Verifying UML/OCL Models with OCL2MSFOL. A Benchmarking Exercise

Carolina Dania and Manuel Clavel

IMDEA Software Institute, Madrid, Spain
{carolina.dania,manuel.clavel}@imdea.org

Abstract. The Object Constraint Language (OCL) is the OMG standard for adding exactness and precision to UML models. OCL is supported by a variety of analysis tools, each one with its own trade-off between expressiveness, termination, automation, and completeness. Recently, a benchmark has been proposed to assess the different techniques for validating and verifying UML/OCL models. In this paper we use this benchmark to asses our own analysis tool, called OCL2MSFOL, and to compare it with other tools for validating and verifying UML/OCL models.

1 Introduction

Modeling, specially software modeling, has traditionally been a synonym for producing diagrams. Most software models consist of a number of bubbles-and-arrows pictures and some accompanying text. The information conveyed by such models tends to be incomplete, informal, imprecise, and sometimes even inconsistent.

Model-Driven engineering (MDE) supports the development of complex software systems by generating software from models. The quality of the generated software depends on the quality of the source models. In particular, if a model does not properly specify the system's intended behavior, one should not expect the generated system to do so either: *Quod natura non dat, Salmantica non praestat* (or, less elegantly said, *garbage in, garbage out*). Experience shows that even when using powerful, high-level modeling languages, it is easy to make logical errors and omissions. Thus, it is critical not only that a modeling language has a well-defined semantics, but also that there is tool support for analyzing the models specified with this language.

The Unified Modeling Language (UML) [8] is a general-purpose, visual modeling language that is used to specify, visualize, construct, and document the artifacts of a software system. Within the UML standard, the Object Constraint Language (OCL) [7] is used to add exactness and precision to UML models. OCL is a strongly-typed, declarative, formal language. To provide support for the validation and verification of UML/OCL models various translations into different formalisms have been proposed (see [6] for a systematic review). In

each case the target formalism imposes a specific trade-off between expressiveness, termination, automation, and completeness. This trade-off is unavoidable since analysing UML/OCL models is in general undecidable [2].

With the intention of helping developers to choose the UML/OCL tool more appropriate for their projects, [3] has proposed a benchmark for assessing validation and verification techniques on UML/OCL models. It includes four models each posing different computational challenges. In this paper we use this benchmark to assess our own UML/OCL verification tool, called OCL2MSFOL [10], and to compare it with other tools for verifying UML/OCL models. OCL2-MSFOL relies on a novel translation of UML/OCL models into many-sorted first-order logic theories that supports the direct use of Satisfiability Modulo theories (SMT) solvers and finite model finders for automatically verifying UML/OCL models.

2 A Benchmark for UML/OCL Models

In this section we briefly recount the benchmark proposed in [3] for comparing UML/OCL analysis tools. It includes four UML/OCL models, namely, CivilStatus, WritesReviews, DisjointSubclasses, and ObjectsAsIntegers, together with a set of questions for each of these models. It is sufficient for our present purpose to consider only the first three models: CivilStatus, WritesReviews, and DisjointSubclasses.[1]

2.1 CivilStatus

Figure 1 shows the first UML class diagram considered in the benchmark. Basically, it models that a person has a name, a gender (either female or male), a civil status (either single, married, divorced, or widowed), and possibly a spouse, and that a person has a husband or a wife or none. The following OCL invariants further constrain this model.[2]

- **attributesDefined:** *A person has a defined name, civil status and gender.*

```
Person.allInstances()->forAll(p|not(p.name.oclIsUndefined())
  and not(p.civStat.oclIsUndefined())
  and not(p.gender.oclIsUndefined())).
```

- **nameIsUnique:** *A person has a unique name.*

```
Person.allInstances()->forAll(p1| Person.allInstances()
  ->forAll(p2|p1 <> p2 implies p1.name <> p2.name)).
```

[1] The fourth model, ObjectAsIntegers, is definitely more "artificial"; furthermore, it requires inductive reasoning, which is out of the scope of both our analysis tool and the tools we are comparing to in this benchmarking exercise.

[2] The benchmark includes an additional constraint about the format of a person's name, which for our present purpose we omit here since it plays no significant role in answering the questions later posed about the model.

