Flexible Interpolation with Local Proof Transformations

Roberto Bruttomesso Simone Rollini Natasha Sharygina Aliaksei Tsitovich

Università della Svizzera Italiana
Lugano, Switzerland
{roberto.bruttomesso,simone.fulvio.rollini,natasha.sharygina,aliaksei.tsitovich}@usi.ch

ABSTRACT

Model checking based on Craig's interpolants ultimately relies on efficient engines, such as SMT-Solvers, to log proofs of unsatisfiability and to derive the desired interpolant by means of a set of algorithms known in literature. These algorithms, however, are designed for proofs that do not contain mixed predicates. In this paper we present a technique for transforming the propositional proof produced by an SMT-Solver in such a way that mixed predicates are eliminated. We show a number of cases in which mixed predicates arise as a consequence of state-of-the-art solving procedures (e.g. lemma on demand, theory combination, etc.). In such cases our technique can be applied to allow the reuse of known interpolation algorithms. We demonstrate with a set of experiments that our approach is viable.

1. INTRODUCTION

Since the seminal work of McMillan [18–20] Craig's interpolants [10] have been extensively applied in SAT-based Model Checking and Predicate Abstraction [16]. Formally, given a pair of mutually unsatisfiable formulæ $\langle A, B\rangle$ an interpolant I is a formula such that $A \Rightarrow I$, $B \wedge I \Rightarrow \bot$, and I is defined on the common language of A and B. The interpolant I can be thought of as an over-approximation of A that still conflicts with B.

There exists a number of the state-of-the-art approaches for the automated generation of interpolants. Pudlák [23] studies it in the context of propositional logic. McMillan [19] proposes an alternative method that also handles the quantifier-free theories of uninterpreted functions, linear arithmetic, and their combination.

Yorsh and Musuvathi [24] generalize McMillan's procedure by proposing an interpolant-generation technique for theory combination. The method relies on the Nelson-Oppen procedure [22] to compute partial interpolants using the communication of interface equalities between the reasoners for the theories to combine. Also, they show how to extend Pudlák's method to quantifier-free theories, providing de facto a modular way for computing interpolants in the context of SMT-Solvers.

Cimatti et al. [9] propose a set of techniques to efficiently compute interpolants in SMT for a number of arithmetic theories (difference logics, utpvi, linear rational arithmetic). They also show how to adapt delayed theory combination (DTC) [6] (a flexible variant of the Nelson-Oppen combination) to obtain interpolants in a union of theories.

The aforementioned state-of-the-art methods [9, 24] compute interpolants in two steps. First, the formula $A \wedge B$ is solved using an SMT-Solver with proof-logging enabled, and then the interpolant is derived from the structure of the resolution proof.

This method, however, requires the proof not to contain AB-mixed predicates, i.e., predicates defined on symbols that are local to A and B. However, there is a number of techniques that (potentially) require the addition of AB-mixed predicates to the input formula before or during solving time [1, 3, 6, 21]. These techniques are extensively used in the state-of-the-art SMT-Solvers.

In this paper we present a technique that rewrites propositional proofs in such a way that it isolates and removes AB-mixed predicates. Our method enables the use of off-the-shelf techniques for solving $A \wedge B$, and, consequently, after the proof is transformed, the application of the method of [24] for computing interpolants.

The proof transformation is based on a set of rules that locally swap pivots in the propositional proof, so that the AB-mixed predicates can be isolated and removed. Even though the application of the rules could in principle lead to an exponential growth of the proof, we demonstrate by means of experiments that in practice the size of the transformed proof is comparable (in some cases even inferior) to the original proof.

The paper is organized as follows. §2 recalls some basic notions, presents an overview of the approach, and discusses the related work. §3 describes our proof transformation algorithm. §4 describes a set of examples in which our approach can be applied. §5 describes experiments that demonstrate that our proof-transformation approach, in practice, does not lead to exponential blow-ups in the proof size. §6 draws the conclusions.

