Towards Domain Refinement For UML/OCL Bounded Verification

Robert Clarisó[1], Carlos A. González[2], and Jordi Cabot[1,3]

[1] Universitat Oberta de Catalunya, Spain
rclariso@uoc.edu
[2] AtlanMod team (Inria, Mines Nantes, LINA), France
carlos.gonzalez@mines-nantes.fr
[3] ICREA, Spain
jordi.cabot@icrea.cat

Abstract. Correctness of UML class diagrams annotated with OCL constraints can be checked using bounded verification, e.g. SAT solvers. Bounded verification detects faults efficiently but, on the other hand, the absence of faults does not guarantee a correct behavior outside the bounded domain. Hence, choosing suitable bounds is a non-trivial process as there is a trade-off between the verification time (faster for smaller domains) and the confidence in the result (better for larger domains). Unfortunately, existing tools provide little support in this choice.
This paper presents a technique that can be used to (i) automatically infer verification bounds whenever possible, (ii) tighten a set of bounds proposed by the user and (iii) guide the user in the bound selection process. This approach may increase the usability of UML/OCL bounded verification tools and improve the efficiency of the verification process.

1 Introduction

Software systems can be described at a high level of abstraction using graphical diagrams such as UML class diagrams. In order to increase their precision and expressiveness, these models can be annotated with textual constraints written in the Object Constraint Language (OCL).

UML/OCL models may contain defects [12], e.g. inconsistent or redundant integrity constraints. Checking the correctness of a UML/OCL model is a complex problem, and in general, undecidable [4]. A popular strategy among verification tools for UML/OCL [10] is *bounded verification*: limiting the search space to a finite domain, e.g. by defining a maximum population for each class and restricting the potential values of attributes. This allows an efficient and automatic analysis without compromising the expressiveness of the modeling language. However, in return the results of the analysis are only meaningful within the defined bounds.

Unfortunately, current tools provide little support in the choice of bounds. Inadequate bounds will cause the analysis to miss defects (if they are too narrow) or to become to slow to be practical (if they are too wide). In this paper, we present a technique that can assist users of UML/OCL bounded verification tools

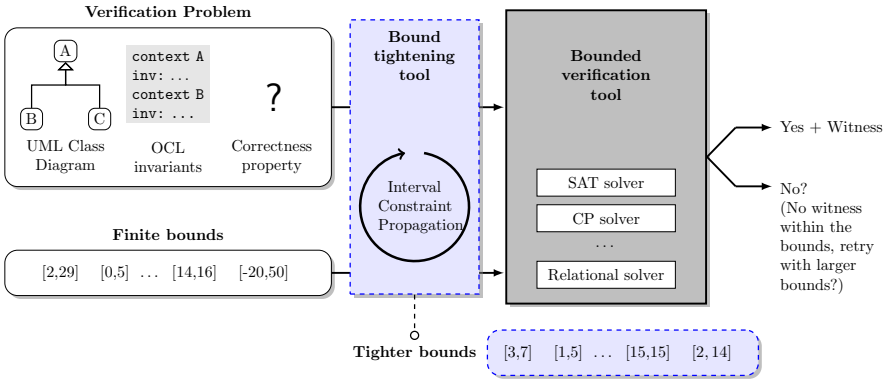

Fig. 1. Typical flow with a bounded verification tool and the role of bound tightening

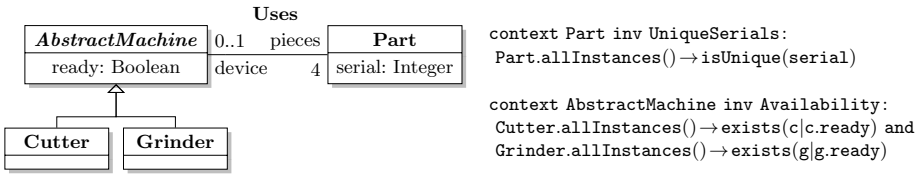

Fig. 2. UML/OCL class diagram used as example

to effectively set the boundaries of the search space. This approach starts from a set of initial bounds and takes advantage of all implicit and explicit constraints in the model to tighten those bounds as much as possible. To this end, an efficient technique called interval constraint propagation, which does *not* require solving the verification problem, is used to discard unproductive values from domain bounds (see Figure 1). We report the performance gains using the USE model validator plug-in [11] as the bounded verification tool.

