Interpolants and Symbolic Model Checking

K.L. McMillan

Cadence Berkeley Labs

Abstract. An interpolant for a mutually inconsistent pair of formulas (A, B) is a formula that is (1) implied by A, (2) inconsistent with B, and (3) expressed over the common variables of A and B. An interpolant can be efficiently derived from a refutation of $A \wedge B$, for certain theories and proof systems. In this tutorial we will cover methods of generating interpolants, and applications of interpolants, including invariant generation and abstraction refinement.

1 Introduction

An interpolant for a mutually inconsistent pair of formulas (A, B) is a formula that is (1) implied by A, (2) inconsistent with B, and (3) expressed over the common variables of A and B. Craig's interpolation lemma [1] states that every pair of inconsistent first-order formulas has an interpolant. For certain theories and proof systems, we can derive an interpolant for (A, B) from a refutation of $A \wedge B$. For example, interpolants can be derived from resolution proofs in propositional logic. We can also derive interpolants from refutation proofs in first-order logic, and in the quantifier-free fragment of first-order logic with various interpreted theories [5].

Interpolants derived from proofs have a variety of applications in model checking. In various contexts, interpolation can be used as a substitute for image computation, which involves quantifier elimination and is thus computationally expensive. The idea is to replace the image operation with a weaker approximation that is still strong enough to prove a given property.

For example, interpolation can be used to construct an inductive invariant of a sequential system, such as a hardware design or a program. This invariant contains only information actually deduced by a prover in refuting counterexamples a given property. Thus, in a certain sense, this method abstracts the invariant relative to a given property, exploiting the prover's ability to focus the proof on a small set of relevant facts. This avoids the complexity of computing the strongest inductive invariant (i.e., the reachable states) as is typically done in model checking, and works well in the the case where a relatively simple, localized invariant suffices to prove a property of a large system.

This approach gives us a complete procedure for model checking temporal properties of finite-state systems that allows us to exploit recent advances in SAT solvers for the proof generation phase. Experimentally, the method is found to be quite robust for industrial hardware verification problems, relative to other

B. Cook and A. Podelski (Eds.): VMCAI 2007, LNCS 4349, pp. 89–90, 2007.

model checking approaches [4]. The same approach can be applied to infinite-state systems, such as programs and parameterized protocols, using first-order provers, or proof-generating decision procedures. Using interpolants to avoid the expense of predicate image computations, we can obtain substantial efficiencies in software model checking [6]. Moreover, using appropriate techniques we can guarantee to find an inductive invariant proving a given property, if one exists in the prover's theory (though in general the verification problem is undecidable).

Interpolants can also be used for abstraction refinement in various contexts. For example, the Blast software model checker uses interpolants to derive predicates for use in predicate abstraction [2], and also to refine the predicate transition relation, to compensate for the inaccuracy of the Cartesian image approximation [3].

The tutorial will cover methods for generating interpolants from proofs, for both the propositional and first-order cases, and the various applications of these methods.

References

1. W. Craig. Three uses of the herbrand-gentzen theorem in relating model theory and proof theory. *J. Symbolic Logic*, 22(3):269285, 1957.
2. T. A. Henzinger, R. Jhala, Rupak Majumdar, and K. L. McMillan. Abstractions from proofs. In *Principles of Prog. Lang. (POPL 2004)*, pages 232244, 2004.
3. R. Jhala and K. L. McMillan. Interpolant-based transition relation approximation. In Kousha Etessami and Sriram K. Rajamani, editors, CAV, volume 3576 of *Lecture Notes in Computer Science*, pages 3951. Springer, 2005.
4. K. L. McMillan. Interpolation and sat-based model checking. In *Computer-Aided Verification (CAV 2003)*, pages 113, 2003.
5. Kenneth L. McMillan. An interpolating theorem prover. *Theor. Comput. Sci.*, 345(1):101121, 2005.
6. Kenneth L. McMillan. Lazy abstraction with interpolants. In Thomas Ball and Robert B. Jones, editors, *CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 123136. Springer, 2006.