Formal Validation and Model Synthesis for Domain-specific Languages by Logic Solvers

Oszkár Semeráth

Budapest University of Technology and Economics
Department of Measurement and Information Systems
MTA-BME Lendület Research Group on Cyber-Physical Systems
semerath@mit.bme.hu

Abstract

Despite the wide range of existing tool support, constructing a design environment for a complex domain-specific language (DSL) is still a tedious task due to the large number of derived features and well-formedness constraints complementing the domain. Additionally, an advanced design environment uses view transformation techniques to highlight different relevant aspects of the system. As any software, modeling tools are not free from errors. For complex domains, derived features and constraints can easily be formalized incorrectly resulting in inconsistent, incomplete or ambiguous DSL specification, or inconsistent view models. Moreover, errors in the modeling environment injects errors to the generated code and invalidates the results of any verification process. Therefore it is important to ensure the correctness (i.e. consistency and unambiguity) of the modeling languages. My research focuses on validation of modeling environments by (i) proving the correctness of Domain Specific Languages, (ii) automatically generating or extending models for DSLs and view models, and (iii) developing efficient decision procedures for DSLs.

Keywords language validation, derived features, partial snapshots, model queries, logic solvers

1. Validation of Domain-specific Languages

The design of integrated development environments for complex domain-specific languages (DSL) is still a challenging task nowadays. Advanced environments such as Xtext, or Sirius built on top of model management frameworks such as Eclipse Modeling Framework (EMF) significantly improve productivity by automating the production of rich editor features (e.g. syntax highlighting, autocompletion, etc.) or by task-specific view models [14] to enhance modeling for domain experts. Furthermore, there is efficient tool support for validating well-formedness constraints and design rules over large model instances of the DSL using tools like Eclipse OCL [45] or VIATRA Query [5]. As a result, Eclipse-based IDEs are widely used in the industry in various domains including business modeling, avionics or automotive.

However, in case of complex, standardized industrial domains (like ARINC 653 for avionics or AUTOSAR in automotive), the sheer complexity of the DSL and the models is a major challenge itself. (1) First, there are hundreds of well-formedness constraints and design rules defined by those standards, and due to the lack of validation, there is no guarantee for their consistency or unambiguity. (2) Moreover, domain metamodels are frequently extended by derived features, which serve as automatically calculated shortcuts for accessing or navigating models in a more straightforward way. In many practical cases, these features are not defined by the underlying standards but introduced during the construction of the DSL environment for efficiency reasons. Anyhow, the specification of derived features can also be inconsistent, ambiguous or incomplete. And finally (3) view models are key concept in domain-specific modeling tools to provide task-specific focus by creating a model which highlights only some relevant aspects of the system. However, views can also be inconsistent with the model, or represent unfeasible requirements. In general, mathematical precise validation of DSL specifications themselves has been attempted by only few approaches so far [24], and even these approaches lack a systematic validation process.

As model-driven tools are frequently used in critical systems design to detect conceptual flaws of the system model early in the development process to decrease verification and validation (V&V) costs, those tools should be validated with the same level of scrutiny as the underlying system as part of a software tool qualification process in order to provide trust in their output. Therefore software tool qualification raises

several challenges for building trusted DSL tools for a specific domain. First, the consistency and unambiguity of the DSL specification have to be ensured. Secondly, the developement of the modeling environment requires systematic testing methods, which is based on the generation of valid (or purposely faulty) models. And finally, the complete analysis of view models requires tracing of view model changes back to source model changes.

Therefore, based on the previous challenges, I have identified the two main research questions:

RQ1 Validation of Domain Specific Languages: *How to prove the consistency and completeness of DSL specs?*

RQ2 Generation of well-formed instance models: *How to generate valid instance models of a complex DSL or View models?*

2. Preliminaries

The precise definition of complex domain-specific languages (DSL) necessitates a combination of different specification techniques. *Metamodels* define the main concepts, relations and attributes of the target domain to specify the basic structure of the models. A metamodel can be represented by a theorem $META$, and consequently, a logic structure M can specify an instance model. Therefore, $M \models META$ denotes if a model satisfies all structural constraints of the metamodel (e.g. multiplicity or containment).

