AlloyInEcore: Embedding of First-Order Relational Logic into Meta-Object Facility for Automated Model Reasoning

Ferhat Erata
UNIT Information Technologies R&D, Turkey

Arda Goknil
University of Luxembourg, Luxembourg

Ivan Kurtev
Altran Netherlands, the Netherlands

Bedir Tekinerdogan
Wageningen University, the Netherlands

ABSTRACT

We present AlloyInEcore, a tool for specifying metamodels with their static semantics to facilitate automated, formal reasoning on models. Software development projects require that software systems be specified in various models (e.g., requirements models, architecture models, test models, and source code). It is crucial to reason about those models to ensure the correct and complete system specifications. AlloyInEcore allows the user to specify metamodels with their static semantics, while, using the semantics, it automatically detects inconsistent models, and completes partial models. It has been evaluated on three industrial case studies in the automotive domain (https://modelwriter.github.io/AlloyInEcore/).

CCS CONCEPTS

• **Software and its engineering → Specification languages**; *Semantics*; *Formal methods*;

KEYWORDS

Formal Reasoning; Modeling; Relational Logic; Alloy; KodKod

ACM Reference Format:
Ferhat Erata, Arda Goknil, Ivan Kurtev, and Bedir Tekinerdogan. 2018. AlloyInEcore: Embedding of First-Order Relational Logic into Meta-Object Facility for Automated Model Reasoning. In *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '18), November 4–9, 2018, Lake Buena Vista, FL, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3236024.3264588

1 INTRODUCTION

Model Driven Engineering (MDE) is becoming a crucial practice in industry due to the increasing complexity of software systems that warrant better support for managing development artifacts [17]. In MDE, software is developed by successively transforming abstract models to more concrete ones. Each model conforms to its metamodel, an artefact usually created using Ecore [7], a de facto industry standard for metamodeling and an example of implementation of the Meta-Object Facility (MOF) [11] which describes the means to create and manipulate models and metamodels. An important challenge in MDE is providing ability of automated, formal reasoning on models, e.g., checking model consistency and completing partial models [13, 16].

We present a tool, AlloyInEcore, which allows specification of metamodels with their static semantics and facilitates multiple forms of automated, formal reasoning on models. AlloyInEcore is targeted at environments that require integration and reasoning on heterogeneous models. Such environments are often encountered within the context of our research [18, 19] in collaboration with Ford-Otosan [12]. The key idea behind AlloyInEcore is that the static semantics of an Ecore metamodel can be specified within a simple first-order logic of sets and relations to support reasoning on models conforming to the metamodel.

Alloy [20] is a declarative modeling language based on first-order relational logic. It has been explored by the MDE community for the purpose of analyzing UML/OCL models [1, 6, 26]. Most of the existing tools and approaches use a transformation of UML/OCL models to Alloy, which, however, does not support directly some important concepts like multiple inheritance, generic types, and type parameters due to the fundamental differences between UML/OCL and Alloy notations [6, 30]. In the case of dealing with various models in different abstraction levels, it is required to enable the specification of such concepts, and herewith multiple forms of model reasoning. To do so, AlloyInEcore provides the following major features: (i) Alloy-like notation embedded into Ecore to specify the static semantics of metamodels based on First-order Logic (FOL), relational operators, and transitive closure; (ii) direct translation of Ecore metamodels into the language of Kodkod [29], an efficient SAT-based constraint solver for FOL with relational algebra. In this way we avoid the problems related to the differences between Alloy and Ecore. Our tool performs two major model reasoning tasks: completing partial models and detecting inconsistent model parts.

2 RELATED WORK

Several formal analysis methods and specification languages have been proposed relying on modern SAT-solvers, SMT solvers and theorem provers (e.g., Formula [21] using Z3 SMT-solver [24], Clafer [2] using Alloy along with Choco CSP solver [22], and Alloy using KodKod [29] that relies on SAT solvers like Minisat [8]).

