# 4. Feature Modeling – Handout

Software Product Lines | Gunter Saake, Elias Kuiter | October 27, 2025

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

# 4. Feature Modeling

# Recap: Software Product Lines [Lecture 1]

**Software Product Line** [Northrop et al. 2012, p. 5]

"A **software product line** is

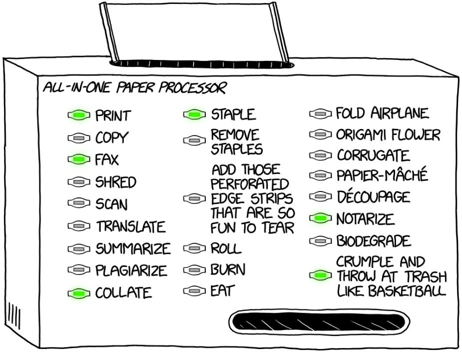
- a set of software-intensive systems
- that share a common, managed set of features
- satisfying the specific needs of a particular market segment or mission
- and that are developed from a common set of core assets in a prescribed way." [Software Engineering Institute, Carnegie Mellon University]

**Product** [Apel et al. 2013, p. 19]

"A **product of a product line** is specified by a valid feature selection (a subset of the features of the product line). A feature selection is **valid** if and only if it fulfills all feature dependencies."

**Feature** [Apel et al. 2013, p. 18]

"A **feature** is a characteristic or end-user-visible behavior of a software system."



# Features Have Dependencies

## Ordering a Waffle ...


## ... with Sugar

## ... with Cherries

## This is Nice, But ...

- plate and sugar seem to always be included, a fork is only included for some orders
  $\Rightarrow$ limitations seem **arbitrary**
- children get special treatment
  $\Rightarrow$ order process is **unfair**
- what exactly am I paying for?
  $\Rightarrow$ investments are **unclear**

## In This Lecture

1. how to **model and configure** features and their dependencies?
2. how to **store and communicate**?
3. how to **analyze and understand**?

# Specifying Valid Configurations

### Configuration

- a **configuration** (Konfiguration) over a set of features $F$ selects and deselects features in $F$
- formally: a pair $(S, D)$ such that $S, D \subseteq F$ and $S, D$ are disjoint ($S \cap D = \emptyset$)
- is **complete** (vollständig) if all features are covered ($S \cup D = F$) and **partial** (partiell) otherwise
- a complete configuration is **valid** (gültig) if it "makes sense" in the domain and **invalid** (ungültig) otherwise
- we often abbreviate complete configurations with $S \equiv (S, F \setminus S)$

Feature set $F = \{\mathbf{C}onfigDB, \mathbf{G}et, \mathbf{P}ut, \mathbf{D}elete, \mathbf{T}ransactions, \mathbf{W}indows, \mathbf{L}inux\}$

Examples for **complete** configurations:

- **valid** (read-only database on Windows): $(\{C, G, W\}, \{P, D, T, L\})$
- **valid** (fully functional database on Linux): $(\{C, G, P, D, T, L\}, \{W\})$
- **invalid** ($\lightning$ no operating system): $(\{C, G\}, \{P, D, T, W, L\})$
- **invalid** (transactions $\lightning$ read-only database): $(\{C, G, T, L\}, \{P, D, W\})$

Examples for **partial** configurations: $(\{C, G\}, \{P, D\})$, $(\emptyset, \emptyset)$

# Specifying Valid Configurations – Natural Language

**Valid Configuration**

A complete configuration over $F$ is valid if it "makes sense" in the domain. $\rightsquigarrow$ **"makes sense"?**

**Natural Language**

- informal description of relationships between features in $F$
- a complete configuration $S$ is **valid** if it conforms to the description

\+ succinct

– sometimes ambiguous

– not machine-readable

"A **configurable database** has an API that allows for at least one of the request types **Get**, **Put**, or **Delete**. Optionally, the database can support **transactions**, provided that the API allows for Put or Delete requests. Also, the database targets a supported operating system, which is either **Windows** or **Linux**."

# Specifying Valid Configurations – Configuration Map

### Valid Configuration

A complete configuration over $F$ is valid if it "makes sense" in the domain. $\rightsquigarrow$ **"makes sense"?**

### Configuration Map

- a **configuration map** over $F$ is a set of complete configurations $M \subseteq \mathcal{P}(F)$
- a complete configuration $S$ is **valid** if it occurs in the configuration map ($S \in M$)
- also known as product map

+ precise
− not human-readable
− redundant, explodes in size ($0 \leq |M| \leq 2^{|F|}$)

Feature set $F = \{\textbf{C}onfigDB, \textbf{G}et, \textbf{P}ut, \textbf{D}elete, \textbf{T}ransactions, \textbf{W}indows, \textbf{L}inux\}$

Configuration map:

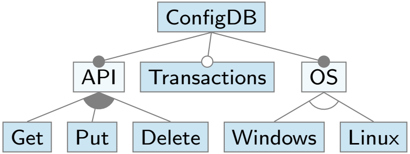
| | |
|---|---|
| $\{C, G, W\}$ | $\{C, G, L\}$ |
| $\{C, P, W\}$ | $\{C, P, L\}$ |
| $\{C, G, P, W\}$ | $\{C, G, P, L\}$ |
| $\{C, D, W\}$ | $\{C, D, L\}$ |
| $\{C, G, D, W\}$ | $\{C, G, D, L\}$ |
| $\{C, P, D, W\}$ | $\{C, P, D, L\}$ |
| $\{C, G, P, D, W\}$ | $\{C, G, P, D, L\}$ |
| $\{C, P, T, W\}$ | $\{C, P, T, L\}$ |
| $\{C, G, P, T, W\}$ | $\{C, G, P, T, L\}$ |
| $\{C, D, T, W\}$ | $\{C, D, T, L\}$ |
| $\{C, G, D, T, W\}$ | $\{C, G, D, T, L\}$ |
| $\{C, P, D, T, W\}$ | $\{C, P, D, T, L\}$ |
| $\{C, G, P, D, T, W\}$ | $\{C, G, P, D, T, L\}$ |

