Generating Feature Models with UVL's Full Expressiveness

Chico Sundermann
University of Ulm
Germany

Tobias Heß
University of Ulm
Germany

Rahel Sundermann
University of Ulm
Germany

Elias Kuiter
University of Magdeburg
Germany

Sebastian Krieter
Paderborn University
Germany

Thomas Thüm
Paderborn University
Germany

ABSTRACT

The Universal Variability Language (UVL) is a textual format for specifying feature models. UVL has optional language levels (i.e., extensions) that add more expressive functionality to the base language, such as numerical constraints over attributes. However, those levels have been a rather recent addition. Also, most other established formats only support a subset of UVL's expressiveness. Consequently, there are currently only very few feature models available that use the more sophisticated language levels of UVL. The lack of such feature models hinders research and tool development that targets more expressive feature models. In particular, this makes it difficult to develop efficient reasoning engines or conversions between the different levels. In this work, we present UVLGenerator, a first prototype for generating UVL models that make use of all available language levels. The generator supports configuring various structural properties that, inter alia, control which language levels are used.

CCS CONCEPTS

• **Software and its engineering → Software product lines**.

KEYWORDS

feature modeling, UVL, feature-model generator, benchmark

ACM Reference Format:
Chico Sundermann, Tobias Heß, Rahel Sundermann, Elias Kuiter, Sebastian Krieter, and Thomas Thüm. 2024. Generating Feature Models with UVL's Full Expressiveness. In *28th ACM International Systems and Software Product Line Conference (SPLC '24), September 2–6, 2024, Dommeldange, Luxembourg*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3646548.3676602

1 INTRODUCTION

The Universal Variability Language (UVL) is a textual notation for feature models [33]. Since the proposal of the first version in 2021 [33], UVL has been repeatedly considered in research and it has already been integrated in several tools for variability modeling [10–12, 14, 15, 17, 20, 26–28, 34].

UVL has several *language levels* that extend the base language with more expressive constructs [35]. For example, the extending levels support numerical constraints over attributes as well as non-Boolean (i.e., numeric or string) features [35]. In general, when designing a variability language, there is a tradeoff between the simplicity of the language and its expressiveness. A language should be easy to learn and understand but also be able to specify regular user applications. The language levels in UVL aim to reduce the drawbacks of this tradeoff. In particular, the base language should be easy to learn for modelers and reduce effort to tool integration, while the higher language levels add more sophisticated constructs for more expressiveness [12, 27].

Independent of their format, the availability of feature models with constructs from more expressive language levels is currently limited [26, 28, 33]. In particular, the vast majority of available UVL models only uses the base language [28, 33]. Furthermore, many established formats, such as FeatureIDE [21] or SXFM [22] do not support constructs of the higher language levels, such as typed features or numerical constraints. Consequently, available feature models from other formats generally also do not contain the more expressive constructs. Transforming available feature models, thus, does not produce UVL models with higher language levels either.

The limited availability of feature models with higher language levels hampers research on those language constructs. For instance, performing automated analyses on more expressive UVL language levels requires according reasoning engines (e.g., SMT [8]). However, testing and optimizing potential solutions for automated reasoning requires UVL instances with the respective levels, which are currently lacking in the literature. Missing scalable solutions may consequently prevent adoption of such levels in practice.

One way to acquire feature models is by generating them randomly. While random instances are not always representative for real instances [2, 16, 19], structural properties (e.g., number of features) and included language levels can be controlled. Also, generating can bridge the gap until additional real instances have been published. Available generators [21, 22, 29] can be combined with conversion approaches, such as TraVarT [11], to get UVL models, but their expressiveness is again limited to the original format.

In this work, we propose `UVLGenerator`, a prototype for generating UVL models that cover all available language constructs in UVL. Our generator supports various configuration options to control properties, such as number of features, distribution of feature types, constraint sizes, and attached attributes. The configurations are intended for reproducible generation of feature models. Furthermore, we implemented an SMT-based reasoning engine to ensure that the generated models induce at least one valid configuration, if needed.