Example 1. Let us consider the class diagram from Figure 2 describing the relationship between *machines* and *parts*. Graphical constraints such as association end multiplicities define constraints on the valid populations for classes and associations, e.g. there are 4 parts per machine. OCL invariants define additional restrictions on these populations and the domains of attributes. For instance, the invariants in the example require serial numbers to be unique (`UniqueSerial`) and at least one machine of each type to be non-idle (`Availability`).

These constraints can be used to automatically infer bounds without any user intervention, e.g. invariant `Availability` imposes a lower bound of 1 for classes `Cutter` and `Grinder`, of 8 for class `Part` and 8 for association `Uses`. However, this inference is most effective when used to refine partial bound information provided by a designer. For instance, just by assuming a limit of 10 serial numbers, we can infer that there is exactly 1 `Cutter` and 1 `Grinder`, between 8 and 10 parts and at most 8 links among machines and parts.

Paper organization: Section 2 describes the bound tightening method and Section 3 presents experimental results. Section 4 covers the related work. Finally, conclusions and future work are presented in Section 5.

Table 1. Definition of the CSP used to tighten verification bounds

Vars (V)	**Domains** (D)	**Constraints** (C)
A variable cl for each class	Potential number of objects in class cl, either $[0, \infty)$ or a user-provided domain	– UML: generalizations, association end multiplicities, class multiplicities – OCL: all invariants – Correctness property under analysis, e.g. no redundant invariants
A variable as for each association	Potential number of links in association as ($[0, \infty)$ or a user-provided domain	– UML: association end multiplicities – OCL: invariants containing navigations through association as
A variable at for each attribute	Potential values of attribute at, e.g. $[0, 1]$ for boolean, $(-\infty, \infty)$ for integers or a user-provided domain	– OCL: invariants accessing the value of attribute at
A variable aux_e for each subexpression e in each OCL constraint	Potential values of the expression e. Non-basic types are abstracted, e.g. collections are abstracted as integers encoding their size	– A constraint establishing the value of e in terms of the values of its subexpressions – Correctness property under analysis, e.g. the root expression of each invariant must evaluate to 1 (all invariants must be true)

2 Bound tightening procedure

The inputs of our procedure will be a UML/OCL model, a correctness property to be checked and a set of bounds for bounded verification. These initial bounds may be unconstrained (i.e. infinite) or finite bounds proposed by the designer. From this input, the output will be a set of refined bounds. These improved bounds can then be relayed to a bounded verification solver, which can take advantage of the reduced search space to perform verification more efficiently.

The computation of tightened bounds is performed in two steps:

- **Abstraction**: We consider all implicit and explicit constraints from the UML/OCL model and abstract those that involve search space boundaries. This abstraction is formalized as a Constraint Satisfaction Problem (CSP), i.e. a finite set of *variables* V, the set of *domains* D of potential values for each variable and the set of *constraints* C over the variables in V.
- **Propagation**: the constraints in the CSP are used to remove unfeasible values from the domains of variables, a process known as *integer bound propagation* [5]. In this paper, we will use the *hybrid integer-real Interval arithmetic Constraint solver* (IC) from the ECL^{i}PSe Constraint Programming System [2]. The IC solver can handle both integral and real variables and it provides powerful interval constraint propagation capabilities.

Given that propagation is a feature provided by most off-the-shelf CSP solvers, we will focus our presentation on the abstraction phase. This step builds upon two previous works from the literature: the definition of a CSP encoding for UML/OCL models [6] and the work on *size abstraction* for OCL properties [17].

In particular, we modify the CSP encoding from [6] such that (a) OCL constraints are not directly encoded in the CSP but rather abstracted as size con-

Table 2. Analysis of OCL invariants from Example 1

OCL Expression (e)	**Size Constraint (e.c)**
$e_1.attr$	$domain(e.v) \subseteq domain(attr)$
$e_1 \rightarrow exists(e_2)$	$(0 \leq e.v \leq 1) \wedge ((e_1.v = 0 \vee e_2.v = 0) \rightarrow (e.v = 0)) \wedge$ $((e_2.v = 1) \rightarrow (e.v = (e_1.v \geq 1))) \wedge e_1.c \wedge e_2.c$
$e_1 \rightarrow isUnique(e_2)$	$(0 \leq e.v \leq 1) \wedge ((e.v = 0) \rightarrow (e_1.v \geq 2)) \wedge$ $(domain\_size(e_2.v) \geq e_1.v) \wedge e_1.c \wedge e_2.c$
$Type :: allInstances()$	$e.v = num\_obj(Type)$
e_1 and e_2	$(e.v = \min(e_1.v, e_2.v)) \wedge e_1.c \wedge e_2.c$

straints and (b) instead of finding a particular solution to the CSP we tighten the initial bounds. Table 1 describes the overall structure of the CSP computed in this phase, defined in terms of its variables, domains and constraints.

