Interpolation and Symbol Elimination in Vampire

Kryštof Hoder¹, Laura Kovács², and Andrei Voronkov¹

¹ University of Manchester
² TU Vienna

Abstract. It has recently been shown that proofs in which some symbols are colored (e.g. local or split proofs and symbol-eliminating proofs) can be used for a number of applications, such as invariant generation and computing interpolants. This tool paper describes how such proofs and interpolant generation are implemented in the first-order theorem prover Vampire.

1 Introduction

Interpolation offers a systematic way to generate auxiliary assertions needed for software verification techniques based on theorem proving [7,10], predicate abstraction [5,7], constraint solving [15], and model-checking [12,1].

In [9] it was shown that symbol-eliminating inferences extracted from proofs can be used for automatic invariant generation. Further, [10] gives a new proof of a result from [7] on extracting interpolants from colored proofs. This proof contains an algorithm for building (from colored proofs) interpolants that are boolean combinations of symbol-eliminating steps. Thus, [10] brings interpolation and symbol elimination together.

Based on the results of [9,10] we implemented colored proof generation in the first-order theorem prover Vampire [14]. Colored proofs form the base for our interpolation and symbol elimination algorithms.

The purpose of this paper is to describe how interpolation and symbol elimination are implemented and can be used in Vampire. We do not overview Vampire itself but only describe its new functionalities. The presented features have been explicitly designed for making Vampire appropriate for formal software verification: symbol elimination for automated assertion (invariant) synthesis and computation of Craig interpolants for abstraction refinement. Unlike its predecessors, the “new” Vampire thus provides functionalities which extend the applicability of state-of-the-art first theorem provers in verification. To the best of our knowledge, it is the first theorem prover that supports both invariant generation and interpolant computation.

The obtained symbol eliminating inferences and interpolants contain quantifiers, and can be further used as invariant assertions to verify properties of programs manipulating arrays and linked lists [13,9]. We believe that software verification may benefit from the interpolant generation engine of Vampire.

Implementation. The new version of Vampire is available from http://www.vprover.org and runs under most recent versions of Linux (both 32 and 64 bits),

* This work has been partly done while the second authors was at ETH Zürich.
¹ Such proofs are also called local and split proofs, in this paper we will call them colored.
MacOS and Windows. Vampire is implemented in C++ and has about 73,000 lines of code.

**Experiments.** We successfully applied Vampire on benchmarks taken from recent work on interpolants and invariants \[6,19,3,4,15,8\] – see Section 4 and the mentioned URL. Our methods can discover required invariants and interpolants in all examples, suggesting its potential for automated software verification.

**Related work.** There are several interpolant generation algorithms for various theories. For example, \[12,5,7,1\] derive interpolants from resolution proofs in the combined ground theory of linear arithmetic and uninterpreted functions. The approach described in \[15\] generates interpolants in the combined theory of arithmetic and uninterpreted functions using constraint solving techniques over an a priori defined interpolants template. The method presented in \[13\] computes quantified interpolants from first-order resolution proofs over scalars, arrays and uninterpreted functions.

Our algorithm implemented in Vampire automatically extracts interpolants from colored first-order proofs in the superposition calculus. Theories, such as arithmetic or theories of arrays, can be handled by adding theory axioms to the first-order problem to be proved. Thus, interpolation in Vampire is not limited to decidable theories for which interpolation algorithms are known. One can use arbitrary first-order axioms. However, a consequence of this generality is that we do not guarantee finding interpolants even for decidable theories. Moreover, if a theory is not finitely axiomatisable, we can only use its incomplete first-order axiomatisation.

As far as we know, symbol elimination has not been implemented in any other system. A somehow related approach to symbol elimination is presented in \[13,16\] where theorem proving is used for inferring loop invariants. Contrary to our approach, the cited works are adapted to prove given assertions as opposed to generating arbitrary invariants. Using the saturation-based theorem prover SPASS \[18\], \[13\] generates interpolants as quantified invariants that are strong enough to prove given assertions. In \[16\] templates over predicate abstraction are used, reducing the problem of invariant discovery to that of finding solutions, by the Z3 SMT solver \[2\], for unknowns in an invariant template formula. Unlike \[13,16\], we automatically generate invariants as symbol eliminating inferences in full-first order logic, without using predefined predicate templates or assertions.

**2 Colored Proofs, Symbol Elimination and Interpolation**

**Colored proofs** are used in a context when some (predicate and/or function) symbols are declared to have colors. In colored proofs every inference can use symbols of at most one color, as a consequence, every term or atomic formula used in such proofs can use symbols of at most one color, too. We will call a symbol, term, clause etc. colored if it uses a color, otherwise it is called transparent.

In **symbol elimination** \[9\] we are interested in inferences having at least one colored premise and a transparent conclusion; such inferences are called symbol-eliminating. Conclusions of symbol-eliminating inferences can be used to find loop invariants. Symbol elimination can be reformulated as consequence-finding: we are trying to find transparent consequences of a theory including both colored and transparent formulas.