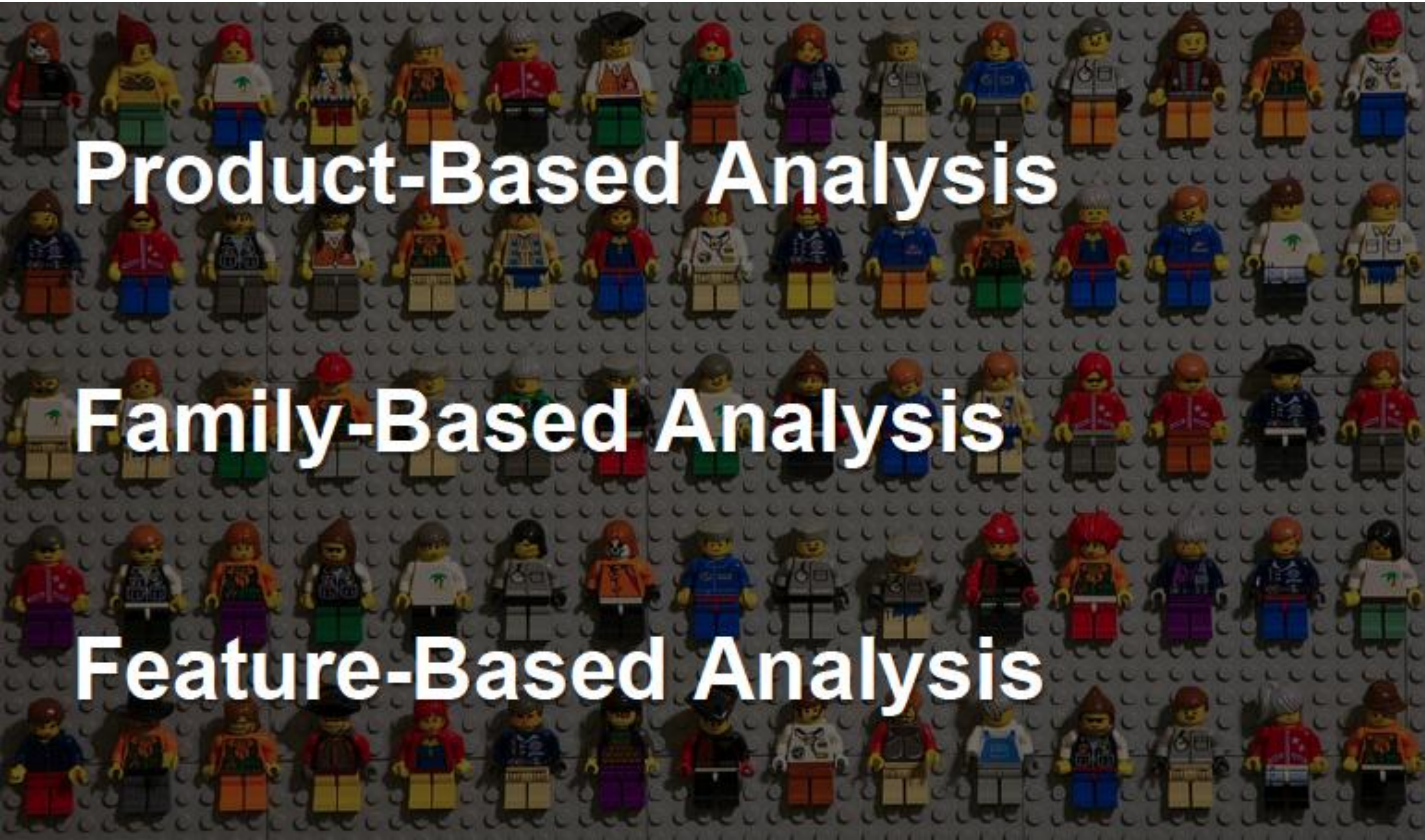
- For decades successful in single product/system engineering
- Not exploited broadly in SPLE, while correctness of artifacts for reuse and correctness of developed products is of crucial importance (many massively produced (embedded) systems and safety- or business-critical applications)

Traditionally

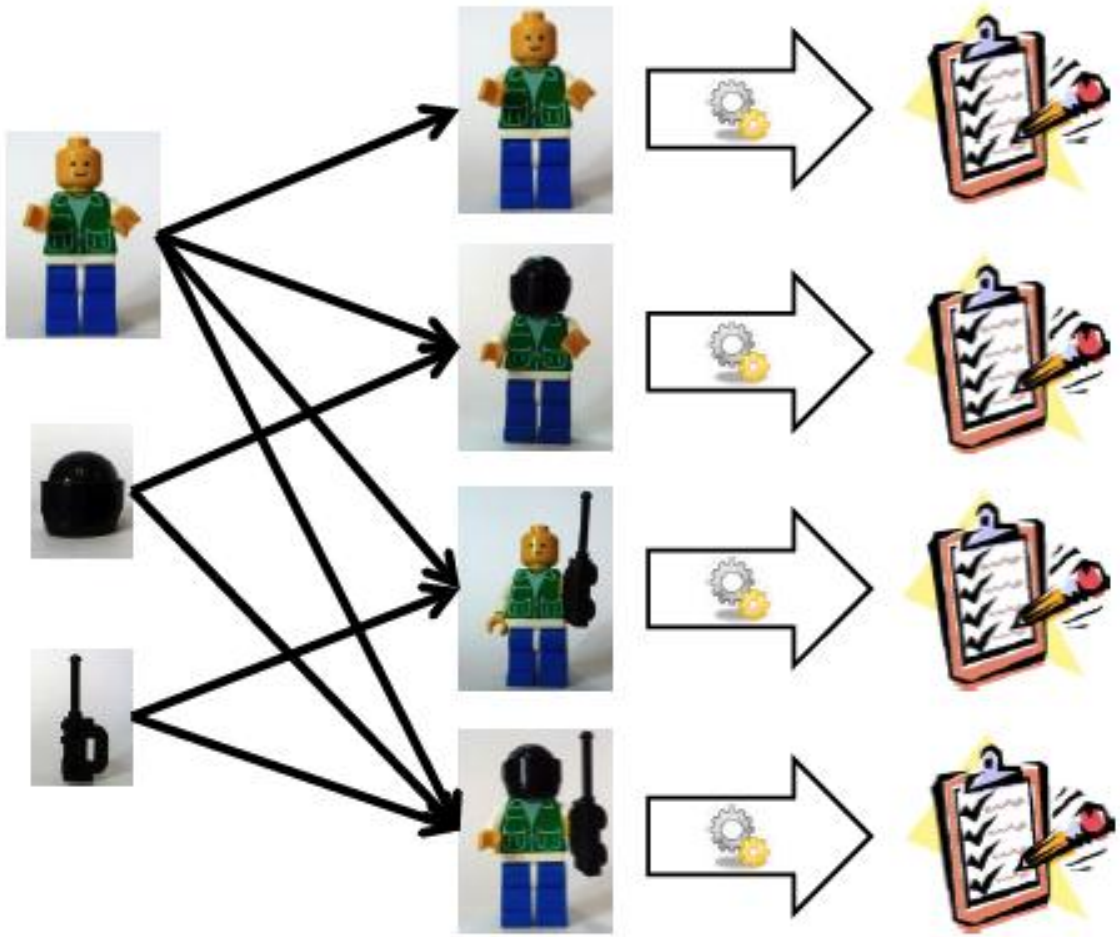
- Mainstream formal methods do *not* consider variability directly
- Formal methods that *have* been applied in SPLE mainly focus on structural rather than behavioural properties (feature model analysis: e.g. dead features, false optional features, etc.)

Variability analysis strategies

(type checking, static analysis, model checking, theorem proving, testing, etc.)

