Towards Automatic Determination of Problem Bounds for Object Instantiation in Static Model Verification

Work-In-Progress

Mathias Soeken Robert Wille Rolf Drechsler

Institute of Computer Science, University of Bremen
Group of Computer Architecture, 28359 Bremen
Bremen, Germany
{msoeken,rwille,drechsle}@informatik.uni-bremen.de

ABSTRACT

The application of formal methods in the detection of inconsistencies and design flaws within models has been intensely studied in recent years. Since consistency checking is in principle undecidable due to the infinite number of possible system states, problem bounds have to be defined making the analysis tractable. However, defining these problem bounds requires detailed design knowledge and, thus, impedes an automatic verification flow.

In this paper, we present first ideas and results of how to automatically determine valid problem bounds for consistency checking algorithms. For this purpose, we make use of automatic proof engines for linear integer arithmetic. We describe the approach by means of class diagrams given in the *Unified Modeling Language* (UML) extended by constraints given in the *Object Constraint Language* (OCL).

1. INTRODUCTION AND BACKGROUND

Verifying and validating the correctness of a model in the absence of a concrete application became an active research area in recent years. Especially in the context of *model driven engineering* (MDE) in which a model is the starting point for all successive design steps, fundamental flaws can be detected before the first code line has been written. Verification techniques exist to detect static inconsistencies or erroneous dynamic behaviors such as dead locks or unreachable operations and system states. While this provides automatic verification methods, it also finds application in the interactive design process with the user. For example, these methods enable to generate certain system states which can assist the user in creating the model and acquire a better understanding due to guided model exploration. With the application of automatic proof engines, these tasks can be handled quite efficiently. In particular, techniques based on Boolean satisfiability or *SAT Modulo Theory* (SMT) have been applied to verify models written in UML/OCL [12, 11], Alloy [8], or Kodkod [13].

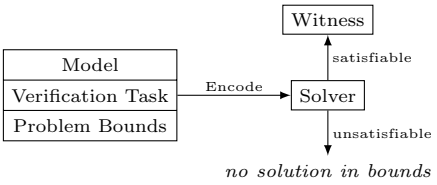


Figure 1: Model Verification

Consistency checks comprise thereby both, static and dynamic questions. In the static view on the model, common verification tasks are e.g. the check whether a non-trivial system state can be generated or if the model includes independent OCL constraints [6]. On the other side, the executability of operations, the reachability of wanted or unwanted system states as well as the detection of dead locks are typical verification tasks targeting the dynamic view of a model. Solutions dealing with both static [12, 9, 1, 2] and dynamic aspects [11, 3] have been presented in the past.

One possible flow which is applied for that purpose is depicted in Figure 1. Given a model and a verification task, the problem is *encoded* as an instance of the satisfiability problem which can be processed by an appropriate solve engine. From the solver's solution, a witness can be constructed, e.g. a valid system state or a sequence diagram depending on the verification task and the input model. Otherwise, if no solution can be found, it can be concluded that no such witness exists.

However, since the models contain data structures which (theoretically) have an infinite range, most verification tasks are in principle undecidable. Thus, it is necessary to specify problem bounds such as the number of objects in a system state or the domain of data types such as integers. While for the latter one established bounds (e.g. 64-bit integers) can be assumed without illegally simplifying the model too much, no obvious bounds for the number of objects exist. Instead, all approaches presented so far require that these problem bounds have to be specified by the designer. This usually requires detailed design knowledge.

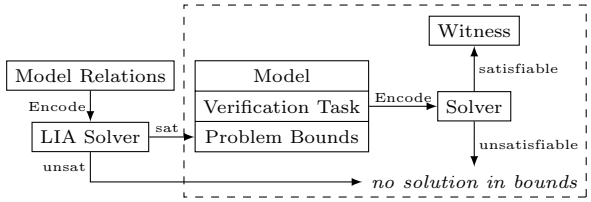


Figure 2: Automatic Determination of Problem Bounds

In this paper, we propose first solutions to automatically determine valid problem bounds for the number of object instantiations that can be used for consistency checks on the model. Thus, if the model is determined to be inconsistent, the user can be assured that this is because of contradictory constraints but not because of invalid specified problem bounds. For this purpose, we consider UML [10] class diagrams that are extended by OCL [14] constraints. Further, we consider static consistency checks, i.e. we are in particular interested in determining a valid number of instances per class for the resulting system state.

The paper is structured as follows. In the next section, the general idea is sketched by means of examples leading to an extension of the current verification flow. Afterwards, details and first experimental results of the proposed approach are presented in Section 3 and Section 4, respectively. Finally, Section 5 discusses related work before the paper is concluded in Section 6.

