Model Checking of UML Class Diagrams including OCL using Relational Logic

Patrick Vogt and Lars-Erik Kimmel

Hochschule RheinMain - University of Applied Sciences Wiesbaden, Germany
{patrick.b.vogt|lars-erik.b.kimmel}@student.hs-rm.de,
WWW home page: http://www.cs.hs-rm.de/~{pvogt002|lkimm002}

Abstract. The popular process models for object-oriented software development do not comprise model checking techniques. Model checking tools which take UML class diagrams including OCL as input could facilitate the incorporation of model checking techniques into everyday software engineering workflows.
This paper describes a two-step model checking approach for UML class diagrams including OCL: the class diagrams are first transformed into relational logic. The actual model checking is then performed by existing model checking tools.

1 Introduction

Formal methods and, in particular, model checking technologies are almost exclusively applied during the development of safety critical software systems. They are mainly omitted in the everyday software engineering process [BA08, Som10].

There are a few special tools [GBR07, CCR08, ABGR] for formal reasoning in the context of UML class diagrams [Obj11] and OCL expressions [Obj10], and several established non-object-oriented model checking tools like [Jac02, Hol97].

Manual transformation of UML class diagrams into existing non-object-oriented model checking languages is possible, but often inefficient. The objective of this paper is to present an approach which enables the average object-oriented software developer to efficiently perform model checking on UML class diagrams.

Section 2 briefly describes existing tools which can be directly applied to UML class diagrams. Section 3 explains a strategy for automatic transformation of UML class diagrams into relational logic. The authors of this paper have developed two model checking tools according to this strategy. The final section summarizes the current status of these development efforts.

2 Related Work

Three of the existing UML model checking approaches are directly related to the work in this paper: USE [GBR07], UML2Alloy [ABGR] and FMC [HW12].

USE is a language and an environment for the specification of UML class diagrams with OCL and for the checking of UML object diagrams (snapshots)

against a given specification. Both class and object diagrams have to be manually provided as text data. It is not possible to automatically generate valid instances of a model that fulfill every specified constraint and to iterate over those instances similar to the Alloy Analyzer or Constraint Programming. Furthermore it is not possible to force USE to show counterexamples with respect to a given specification.

UML2Alloy transforms an UML class diagram including a subset of OCL into an Alloy model. The class diagram has to be provided in the form of an XMI file. The tool also provides an external view for showing valid instances for the UML class diagrams as object diagrams [Anab]. Unfortunately the reference manual of UML2Alloy suggests that the produced class diagrams should be specified with one specific outdated version of ArgoUML [Anaa]. UML2Alloy is still in beta state.

[HW12] follow an approach similar to this paper. They transform a class diagram including OCL into the CPL based language Formula. Constraint logic is used for finding valid instances of the class diagram resp. counterexamples for a given assumption.

3 Basic Concepts

This paper's strategy for automatic transformation of class diagrams into relational logic is mainly based on [BCG05]. The rules have been slightly adapted for relational logic:

- A class $Class_1$ and its instances $c_0, c_1, \ldots$ are transformed into a unary relation, e.g.
  $$Class_1 = \{(c_0), (c_1), \ldots\}, \ldots$$
- For every binary association $asso$ between $Class_1$ and $Class_2$ the following relational formulae must be valid:
  - type assertion: $\forall c_1, c_2.\ ((c_1, c_2) \in asso) \rightarrow (c_1 \in Class_1) \wedge (c_2 \in Class_2)$
  - multiplicity: $\forall c_1.\ (c_1 \in Class_1) \rightarrow$
    $$(lowerBound \leq \#\{c_2 \mid (c_1, c_2) \in association\} \leq upperBound)$$
- A class attribute is modeled simply like a binary association but with fixed bounds of 1 for its upper and lower bound
- An abstract class $AbstractClass$ and a corresponding generalization of classes $ConcreteClass_1$ and $ConcreteClass_2$ are transformed into the following expression: $\forall c.\ (c \in AbstractClass) \rightarrow$
  $$(c \in ConcreteClass_1)\ \texttt{xor}\ (c \in ConcreteClass_2)$$

These transformations present only a subset of the UML specification for UML class diagrams. In addition there are still more transformations e.g. for n-ary associations and association classes.

There are two approaches for the processing of OCL constraints in this context: On the one hand it is possible to transform (a subset of) OCL as well into relational logic and transform the OCL constraints into constraints within the relational model. On the other hand a hybrid approach can be used: A model

checking backend for relational logic such as KodKod [Tor09] creates several iterable instances. The OCL constraints are checked afterwards by another tool.

The transformation of OCL pre- and postcondition into relational logic requires the concept of linear time. Therefore the relational model is augmented by a unary relation $Time$ with corresponding linear ordering constraints. Every relation of the non-temporal model has to be extended to incorporate a $Time$-value, e.g. an operation `setValue(v: String)` which is called on the object x_0 in time t_1 and replaces the old value v_0 of the attribute `value` with the new value v_1 could therefore be modeled by the following relations:

$$Time = \{(t_0), (t_1), (t_2), \ldots\},\ value = \{(x_0, v_0, t_1), (x_0, v_1, t_2\}$$

There are two versions of this approach: *Time Axis* and *Global State*. Both are further described in [Jac12], e.g.:

In addition, it is necessary to model the concept of object creation and destruction. This is achieved by introducing a relation $isAlive$ which represents the lifetime of an object. If and only if an element of the unary relation $Time$ is related with this object, the object is alive at this point in time.

