Program Slicing for Declarative Models

Engin Uzuncaova and Sarfraz Khurshid
The University of Texas at Austin
Austin, TX 78712
{uzuncaov, khurshid}@ece.utexas.edu

ABSTRACT

The declarative modeling language Alloy and its automatic analyzer provide an effective tool-set for building designs of systems and checking their properties. The Alloy Analyzer performs bounded exhaustive analysis using off-the-shelf SAT solvers. The analyzer's performance hinges on the complexity of the models and so far, its feasibility has been shown only within small bounds. With the growing popularity of analyzable declarative modeling languages, in general, and Alloy, in particular, it is imperative to develop new techniques that allow the underlying solvers to scale to real systems.

We present Kato, a novel technique that defines program slicing for declarative models and enables efficient analyses using existing analyzers, such as the Alloy Analyzer. Given a declarative model, Kato identifies a slice, which represents the model's *core*: a satisfying solution to the slice can be systematically extended to generate a solution for the entire model, while unsatisfiability of the core implies unsatisfiability of the entire model. The experimental results show that it is possible to achieve a significant improvement in the solving time.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification

1. INTRODUCTION

As software systems grow in complexity, the need for efficient automated techniques for design, testing and verification becomes even more critical. The declarative modeling language Alloy [3] and its fully automatic analyzer [4] provide an effective tool-set for building designs of systems and checking their properties.

Alloy is a first-order relational logic with transitive closure, which allows expressing rich structural properties using succinct and intuitive path expressions. The Alloy Analyzer translates Alloy models into boolean formulas using a *scope*—bound on the universe of discourse—provided by the user, and uses off-the-shelf SAT technology to solve the resulting boolean formulas.

For practical examples, Alloy's analysis is often limited to small scopes, which can be impractically small for modeling realistic systems. The Alloy Analyzer already incorporates a variety of optimizations, such as symmetry-breaking, partial functions, type-based reduction of variables, and guidelines for manual rewriting of Alloy formulas [6] to optimize the solving time. In past work [5], we presented a suite of optimizations inspired by traditional compiler optimizations, such as common subexpression elimination and loop unrolling, to perform source-to-source translations on Alloy models to enable the SAT solvers to perform more efficiently.

In this paper, we present a new class of optimizations, which are inspired by program slicing for imperative languages [7] but are applicable to analyzable declarative languages, in general, and Alloy, in particular. We present Kato, a novel algorithm for slicing declarative models. Given an Alloy model, Kato identifies a slice, which represents the model's *core*: a satisfying instance for the core can systematically be extended into a satisfying instance for the entire model, while unsatisfiability of the core implies unsatisfiability of the entire model.

2. Kato: PROGRAM SLICING FOR ALLOY

Kato performs a static analysis of the given model to identify its core. The static analysis traverses the abstract syntax tree of the given Alloy formula, which is a conjunction of several sub-formulas, to build a use-set of relations that appear in each sub-formula. Kato uses these use-sets to compute the set of core relations. We take the model slice that consists of the core relations and the constraints that apply to only these relations, and use the Alloy Analyzer to find a satisfying instance for the model's core. Since the model slice typically consists of only a strict subset of the original model, the slice translates to smaller boolean formulas with fewer variables, which improves the performance of the underlying SAT solvers.

We have evaluated the potential speedup in solving time that Kato can provide using a suite of benchmark examples that model structurally complex data. The results evince the existence of opportunities for significant performance gains. For the binary search tree example, we observe maximum speed-up of 19.13X and for the linked-list example, a maximum speed-up of 6.16X.

2.1 Core and Derived Relations

We partition the set of relations declared in an Alloy model into two sets: *core* and *derived*. We use core relations to define a slice of the given Alloy model, which is analyzed first using SAT. Next, we use the constraints on the derived relations to extend satisfying instances of core relations into satisfying instances of the complete model.

Definition 1. Let R be the set of all relations. Let C and D partition R. Let f_C be the formulas in f that only involve relations in C. Let I be the set of all instances of f. Let I_C be the set of all valuations to relations in C and I_D be the set of all valuations to relations in D. C is a *core set* if and only if:

$$\forall i_C \in I_C \mid f_C(i_C) \Rightarrow \exists i_D \in I_D \mid f(i_C + + i_D)$$

Definition 2. A relation r is *core* if and only if there exists a core set and r belongs to that set. Similarly, a relation r is *derived* if and only if a core set exists and r is not in the core set.

Given the constructive nature of our definition of core relations (Definition 2), a simple algorithm suffices to construct the set of core relations for a given Alloy model. In order to prevent too conservative core sets, the relations that appear as the bounds on the quantifiers are also considered as core relations.