To create an advanced modeling environment, a DSL is typically augmented with *well-formedness* constraints (WF), which capture additional restrictions any well-formed instance model needs to respect (denoted with $M \models WF$). Such constraints can be defined by model queries (often captured by graph patterns) [6] or as OCL invariants [30]. Furthermore, the metamodel can also be enhanced with *derived features* (DF), i.e. attributes and relations calculated from core model elements during model use, which can be also specified using by graph patterns [33]. If the attributes and references have the correct values then it is denoted by $M \models DF$. The axiom set DSL of a domain-specific language is summarized as $DSL = META \wedge WF \wedge DF$, and a valid instance model M satisfies those constraints: $M \models DSL$.

In a domain-specific modeling tool, the underlying domain model is presented to the engineers in different views ($VIEW$). These views are populated from the source model (by abstraction). One source model may populate multiple view models. A view $VIEW$ is derived from the source model M by a unidirectional forward transformation $view$ [14]. An instance model M is consistent with a view model $VIEW$ (denoted by $M \models VIEW$) if forward transformation creates the same view $VIEW = view(M)$. Deriving different views from an instance model is an efficient way to highlight selected properties. However, calculating models for views requires logic reasoning.

Reasoning over a metamodel or view models has two main challenges, which hinders the efficient analysis of domain-specific languages:

Ch1 *Proving a property P over a language $DSL \models P$ is undecidable in general.*

Ch2 *Existing logic solvers (SAT/SMT) fail to derive instance models for complex domain-specific languages with views to create example or counter-example models.*

3. Related Work

Logic Solver Approaches There are several approaches and tools aiming to validate models enriched with OCL constraints [19] relying upon different logic formalisms such as constraint logic programming [10, 11], SAT-based model finders (like Alloy) [3, 9, 27, 43, 44], first-order logic [4], constructive query containment [32] or higher-order logic [8, 20]. Some of these approaches (like e.g. [9, 11, 27, 43]) offer bounded validation (where the search space needs to be restricted explicitly) in order to execute the validation and thus results can only be considered within the given scope, others (like [8]) allow unbounded verification (which normally results in increased level of interaction and decidability issues).

Uncertain Models Partial models are also similar to uncertain models, which offer a rich specification language [34] amenable to analysis. Uncertain models provide a more expressive language compared to partial snapshots but without handling additional WF constraints. Such models document semantic variation points generically by annotations on a regular instance model, which are gradually resolved during the generation of concrete models. An uncertain model is more complex (or informative) than a concrete one, thus an a priori upper bound exists for the derivation, which is not an assumption in our case.

Potential concrete models compliant with an uncertain model can synthesized by the Alloy Analyzer [35], or refined by graph transformation rules [36]. Each concrete model is derived in a single step, thus their approach is not iterative like ours. Scalability analysis is omitted from the respective papers, but refinement of uncertain models is always decidable.

View Models Using logic solvers for generating possible source and target candidates is common part of several approaches. [13] uses Answer Set Programming, [12] maps the problem to Mixed Integer Linear Programming. [28] uses Alloy to generate change operations on the source model which leads to a modified source model which is (i) well-formed and (i) consistent with the changed target model. [18] and [17] converts the transformation to Alloy similarly, but do not handle WF constraints of the source model, and changes the whole source model.

Rule-based Instance Generators A different class of model generators relies on rule-based synthesis driven by random-

ized, statistical or metamodel coverage information for testing purposes [7, 16]. Some approaches support the calculation of effective metamodels [42], but partial snapshots are excluded from input specifications. Moreover, WF constraints are restricted to local constraints evaluated on individual objects while global constraints of a DSL are not supported. On the positive side, these approaches guarantee the diversity of models and scale well in practice.

Iterative approaches. An iterative approach is proposed (specifically for allocation problems) in [26] based on Formula. Models are generated in two steps to increase diversity of results. First, non-isomorphic submodels are created only from an effective metamodel fragment. In the second step the algorithm completes the different submodels according to the full model, but constraints are only checked at the very final stage.An iterative, counter-example guided synthesis is proposed for higher-order logic formulae in [29], but the size of derived models is fixed.

