Interpolants from Z3 proofs

Kenneth L. McMillan Microsoft Research

Abstract—**Interpolating provers have a number of applications in formal verification, including abstraction refinement and invariant generation. It has proved difficult, however, to construct efficient interpolating provers for rich theories. We consider the problem of deriving interpolants from proofs generated by the highly efficient SMT solver Z3 in the quantified theory of arrays, uninterpreted function symbols and linear integer arithmetic (AUFLIA) a theory that is commonly used in program verification. We do not directly interpolate the proofs from Z3. Rather, we divide them into small lemmas that can be handled by a secondary interpolating prover for a restricted theory. We show experimentally that the overhead of this secondary prover is negligible. Moreover, the efficiency of Z3 makes it possible to handle problems that are beyond the reach of existing interpolating provers, as we demonstrate using benchmarks derived from bounded verification of sequential and concurrent programs.**

I. Introduction

Interpolating provers have a number of applications in formal verification, including abstraction refinement [9] and invariant generation [17]. Given a valid implication $P \rightarrow Q$, an interpolating prover can produce an *interpolant* for the implication, that is, a formula I, expressed using the common vocabulary of P and Q, such that $P \rightarrow I$, $I \rightarrow Q$. There are various methods to accomplish this, but a common one is to extract the interpolant from a proof refuting $P \wedge \neg Q$, using an *interpolation calculus* [18]. Typically, one computes interpolants modulo a theory, for example, the theory of linear arithmetic over the integers. In this case, the interpolant is allowed to contain any interpreted symbols of the theory. The required proof may be obtained from a satisfiability modulo theories (SMT) solver, instrumented to produce proofs.

A significant practical difficulty with this approach is to obtain an efficient SMT solver that produces proofs in the required proof system. The theory solvers in a modern SMT solver are complex, in part due to the requirement of fast incremental operation to support backtracking, and in part due to the complexity of the theories themselves (for example, efficient solving of integer linear arithmetic constraints has long been a topic of research). Because of the difficulty of producing efficient proof-generating theory solvers, existing interpolating provers are typically less efficient than state-of-the-art SMT solvers, or do not support all of the desired theories. In practice, this inefficiency has been compensated somewhat by reducing the complexity of the input formulas, for example by considering only a single program execution path, as in [9], [19]. If interpolating solvers matching the performance of the best SMT solvers were available, however, it might be possible to use interpolation in a broader context, for example, considering more complex control flow, or perhaps concurrency.

In this paper, we consider the problem of deriving interpolants from proofs generated by the state-of-the-art SMT solver Z3 [8] in a rich theory, namely, the quantified theory of arrays and linear integer arithmetic (AUFLIA, according to the SMT-LIB nomenclature [2]).

Z3's proof calculus is complex, and rich enough to polynomially simulate proofs systems such as extended resolution that do not admit feasible interpolation[1]. Moreover, it allows "theory lemmas" that can introduce any validity of the theory without proof. Thus, for example, to refute a pair of complex formulas A and B, the proof system would allow a theory lemma that simply says $A \wedge B \rightarrow \text{FALSE}$. As a result, there is no reason in principle why a Z3 proof should contain sufficient information to construct an interpolant.

For this reason, we will take an approach that considers a Z3 proof as guide for construction of a proof by a secondary, less efficient, interpolating prover. We will translate Z3 proofs into a proof calculus that does admit feasible interpolation, with "gaps", or lemmas, that must be discharged by the secondary prover. This approach succeeds if the secondary prover can in practice discharge these lemmas in time that is small in relation to the time Z3 used to construct the original proof. A key test in this regard is the number of backtracks that the secondary solver must perform. If this is low, then the secondary solver need not have highly efficient incremental theory solvers. There is then no need to modify Z3 for the purpose of interpolant generation.

An additional benefit of this approach is that the secondary solver need not implement the entire theory. Our secondary solver implements only the *quantifier-free* theory of linear arithmetic and uninterpreted function symbols (QF_UFLIA). Quantifier instantiation is performed by Z3, as is instantiation of the axioms of the array theory. Thus, we can in principle use any of the available interpolating provers for QF_UFLIA as our secondary solver [3], [4].

To test these ideas, we use a collection of interpolation problems in AUFLIA, derived from bounded verification of sequential and concurrent programs using the Poirot tool [15]. Because these formulas have complex Boolean structure, they exploit the ability of Z3 to backtrack efficiently. We observe experimentally that interpolants can be efficiently derived from the Z3 proofs, while existing interpolating provers are unable to handle these formulas.

Related work Previous to this work there were no interpolating provers available for AUFLIA. A number of interpolating SMT solvers have been produced for subsets of this theory,

[1]Extended resolution proofs generalize resolution proofs by allowing resolution on arbitrary formulas, rather than just propositional atoms. This system is known, under cryptographic assumptions, not to admit feasible interpolation [13].

including Princess [3] (UFLIA), MathSAT4 [4] (QF_UFLRA) and SMTInterpol[2] (QF_LIA). Their performance is not comparable to Z3, as we will observe in in section V, using Z3 to instantiate the quantifiers and array axioms. Any solvers supporting QF_UFLIA can be used as the secondary solver in the present approach.

Interpolation has also been implemented in the first-order prover Vampire [10], however it is complete only in the ground case and applies only to rational (not integer) arithmetic. Moreover, it lacks an SMT solver's efficiency in combining Boolean and theory reasoning.

Interpolation in the theory of arrays has been handled in different ways. The method of [11] is based on discovery of local instantiations of the array axioms (a local predicate is expressed entirely in the vocabulary of A or the vocabulary of B). It is necessarily incomplete, but is guaranteed to produce quantifier-free interpolants for quantifier-free formulas. The present method is complete but may introduce quantifiers in the interpolants caused by non-local axiom instantiations. The method of [12] is similar to the present one in this respect. The primary difference is that it eagerly instantiates the array axioms, whereas here we rely on instantiations generated by Z3. Also, we should note that the present method is not specific to the array theory. It can handle any theory which Z3 handles by axiom instantiation (though it cannot in general handle axiom schemas).

In [5] an entirely different approach to arrays is taken, extending the signature of the array theory to allow quantifier-free interpolation. If such an approach were used in the secondary solver, we could safely discard the array axiom instances produced by Z3. In this way, the present method can either accommodate the weaknesses or exploit the strengths of the secondary interpolating prover.

A significant hurdle in interpolating proofs generated by SMT solvers is that interpolating proof calculi require the pivots of resolution steps to be local, but SMT solvers may for various reasons resolve on non-local predicates. In [6] a method is introduced to raise non-local pivots to the leaves of a resolution proof by re-ordering resolution steps. This method is worst-case exponential. Here we take a less general but linear-time approach that relies on knowledge of the structure of Z3 proofs. It is sufficient to raise resolutions on non-local pivots introduced by equational rewriting in Z3, which accounts for most cases of non-local resolution pivots. The remaining cases are handled by a different technique called "lemma extraction".