A number of MDE solutions and tools provide automated model reasoning using existing formal analysis methods and specification languages based on constraint logic programming (e.g., [4, 5]), SAT-based model finders (e.g., [1, 6, 26, 27]), and SMT solvers (e.g., [15, 25]). However, to the best of our knowledge, none of

ESEC/FSE '18, November 4–9, 2018, Lake Buena Vista, FL, USA

ACM ISBN 978-1-4503-5573-5/18/11…$15.00
https://doi.org/10.1145/3236024.3264588

them provides a method that embeds FOL augmented with relational calculus into MOF/Ecore to specify the static metamodel semantics with the support for partial models, composition, cardinality constraints, multiple inheritance, generic types, and type parameters. For instance, Anastakis et al. [1] advocate to transform UML/OCL specifications to Alloy for automated model reasoning. The proposed transformation does not support multiple inheritance, generic types, and type parameters because of the differences between UML/OCL and Alloy notations. Lightning [14] is a tool-supported approach for defining some aspects of Domain-Specific Languages (e.g. abstract syntax and semantics) entirely in Alloy. In our approach, language designers can use AlloyInEcore to specify the abstract syntax of a language as an Ecore metamodel enriched with embedded Alloy-like statements. USE [23] is a tool for analyzing models expressed in UML and OCL. Similarly to our approach, USE translates models into relational logic and relies on the Kodkod library.

3 TOOL OVERVIEW

AlloyInEcore is a generalization of our previous tool Tarski [9, 10], which reasons on trace links in traceability models using configurable trace link semantics. Fig. 1 presents an overview of our tool. In Step 1, the user specifies an Ecore metamodel and its static semantics expressed in FOL augmented with the relational calculus [28] embedded in Ecore. To do so, AlloyInEcore natively supports *Alloy in Ecore* with a custom Eclipse editor.

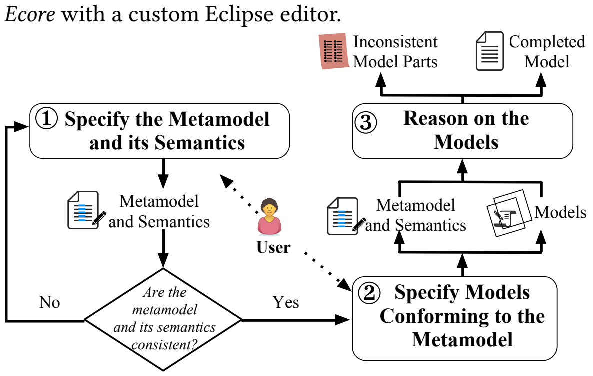

Figure 1: Tool Overview

Once the user specifies the metamodel and its semantics, AlloyInEcore allows creation of instance model(s) conforming to the metamodel (Step 2). After the model is created, the tool proceeds to Step 3 using automated model reasoning. In the following, we elaborate each step using the *theory of lists* as a running example.

3.1 Specification of Metamodels and Semantics

As the first step, the user specifies a metamodel and its semantics in our Alloy-like notation embedded into Ecore. The user can create the metamodel using any graphical, textual, or tree-based Ecore model editor including our AlloyInEcore editor. Fig. 2 gives the *theory of lists* metamodel in the Ecore graphical editor.

The user uses our editor to specify the static semantics of the metamodel (see Fig. 3). The keywords in blue and brown are introduced by AlloyInEcore for specifying the metamodel semantics, while the ones in red are the Ecore keywords for defining the metamodel itself. Some of the AlloyInEcore keywords (e.g., *ghost*, *model*, and *nullable*) were borrowed from JML [3].

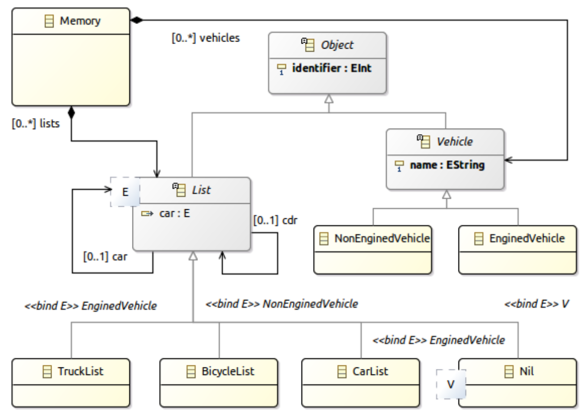

Figure 2: The Metamodel for the Theory of Lists

There are three abstract classes in Fig. 3: *Object*, *List*, and *Vehicle*. The *Object* is an abstract class at the root of the class hierarchy. The *ghost* keyword indicates that the *identifier* attribute will not be considered in the model reasoning (Line 5). The cardinality constraint in Line 8 specifies the lower and upper bounds on the *List* instances. The abstract class *List* is composed of two properties: the *car* mapping each *List* instance to an instance of another class (e.g., a *Vehicle* instance) and the *cdr* pointing to another *List* instance. The *?* keyword constrains these properties to be partial functions (Lines 9-10). To rule out cyclic lists, the *acyclic* keyword is used in the *cdr* property (Line 10). The *model* keyword defines the *eq* property as a relation to be inferred in the reasoning (Line 11). *Model* properties are not mapped to Ecore features.