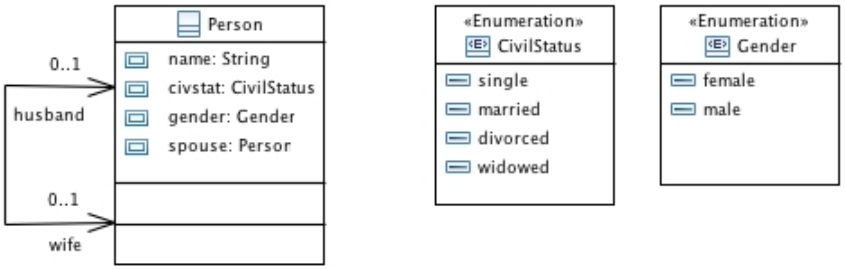

Fig. 1. CivilStatus model

– **femaleHasNoWife:** *A female person does not possess a wife.*

```
Person.allInstances()->forAll(p|p.gender = Gender::female
  implies p.wife.oclIsUndefined()).
```

– **maleHasNoHusband:** *A male person does not possess a husband.*

```
Person.allInstances()->forAll(p|p.gender = Gender::male
  implies p.husband.oclIsUndefined()).
```

– **hasSpouse_EQ_civstatMarried:** *A person has a spouse, if and only if his/her civil status is married.*

```
Person.allInstances()->forAll(p|
  (not(p.spouse.oclIsUndefined())
    implies p.civStat = CivilStatus::married)
  and (p.civStat = CivilStatus::married
    implies not(p.spouse.oclIsUndefined()))).
```

In the benchmark the following questions are posed about this model:

1. **ConsistentInvariants:** Is the model *consistent*? That is, is there at least one instance of the model satisfying all the stated invariants?
2. **Independence:** Are all the invariants *independent*? Or, on the contrary, is there at least one invariant which is a consequence of the conditions imposed by the model and the other invariants?
3. **Consequences:** Is the model bigamy-free? Or, on the contrary, is it possible for a person to have both a wife and a husband?

2.2 WritesReviews

Figure 2 shows the second UML class diagram considered in the benchmark. Basically, it models a simple conference review system. There are papers and

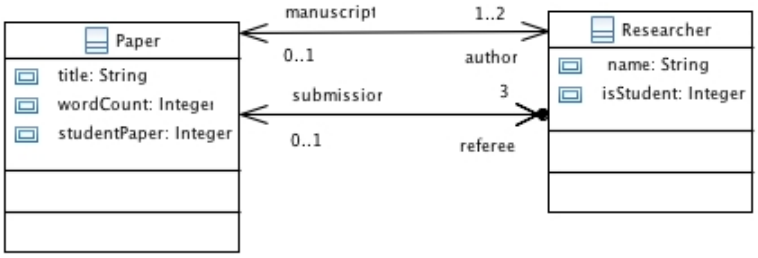

Fig. 2. WritesReviews model

researchers. A paper has a title and a number of words, and can be a studentPaper. A researcher has a name and can be a student. A researcher can be assigned at most one submission to review it and can submit at most one manuscript. A paper can have one or two authors and must be assigned exactly three referees. The following invariants further constrain this model:

- **oneManuscript:** *A researcher must submit one manuscript.*

```
Researcher.allInstances()->forAll(r|
   not(r.manuscript.oclIsUndefined())).
```

- **oneSubmission:** *A research must be assigned one submission.*

```
Researcher.allInstances()->forAll(r|
   not(r.submission.oclIsUndefined())).
```

- **noSelfReviews:** *A paper cannot be refereed by one of its authors.*

```
Researcher.allInstances()->forAll(r|
   not(r.submission.oclIsUndefined())
   implies (r.submission.author->forAll(a|a <> r))).
```

- **paperLength:** *A paper must have at most 10000 words.*

```
Paper.allInstances()->forAll(p|p.wordCount < 10000).
```

- **authorsOfStudentPaper:** *One of the authors of a student paper must be a student.*

```
Paper.allInstances()->forAll(p|(p.studentPaper = 1) implies
  (p.author->exists(x|x.isStudent = 1))).
Paper.allInstances()->forAll(p|
  (p.author->exists(x|x.isStudent  = 1)) implies
  (p.studentPaper = 1)).
```