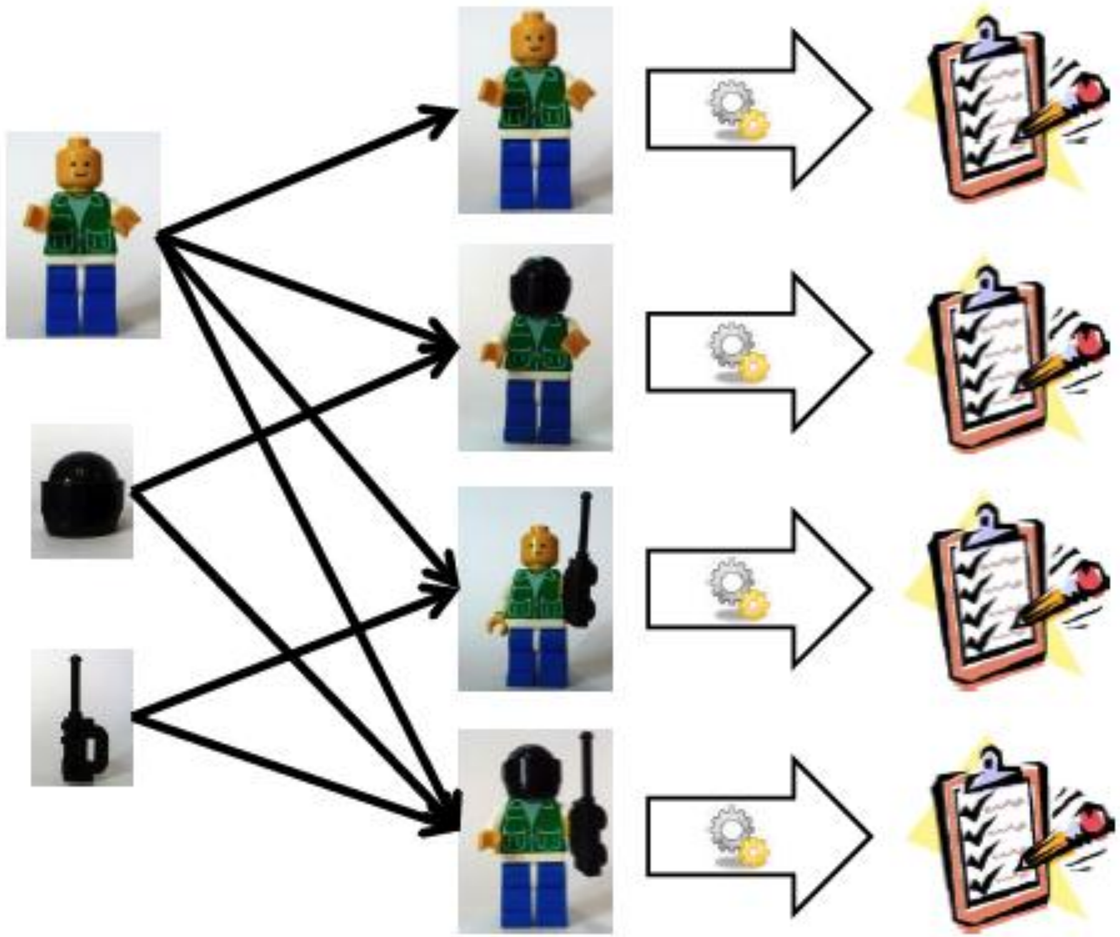


Product-based analysis



$O(2^n)$ for n features

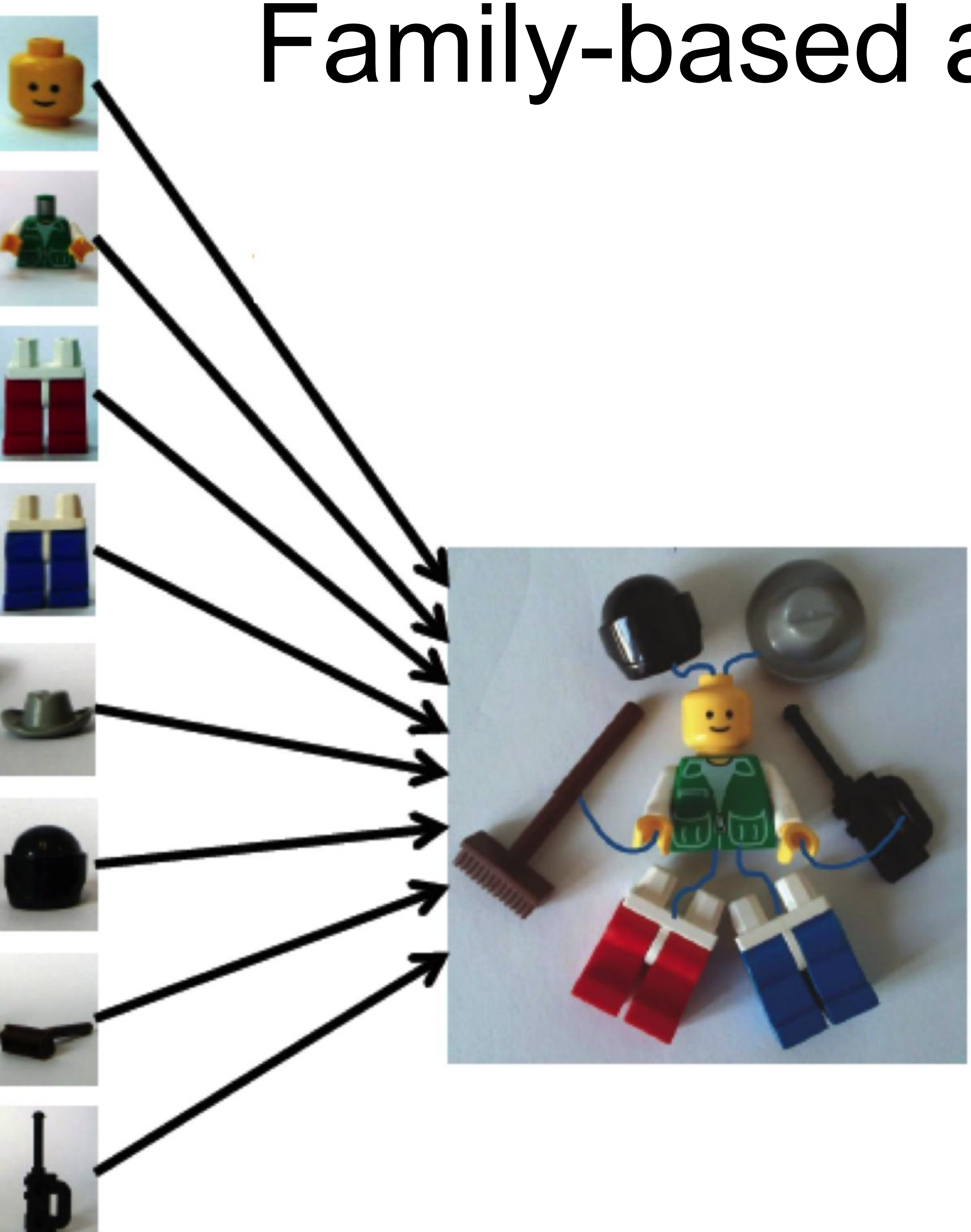
Product-based analysis

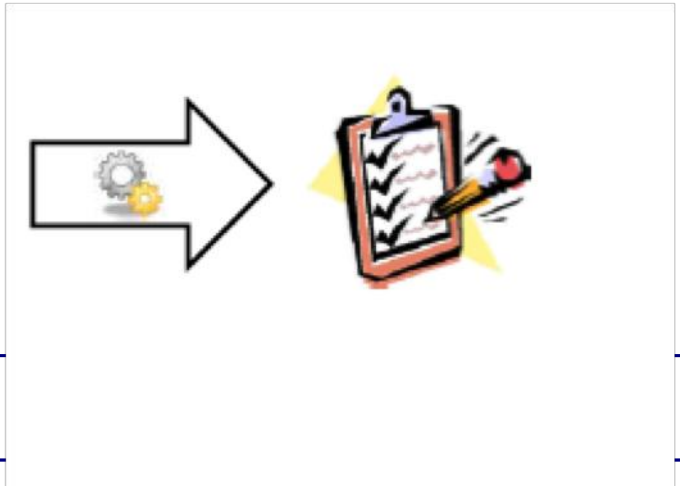


- Simple approach
- Standard tools available
- Infeasible for large product sets
- Same behaviour / code verified numerous times

