Automated Verification for Domain-Specific Languages

Combining Domain Knowledge and Automated Theorem Provers/Solvers

Sylvia Grewe
TU Darmstadt

Sebastian Erdweg
TU Delft

Mira Mezini
TU Darmstadt

ABSTRACT

In the last years, computer science has seen much progress in the area of automated verification of domain-specific properties such as loop invariants and simple assertion verification in programs. Yet, there remain domains where the abstract strategies for verifying standard properties is well understood by domain experts, but still not automated to a satisfactory degree. An example for such a domain are type soundness proofs. Being able to express domain-specific verification strategies using domain-specific terminology and concepts can help to narrow down this gap towards more automated verification. We present an infrastructure that allows for expressing such domain knowledge and for interfacing with existing automated theorem provers and solvers to verify individual steps within proofs.

1 MOTIVATION

Automated theorem provers such as Vampire and SMT solvers such as Z3 are able to solve a large number of individual difficult proof problems from different domains. Interactive theorem provers such as Isabelle and Coq enable the mechanization of large and complex proofs, as well as the partial automation of such proofs by writing custom tactics. Still, a domain expert who wants to verify a standard property in her domain often finds herself without automated tactics that she can apply or modify to suit her needs. One reason for this is the lack of means for expressing domain-specific verification strategies via domain-specific concepts in existing systems. We briefly illustrate the problem and our idea for a solution with a small, widely-known problem domain.

Example. Sudoku is a logic-based number puzzle. The objective is to fill a 9x9 grid with digits so that each column, each row, and each of the nine 3x3 sub-grids (i.e. *boxes*) contains all of the digits from 1 to 9. Modern computer programs and solvers such as Z3 typically find a solution in at most a couple of seconds via exhaustive search strategies or constraint solving. However, human Sudoku experts are usually interested in finding a solution by applying a set of domain-specific, well-understood solving strategies. A simple example for such a strategy is *Naked Pair*: "A *Naked Pair* is a set of two *candidate* numbers sited in two *cells* that have at least one *unit* (i.e. *row*, *column* or *box*) in common. The solution will contain those values in those two cells, and all other candidates with those numbers can be removed from whatever *unit(s)* they have in common".[1]

Assume a Sudoku expert wanted to implement a stepwise solver which tries to apply strategies such as *Naked Pair*. To do that using current tools, she would have to express domain-specific constraints such as *have at one least unit in common* via low-level commands like series of array accesses and nested loops that iterate through the grid. Programming this can be a time-consuming and error-prone task. Her task would be much simpler if she had a framework that enabled her to flexibly create domain-concepts, use them to express domain-specific solving strategies, combine them to obtain stepwise solution steps, and connect to existing solvers for verifying that each step is indeed correct.

[1] http://www.sudokuwiki.org/Naked_Candidates#NP

Conference'17, Washington, DC, USA
YYYY. 978-x-xxxx-xxxx-x/YY/MM…$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

2 APPROACH

We are designing Veritas [1], a Scala API for combining domain-specific verification strategies with existing theorem provers and solvers. Veritas allows for constructing *proof graphs*, where intermediate *proof obligations* (or intermediate states for solving a particular problem) are nodes, and deterministic *tactics* describe the relation between a parent obligation and the child obligations. Veritas is generic in formats for problem specifications and obligations, thus enabling domain-specific formats. Tactic steps and leaf obligations may be verified by connecting different existing provers or solvers.

For instance, for solving Sudokus using domain-specific strategies, we proceed as follows using Veritas: We first implement Sudoku fields with empty fields and candidate sets. We use instances of Sudoku fields for instantiating the proof obligations within proof graphs. Next, we implement accessing the domain-specific units of Sudoku fields (rows, columns, boxes) and useful domain-specific queries such as "Which units do two given cells share?". Building on these concepts, we implement domain-specific solving tactics such as *Naked pair*. We assemble these tactics into strategies that automatically construct a proof graph for solving a given Sudoku. We connect external provers for verifying that the individual steps indeed comply with the Sudoku rules.

3 APPLICATION: TYPE SOUNDNESS

We are studying the area of *type soundness proofs* for small languages as an application scenario for Veritas: Proving *progress* and *preservation* properties ensures that well-typed programs do indeed not get stuck with type errors. We study how to express abstract strategies for such proofs using domain knowledge and terminology, and how to combine the domain-specific strategies with existing automated theorem provers.

REFERENCES

[1] Grewe, Erdweg, Wittmann, and Mezini. 2015. Type Systems for the Masses: Deriving Soundness Proofs and Efficient Checkers. In *Proceedings of International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software (ONWARD)*. ACM, 137–150.