QMaxUSE: A Query-based Verification Tool for UML Class Diagrams with OCL Invariants

Hao Wu (✉)

Computer Science Department, Maynooth University, Maynooth, Ireland
haowu@cs.nuim.ie

Abstract. Verifying whether a UML class diagram annotated with Object Constraint Language (OCL) constraints is consistent involves finding valid instances that provably meet its structural and OCL constraints. Recently, many tools and techniques have been proposed to find valid instances. However, they often do not scale well when the number of OCL constraints significantly increases. In this paper, we present a new tool called QMaxUSE that is capable of automatically verifying a large number of OCL invariants. QMaxUSE works by decomposing them into a set of different queries. It then uses an SMT solver to concurrently verify each query and pinpoints conflicting OCL invariants. Our evaluation results suggest that QMaxUSE can offer up to 30x efficiency improvement in verifying UML class diagrams with a large number of OCL invariants.

1 Introduction

Verifying the consistency of a UML class diagram annotated with OCL constraints is a challenging task [1,2,3]. This is because it requires finding an instance satisfying both structural and OCL constraints at the same time. To tackle this challenge, many tools and techniques have been proposed [4,5,6,7,8]. However, most of these tools and techniques do not scale well when the number of OCL invariants significantly increases [9,10,11,12,13,5,14,15,16]. These tools often time out or cannot pinpoint the conflicting OCL invariants that cause a UML class diagram to become inconsistent.

In this paper, we present a new tool QMaxUSE that is capable of verifying a large number of complex OCL invariants in an efficient manner. This is achieved by two distinct features provided within QMaxUSE. (1) a query language that allows users to select parts of a UML class diagram to be verified. (2) a new specialised algorithm that is able to decompose a UML class diagram that has a large number of complex OCL invariants into different queries. These queries can then be verified concurrently via efficient SMT solving. The detailed explanation of our approach can be found in [17].

Related Work. Verifying the consistencies of a UML class diagram has gained much attention in recent years and many approaches and tools are proposed. A UML class diagram can be considered as a graph, so graph-based approaches are naturally employed for reasoning about consistencies [18,19,20,7,21]. Semeráth

E. B. Johnsen and M. Wimmer (Eds.): FASE 2022, LNCS 13241, pp. 310–317, 2022.
https://doi.org/10.1007/978-3-030-99429-7_17

et al. proposed a new graph solver that is able to generate much larger number of objects [22]. Their approach utilises a combination of multiple advanced graph-based and SAT-solving techniques to achieve large-scale graphs generation. On the other hand, many tools incorporate logic solvers to support OCL constraints solving [14,16,23,24,25]. However, many of them do not scale well and cannot pinpoint conflicting OCL constraints when a UML class diagram is inconsistent. Our goal here is to provide an open-source tool that is capable of not only locating conflicting OCL constraints but also preserves high-performance when the number of OCL constraints significantly increases.

2 Architecture

QMaxUSE is fully automatic and integrated with the USE modelling tool [26]. Currently, it is command-line based and can be run under operating system Windows 10 (x64), Ubuntu 20.04 (x64) and Mac OS Big Sur(x64). QMaxUSE is implemented in Java. It consists of nearly 33k lines of code, and approximately $3.5k$ lines of code are dedicated to its algorithms. The latest version of QMaxUSE is available at [27]:

https://github.com/classicwuhao/qmaxuse

The architecture of QMaxUSE is shown in Figure 1. Overall, it has four layers: front-end, query engine, translation and solver.

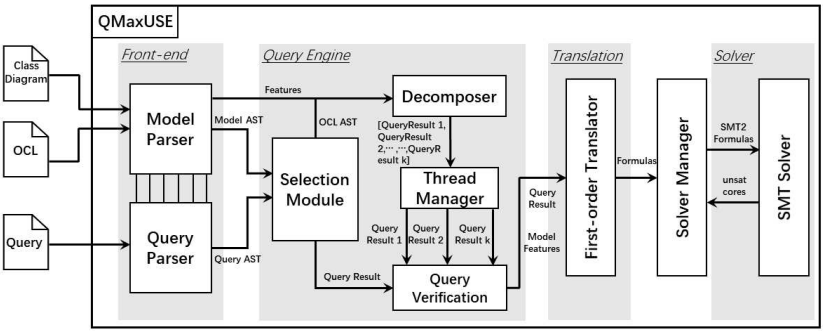


Fig. 1. The overall architecture of QMaxUSE.

