Science of Computer Programming 94 (2014) 203–216

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Preventing arithmetic overflows in Alloy

Aleksandar Milicevic *, Daniel Jackson

Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Laboratory, 32 Vassar St, Cambridge, MA 02139, USA

HIGHLIGHTS

- A new treatment of partial functions in classical logic.
- Goal: suppress variable bindings causing out-of-domain applications.
- Idea: assign truth values to undefined formulas to make them globally irrelevant.
- Applied to Alloy (a bounded model finder) to prevent arithmetic overflows.
- Result: the Alloy Analyzer made sound with respect to counterexamples.

ARTICLE INFO

Article history:
Received 15 February 2013
Received in revised form 7 May 2014
Accepted 9 May 2014
Available online 23 May 2014

Keywords:
Arithmetic overflows
Partial functions
Logic
First order
Alloy

ABSTRACT

In a bounded analysis, arithmetic operators become partial, and a different semantics becomes necessary. One approach, mimicking programming languages, is for overflow to result in wrap-around. Although easy to implement, wrap-around produces unexpected counterexamples that do not correspond to cases that would arise in the unbounded setting. This paper describes a new approach, implemented in the latest version of the Alloy Analyzer, in which instances that would involve overflow are suppressed, and consequently, spurious counterexamples are eliminated. The key idea is to interpret quantifiers so that bound variables range only over values that do not cause overflow.

1. Introduction

A popular approach to the analysis of undecidable logics artificially bounds the universe, making a finite search possible. In model checking, the bounds may be imposed by setting parameters at analysis time, or even hardcoded into the system description. The Alloy Analyzer [1] is a model finder for the Alloy language that follows this approach, with the user providing a 'scope' for an analysis that sets the number of elements for each basic type.

Such an analysis is not sound with respect to proof; just because a counterexample is not found (in a given scope) does not mean that no counterexample exists (in a larger scope). But it is generally sound with respect to counterexamples. That is, no spurious counterexamples are generated, so if a counterexample is found, the putative theorem does not hold.

The soundness of Alloy's counterexamples is a consequence of the fact that the interpretation of a formula in a particular scope is always a valid interpretation for the unbounded model. There is no special semantics for interpreting formulas in the bounded case. This is possible because the relational operators are closed, in the sense that if two relations draw

* Corresponding author.
E-mail addresses: aleks@csail.mit.edu (A. Milicevic), dnj@csail.mit.edu (D. Jackson).
URLs: http://people.csail.mit.edu/aleks (A. Milicevic), http://people.csail.mit.edu/dnj (D. Jackson).

http://dx.doi.org/10.1016/j.scico.2014.05.009
0167-6423/

their elements from a given universe of atoms, then any relation formed from them (for example, by union, intersection, composition, and so on) can be expressed with the same universe.

Arithmetic operators, in contrast, are not closed. For example, the sum of two integers drawn from a given range may fall outside that range. So the arithmetic operators, when interpreted in a bounded context, appear to be partial and not total functions, and call for special treatment. One might therefore consider applying the standard strategies that have been developed for handling logics of partial functions.

A common strategy is to make the operators total functions by selecting appropriate values when the function is applied out of domain. In some logics (e.g., [2]) the value is left undetermined, but this approach is not easily implemented in a search-based model finder. Alternatively, the value can be determined. In the previous version of the Alloy Analyzer, arithmetic operators were totalized in this way by giving them wraparound semantics, so that the smallest negative integer is regarded as the successor of the largest positive integer. This matches the semantics in some programming languages (e.g., Java), and is relatively easy to implement. Unfortunately, however, it results in counterexamples that would not arise in the unbounded context, so the soundness of counterexamples is violated. This approach leads to considerable confusion among users, and imposes the burden of having to filter out the spurious cases.

Another common strategy is to introduce a notion of undefinedness—at the value, term or formula level—and extend the semantics of the operators accordingly. However this is done, its consequence will be that formulas expressing standard properties will not hold. The associativity of addition, for example, will be violated, because the definedness of the entire expression may depend on the order of summation. In logics that take this approach, the user is expected to insert explicit guards that ensure that desired properties do not rely on undefined values. In our setting, however, where the partiality arises not from any feature of the system being described, but from an artifact of the analysis, demanding that such guards be written would be unreasonable, and would violate Alloy's principle of separating description from analysis bounds.

This paper provides a different solution to the dilemma. Roughly speaking, counterexamples that would result in arithmetic overflow are excluded from the analysis, so that any counterexample that is presented to the user is guaranteed not to be spurious. This is achieved by redefining the semantics of quantifiers in the bounded setting so that the models of a formula are always models of the formula in the unbounded setting. This solution has been implemented in Alloy 4.2 and it is by default turned on; it can be deactivated via the "Prevent Overflows" option.

The rest of the paper is organized as follows. Section 2 introduces the Alloy Analyzer. Section 3 illustrates some of the anomalies that arise from treating overflow as wraparound. Section 4 shows the problem in a more realistic context, by presenting an Alloy model of a minimum spanning tree algorithm that combines arithmetic and relational operators, and shows how a valid theorem can produce spurious counterexamples. Section 5 explains and formalizes our new semantics, which is the key contribution of this paper. Section 6 explains our implementation in boolean circuits and discusses how it ensures the desired semantics. Section 7 presents evaluation, showing (1) a case study where the analysis time is cut by 33% due to the reduced search space imposed by the new semantics, and (2) an exhaustive set of tests we applied to ensure our implementation meets the specification. Section 8 presents related work on the topic of partial functions in logic, compares our approach with the existing ones, and discusses alternatives for solving the issue of overflows in Alloy. Section 9 concludes.

2. Alloy background

Alloy [3] is a first-order relational modeling language. Alloy models lend themselves to fully automated bounded analysis—embodied in a tool called the Alloy Analyzer [1]. The expressiveness of the relational language is one of the characteristic features of Alloy which makes it particularly suitable for checking deep properties of structurally complex systems. To keep the logic decidable and the analysis fully automated, the Alloy Analyzer requires, however, that all domains be bounded.

The model finding part of the analysis is offloaded to Kodkod [4], a constraint solver for relational first-order logic. Kodkod works by translating a given relational formula (together with provided bounds for each relation) into an equivalent propositional boolean formula and using an of-the-shelf SAT solver to check its satisfiability.

In addition to pure relations, Kodkod also provides support for integers and arithmetic operations. Integers are an important part of Alloy, because they enable various program analysis tools that build on top of it; examples include tools for testing [5], verification [6], and specification execution [7].

Integers in Alloy must also be bounded. The user is required to explicitly specify the number of bits (*bitwidth*) to be used for their representation. In Alloy, signed integers are represented in a twos-complement system, restricting the analysis to using the integers from $\{-2^{bitwidth-1}, \ldots, 2^{bitwidth-1} - 1\}$. As explained above, this poses certain dilemmas about the semantics of arithmetic operations. The previous versions of Alloy (up to and including v4.1.2) implement wraparound semantics; in this paper we explain the new semantics implemented in Alloy 4.2, which excludes the models containing arithmetic overflows from the search space.

3. Prototypical overflow anomalies

While a wraparound semantics for integer overflow is consistent and easily explained, its lack of correspondence to unbounded arithmetic produces a variety of anomalies. Most obviously, the expected properties of arithmetic do not necessarily