- **noStudentReviewers:** *Students are not allowed to review any paper.*

```
Paper.allInstances()->forAll(p|
   p.referee->forAll(r|r.isStudent <> 1)).
```

– **limitsOnStudentPapers:** *There must be at least one student paper.*[3]

```
Paper.allInstances()->exists(p|p.studentPaper = 1).
```

In the benchmark the following questions are posed about this model:

1. **InstantiateNonemptyClass:** Can the model be instantiated with non-empty populations for all classes? That is, is there at least one instance of this model with at least one paper and one researcher?
2. **InstantiateNonemptyAssoc:** Can the model be instantiated with non-empty populations for all classes and all associations? That is, is there at least one instance of this model with at least one paper, one researcher, one instance of the manuscript-author association, and one instance of the submission-referee association?
3. **InstantiateInvariantIgnore:** Can the model be instantiated if the invariants **oneManuscript** and **oneSubmission** are ignored?

2.3 DisjointSubclasses

Figure 3 shows the third UML class diagram considered in the benchmark. There are four classes: A, B, C, and D. Class B and C inherit from class A, while class D inherits from both class B and class C. The following invariant further constrains this model:

– **disjointBC:** *Class B and class C are disjoint.*

```
C.allInstances()->forAll(x|B.allInstances()->forAll(y|x<>y))
```

In the benchmark the following questions are posed about this model:

1. **InstantiateDisjointInheritance:** Can all classes be populated? That is, is there at least one instance of this model with at least one element of each class? In particular, is there at least one instance of this model with at least one element of class D?
2. **InstantiateMultipleInheritance:** Can class D be populated if the constraint **disjointBC** is ignored?

[3] The benchmark requires also that the number of student papers should be less than 5. For the sake of simplicity, we only consider here the first part of the constraint since the second one plays no significant role in answering the questions later posed about the model.

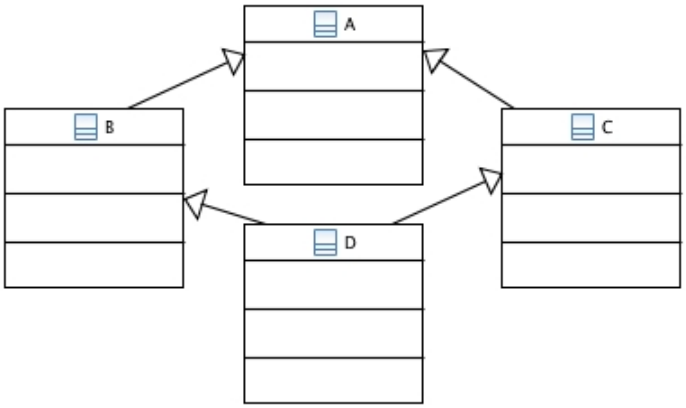

Fig. 3. Disjointsubclasses model

3 The OCL2MSFOL Tool

OCL2MSFOL [10] is a UML/OCL verification tool. It relies on a novel translation of UML/OCL models into many-sorted first-order logic that supports the direct use of Satisfiability Modulo Theories (SMT) solvers and finite model finders for automatically verifying UML/OCL models.

For the purpose of this benchmarking exercise, we use CVC4 [1] as the OCL2MSFOL's back-end SMT solver. Let $\mathcal{M}$ be a UML/OCL model and let o2f($\mathcal{M}$) be its translation into many-sorted first-order logic, as generated by the OCL2MSFOL's underlying mapping. In general, when we check o2f($\mathcal{M}$) with CVC4, we can expect one of the following three answers:

- **sat**: it means that there exists at least one *valid* instance of $\mathcal{M}$. A valid instance of $\mathcal{M}$ is an instance of the UML class diagram in $\mathcal{M}$ that also satisfies all the OCL invariants in $\mathcal{M}$.
- **unsat**: it means that no valid instance of $\mathcal{M}$ exists.
- **unknown**: it means that the check is inconclusive.[4]