# Specifying Valid Configurations – Configuration Map in Excel

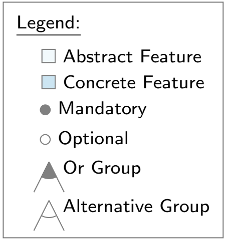
| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | **ConfigDB** | **Get** | **Put** | **Delete** | **Transactions** | **Windows** | **Linux** |
| 2 | x | x | | | | x | |
| 3 | x | | x | | | x | |
| 4 | x | x | x | | | x | |
| 5 | x | | | x | | x | |
| 6 | x | x | | x | | x | |
| 7 | x | | x | x | | x | |
| 8 | x | x | x | x | | x | |
| 9 | x | | x | | x | x | |
| 10 | x | x | x | | x | x | |
| 11 | x | | | x | x | x | |
| 12 | x | x | | x | x | x | |
| 13 | x | | x | x | x | x | |
| 14 | x | x | x | x | x | x | |
| 15 | x | x | | | | | x |
| 16 | x | | x | | | | x |
| 17 | x | x | x | | | | x |
| 18 | x | | | x | | | x |
| 19 | x | x | | x | | | x |
| 20 | x | | x | x | | | x |
| 21 | x | x | x | x | | | x |
| 22 | x | | x | | x | | x |
| 23 | x | x | x | | x | | x |
| 24 | x | | | x | x | | x |
| 25 | x | x | | x | x | | x |
| 26 | x | | x | x | x | | x |
| 27 | x | x | x | x | x | | x |

**Can we do better?**

# Feature Models – Syntax [Apel et al. 2013; Kang et al. 1990, pp. 63–72; Batory 2005]




$Transactions \rightarrow Put \lor Delete$



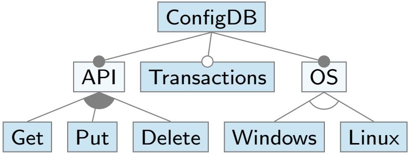

**Feature Model (Feature-Modell)**

- hierarchy of features
- dependencies between features modeled by tree and cross-tree constraints
- **tree constraints**: defined by the hierarchy
- **cross-tree constraints**: propositional formulas over features
- **abstract features** are used to group other features
- **concrete features** have an implementation
- also known as feature diagram or feature tree
- notation for **optional/mandatory features** and **or/alternative groups**

# Feature Models – Semantics [Apel et al. 2013; Batory 2005]

### Tree Constraints

- the **root feature** is always required
- each feature requires its parent (aka. **parent-child-relationship**)
- an **optional feature** can be (de-)selected freely when its parent is selected
- a **mandatory feature** is required by its parent
- **or group**: at least one child feature must be selected when the parent is selected
- **alternative group**: exactly one child feature must be selected when the parent is selected

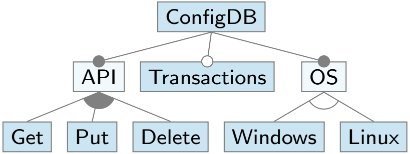



$$\textit{Transactions} \rightarrow \textit{Put} \vee \textit{Delete}$$

### Cross-Tree Constraints

- a list of **propositional formulas** expressing further dependencies between features
- each cross-tree constraint must be satisfied

# Feature Models – Examples

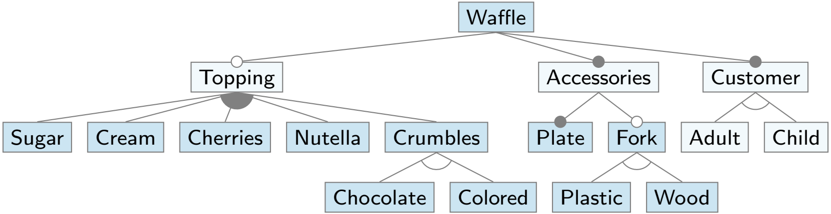



$$Transactions \rightarrow Put \vee Delete$$

**Is This a Valid Configuration?**

- **valid** (read-only database on Windows): $(\{C, A, G, O, W\}, \{P, D, T, L\})$
- **valid** (fully functional database on Linux): $(\{C, A, G, P, D, T, O, L\}, \{W\})$
- **invalid** ($\lightning$ no operating system): $(\{C, A, G\}, \{P, D, T, O, W, L\})$
- **invalid** (transactions $\lightning$ read-only database): $(\{C, A, G, T, O, L\}, \{P, D, W\})$

# Feature Models – Examples




$$\mathit{Sugar}$$
$$\mathit{Cherries} \rightarrow \mathit{Sugar} \land \mathit{Fork}$$
$$\mathit{Nutella} \lor \mathit{Crumbles} \rightarrow \mathit{Child}$$
$$\mathit{Fork} \rightarrow \mathit{Adult}$$

- every feature (leaf or compound) can be abstract or concrete
- groups can be used anywhere
- directly below groups, no optional or mandatory markers are allowed

# Discussion of Feature Models

## Pro: Making Tacit Knowledge Explicit

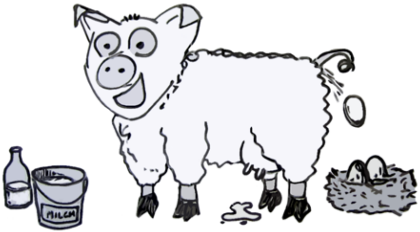
"I think the best [about feature modeling] is you can see relationships, to actually know what configurations are allowed and what are not allowed. That was also not so easy to express in the past [...] This is from the developer's point of view. But it's also [...] important, because before we noticed that **the same functionality was implemented twice** within the same project, basically they haven't realized that. They implemented the same features." – Interview with Practitioners [Berger et al. 2014]

## Pro: Tool Support

FeatureIDE, Gears, pure::variants, ...