Front-end. At the front-end layer, QMaxUSE uses parsers from USE to generate ASTs (abstract syntax trees) for a class diagram and OCL invariants. QMaxUSE provides a simple query language that allows users to choose a part of a class diagram and its OCL invariants to be verified. To parse a query issued by a user, we have designed and implemented a query parser. This parser is able to read multiple queries simultaneously in a specification file and produces corresponding ASTs.

Query Engine. QMaxUSE's query engine uses a set of selection algorithms to traverse the ASTs generated from the front-end layer to produce a query result. A query result essentially contains a set of classes, attributes, associations and

OCL invariants to be verified. At this layer, QMaxUSE also provides a *specialised algorithm* (Decomposer) that is able to decompose a class diagram along with OCL invariants into a set of different queries. These queries can then be verified concurrently using a query verification procedure.

Translation. At the translation layer, QMaxUSE uses a *first-order translator* to translate a query into a set of first-order formulas that can be verified by the SMT solver. The translation here is similar to the one described in [8]. We use uninterpreted functions to encode classes or attributes and linear integer inequalities to capture the multiplicities at an association-end. For an OCL invariant, we traverse its AST and generate an SMT formula by using a combination of first-order theories.

Solver. We have designed a new interface (*SolverManager*) to optimise the interaction between QMaxUSE and the SMT solver. This interface can reduce extra overhead between our first-order translator and an SMT solver by minimising the number of APIs calls. Currently, QMaxUSE uses Z3 as its default SMT solver and this new interface easily allows us to plug in other SMT solvers [28].

3 Design

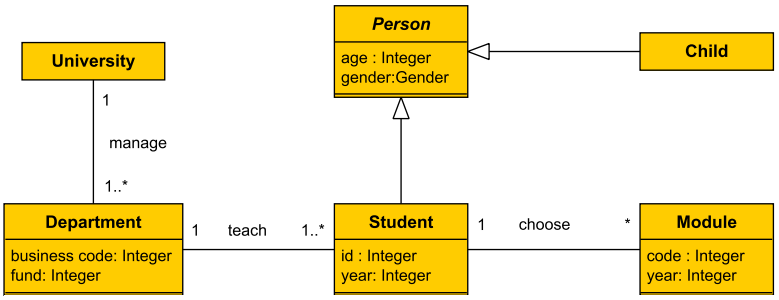
3.1 Query

QMaxUSE allows a user to verify a particular set of features of a UML class diagram through a query language. A *query* expression accepted by QMaxUSE must use a *select* statement. It allows users to choose multiple features along with OCL invariants from a UML class diagram. A *feature* here may include a *class*, an *attribute*, an *association* or an OCL *invariant*. For example, the following query (*query* 1) first selects the *University*, *Department*, *Student* and *Module* class, an association *teach* along with the invariant defined under the *Module* class from the UML class diagram in Figure 2.

query 1 : select University, Student.*, Department:teach:Student with Student::inv2, Module::*

Notably, we allow users to use a wild character $*$ to represent a set of features under a specified classifier. Further, it is quite common that an OCL invariant may use features from other classes in its expression. Hence, our selection algorithm implicitly selects these features during the execution of a query. Thus, *query* 1 also selects the *Person* class from Figure 2 since *inv*2 defined under the *Student* class imposes a constraint on the *age* attribute that is inherited from the *Person* class.

