

Interpolation Strategies

Kenneth L. McMillan

Microsoft Research

We can think of interpolation as construction of a modular proof. We want a proof to have a particular structure because we hope that such a proof will generalize beyond a particular case, perhaps providing us with an inductive argument. From a practical point of view, however, we need more than modularity. We need strategies for producing proofs that actually do generalize, with reasonable computational effort.

In fact, it is difficult to separate the questions of proof search and interpolation, since effective proof search approaches are often predicated on effective interpolation. For instance, the “learned” clauses in a CDCL SAT solver and the theory lemmas in a lazy SMT solver are interpolants. The quality of these interpolants is a major determinant of the performance of such provers. What they lack is any explicit mechanism to exploit the modular structure of the problem. We can translate the resulting non-modular proofs into a modular form (that is, if we have a feasible interpolation result) but we may obtain a proof that is far more complex than necessary.

An alternative strategy is to fit the proof search (that is, decision making and interpolation) to the structure of the problem. A number of techniques do this, including Lazy Annotation and PDR (and techniques that are not conflict-driven such as local saturation methods). Within this general approach, there are many choices to be made in computing “good” interpolants. For example, what space of proofs should we consider, and how much effort should be put into optimization within this space? More specific decisions can make interpolation easier, but may result in less relevant or general interpolants. Intuitively, by considering a very specific case, we may not see enough of the problem to generalize well.

Approaching these issues empirically, we can make a number of observations. Most importantly, a structured decision making and interpolation approach seems to be more effective in practice than an unstructured one. However, a purely local proof search approach, as in PDR, can lead to over-specific decision making. At some point it is better to switch to a non-local approach based on proof translation. This point can be determined dynamically. That is, we may resort to further case splitting when the obtained interpolant is too complex. The interpolant complexity question becomes acute when the interpolants contain quantifiers. In particular, an array theory interpolation technique that introduces quantifiers can be practical, but sufficient attention must be paid to simplifying the quantifier structure of the interpolants. Finally, greedy strengthening of interpolants (for example by disjunct dropping) does seem to be justified, though it is not a large contributor to performance.

The Duality fixed point enigne in Z3 provides platform for investigating these questions. We will discuss how Duality deals with some of the important trade-offs, and how these decisions effect performance in software model checking applications.