Table 1: UVL Language Levels

Level	Included Constructs
Boolean	Boolean features
	Parent-child groups
	Boolean CTC
	Attributes
Boolean *Group Cardinality*	Group cardinality [n..m]
Arithmetic	Numeric constraints over attributes
Arithmetic *Feature Cardinality*	Feature cardinality [n..m]
Arithmetic *Aggregates*	avg(attribute), sum(attribute)
Type	Integer, Real, String features
	Numeric constraints over features
Type *String Constraints*	String comparison
	len(string)

2 THE UNIVERSAL VARIABILITY LANGUAGE

In this section, we give a short introduction on the Universal Variability Language (UVL). Table 1 shows the different language constructs in UVL, grouped by their respective language level. The levels are separated in major (e.g., **Boolean**) and minor (e.g., *Aggregates*) language levels which encapsulate certain constructs. Minor levels always belong to a major level. Using a minor language level for a UVL model always required to include its major level. Furthermore, using a higher major level requires to include the lower language levels. In Table 1 the levels are sorted in ascending order from top to bottom. For instance, *type* is the highest major level and, thus, mandates to include *arithmetic* and *Boolean*. Listing 1 shows an example UVL model that describes a pizza product line, which contains all language levels. In the following, we shortly explain the different language levels using our running example.

2.1 Boolean Level

The *Boolean* core level is the base language of UVL. Every other level automatically includes the Boolean level. The core level includes *Boolean features*, *parent-child groups*, and *Boolean cross-tree constraints*. The parent-child groups consist of optional (i.e., select any), mandatory (i.e., select all), or (i.e., select at least one), and alternative (i.e., select exactly one). Examples for the groups can be seen in lines 2–13 in our running example Listing 1. Each pizza needs a `Base`, exactly one type of `Sauce`, and may contain one or multiple types of `Cheese`. Boolean cross-tree constraints are Boolean formulas with common operators, such as conjunctions or implications. For instance, `Cheddar` cannot be selected with `Creme Fraiche` (l. 28). Attributes (e.g., the price of a feature) can be attached to features as additional information (e.g., l. 4), but cannot be used in constraints in the Boolean level. UVL supports different types of attributes, such as numbers, strings, Booleans, and a list of nested attributes. As a minor level of the Boolean level, *group cardinality* introduces a parent-child relationship that enables the selection of [n..m] child features. For example, between one and three `Toppings` can be selected (l. 15–19). Most feature models currently available in UVL and other available formats are specified in the Boolean language level [1, 3, 14, 19, 21, 22, 28, 33].

Listing 1: UVL Model Example

```
1   features
2       Pizza {City 'Omashu'}
3           mandatory
4               Base {Price 2.0}
5               Sauce
6                   alternative
7                       Tomato {Price 0.8}
8                       "Creme Fraiche" {Price 1.1}
9           optional
10              Cheese
11                  or
12                      Mozarella {Price 0.4}
13                      Cheddar {Price 0.6}
14              Toppings
15                  [1..3]
16                      Mushrooms {Price 0.5}
17                      Onion {Price 0.3}
18                      Olive {Price 0.3}
19                      Pineapple {Price 0.8}
20          mandatory
21              Integer Size
22              DeliveryAddress cardinality [1..10]
23                  mandatory
24                      String Street
25                      String City
26
27  constraints
28      Cheddar => !"Creme Fraiche"
29      sum(Price) < 5
30      20 < Size & Size < 60
31      City == Pizza.City
32      len(Street) < 100
```

2.2 Arithmetic Level

The *arithmetic* core level introduces predicates over numeric attributes. Here, common arithmetic operators, such as +, -, *, =, and <, are supported. The predicates always need to evaluate to a Boolean value (i.e., true or false) and can be embedded in other cross-tree constraints (e.g., l. 30).