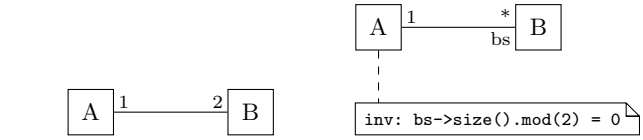
2. GENERAL IDEA

We propose to extend the flow from Figure 1 as shown in Figure 2. That is, before the actual model verification is executed, a pre-process is applied which automatically determines appropriate bounds for the object instantiation. For this purpose, the relations between the respective classes are considered. In the context of UML class diagrams, these relations are expressed by means of both associations (and their multiplicities) and OCL constraints. Those are encoded in terms of a *Linear Integer Arithmetic* (LIA) [4] instance and, afterwards, solved by an according LIA solver. From the result of this solving process, the respective problem bounds can be obtained.

The following example illustrates the general idea in more detail.

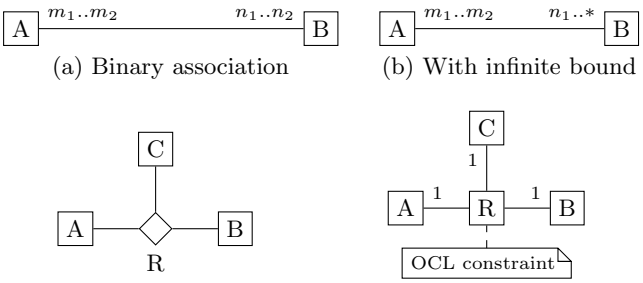
Example 1. *Figure 3 shows two class diagrams. From the associations of the first diagram, one can conclude that in a valid system state at least twice as many instances of class* B *than instances of class* A *must exist. In contrast, the second diagram bounds its instantiations by means of an OCL invariant. More precisely, the invariant constraints that in a valid system state the number of instances of class* B *must be even.*

That is, the bounds of object instantiation can be deduced from the corresponding UML and OCL constraints. However, while the bounds can easily be determined in simple UML diagrams as shown in Figure 3, this determination becomes more complex considering models composed of hundreds or even thousands of classes. In order to cope with increasing complexity, all these relations and constraints are encoded as a set of linear (in)equations. Then, these



(a) Expressed by associations (b) Expressed by OCL

Figure 3: Relations between UML classes



(a) Binary association (b) With infinite bound

(c) Ternary association (d) Alternative repr.

Figure 4: UML constraints

(in)equations can be solved e.g. by SMT solvers with LIA [4] support. From the solution of the solver, valid problem bounds can be deduced. On the other hand, if no valid problem bounds can be determined, it can already be deduced that no solution exists for any bounds, i.e. the model is inconsistent due to an invalid specification of the model bounds.

3. IMPLEMENTATION

In this section, a first implementation of the proposed idea is presented. So far, the approach fully supports relations expressed by associations. Possible directions how to support relations expressed by OCL constraints are left for future work. However, the general idea is discussed by means of an example.

To encode the outlined problem, a formula

$$f : \mathbb{N}_0 \times \mathbb{N}_0 \times \cdots \times \mathbb{N}_0 \to \mathbb{B}$$

is created. For this purpose, a variable $x_C \in \mathbb{N}_0$ is introduced for each class C in the considered UML model. Each x_C-variable represents the number of objects to be derived from class C. Using these variables, all restrictions enforced by the associations are encoded.

Figure 4(a) shows a generic binary UML association including multiplicities defined over intervals. This UML constraint expresses that each object of class B must be linked to at least m_1 (lower bound), but at most m_2 (upper bound) objects of class A ($0 \leq m_1 \leq m_2$). The same applies to class A analogously. The conjunction of the following LIA constraints are used to encode this relation.

First, constraints ensuring the existence of the minimal number of objects are added. This is expressed by

$$x_A \geq \max\{1, m_1\} \quad \wedge \quad x_B \geq \max\{1, n_1\}. \tag{1}$$

The terms $\max\{1, m_1\}$ and $\max\{1, n_1\}$ imply that each class is instantiated at least once. This is necessary, since empty system states are not considered.