Overview of the paper In the next section, we cover some background including definitions and notations used in the paper. Section III introduces a simple proof calculus allowing feasible interpolation, while section IV describes our approach of translating Z3 proofs into this calculus. Section V then describes our experimental evaluation.

II. Background

We use standard first-order logic over a countable vocabulary Σ of function and predicate symbols, with associated arities. Function symbols with arity zero will be called constants. We will use t, u, v to represent first-order terms and ϕ, ψ, p, q and capital Roman letters to represent first-order formulas. We distinguish a finite subset Σ_I of Σ as *interpreted* symbols. In particular, we assume that Σ_I contains the binary predicate symbol $=$, representing equality. We assume a countable set $\mathcal{V}$ of variables, distinct from Σ. We will use x, y, z to represent variables. The vocabulary of a term or formula ϕ, denoted $L(\phi)$ is the set of *uninterpreted* function and constant symbols occurring in ϕ. If S is a vocabulary, we say $\mathcal{L}(S)$ is the set of first-order terms and formulas ϕ such that $L(\phi) \subseteq S$. We will also write $\mathcal{L}(\phi)$ for $\mathcal{L}(L(\phi))$ and $s \ll \phi$ to indicate that a symbol s occurs in ϕ.

A *theory* is a set of first-order formulas over Σ. We say a formula ϕ is valid relative to a theory $\mathcal{T}$ if every model of $\mathcal{T}$ is a model of ϕ, and we write this $\models_{\mathcal{T}} \phi$. We use capitol Greek letters Γ and Δ to stand for multisets of formulas. We will write a formula multiset as list of formulas and formula multisets. Thus, if Γ is a multiset of formulas and ϕ a formula, then Γ, ϕ represents $\Gamma \cup \{\phi\}$. We write $\wedge\Gamma$ for the conjunction of the formulas in Γ, $\vee\Gamma$ for the disjunction and $\neg\Gamma$ for the multiset of negations of formulas in Γ.

A *sequent* is written $\Gamma \vdash \Delta$, where Γ and Δ are multisets of formulas. Here, Γ is said to be the *antecedent* and Δ the *consequent*. We also call the elements of Γ *assumptions*. This sequent is *valid* if the conjunction of the formulas in Γ implies the disjunction of the formulas in Δ, given a background theory $\mathcal{T}$. That is, $\Gamma \vdash \Delta$ is valid if $\models_{\mathcal{T}} \bigwedge\Gamma \rightarrow \bigvee\Delta$. An empty antecedent or consequent will be represented by a blank. Thus $\vdash \phi$ means ϕ is valid, and $\phi \vdash$ means ϕ is a contradiction (implies the empty disjunction or FALSE). We will sometimes use calligraphic letters such as $\mathcal{J}$ to stand for sequents.

A formula or term is said to be *ground* if it contains no variables. A *position* π is a finite sequence of natural numbers, representing a syntactic position in a term or formula. If ϕ is a formula or term, then $\phi|_\pi$ represents the subformula or subterm of ϕ at position π. Thus, $\phi|_\epsilon$ is ϕ itself, $\phi|_i$ is the i-th argument of ϕ, $\phi|_{ij}$ is the j-th argument of the i-th argument, and so on. The notation $\phi[\psi]_\pi$ means ϕ with ψ substituted in position π.

An *interpolant* for a valid implication $A \rightarrow B$ is a formula I such that $A \rightarrow I$ and $I \rightarrow B$ are valid, and such that I is written using the vocabulary common to A and B, that is, $I \in \mathcal{L}(A) \cap \mathcal{L}(B)$. The Craig interpolation lemma [7] states that an interpolant always exists for a valid implication in first order logic (FOL). Validity in this definition may also be relative to a theory $\mathcal{T}$, though interpolants may not always exist in this case. When dealing with refutation systems, it is more convenient to speak of an interpolant for an unsatisfiable conjunction $A \wedge B$. An interpolant for the conjunction $A \wedge B$ is a formula $I \in \mathcal{L}(A) \cap \mathcal{L}(B)$ such that $A \rightarrow I$ and $B \rightarrow \neg I$ are both valid. In the sequel, let A and B be fixed formulas.

An *inference* is of the form

$$\frac{\mathcal{P}_1 \ \cdots \ \mathcal{P}_k}{\mathcal{C}}$$

where $\mathcal{P}_1 \cdots \mathcal{P}_k$ is a multiset of sequents called *premises*

[2]http://swt.informatik.uni-freiburg.de/research/tools/smtinterpol

(which we will often abbreviate $\{\mathcal{P}_i\}$) and $\mathcal{C}$ is a sequent called the *conclusion*. An inference is *sound* if validity of the premises implies validity of the conclusion. Generally, inferences are instances of *inference rules*, or patterns. Such a rule is sound when every instance matching the pattern, and satisfying any side conditions, is sound.

A *derivation tree* is a directed tree whose nodes are labeled with inferences. The premises of each node must contain the multiset of conclusions of its children. A derivation tree may be *open*, however, in the sense that some premises of inferences are not conclusions of any child. We call these unproved sequents the *premises* of the derivation tree. A tree with no premises is said to be *closed*. The conclusion of a derivation tree is the conclusion of its root node.

III. Interpolating proof calculi

We begin by introducing a very simple proof calculus and a corresponding *interpolation calculus* [18] that allows us to derive interpolants from proofs. Our eventual goal is to translate proofs from Z3 into this calculus.

We will say a formula is *local* when it is expressed either in the the vocabulary of A or in the vocabulary of B. A sequent is local when *all* its formulas are expressed in the the vocabulary of A, or *all* are expressed in the vocabulary of B. That is, $\Gamma \vdash \Delta$ is local when $\Gamma, \Delta \subseteq \mathcal{L}(A)$ or $\Gamma, \Delta \subseteq \mathcal{L}(B)$. We will say that a sequent is *strict* if each individual formula in the sequent is local, that is, if $\Gamma, \Delta \subseteq \mathcal{L}(A) \cup \mathcal{L}(B)$. Similarly, we will say that an inference or derivation tree is local (respectively strict) if all of its premises and conclusions are local (respectively strict). In writing proof rules, we will use the notation $\Gamma \vdash_l \Delta$ to indicate a local sequent and $\Gamma \vdash_s \Delta$ to indicate a strict sequent.

We should note that strictness is not an issue in purely propositional clausal proofs. Every clause in such a proof is necessarily strict because every propositional atom in it occurs in either A or B. In the current more general setting, non-strictness may occur either because of mixed terms within an atomic formula, or because the formulas in the sequent are not atomic.