For each query issued by a user, QMaxUSE launches a verification procedure that is able to verify the consistencies of the collected features. This verification procedure casts the set of collected features to a set of SMT formulas that can be checked by an SMT solver. If the formulas are not satisfied, QMaxUSE reports inconsistencies by pinpointing the OCL invariants that cause conflicts. For example, QMaxUSE reports that there is a conflict between OCL invariant *inv*1 and *inv*2 after verifying the following query (*query* 2). It shows that both




```
context Person
inv1: Person.allInstances()->forAll(p|p.age>0 and p.age<18)

context Student
inv2: self.age>18
inv3: self.year>=1 and self.year<=6
inv4: Student.allInstances()->forAll(s1,s2:Student|s1<>s2 implies s1.id <> s2.id)
inv5: Student.allInstances()->forAll(s|s.modules->forAll(m|s.year=m.year))
inv6: Student.allInstances()->exists(s|s.year=6) and Student.allInstances()->exists(s|s.year<6)
inv7: Student.allInstances()->forAll(s|s.modules->notEmpty())

context Module
inv8: self.year>=1 and self.year<=5
```

Fig. 2. A UML class diagram with the 8 OCL class invariants shows how the students in each department can choose multiple modules to study.

*inv*1 and *inv*2 can make the *Student* class impossible to instantiate. Figure 3 shows a screenshot of QMaxUSE after executing *query* 2.

query 2 : select Person.*, Student.* with Person::inv1, Student::inv2

3.2 Concurrent Verification

QMaxUSE has a crafted algorithm that is designed for performing concurrent verification on UML class diagrams with a large number of OCL invariants. The main idea of this algorithm is that it is able to decompose a large number of complex OCL invariants into different queries. For each query, it launches a thread of verification procedure to verify that query. In this way, QMaxUSE is able to shift solving a large number of complex formulas from a single run into multiple simultaneous runs on a collection of much smaller and less complex formulas. Therefore, it is particularly powerful when the number of OCL invariants grows significantly.

A high-level structure of this dedicated algorithm is shown in Algorithm 1 [17]. This algorithm takes a UML class diagram annotated with OCL invariants (denoted as *model*) as its input and outputs a set S that contains all possible conflicting features. It first employs a novel decomposition algorithm to decompose a *model* into different parts and produces a query for each part of this model. It then executes each query and produces a new query result by explicitly choosing those features that are used by an OCL invariant expression in

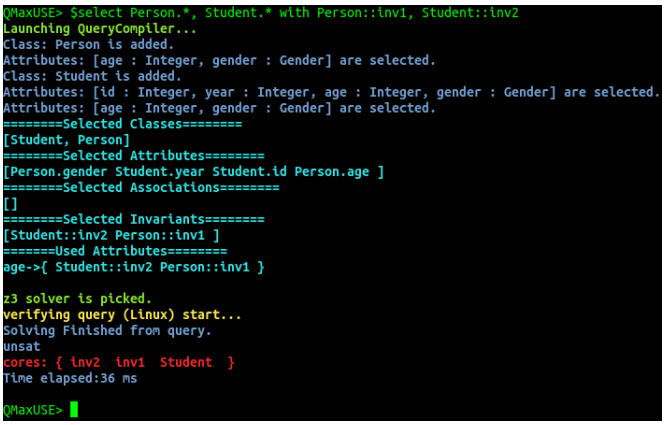


Fig. 3. A screenshot of running *query* 2 in QMaxUSE.

a query. Once the set of query results are generated, Algorithm 1 launches a number of threads to verify the formulas (Φ) that encode query results. If the Φ are not satisfied, then this means that there must be conflicts. Finally, our algorithm extracts those conflicting features and saves them into the set S.

Algorithm 1: ConcurrentVerification

Input : A UML class diagram annotated with OCL invariants (*model*)
Output: A set of conflicting features cause inconsistencies (S).