The abstraction of OCL constraints is similar to [17] but it has been extended to cover further OCL constructs and consider the domain of attributes. Furthermore, the goal of the abstraction is not checking properties that only need size-related information as in [17], but to accelerate the verification of arbitrary properties. Table 2 details this abstraction, i.e. the last row of Table 1, for the OCL invariants in Example 1. The first column represents the OCL subexpression e being abstracted and the second column shows the size constraint $e.c$ derived from the analysis of e. This size constraint is expressed with the help of an auxiliary variable $e.v$ abstracting the "size" of expression e, e.g. the number of elements in a collection or the length of a string.

Example 2. Let us revisit the model from Example 1. The following constraints on the population of classes and associations can be derived from UML constructs (top 4, using [6]) and OCL invariants (bottom 3, using Table 2):

$$\texttt{AbstractMachine} = \texttt{Cutter} + \texttt{Grinder} \quad \text{Inheritance}$$
$$\texttt{Uses} \leq \texttt{Part} * \texttt{AbstractMachine} \quad \text{Association}$$
$$\texttt{Uses} = 4 * \texttt{AbstractMachine} \quad \text{Association end}$$
$$\texttt{Uses} \leq \texttt{Part} \quad \text{Association end}$$
$$\texttt{Part} \leq domain\_size(\texttt{Serial}) \quad \text{Invariant } \texttt{UniqueSerials}$$
$$\texttt{Cutter} \geq 1 \quad \text{Invariant } \texttt{Availability}$$
$$\texttt{Grinder} \geq 1 \quad \text{Invariant } \texttt{Availability}$$

3 Experimental Results

In this section, we evaluate the speedup achieved by bound tightening in the bounded verification process. To this end, we consider *strong satisfiability*, i.e. checking if there is a valid instance that populates each non-abstract class. Results are measured on the SAT-based USE model validator plug-in [11].

For our experiments, we have used two UML/OCL models: "Teams" (5 classes, 3 associations, 6 attributes and 6 invariants) and "Company" (6 classes, 8 associations, 21 attributes, 16 invariants). For the sake of representativity, we have defined two versions of each model, one which is strongly satisfiable

(sat) and one which is not (unsat). For each one, different sets of initial bounds (number of objects and links and ranges for attributes) have been considered.

Table 3 summarizes the results obtained in an Intel Core i7 3Ghz with 8 Gb RAM. Each entry describes the experiment (model, verification bounds and sat/unsat), and the execution time in seconds for USE with the original bounds (**USE**) and for USE with bound tightening (**Tight**). Finally, we measure the speedup in the execution time (**Spd**, 1 if no change, higher is better).

In all models, the overhead of tightening is less than 1 second. Regarding verification time, the effect of bound tightening is most noticeable in models where verification is most complex. There, significant reductions can be achieved with some examples running 50 times faster.

4 Related Work

Bounded verification is a popular strategy for analyzing UML/OCL models [1,6, 11,16]. Several techniques can be used to accelerate it: *parallelization* (use several solvers running in parallel over different parts of the formula or the domains), *slicing* (partition the problem into independent components that can be analyzed separately) and *bound reduction* (reduce the size of the verification bounds).

In the context of UML/OCL verification, [14, 15] describe slicing techniques to partition class diagrams and ParAlloy [13] studies the parallel verification of Alloy models. Considering UML class diagrams without OCL, [3, 8] study the potential interactions among association multiplicities to detect situations where multiplicities can be strengthened or are unsatisfiable. However, this paper is the first work addressing bound reduction for the verification of UML/OCL models.

In other fields, there are related approaches to bound reduction. In static program analysis, the most related one is TACO [9], a tool for the verification of JML-annotated Java programs. Meanwhile, in the model checking of hybrid systems, *Domain reduction abstraction* [7] partitions the input domains into equivalence classes with the same behavior.