The rules of our proof calculus $\mathcal{S}_P$ are as follows:

$$\textsc{Local}\ \frac{}{\Gamma \vdash_l \Delta}\quad \models_{\mathcal{T}} \wedge\Gamma \rightarrow \vee\Delta$$

$$\textsc{Res}\ \frac{\Gamma \vdash_s \Delta, p \quad \Gamma \vdash_s \Delta', \neg p}{\Gamma \cup \Gamma' \vdash_s \Delta \cup \Delta'}$$

$$\textsc{Contra}(\Gamma)\ \frac{\Gamma, \Gamma' \vdash_s}{\Gamma' \vdash_s \neg\Gamma}$$

The first rule, Local, allows us to introduce any valid *local* sequent. As we will see, computing interpolations for local sequents is trivial. The second rule, Res, allows us to resolve two strict sequents on some pivot formula p. Note that resolving two local sequents might result in a strict but not local sequent, since the pivot p might be in both $\mathcal{L}(A)$ and $\mathcal{L}(B)$. Note also that the pivot p need not be an atomic formula. It is only required to be local. The third rule, Contra, allows us to move formulas Γ from the left- to the right-hand side of a strict sequent. That is, if assuming Γ entails a contradiction, then one of the formulas in Γ must be false. Notice that the rules of our system allow us to produce only strict sequents. The soundness of these rules is easily verified. Completeness is also easily shown for theories that have the Craig interpolation property, though this is not relevant to the current discussion.

Now, given a derivation of a sequent $A, B \vdash$, we would like to derive an interpolant for $A \wedge B$. We can do this using an interpolation calculus in the style of [18]. We sketch one such system here, though a detailed understanding of this system is not needed for what follows.

For any set Γ of formulas, we will write Γ_B for $\Gamma \cap \mathcal{L}(B)$ and Γ_A for $\Gamma \setminus \mathcal{L}(B)$ (note the asymmetry in these definitions). A sequent in the interpolation calculus (also called an *interpolation*) is of the form $(A, B) \vdash \Delta\ [\phi]$. The antecedent is a pair of formulas, A and B, the consequent is a multiset of formulas Δ and the formula ϕ acts as an interpolant for the sequent. The sequent is said to be *valid* when

1) A and $\neg\Delta_A$ imply ϕ,
2) B and $\neg\Delta_B$ imply $\neg\phi$, and
3) $\phi \in \mathcal{L}(A) \cap \mathcal{L}(B)$.

Another way to say this is that the interpolation is valid when ϕ is an interpolant for $A \wedge (\wedge\neg\Delta_A)$ and $B \wedge (\wedge\neg\Delta_B)$. Moreover, when Δ is the empty set, ϕ is an interpolant for $A \wedge B$. The set of interpolation rules $\mathcal{S}_I$, shown in Figure 1 is sound in the sense that they produce valid interpolations from valid interpolations. These rules can be interpreted roughly as follows. To interpolate a purely local sequent on the A side, we take the disjunction of the formulas of Δ that are in the common vocabulary of A and B. To interpolate a purely local sequent on the B side, we simply take True as the interpolant. If we resolve on an A-side formula, we take the disjunction of the interpolants, while resolving on the B side gives the conjunction. The rule for proof by contradiction has no effect on the interpolation.

Now suppose we have a derivation in system $\mathcal{S}_P$ of a sequent $A, B \vdash_s$. This is, we have proved that formulas A and B are inconsistent. We can transform this into a derivation of an interpolation $(A, B) \vdash [\phi]$ in the system $\mathcal{S}_I$. To do this, we replace each inference in the proof by a corresponding inference in $\mathcal{S}_I$, so that each sequent $\Gamma \vdash \Delta$ in the proof is replaced by an interpolation of the form $(A, B) \vdash \neg(\Gamma \setminus \{A, B\}),\ \Delta\ [\phi]$. That is, in the derived interpolation, the assumptions other than A and B are moved to the consequent side. As an example transformation step, if $\psi \in \mathcal{L}(A) \cap \mathcal{L}(B)$, and $\phi \in \mathcal{L}(A) \setminus \mathcal{L}(B)$, we have

$$\textsc{Local}\ \frac{}{A, \phi \vdash_l \psi} \quad \rightarrow \quad \textsc{LocalA}\ \frac{}{(A, B) \vdash \neg\phi \vee \psi\ [\psi]}$$

Because we move assumptions to the right in the translation, the Contra rule becomes particularly trivial. For example, we have:

$$\textsc{Contra}\ \frac{A, \psi \vdash_l}{A \vdash_l \neg\psi} \quad \rightarrow \quad \textsc{Contra}\ \frac{(A, B) \vdash \neg\psi\ [\phi]}{(A, B) \vdash \neg\psi\ [\phi]}$$

Note that our ability to replace each inference of $\mathcal{S}_P$ by a corresponding inference of $\mathcal{S}_I$ depends critically on the strictness and locality conditions in $\mathcal{S}_P$. For example, we can

$$\textsc{LocalA}\ \frac{}{(A,B) \vdash \Delta\ [\vee \Delta_B]}\ \wedge A \models_{\mathcal{T}} \vee \Delta,\ \Delta \subseteq \mathcal{L}(A)$$

$$\textsc{LocalB}\ \frac{}{(A,B) \vdash \Delta\ [\textsc{True}]}\ \wedge B \models_{\mathcal{T}} \vee \Delta,\ \Delta \subseteq \mathcal{L}(B)$$

$$\textsc{ResA}\ \frac{(A,B) \vdash \Delta, p\ [\phi] \quad (A,B) \vdash \Delta', \neg p\ [\phi']}{(A,B) \vdash \Delta \cup \Delta'\ [\phi \vee \phi']}\ p \in \mathcal{L}(A) \setminus \mathcal{L}(B)$$

$$\textsc{ResB}\ \frac{(A,B) \vdash \Delta, p\ [\phi] \quad (A,B) \vdash \Delta', \neg p\ [\phi']}{(A,B) \vdash \Delta \cup \Delta'\ [\phi \wedge \phi']}\ p \in \mathcal{L}(B)$$

$$\textsc{Contra}\ \frac{(A,B) \vdash \Delta\ [\phi]}{(A,B) \vdash \Delta\ [\phi]}$$

Fig. 1. Interpolation system $\mathcal{S}_I$.

always replace an instance of LOCAL by an instance of either LOCALA or LOCALB because the locality condition demands that $\Gamma \vdash \Delta$ is written in either $\mathcal{L}(A)$ or $\mathcal{L}(B)$.