4. Formal Validation of Domain-Specific Languages by Logic Solvers

Approach Addressing research problem **RP 1.**, we created a novel approach presented in [38, 39] to analyze the DSL specification of *modeling tools* by mapping them into first order logic (FOL) formulae that can be processed by advanced *reasoners* such as *SMT solvers* (Z3) or *SAT solvers* (Alloy, see Figure 1). The outcome of a reasoning problem is either satisfiable or unsatisfiable. If the problem is satisfiable, the solver constructs an output (or completed) model (which is interpreted as witness or counterexample depending on the validation task), while an unsatisfiable result means a contradiction. Because certain validation tasks are undecidable in FOL it is also possible that validation terminates with an unknown answer or a timeout.

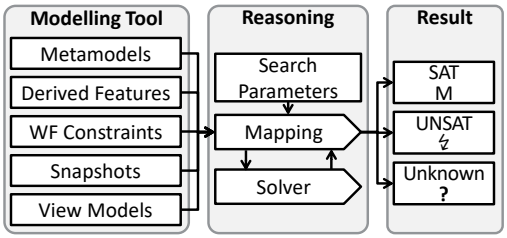

Figure 1. Functional overview of the approach

We carry out a wide range of validation tasks by automated theorem proving based on this formalization to prove different properties of a DSL. To decrease the development time and cost of DSL tools, we aim to detect design flaws in the early phase of DSL development by highlighting validation problems to the developer directly in the tool itself by back-annotating analysis results. Linking the independent reasoning tool to the modeling tool allows the DSL developer to make mathematically precise deductions over the developed languages.

Contributions My first contribution aims the formalization of graph patterns used in the specification of DSLs to first order logic expression (FOL).

Con1.1 Mapping of graph patterns to effectively propositional logic: *I introduced a technique to transform WF and DF rules captured by graph patterns to a decidable fragment of FOL [31] by using over- and underapproximation techniques. [38]*

With model queries, meta- and optionally instance models can be translated to logic expression in order to analyze their consistency.

Con1.2 Uniform analysis of DSL specification: *I introduced a technique that uniformly translates DLS elements to FOL to analyze the consistency of the whole DSL specification, which includes metamodels, instance models, well-formedness (OCL or graph pattern) and derived features.*

Based on the consistency analysis technique, several validation rules are specified, which can be checked by the underlying logic solver.

Con1.3 Identification of context dependent DSL validation criteria: *I defined completeness and unambiguity properties of derived features, subsumption and equivalence relations of well-formedness constraints, derived features and instance models. These properties can be checked on the full DSL, or on a specific fragment of it.*

In order to systematically carry out the validation process for the whole DSL, we propose a validation workflow, which consequently investigates each language feature, and can be used without any theorem-proving skills.

Con1.4 Validation workflow for DSL specifications: *We recommended a validation process for the DSL, which systematically checks the language properties, and in case of inconsistencies, helps the developer to correct the DSL specification (or refine the validation context) by showing representative counterexamples to the assumed properties.*

Our technique is successfully applied on a case study taken from the avionics domains.

Con1.5 Application: Validation of an avionics DSL: *I carried out the validation of the functional architecture modeling language of of avionics systems, developed in Trans-IMA project [22].*

Added Value The main added value of approach is to cover rich DSL constructs such as derived features and well-formedness constraints captured in declarative languages such as graph patterns and OCL invariants. While other approaches use bounded verification or simply ignore unsupported features, I proposed approximations to transform language features into a decidable fragment of first-order logic (called effectively propositional logic), and to handle language features which cannot be represented in FOL.

Therefore, the correctness of a DSL can be proved using our method, while others only can detect errors.

Our approach is supported by a prototype tool integrated into Eclipse, which takes EMF metamodels, instance models, EMF-IncQuery graph patterns and OCL constraints as input to carry out DSL validation. As a technological difference, our tool is compliant with standard Eclipse based technologies, while Formula and Alloy use their own modeling language. When an output model is derived as a witness or counterexample, this model is back-annotated to the DSL tool itself so that language engineers could observe the source of the problem in their custom language. Therefore it does not require additional theorem proving skills.