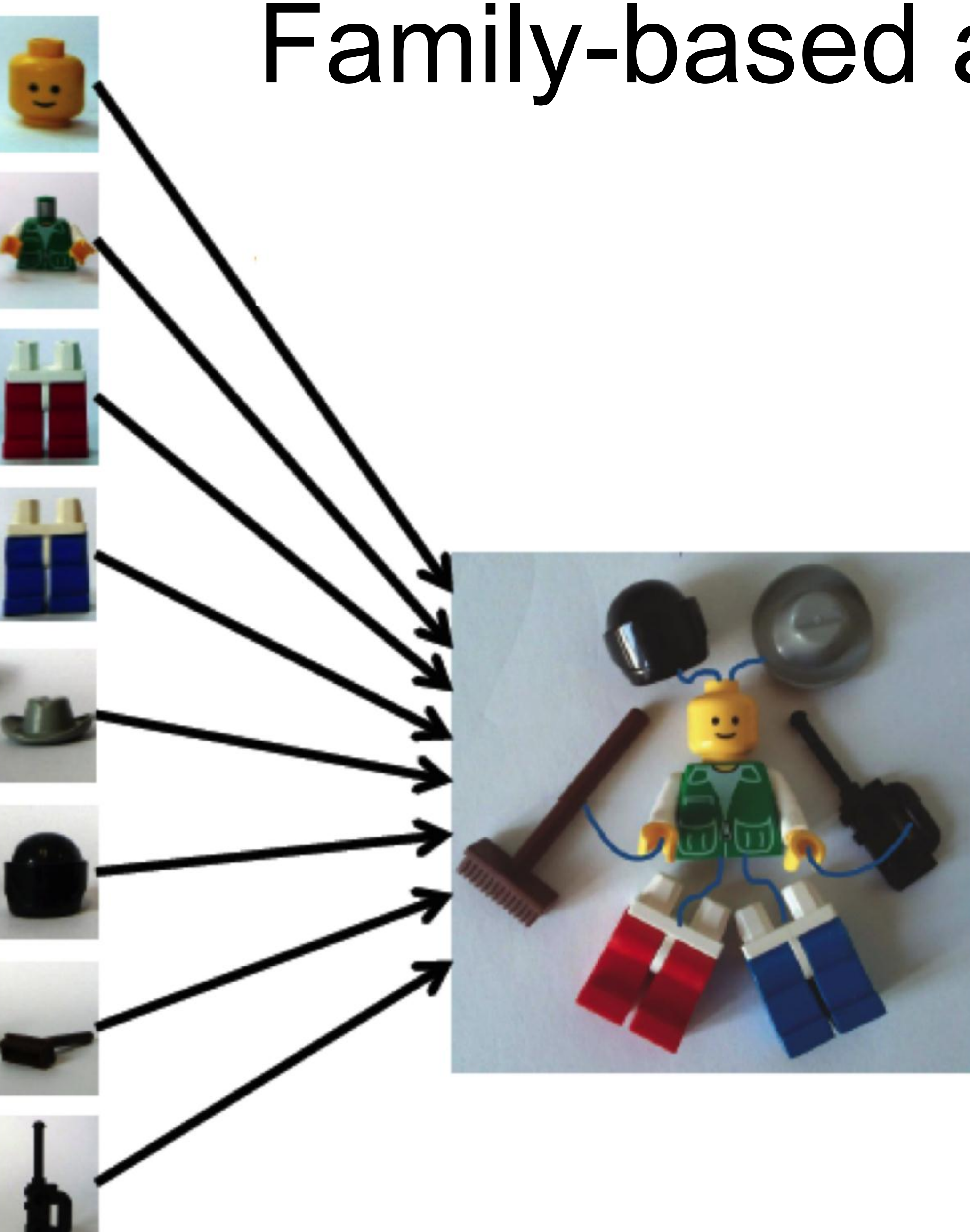
$O(2^n)$ for n features

Family-based analysis





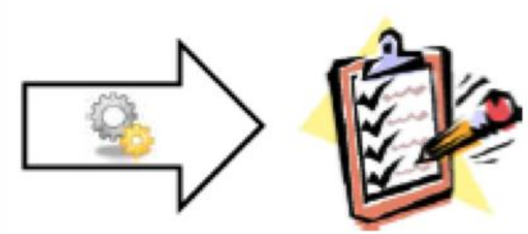
Family-based analysis



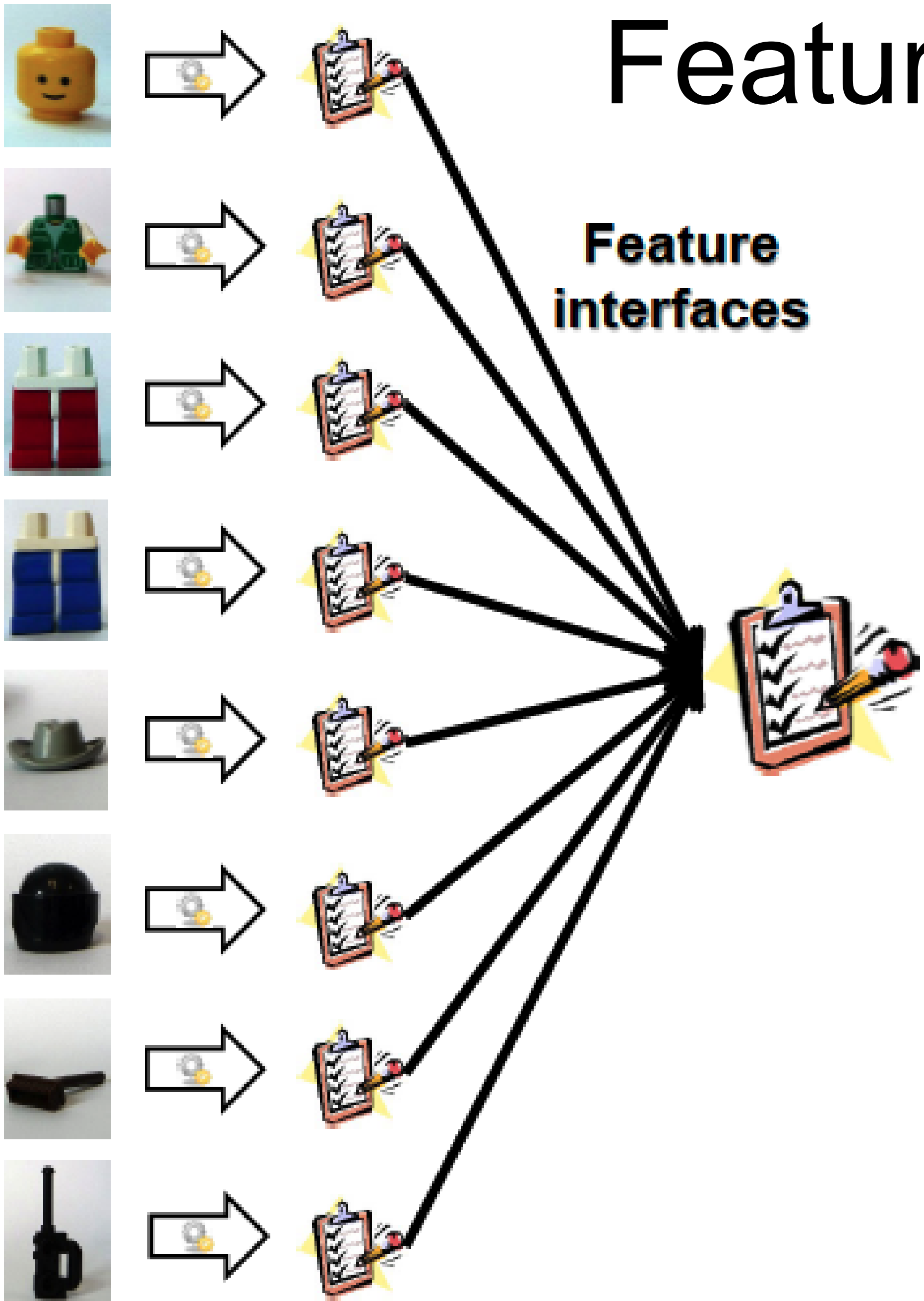
Beneficial for many products with substantial similarities

Generates complex analysis tasks

Requires (compact) family metamodels

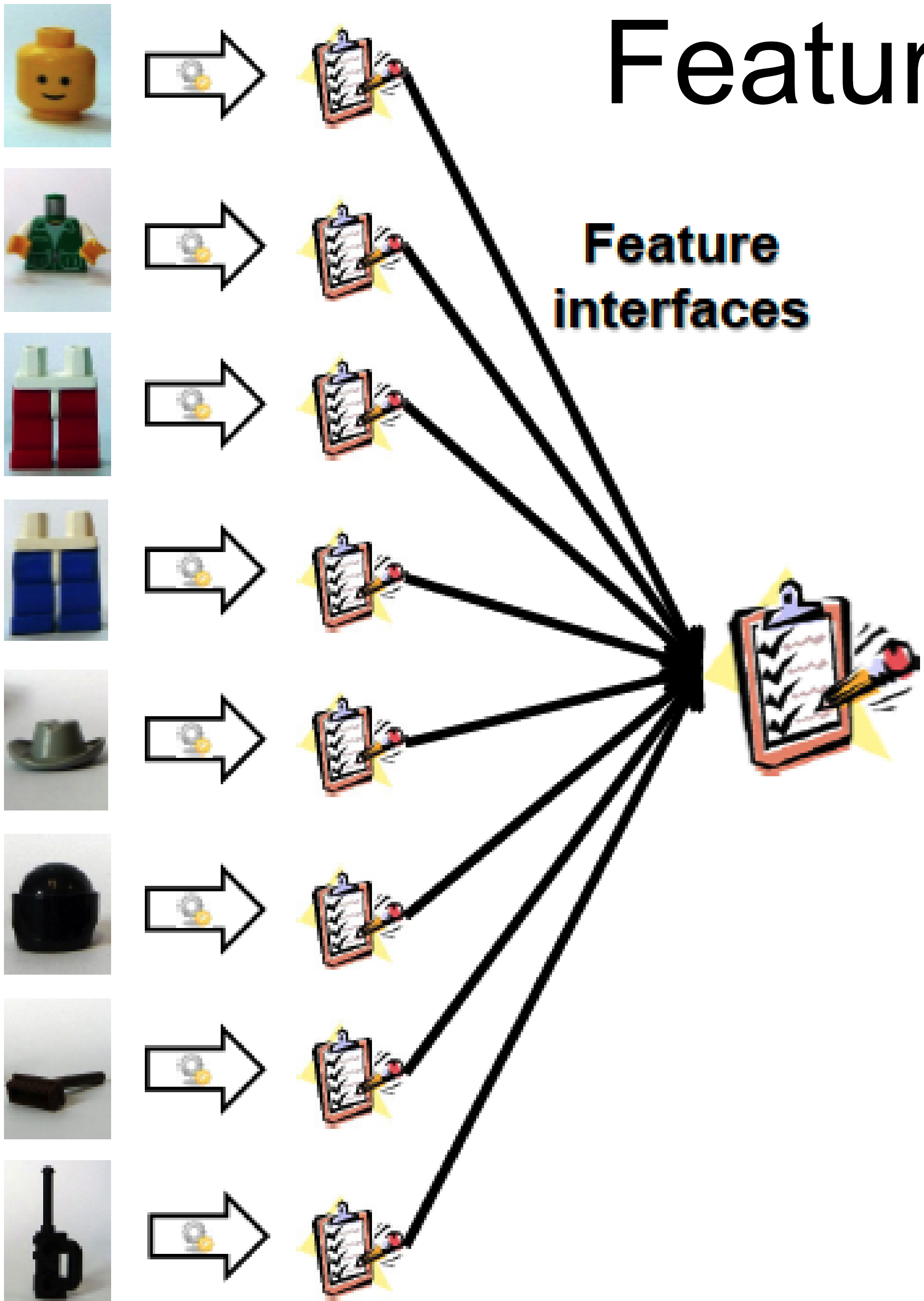


Feature-based analysis



(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

$\Rightarrow$ 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)
  Classen *et al.* @ ICSE'10, *IEEE TSE*, 2013, *Sci. Comput. Program.*, 2014
  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)

  Classen *et al.* @ ICSE'10, *IEEE TSE*, 2013, *Sci. Comput. Program.*, 2014

  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

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:

Classen *et al.* @ ICSE'10, *IEEE TSE*, 2013, *Sci. Comput. Program.*, 2014

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 for SPLE

Recall (atomic propositions used for verification purposes):