## Con: Challenges

- **domain scoping**: which features?
- **feature interactions**: which dependencies?
- requires infrastructure, consulting, and training



# Feature Models and Configurations – Summary

## Lessons Learned

- features, dependencies between features, and configurations
- feature models: abstract and concrete features, tree and cross-tree constraints
- tree constraints: optional, mandatory, or group, alternative group

## Further Reading

- Apel et al. 2013, Section 2.3, pp. 26–39
  — introduction to feature modeling
- Thorsten Berger et al. (2013): A Survey of Variability Modeling in Industrial Practice
- Damir Nešić et al. (2019): Principles of Feature Modeling

## Practice

1. sketch a feature model with features $A, B, C, D, E, F$ that has exactly those 5 valid configurations (pen and paper preferred):

   $\{A, B\}$ $\{A, C, E\}$ $\{A, C, E, F\}$
   $\{A, B, D\}$ $\{A, C, F\}$

2. discuss in groups whether your feature models are syntactically correct and specify exactly the above configurations

bit.ly/spl-practice

# 4. Feature Modeling

# Representations and Transformations

## Natural Language

"A **configurable database** has an API that allows for at least one of the request types **Get**, **Put**, or **Delete**. Optionally, the database can support **transactions**, provided that the API allows for Put or Delete requests. Also, the database targets a supported operating system, which is either **Windows** or **Linux**."

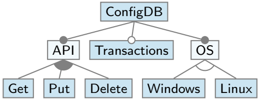
## Configuration Map

$\{C, G, W\}$ $\{C, G, L\}$

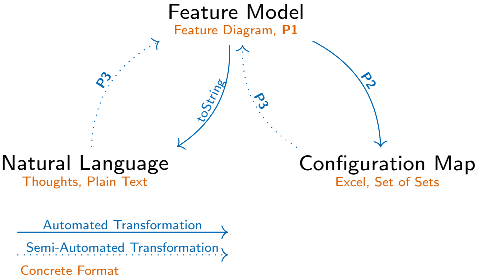
$\vdots$ $\vdots$

$\{C, G, P, D, T, W\}$ $\{C, G, P, D, T, L\}$

## Feature Diagram (Graphical Feature Model)




$Transactions \rightarrow Put \vee Delete$




## Problems

P1 How to express feature models **textually**?

P2 How to (a) validate configurations and (b) get all valid configurations **automatically**?

P3 (How to reverse engineer feature models?)

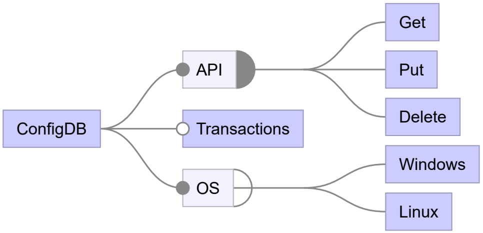
# UVL, the Universal Variability Language [UVL]

```
features
  ConfigDB
    mandatory
      API {abstract}
        or
          Get
          Put
          Delete

    optional
      Transactions
    mandatory
      OS {abstract}
        alternative
          Windows
          Linux

constraints
  Transactions => Put | Delete
```

**A Feature Model "Sideways"**

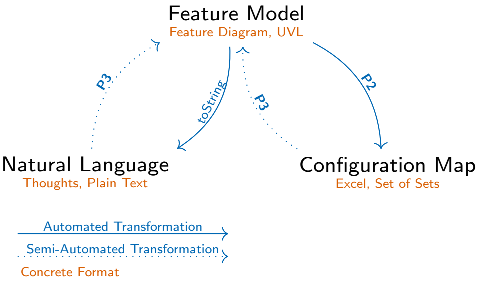



*Transactions* $\rightarrow$ *Put* $\vee$ *Delete*

**Universal Variability Language (UVL)**

- textual language for feature modeling
- adds advanced modeling constructs (e.g., attributes, cardinalities, submodels, . . . )

# Representations and Transformations




**Problems**

P1 How to express feature models **textually**?

P2 How to (a) validate configurations and (b) get all valid configurations **automatically**?

P3 (How to reverse engineer feature models?)

**Solutions**

P1 Universal Variability Language ⇒ **Syntax**

P2 **Semantics**?

P3 –

# Propositional Formulas – Recap

## Syntax of Propositional Formulas

Inductive definition of **propositional formulas** (aussagenlogische Formeln)

- the **Boolean truth values** $\top$, $\bot$
- any **Boolean variable** $X$
- any **negation** $\neg\phi$ of a formula $\phi$
- any **conjunction** $(\phi \land \psi)$ of formulas $\phi$ and $\psi$
- any **disjunction** $(\phi \lor \psi)$, **implication** $(\phi \rightarrow \psi)$, or **biimplication** $(\phi \leftrightarrow \psi)$

## Informal Semantics of Propositional Formulas

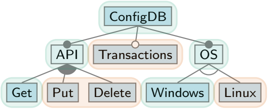
$$\left.\begin{array}{l} \top \\ \bot \\ \neg\phi \\ \phi \land \psi \\ \phi \lor \psi \\ \phi \rightarrow \psi \\ \phi \leftrightarrow \psi \end{array}\right\} \text{ means } \left\{\begin{array}{l} \text{“true” (or **tautology**)} \\ \text{“false” (or **contradiction**)} \\ \text{“not } \phi\text{”} \\ \text{“}\phi \text{ and } \psi\text{”} \\ \text{“}\phi \text{ or } \psi\text{” (inclusive or!)} \\ \text{“if } \phi\text{, then } \psi\text{” (and else?)} \\ \text{“}\phi \text{ if and only if } \psi\text{”} \end{array}\right.$$

## Operator Precedence: $\neg$, $\land$, $\lor$, $\rightarrow$, $\leftrightarrow$

$$\begin{aligned} & \mathit{Transactions} \rightarrow (\mathit{Put} \lor \mathit{Delete}) \\ \equiv\ & \mathit{Transactions} \rightarrow \mathit{Put} \lor \mathit{Delete} \\ \not\equiv\ & (\mathit{Transactions} \rightarrow \mathit{Put}) \lor \mathit{Delete} \end{aligned}$$