$R \leftarrow \emptyset \wedge S \leftarrow \emptyset$;
$Q \leftarrow Decompose(model)$; /*produce a set of queries Q*/
foreach $q \in Q$ **do**
  $q_r = q.execute()$; /* create a new query result q_r*/
  /* add features used in an OCL invariant into a query result q_r*/
  foreach $inv \in q$ **do**
    $q_r.add(inv.classes(), inv.attributes(), inv.associations(), inv)$;
  end
  $R.add(q_r)$;
end
/* verify *model* with $|R|$ number of threads. */
foreach $q_r \in R$ **do**
  $\Phi \leftarrow Translate(q_r)$; /*cast q_r to SMT formulas*/
  $ThreadManager.start(QueryVerification(\Phi, S))$;
  /*check satisfiability of Φ and saves each conflict occurred in Φ in the set S*/
end
return S;

	Name	OCL Structure Size				MaxUSE [12] (sec)	QMaxUSE (sec)	
		Invs	Nodes	Quant	Op	Time	Threads	Time
Part A	A1	6	30	3	9	0.38	2	0.364
	A2	7	52	8	1	0.148	1	0.087
	A3	1	7	2	1	0.174	3	0.426
	A4	8	73	7	18	0.204	3	0.241
Part B	B3	68	430	9	111	131.23	42	4.604
	B4	90	599	23	152	159.378	68	7.151
	C4	98	698	69	137	TO	68	8.111
	C5	156	1008	100	184	TO	90	114.41
	B5	136	925	44	228	TO	90	118.64
	D4	101	753	102	163	TO	56	8.211
	D5	166	1143	131	225	TO	95	14.026
	E3	37	403	31	102	59.535	27	2.587
	E4	105	985	56	246	TO	42	4.464
	E5	167	1134	68	325	TO	45	3.653

Table 1. Evaluation results. Invs=number of OCL invariants, Nodes=size of invariant ASTs, Quant=number of quantifiers, Op=number of operators. TO= Timeout (20min), MaxUSE=QMaxUSE without query and concurrent verification support.

4 Results

We use a benchmark from [8] to show the size and the complexities of OCL invariants QMaxUSE can handle. This benchmark has two parts. Part A only covers a small number of toy examples from [29] and Part B covers a wide range of OCL language features including: nested quantifiers, collections, logical/arithmetic operations and navigations. In particular, Part B contains a large number of complex and conflicting OCL invariants. Table 1 summarises part of our evaluation results for QMaxUSE [1]. The evaluation is carried out on an Intel(R) Core (TM) machine that has six 2.8GHz cores with 16G memory. The underlying SMT solver is the Z3 SMT solver (version 4.8.10). As it can seen that QMaxUSE is able to handle much larger size of OCL invariants. It is able to gain upto 30x efficiency in improvement in verifying large number of complex OCL invariants. For example, it takes 131.23 seconds to verify $B3$ in Group B without using our query and concurrent techniques while QMaxUSE is able to finish its verification in just 4.6 seconds.

5 Conclusion

In this paper, we have presented our latest verification tool QMaxUSE. We believe that QMaxUSE can add significant value in modelling community for two reasons. (1) Users now are able to use QMaxUSE to incrementally verify different parts of their class diagrams by issuing different queries. (2) Our preliminary evaluation results indicate that QMaxUSE can scale well on a large number of complex OCL invariants because of our concurrent verification algorithm.

[1] The complete benchmark is packed within QMaxUSE release files.