2. PRELIMINARIES

In this paper we shall use A and B to denote two quantifier-free formulæ in a theory $\mathcal{T}$, for which we would like to compute an interpolant. Theories of interest are equality with uninterpreted functions $\mathcal{EUF}$, linear arithmetic over the rationals $\mathcal{LRA}$ and the integers $\mathcal{LIA}$, the theory of arrays $\mathcal{AX}$, or a combination of theories, such as $\mathcal{EUF} \cup \mathcal{LRA}$. Variables that appear only in A or B are called A-local and B-local respectively. Variables that appear in both A and B are called AB-common. A predicate is called AB-mixed if it is defined on both A-local and B-local variables, it is called AB-pure otherwise. Notice that AB-mixed predicates cannot appear in A and B. Sometimes we will say that a clause or a variable is *colored* A, meaning that it belongs to A.

Example 1. *Let $A \equiv (a = e \wedge f(a) = c)$, $B \equiv (b = e \wedge f(b) = d \wedge c \neq d)$ be two formulæ in the $\mathcal{EUF}$ theory. Variable a is A-local, b is B-local, c, d, e are AB-common (a predicate $a = b$ would be AB-mixed). An interpolant I for $A \wedge B$ is $f(e) = c$, which is an AB-pure predicate.*

In the following we shall use p, q, r (possibly with subscripts) to denote Boolean variables, s, t to denote literals, $\alpha, \beta, \ldots$ to denote clauses, and $C, D, \ldots$ to denote subclauses. The empty clause is denoted by $\bot$. We will write clauses as lists of literals and sub-clauses, omitting the "$\vee$" symbol, as for instance $p\overline{q}C$. We will use the form $\alpha \subseteq \beta$ to indicate that α *subsumes* β, that is the set of literals α is a subset (note necessarily proper) of the set of literals β. Also we will assume that clauses do not contain duplicated literals or both the occurrences of a literal and its negation. Finally, we use $var(s)$ to denote the variable associated with the literal s.

2.1 Resolution Proofs

Resolution is the following proof rule:

$$\frac{pC \qquad \overline{p}D}{CD}\, p$$

Clauses pC and $\overline{p}D$ are called *antecedents*, CD is the *resolvent*, and p is the *pivot* variable. Throughout the paper we shall use the notion of *resolution proof.*

Definition 1 (Resolution Proof). *A* resolution proof *of a clause λ from a set of clauses $\mathbb{S}$ is a tree such that* (*i*) *its leaves are clauses in $\mathbb{S}$,* (*ii*) *the root is λ,* (*iii*) *intermediate clauses are derived by means of an application of the resolution rule.*

We say that a proof P is a *proof of unsatisfiability* if $\lambda \equiv \bot$. Finally a subproof P', rooted in μ, of a proof P is the proof subtree that derives μ from a subset of leaves of P.

Notice that since we deal with SMT formulæ the variables in the resolution proof may represent the Boolean abstraction of a theory predicate in a theory $\mathcal{T}$, such as $x + y < 1$. In this case we say that a variable is AB-mixed if so is the predicate associated with it.

2.2 Overview and Previous Work

Pudlák shows in [23] a method to compute an interpolant from a proof of unsatisfiability of a formula in pure propositional logic. An interpolant can be computed by traversing the proof, taking into account the color of pivots and leaf clauses.

Yorsh and Musuvathi present in [24] a generalization of Pudlák's method that can compute interpolants for a formula defined modulo a theory $\mathcal{T}$. The leaves of the proof of unsatisfiability in this case are original clauses as well as *theory lemmata* (clauses that encode valid facts in $\mathcal{T}$) involving original predicates, generated by the prover during the solving process. It is then sufficient to compute a partial interpolant for each theory lemma in order to derive the global interpolant.

The last technique, for its modularity, finds its natural implementation within SMT-Solvers [4], tools that combine SAT-Solvers and domain specific procedures for a theory $\mathcal{T}$ in an efficient way. Cimatti et al. [9] show that interpolant generation within SMT-Solvers can outperform other known methods (e.g. [19]), as a result of using optimized domain-specific procedures for $\mathcal{T}$.

One limitation of the approach of [24] is that theory lemmata, appearing in the proof of unsatisfiability, must not contain AB-mixed predicates. However, several decision procedures defined for SMT-Solvers heavily rely on the creation of new predicates during the solving process. Examples are delayed theory combination (DTC) [6], Ackermann's Expansion [1], Lemmas on Demand [21] and Splitting on Demand [3] (see §4). All these methods may introduce new predicates, which can potentially be colored as AB-mixed.