TheoryOfLists.ecore Memory.xmi Memory.xmi

```
1 import Ecore : 'http://www.eclipse.org/emf/2002/Ecore';
2
3 package theoryoflists: tol= 'eu.modelwriter.examples.theoryoflists'{
4   public abstract class Object {
5      ghost attribute identifier : Integer;
6   }
7
8   public abstract class List<E> extends Object [5,7] {
9      property car : E [?];
10     property cdr : List<E> [?] {acyclic};
11     model property eq : List<E> [*] ;
12
13     invariant: all a, b: List | a in b.eq iff (a.car = b.car
14                  and a.cdr in b.cdr.eq and a.class = b.class);
15
16     invariant noStrayObjects: all v: Object - List | some v.~car;
17  }
18
19  public class one Nil<V> extends List<V>   {
20     invariant : no Nil.car;
21     invariant : no Nil.cdr;
22     invariant : all l: List - Nil | some l.cdr && some l.car;
23     invariant : all l: List | Nil in l.*cdr;
24  }
25
26  private class one Memory {
27     property some vehicles : Vehicle [*] {composes};
28     property some lists : List<? extends Vehicle> [*] {composes};
29  }
30
31  abstract class Vehicle extends Object [2,4] {
32     attribute name : String;
33     invariant : all disj a, b: Vehicle | a.name != b.name;
34     invariant : one v: Vehicle | v.name = "FORD F-150 XLT";
35  }
36
37  class EnginedVehicle extends Vehicle;
38  class NonEnginedVehicle extends Vehicle;
39
40  class TruckList extends List<EnginedVehicle>;
41  class CarList extends List<EnginedVehicle>;
42  class BicycleList extends List<NonEnginedVehicle>;
43 }
```

Figure 3: Example Metamodel Semantics in AlloyInEcore

Metamodel semantics is mainly given as *invariants*. For instance, the invariant in Lines 13-14 ensures that two *List* instances are equal (*'a in b.eq'* in Line 13) if and only if the head instances in the two lists are the same (*'a.car = b.car'* in Line 13), the subsequent *List* instances are equal (*'a.cdr in b.cdr.eq'* in Line 14), and the *List* instances are of the same type (*'a.class = b.class'* in Line 14). The invariant in Line 16 guarantees that each *Vehicle* instance is in at least one list (see the *some* keyword).

The *Nil* class represents the empty list (Line 19). The *one* keyword makes the *Nil* class a singleton set, which means there can be only one *Nil* instance in a model. A *Nil* instance has neither the *car* nor the *cdr* property (Lines 20-21), while a Non-nil *List* instance has both *car* and *cdr* (Line 22). A *Nil* instance is always a subsequent list of any *List* instance (Line 23). The singleton class *Memory* holds the *Vehicle* and *List* classes (see the `composes` keyword in Lines 27-28). It is important to note that '*List<? extends Vehicle>*' represents the *List* instances of any subclass of *Vehicle* (Line 28). Each *Vehicle* instance has a unique name (Line 33) and there is always exactly one *Vehicle* instance with the name "Ford F-150 XLT" (Line 34). There are two types of vehicles: *NonEnginedVehicle* and *EnginedVehicle* (Line 37-38). *TruckList* and *CarList* are lists of *EnginedVehicles* (Lines 40-41), while *BicycleList* is a list of *NonEnginedVehicle* (Line 42).

3.2 Specification of Models

The user can use any graphical, textual, or tree-based Ecore model editor to specify models conforming to the metamodel (Step 2 in Fig. 1). Before creating any model, AlloyInEcore automatically checks if the user can specify at least one valid model that conforms to the metamodel and its static semantics. The user may have specified some contradicting invariants where it is not possible to create a valid model. AlloyInEcore automatically identifies the contradictions in the metamodel specification and notifies the user.