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

A Doubly-Labelled Transition System (L^2TS) 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 $L : S \rightarrow 2^{AP}$

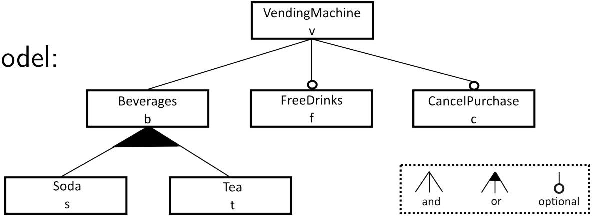
An FTS adds to this a feature model and feature expressions:

Classen *et al.* @ ICSE'10, *IEEE TSE*, 2013, *Sci. Comput. Program.*, 2014

A Featured Transition System (FTS) is an octuple $(Q, A, \overline{q}, \rightarrow, AP, L, FD, \gamma)$ with *underlying* L^2TS $(Q, A, \overline{q}, \rightarrow, AP, L)$, 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

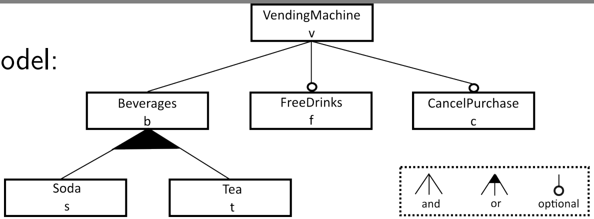
Feature model:



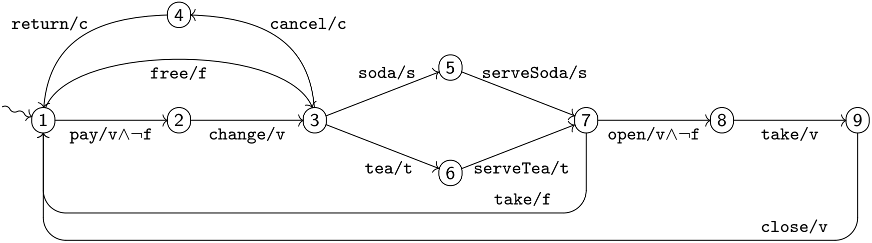
12 valid products e.g. $\{\texttt{v}, \texttt{b}, \texttt{s}, \texttt{t}\}$, $\{\texttt{v}, \texttt{b}, \texttt{s}, \texttt{c}\}$

FTS of example SPL: a vending machine

Feature model:

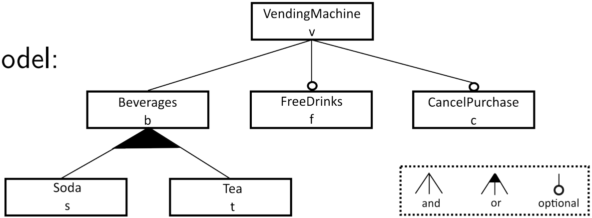


FTS of 12 valid products (LTS) e.g. $\{\mathtt{v},\mathtt{b},\mathtt{s},\mathtt{t}\}$, $\{\mathtt{v},\mathtt{b},\mathtt{s},\mathtt{c}\}$

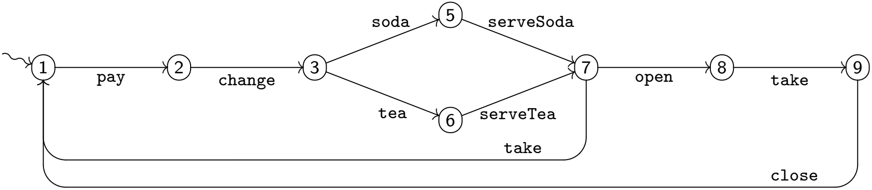


FTS of example SPL: a vending machine

Feature model:

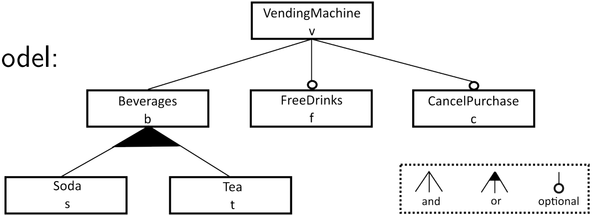


e.g. $\{\texttt{v,b,s,t}\}$, $\{\texttt{v,b,s,c}\}$

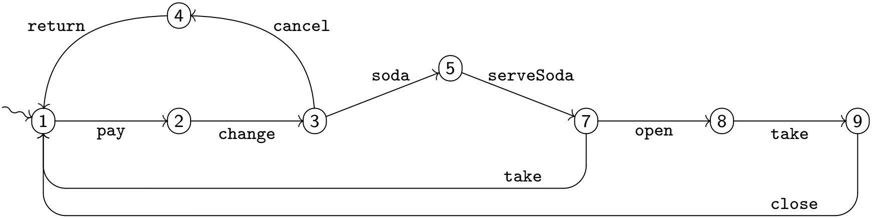


FTS of example SPL: a vending machine

Feature model:



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

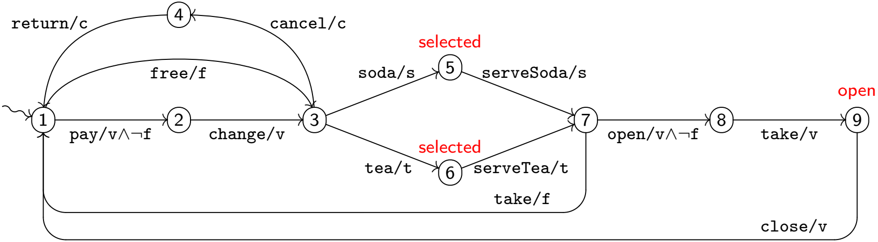


Dedicated FTS model checker SNIP (now ProVeLines)

fPROMELA:

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

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



fLTL: $[\neg f]\ \Box\,(\text{selected} \Rightarrow \Diamond\,\text{open})$

Similarly fNuSMV with fSMV/fCTL

MTSυ 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

MTSυ 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

MTSυ 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

MTS with...

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 $(Q, A, \overline{q}, \rightarrow_\Box, \rightarrow_\Diamond)$ such that $(Q, A, \overline{q}, \rightarrow_\Box \cup \rightarrow_\Diamond)$ is an LTS, called its underlying LTS

An MTS has two distinct transition relations

1. may transition relation $\rightarrow_\Diamond \subseteq Q \times A \times Q$: possible transitions
2. must transition relation $\rightarrow_\Box \subseteq Q \times A \times Q$: required transitions

By definition, any required transition is also possible: $\rightarrow_\Box \subseteq \rightarrow_\Diamond$
(denote $\dashrightarrow \equiv \rightarrow_\Diamond \setminus \rightarrow_\Box$: optional transitions)

MTS with...

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 $(Q, A, \overline{q}, \rightarrow_{\Box}, \rightarrow_{\Diamond})$ such that $(Q, A, \overline{q}, \rightarrow_{\Box} \cup \rightarrow_{\Diamond})$ is an LTS, called its underlying LTS

An MTS has two distinct transition relations

1. may transition relation $\rightarrow_{\Diamond} \subseteq Q \times A \times Q$: possible transitions
2. must transition relation $\rightarrow_{\Box} \subseteq Q \times A \times Q$: required transitions

By definition, any required transition is also possible: $\rightarrow_{\Box} \subseteq \rightarrow_{\Diamond}$ (denote $\dashrightarrow \equiv \rightarrow_{\Diamond} \setminus \rightarrow_{\Box}$: optional transitions)

...variability constraints (accepted by VMC)

Variability constraints of form ALTernative, EXCludes, REQuires, etc.

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

b_1 $\mathtt{OR} \cdots \mathtt{OR}$ 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 $\mathcal{L}$ or $b_i = \neg\, a_i$ is not reachable in $\mathcal{L}$

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

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

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

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

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

$\Rightarrow$ Each selection gives rise to a different variant

Let $(Q, A, \overline{q}, \delta^{\Diamond}, \delta^{\Box}, \Upsilon)$ be a coherent MTSυ, i.e. $\exists \overset{a}{\dashrightarrow} \implies \nexists \xrightarrow{a}$

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 $\delta_i \subseteq \delta^{\Diamond} \cup \delta^{\Box}$ to be defined s.t.

1. every $q \in Q_i$ is reachable in $\mathcal{P}_i$ from $\overline{q}$ via transitions from δ_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 $\overset{a}{\dashrightarrow} \leadsto \xrightarrow{a}$ and $\cancel{\xrightarrow{a}}$ not allowed
4. $\mathcal{P}_i$ satisfies all variability constraints in Υ

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

$\Rightarrow$ Each selection gives rise to a different variant

Let $(Q, A, \overline{q}, \delta^{\Diamond}, \delta^{\Box}, \Upsilon)$ be a coherent MTSυ, i.e. $\exists \overset{a}{\dashrightarrow} \implies \nexists \xrightarrow{a}$

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 $\delta_i \subseteq \delta^{\Diamond} \cup \delta^{\Box}$ to be defined s.t.