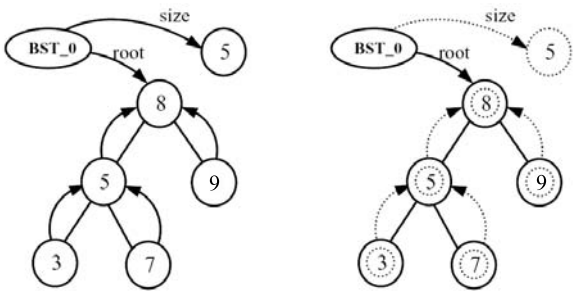
3. CASE-STUDY: BINARY SEARCH TREE

This section presents a case-study of using Kato to slice a model of binary search trees [1]. The model that we slice with Kato defines the constraints on acyclicity, connectivity, `parent` relation, and `size` relation, and also specifies the ordering property of a general binary search tree regarding the values in the nodes.

For the boolean formula corresponding to the complete model, the SAT solver, takes 25.44 seconds (on average) and produces a valid instance for the binary search tree with 14 nodes. Figure 1(a) illustrates this instance with five nodes.

To intuitively see which relations are core, notice that `parent` relation can be computed, once the left and right relations are determined. Similarly `size` and in fact even `key` can also be computed given the values for the other relations. Figure 1(b) shows a partial instance corresponding to the core relations using solid lines; the dotted lines depict the derived relations. The core relations computed by Kato are `left`, `right` and `root`. The generated core slice removes the relations `key`, `size` and `parent` and also the constraints defined on these relations from the complete model.

Running the analyzer for the slice computed by Kato takes only 1.33 seconds (on average) to generate a satisfying instance for the slice, a 19.13X improvement in SAT solving time. Besides the improvements in SAT solving, slicing also enables significant reduction in compilation time in the Alloy Analyzer.



(a) (b)

Figure 1: The binary search tree instance generated by the analyzer. The tree in (a) presents the complete solution. In (b), derived properties are indicated by dotted lines.

4. CONCLUSIONS

We have presented Kato, a novel technique that defines program slicing for declarative models and enables efficient analyses using existing analyzers, such as the Alloy Analyzer. Given a declarative model, Kato identifies a slice, which represents the model's *core*: a satisfying solution to the slice can be systematically extended to generate a solution for the entire model, while unsatisfiability of the core implies unsatisfiability of the entire model. The experimental results show that it is possible to achieve a significant improvement in the solving time for Alloy models. We believe analyses based on program slicing hold a lot of promise for efficiently checking declarative specifications.

It is worth pointing that generating boolean formulas that optimize analysis of underlying SAT solvers is particularly challenging because the performance of SAT solvers cannot be described in any simple terms: it is generally based on heuristics and does not always hold [2]. In the context of Alloy, the problem is even more interesting because of the optimizations that the Alloy Analyzer does internally. We plan to systematically explore these issues.

The problem of augmenting a partial solution to represent a complete solution is non-trivial to solve in general; in the most general case, it is as complex as SAT solving. However, even if the computation of derived relations is expensive, our approach still offers potential benefits. For example, analysis of just the core slice may reveal that the whole model is infeasible. In another scenario, it may turn out that enumerating and augmenting partial instances is actually faster than directly solving the complete model.

5. REFERENCES

[1] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1990.

[2] M. K. Ganai, L. Zhang, P. Ashar, A. Gupta, and S. Malik. Combining strengths of circuit-based and CNF-based algorithms for a high-performance SAT solver. In *Proc. 39thConference on Design Automation (DAC)*, pages 747–750, June 2002.

[3] D. Jackson. *Software Abstractions: Logic, Language and Analysis*. The MIT Press, Cambridge, MA, 2006.

[4] D. Jackson, I. Schechter, and I. Shlyakhter. ALCOA: The Alloy constraint analyzer. In *Proc. 22nd International Conference on Software Engineering (ICSE)*, Limerick, Ireland, June 2000.

[5] D. Marinov, S. Khurshid, S. Bugrara, L. Zhang, and M. Rinard. Optimizations for compiling declarative models into boolean formulas. In *8th International Conference on Theory and Applications of Satisfiability Testing (SAT 2005)*, St.Andrews, Scotland, 2005.

[6] I. Shlyakhter. *Declarative Symbolic Pure Logic Model Checking*. PhD thesis, MIT, February 2005.

[7] M. Weiser. Program slicing. In *Proc. 5th International Conference on Software Engineering (ICSE)*, pages 439–449, San Diego, California, Mar. 1981. IEEE Computer Society Press.