There are several transformations of OCL constraints into relational logic which have to bridge semantic discrepancies. Challenging yet feasible is e.g. the transformation of OCL expressions which operate on bags, such as `sum()`. This can be solved as follows: The elements of a collection $E = \{e_0, e_1, \ldots\}$ must be combined with an ordered index set $I = \{(i_1, i_2), (i_2, i_3), (i_3, i_4) \ldots\}$. The last column of the relation $E \times I \times Integer$ is then constrained to contain the temporary sum while adding all the elements of E and iterating over the index set I with the defined sequence in I. With this pattern the result of the `sum()` operation is then given as $Integer$-value of the last tuple of $E \times I \times Integer$.

4 Conclusion

The potential field of application of UML model checking tools ranges from teaching of UML/OCL in software engineering classes to the incorporation of model checking techniques into everyday object-oriented software development.

We have developed two model checking tools based on the approach presented in this paper. The tools are available for download at the authors' home pages. The current scope comprises the most frequent concepts of class diagrams (associations, attributes, inheritance) and the basic concepts of pre- and post-conditions. Alloy, KodKod and USE are used as model checking backends.

Further development will focus on symmetry filtering and on the more challenging transformations, e.g.: Arithmetic operations cannot be formulated efficiently in relational logic. This is the domain of CP-based tools. Some of the OCL aggregate functions are based on the concept of a *bag*, which directly contradicts the set-based concept of relations. The transformation of a simple bag therefore involves several relations. In OCL operations can be combined by, possibly recursive, method calls. This kind of nesting is not directly supported in Alloy.

References

[ABGR] Anastasakis, Kyriakos ; Bordbar, Behzad ; Georg, Geri ; Ray, Indrakshi: UML2Alloy: A Challenging Model Transformation. http://dx.doi.org/10.1007/978-3-540-75209-7_30. In: Engels, Gregor ; Opdyke, Bill ; Schmidt, Douglas ; Weil, Frank : *Model Driven Engineering Languages and Systems* Bd. 4735. Springer Berlin / Heidelberg. – ISBN 978–3–540–75208–0, 436-450

[Anaa] Anastasakis, Kyriakos : *UML2Alloy - Reference Manual*. http://www.cs.bham.ac.uk/~bxb/UML2Alloy/files/uml2alloy_manual.pdf, Last checked: 30. January. 2012

[Anab] Anastasakis, Kyriakos : *UML2Alloy - webpage*. http://www.cs.bham.ac.uk/~bxb/UML2Alloy/index.php, Last checked: 30. January. 2012

[BA08] Ben-Ari, Mordechai: *Principles of the Spin Model Checker*. Springer, 2008. – ISBN 978–1–84628–769–5

[BCG05] Berardi, Daniela ; Calvanese, Diego ; Giacomo, Giuseppe D.: Reasoning on UML class diagrams. In: *Artificial Intelligence* 168 (2005), Nr. 1–2, 70 - 118. http://dx.doi.org/10.1016/j.artint.2005.05.003. – DOI 10.1016/j.artint.2005.05.003. – ISSN 0004–3702

[CCR08] Cabot, J. ; Clariso, R. ; Riera, D.: Verification of UML/OCL Class Diagrams using Constraint Programming. In: *Software Testing Verification and Validation Workshop, 2008. ICSTW '08. IEEE International Conference on*, 2008, S. 73 –80

[GBR07] Gogolla, Martin ; Büttner, Fabian ; Richters, Mark: USE: A UML-Based Specification Environment for Validating UML and OCL. In: *Science of Computer Programming* 69 (2007), S. 27–34

[Hol97] Holzmann, G.J.: The model checker SPIN. In: *Software Engineering, IEEE Transactions on* 23 (1997), may, Nr. 5, S. 279 –295. http://dx.doi.org/10.1109/32.588521. – DOI 10.1109/32.588521. – ISSN 0098–5589

[HW12] Horn, Julian ; Wenz, Carl-Phillip: *Model Checking of UML Class Diagrams with OCL using CPL*. http://www.cpwenz.de/FMC. Version: 2012

[Jac02] Jackson, Daniel: Alloy: a lightweight object modelling notation. In: *ACM Trans. Softw. Eng. Methodol.* 11 (2002), April, 256–290. http://dx.doi.org/http://doi.acm.org/10.1145/505145.505149. – DOI http://doi.acm.org/10.1145/505145.505149. – ISSN 1049–331X

[Jac12] Jackson, Daniel: *Software Abstractions: Logic, Language, and Analysis*. Revised. The MIT Press, 2012. – ISBN 978–0262017152

[Obj10] Object Management Group: *Object Constraint Language - Version 2.3*. http://www.omg.org/spec/OCL/2.3.1/. Version: 2010

[Obj11] Object Management Group: *Unified Modeling Language*. http://www.omg.org/spec/UML/. Version: 2011

[Som10] Sommerville, Ian: *Software Engineering*. 9th. Addison Wesley, 2010. – ISBN 978–0137035151

[Tor09] Torlak, Emina : *A Constraint Solver for Software Engineering: Finding Models and Cores of Large Relational Specifications*. Version: 2009. http://people.csail.mit.edu/emina/pubs/kodkod.phd.pdf, Last checked: 03. February. 2012