Feature cardinality is a minor level of the arithmetic level and enables selecting a single feature multiple times. For instance, multiple delivery addresses can be configured (l. 22). *Aggregates* are another minor level of the arithmetic level and simplifies the specification of arithmetic constraints with aggregate functions, namely sum and average, over attributes with the same name. For example, the overall price needs to be smaller than 5 (l. 29).

2.3 Type Level

The *type* core level supports *typed* features with the types real, integer, and string. For instance, the `Size` of the pizza is an integer (l. 21) and the `Street` and `City` are strings (l. 24–25). Real and integer features can also be used in the numerical predicates from the arithmetic level. For instance, the `Size` needs to be between 20 and 60 (l. 30). *String constraints* introduce constraints over string values supporting attributes (e.g., `Pizza.City`), features (e.g., `City`), and constants (e.g., `'Omashu'`). Hereby, one can reason about equality of strings and their length. String constants are always enclosed by single parentheses (i.e., `'..'`). In our example, the delivery location needs to be in the same `City` as the pizza place (l. 31) and the length of the `Street` may be at most 100 (l. 32).

3 UVLGENERATOR

Our proposal `UVLGenerator` creates a set of random UVL feature models which meet different requirements that can be configured. Controlling these parameters can also be used to implicitly limit the used language levels in generated feature models. In the following, we describe the key aspects of the generator. For more technical details, we refer to the implementation.[1]

3.1 Parameter Configuration

The parameters in `UVLGenerator` support specifying properties for different parts of the feature model, namely general information, the tree hierarchy, constraints, and attributes. A parameter configuration can be specified as a JSON file and given to the generator. Listing 2 shows an example configuration JSON, which we use to explain the different options in the following.

General Information. The *general* (l. 2–7) information provides (1) information on the created set of feature models, and (2) properties to reproduce the generated set of feature models. For (1), the number of models to generate and whether only satisfiable (i.e., at least one valid configuration) models should be included can be specified. For (2), one can define the seed for the random number generation and the version of the used generator.

Tree Hierarchy. The configuration for the *tree* (l. 8–33) hierarchy controls the properties for generating features and parent-child relationships. For features, the user can specify their number (l. 10), the distribution of feature types (l. 11–16), and the usage of cardinalities (l. 17–21). For specifying the number of features (and similar options), either a number or a range can be given. With the latter, a random number within the range is selected for every feature model. In our example Listing 2, each feature model in the generated set contains 800–1,000 features. The distribution supports selecting percentages with which a feature is of the specified type, which always adds up to exactly one. For feature cardinalities, one can specify numbers or ranges for the upper and lower bound and a probability with which a feature has a cardinality. Also, a maximum tree depth can be specified (l. 23). Then, features which are at the maximum specified depth will not be considered as parent for subsequently generated features. To specify parent-child groups, we also use a distribution between the group types (l. 25–32).

Constraints. For *constraints* (l. 34–44), one can also specify the number of constraints to include. The Extra Constraint Representativeness (ECR) [24] specifies the share of features that appear in cross-tree constraints. The size of constraints can be determined as well as a distribution for the types of constraints. Here, operators and variables are chosen randomly from eligible options.

Attributes. For *attributes* (l. 45–52), we decided not to use a fully random generation, as we expect attributes to be used over several features across the feature model [30]. Aggregate functions also can only be reasonably applied for attributes used across multiple features. Consequently, users need to define specific attributes, their value range, and the probability to attach the attribute to a feature. For instance, in Listing 2, `Price` attributes are attached to features with a 50% chance and can have values between 10 and 1,000.