As for completeness, CVC4 —or any SMT solver for that matter— cannot guarantee that, if a valid instance of $\mathcal{M}$ exists, it will find it and return **sat**, nor it can guarantee that, if no valid instance of $\mathcal{M}$ exists, it will return **unsat**. Fortunately, CVC4 includes a finite model finder [9] that, although limited to SMT formulas with quantifiers ranging over free sorts, it seems to be all we need in the context of OCL2MSFOL-based verification of UML/OCL models. This is so because: (i) OCL invariants are translated by OCL2MSFOL into quantified formulas over free sorts, where these free sorts correspond to the classes

[4] Recall that SMT solvers cannot be complete when dealing with quantifiers and therefore they may return *unknown* when they fail, after some predetermined amount of time, to prove that the input problem is unsatisfiable.

in the given UML/OCL model; and (ii) when verifying UML/OCL models we are interested in instances of the given UML/OCL model that contain a finite number of objects for each class, and therefore, in the context of OCL2MSFOL-based verification, we are interested in interpretations of the free sorts as finite domains.

Comparison with USE and EMFtoCSP

USE [4] and EMFtoCSP [5] are two different tools for automatically verifying and validating UML/OCL models. While USE checks UML/OCL consistency using enumeration and SAT-based techniques, EMFtoCSP turns a UML/OCL consistency problem into a constraint satisfaction problem (CSP) and uses constraint solvers to solve it. In both cases, the user is required to specify *ranges* for the class and association extents and for the attribute domains. The fact that both USE and OCL2MSFOL operate on *bounded* search state spaces implies, on the one hand, that, when there is a valid instance of a given UML/OCL model *within* the selected range, both USE and EMFtoCSP will find it —assuming that the selected range is sufficiently small, of course. But, on the other hand, it also means that, when either USE or EMFtoCSP communicates to the user that no valid instance of a model has been found, this answer is *inconclusive*, since a valid instance may still exist outside of the the selected range.

4 Handling the Benchmark in OCL2MSFOL

In what follows we use the benchmark discussed in Section 2 to assess OCL2-MSFOL. The results of USE and EMFtoCSP on this same benchmark were reported in [3]. For the sake of comparison with OCL2MSFOL, the results obtained by USE and EMFtoCSP are entirely analogous, and we will draw explicit comparisons only with the former.

All OCL2MSFOL checks were run on a laptop computer, with an Intel Core i7 processor running at 1.8GHz with 4Gb of RAM. As back-end SMT solver, we use CVC4 (version 1.5-prerelease) with the option `finite-model-find`. We denote this configuration as CVC4^{fmf}.

4.1 CivilStatus

In Table 1 we show the results of analysing, using OCL2MSFOL, the questions posed in the benchmark about CivilStatus. In particular,

- **ConsistentInvariants:**. CVC4^{fmf} finds a valid instance of CivilStatus and returns **sat**. Thus, we can conclude that the model is consistent.
- **Independence:** For each of the five invariants, CVC4^{fmf} finds a valid instance of a modified version of CivilStatus, where the given invariant is negated while the other are still affirmed, returning **sat** in each case. Thus, we can conclude that the invariants are independent.

– **Consequences:** CVC4$^{\text{fmf}}$ returns **unsat** when the following invariant is added to CivilStatus:

```
Person.allInstances()->exists(p|
   not(p.husband.oclIsUndefined())
   and not(p.wife.oclIsUndefined())).
```

Thus, we can conclude that the model is bigamy-free.

Question	*Answer*	*Time (in secs)*	*Remarks*
ConsistentInvariants	**sat**	0.08	
Independence	**sat**	0.29	For invariant 1
		0.40	For invariant 2
		0.32	For invariant 3
		0.34	For invariant 4
		0.16	For invariant 5
Consequences	**unsat**	0.24	

Table 1. Analyzing CivilStatus with OCL2MSFOL

We can now compare these results with the ones obtained by analyzing CivilStatus using USE, as reported in [3]. In particular,

– **ConsistentInvariants:** Selecting as search state space the instances of CivilStatus with exactly one male person and one female person, USE finds a valid instance of CivilStatus. Thus, as with OCL2MSFOL, we can conclude that the model is consistent.
– **Independence:** Selecting as search state space the instances of CivilStatus with exactly one male person and one female person, for each of the five invariants, USE finds an instance of CivilStatus such that this invariant is not satisfied while the others are satisfied. Thus, as with OCL2MSFOL, we can conclude that the invariants are independent.
– **Consequences:** Selecting as search state space the instances of CivilStatus with at most three persons, USE is not able to find an instance of CivilStatus which is bigamy-free. Notice that this answer is *inconclusive*, since a bigamy instance of CivilStatus may still exist outside of the selected range. On the contrary, the answer provided by OCL2MSFOL guarantees that CivilStatus is bigamy-free.