IV. TRANSLATING Z3 PROOFS

Our goal in this section will be to convert proofs from Z3 into proofs in our simple proof calculus $\mathcal{S}_P$, and from there into our interpolation calculus $\mathcal{S}_I$ to obtain an interpolant.

Given a set of assumptions Γ that are inconsistent relative to a theory $\mathcal{T}$ that Z3 supports, it can produce a proof of a sequent $\Gamma \vdash$. However Z3's proof system is much richer than the simple one we have sketched. At present the system contains 38 documented rules. Many of these relate to particular theories that Z3 supports such as linear arithmetic and the theory of arrays. There is a also a rule, for example, for universal quantifier instantiation. The system also contains rules equivalent to our RES and CONTRA rules.

A very powerful rule in the Z3 system is the THLEMMA rule. This rule takes an arbitrary set of sequents as premises and can produce as a conclusion any sequent implied under one of Z3's theories. The theory solver may provide some hints as to how the proof should be performed, but in general complete proofs of theory lemmas are not provided.

To cope with this, our approach will be to construct a proof in $\mathcal{S}_P$ that is as detailed as possible, leaving unproved "lemmas" at the leaves of the derivation tree. To fit within our system, these lemmas must be strict. It will be the job of a secondary prover to provide interpolations for these lemmas. In the worst case, the proof might reduce to a single big lemma of the form $A, B \vdash$. In practice, though, we will observe that the lemmas tend to be small, and are easily handled by an interpolating prover much less efficient than Z3. Moreover, the lemmas never require quantifier instantiation or the theory of arrays, allowing us to use a secondary prover supporting only equality and integer arithmetic.

We approach the proof translation in several stages. The first stage, called *axiom elimination*, removes any non-local instances of axioms. In the next stage, *localization*, we find any possible applications of the LOCAL rule. Any closed sub-tree of the proof whose conclusion is local can be simply replaced by a single instance of the LOCAL rule. This typically removes a large fraction of the proof. In the last stage, *lemma extraction* we eliminate any inferences that are not available in $\mathcal{S}_P$. This is done by replacing sub-trees of the proof with lemmas to be interpolated by a secondary prover.

We now consider each of the proof translation stages in detail, beginning with the simplest, localization, and proceeding to lemma extraction and axiom elimination. We then cover a few additional optimizations. We will describe these transformations in terms of *replacement rules*, that is, substitutions of a sound derivation sub-tree by another sound derivation tree with the same premises and conclusion.[3]

A. Localization

Any local closed sub-tree of a derivation can be replaced by an instance of the LOCAL rule. We represent this by the following replacement rule:

$$*\ \frac{}{\Gamma \vdash_l \Delta} \quad \rightarrow \quad \textsc{Local}\ \frac{}{\Gamma \vdash_l \Delta}$$

We use the label $*$ here to indicate application of any number of sound inference rules. This rule says that any closed sub-tree using rules of the Z3 proof calculus whose conclusion is $\Gamma \vdash_l \Delta$ can be replaced with an instance of LOCAL with the same conclusion. A *maximal local sub-tree* is a local closed sub-tree that is not a sub-tree of any other local sub-tree. In the localization stage, we apply this replacement rule to all maximal local sub-trees.

B. Lemma extraction

Consider a sound (possibly open) sub-tree whose premises $\vdash_l p_1$ through $\vdash_l p_k$ are local and whose conclusion $\vdash_s \Delta$ is strict. This sub-tree demonstrates that the premises imply the conclusion, that is, the sequent $\vdash \neg p_1, \ldots, \neg p_k, \Delta$ is valid. We can thus introduce this as a lemma, using the following rule, which we add to $\mathcal{S}_P$ to allow introduction of any valid strict sequent:

$$\textsc{Lemma}\ \frac{}{\Gamma \vdash_s \Delta}\quad \models_{\mathcal{T}} \wedge\Gamma \rightarrow \vee\Delta$$

[3]We should also note that in the proof representation provided by Z3, the antecedents of sequents are not explicit. These can be reconstructed, however, by a preliminary pass over the proof structure.

Note that the LEMMA rule generalizes the LOCAL rule in that it allows a conclusion that is strict but not necessarily local.

Having introduced $\vdash \neg p_1, \ldots, \neg p_k, \Delta$ as a lemma, we can then resolve it with all the premises $\vdash p_i$ in turn to obtain the conclusion $\vdash \Delta$. This gives us a way to replace sub-trees with lemmas. This is important, as Z3 often sprinkles short segments of equality reasoning between resolution steps in its proofs. To express this transformation as a replacement rule, we will use the notation RES* to indicate multiple applications of the resolution rule. We then have the following replacement rule:

$$* \, \frac{\{\Gamma_i \vdash_s p_i\}}{\cup_i \Gamma_i \vdash_s \Delta} \quad \rightarrow$$

$$\textsc{Res}^* \frac{\textsc{Lemma}\dfrac{}{\cup_i \Gamma_i \vdash_s \{\neg p_i\} \cup \Delta} \qquad \{\Gamma_i \vdash_s p_i\}}{\cup_i \Gamma_i \vdash_s \Delta}$$

We can make several improvements to this basic transformation. First, note that it requires all assumptions in the premises to be present in the conclusion. If this is not the case, we can rewrite a premise $\Gamma_i, \Gamma'_i \vdash_s p_i$ to $\Gamma_i \vdash_s (\wedge \Gamma'_i) \rightarrow p_i$, where Γ'_i are not assumptions in the conclusion, provided $(\wedge \Gamma'_i) \rightarrow p_i$ is local.

Moreover, assumptions in the conclusion can be dropped if they are not actually used in the sub-tree. The resulting lemma will still be valid. In fact, there is only one rule in the Z3 calculus that uses assumptions. This is the ASSUMP rule, introducing sequents of the form $\phi \vdash \phi$. If an assumption does not appear in an occurrence of ASSUMP within the sub-tree, it can be dropped from the lemma. Finally, we can use the LOCAL rule in the replacement instead of LEMMA if the lemma happens to be local, saving a lemma.

We will call the above transformation *lemma extraction*. Lemma extraction applies to any subtree that is strict, where the consequents of all premises are singletons[4]. We will call such a sub-tree *extractable*. Every node is contained in a unique minimum extractable subtree. This sub-tree can be found by moving up the tree to the first ascendant with a strict conclusion, then extending downward to the first descendant along each branch whose conclusion is strict and has a singleton consequent. Note that a minimum tree must exist containing any given node, because the conclusion of the root node of the tree is $A, B \vdash$ which is strict.

We wish to use lemma extraction to remove from the proof any inferences that do not occur in $\mathcal{S}_P$. The question is which sub-trees to transform into lemmas. Since we want the lemmas to be as small as possible, we will always extract minimal extractable subtrees.