5. Iterative and incremental model generation by logic solvers

As a side effect, our DSL validation framework of section 4 can also *generate prototypical well-formed instance models* for a DSL, which can be used for synthesizing test cases, for instance. As the metamodel of an industrial DSL may contain hundreds of model elements, any realistic instance model should be of similar size. Unfortunately, this cannot currently be achieved by a single direct call to the underlying solver [23, 24, 39], thus existing logic based model generators *fail to scale*. Furthermore, logic solvers tend to retrieve simple *unrealistic models* [25] consisting of unconnected islands of model fragments and many isolated nodes, which is problematic in an industrial setting.

Approach Addressing the model generation challenge of **RP2**, we propose an iterative process for incrementally generating valid instance models for DSLs with views by calling existing model generators as black-box components, and using various abstractions and approximations to improve overall scalability. Therefore, as seen in Figure 2, instance models can be incrementally generated in multiple steps as a sequence of extending partial models $M_1, \ldots, M_n$, where each step is an independent call to the underlying solver. The main idea behind this approach is that the solver can be guided by smaller logic problems, where only the newly created elements have to be added (marked by Δ).

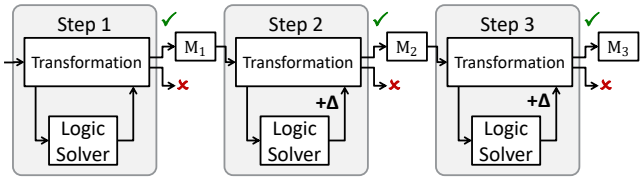

Figure 2. Overview of iterative model generation

Contributions First, incremental model generation requires the decomposition of the problem into smaller tasks, which can be solved sequentially, each step in incrementing the synthesized model while trying to keep them well-formed. The decomposition is enabled by two techniques: *metamodel pruning* [16, 42] to reduce the types in a problem, and *partial models* [15] to extend the model in multiple steps.

Con2.1 Decomposition of Model Generation Problems: *I specified a decomposition technique for instance models in order to specify a partial solutions for model generation using partial modeling, and metamodel pruning. [41]*

When removing certain metamodel elements by pruning, or creating only partial models, related well-formedness constraints need special care. Simple removal of the constraints significantly increase the rate of false positives in a later phase of model generation to such an extent that no intermediate models can be extended to a valid final model. Based on some first-order logic representation of the constraints (derived e.g. in accordance with [39]), we propose to maintain approximated versions of constraint sets during metamodel pruning.

Con2.2 Approximation of Well-formedness Constraints: *I specified an overapproximation technique for well-formedness constraints on partial models with pruned metamodels. [37, 41]*

Using approximated (simplified) model generation steps an incremental model generation process is created which iteratively calls black-box logic solvers to guarantee well-formedness by feeding instance models obtained in a previous step as partial solution to a subsequent phase. In each step, the number of types, elements and constraints is strongly limited by metamodel pruning, partial modeling and constraint approximation techniques. Our experiments show that significantly larger model instances (up to 250 objects instead of 20 using Alloy [40]) can be generated with the same solvers using such an incremental approach especially in the presence of complex well-formedness constraints.

Con2.3 Incremental Model Generation: *I proposed an iterative workflow to incrementally generate instance models of increasing complexity. [41]*

View models are extensively used in model-driven engineering to highlight different relevant properties of a system. However, the generation of valid instance models M for view models $VIEW$ (such as $M \models VIEW \wedge DSL$) remains a challenge. The generation problem can be initiated from an existing source and view models by breaking the consistency with a change in some of the view models. In this case, an existing source model and existing unchanged parts of the view model can be used to reduce the difficulty of the logic problem further. We illustrated our change propagation technique on a healthcare example [1].

Con2.4 Incremental Model Synthesis for Bidirectional Transformations of View Models: *I transformed query-based view specification into logic formulae to automatically synthesize possible source model changes consistent to a view model change.*

Finally, the incremental generation technique is applied in one of our research project:

Con2.4 Application: Generation of Context Models: *I successfully applied my technique in test context generation for autonomous and cooperative robot systems for R3COP ARTEMIS project [2]*

Added value The validation of DSL tools frequently necessitates the synthesis of well-formed *and* nontrivial instance models, which satisfy the language specification.