References

1. Berardi, D., Calvanese, D., Giacomo, G.D.: Reasoning on UML class diagrams is EXPTIME-hard. In: International Workshop on Description Logics. (2003)
2. Berardi, D., Calvanese, D., Giacomo, G.D.: Reasoning on UML class diagrams. Artificial Intelligence **168**(1–2) (2005) 70–118
3. Queralt, A., Teniente, E.: Reasoning on uml class diagrams with ocl constraints. In: Conceptual Modeling, Springer (2006) 497–512
4. Queralt, A., Artale, A., Calvanese, D., Teniente, E.: OCL-Lite: Finite reasoning on UML/OCL conceptual schemas. Data & Knowledge Engineering **73** (2012) 1–22
5. Dania, C., Clavel, M.: Ocl2msfol: A mapping to many-sorted first-order logic for efficiently checking the satisfiability of ocl constraints. In: International Conference on Model Driven Engineering Languages and Systems, ACM (2016) 65–75
6. Maraee, A., Balaban, M.: Efficient reasoning about finite satisfiability of UML class diagrams with constrained generalization sets. In: 3rd European Conference Model Driven Architecture, Springer (2007) 17–31
7. Balaban, M., Maraee, A.: Finite Satisfiability of UML Class Diagrams with Constrained Class Hierarchy. ACM Transactions on Software Engineering and Methodology **22**(3) (2013) 24:1–24:42
8. Wu, H., Farrell, M.: A formal approach to finding inconsistencies in a metamodel. Software and Systems Modeling (2021)
9. González Pérez, C.A., Buettner, F., Clarisó, R., Cabot, J.: EMFtoCSP: A tool for the lightweight verification of EMF models. In: International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches, IEEE (2012) 44–50
10. Kuhlmann, M., Gogolla, M.: From uml and ocl to relational logic and back. In: International Conference on Model Driven Engineering Languages and Systems, Springer (2012) 415–431
11. Queralt, A., Artale, A., Calvanese, D., Teniente, E.: Ocl-lite: Finite reasoning on UML/OCL conceptual schemas. Data & Knowledge Engineering **73** (2012) 1 – 22
12. Wu, H.: Maxuse: A tool for finding achievable constraints and conflicts for inconsistent UML class diagrams. In: Integrated Formal Methods, Springer (2017) 348–356
13. Wille, R., Soeken, M., Drechsler, R.: Debugging of inconsistent UML/OCL models. In: Design, Automation Test in Europe, IEEE (2012) 1078–1083
14. Wu, H., Monahan, R., Power, J.F.: Exploiting attributed type graphs to generate metamodel instances using an SMT solver. In: International Symposium on Theoretical Aspects of Software Engineering, IEEE (2013) 175–182
15. Wu, H.: Generating metamodel instances satisfying coverage criteria via SMT solving. In: International Conference on Model-Driven Engineering and Software Development, IEEE (2016) 40–51
16. Soeken, M., Wille, R., Drechsler, R.: Verifying dynamic aspects of uml models. In: Design, Automation Test in Europe, IEEE (March 2011) 1–6
17. Wu, H.: A query-based approach for verifying UML class diagrams with OCL invariants (to appear). In: 18th European Conference on Modelling Foundations and Applications
18. Ehrig, K., Küster, J.M., Taentzer, G.: Generating instance models from meta models. Software and Systems Modeling **8**(4) (2009) 479–500
19. Hoffmann, B., Minas, M.: Defining models - meta models versus graph grammars. Electronic Communications of the EASST **29** (2010) 1–14

20. Hoffmann, B., Minas, M.: Generating instance graphs from class diagrams with adaptive star grammars. In: International Workshop on Graph Computation Models, Electronic Communications of the EASST (2011)
21. Maraee, A., Balaban, M.: Removing redundancies and deducing equivalences in UML class diagrams. In: International Conference Model-Driven Engineering Languages and Systems, Springer (2014) 235–251
22. Semeráth, O., Nagy, A.S., Varró, D.: A graph solver for the automated generation of consistent domain-specific models. In: Proceedings of the 40th International Conference on Software Engineering. ICSE '18, Association for Computing Machinery (2018) 969–980
23. Anastasakis, K., Bordbar, B., Georg, G., Ray, I.: UML2Alloy: A challenging model transformation. In: International Conference on Model Driven Engineering Languages and Systems, Springer (2007) 436–450
24. Semeráth, O., Vörös, A., Varró, D.: Iterative and incremental model generation by logic solvers. In: 19th International Conference on Fundamental Approaches to Software Engineering, Springer (2016) 87–103
25. Kuhlmann, M., Gogolla, M.: Strengthening SAT-based validation of UML/OCL models by representing collections as relations. In: Modelling Foundations and Applications. Volume 7349. Springer (2012) 32–48
26. Gogolla, M., Büttner, F., Richters, M.: USE: A UML-based specification environment for validating UML and OCL. Science of Computer Programming **69**(1-3) (2007) 27–34
27. QMaxUSE: https://doi.org/10.5281/zenodo.5804509
28. De Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer (2008) 337–340
29. Gogolla, M., Büttner, F., Cabot, J.: Initiating a benchmark for UML and OCL analysis tools. In: International Conference on Tests and Proofs, Springer (2013) 115–132