We will say that an inference is *foreign* if it does not occur in $\mathcal{S}_P$. This can be because it uses a rule not present in $\mathcal{S}_P$, or because it does not meet the strictness condition. A derivation tree node is foreign if the inference labeling it is foreign. A foreign node is *maximal foreign* in a given derivation if it is not a strict descendant of any foreign node. In applying lemma extraction, we eliminate the minimal extractable sub-tree of some maximal foreign node. Note that this sub-tree contains the foreign node, but not always at the root. This process is repeated until no foreign nodes remain. Thus, lemma extraction proceeds from the root to the leaves of the proof tree, extracting the smallest possible lemmas. Of course it is conceivable that the root node is foreign, and the the minimal extractable sub-tree is the entire tree. In this case the entire proof reduces to one large lemma, and we have gained nothing. However, in practice we find that the extracted lemmas are quite small.

Finally, having introduced the LEMMA rule into our proof calculus, we require a corresponding interpolation rule:

$$\textsc{Lemma} \; \frac{}{(A, B) \vdash \Delta \; [\phi]} \;\; \dagger$$

The side condition † is that ϕ is an interpolant for $(\wedge \neg \Delta_A) \wedge A)$ and $(\wedge \neg \Delta_B) \wedge B$. This is just a statement of the condition for validity of the conclusion. We have no syntactic way of computing an interpolant for a lemma. Rather, we use the secondary interpolating prover to compute an interpolant ϕ for the formulas $((\wedge \neg \Delta_A) \wedge A)$ and $((\wedge \neg \Delta_B) \wedge B)$. Thus, each lemma we introduce by lemma extraction entails one call to the secondary prover.

C. *Axiom elimination*

Z3 uses a variety of axioms in its proofs. Instances of these axioms are introduced as conclusions of the form $\vdash \phi$ with no premises. If the secondary prover is unaware of these axioms (for example, it does not support the theory of arrays) then it is essential to capture the axiom instances in the Z3 proof using the LOCAL rule. Otherwise, the secondary prover may fail to prove a lemma.

Unfortunately, axiom instances are not always local. A prominent example of this is the axiom for universal quantifier instantiation:

$$\textsc{QuantI} \; \frac{}{\vdash (\forall x.\ \phi) \rightarrow \phi[t/x]} \;\; t \text{ is ground}$$

This says that a formula universally quantified over variable x implies the same formula under substitution of any ground term t for free instances of x. The difficulty with this rule is that t is an arbitrary ground term. Thus, the conclusion of the rule may not be local, even if ϕ is local.

We can, however, force an axiom instance to be local if it is truly needed, at the possible expense of adding quantifiers to the interpolant. To do this we add a fresh set of *localization symbols* $\mathcal{X}$ to Σ_I. That is, we take these symbols to be interpreted so they do not count as part of the vocabulary of a term and may always occur in interpolants. We assume a total, well-founded order $\prec$ on $\mathcal{X}$. We will write $s \doteq t$ to stand for an equation $s = t$ such that $s \in \mathcal{X}$ and t is a ground term such that for all symbols $s' \in \mathcal{X}$ occurring in t, $s' \prec s$. Such an equation will be called a *definition*. Note that the well-founded order prevents circular definitions.

[4]In fact it can be generalized to the case where the consequents of the premises are local multisets, though this has not been implemented and does not appear to be necessary in practice

We introduce the following rule to allow us to drop a definition no longer in use:

$$\textsc{Elim}\ \frac{s \doteq t, \Gamma \vdash_s \Delta}{\Gamma \vdash_s \Delta}\ \ s \not\preccurlyeq \Gamma, \Delta$$

Now consider an axiom $\phi[\cdot]$, with a placeholder to be filled by an arbitrary term of a given sort. Suppose that ϕ itself is local, but we are given an instance $\phi[t]$ that is not local. Let π be a highest local position in t. That is, π is a syntactic position in formula t such that $t|_\pi$ is local, but no higher position in t is local. We can eliminate this non-locality by choosing a fresh localization symbol s, defining $s \doteq t|_\pi$, and substituting s into position π in t. Note that for this to be legitimate, the symbol s must be greater in the order $\prec$ than any localization symbol occurring in $t|_\pi$.

We can apply this idea to localize axiom instances using replacements of the following form:

$$* \frac{\overline{\vdash \phi[t]} \quad \{\mathcal{J}_i\}}{\Gamma \vdash_s \Delta} \quad \to \quad \textsc{Elim}\ \frac{* \dfrac{\overline{\vdash \phi[t[s]_\pi]} \quad \{\mathcal{J}_i\}}{s \doteq t|_\pi, \Gamma \vdash_s \Delta}}{\Gamma \vdash_s \Delta}$$

That is, suppose we can prove some strict sequent $\Gamma \vdash_s \Delta$ from the axiom instance $\phi[t]$ and some other premises $\{\mathcal{J}_i\}$. If we *assume* the definition $s \doteq t|_\pi$, we can prove the same result from the alternative axiom instance $\phi[t[s]_\pi]$. This can be done by carrying the assumption up to the level of the axiom instance and applying substitution to yield the original formula $\phi[t]$. A definition elimination step is then used to remove the assumption. Note this inference is strict since the definition $s \doteq t|_\pi$ is constructed to be local. In this way we obtain the original conclusion $\Gamma \vdash_s \Delta$ from the altered axiom instance $\phi[t[s]_\pi]$.

By repeated applying this rule, we eventually reach the top position of t. At this point, we obtain $\phi[s]$, which is a local instance of the axiom. Thus it can be replaced with an instance of the $\textsc{Local}$ rule. Note this procedure easily generalizes to axioms with multiple placeholders. We apply the above transformation to all the minimal closed strict sub-trees of the proof. The result is that all axiom instances are eliminated from the proof, hence the secondary prover need not be aware of these axioms.

Now, since we have introduced the $\textsc{Elim}$ rule into our proof system, we must also introduce corresponding interpolation rules. These rules are as follows:

$$\textsc{ElimA}\ \frac{(A,B) \vdash s \doteq t, \Delta\ [\phi]}{(A,B) \vdash \Delta\ [\exists s.\ \phi]}\ s \not\preccurlyeq A, B, \Delta\ \ t \in \mathcal{L}(A) \backslash \mathcal{L}(B)$$

$$\textsc{ElimB}\ \frac{(A,B) \vdash s \doteq t, \Delta\ [\phi]}{(A,B) \vdash \Delta\ [\forall s.\ \phi]}\ s \not\preccurlyeq A, B, \Delta\ \ t \in \mathcal{L}(B)$$

Notice that eliminating a definition on the A side adds an existential quantifier to the interpolant, while eliminating a definition on the B side adds a universal. Also note that the side condition that s not occur in A or B is critical to the soundness of the rule. That is, if A implies ϕ and s does not occur in A, then A implies $\forall s.\ \phi$. Similarly, if A and $s = t$ imply ϕ and s does not occur in A, then A implies $\exists s.\ \phi$, with t providing the witness for the existential.