# Propositional Formulas – Example

## A Feature Model *FM* ...




## ... as a Propositional Formula $\Phi(FM)$

$$
\begin{aligned}
\Phi(FM) = \; & ConfigDB \\
& \land (API \leftrightarrow ConfigDB) \\
& \land (Transactions \rightarrow ConfigDB) \\
& \land (OS \leftrightarrow ConfigDB) \\
& \land (Get \lor Put \lor Delete \leftrightarrow API) \\
& \land (Windows \lor Linux \leftrightarrow OS) \\
& \land \neg(Windows \land Linux) \\
& \land (Transactions \rightarrow Put \lor Delete)
\end{aligned}
$$

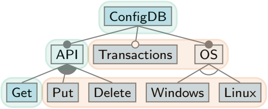
## Is This a Valid Configuration?

$$
\begin{aligned}
& \Phi(FM)(\{C, A, G, O, W\}) \\
\equiv \; & \Phi(FM)((\{C, A, G, O, W\}, \{P, D, T, L\})) \\
\equiv \; & C \land (A \leftrightarrow C) \land (T \rightarrow C) \land (O \leftrightarrow C) \\
& \land (G \lor P \lor D \leftrightarrow A) \land (W \lor L \leftrightarrow O) \\
& \land \neg(W \land L) \land (T \rightarrow P \lor D) \\
\equiv \; & \top \land (\top \leftrightarrow \top) \land (\bot \rightarrow \top) \land (\top \leftrightarrow \top) \\
& \land (\top \lor \bot \lor \bot \leftrightarrow \top) \land (\top \lor \bot \leftrightarrow \top) \\
& \land \neg(\top \land \bot) \land (\bot \rightarrow \bot \lor \bot) \\
\equiv \; & \top \land \top \land \top \land \top \land \top \land \top \land \top \land \top \\
\equiv \; & \top
\end{aligned}
$$

$\rightsquigarrow$ **configuration is valid**
(read-only database on Windows)

# Propositional Formulas – Example

## A Feature Model *FM* . . .




$Transactions \rightarrow Put \vee Delete$

## . . . as a Propositional Formula $\Phi(FM)$

$$
\begin{aligned}
\Phi(FM) = {} & ConfigDB \\
& \wedge (API \leftrightarrow ConfigDB) \\
& \wedge (Transactions \rightarrow ConfigDB) \\
& \wedge (OS \leftrightarrow ConfigDB) \\
& \wedge (Get \vee Put \vee Delete \leftrightarrow API) \\
& \wedge (Windows \vee Linux \leftrightarrow OS) \\
& \wedge \neg(Windows \wedge Linux) \\
& \wedge (Transactions \rightarrow Put \vee Delete)
\end{aligned}
$$

## Is This a Valid Configuration?

$$
\begin{aligned}
& \Phi(FM)(\{C, A, G\}) \\
\equiv {} & \Phi(FM)((\{C, A, G\}, \{P, D, T, O, W, L\})) \\
\equiv {} & C \wedge (A \leftrightarrow C) \wedge (T \rightarrow C) \wedge (O \leftrightarrow C) \\
& \wedge (G \vee P \vee D \leftrightarrow A) \wedge (W \vee L \leftrightarrow O) \\
& \wedge \neg(W \wedge L) \wedge (T \rightarrow P \vee D) \\
\equiv {} & \top \wedge (\top \leftrightarrow \top) \wedge (\bot \rightarrow \top) \wedge (\bot \leftrightarrow \top) \\
& \wedge (\top \vee \bot \vee \bot \leftrightarrow \top) \wedge (\bot \vee \bot \leftrightarrow \bot) \\
& \wedge \neg(\bot \wedge \bot) \wedge (\bot \rightarrow \bot \vee \bot) \\
\equiv {} & \top \wedge \top \wedge \top \wedge \bot \wedge \top \wedge \top \wedge \top \wedge \top \\
\equiv {} & \bot
\end{aligned}
$$

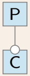
$\rightsquigarrow$ **configuration is invalid**
($\lightning$ no operating system)

# Propositional Formulas – Algorithm

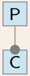
**Algorithm: Translate *FM* Into $\Phi(FM)$**

1. translate each tree constraint
   - **Root feature**: $R$ is always required
   - **Optional feature**: $C$ requires $P$
   - **Mandatory feature**:
     Optional $+$ $P$ requires $C$
   - **Or group**:
     Optional $+$ $P$ requires at least one $C_i$
   - **Alternative group**:
     Optional $+$ $P$ requires exactly one $C_i$
2. conjoin translated tree constraints
   $\Phi(TC) \leftarrow \bigwedge_{tc \in TC} \Phi(tc)$
3. conjoin **cross-tree constraints**
   $\Phi(CTC) \leftarrow \bigwedge_{ctc \in CTC} ctc$
4. $\Phi(FM) \leftarrow \Phi(TC) \wedge \Phi(CTC)$

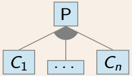
$$\Phi\left(\boxed{\text{Root}}\right) = Root$$

$\Phi\Big($ 

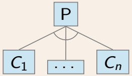
$\Big) = C \rightarrow P$

$\Phi\Big($ 

$\Big) = P \leftrightarrow C$

$\Phi\Big($ 

$\Big) = P \leftrightarrow \bigvee_{1 \le i \le n} C_i$

$\Phi\Big($ 

$\Big) = P \leftrightarrow \bigvee_{1 \le i \le n} C_i \wedge \bigwedge_{1 \le i < j \le n} \neg(C_i \wedge C_j)$

# CNF as a Universal Formula Language

## Recap: Conjunctive Normal Form

- a **literal** $L$ is a variable $X$ or its negation $\neg X$
- a **clause** $C$ is a disjunction of literals $\bigvee_j L_j$
- a **conjunctive normal form (CNF)** is a conjunction of clauses $\bigwedge_i C_i = \bigwedge_i \bigvee_j L_j$
- intuitively: a set of "rules" to be satisfied
- any formula $\phi$ can be transformed into a CNF $\phi'$ that is logically equivalent ($\phi \Leftrightarrow \phi'$)