[1]https://github.com/SoftVarE-Group/uvlgenerator

Listing 2: Configuration Example

```
1  {
2      "general" : {
3          "numberModels" : 100,
4          "seed" : 42,
5          "ensureSAT" : true,
6          "generatorVersion" : "0.1"
7      },
8      "tree" : {
9          "features" : {
10             "number" : [800,1000],
11             "distribution" : {
12                 "Boolean" : 0.7,
13                 "integer" : 0.2,
14                 "real" : 0.1,
15                 "string" : 0
16             }
17             "cardinality" : {
18                 "min" : [1,2],
19                 "max" : 5,
20                 "attachProbability" : 0.05
21             }
22         },
23         "maxTreeDepth" : 5,
24         "groups" : {
25             "distribution": {
26                 "optional" : 0.1,
27                 "mandatory" : 0.2,
28                 "alternative" : 0.5,
29                 "or" : 0.2,
30                 "groupCardinality" : 0
31             }
32         }
33     },
34     "constraints" : {
35         "number" : [200,300],
36         "ecr" : 0.8,
37         "variablesPerConstraint" : [2,15],
38         "distribution" : {
39             "Boolean" : 0.9,
40             "numeric" : 0.07,
41             "aggregate" : 0.03,
42             "string" : 0
43         }
44     },
45     "attributes" : [
46         {
47             "name" : "Price",
48             "value" : [10,1000],
49             "attachProbability" : 0.5,
50             "useInConstraints" : true
51         }
52     ]
53 }
```

3.2 Reasoning

To ensure satisfiability of the generated models, we use SMT solving [8]. Hereby, we convert every constraint imposed by the hierarchy or the cross-tree constraints to a semantically equivalent SMT formula. Then, we create a formula equivalent to the entire feature model by creating a conjunction over these sub-formulas.

Table 2 shows the mapping between UVL constructs and SMT formulas, which extends common translation proposals for feature models [3, 4, 6] with translations for the higher language levels of

UVL. The root is mandatory for UVL feature models. As the inclusion of a feature always mandates the inclusion of its parent, we add an implication for every feature to its parent feature. Feature cardinalities [n..m] are not trivial to translate since a feature with a cardinality may have a subtree [6]. Hence, for conversion into a formula, we create m clones of the subtrees and then create the constraints for a group cardinality over their respective root features. The numeric and Boolean cross-tree constraints use common operators and can be equivalently specified in SMT. Here, we always multiply the Boolean variable of the corresponding feature (0 or 1) with the attribute value to discard attributes of unselected features. For the implementation,[2] we use the JavaSMT [18] interface with the Z3 [7] solver as a backend. The project can also be used as simple standalone reasoner for UVL.

Table 2: Mapping UVL To SMT

UVL Construct	SMT Formula
Root r	r
Child c of p	$c \Rightarrow p$
Mandatory c of p	$p \Rightarrow c$
or $o_1, .., o_n$ of p	$p \Rightarrow (o_1 \vee .. \vee o_n)$
alternative $a_1, .., a_n$ of p	$p \Rightarrow ((a_1 \vee .. \vee a_n) \wedge \text{atMost}(a_1, .., a_n))$
Group [n..m] $c_1, .., c_l$ of p	$p \Rightarrow (\text{atLeast}(n, c_1, .., c_l)$ $\wedge \text{atMost}(m, c_1, .., c_l))$
Feature [n..m]	Expand clones and group [n..m]
Boolean constraints	Boolean formula
Numeric constraints	Numeric formula
Aggregate sum(att)	f1 * f1.att + f2 * f2.att + .. + fn * fn.att
Aggregate avg(att)	sum(att) / (f1 + f2 + .. + fn) 0 if none of f1, .., fn is selected
String equality str1, str2	str1 == str2
String length len(const)	Numeric constants
String length len(v)	Numeric variable v-len

4 RELATED WORK

Feature-Model Generators. BeTTy [29] is a feature-model generator that supports the constructs from the UVL Boolean level. As our proposal, BeTTy is highly customizable and is known to produce more complex and higher constrained feature models compared to previous generators, such as the ad-hoc generator of Mendonça et al. [24] or the built-in generator of S.P.L.O.T. [22]. FeatureIDE [21] also comes with an integrated feature-model generator [36]. All the listed feature-model generators [21, 24, 29], only support language constructs from the UVL Boolean level. Derks et al. presented `vpbench`, a modular generator that generates an evolution history of a configurable system [9]. While their generator also generates solution-space artifacts together with additional metadata, the generation of the feature model is limited to the Boolean level. Finally, Galindo et al. proposed an LLM-based feature model generator trained on a set of feature models [13]. They found that their approach closely mirrored the input models in terms of structure and generated models with sufficient complexity. Again, their generator only supports the Boolean level. However, it is feasible that their approach could also be used for other language levels, although it is questionable if meaningful models can be generated in other language levels without solver guidance.