In [9] the problem is addressed, only for the case of DTC, by tweaking the decision heuristics of the solver, in such a way that it guarantees that the produced proof can be handled with known methods. In particular they define a notion of *ie*-local proofs, and they show (*i*) how to compute interpolants for this class of proofs, (*ii*) how to adapt the SMT-Solver to produce only *ie*-local proofs. In [15] the authors relax the constraint on generating *ie*-local proofs by introducing the notion of *almost-colorable* proofs. We argue that our technique is simpler and more flexible, as different strategies can be derived with different applications of our local transformation rules. Also there is no experimental evidence that method of [15] can be applied in practice.

In this paper we also show how to compute an AB-pure proof from an AB-mixed one but without interfering with the internals of the SMT-Solver. Our method is more general, i.e., it applies not only to theory combination but to any approach that requires the addition of AB-mixed predicates (see §4 for a set of examples). We define a set of local transformations that can effectively modify the proof in such a way that the generic method of [24] can be applied. In this way it is possible to achieve a complete decoupling between the solving phase and the interpolant-generation phase, provided that an interpolant-generation procedure is available for a conjunction of input constraints in $\mathcal{T}$.

The sketch of the approach is depicted in Figure 1. The idea is to move all AB-mixed predicates (in grey) toward the leaves of the proof (Figure 1b) within maximal AB-mixed subproofs.

Definition 2 (AB-mixed sub-proof). *Given a resolution proof P, an* AB-mixed sub-proof *is a sub-proof of P rooted in a clause λ, whose intermediate pivots are all AB-mixed predicates. An AB-mixed sub-proof is* maximal *if the root λ does not contain AB-mixed predicates.*

When dealing with a background theory $\mathcal{T}$ we have the following remark.

Observation 1. *Let P' be a maximal AB-mixed sub-proof rooted in λ. Then λ is a valid theory lemma for $\mathcal{T}$.*

This observation derives from Definition 2 and from the fact that (*i*) AB-mixed predicates can only appear in theory

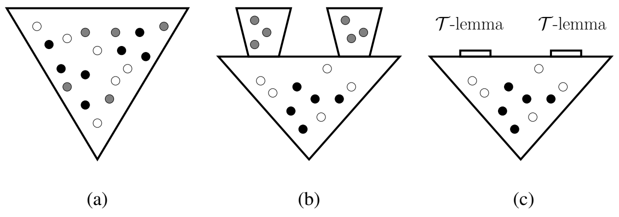

Figure 1: An overview of our approach. (a) is the proof generated by the SMT-Solver. White points represent A-local predicates, black points represent B-local predicates, grey points represent AB-mixed predicates. (b) AB-mixed predicates are confined inside AB-mixed trees. (c) AB-mixed trees are removed and their roots are valid theory lemmata in $\mathcal{T}$.

lemmata (as they do not appear in the original formula) and (*ii*) a propositional resolution of two theory lemmata generates another theory lemma.

Once AB-mixed maximal sub-proofs are formed, it is possible to replace them with their root clauses (Figure 1c). The obtained proof is now free of AB-mixed predicates and can be used to derive an interpolant using the method of [24], provided that an interpolant-generating procedure is available for $\mathcal{T}$.

The crucial part of our approach is an algorithm for proof transformation. It relies on a set of transformation rules that extends those introduced in [17]. These rules cannot be used directly for proof transformations: they may derive proof consequences that are weaker than those in the original proof.

Our paper instead is based on a set of rules that never derives weaker consequences. The exhaustive application of the rules can be used to transform a proof P into a proof P', where all AB-mixed variables are confined in AB-mixed sub-proofs. Each rewrite rule can effectively swap two pivots p and q in the resolution proof, or perform simplifications, depending on a particular context.

In the following to facilitate the understanding of the algorithm we will call AB-mixed and AB-pure predicates *light* and *heavy* respectively. Our rules are applied when a light predicate is below a heavy predicate in the proof tree. The effect of the exhaustive application of our rules is to lift light predicates over heavy predicates as bubbles in water.

2.2.1 Other Related Work

In the context of proof and interpolants, D'Silva et al. [12] illustrate a set of global transformation rules and an algorithm that reorder the proof w.r.t. a partial order among pivots. Our approach works instead locally, and leaves more freedom in choosing the strategies for rule applications. Also our focus is not directly in computing interpolants, but rather in rewriting the proof in such a way that previous techniques could be applied.