## Recap: Laws of Propositional Logic

- implication: $\phi \rightarrow \psi \Leftrightarrow \neg\phi \vee \psi$
- biimplication: $\phi \leftrightarrow \psi \Leftrightarrow (\neg\phi \vee \psi) \wedge (\neg\psi \vee \phi)$
- De Morgan's laws: $\neg(\phi \wedge \psi) \Leftrightarrow \neg\phi \vee \neg\psi$
- distributivity: $(\phi \wedge \psi) \vee \chi \Leftrightarrow (\phi \vee \chi) \wedge (\psi \vee \chi)$

## Transforming Part of $\Phi(FM)$ into $CNF(\Phi(FM))$

$$
\begin{aligned}
&C\\
&\wedge (T \rightarrow C)\\
&\wedge (O \leftrightarrow C)\\
&\wedge (W \vee L \leftrightarrow O)\\
&\wedge \neg(W \wedge L)
\end{aligned}
$$

$$
\begin{aligned}
&C\\
&\wedge (\neg T \vee C)\\
&\wedge (\neg O \vee C) \wedge (\neg C \vee O)\\
&\wedge (\neg(W \vee L) \vee O)\\
&\wedge (\neg O \vee W \vee L)\\
&\wedge \neg(W \wedge L)
\end{aligned}
$$

$$
\begin{aligned}
&C\\
&\wedge (\neg T \vee C)\\
&\wedge (\neg O \vee C) \wedge (\neg C \vee O)\\
&\wedge ((\neg W \wedge \neg L) \vee O)\\
&\wedge (\neg O \vee W \vee L)\\
&\wedge (\neg W \vee \neg L)
\end{aligned}
$$

$$
\begin{aligned}
&C\\
&\wedge (\neg T \vee C)\\
&\wedge (\neg O \vee C) \wedge (\neg C \vee O)\\
&\wedge (\neg W \vee O) \wedge (\neg L \vee O)\\
&\wedge (\neg O \vee W \vee L)\\
&\wedge (\neg W \vee \neg L)
\end{aligned}
$$

# CNF as a Universal Formula Language – DIMACS

$$C \\ \wedge (\neg T \vee C) \\ \wedge (\neg O \vee C) \wedge (\neg C \vee O) \\ \wedge (\neg W \vee O) \wedge (\neg L \vee O) \\ \wedge (\neg O \vee W \vee L) \\ \wedge (\neg W \vee \neg L)$$

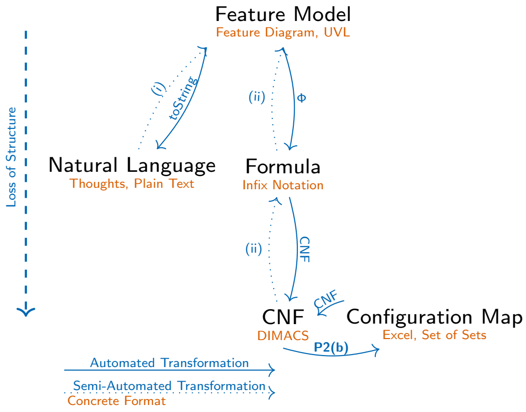
```
c 1 C
c 2 T
c 3 O
c 4 W
c 5 L
p cnf 5 8
1 0
-2 1 0
-3 1 0 -1 3 0
-4 3 0 -5 3 0
-3 4 5 0
-4 5 0
```

**DIMACS Format** [DIMACS 1993]

- de facto industry standard for storing CNF
- machine-readable, automated analyses, ...
- comments start with `c ...`
- problem line:
  `p cnf #variables #clauses`
- clause $\bigvee_i L_i$ translates to `L1 ... Ln 0`
- intuitively:

$$\left.\begin{array}{l} \texttt{0} \\ \texttt{␣} \\ \texttt{-} \end{array}\right\} \text{means} \left\{\begin{array}{l} \wedge \\ \vee \\ \neg \end{array}\right.$$

# Representations and Transformations




**Problems**

P1 How to express feature models **textually**?

P2 How to
(a) validate configurations and
(b) get all valid configurations
**automatically**?

P3 (How to reverse engineer feature models?)

**Solutions**

P1 Universal Variability Language ⇒ **Syntax**

P2 Propositional Formulas ⇒ **Semantics**
(a) evaluate feature-model formula
(b) Lecture 4c

P3 (i) e.g., Bakar et al. 2015
(ii) e.g., Czarnecki and Wasowski 2007

# Transforming Feature Models – Summary

### Lessons Learned

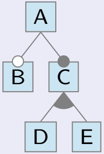
- to understand large configuration spaces, we need formal semantics and machine-readable representations
- propositional formulas satisfy many (though not all) needs for such a representation

### Further Reading

- Don Batory (2005): Feature Models, Grammars, and Propositional Formulas
- UVL — official website for the Universal Variability Language with examples, grammar, literature pointers
- Alexander Knüppel et al. (2017): Is There a Mismatch Between Real-World Feature Models and Product-Line Research?