1. every $q \in Q_i$ is reachable in $\mathcal{P}_i$ from $\overline{q}$ via transitions from δ_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 $\overset{a}{\dashrightarrow} \;\rightsquigarrow\; \xrightarrow{a}$ and $\cancel{\overset{a}{\dashrightarrow}}$ not allowed
4. $\mathcal{P}_i$ satisfies all variability constraints in Υ

MTSυ equally expressive as FTS

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

Theorem (FTS2MTSυ transformation is sound and complete)

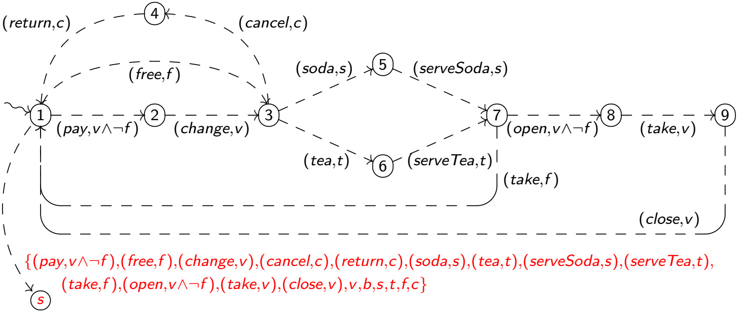
Let $\mathcal{F}$ be an FTS and let $\mathcal{M}$ be the MTSυ generated from $\mathcal{F}$ according to the FTS2MTSυ model transformation algorithm. Then the sets of derived variants $\mathsf{lts}(\mathcal{F})$ *and* $\mathsf{lts}(\mathcal{M})$ *coincide, up to dummy transitions and action relabelling.*

MTSυ equally expressive as FTS

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

Theorem (FTS2MTSυ transformation is sound and complete)

Let $\mathcal{F}$ be an FTS and let $\mathcal{M}$ be the MTSυ generated from $\mathcal{F}$ according to the FTS2MTSυ model transformation algorithm. Then the sets of derived variants $\mathrm{lts}(\mathcal{F})$ *and* $\mathrm{lts}(\mathcal{M})$ *coincide, up to dummy transitions and action relabelling.*



$\Upsilon = \{(\textit{pay}, v\wedge\neg f) \Leftrightarrow (v\wedge\neg f),\ (\textit{free}, f) \Leftrightarrow f,\ (\textit{change}, v) \Leftrightarrow v,\ (\textit{cancel}, c) \Leftrightarrow c,\ (\textit{return}, c) \Leftrightarrow c,\ (\textit{soda}, s) \Leftrightarrow s,\ (\textit{tea}, t) \Leftrightarrow t,$
$(\textit{serveSoda}, s) \Leftrightarrow s,\ (\textit{serveTea}, t) \Leftrightarrow t,\ (\textit{take}, f) \Leftrightarrow f,\ (\textit{open}, v\wedge\neg f) \Leftrightarrow (v\wedge\neg f),\ (\textit{take}, v) \Leftrightarrow v,\ (\textit{close}, v) \Leftrightarrow v\}$

MTSυ equally expressive as FTS

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

Theorem (FTS2MTSυ transformation is sound and complete)

Let $\mathcal{F}$ be an FTS and let $\mathcal{M}$ be the MTSυ generated from $\mathcal{F}$ according to the FTS2MTSυ model transformation algorithm. Then the sets of derived variants $\mathrm{lts}(\mathcal{F})$ and $\mathrm{lts}(\mathcal{M})$ coincide, up to dummy transitions and action relabelling.

Theorem (MTSυ2FTS transformation is sound and complete)

Let $\mathcal{M}$ be an MTSυ and let $\mathcal{F}$ be the FTS generated from $\mathcal{M}$ according to the MTSυ2FTS model transformation algorithm. Then $\mathrm{lts}(\mathcal{M}) = \mathrm{lts}(\mathcal{F})$.

Dedicated MTSυ 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

Dedicated MTSυ 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)

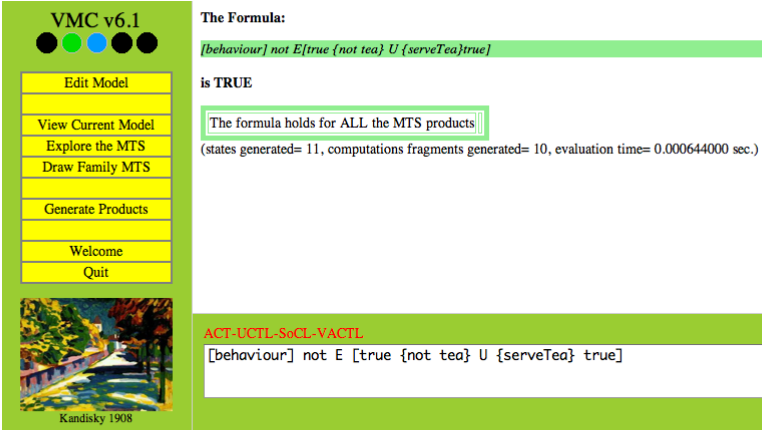
   ter Beek *et al.* @ *J. Logic Algebr. Meth. Program.*, 2016

VMC v6.4 is freely usable online:

`http://fmt.isti.cnr.it/vmc/`

Vending Machine: family-based analysis

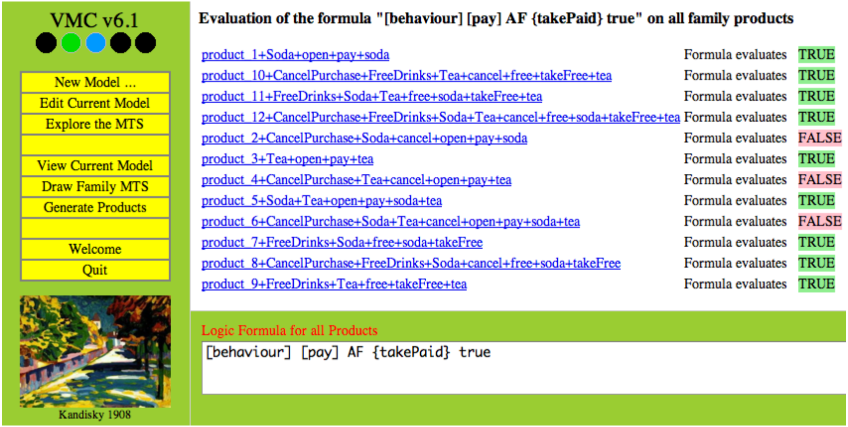
VMC notifies whenever preservation of an analysis result is applicable



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 χ), state formulae (ϕ) and path formulae (π)

$$
\begin{aligned}
\phi ::= {} & \mathit{true} \mid \neg\,\phi \mid \phi \wedge \phi \mid [\chi]\,\phi \mid \langle\chi\rangle\,\phi \mid E\,\pi \mid A\,\pi \mid \\
& \mu\,Y.\phi(Y) \mid \nu\,Y.\phi(Y) \\
\pi ::= {} & X\,\{\chi\}\,\phi \mid [\phi\,\{\chi\}\,U\,\{\chi'\}\,\phi'] \mid [\phi\,\{\chi\}\,W\,\{\chi'\}\,\phi'] \mid \\
& [\phi\,\{\chi\}\,U\,\phi'] \mid [\phi\,\{\chi\}\,W\,\phi'] \mid F\,\phi \mid F\,\{\chi\}\,\phi \mid G\,\phi \mid G\,\{\chi\}\,\phi
\end{aligned}
$$

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

μ and ν: 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 χ), state formulae (ϕ) and path formulae (π)

$$
\begin{aligned}
\phi \;::=\;& \mathit{true} \mid \neg\,\phi \mid \phi \wedge \phi \mid [\chi]\,\phi \mid \langle\chi\rangle\,\phi \mid E\,\pi \mid A\,\pi \;\mid \\
& \mu\,Y.\phi(Y) \mid \nu\,Y.\phi(Y) \\
\pi \;::=\;& X\,\{\chi\}\;\phi \mid [\phi\;\{\chi\}\,U\,\{\chi'\}\;\phi'] \mid [\phi\;\{\chi\}\,W\,\{\chi'\}\;\phi'] \;\mid \\
& [\phi\;\{\chi\}\,U\;\phi'] \mid [\phi\;\{\chi\}\,W\;\phi'] \mid F\,\phi \mid F\,\{\chi\}\;\phi \mid G\,\phi \mid G\,\{\chi\}\;\phi
\end{aligned}
$$

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

μ and ν: 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 χ), state formulae (ϕ) and path formulae (π)

$$
\begin{aligned}
\phi & ::= \mathit{true} \mid \neg\,\phi \mid \phi \wedge \phi \mid [\chi]\,\phi \mid \langle\chi\rangle\,\phi \mid E\,\pi \mid A\,\pi \mid \\
& \quad\;\; \mu\,Y.\phi(Y) \mid \nu\,Y.\phi(Y) \\
\pi & ::= X\,\{\chi\}\;\phi \mid [\phi\;\{\chi\}\,U\,\{\chi'\}\;\phi'] \mid [\phi\;\{\chi\}\,W\,\{\chi'\}\;\phi'] \mid \\
& \quad\;\; [\phi\;\{\chi\}\,U\;\phi'] \mid [\phi\;\{\chi\}\,W\;\phi'] \mid F\,\phi \mid F\,\{\chi\}\;\phi \mid G\,\phi \mid G\,\{\chi\}\;\phi
\end{aligned}
$$

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

μ and ν: 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”)

v-ACTL: variability-aware, action-based CTL

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

$\langle\chi\rangle^{\square}\,\phi$	a next state exists, reachable by a must transition, executing an action satisfying χ, in which ϕ holds
$[\chi]^{\square}\,\phi$	in all next states reachable by a must transition, executing an action satisfying χ, ϕ holds
$X^{\square}\,\phi$ $(X^{\square}\,\{\chi\}\,\phi)$	in the next state in the path, reached by a must transition (and executing an action satisfying χ), ϕ holds
$F^{\square}\,\phi$ $(F^{\square}\,\{\chi\}\,\phi)$	there exists a future state in the path in which ϕ holds (reached by executing an action satisfying χ) and all transitions until that state are must transitions
$G^{\square}\,\phi \equiv \neg F^{\square}\,\neg\phi$	in all states in the path, ϕ holds, and all transitions 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 χ, in which ϕ holds