View model is a convenient and precise way to be used as specification language for model generation. For generating source model candidates for views, our approach takes the whole DSL specification into the account including the interaction of the metamodel, the WF constraints and other view models.

6. Towards a Solver for DSL Models

The technique in section 5 is able to automatically creates valid instance models, but the sequence might lead to a dead-end if an intermediate solution can not be completed. By automatically backtracking unsatisfiable partial models a (semi-)automated process is able to explore the possible solutions for a model generation process. This section presents our newest model generation approach, which is currently under developement.

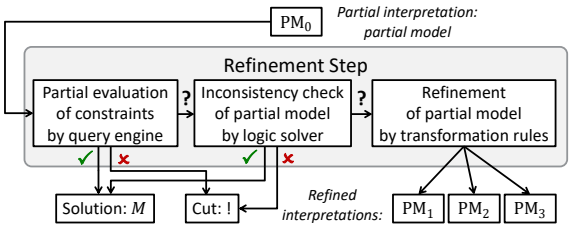

Figure 3. Model refinement strategy

Approach Figure 3 illustrates a model refinement step combining the advantages of multiple model generation techniques. As input, the step gets a partial interpretation of an instance model, which is represented as a partial model (as opposed to partial interpretations or proofs in solvers). The partial model is refined in three steps:

1. A *model query engine* creates a *partial evaluation*[37] on the partial model to determine if the partial solution satisfies all constraints (therefore it is a valid model), violates a constraint (therefore it can not be finished).
2. Then a *solver* tries to prove that the logic problem cannot be solved with this partial solution, or to create a valid solution as a counterexample.
3. If the solver fails, the solution is refined by *partial model refinement transformations* [7, 34] which automatically produces several partial solution candidates.

Therefore, constraints can be evaluated with an efficient query engine, and the matches can be used to simplify the logic problem by the removal of satisfied constraints on the partial model. Solvers are efficient detecting inconsistencies in a specification, but often fails to create models. Our solution creates models by trying to add elements. Each refinement step may produce several partial model candidates, create a valid solution or cut off branches by proving that the partial solution cannot be finished. This requires the management of a search space, which can be efficiently handled by advanced *design space exploration techniques* [21]

In summary, a search based model refinement with integrated solvers is able to prove inconsistencies in a DSL, and expected to create models more efficiently with model-generation specific heuristics.

Acknowledgments

I would like to thank my advisor, Dániel Varró for his support during my research, and Ágnes Barta and Ákos Horváth for the joint work.

References

[1] CONCERTO ARTEMIS project. `concerto-project.org/`.

[2] R3Cop (Resilient Reasoning Robotic Co-operative Systems). ARTEMIS project n°100233, `http://www.r3-cop.eu/`.

[3] K. Anastasakis, B. Bordbar, G. Georg, and I. Ray. On challenges of model transformation from UML to Alloy. *Softw. Syst. Model.*, 9(1):69–86, 2010.

[4] B. Beckert, U. Keller, and P. H. Schmitt. Translating the Object Constraint Language into first-order predicate logic. In *Proc of the VERIFY, Workshop at Federated Logic Conferences (FLoC), Copenhagen, Denmark*, 2002.

[5] G. Bergmann, Á. Horváth, I. Ráth, D. Varró, A. Balogh, Z. Balogh, and A. Ökrös. Incremental Evaluation of Model Queries over EMF Models. In *MODELS'10*, volume 6395 of *LNCS*. Springer, 2010.

[6] G. Bergmann, A. Hegedüs, A. Horváth, I. Ráth, Z. Ujhelyi, and D. Varró. Implementing efficient model validation in EMF tools. In *26th IEEE/ACM International Conference on Automated Software Engineering*, pages 580 –583, 2011.

[7] E. Brottier, F. Fleurey, J. Steel, B. Baudry, and Y. Le Traon. Metamodel-based Test Generation for Model Transformations: an Algorithm and a Tool. In *17th International Symposium on Software Reliability Engineering, 2006. ISSRE '06.*, pages 85–94, 2006.