[2]https://github.com/SoftVarE-Group/uvl-smt

Feature-Model Collections. Various publications provide a set of feature models [5, 19, 22, 28, 32, 33]. While not all of those are available in UVL, they can be translated to UVL using converters such as TraVart [11] or FeatureIDE [21]. Many of those feature models represent real instances from industries. However, none of these feature models contain all language constructs of UVL. In particular, DIMACS [5] can only represent a subset of UVL core language, FeatureIDE models [19] support the same constructs as the UVL core language, and SXFM [22] supports the Boolean level including group cardinalities. The available UVL models [28, 33] only use the Boolean core level. Consequently, many UVL constructs do not appear at all in available feature models.

5 CONCLUSION

Currently available feature models in UVL [28, 33] and other formats [19, 21, 22] are generally limited to a small subset of the language levels available in UVL. Hence, research on the missing language levels is hampered. In this work, we provide a first proposal for generating UVL models that supports the creation of all current language constructs. We expect that the generator facilitates research that targets higher language levels, such as reasoning engines for UVL or conversion strategies between levels [35]. Still, we expect that our prototype generator can be further extended to cover more specific use cases. We consider various future research directions for the generation that target practice relevance, scalability, and covering more use cases.

Realistic Feature Models. Generated instances often behave vastly differently than real ones [2]. Currently, our generator targets fully randomized UVL models. In the future, it could be valuable to generate models that follow properties from real-world instances.

Scalability. With the generator, we can create feature models that induce computationally complex problems. For instance, many constructs cannot be trivially represented as Boolean formulas and, thus, SAT solvers, which are widely used and very fast for analyzing many feature models [23] cannot be easily applied. In particular, the added complexity can vastly increase solving times [25, 31]. When mandating satisfiable feature models, the generation process needs to solve satisfiability during the creation. For large feature models, especially with complex constraints, the generation may induce large runtimes or not scale at all. In the future, we plan to (1) empirically evaluate the scalability of the generator and (2) further optimize the generation process to use less and potentially simplified SMT computations.

Community Discussion. The current requirements are mostly motivated by experiences of the authors. Hence, their scope may be rather limited. We expect that discussing different use cases, requirements, and properties that should be configurable with the community will lead to relevant future research and valuable extensions for `UVLGenerator`. Furthermore, it may be valuable to empirically evaluate how well the current or adapted future versions cover the requirements of the community.

REFERENCES

[1] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B. France. 2013. Familiar: A Domain-Specific Language for Large Scale Management of Feature Models. *Science of Computer Programming (SCP)* 78, 6 (2013), 657–681.

[2] Carlos Ansótegui, María Luisa Bonet, and Jordi Levy. 2009. On the Structure of Industrial SAT Instances. In *Proc. Int'l Conf. on Principles and Practice of Constraint Programming (CP)*. Springer, 127–141.

[3] Don Batory. 2005. Feature Models, Grammars, and Propositional Formulas. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. Springer, 7–20.

[4] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. 2010. Automated Analysis of Feature Models 20 Years Later: A Literature Review. *Information Systems* 35, 6 (2010), 615–708.

[5] Thorsten Berger, Steven She, Rafael Lotufo, Andrzej Wąsowski, and Krzysztof Czarnecki. 2013. A Study of Variability Models and Languages in the Systems Software Domain. *IEEE Trans. on Software Engineering (TSE)* 39, 12 (2013), 1611–1640.