4.2 WritesReviews

In Table 2 we show the results of analysing, using OCL2MSFOL, the questions posed in the benchmark about WritesReviews. In particular,

– **InstantiateNonemptyClass:** CVC4$^{\text{fmf}}$ returns **unsat** when the following invariants are added to WritesReviews:

```
Paper.allInstances()->notEmpty().
Researcher.allInstances()->notEmpty()
```

Thus, we can conclude that there is no valid instance of WritesReviews with at least one paper and one researcher.

- **InstantiateNonemptyAssoc:** As expected, CVC4$^{\text{fmf}}$ also returns **unsat** when the following invariants are added to WritesReviews:

```
Paper.allInstances()->notEmpty().
Researcher.allInstances()->notEmpty().
Paper.allInstances()->exists(p|p.author->notEmpty()).
Paper.allInstances()->exists(p|p.referee->notEmpty())
```

Thus, we can conclude that there is no valid instance of WritesReviews with at least one paper, one researcher, one instance of the manuscript-author association, and one instance of the submission-referee association.

- **InstantiateInvariantIgnore:** CVC4$^{\text{fmf}}$ returns **sat** when the invariants **oneManuscript** and **oneSubmission** are removed from WritesReviews. Thus, we can conclude that, if the invariants **oneManuscript** and **oneSubmission** are ignored, there exists at least one valid instance of WritesReviews.

Question	*Answer*	*Time (in secs)*	*Remarks*
InstantiateNonemptyClass	**unsat**	0.66	
InstantiateNonemptyAssoc	**unsat**	1.70	
InstantiateInvariantIgnore	**sat**	0.22	

Table 2. Analyzing WritesReviews with OCL2MSFOL

We can now compare these results with the ones obtained by analyzing WritesReviews using USE, as reported in [3]. In particular,

- **InstantiateNonemptyClass:** Selecting as search state space the instances of WritesReviews with at most four researchers and four papers, USE is not able to find a valid instance of WritesReviews with at least one researcher and one paper. Again, notice that this answer is *inconclusive*, since a valid instance of WritesReviews with at least one researcher and one paper may still exist outside the selected range. On the contrary, the answer provided by OCL2MSFOL guarantees that no valid instance of WritesReviews exists with at least one researcher and one paper.
- **InstantiateNonemptyAssoc:** The result is exactly as in the case of **InstantiateNonemptyClass**.
- **InstantiateInvariantIgnore** Having removed from WritesReviews the constraints **oneManuscript** and **oneSubmission**, and selecting as search state space the instances of WritesReviews with exactly one paper and at most

four researchers, USE finds a valid instance of WritesReviews. Thus, as with OCL2MSFOL, we can conclude that, if the constraints **oneManuscript** and **oneSubmission** are ignored, there is at least one valid instance of WritesReviews.

4.3 DisjointSubclasses

In Table 3 we show the results of analysing, using OCL2MSFOL, the questions posed in the benchmark about DisjointSubclasses. In particular,

- **InstantiateDisjointInheritance:** CVC4$^{\text{fmf}}$ returns **unsat** when the following invariant is added to DisjointSubclasses:

  ```
  D.allInstances()->notEmpty()
  ```

  Thus, we can conclude that there is no valid instance of DisjointSubclasses with at least one element of class D.
- **InstantiateMultipleInheritance:** CVC4$^{\text{fmf}}$ returns **sat** when the following invariant is added to DisjointSubclasses

  ```
  D.allInstances()->notEmpty()
  ```

  and at the same time the invariant **disjointBC** is removed from DisjointSubclasses. Thus, we can conclude that, if the constraint **disjointBC** is ignored, there is at least one instance of DisjointSubclasses with at least one element of class D.