[8] A. D. Brucker and B. Wolff. The HOL-OCL tool, 2007. `http://www.brucker.ch/`.

[9] F. Büttner, M. Egea, J. Cabot, and M. Gogolla. Verification of ATL transformations using transformation models and model finders. In *14th International Conference on Formal Engineering Methods*, pages 198–213. Springer, 2012.

[10] J. Cabot, R. Clarisó, and D. Riera. UMLtoCSP: a tool for the formal verification of UML/OCL models using constraint programming. In *22nd IEEE/ACM International Conference*

on Automated Software Engineering (ASE'07), pages 547–548. ACM, 2007.

[11] J. Cabot, R. Clariso, and D. Riera. Verification of UML/OCL class diagrams using constraint programming. In *Software Testing Verification and Validation Workshop, 2008. ICSTW '08. IEEE International Conference on*, pages 73–80, 2008.

[12] G. Callow and R. Kalawsky. A satisficing bi-directional model transformation engine using mixed integer linear programming. *Journal of Object Technology*, 12(1):1: 1–43, 2013.

[13] A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. JTL: a bidirectional and change propagating transformation language. In *Software Language Engineering*, pages 183–202. Springer, 2010.

[14] C. Debreceni, Á. Horváth, Á. Hegedüs, Z. Ujhelyi, I. Ráth, and D. Varró. Query-driven incremental synchronization of view models. In *Proceedings of the 2nd Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling*, page 31. ACM, 2014.

[15] M. Famelis, R. Salay, and M. Chechik. Partial models: Towards modeling and reasoning with uncertainty. In *Proceedings of the 34th International Conference on Software Engineering*, pages 573–583, 2012.

[16] F. Fleurey, J. Steel, and B. Baudry. Validation in model-driven engineering: Testing model transformations. In *International Workshop on Model, Design and Validation*, pages 29–40, Nov 2004.

[17] L. Gammaitoni and P. Kelsen. F-alloy: An alloy based model transformation language. In *Theory and Practice of Model Transformations*, pages 166–180. Springer, 2015.

[18] H. Gholizadeh, Z. Diskin, S. Kokaly, and T. Maibaum. Analysis of source-to-target model transformations in quest. In *Proceedings of the 4th Workshop on the Analysis of Model Transformations*, pages 46–55, 2015.

[19] M. Gogolla, J. Bohling, and M. Richters. Validating UML and OCL models in USE by automatic snapshot generation. *Softw. Syst. Model.*, 4(4):386–398, 2005.

[20] H. Grönniger, J. O. Ringert, and B. Rumpe. System model-based definition of modeling language semantics. In *Formal Techniques for Distributed Systems*, volume 5522 of *LNCS*, pages 152–166. Springer, 2009.

[21] Á. Hegedüs, Á. Horváth, I. Ráth, and D. Varró. A model-driven framework for guided design space exploration. In *26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. IEEE Computer Society, 2011.

[22] Á. Horváth, Á. Hegedüs, M. Búr, D. Varró, R. R. Starr, and S. Mirachi. Hardware-software allocation specification of ima systems for early simulation. In *Digital Avionics Systems Conference (DASC)*. IEEE, 2014.

[23] D. Jackson. Alloy Analyzer. `http://alloy.mit.edu/`.

[24] E. K. Jackson, T. Levendovszky, and D. Balasubramanian. Reasoning about metamodeling with formal specifications and automatic proofs. In *Proc. of the 14th Int. Conf. on MODELS*, volume 6981 of *LNCS*, pages 653–667, 2011.

[25] E. K. Jackson, G. Simko, and J. Sztipanovits. Diversely enumerating system-level architectures. In *Proceedings of the 11th ACM Int. Conf. on Embedded Software*, page 11. IEEE Press, 2013.

[26] E. Kang, E. Jackson, and W. Schulte. An approach for effective design space exploration. In R. Calinescu and E. Jackson, editors, *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*, volume 6662 of *LNCS*, pages 33–54. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-21291-8.