[6] Krzysztof Czarnecki and Chang Hwan Peter Kim. 2005. Cardinality-Based Feature Modeling and Constraints: A Progress Report. In *Proc. Int'l Workshop on Software Factories (SF)*. 16–20.

[7] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Proc. Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer, 337–340.

[8] Leonardo De Moura and Nikolaj Bjørner. 2011. Satisfiability Modulo Theories: Introduction and Applications. *Comm. ACM* 54, 9 (2011), 69–77.

[9] Christoph Derks, Daniel Strüber, and Thorsten Berger. 2023. A Benchmark Generator Framework for Evolving Variant-Rich Software. *J. Systems and Software (JSS)* 203, Article 111736 (2023).

[10] Hafiyyan Sayyid Fadhlillah, Kevin Feichtinger, Philipp Bauer, Elene Kutsia, and Rick Rabiser. 2022. V4rdiac: Tooling for Multidisciplinary Delta-Oriented Variability Management in Cyber-Physical Production Systems. In *Proc. Int'l Workshop on Languages for Modelling Variability (MODEVAR) (Proc. Int'l Systems and Software Product Line Conf. (SPLC))*. ACM, 34–37.

[11] Kevin Feichtinger, Johann Stöbich, Dario Romano, and Rick Rabiser. 2021. TRAVART: An Approach for Transforming Variability Models. In *Proc. Int'l Working Conf. on Variability Modelling of Software-Intensive Systems (VaMoS)*. ACM, Article 8, 10 pages.

[12] Kevin Feichtinger, Chico Sundermann, Thomas Thüm, and Rick Rabiser. 2022. It's Your Loss: Classifying Information Loss During Variability Model Roundtrip Transformations. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 67–78.

[13] José A. Galindo, Antonio J. Dominguez, Jules White, and David Benavides. 2023. Large Language Models to Generate Meaningful Feature Model Instances. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 15–26.

[14] José A. Galindo, José Miguel Horcas, Alexander Felfernig, David Fernández-Amorós, and David Benavides. 2023. FLAMA: A Collaborative Effort to Build a New Framework for the Automated Analysis of Feature Models. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 16–19.

[15] Tobias Heß, Tobias Müller, Chico Sundermann, and Thomas Thüm. 2022. ddueruem: A Wrapper for Feature-Model Analysis Tools. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 54–57.

[16] Tobias Heß, Chico Sundermann, and Thomas Thüm. 2021. On the Scalability of Building Binary Decision Diagrams for Current Feature Models. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 131–135.

[17] Jose M. Horcas, Jose A. Galindo, Mónica Pinto, Lidia Fuentes, and David Benavides. 2022. FM Fact Label: A Configurable and Interactive Visualization of Feature Model Characterizations. In *Proc. Int'l Workshop on Languages for Modelling Variability (MODEVAR) (Proc. Int'l Systems and Software Product Line Conf. (SPLC))*. ACM, 42–45.

[18] Egor George Karpenkov, Karlheinz Friedberger, and Dirk Beyer. 2016. JavaSMT: A Unified Interface for SMT Solvers in Java. In *Proc. IFIP Working Conf. on Verified Software: Theories, Tools, Experiments (VSTTE)*. Springer, 139–148.

[19] Alexander Knüppel, Thomas Thüm, Stephan Mennicke, Jens Meinicke, and Ina Schaefer. 2017. Is There a Mismatch Between Real-World Feature Models and Product-Line Research?. In *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*. ACM, 291–302.

[20] Jacob Loth, Chico Sundermann, Tobias Schrull, Thilo Brugger, Felix Rieg, and Thomas Thüm. 2023. UVLS: A Language Server Protocol for UVL. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 43–46.

[21] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. 2017. *Mastering Software Variability With FeatureIDE*. Springer.

[22] Marcílio Mendonça, Moises Branco, and Donald Cowan. 2009. S.P.L.O.T.: Software Product Lines Online Tools. In *Proc. Conf. on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*. ACM, 761–762.