3.3 Automated Reasoning on Models

Model completion and consistency checking aim at deriving new instances and relations in the given model, and determining model parts violating the metamodel semantics, respectively. These two activities are processed as a single reasoning activity because they use the same reasoning machinery. The consistency checking can be considered as part of model completion because a partial model is completed only if it is consistent.

3.3.1 Checking Model Consistency. AlloyInEcore takes a model and its metamodel as input, and automatically identifies, using the static metamodel semantics, inconsistent model parts as output. AlloyInEcore provides an explanation of the inconsistency by giving all the instances and relations causing the inconsistency. Fig. 4 gives three AlloyInEcore panes for an example inconsistent model of the theory of lists. The first pane in Fig. 4 gives the inconsistent model, while, in the second pane, AlloyInEcore highlights part of the metamodel semantics causing the inconsistency. Although the *cdr* property of the *List* class is given *acyclic* (see the highlighted *acyclic* keyword in Line 10), the red colored *cdr* in *TruckList$0* is referring to the instance itself. The third pane in Fig. 4 gives the first order relational logic formula that corresponds to the *acyclic* keyword for further explanation of the inconsistency.

3.3.2 Completing Partial Models. If the given model is consistent, AlloyInEcore automatically deduces new instances and relations in the input model using the static metamodel semantics. The model is completed only if it is consistent and not an exact model (i.e., a model where it is not possible to infer anything more). For instance, the user removes the cyclic *cdr* relation in Fig. 4 to make the model consistent, and then AlloyInEcore completes the model. Our tool infers 14 different complete models for the partial model in Fig. 4. Fig. 5 shows one of these inferred models.

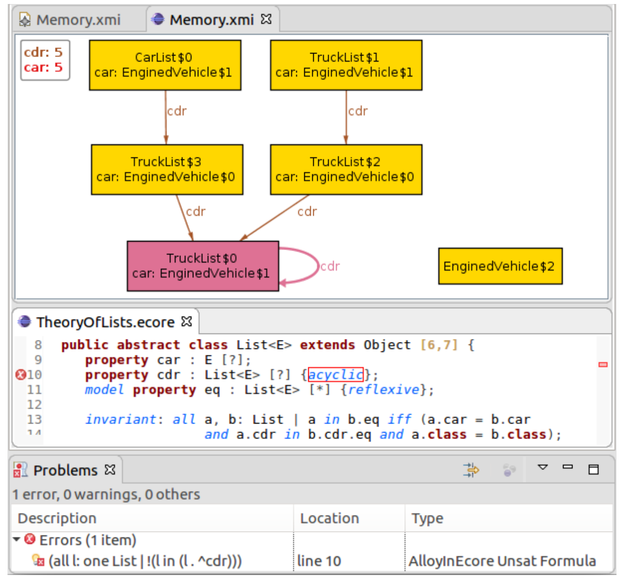

Figure 4: Example Inconsistent Model of the Theory of Lists

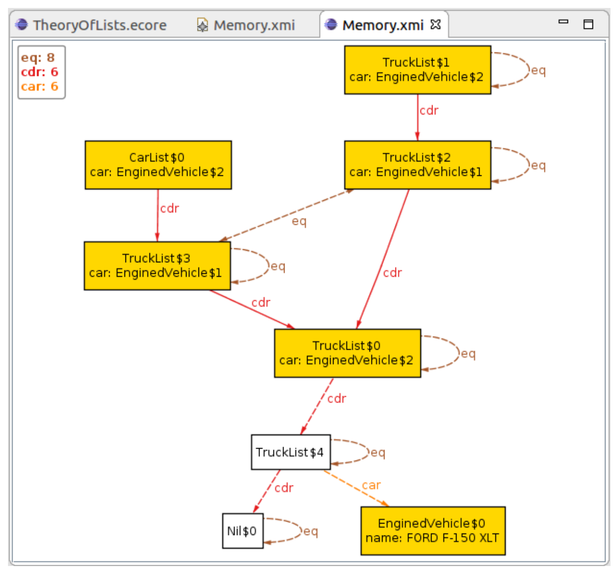

Figure 5: Example Completed Model of the Theory of Lists

The dashed arrows and the white boxes represent the automatically inferred relations and instances, respectively. AlloyInEcore automatically infers one *Nil* instance (see the *one* keyword in Line 19 in Fig 3) and one *TruckList* instance (see the invariant in Line 16 and the cardinality constraint in Line 8). It also infers the name for *EnginedVehicle$0* (see the invariant in Line 34) and the *car* relation between *TruckList$4* and *EnginedVehicle$0* (see the invariant in Line 16). The inferred *Nil* instance is a subsequent list of the inferred *TruckList* instance (see the invariant in Line 23). Each *List* instance is equal to itself, while *TruckList$3* is equal to *TruckList$2* (see the invariant in Lines 13-14).