### Practice

1. translate the following feature diagram into a propositional formula:




$D \rightarrow B$

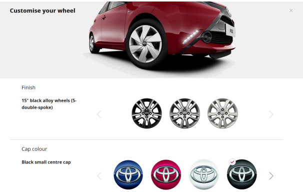
2. check formulas of your colleagues

bit.ly/spl-practice

# 4. Feature Modeling

# Configurators in the Wild – Cars

## Configuring a Car ...




Demo Video

## with a Weird Price

**Your AYGO**

5 Door Hatchback x-play 1.0 Petrol (69 hp) Automatic (Front Wheel Drive - FWD)

| | |
|---|---|
| Retail price | **£11,610.00** |
| Red Pop | |
| Gleam fabric | |
| 15" black alloy wheels (5-double-spoke) | £500.00 |
| Black small centre cap | |
| **Total** | £12,610.00 |

Summary and save ›

## with 8 Wheels!

**Your customisation** — Change customisation

**Colours & wheels**

Red Pop

| | |
|---|---|
| 15" colour-customisable alloy wheels (5 double-spoke) | £500.00 |
| 15" black machined-face alloy wheels (5-double-spoke) | £500.00 |
| Black small centre cap | |

**Interior trim**

Gleam fabric

**Total**

£12,610.00

- canceling the dialog was not considered and lead to an invalid state (i.e., configuration)
- humans check these configurations, but some errors are only found during production
- many constraints: appear arbitrary, not explained

# Configurators in the Wild – Cars

## Configuring a German Car

[example from Lecture 1]

Configuration Assistant.

› Show instructions

Your most recent action requires your configuration to be adjusted.

| Your choice | Price |
|---|---|
| + Enhanced Bluetooth telephone with USB & Voice Control | + £ 350.00 |

Adding

| | |
|---|---|
| + BMW Navigation | £ 0.00 |

Removing

| | |
|---|---|
| - Enhanced Bluetooth with wireless charging | - £ 395.00 |
| - Navigation system Professional | £ 0.00 |
| - WiFi hotspot preparation | £ 0.00 |
| - Media package - Professional | - £ 900.00 |
| - Online Entertainment | £ 0.00 |
| - Microsoft Office 365 | - £ 150.00 |

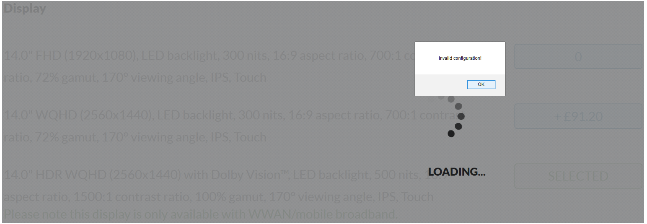
Why does the telephone conflict with Microsoft Office?

# Configurators in the Wild – Notebooks

## Configuring a Notebook

**Display**

| | |
|---|---|
| 14.0" FHD (1920x1080), LED backlight, 300 nits, 16:9 aspect ratio, 700:1 contrast ratio, 72% gamut, 170° viewing angle, IPS, Touch | SELECTED |
| 14.0" WQHD (2560x1440), LED backlight, 300 nits, 16:9 aspect ratio, 700:1 contrast ratio, 72% gamut, 170° viewing angle, IPS, Touch | + £91.20 |
| 14.0" HDR WQHD (2560x1440) with Dolby Vision™, LED backlight, 500 nits, 16:9 aspect ratio, 1500:1 contrast ratio, 100% gamut, 170° viewing angle, IPS, Touch<br>Please note this display is only available with WWAN/mobile broadband. | + £159.60 |




can detect mistakes, but provides no explanations or fixes

# Configurators in the Wild – Notebooks

## Still Configuring a Notebook

**Microsoft Productivity Software**

| | |
|---|---|
| None | SELECTED |
| Microsoft Office 365 Home | + £59.99 |
| Microsoft Office 365 Personal | + £79.99 |
| Microsoft Office Home and Student 2016 | + £119.99 |
| Microsoft Office Home and Business 2016 | + £229.99 |
| Adobe Acrobat Standard 2017 and Microsoft Office Home and Business 2016 | + £399.60 |
| Adobe Acrobat Standard 2017 and Microsoft Office | + £628.80 |

Microsoft Office Not Included

For your best experience, Lenovo recommends selecting a Microsoft Office product with your new purchase.

NEED HELP DECIDING?

Roll over each product to get specific details on each Office product

allows some feature combinations and not others, prices are opaque

# Automated Analysis of Feature Models

## Open Questions

- How do such configurators work?
- How to avoid inconsistencies?
- How to provide explanations and fixes?
- How to get all valid configurations automatically? (**P2(b)**)

## Automated Analysis of Feature Models

- up until now: **creation** and **transformation** of feature models
- now: **analysis** of feature models to improve our understanding of a configuration space
- for brevity: product = valid configuration

## Asking Questions About Feature Models

- Is a given configuration valid?
- Is there any product at all?
  How many/which products are there?
- Is a given feature (de-)selectable at all?
  How many/which products include it?
- Is a given partial configuration consistent?
  How many/which products include it?
- (Which features always occur together?)
- (Is a given constraint redundant?)
- (How do two feature model versions differ?)
- (Why is . . . ? How to fix . . . ?)

# SAT, #SAT, and AllSAT

**Recap: Boolean Satisfiability Problem (SAT)**

- **decision problem**: is there any assignment $A$ that satisfies a given formula?
- formally: $SAT(\phi) \Leftrightarrow \exists A\colon \phi(A) = \top$
- known to be **NP-complete**:
  in theory, difficult to solve if $P \neq NP$;
  in practice, solvability depends on domain
- answered by **SAT solvers**:
  highly-optimized, off-the-shelf tools;
  competitively developed over several decades;
  takes a CNF in DIMACS format as input

- $X \rightarrow Y$ is satisfiable
- $X \lor \neg X$ is satisfiable (even a tautology)
- $X \land \neg X$ is not satisfiable (why?)

**Sharp Satisfiability Problem (#SAT)**

- **counting problem**: how many assignments satisfy a given formula?
- $\#SAT(\phi) = |\{A \mid \phi(A) = \top\}|$
- known to be **#P-complete**:
  at least as hard as SAT (probably harder)
- answered by **#SAT solvers**

**Solution Enumeration Problem (AllSAT)**

- **enumeration problem**: which assignments satisfy a given formula?
- $AllSAT(\phi) = \{A \mid \phi(A) = \top\}$
- at least as hard as #SAT (probably harder)
- answered by **AllSAT solvers**

# Automated Analysis of Feature Models

## Asking Questions About Feature Models

- Is a given configuration valid? $\Rightarrow$ **evaluate**
- **Is** there any valid configuration at all?
  **How many**/**which** valid configurations are there?
- **Is** a given feature (de-)selectable at all?
  **How many**/**which** valid configurations include it?
- **Is** a given partial configuration consistent?
  **How many**/**which** valid configurations include it?