[27] M. Kuhlmann, L. Hamann, and M. Gogolla. Extensive validation of OCL models by integrating SAT solving into use. In *TOOLS'11 - Objects, Models, Components and Patterns*, volume 6705 of *LNCS*, pages 290–306, 2011.

[28] N. Macedo and A. Cunha. Implementing QVT-R bidirectional model transformations using Alloy. In *Fundamental Approaches to Software Engineering*, pages 297–311. 2013.

[29] A. Milicevic, J. P. Near, E. Kang, and D. Jackson. Alloy*: A general-purpose higher-order relational constraint solver. In *37th IEEE/ACM Int. Conf. on Software Engineering, ICSE*, pages 609–619, 2015.

[30] *Object Constraint Language, v2.0*. The Object Management Group, May 2006. `http://www.omg.org/spec/OCL/2.0/`.

[31] R. Piskac, L. de Moura, and N. Bjorner. Deciding effectively propositional logic with equality, 2008. Microsoft Research, MSR-TR-2008-181 Technical Report.

[32] A. Queralt, A. Artale, D. Calvanese, and E. Teniente. OCL-Lite: Finite reasoning on UML/OCL conceptual schemas. *Data Knowl. Eng.*, 73:1–22, 2012.

[33] I. Ráth, A. Hegedüs, and D. Varró. Derived features for EMF by integrating advanced model queries. In *Modelling Foundations and Applications*, LNCS, pages 102–117. Springer Berlin / Heidelberg, 2012.

[34] R. Salay and M. Chechik. A generalized formal framework for partial modeling. In *Fundamental Approaches to Software Engineering*, volume 9033 of *LNCS*, pages 133–148. Springer Berlin Heidelberg, 2015. ISBN 978-3-662-46674-2.

[35] R. Salay, M. Famelis, and M. Chechik. Language independent refinement using partial modeling. In J. de Lara and A. Zisman, editors, *Fundamental Approaches to Software Engineering*, volume 7212 of *LNCS*, pages 224–239. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-28871-5.

[36] R. Salay, M. Chechik, M. Famelis, and J. Gorzny. A methodology for verifying refinements of partial models. *Journal of Object Technology*, 14(3):3:1–31, 2015.

[37] O. Semeráth and D. Varró. Validation of well-formedness constraints on uncertain model. In *Proceedings of the 10ht Conference of PhD Students in Computer Science*, 2016.

[38] O. Semeráth, Á. Horváth, and D. Varró. Validation of derived features and well-formedness constraints in dsls. In *International Conference on Model Driven Engineering Languages and Systems*, pages 538–554. Springer, 2013.

[39] O. Semeráth, A. Barta, A. Horváth, Z. Szatmári, and D. Varró. Formal validation of domain-specific languages with derived features and well-formedness constraints. *Software and Systems Modeling*, pages 1–36, 2015. ISSN 1619-1366.

[40] O. Semeráth, C. Debreceni, Á. Horváth, and D. Varró. Incremental backward change propagation of view models by logic solvers. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, pages 306–316. ACM, 2016.

[41] O. Semeráth, A. Vörös, and D. Varró. Iterative and incremental model generation by logic solvers. *Fundamental Approaches to Software Engineering, 19th International Conference, FASE 2016*, 2016.

[42] S. Sen, N. Moha, B. Baudry, and J.-M. Jézéquel. Meta-model Pruning. In *Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MODELS)*, Denver, Colorado, USA, Oct 2009.

[43] S. M. A. Shah, K. Anastasakis, and B. Bordbar. From UML to Alloy and back again. In *MoDeVVa '09: Proceedings of the 6th International Workshop on Model-Driven Engineering, Verification and Validation*, pages 1–10. ACM, 2009. ISBN 978-1-60558-876-6.

[44] M. Soeken, R. Wille, M. Kuhlmann, M. Gogolla, and R. Drechsler. Verifying UML/OCL models using boolean satisfiability. In *Design, Automation and Test in Europe, (DATE'10)*, pages 1341–1344. IEEE, 2010.

[45] E. D. Willink. An extensible OCL virtual machine and code generator. In *Proc. of the 12th Workshop on OCL and Textual Modelling*, pages 13–18. ACM, 2012.