Finally, notice that in case multiple definitions are introduced, their order of elimination is the reverse of the order of introduction. Thus, definitions corresponding to larger terms produce the inner quantifiers.

In practice, we apply this transformation to three axioms: the quantifier instantiation axiom shown above, and the two standard axioms of the non-extensional array theory. In general, this method can be applied to any theory that is finitely axiomatizable in FOL, provided the prover provides the required axiom instances. However, it does not apply to axiom schemas (such as the congruence axiom schema for the theory of uninterpreted functions) because we cannot quantify over functions and predicates in FOL. Though the $\textsc{Elim}$ rule introduces quantifiers in the interpolants, in practice these can often be eliminated using simple rules, for example, by replacing $\exists s.\ s = x \wedge \phi$ with $\phi[x/s]$.

D. Accounting for rewriting

One of the most common reasons that non-strict inferences occur in Z3 proofs is rewriting. That is, if resolution is only performed on predicates that occur in the original assumptions A and B, then only strict inferences can occur in a resolution tree. However, Z3 typically generates some non-local predicates by rewriting. That is, for some predicate p occurring in A or B, Z3 infers $p \Leftrightarrow p'$, where p' is not local, by rewriting p with some unconditional equations in A and B. The non-local predicate p' is then substituted for p, resulting in resolutions on non-local predicates. Since non-strict inferences result in larger lemmas, we would like to substitute the original p back in for p' in the proof to increase strictness.

There may be many possible ways to achieve this. We briefly sketch here one simple approach that has proved effective. We first scan the proof for any sequents of the form $\Gamma \vdash p \Leftrightarrow p'$, where $\Gamma \subseteq \{A, B\}$, p is local, and p' is not local. We can use this equivalence to push resolutions on p' towards the leaves of the derivation tree. From the equivalence $p \Leftrightarrow p'$, we can derived the two implications $p \to p'$ and $p' \to p$. Let $\textsc{Rpl}(p', p)$ be a shorthand for a derivation tree of the following form:

$$\textsc{Res}\ \frac{\Gamma \vdash \Delta, p' \qquad * \overline{\Gamma \vdash p' \to p}}{\Gamma \vdash \Delta, p}$$

That is, $\textsc{Rpl}(p, p')$ uses the implication $p \to p'$ to replace p with p'. Now we can replace any occurrence of resolution on p' using the following rule:

$$\textsc{Res}\ \frac{\Gamma \vdash \Delta, p' \quad \Gamma \vdash \Delta', \neg p'}{\Gamma \cup \Gamma' \vdash \Delta \cup \Delta'} \quad \to$$

$$\textsc{Res}\ \frac{\textsc{Rpl}(p',p) \dfrac{\Gamma \vdash \Delta, p'}{\Gamma \vdash \Delta, p} \qquad \textsc{Rpl}(\neg p', \neg p) \dfrac{\Gamma' \vdash \Delta', \neg p'}{\Gamma' \vdash \Delta', \neg p}}{\Gamma \cup \Gamma' \vdash \Delta \cup \Delta'}$$

That is, we eliminate a resolution on p' by replacing p' with the equivalent p and resolving on p. The resulting instances of $\textsc{Rpl}$ can be pushed up the resolution tree by simply re-ordering resolutions. Here we show only one case (omitting the RES labels to save space):

$$\dfrac{\dfrac{\Gamma \vdash \Delta, p', q \quad \Gamma' \vdash \Delta', \neg q}{\Gamma, \Gamma' \vdash \Delta, \Delta', p'} \qquad * \dfrac{}{\Gamma, \Gamma' \vdash p' \to p}}{\Gamma, \Gamma' \vdash \Delta, \Delta', p} \quad \to$$

$$\dfrac{\dfrac{\Gamma \vdash \Delta, p', q \qquad * \dfrac{}{\Gamma, \Gamma' \vdash p' \to p}}{\Gamma, \Gamma' \vdash \Delta, \Delta', p, q} \qquad \Gamma' \vdash \Delta', \neg q}{\Gamma, \Gamma' \vdash \Delta, \Delta', p}$$

In this way, the resolutions on non-local atoms are pushed upward in the derivation tree until they meet a non-resolution inference. Since these resolutions are foreign they will eventually be eliminated by lemma extraction. By moving them upward in the derivation tree, we make the resulting minimal extractable sub-trees smaller and thus reduce the size of lemmas that must be proved by the secondary prover.

E. Accounting for sub-tree sharing

The proofs generated by Z3 are represented not as trees, but as DAG's. That is, in the proof representation it is possible (and in fact common) for two nodes to share children. Of course we must take care not to process shared sub-trees twice in the translation process. This is easily done for the localization step, which remains linear time in the proof size. Lemma extraction is quadratic on DAG-like proofs because the minimal extractable sub-trees of distinct foreign inferences can overlap. In practice, though, since these sub-trees tend to be small, this effect is insignificant. Axiom elimination is in principle also quadratic on DAG's, since the definitions needed to localize each axiom instance may need to be eliminated at many nodes in the DAG. If this is a problem in practice, it can be solved by placing all instances of ELIM at the root of the derivation tree. The method of Section IV-D is also linear time for DAG-like proofs.

F. Summary of interpolation procedure

To summarize, the translation from a Z3 proof of $A, B \vdash$ to an interpolant for $A \wedge B$ proceeds in the following steps. We first push resolutions on non-local atoms upward in the derivation by using proved equivalences with local atoms. Next we convert the axiom instances to local formulas. This involves introducing fresh defined symbols, which are later eliminated using the ELIM rule. The localization phase then eliminates all closed sub-trees with local conclusions (including axiom instances) using the LOCAL rule. Lemma extraction is then used to eliminate sub-trees that cannot be represented in $\mathcal{S}_P$. This phase introduces the LEMMA rule. The resulting proof is translated inference-by-inference into a derivation in the interpolation calculus $\mathcal{S}_I$. In this process, lemmas are interpolated by calls to the secondary prover. Quantifiers are introduced in translating the ELIM rule. The result is a derivation of $(A, B) \vdash [\phi]$ where ϕ is an interpolant for $A \wedge B$.

V. EXPERIMENTAL RESULTS

In this section, we describe some experiments to evaluate the efficiency of the above approach in practice.