5 Conclusions

The bounded verification of UML/OCL models can be accelerated by assisting designers in the selection of verification bounds, a task which currently lacks

Table 3. Experimental results (timeout set at 10.000 seconds)

	CPU time				CPU time		
Experiment	**USE**	**Tight**	**Spd**	**Experiment**	**USE**	**Tight**	**Spd**
Team-small-sat	1,8s	2,5s	x0,76	Company-small-sat	258,4s	20,5s	x11,54
Team-mid-sat	3,7s	7,3s	x0,50	Company-mid-sat	100,4s	61,3s	x1,64
Team-large-sat	5,8s	7,3s	x0,79	Company-large-sat	1.479,5s	258,5s	x5,94
Team-small-unsat	0,8s	1,4s	x0,50	Company-small-unsat	904,7s	17,9s	x50,42
Team-mid-unsat	2,0s	2,9s	x0,69	Company-mid-unsat	4.452,5s	2087,2s	x2,13
Team-large-unsat	7,6s	5,0s	x1,53	Company-large-unsat	timeout	4426,1s	–

Bounds: [1,N] objects, [1,2*N] links. Small (N=5), Mid (N=10) and Large (N=15).

adequate support. The proposed method abstracts the UML/OCL model as a constraint satisfaction problem. Then, interval constraint propagation is used to tighten the analysis bounds. Smaller bounds can reduce the verification time.

This approach can be used in different ways: as a preprocessing stage before verification or as part of an interactive process to guide the choice of bounds. As future work, we plan to investigate heuristics regarding the best order for selecting bounds, i.e. one that reduces the number of choices and maximizes the amount of information that can be inferred automatically by bound propagation.

References

1. K. Anastasakis, B. Bordbar, G. Georg, and I. Ray. On challenges of model transformation from UML to Alloy. *Software and Systems Modeling*, 9(1):69–86, 2010.
2. K. R. Apt and M. Wallace. *Constraint Logic Programming using ECLiPSe*. Cambridge University Press, 2007.
3. M. Balaban and A. Maraee. Simplification and correctness of UML class diagrams - focusing on multiplicity and aggregation/composition constraints. In *MODELS'2013*, volume 8107 of *LNCS*, pages 454–470. Springer, 2013.
4. D. Berardi, D. Calvanese, and G. D. Giacomo. Reasoning on UML class diagrams. *Artificial Intelligence*, 168(1-2):70–118, 2005.
5. L. Bordeaux, G. Katsirelos, N. Narodytska, and M. Y. Vardi. The complexity of integer bound propagation. *J. Artif. Intell. Res. (JAIR)*, 40:657–676, 2011.
6. J. Cabot, R. Clarisó, and D. Riera. On the verification of UML/OCL class diagrams using Constraint Programming. *Journal of Systems and Software*, 93:1–23, 2014.
7. Y. Choi and M. Heimdahl. Model checking software requirement specifications using domain reduction abstraction. In *ASE'2003*, pages 314–317. IEEE, 2003.
8. I. Feinerer, G. Salzer, and T. Sisel. Reducing multiplicities in class diagrams. In *MODELS 2011,*, volume 6981 of *LNCS*, pages 379–393. Springer, 2011.
9. J. P. Galeotti, N. Rosner, C. G. L. Pombo, and M. F. Frias. Taco: Efficient SAT-based bounded verification using symmetry breaking and tight bounds. *IEEE Transactions on Software Engineering*, 39(9):1283–1307, 2013.
10. C. A. González and J. Cabot. Formal verification of static software models in MDE: A systematic review. *Information and Software Tech.*, 56(8):821–838, 2014.
11. M. Kuhlmann and M. Gogolla. From UML and OCL to relational logic and back. In *MODELS'2012*, volume 7590 of *LNCS*, pages 415–431. Springer, 2012.
12. A. Queralt and E. Teniente. Verification and validation of UML conceptual schemas with OCL constraints. *ACM TOSEM*, 21(2):13:1–13:41, 2012.
13. N. Rosner, J. P. Galeotti, C. L. Pombo, and M. F. Frias. ParAlloy: Towards a framework for efficient parallel analysis of Alloy models. In *ABZ'2010*, volume 5977 of *LNCS*, pages 396–397. Springer, 2010.
14. J. Seiter, R. Wille, M. Soeken, and R. Drechsler. Determining relevant model elements for the verification of UML/OCL specifications. In *DATE'2013*, pages 1189–1192. EDA Consortium, 2013.
15. A. Shaikh, R. Clarisó, U. K. Wiil, and N. Memon. Verification-driven slicing of UML/OCL models. In *ASE'2010*, pages 185–194. ACM, 2010.
16. M. Soeken, R. Wille, M. Kuhlmann, M. Gogolla, and R. Drechsler. Verifying UML/OCL models using boolean satisfiability. In *DATE'2010*, pages 1341–1344. IEEE, 2010.
17. F. Yu, T. Bultan, and E. Peterson. Automated size analysis for OCL. In *FSE'2007*, pages 331–340. ACM, 2007.