4 EVALUATION

Our goal was to assess, in an industrial context, the benefits of using AlloyInEcore to facilitate automated model reasoning using user-defined metamodel semantics. We selected three case studies from the Electronically Controlled Air Suspension (ECAS) system developed by Ford-Otosan [12]. Each case study is with an ECAS artifact conforming to a different metamodel (i.e., a requirements specification, a data flow diagram and a SysML model).

Before conducting the case studies, the Ford-Otosan engineers were given presentations illustrating the AlloyInEcore steps and a tool demo. The engineers held various roles (e.g., senior software and system engineers) with substantial development experience. For each case study, we assisted the engineers in specifying metamodel semantics in AlloyInEcore (the 1st, 2nd and 3rd columns in Table 1).

Table 1: Number of Classes, Properties, Invariants, Models and Completed & Inconsistent Parts in the Case Studies

	Meta-classes	Pro-perties	Invar-iants	Model Elements	Comple. Elements	Inconsis. Elements
#1	3	7	42	116	480	5
#2	5	8	6	51	114	1
#3	15	12	10	135	432	2

To evaluate the output, we had semi-structured interviews with the engineers. All the completed model parts and the identified inconsistencies were confirmed by the engineers to be correct (the 4th and 5th columns). The engineers considered the automated reasoning on models to be highly valuable. The Alloy-like notation embedded into Ecore was sufficient and easy for the engineers to specify the metamodel semantics in the case studies. For the largest model (the 3rd row), it took 126 secs to perform the reasoning.

5 IMPLEMENTATION & AVAILABILITY

AlloyInEcore has been implemented as an Eclipse plug-in. We use Kodkod [29] to perform automated reasoning on models based on the metamodel semantics. AlloyInEcore translates the input metamodel and semantic specification into a first order relational formula. It also translates the input model into a *Universe* and *Bounds* in KodKod. Kodkod translates the formula and the bounds into a Boolean satisfiability (SAT) problem to invoke an off-the-shelf SAT solver. If the SAT solver finds a SAT solution to the problem, Kodkod translates that SAT solution into a solution to the formula from which AlloyInEcore derives the completed model.

AlloyInEcore is approximately 31K lines of Java code, excluding comments and third-party libraries. Additional details about AlloyInEcore, including executable files and a screencast covering motivations, are available on the tool's website at:

https://modelwriter.github.io/AlloyInEcore/

6 CONCLUSION

We presented a tool that enables the specification of metamodel semantics for automated model reasoning. The key characteristics of our tool are (1) enabling the user to specify metamodels and their semantics in a single environment, (2) identifying inconsistent model parts, and (3) completing partial models. The tool has been evaluated over three industrial case studies.

ACKNOWLEDGMENTS

This work is conducted within ASSUME [19] project and partially supported by the Scientific and Technological Research Council of Turkey under project #9150181, and by the European Research Council under the European Union's Horizon 2020 research and innovation programme (grant no 694277).