[23] Marcílio Mendonça, Andrzej Wąsowski, and Krzysztof Czarnecki. 2009. SAT-Based Analysis of Feature Models Is Easy. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. Software Engineering Institute, 231–240.

[24] Marcílio Mendonça, Andrzej Wąsowski, Krzysztof Czarnecki, and Donald Cowan. 2008. Efficient Compilation Techniques for Large Scale Feature Models. In *Proc. Int'l Conf. on Generative Programming and Component Engineering (GPCE)*. ACM, 13–22.

[25] Daniel-Jesus Munoz, Jeho Oh, Mónica Pinto, Lidia Fuentes, and Don Batory. 2019. Uniform Random Sampling Product Configurations of Feature Models That Have Numerical Features. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 289–301.

[26] Daniel-Jesus Munoz, Mónica Pinto, Lidia Fuentes, and Don Batory. 2023. Transforming Numerical Feature Models into Propositional Formulas and the Universal Variability Language. *J. Systems and Software (JSS)* 204, Article 111770 (2023).

[27] Dario Romano, Kevin Feichtinger, Danilo Beuche, Uwe Ryssel, and Rick Rabiser. 2022. Bridging the Gap Between Academia and Industry: Transforming the Universal Variability Language to Pure::Variants and Back. In *Proc. Int'l Workshop on Languages for Modelling Variability (MODEVAR) (Proc. Int'l Systems and Software Product Line Conf. (SPLC))*. ACM, 123–131.

[28] David Romero-Organvídez, José A Galindo, Chico Sundermann, Jose-Miguel Horcas, and David Benavides. 2024. UVLHub: A Feature Model Data Repository Using UVL and Open Science Principles. *J. Systems and Software (JSS)* (2024). To appear.

[29] Sergio Segura, José A. Galindo, David Benavides, José A. Parejo, and Antonio Ruiz-Cortés. 2012. BeTTy: Benchmarking and Testing on the Automated Analysis of Feature Models. In *Proc. Int'l Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*. ACM, 63–71.

[30] Norbert Siegmund, Stefan Sobernig, and Sven Apel. 2017. Attributed Variability Models: Outside the Comfort Zone. In *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*. ACM, 268–278.

[31] Joshua Sprey, Chico Sundermann, Sebastian Krieter, Michael Nieke, Jacopo Mauro, Thomas Thüm, and Ina Schaefer. 2020. SMT-Based Variability Analyses in FeatureIDE. In *Proc. Int'l Working Conf. on Variability Modelling of Software-Intensive Systems (VaMoS)*. ACM, Article 6, 9 pages.

[32] Chico Sundermann, Vincenzo Francesco Brancaccio, Elias Kuiter, Sebastian Krieter, Tobias Heß, and Thomas Thüm. 2024. Collecting Feature Models from the Literature: A Comprehensive Dataset for Benchmarking. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM. To appear.

[33] Chico Sundermann, Kevin Feichtinger, Dominik Engelhardt, Rick Rabiser, and Thomas Thüm. 2021. Yet Another Textual Variability Language? A Community Effort Towards a Unified Language. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 136–147.

[34] Chico Sundermann, Tobias Heß, Dominik Engelhardt, Rahel Arens, Johannes Herschel, Kevin Jedelhauser, Benedikt Jutz, Sebastian Krieter, and Ina Schaefer. 2021. Integration of UVL in FeatureIDE. In *Proc. Int'l Workshop on Languages for Modelling Variability (MODEVAR)*. ACM, 73–79.

[35] Chico Sundermann, Stefan Vill, Thomas Thüm, Kevin Feichtinger, Prankur Agarwal, Rick Rabiser, José A. Galindo, and David Benavides. 2023. UVLParser: Extending UVL With Language Levels and Conversion Strategies. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 39–42.

[36] Thomas Thüm, Don Batory, and Christian Kästner. 2009. Reasoning About Edits to Feature Models. In *Proc. Int'l Conf. on Software Engineering (ICSE)*. IEEE, 254–264.