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

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

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

$G^{\Box}\,\phi \equiv \neg F^{\Box}\,\neg\phi$ in all states in the path, ϕ holds, and all transitions 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 χ, in which ϕ holds
$[\chi]^{\Box}\,\phi$	in all next states reachable by a must transition, executing an action satisfying χ, ϕ holds
$X^{\Box}\,\phi$ $(X^{\Box}\,\{\chi\}\ \phi)$	in the next state in the path, reached by a must transition (and executing an action satisfying χ), ϕ holds
$F^{\Box}\,\phi$ $(F^{\Box}\,\{\chi\}\ \phi)$	there exists a future state in the path in which ϕ holds (reached by executing an action satisfying χ) and all transitions until that state are must transitions
$G^{\Box}\,\phi \equiv \neg F^{\Box}\,\neg\phi$	in all states in the path, ϕ holds, and all transitions 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 χ, in which ϕ holds
$[\chi]^{\Box}\,\phi$	in all next states reachable by a must transition, executing an action satisfying χ, ϕ holds
$X^{\Box}\,\phi$ $(X^{\Box}\,\{\chi\}\;\phi)$	in the next state in the path, reached by a must transition (and executing an action satisfying χ), ϕ holds
$F^{\Box}\,\phi$ $(F^{\Box}\,\{\chi\}\;\phi)$	there exists a future state in the path in which ϕ holds (reached by executing an action satisfying χ) and all transitions until that state are must transitions
$G^{\Box}\,\phi \equiv$ $\neg F^{\Box}\,\neg\phi$	in all states in the path, ϕ holds, and all transitions 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 χ, in which ϕ holds
$[\chi]^{\Box}\,\phi$	in all next states reachable by a must transition, executing an action satisfying χ, ϕ holds
$X^{\Box}\,\phi$ ($X^{\Box}\,\{\chi\}\,\phi$)	in the next state in the path, reached by a must transition (and executing an action satisfying χ), ϕ holds
$F^{\Box}\,\phi$ ($F^{\Box}\,\{\chi\}\,\phi$)	there exists a future state in the path in which ϕ holds (reached by executing an action satisfying χ) and all transitions until that state are must transitions
$G^{\Box}\,\phi \equiv \neg F^{\Box}\,\neg\phi$	in all states in the path, ϕ holds, and all transitions are must transitions

Preservation of formulae in v-ACTL$^\Box$/v-ACTLive$^\Box$

v-ACTL$^\Box$/v-ACTLive$^\Box$:

$$\begin{aligned}\phi \ ::= \ & \mathit{false} \mid \mathit{true} \mid \phi \wedge \phi \mid \phi \vee \phi \mid [\chi]\,\phi \mid \langle\chi\rangle^\Box\,\phi \mid \\ & \mathit{EX}^\Box\,\phi \mid \mathit{EX}^\Box\,\{\chi\}\,\phi \mid \mathit{EF}^\Box\,\phi \mid \mathit{EF}^\Box\,\{\chi\}\,\phi \mid \\ & \mathit{AF}^\Box\,\phi \mid \mathit{AF}^\Box\,\{\chi\}\,\phi \mid \mathit{AG}\,\phi \mid \mathit{AF}\,\phi \mid \mathit{AF}\,\{\chi\}\,\phi\end{aligned}$$

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

v-ACTL$^-$:

$$\begin{aligned}\psi \ ::= \ & \mathit{false} \mid \mathit{true} \mid \psi \wedge \psi \mid \psi \vee \psi \mid [\chi]^\Box\,\psi \mid \langle\chi\rangle\,\psi \mid \\ & \mathit{EX}\,\phi \mid \mathit{EX}\,\{\chi\}\,\phi \mid \mathit{EF}\,\psi \mid \mathit{EF}\{\chi\}\,\psi \mid \mathit{AF}\,\phi \mid \mathit{AF}\,\{\chi\}\,\phi\end{aligned}$$

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

Preservation of formulae in v-ACTL$^\Box$/v-ACTLive$^\Box$

v-ACTL$^\Box$/v-ACTLive$^\Box$:

$$\begin{aligned}\phi ::= \ & \mathit{false} \mid \mathit{true} \mid \phi \wedge \phi \mid \phi \vee \phi \mid [\chi]\,\phi \mid \langle\chi\rangle^\Box\,\phi \mid \\ & \mathit{EX}^\Box\,\phi \mid \mathit{EX}^\Box\,\{\chi\}\,\phi \mid \mathit{EF}^\Box\,\phi \mid \mathit{EF}^\Box\,\{\chi\}\,\phi \mid \\ & \mathit{AF}^\Box\,\phi \mid \mathit{AF}^\Box\,\{\chi\}\,\phi \mid \mathit{AG}\,\phi \mid \mathit{AF}\,\phi \mid \mathit{AF}\,\{\chi\}\,\phi\end{aligned}$$

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

v-ACTL$^-$:

$$\begin{aligned}\psi ::= \ & \mathit{false} \mid \mathit{true} \mid \psi \wedge \psi \mid \psi \vee \psi \mid [\chi]^\Box\,\psi \mid \langle\chi\rangle\,\psi \mid \\ & \mathit{EX}\,\phi \mid \mathit{EX}\,\{\chi\}\,\phi \mid \mathit{EF}\,\psi \mid \mathit{EF}\{\chi\}\,\psi \mid \mathit{AF}\,\phi \mid \mathit{AF}\,\{\chi\}\,\phi\end{aligned}$$

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

Live states use SPL-specific information

$$S \models \phi \;\Rightarrow\; S_p \models \phi \;\; \forall \text{ product LTS } S_p \text{ of MTS } S$$

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

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

MTS

Assume

a OR b

LTS

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

$$S \models \phi \;\Rightarrow\; S_p \models \phi \;\; \forall \text{ product LTS } S_p \text{ of MTS } S$$

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

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

MTS

Assume

a OR b

LTS

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 $\{a, b\}$

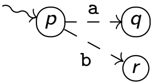
Live states use SPL-specific information

$$S \models \phi \;\Rightarrow\; S_p \models \phi \quad \forall \text{ product LTS } S_p \text{ of MTS } S$$

Recall: all (reachable) must transitions are preserved ($\langle\,\rangle^{\Box}, [\,]^{\Box}, X^{\Box}, F^{\Box}$)

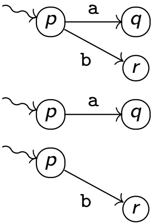
Live action sets define live states (not occur as final in any product)

MTS



Assume

a OR b



LTS

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 $\{a, b\}$

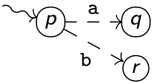
Live states use SPL-specific information

$$S \models \phi \;\Rightarrow\; S_p \models \phi \;\; \forall \text{ product LTS } S_p \text{ of MTS } S$$

Recall: all (reachable) must transitions are preserved ($\langle\,\rangle^{\Box}, [\,]^{\Box}, X^{\Box}, F^{\Box}$)

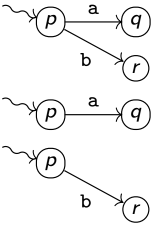
Live action sets define live states (not occur as final in any product)

MTS



Assume

a OR b



LTS

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

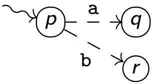
$\Rightarrow$ p is a live state, since a OR b gives rise to a live action set $\{a, b\}$

Live states use SPL-specific information

$$S \models \phi \;\Rightarrow\; S_p \models \phi \;\;\forall \text{ product LTS } S_p \text{ of MTS } S$$

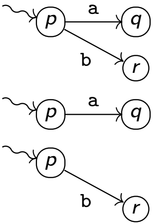
Recall: all (reachable) must transitions are preserved ($\langle\,\rangle^\Box, [\,]^\Box, X^\Box, F^\Box$)

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

MTS 

Assume

a OR b



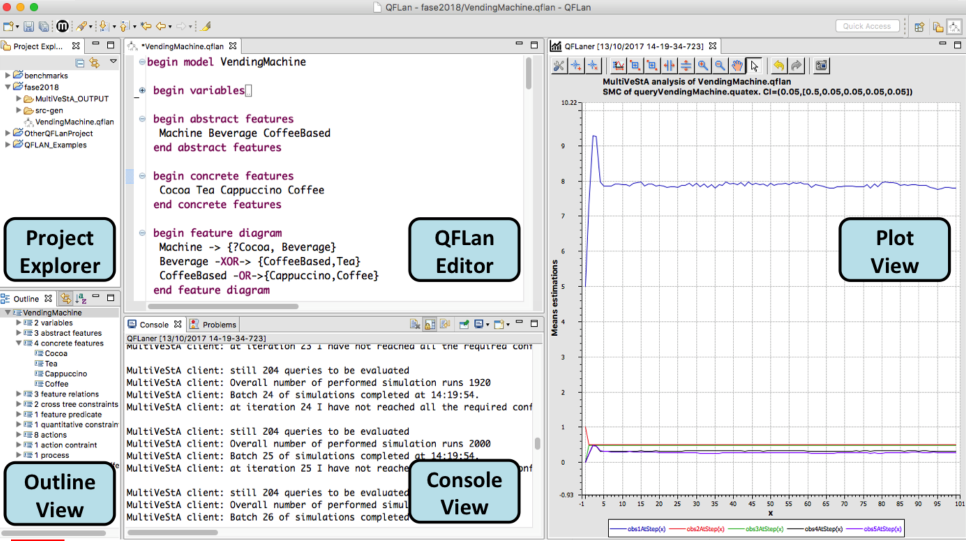
LTS

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 $\{a, b\}$

QFLan: a framework for quantitative modelling and analysis of highly (re)configurable systems