REFERENCES

[1] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. 2007. UML2Alloy: A challenging model transformation. In *MODELS'07*. 436–450.
[2] Kacper Bąk, Zinovy Diskin, Michał Antkiewicz, Krzysztof Czarnecki, and Andrzej Wąsowski. 2016. Clafer: unifying class and feature modeling. *Software & Systems Modeling* 15, 3 (2016), 811–845.
[3] Lilian Burdy, Yoonsik Cheon, David R. Cok, Michael D. Ernst, Joseph R. Kiniry, Gary T. Leavens, K. Rustan M. Leino, and Erik Poll. 2005. An overview of JML tools and applications. *International Journal on Software Tools for Technology Transfer* 7, 3 (2005), 212–232.
[4] Jordi Cabot, Robert Claris, Daniel Riera, et al. 2008. Verification of UML/OCL Class Diagrams using Constraint Programming. In *ICSTW'08*. 73–80.
[5] Jordi Cabot, Robert Clarisó, and Daniel Riera. 2007. UMLtoCSP: a Tool for the Formal Verification of UML/OCL Models using Constraint Programming. In *ASE'07*. 547–548.
[6] Alcino Cunha, Ana Garis, and Daniel Riesco. 2015. Translating between Alloy specifications and UML class diagrams annotated with OCL. *Software & Systems Modeling* 14, 1 (2015), 5–25.
[7] Ecore. 2017. https://www.eclipse.org/modeling/emf/. (2017).
[8] Niklas Eén and Niklas Sörensson. 2003. An Extensible SAT-solver. In *SAT'03*. 502–518.
[9] Ferhat Erata, Moharram Challenger, Bedir Tekinerdogan, Anne Monceaux, Eray Tüzün, and Geylani Kardas. 2017. Tarski: A Platform for Automated Analysis of Dynamically Configurable Traceability Semantics. In *SAC'17*. 1607–1614.
[10] Ferhat Erata, Arda Goknil, Bedir Tekinerdogan, and Geylani Kardas. 2017. A Tool for Automated Reasoning about Traces based on Configurable Formal Semantics. In *ESEC/FSE'17*. 959–963.
[11] MOF (Meta-Object Facility). 2017. http://www.omg.org/mof/. (2017).
[12] Ford-Otosan. 2017. http://www.fordotosan.com.tr. (2017).
[13] Robert France and Bernhard Rumpe. 2007. Model-driven Development of Complex Software: A Research Roadmap. In *FOSE'07*. 37–54.
[14] Loïc Gammaitoni, Pierre Kelsen, and Christian Glodt. 2015. Designing languages using lightning. In *SLE'15*. 77–82.
[15] Aboubakr Achraf El Ghazi and Mana Taghdiri. 2011. Relational Reasoning via SMT Solving. In *FM'11*. 133–148.
[16] Arda Goknil, Ivan Kurtev, Klaas van den Berg, and Jan-Willem Veldhuis. 2011. Semantics of Trace Relations in Requirements Models for Consistency Checking and Inferencing. *Software & Systems Modeling* 10, 1 (2011), 31–54.
[17] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. 2011. Empirical Assessment of MDE in Industry. In *ICSE'11*. 471–480.
[18] ITEA. 2014. ModelWriter: Text & Model Synchronized Document Engineering Platform. https://itea3.org/project/modelwriter.html. (Sep 2014).
[19] ITEA. 2015. ASSUME: Affordable Safe & Secure Mobility Evolution. https://itea3.org/project/assume.html. (Sep 2015).
[20] Daniel Jackson. 2012. *Software Abstractions: Logic, Language, and Analysis*. MIT.
[21] Ethan Jackson, Tihamér Levendovszky, and Daniel Balasubramanian. 2011. Reasoning about metamodeling with formal specifications and automatic proofs. In *MODELS'11*. 653–667.
[22] Narendra Jussien, Guillaume Rochart, and Xavier Lorca. 2008. Choco: an open source java constraint programming library. In *OSSICP'08*. 1–10.
[23] Mirco Kuhlmann and Martin Gogolla. 2012. From UML and OCL to Relational Logic and Back. In *MODELS'12*. 415–431.
[24] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *TACAS'08*. 337–340.
[25] Nils Przigoda, Frank Hilken, Judith Peters, Robert Wille, Martin Gogolla, and Rolf Drechsler. 2016. Integrating an SMT-Based ModelFinder into USE. In *MoDeVVa@MODELS'16*. 40–45.
[26] Seyyed MA Shah, Kyriakos Anastasakis, and Behzad Bordbar. 2009. From UML to Alloy and back again. In *MODELS'09*. 158–171.
[27] Mathias Soeken, Robert Wille, Mirco Kuhlmann, Martin Gogolla, and Rolf Drechsler. 2010. Verifying UML/OCL Models using Boolean Satisfiability. In *DATE'10*.
[28] Alfred Tarski. 1941. On the Calculus of Relations. *The Journal of Symbolic Logic* 6, 03 (1941), 73–89.
[29] Emina Torlak and Daniel Jackson. 2007. Kodkod: A Relational Model Finder. In *TACAS'07*. 632–647.
[30] Mandana Vaziri and Daniel Jackson. 2000. Some Shortcomings of OCL, the Object Constraint Language of UML. In *TOOLS'00*. 555–562.