Question	*Answer*	*Time (in secs)*	*Remarks*
InstantiateDisjointInheritance	**unsat**	0.08	
InstantiateMultipleInheritance	**sat**	0.06	

Table 3. Analyzing DisjointSubclasses with OCL2MSFOL

We can now compare these results with the ones obtained by analyzing DisjointSubclasses using USE, as reported in [3]. In particular,

- **InstantiateDisjointInheritance:** Selecting as search state space the instances of DisjointSubclasses with exactly one element of class A, one of class B, one of class C, and one of class D, USE is not able to find a valid instance of DisjointSubclasses. Notice that this answer is again *inconclusive*, since a valid instance of DisjointSubclasses may still exist outside of the selected range. On the contrary, the answer provided by OCL2MSFOL guarantees that no valid instance of DisjointSubclasses exists at all with at least one element of class D.

– **InstantiateMultipleInheritance:** Having eliminated from DisjointSubclasses the constraint **disjointBC**, and selecting as search state space the instances of DisjointSubclasses with exactly one element of class A, one of class B, one of class C, and one of class D, and removing from DisjointSubclasses the contraint **DisjointBC**, USE finds an instance of DisjointSubclasses. Therefore, as with OCL2MSFOL, we can conclude that, if the constraint **DisjointBC** is ignored, there is at least one instance of DisjointSubclasses with at least one element of class D.

5 Conclusions

We have reported the results of testing our own UML/OCL verification tool, called OCL2MSFOL [10], on a recently proposed benchmark [3]. We have also compared our tool with two other UML/OCL verification tools, namely, USE [4] and EMFtoCSP [5], on the same benchmark. As illustrated by the benchmark tests, OCL2MSFOL can provide a more definitive answer when the UML/OCL model under consideration is *inconsistent*, while freeing the modeler from the responsibility of "guessing it right" when the model under consideration is *consistent*. This is due to the fact that OCL2MSFOL relies on the use of SMT solvers and finite model finders for checking UML/OCL models. On the contrary, USE [4] and EMFtoCSP [5] rely, respectively, on enumeration and SAT-based techniques and on constraint solving techniques, which, by definition, operates on *bounded* search spaces.

References

1. C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. CVC4. In *Proceedings of the CAV 11.*, pages 171–177, Berlin, Heidelberg, 2011. Springer-Verlag.
2. D. Berardi, D. Calvanese, and G. De Giacomo. Reasoning on UML class diagrams. *Artif. Intell.*, 168(1-2):70–118, 2005.
3. M. Gogolla, F. Büttner, and J. Cabot. Initiating a benchmark for UML and OCL analysis tools. In *TAP 2013, Budapest, Hungary. Proceedings*, volume 7942 of *LNCS*, pages 115–132. Springer, 2013.
4. M. Gogolla, F. Büttner, and M. Richters. USE: A UML-based specification environment for validating UML and OCL. *Sci. Comput. Program.*, 69(1-3):27–34, 2007.
5. C. A. González, F. Büttner, R. Clarisó, and J. Cabot. EMFtoCSP: a tool for the lightweight verification of EMF models. In S. Gnesi, S. Gruner, N. Plat, and B. Rumpe, editors, *Proceedings of FormSERA 2012, Zurich, Switzerland, June 2, 2012*, pages 44–50. IEEE, 2012.
6. C. A. González and J. Cabot. Formal verification of static software models in MDE: A systematic review. *IST*, 56(8):821–838, 2014.
7. Object Management Group. Object Constraint Language specification version 2.4. Technical report, OMG, 2014. `http://www.omg.org/spec/OCL/2.4`.
8. Object Management Group. Unified Modeling Language specification version 2.5. Technical report, OMG, 2015. `http://www.omg.org/spec/UML/2.5`.

9. A. Reynolds, C. Tinelli, A. Goel, and S. Krstic. Finite model finding in SMT. In N. Sharygina and H. Veith, editors, *CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *LNCS*, pages 640–655. Springer, 2013.
10. WebOCL2MSFOL Project, 2016. `http://software.imdea.org/~dania/tools/ocl2msfol`.