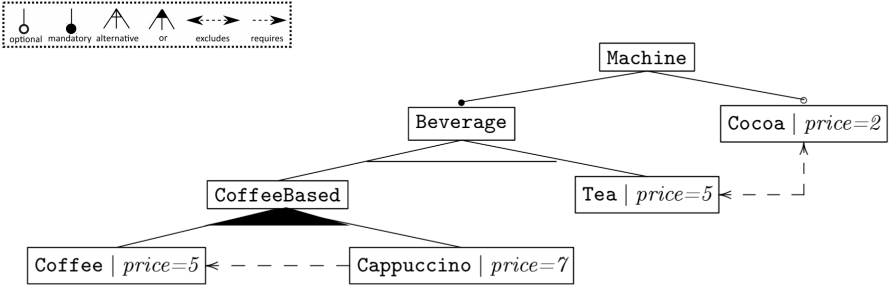
ter Beek *et al.* @ *IEEE TSE*, 2018, FM'18



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

These slides on QFLan are largely based on slides by Andrea Vandin

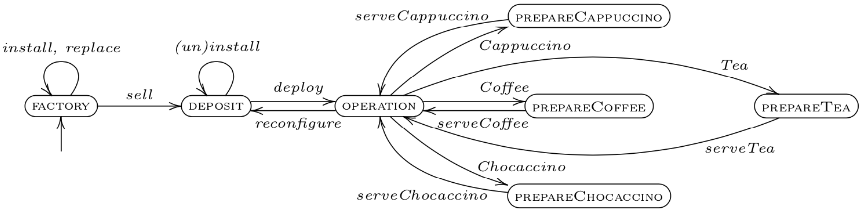
A simple coffee vending machine product line: attributed feature model with quantitative constraints




```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

begin concrete features
 Cocoa Tea Cappuccino Coffee
end concrete features

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

begin cross-tree constraints
 Cappuccino requires Coffee
 Tea excludes Cocoa
end cross-tree constraints
```

```
begin feature predicates
 price= { Cappuccino = 7, Coffee = 5,
          Cocoa = 2, Tea = 5 }
end feature predicates
begin quantitative constraints
{ price(Machine) <= 10 }
end quantitative constraints
```

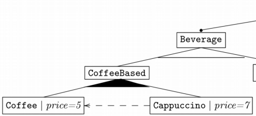
A simple coffee vending machine product line: probablistic behaviour with action constraints

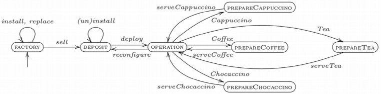



```
begin variables
 sold = 0
 deploys = 0
end variables

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 processes diagram
 begin process dynamics

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

 transitions =
   //Factory
   factory -(replace(Coffee,Tea),20)->factory,
   factory -(install(Cocoa),10)->factory,
   factory -(install(Cappuccino),10)->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,{deploys=deploys+1})-> operating

   //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,

   operating -(reconfigure,1) -> deposit
  end process
 end processes diagram
```

Pinpoint the precise moment in which a coffee machine is sold






```
begin quantitative constraints
 { price(Machine) <= 10 }
end quantitative constraints
```

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

```
begin variables
 sold = 0
 deploys = 0
end variables
```

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

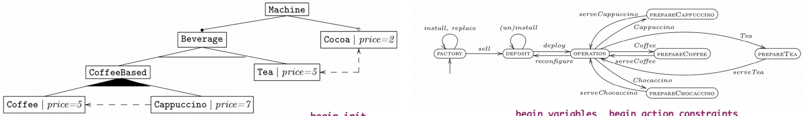
```
begin analysis

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),20)->factory,
   factory -(install(Cocoa),10)->factory,
   factory -(install(Cappuccino),10)->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,{deploys=deploys+1})-> operating ,
```

A simple coffee vending machine product line: statistical model checking with MultiVeSta

Sebastio & Vandin @ VALUETOOLS'13

Average price and probability of features of sold coffee machines




```
begin quantitative constraints
 { price(Machine) <= 15 }
end quantitative constraints
```

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

```
begin variables
 sold = 0
 deploys = 0
end variables
```

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

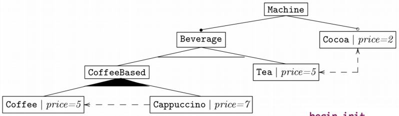
```
begin analysis

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
```

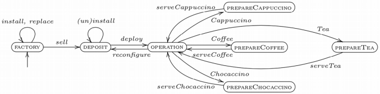
	Price	Coffee	Tea	Cappuccino	Cocoa
price(machine) <= 10	5.68	0.36	0.64	0.00	0.34
price(machine) <= 15	9.07	0.49	0.51	0.45	0.44

A simple coffee vending machine product line: statistical model checking with MultiVeSta ¬

Sebastio & Vandin @ VALUETOOLS'13

Average price and probability of features at varying of time





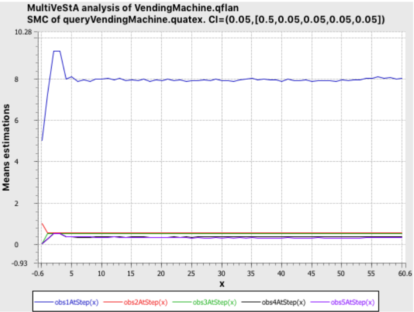

```
begin quantitative constraints
 { price(Machine) <= 15 }
end quantitative constraints
```

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

```
begin variables
 sold = 0
 deploys = 0
end variables
```

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

```
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
Chrszon *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
Chrszon *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 μ-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
Chrszon *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
Chrszon *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 μ-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
Chrszon *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
Chrszon *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 μ-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

Model checking the coffee machine with mCRL2

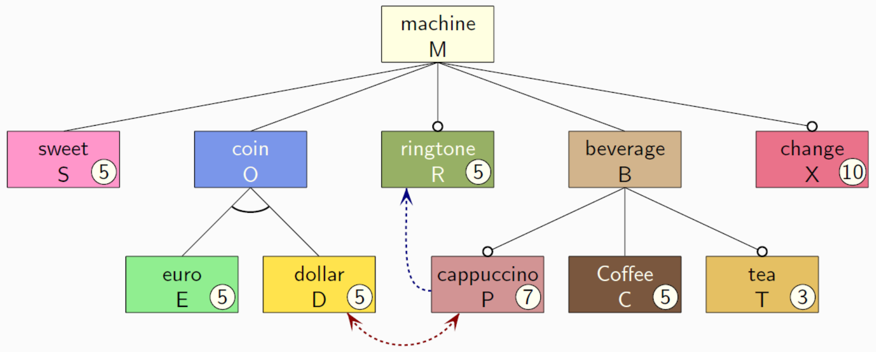
Recall: a product line or family of coffee machines

- 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



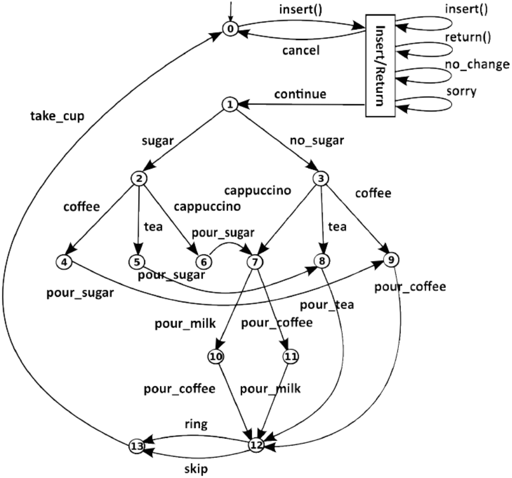
total cost $\leqslant 30$

Recall:

Non-functional attributes: $cost\,(product) = \sum\{\, cost\,(feature) \mid feature \in product \,\}$

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

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 ;
```

mCRL2 code: auxiliary operations (1/3)

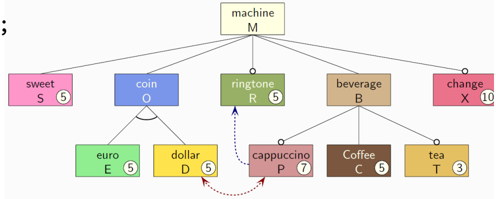
```
map isSorted: FSet -> Bool;
    noDuplicates: FSet -> Bool;
    isSet: FSet -> Bool;
var ft,ft': Feature;
    fset: FSet;
eqn isSorted([]) = true;
    isSorted([ft]) = true;
    isSorted(ft |> (ft' |> fset)) =
      ft <= ft' && isSorted(ft' |> fset);
    noDuplicates([]) = true;
    noDuplicates(ft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

mCRL2 code: auxiliary operations (2/3)

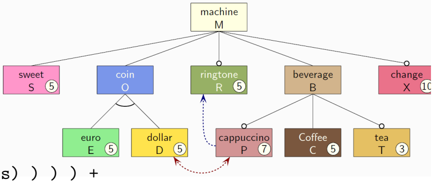
```
map insert: Feature # FSet -> FSet;
var ft, ft': Feature;
    fset: FSet;
eqn insert(ft, []) = [ft];
    (ft < ft')  -> insert(ft, ft' |> fset) = ft |> ft' |> fset;
    (ft == ft') -> insert(ft, ft' |> fset) = ft' |> fset;
    (ft > ft')  -> insert(ft, ft' |> fset) = ft' |> insert(ft, fset);

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');
```

mCRL2 code: auxiliary operations (3/3)