Our implementation is written in C++, calling directly to Z3 via its API. We use version 2.19 of Z3 without modification. This is important because we do not wish to degrade the performance of Z3 in any way, except insofar as proof generation degrades performance. Except for proof generation, we use Z3 with default options. Our secondary prover is a simple SMT solver supporting QF_UFLIA and interpolation. It uses a standard Nelson/Oppen theory combination, with theory propagation. Linear arithmetic is handled by the Simplex algorithm, with a branch-and-cut approach for integer arithmetic. Interpolation is done using essentially the system of [18], with the addition of the DIV rule of [22] to handle Gomory cuts. In principle, however, any interpolating prover that handles QF_UFLIA can be used as the secondary prover.

For benchmarks, we need a set of problems that require the power of Z3, and at the same time are representative of a realistic application of interpolation. Unfortunately, existing benchmarks are either very simple or not realistic. Earlier evaluations, such as [14], have used either formulas involving a single program execution path, or synthetic benchmarks derived from arbitrarily partitioning formulas derived from SMT-LIB benchmarks into conjuncts A and B. The former are inappropriate because by construction they are too simple to test the performance of the solver, while the latter are inappropriate because of the arbitrary partitioning. Since the performance of our method depends on locality in the proof, a realistic partitioning is essential for evaluation. Moreover, we would like to evaluate the method on problems for which interpolation is actually relevant.

For these reasons, we instead use a set of benchmark interpolation problems derived from bounded verification of safety properties of sequential and concurrent programs. These formulas are generated by the tool Poirot [15]. This tool unwinds the loops in a program and in-lines procedure calls up to some determined bound. The result is a conjunction of formulas in AUFLIA, each of which represents the semantics of a single procedure instance, plus one additional constraint representing a standard background theory and containing quantifiers. The procedure instances form a tree, such that the children of any node represent the procedures called within that node. The formulas may represent an under-approximation of the program behavior, in which case the leaf procedures are replaced by the summary FALSE, or an over-approximation, in which case the leaves are replaced by the summary TRUE. The conjunction of the formulas is satisfiable when the given safety property fails in the given over- or under-approximation.

For our benchmarks, we use the under-approximations, which are typically unsatisfiable. We choose the sub-tree rooted at an arbitrary procedure instance as the A formula, and the remainder of the conjuncts as the B formula. An interpolant for this pair is a formula involving only symbols that represent the pre-state and post-state of this particular procedure instance. Note that this can in principle be a large set of symbols, since it can include symbols representing any global variables referenced in the procedure or any of its transitive callees. This can include symbols representing the state of the heap.

We can think of the interpolant for $A \wedge B$ as a potential summary for the given procedure. It is guaranteed by the procedure instance and is sufficient to prove the given property in the given calling context. However, since the unwinding is approximate, this summary is also approximate. Nonetheless, it is possible that such approximate summaries can be used to construct true inductive summaries, as in [21], or to derive predicates for predicate abstraction, as in [9] or that they can themselves be used as approximations of procedures in further unwinding of the call tree.

For our purposes, the interest of these benchmarks is that, because they represent a large space of possible program executions, they cannot be easily solved by existing interpolating provers. To evaluate our method using these problems, we will measure two quantities: the overhead incurred in the interpolation process, relative to the run time of Z3, and the relative performance of our method compared to three existing provers. The former is easy to measure. The latter is made difficult by the fact that no interpolating provers exist that can handle the full AUFLIA theory.

To work around this problem, we will make things easier for the existing provers by providing them with the necessary quantifier instantiations and array axiom instances. We can do this by first applying axiom elimination to the Z3 proof, then applying lemma extraction to the entire proof tree, less the axiom instances. The result is an interpolation problem in QF_UFLIA. It should be kept in mind that the performance of a prover on this problem puts a lower bound on performance on the original problem, since the prover is relieved of the need to handle quantifier instantiation and the array theory.

The results are summarized in Table I. All run times are using one core of a 4-core 3.06 GHz Intel Xeon processor. Memory usage is limited to 2.5GB. The first column gives a name for the benchmark, the subscripts indicating different under-approximations and sub-trees. The "mouser", "serial" and "fdc" examples are safety properties of Windows device drivers from the Windows Static Driver Verifier [1]. The "ndisprot" and "wmm" examples are derived from threaded programs via the Lal/Reps construction [16], the latter using a weak memory model. The next two columns give the size of the A and B formulas in number of procedure instances (not considering those approximated by the summary FALSE). The next column shows the size of the Z3 proof in number of inferences. The next three columns show the Z3 run time, the interpolation time (including execution of the secondary prover) and the fractional overhead introduced by interpolation. The next column shows the number of lemmas produced.

Finally, the last three columns show the run times of three existing interpolating provers on the full problems plus quantifier and array axiom instantiations generated by Z3. Run times longer than 1800s are notated > 1800. Memory exhaustion is indicated by MEM. The MathSAT4 solver [4] supports quantifier-free linear rational and integer difference bound arithmetic. Though it does not support full LIA, we still find that it can handle the smaller problems (meaning these problems have no models in the LRA or difference bound theories). It is, however, one to two orders of magnitude slower than Z3 on these problems (note Z3 is handling quantifiers and array axioms, while MathSAT4 is not). On the larger problems, MathSAT4 exhausts memory. In two cases marked CRASH, MathSAT4 crashed. The Princess prover [3] handles UFLIA. Though in principle it can handle quantifiers, we nonetheless eliminated the quantifiers from the input formulas. Despite this, Princess failed to solve any problem within 1800s. We also tested the SMTInterpol solver, which supports QF_LIA, but this tool exhausted memory on all problems.[5]

We can make two general observations from these data. First, the overhead of interpolation relative to proof production in Z3 is small, and in fact is smaller on the larger proofs. This is in spite of the fact that the secondary prover is far less sophisticated than Z3. By dividing the proof into relative small lemmas, we have lessened the burden on the secondary prover to the point that run time is dominated by Z3.

Second, these problems are out of range for existing interpolating provers. Even with assistance provided by Z3 in instantiating quantified formulas and axioms, the best of the existing provers can handle only the smaller problems. By exploiting a state-of-the-art SMT solver, we have obtained a multiple order-of-magnitude performance improvement.

VI. Conclusion

In this work we have described an interpolating prover that is simultaneously as efficient as state-of-the-art SMT solvers and that handles the rich theory required by program verification. This was accomplished by using an efficient *proof-generating* SMT solver as a guide to a less efficient interpolating prover. By dividing the proof generated by Z3 into small lemmas, we create interpolation problems small enough for the interpolating prover to handle efficiently. In this way, we obtain a heuristically efficient interpolation procedure without requiring Z3 to produce proofs in a restricted system that allows feasible interpolation. In fact, the system does not depend on the specific set of proof rules used by Z3, with the exception of a few, such as RES and CONTRA. Thus, the Z3 proof system can potentially be expanded without any modification to the interpolation system. Moreover, any interpolating prover can be used as the secondary prover. This may allow a variety of interpolation methods to be used.