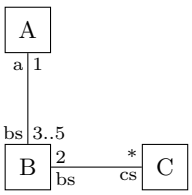


Figure 5: Example Model

Second, the correlation of x_A and x_B is constrained. For the case of $m_1 = m_2 = 1$ and $n_1 = n_2$, constraining the correlation is straightforward. Then, for each object derived from class A, n_1 objects from class B are needed. To encode this correlation, the LIA constraint $x_B = n_1 x_A$ needs to be added. If additionally $m_1 = m_2 > 1$, this constraint is extended to $m_1 x_B = n_1 x_A$. Given that, the generic LIA constraint additionally considering intervals (i.e. $m_1 < m_2$ and $n_1 < n_2$) can be deduced

$$m_2 x_B \geq n_1 x_A \quad \wedge \quad m_1 x_B \leq n_2 x_A. \tag{2}$$

Using these formulations most of the UML constraints can be encoded. Beyond that, only the following special cases have to be addressed separately:

- *Infinite upper bounds*
  In fact, infinite upper bounds (i.e. associations with $m_2 = \infty$ or $n_2 = \infty$) weaken the restrictions on the number of objects derived from a class. Accordingly, parts of the LIA constraints from Eq. (2) can be removed. As an example, for the association depicted in Figure 4(b) with $n_2 = \infty$, the term $m_1 x_B \leq n_2 x_A$ evaluates to $\lim_{n_2 \to \infty} m_1 x_B \leq n_2 x_A = m_1 x_B \leq \infty$. This is always true and, thus, the term can be omitted. Analogously, this can be done for $m_2 = \infty$.

- *Unary associations*
  Unary associations represent a special case of binary associations. Their mapping to LIA constraints is already covered by the encoding from Eq. (2). Note that unary associations are only valid, if they define an *n-to-n* relation or if they have infinite bounds.

- *n-ary associations*
  Arbitrary n-ary associations (with $n > 2$) can be mapped to LIA constraints in a recursive manner. To illustrate this, consider the ternary association given in Figure 4(c). According to [7], this can be transformed to equivalent binary associations by adding a *helper class* (denoted by R) and the following OCL constraint (see also Figure 4(d)):

```
R->forAll(r,r'|
  (r.ra=r'.ra and r.rb=r'.rb and r.rc=r'.rc)
  implies r=r'
)
```

  By applying this method recursively, n-ary associations with $n > 3$ can be transformed accordingly. From this representation, the respective LIA constraints can be derived. Note that the OCL constraint is not considered in the LIA instance, and therefore, this may lead to false positives. However, contradictions caused by such an association will then be detected in the successive consistency checking step.

Table 1: First results

Model	Classes	Associations	Run-Time (s)
CarRental	6	4	0.00
OCLMetamodel	16	4	0.00
Sudoku	5	7	0.00
Train	6	9	0.00
PyQt4	631	757	2.11
Android	713	3459	2.64
Java6	2462	53985	651.44

EXAMPLE 2. *Consider the model depicted in Figure 5. Applying Eq. (1) and Eq. (2) to this model leads to the following encoding:*

$$\begin{array}{rlll} & (x_A \geq 1) & \wedge & (x_B \geq 3x_A) \\ \wedge & (x_B \geq 3) & \wedge & (x_B \leq 5x_A) \\ \wedge & (x_C \geq 1) & & \end{array}$$

Passing this encoding to a respective solving engine might lead to the results $x_A = 1$, $x_B = 3$, and $x_C = 1$. That is, one object each of class A *and class* C *and three objects of class* B *are a valid number of instances for this model.*

4. EXPERIMENTAL RESULTS

Table 1 shows first experimental results obtained by this approach. Here, the bounds on the number of objects have been determined for UML models where the relations between the classes are specified by associations only. As examples, selected benchmarks from the USE tool [5] and larger models that have been re-engineered from software libraries are applied. As can be seen, even for class diagrams composed of more than 2,000 classes, the respective bounds can be determined within a couple of minutes. The obtained results can be used in successive verification steps such as consistency checking.

Besides associations also OCL constraints may imply dependencies between numbers of object instances. One obvious expression is `size()`. For example, in conjunction with `allInstances()` the expression

```
A.allInstances()->size()
```

directly corresponds to x_A. A similar situation exists when `size()` is applied to a navigation expression as in Example 1.

It is left for future work to determine further OCL operations and more complex expressions that can be transformed into LIA constraints. Also the limitations of these transformations have to be examined. Some expressions are hard to translate, e.g.:

```
some_int = 5 implies bs.size() = 3
```

This would require to transform also attribute values into the LIA instance which significantly increases the search space.

5. RELATED WORK

All presented automatic methods for consistency checking require either an exact number of instances or an interval. In Alloy [8], the problem bound is denoted as *scope* which can be defined exactly or by lower and upper bounds. If no scope is given, Alloy performs some basic analysis to automatically detect a good scope, e.g. a singleton is only instantiated once. However, no relation between model elements (*signatures* in Alloy) is taken into account. If no information can be obtained from the model, a default scope of 3 elements per signature is applied.

