Sound Object Model Refactorings

Rohit Gheyi[⋆] and Paulo Borba

Informatics Center
Federal University of Pernambuco
{rg,phmb}@cin.ufpe.br

Abstract. Refactorings are usually proposed in an *ad hoc* way because it is hard to guarantee their soundness with respect to a formal semantics. Usually, even using refactoring tools, developers have to rely on compilation and tests in order to improve their confidence that semantics is preserved, which may not be satisfactory to critical software development. We propose a set of semantics-preserving transformations based on a refinement theory that we propose for Alloy. This set of transformations is shown to be relatively complete in the sense that it can derive a representative set of model transformations, and proven sound with respect to a formal semantics for Alloy in a theorem prover. Moreover, we show how they can be composed to derive refactorings and optimizations. They are a powerful tool for reasoning about object model transformations, and can be used to improve refactoring tools.

1 Introduction

Evolution is a demanding software development activity, as the originally defined structure usually does not accommodate adaptations, demanding new ways to reorganize software. Modern development practices, such as program refactoring [1], improve programs while maintaining their original behavior, in order, for instance, to prepare software for change. An *object model refactoring* is a transformation that improves design structure preserving semantics. They might bring similar benefits but with a greater impact on cost and productivity, since they are used in earlier stages of the software development process.

In current practice, in spite of refactoring tool support, programmers still rely on successive compilation and test suite executions in order to improve confidence that the behavior is preserved [1]. However, a test suite is able only to uncover errors, not to prove their absence. Moreover, modifying the structure of a program may imply updating the unit tests. Therefore, besides being a time-consuming activity, relying on a test suite is not a good way to improve confidence that the behavior is preserved. In case of structural model refactorings, most proposed transformations rely on informal argumentation. It is difficult to prove that refactorings are sound with respect to a formal semantics. Defining all enabling conditions required for a transformation to be semantics-preserving is not an easy task. Even a number of object model transformations proposed

⋆ We would like to thank all anonymous referees.

in the literature, which are intended to be semantics-preserving, may lead to models with type errors or subtle semantic changes in some situations. Moreover, there is no comprehensive set of structural model transformations to help designers to improve their models.

Related work [2,3,4,5,6] has been carried out on semantics-preserving transformations for UML-like class diagrams. These approaches do not state in which conditions a transformation can be applied. Therefore, some transformations may not preserve semantics in some situations. These transformations do not preserve semantics because some of them use a semi-formal semantics. Others partially define semantics but do not verify soundness of transformations, or do not consider constraints. Although they propose transformations relating equivalent UML-like class diagrams models, some of them do not state when two models are equivalent.

2 Goal Statement

In this research, we propose a comprehensive set of semantics-preserving transformations for Alloy [7], which is a formal object-oriented modeling language. By composing them, we derived some model refactorings. Moreover, we show that this set of transformations is relatively complete, in the sense that it is sufficient to reduce an arbitrary Alloy model to an equivalent one in a normal form. We follow a similar approach used for imperative and object-oriented languages [8,9]. So, we can derive a representative set of transformations by composing them.

Our transformations relate equivalent models based on an abstract equivalence notion that we propose for Alloy [10], and encode it in the Prototype Verification System (PVS) [11], which encompasses a formal specification language and a theorem prover. Moreover, we prove some properties of this notion in PVS, such as compositionality. Our transformations are proven sound in PVS with respect to a formal semantics for Alloy encoded in PVS and the equivalence notion proposed [12].

3 Approach and Evaluation

We propose a set of transformations that not only preserves semantics, but also that it does not introduce any type errors or break the well-formedness properties of Alloy models. Consequently, we extend a previously defined semantics for Alloy and specify a type system for Alloy, and encode them in PVS.

Since we are proposing transformations that relate equivalent object models, we need to use an equivalence notion stating when two object models are equivalent. The common equivalence notion states that two object models are equivalent if they have the same semantics. This notion is useful, but not flexible enough to compare equivalent models with auxiliary elements [10]. Therefore, we propose and encode in PVS a more flexible and abstract equivalence notion for object models.

We used these transformations to refactor a real case study. We have applied them to refactor a graphical framework (Swing and AWT) of Java. Besides being useful for clarifying Alloy's semantics and a powerful tool for reasoning about Alloy models, our model transformations can be used to improve the analysis performance of a tool. Moreover, these model transformations can be used to derive model refactorings, such as *Extract Interface* and *Pull Up Field* refactorings. One of the most difficult tasks for proposing refactorings is to define required enabling conditions. Proposing and proving refactorings in PVS help identify when transformations do not introduce type errors or inconsistencies. Even popular program refactoring tools, such as Eclipse [13], may introduce some simple errors, such as making a program ill-typed or behaviorally different. In case of model refactoring, this scenario is even worse since there are a few model transformations proposed in the literature, most of them in an *ad hoc* way. Consequently, following our approach for proposing refactorings can help improve tool support, adding reliability to software refactoring.

So far, we have proposed an equivalence notion and proved some related properties, a formal semantics and type system for Alloy in PVS [10]. Moreover, we have proposed 50 primitive semantics-preserving transformations for Alloy. As a future work, we intend to derive more coarse-grained transformations, such as refactorings. We have shown that our set of transformations is relatively complete. As a future work, we will investigate whether we can find a stronger completeness notion. Finally, we aim at formally relating our refinement notion to the traditional notion of refinement.

References

1. Fowler, M.: Refactoring: Improving the Design of Existing Code. Addison-Wesley (1999)
2. Bergstein, P.: Object-preserving class transformations. (In: OOPSLA'91)
3. Evans, A.: Reasoning with UML class diagrams. In: 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques. (1998) 102–113
4. Marković, S., Baar, T.: Refactoring OCL annotated UML class diagrams. In: 8th MoDELS. Volume 3713 of LNCS. (2005) 280–294
5. Sunyé, G., Pollet, D., Traon, Y., Jézéquel, J.M.: Refactoring UML models. In: 4th Conference on UML. Volume 2185 of LNCS., Springer-Verlag (2001) 134–148
6. Lano, K., Bicarregui, J.: Semantics and transformations for UML models. In: 1st Conference on UML. (1998) 97–106
7. Jackson, D.: Software Abstractions: Logic, Language and Analysis. MIT press (2006)
8. Hoare, T., et al.: Laws of programming. CACM **30** (1987) 672–686
9. Borba, P., Sampaio, A., Cavalcanti, A., Cornélio, M.: Algebraic Reasoning for Object-Oriented Programming. Science of Comp. Programming **52** (2004) 53–100
10. Gheyi, R., Massoni, T., Borba, P.: An Abstract Equivalence Notion for Object Models. Electronic Notes in Theoretical Computer Science **130** (2005) 3–21
11. Owre, S., et al.: PVS Language Reference. At http://pvs.csl.sri.com (2006)
12. Gheyi, R., Massoni, T., Borba, P.: A Rigorous Approach for Proving Model Refactorings. (In: 20th IEEE/ACM Conference on ASE)
13. Eclipse.org: Eclipse project. At http://www.eclipse.org (2006)