Science of Computer Programming 155 (2018) 146–172

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Exploration of language specifications by compilation to first-order logic

Sylvia Grewe [a,*], Sebastian Erdweg [b,**], André Pacak [a], Michael Raulf [a], Mira Mezini [a]

[a] Technische Universität Darmstadt, Karolinenpl. 5, 64289 Darmstadt, Germany
[b] Delft University of Technology, Mekelweg 4, 2628 CD Delft, Netherlands

ARTICLE INFO

Article history:
Received 7 January 2017
Received in revised form 4 June 2017
Accepted 10 August 2017

Keywords:
Type systems
Formal specification
Declarative languages
First-order theorem proving
Domain-specific languages

ABSTRACT

Exploration of language specifications helps to discover errors and inconsistencies early during the development of a programming language. We propose exploration of language specifications via application of existing automated first-order theorem provers (ATPs). To this end, we translate language specifications and exploration tasks to first-order logic, which many ATPs accept as input. However, there are several different strategies for compiling a language specification to first-order logic, and even small variations in the translation may have a large impact on the time it takes ATPs to find proofs.

In this paper, we first present a systematic empirical study on how to best compile language specifications to first-order logic such that existing ATPs can solve typical exploration tasks efficiently. We have developed a compiler product line that implements 36 different compilation strategies and used it to feed language specifications to 4 existing first-order theorem provers. As benchmarks, we developed language specifications for typed SQL and for a Questionnaire Language (QL), with 50 exploration goals each. Our study empirically confirms that the choice of a compilation strategy greatly influences prover performance in general and shows which strategies are advantageous for prover performance. Second, we extend our empirical study with 4 domain-specific strategies for axiom selection and find that axiom selection does not influence prover performance in our benchmark specifications.

© 2017 Elsevier B.V. All rights reserved.

1. Introduction

The correct specification and implementation of programming languages is a difficult task. In a previous study, Klein et al. have found that language specifications often contain errors, even when drafted and reviewed by experts [1]. To uncover such errors, Klein et al. propose lightweight mechanization and exploration of language specifications via execution and automated test generation instead of using powerful interactive theorem provers such as Isabelle [2] and Coq [3] for mechanizing language specifications. In this paper, we investigate an approach that is orthogonal to the one from Klein et al. [1]: We propose the application of automated first-order theorem provers (ATPs) for exploration of language specifications. To this end, we study the compilation of language specifications from a lightweight specification language to first-order logic.

* Principal corresponding author.
** Corresponding author.
E-mail addresses: grewe@cs.tu-darmstadt.de (S. Grewe), S.T.Erdweg@tudelft.nl (S. Erdweg), mezini@cs.tu-darmstadt.de (M. Mezini).

http://dx.doi.org/10.1016/j.scico.2017.08.001
0167-6423/© 2017 Elsevier B.V. All rights reserved.

We investigate five typical exploration tasks, which we formulate as proof goals in first-order logic:

Execution of t:	$\exists\, v.\, ground(v) \wedge f(t) = v$?
Synthesis for v:	$\exists t.\, ground(t) \wedge f(t) = v$?
Testing of t and v:	$f(t) = v$?
Verification of P:	$\forall t.\, P(t)$?
Counterexample for P:	$\exists t.\, ground(t) \wedge \neg P(t)$?

Above, f can for example represent the semantics of a language and $ground(t)$ is true if term t is a value. The property P can for example be the associativity of a language construct (e.g. union of sets), or that a certain language construct always returns a term of a certain form.

The technical challenge we address is how to best compile language specifications to first-order logic such that existing ATPs perform well on the proof problems which result from exploration tasks. Here, we define prover performance from the pragmatic perspective of an ATP user: Given a certain amount of hardware resources (e.g. CPU time), we define the performance of an ATP as its overall success rate on a given set of input problems, that is, the rate of problems for which the ATP finds a definite answer such as "refutation" or "satisfiable". Our goal is to find compilation strategies which maximize the overall success rate of ATPs for the proof problems we are interested in. Our early experiments with compilation of language specifications to first logic showed that even a small change to the compilation strategy can have a large impact on prover performance. The general explanation for this behavior is that ATPs employ heuristics-driven proof strategies which often behave differently on semantically equivalent, but syntactically different input problems. Unfortunately, even for ATP experts, it is next to impossible to foresee which compilation strategy will be advantageous for prover performance.

Therefore, we tackle this challenge by conducting an empirical study where we systematically compare a number of different compilation strategies against each other with regard to how they affect the performance of theorem provers. In our study, we include three compilation strategies for encoding the syntactic sorts of a language specification to first-order logic (typed logic, type guards, type erasure), four compilation strategies for handling specification metavariables (unchanged, inlining, naming, partial naming), and three compilation strategies that apply different simplifications on the resulting problems (none, general-purpose, domain-specific). To this end, we have developed a compiler product line from language specifications to first-order logic. We evaluated the performance of four theorem provers (eprover [4], princess [5], Vampire 3.0, Vampire 4.0 [6]) for each compilation strategy on the five exploration tasks above. As benchmarks for programming language specifications, we used a typed variant of SQL and a Questionnaire Language (QL). To focus on comparing the compilation strategies against each other, we first deliberately omit any self-implemented domain-specific strategies for axiom selection on the compiled problems and rely on the ATPs to perform axiom selection internally. That is, for each case study, we pass all axioms that we generate for the corresponding language specification to the ATP.

Next, we extend our study with four domain-specific strategies for axiom selection and investigate their effect on prover performance. In total, we collected the data of 52800 proof attempts.

While we focus on language specifications, the strategies we identify and our experimental results are relevant for any project that generates first-order proof goals. In summary, this paper makes the following contributions:

- We propose to apply existing ATPs for exploring language specifications by compiling the specifications to first-order logic.
- We present 36 different compilation strategies along 3 dimensions. We have developed a compiler product line that implements all strategies.
- We present two language specifications with 50 exploration proof goals each as benchmark specifications: a typed variant SQL and a Questionnaire Language (QL).
- We systematically evaluate the performance of each compilation strategy on our benchmark specifications for 4 theorem provers. Our results confirm that the choice of a compilation strategy greatly influences prover performance and indicate the most advantageous of our 36 compilation strategies: typed logic and type erasure, inlining of variables, and, in certain cases, domain-specific simplification.
- We systematically evaluate the effect of four domain-specific strategies on our benchmark specifications, finding that axiom selection does not influence prover performance in our case.

We structure this article as follows: We start by explaining the elements of the core language "SPL", which we defined for developing language specifications (Section 2). Next, we first show a basic compilation scheme from SPL to first-order logic (Section 3) and then explain variants from the basic compilation scheme (Section 4). We proceed by presenting our two benchmark specifications (Section 5) and the setup of our empirical study for evaluating the performance of compilation strategies (Section 6). We present and discuss the results of the evaluation of our compilation strategies next (Section 7). Afterwards, we extend our empirical study with strategies for axiom selection and present and discuss the effects (Section 8). The present article extends our previous conference paper with the same title [7]. We explain the differences to the conference paper in the related work section (Section 9), where we also extensively discuss further work related to the present article.

Download English Version:

https://daneshyari.com/en/article/6875274

Download Persian Version:

https://daneshyari.com/article/6875274

Daneshyari.com