Evaluation on a set of benchmarks derived from program verification seems to indicate that the performance of an efficient solver such as Z3 can expand the range of application of interpolating provers beyond what was previously possible. This might in turn support new classes of interpolation-based algorithms for verification. One such class might be algorithms that analyze whole programs rather than program paths.

An interesting question to address in the future is how this method affects the quality of interpolants produced. Some methods have been proposed that, in effect, search the space of available proofs for one producing an interpolant satisfying certain criteria, with the goal of preventing the interpolants from diverging with deeper unwindings [11], [20]. It seems possible that using larger lemmas may allow greater flexibility to the secondary prover in constructing high quality interpolants. Thus a trade-off of performance and interpolant

[5]The benchmarks and scripts to run the provers are available at `http://www.kenmcmil.com/z3interp`.

Problem	Procedures		Proof	Time (s)		interp/Z3	Lemmas	Time (s)		
	A	B	size	Z3	interp			MathSAT4	Princess	SMTInterpol
mouserA_1	22	12	15864	0.098	0.010	0.102	5	0.986	> 1800	MEM
mouserA_2	1	34	24270	0.421	0.011	0.026	0	1.804	> 1800	MEM
mouserA_3	1	38	23331	0.232	0.008	0.034	0	1.718	> 1800	MEM
serial$_1$	111	23	69006	3.309	0.042	0.013	11	115.947	> 1800	MEM
serial$_2$	1	138	70341	3.375	0.039	0.012	0	121.928	> 1800	MEM
mouserB_1	456	12	253078	28.0	0.345	0.012	162	MEM	> 1800	MEM
mouserB_2	454	12	249548	29.4	0.276	0.009	176	MEM	> 1800	MEM
mouserB_3	1	468	269550	26.2	0.183	0.007	21	MEM	> 1800	MEM
fdc$_1$	148	5	115090	3.78	0.107	0.028	91	MEM	> 1800	MEM
fdc$_2$	1	153	114109	3.67	0.101	0.028	0	MEM	> 1800	MEM
fdc$_3$	1	155	115420	3.28	0.073	0.022	16	MEM	> 1800	MEM
ndisprot$_1$	1	29	31468	0.460	0.089	0.193	283	CRASH	> 1800	MEM
ndisprot$_2$	1	71	133863	5.61	0.208	0.037	0	CRASH	> 1800	MEM
wmm$_1$	1	2	15657	0.082	0.014	0.170	20	0.313	> 1800	MEM

TABLE I
RESULTS OF INTERPOLATION EXPERIMENTS ON POIROT FORMULAS.

quality might be achieved by adjusting the proof translation process.

Acknowledgments Thanks to Shaz Qadeer and Akash Lal for providing the formulas generated by Poirot, to Alberto Griggio for assistance with MathSAT4, to Philipp Rümmer for assistance with Princess, and to the anonymous reviewers for thorough and insightful comments.

REFERENCES

[1] Thomas Ball, Byron Cook, Vladimir Levin, and Sriram K. Rajamani. Slam and Static Driver Verifier: Technology transfer of formal methods inside Microsoft. In *IFM*, pages 1–20, 2004.

[2] Clark Barrett, Morgan Deters, Albert Oliveras, and Aaron Stump. Design and results of the 3rd annual satisfiability modulo theories competition (SMT-Comp 2007). *International Journal on Artificial Intelligence Tools*, 17(4):569–606, 2008.

[3] Angelo Brillout, Daniel Kroening, Philipp Rümmer, and Thomas Wahl. Beyond quantifier-free interpolation in extensions of Presburger arithmetic. In *VMCAI*, pages 88–102, 2011.

[4] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, and Roberto Sebastiani. The MathSAT 4 SMT solver. In *CAV*, pages 299–303, 2008.

[5] Roberto Bruttomesso, Silvio Ghilardi, and Silvio Ranise. Rewriting-based quantifier-free interpolation for a theory of arrays. In *RTA*, pages 171–186, 2011.

[6] Roberto Bruttomesso, Simone Rollini, Natasha Sharygina, and Aliaksei Tsitovich. Flexible interpolation with local proof transformations. In *ICCAD*, pages 770–777, 2010.

[7] W. Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *J. Symbolic Logic*, 22(3):269–285, 1957.

[8] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *TACAS*, pages 337–340, 2008.

[9] T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan. Abstractions from proofs. In *POPL*, pages 232–244. ACM, 2004.

[10] Krystof Hoder, Laura Kovács, and Andrei Voronkov. Interpolation and symbol elimination in vampire. In *IJCAR*, pages 188–195, 2010.

[11] Ranjit Jhala and Kenneth L. McMillan. A practical and complete approach to predicate refinement. In *TACAS*, volume 3920 of *LNCS*, pages 459–473. Springer, 2006.

[12] Deepak Kapur, Rupak Majumdar, and Calogero G. Zarba. Interpolation for data structures. In *SIGSOFT FSE*, pages 105–116, 2006.

[13] J. Krajíček and P. Pudlák. Some consequences of cryptographical conjectures for S_2^1 and EF. *Information and Computation*, 140(1):82–94, January 1998.

[14] Daniel Kroening, Jérôme Leroux, and Philipp Rümmer. Interpolating quantifier-free presburger arithmetic. In *LPAR (Yogyakarta)*, pages 489–503, 2010.

[15] Akash Lal, Shaz Qadeer, and Shuvendu Lahiri. Corral: A whole-program analyzer for Boogie. Technical Report MSR-TR-2011-60, Microsoft Research, May 2011.

[16] Akash Lal, Tayssir Touili, Nicholas Kidd, and Thomas W. Reps. Interprocedural analysis of concurrent programs under a context bound. In *TACAS*, pages 282–298, 2008.

[17] K. L. McMillan. Interpolation and SAT-based model checking. In *CAV*, volume 2725 of *LNCS*, pages 1–13. Springer, 2003.

[18] K. L. McMillan. An interpolating theorem prover. *Theor. Comput. Sci.*, 345(1):101–121, 2005.

[19] K. L. McMillan. Lazy abstraction with interpolants. In *CAV*, volume 4144 of *LNCS*, pages 123–136. Springer, 2006.

[20] Kenneth L. McMillan. Quantified invariant generation using an interpolating saturation prover. In *TACAS*, pages 413–427, 2008.

[21] Kenneth L. McMillan. Lazy annotation for program testing and verification. In *CAV*, pages 104–118, 2010.

[22] P. Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *J. Symbolic Logic*, 62(2):981–998, June 1997.