Also the Kodkod-based approach integrated in USE [9] as well as the CSP-based approach presented in [2] allow the specification of intervals, but not only for classes, also for associations and attribute domains.

However, even specifying an appropriate interval requires detailed design knowledge. Furthermore, with a growth of the size of the intervals also the search space is increasing which usually results in larger run-times. Thus, also these approaches can benefit from the proposed pre-process as presented in this work. By finding valid problem bounds, there is no need to specify intervals for the selection of object instances, which likely results in an acceleration of the solving process.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we sketched an approach to automatically determine problem bounds that can be used for successive consistency checks in UML/OCL models. For this purpose, dependence relations between the number of instances in the final system state are encoded as LIA constraints that can be solved e.g. with an SMT solver. While encodings expressing the UML associations have been proposed, the incorporation of OCL expressions into the LIA instance has been discussed and is left for future work. However, first results already show that even for class diagrams composed of more than thousand classes, bounds can automatically be determined within a couple of minutes.

By using the proposed approach, no detailed design knowledge is required when applying automatic consistency checks. Further, techniques that support to specify the problem bounds as intervals can benefit from this approach since the search space can be reduced when applying exact problem bounds. Although we described the approach based on UML/OCL class diagrams, it is applicable to other models (e.g. in Alloy) as well since the same concepts are supported just in a different syntax.

The development of a general approach considering both UML and OCL constraints is left for future work. This further applies to a tool integration to evaluate its scalability and the influence to the flow of consistency checking. Further, it is interesting how to obtain problem bounds in dynamic consistency checking such as the number of operation calls to consider.

7. REFERENCES

[1] K. Anastasakis, B. Bordbar, G. Georg, and I. Ray. UML2Alloy: A Challenging Model Transformation. In *Int'l Conf. on Model Driven Engineering Languages and Systems*, pages 436–450. Springer, Oct. 2007.

[2] J. Cabot, R. Clarisó, and D. Riera. Verification of UML/OCL Class Diagrams using Constraint Programming. In *IEEE Int'l. Conf. on Software Testing Verification and Validation Workshop*, pages 73–80, Apr. 2008.

[3] J. Cabot, R. Clarisó, and D. Riera. Verifying UML/OCL Operation Contracts. In M. Leuschel and H. Wehrheim, editors, *Integrated Formal Methods*, volume 5423 of *Lecture Notes in Computer Science*, pages 40–55. Springer, Feb. 2009.

[4] B. Dutertre and L. M. de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In T. Ball and R. B. Jones, editors, *Int'l Conf. on Computer Aided Verification*, volume 4144 of *Lecture Notes in Computer Science*, pages 81–94. Springer, Aug. 2006.

[5] M. Gogolla, F. Büttner, and M. Richters. USE: A UML-based specification environment for validating UML and OCL. *Science of Computer Programming*, 69(1-3):27–34, 2007.

[6] M. Gogolla, M. Kuhlmann, and L. Hamann. Consistency, Independence and Consequences in UML and OCL Models. In *Tests and Proofs*, pages 90–104. Springer, July 2009.

[7] M. Gogolla and M. Richters. Expressing UML Class Diagrams Properties with OCL. In T. Clark and J. Warmer, editors, *Object Modeling with the OCL*, volume 2263 of *Lecture Notes in Computer Science*, pages 85–114. Springer, 2002.

[8] D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, Cambridge, MA, USA, Apr. 2006.

[9] M. Kuhlmann, L. Hamann, and M. Gogolla. Extensive Validation of OCL Models by Integrating SAT Solving into USE. In *Int'l. Conf. on Objects, Models, Components, Patterns*, volume 6705 of *Lecture Notes in Computer Science*, pages 290–306. Springer, June 2011.

[10] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language reference manual*. Addison-Wesley Longman, Essex, UK, Jan. 1999.

[11] M. Soeken, R. Wille, and R. Drechsler. Verifying Dynamic Aspects of UML Models. In *Design, Automation and Test in Europe*, pages 1077–1082. IEEE Computer Society, Mar. 2011.

[12] 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*, pages 1341–1344. IEEE Computer Society, Mar. 2010.

[13] E. Torlak and D. Jackson. Kodkod: A Relational Model Finder. In *TACAS*, volume 4424 of *Lecture Notes in Computer Science*, pages 632–647. Springer, Apr. 2007.

[14] J. Warmer and A. Kleppe. *The Object Constraint Language: Precise modeling with UML*. Addison-Wesley Longman, Boston, MA, USA, Mar. 1999.