```
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) ;
```

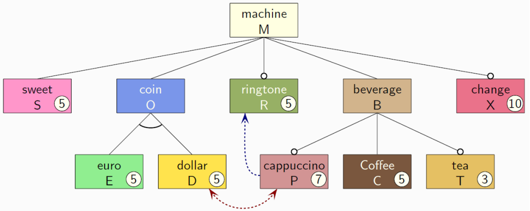


mCRL2 code: selection process Sel (1/2)




```
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(O,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 ) -> (
    ( O 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) ) ) ) +
    ...
```

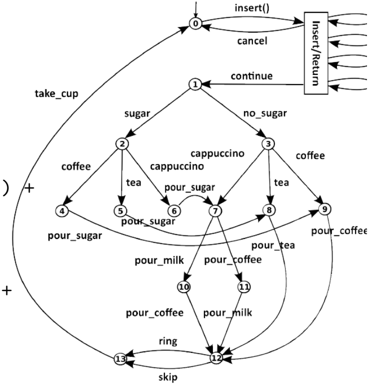
mCRL2 code: selection process `Sel` (2/2)




```
  ...
( 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
) ;
```

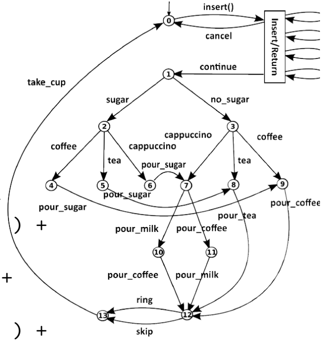
mCRL2 code: product behaviour process Prod (1/2)

```
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) ) ) +
    ...
```



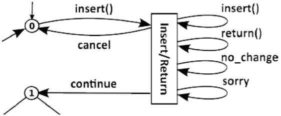
mCRL2 code: product behaviour process Prod (2/2)

```
...
( st == 7 ) -> (
  ( M in fs ) -> ( pour_milk . Prod(10,fs) ) +
  ( M in fs ) -> ( pour_coffee . Prod(11,fs) ) ) +
( st == 8 ) -> (
  ( M in fs ) -> ( pour_tea . 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) ) ) ;
```

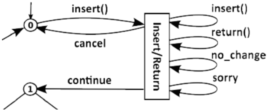


mCRL2 code: money handling process Insert

```
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) )
  ) ;
```

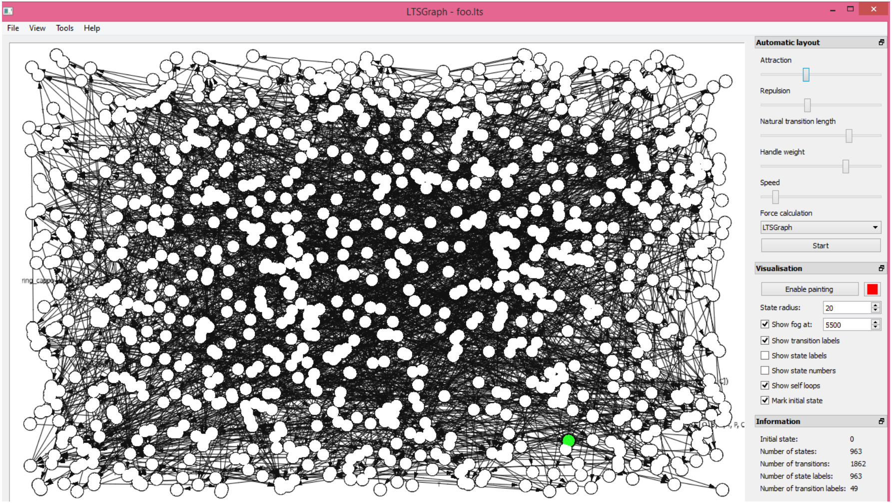


mCRL2 code: money handling process Return

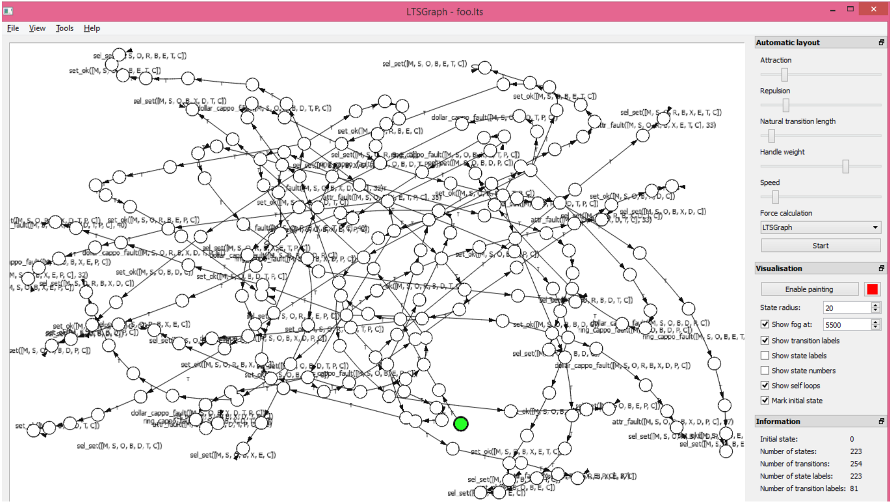



```
proc Return(bal:Nat,fs:FeatSet) =
  ( bal == 0 ) -> tau +
  ( D in fs ) -> (
    ( bal >= 50 ) ->
      return(half) . Return(Int2Nat(bal-50),fs) +
    ( ( bal < 50 ) && ( bal >= 25 ) ) ->
      return(quarter) . Return(Int2Nat(bal-25),fs) +
    ( ( bal < 25 ) && ( bal >= 10 ) ) ->
      return(dime) . 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)
  ) ;
```

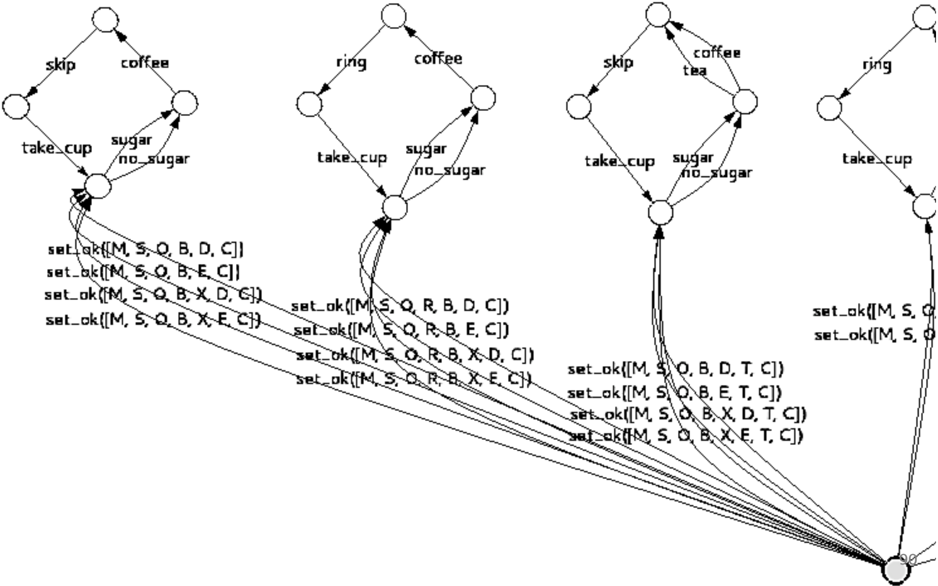
Visual inspection of the state space



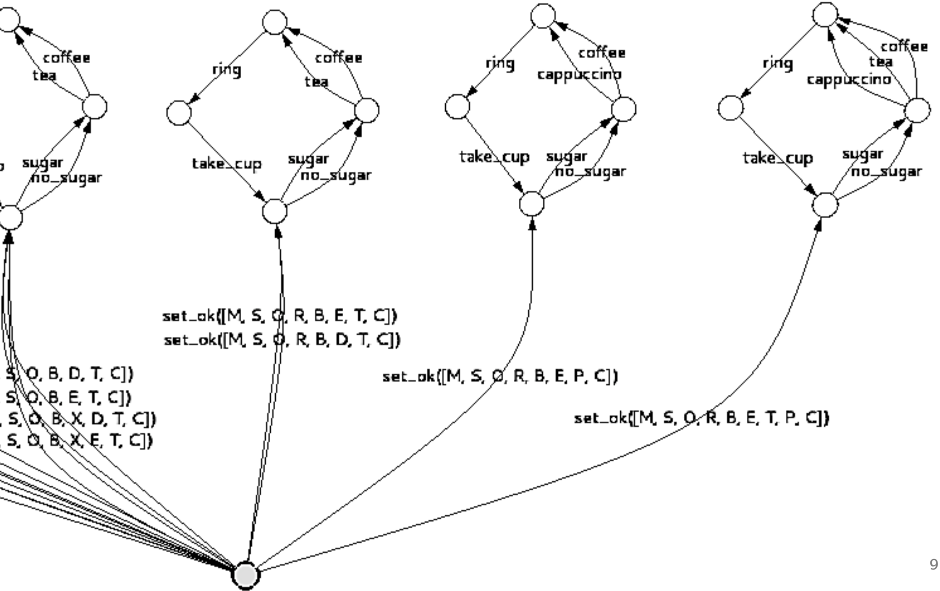
Visual inspection of the state space: behavioural reduction



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.`
  `      <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 fs:FSet.set_ok(fs)> true ||`
  `      <exists 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.(<cancel||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

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,
- Probablistic/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,
  - Probablistic/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,
  - Probablistic/Statistical Model Checking, SMT solving
    - SNIP-Z3, ProFeat, QFLan