## Choosing the Right Solver

- "is?" $\approx$ SAT solver query
- "how many?" $\approx$ #SAT solver query
- "which?" $\approx$ AllSAT solver query

## Typical Feature-Model Analysis Process

Feature Model $\xrightarrow{\Phi}$ Formula $\xrightarrow{\mathit{CNF}}$ DIMACS

formulate

$\rightarrow$ Query $\xrightarrow[\text{SAT, \#SAT, AllSAT}]{\text{solve}}$ Answer $\xrightarrow{\text{interpret}}$ Result

for brevity, we assume that $\phi = \mathit{CNF}(\Phi(\mathit{FM}))$ for a given feature model $\mathit{FM}$

# Consistency, Cardinality, and Enumeration – Feature Model

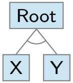
## Consistency of Feature Models (SAT)

### Void/Consistent Feature Model

- are there grave modeling errors?
- is it possible to configure any product at all?

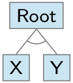
$\phi \xrightarrow{\text{SAT}} \bot/\top$ $\xrightarrow{\bot}$ *FM* is **void**

$\phantom{\phi \xrightarrow{\text{SAT}} \bot/\top}$ $\xrightarrow{\top}$ *FM* is **consistent**




$\neg(X \vee Y)$

void




$X \vee Y$

consistent

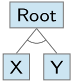
## Cardinality of Feature Models (#SAT)

### How Many Products Are There?

$$\phi \xrightarrow{\#\text{SAT}} |C|$$

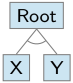
### Variability Factor: Share of Products?

$$\phi \xrightarrow{\#\text{SAT}} |C| \longrightarrow \frac{|C|}{2^{|F|}}$$




$\neg(X \vee Y)$

0 products, VF 0




$X \vee Y$

2 products, VF $\frac{2}{8}$

# Consistency, Cardinality, and Enumeration – Feature Model

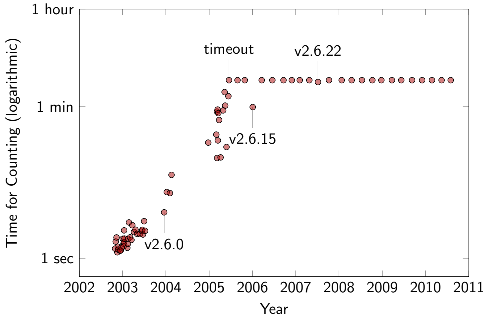
## Feasibility of SAT-Based Analyses

### Is SAT-Based Analysis "Easy"?

- provocative claim: "SAT-based analysis of feature models is easy" [Mendonca et al. 2009]
- easy = performs much better than expected (although NP-complete)
- easy = fast?
  - what about formulating the query? (e.g., CNF transformation)
  - what about many queries? (e.g., what we discuss next)

## Feasibility of #SAT-Based Analyses

### Time to Count Products of Linux




- the solver from 2023 can solve models from 2003
- can we analyze the models from 2023 in 2043?

# Consistency, Cardinality, and Enumeration – Feature Model

## Enumeration of Feature Models (AllSAT)

### Which Products Are There?

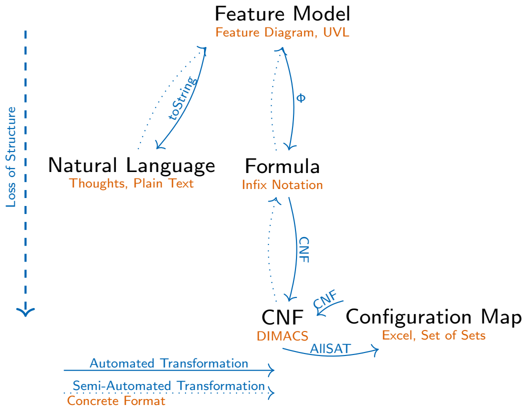
- **P2(b)**: How to get all products?

$$\phi \xrightarrow{\text{AllSAT}} C$$

AllSAT does not scale to realistic feature models!
50 features, configurations à 1 Byte ≈ 1 Petabyte


$\neg(X \vee Y)$ | $X \vee Y$

$\varnothing$ | $\{\{Root, X\}, \{Root, Y\}\}$



# Consistency, Cardinality, and Enumeration – Features

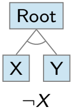
## Consistency of Features (SAT)

### Core/Dead Feature

- can a feature $F$ be (de-)selected at all?

$$\phi \wedge F \xrightarrow{\text{SAT}} \bot/\top \begin{cases} \xrightarrow{\bot} F \text{ is dead} \\ \xrightarrow{\top} F \text{ is not dead} \end{cases}$$

$$\phi \wedge \neg F \xrightarrow{\text{SAT}} \bot/\top \begin{cases} \xrightarrow{\bot} F \text{ is core} \\ \xrightarrow{\top} F \text{ is not core} \end{cases}$$




$X$ is dead

*Root* and $Y$ are core

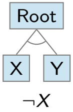
## Cardinality of Features (#SAT)

### How Many Products Include Feature $F$?

$$\phi \wedge F \xrightarrow{\#\text{SAT}} |\{S \in C \mid F \in S\}|$$

### Commonality: How Constrained is This Feature?

$$\phi \wedge F \xrightarrow{\#\text{SAT}} |\{S \in C \mid F \in S\}| \rightarrow \frac{|\{S \in C | F \in S\}|}{|C|}$$




$X$: 0 products

*Root* and $Y$: 1 products

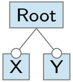
# Consistency, Cardinality, and Enumeration – Partial Configurations

## Consistency of Partial Configurations (SAT)

### Valid Partial Configuration

Is a partial configuration $C = (S, D)$ consistent with the feature model?

$$\phi \wedge \bigwedge_{s \in S} s \wedge \bigwedge_{d \in D} \neg d \xrightarrow{\text{SAT}} \bot / \top \quad \xrightarrow{\bot} C \times \qquad \xrightarrow{\top} C \checkmark$$



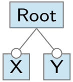

$$X \rightarrow Y$$

$$(\{Root\}, \{X\}) \checkmark$$

## Cardinality of Partial Configurations (#SAT)

### How Many Products Are Still Configurable for $C$?

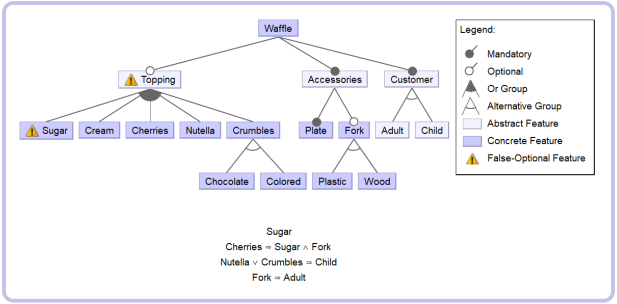
$$\phi \wedge \ldots \xrightarrow{\#\text{SAT}} |\{(S', D') \in C \mid S \subseteq S', D \in D'\}|$$

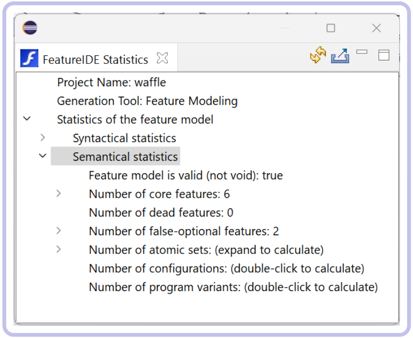



$$X \rightarrow Y$$

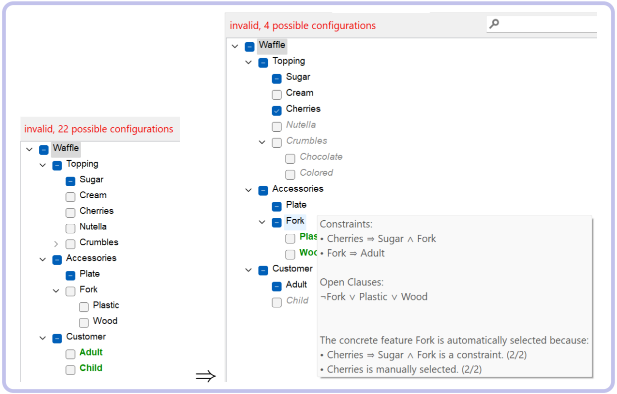
$(\{Root\}, \{X\})$: 2 products

# Automated Analyses in FeatureIDE – Feature-Model Editor



# Automated Analyses in FeatureIDE – Configuration Editor




## Decision Propagation

- after each decision (i.e., click) ...
  - ... select features that are now **conditionally core**
  - ... deselect features that are now **conditionally dead**
- this way it is impossible to configure an invalid configuration
- explanations for all propagated decisions

# Automated Analysis of Feature Models

## The Road So Far ...

Feature Model $\xrightarrow{\Phi}$ Formula $\xrightarrow{\mathit{CNF}}$ DIMACS

formulate

Query $\xrightarrow{\text{solve}}$ Answer $\longrightarrow$

void feature model
variability factor
all products
core/dead features
commonality
valid partial configuration
decision propagation

SAT
#SAT
AllSAT

## ... and Beyond

attributes
cardinalities
submodels

quantifiers
predicates
functions

DNF
d-DNNF
BDD

Feature Model $\xrightarrow{\Phi}$ Formula $\xrightarrow[\text{Tseitin}]{\mathit{CNF}}$ DIMACS

formulate

Query $\xrightarrow[\text{SAT, \#SAT, AllSAT}]{\text{solve}}$ Answer $\longrightarrow$

false-optional features
atomic sets
redundant constraints
feature-model edits
explanations
sampling
slicing

MAX-SAT
Solution-SAT
MUS
SMT, CSP
QBF
WMC, PMC

- develop new analyses
- improve efficiency of existing analyses
- investigate correctness and compositionality

# Analyzing Feature Models – Summary

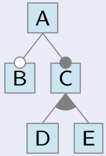
## Lessons Learned

- with solvers, we can build reliable configurators for product lines
- SAT-based analyses: void feature model, core/dead features, decision propagation
- #SAT-based analyses: variability factor, feature commonality

## Further Reading

- Apel et al. 2013, Section 10.1, pp. 244–254
  — introduction to feature-model analysis
- David Benavides et al. (2010): Automated Analysis of Feature Models 20 Years Later: A Literature Review
  — old but extensive literature survey
- Chico Sundermann et al. (2021): Applications of #SAT Solvers on Feature Models
  — experiments on the scalability of #SAT solvers

## Practice




think of a constraint that would make exactly one feature dead

# FAQ – 4. Feature Modeling

## Lecture 4a

- What is feature modeling? When is it needed?
- How can we specify valid combinations of features?
- What is a complete, partial, valid, invalid configuration?
- What are (dis-)advantages of natural language, configuration map, and feature models?
- What is the graphical syntax and semantics of feature models?
- Give an example feature model!

## Lecture 4b

- What representations of feature models are available? Are they equivalent?
- How to represent feature models textually?
- What is UVL (used for)?
- How to identify whether a configuration is valid?
- How to translate feature model into a propositional formula?
- What are DIMACS and KConfig (used for)?
- Would you recommend Excel for feature model? Why (not)?

## Lecture 4c

- Why can configuration become challenging?
- How can we identify problems with feature models and configurations?
- How can feature models by analyzed? What analyses are available?
- What solvers can be used to analyze feature models?
- What is the difference between SAT, #SAT, and ALLSAT?
- Why are solvers useful when creating configurations?