The same authors in [13] focus on the concept of strength of an interpolant. They present an analysis and a comparison of existing interpolation systems, together with a method to combine systems in order to obtain weaker or stronger interpolants from a given proof of unsatisfiability. They also discuss the use and the limitations of the local transformation rules of Jhala and McMillan [17].

3. MANIPULATING THE PROOF

In this section we describe our proof transformation framework in terms of simple transformation rules. We show how to adjust the proof with a proof-reconstruction algorithm, and we show how to derive an algorithm for lifting light variables over heavy variables.

3.1 Local Proof Transformation Rules

$$\cfrac{\cfrac{\beta \qquad \gamma}{\delta}\,p \qquad \alpha}{\eta}\,q$$

Figure 2: Rule context

Figure 2 shows two consecutive resolution steps of a generic proof: we shall call a similar structure a *context*. A context involves two pivots p and q and five clauses $\beta, \gamma, \delta, \alpha, \eta$. Clearly p is contained in β and γ (with opposite polarity), and q is contained in δ and α (again with opposite polarity). The reader may observe that q must be contained in $\beta \cup \gamma$.

Figure 3 shows five proof transformation rules. Each rule is associated with a different context, and, conversely, each context can be mapped to exactly one rule (i.e., the set of rules is exhaustive, modulo symmetry, for every possible context).

The transformations corresponding to $A1$ and $A2$ were first introduced in [17] and further discussed in [13]. We devised the remaining three rules after a careful analysis of all the possible contexts. Notice that a side-effect of the new three rules is that they generate proofs with stronger roots.

A first property that distinguishes our set of rules is *locality*: only the limited information represented by a context is in fact needed to determine which rule is applicable.

The rules can be effectively employed to perform a *local reordering* of the pivots. Each rule either swaps the position of two pivots ($A1$, $A2$, $B2$), or it eliminates at least one pivot ($B1$, $B3$). Later in the section, it will be shown how this characteristic can be used to create an application strategy aimed at sorting the pivots in a proof.

Notice also that rules $B1$-$B3$ produce a clause $\eta' \subseteq \eta$. The literals that are in $\eta \setminus \eta'$ need not to be resolved anymore down in the proof. The proof can be therefore effectively simplified. Algorithm 1 has exactly the purpose of propagating the effect of the replacement of η by $\eta' \subseteq \eta$ along the path leading from η' to the empty clause.

3.2 The Propagation Algorithm

As mentioned in §2, a resolution proof can be thought of as a tree. A node n is associated with an occurrence of a clause in the proof (we will refer to it as n_{cl}) and for each pair of nodes m and n there is a directed edge $m \rightarrow n$ if and only if m_{cl} is an antecedent of n_{cl} (by extension, we will call m an antecedent of n and m a resolvent of n). For each n which is not a leaf, n^{ant1} and n^{ant2} will be the two antecedents (in an arbitrary order), n_{piv} the pivot of the resolution step of which n_{cl} is the resolvent.

The idea of the algorithm reflects the mechanisms of the restructuring procedures proposed in [2, 12]:

1. it determines the effect range of the substitution of η with η', which corresponds to the set of nodes reachable from the node associated with η';

Case $A1$: $s \notin \alpha$, $t \in \gamma$

$$\dfrac{\dfrac{\beta : stC \qquad \gamma : \bar{s}tD}{\delta : tCD}\,var(s) \qquad \alpha : \bar{t}E}{\eta : CDE}\,var(t) \quad \Rightarrow \quad \dfrac{\dfrac{\beta : stC \qquad \alpha : \bar{t}E}{\delta' : sCE} \qquad \dfrac{\alpha : \bar{t}E \qquad \gamma : \bar{s}tD}{\delta'' : \bar{s}DE}\,var(t)}{\eta' : CDE}\,var(s)$$

Case $A2$: $s \notin \alpha$, $t \notin \gamma$

$$\dfrac{\dfrac{\beta : stC \qquad \gamma : \bar{s}D}{\delta : tCD}\,var(s) \qquad \alpha : \bar{t}E}{\eta : CDE}\,var(t) \quad \Rightarrow \quad \dfrac{\dfrac{\beta : stC \qquad \alpha : \bar{t}E}{\delta' : sCE}\,var(t) \qquad \gamma : \bar{s}D}{\eta' : CDE}\,var(s)$$

Case $B1$: $s \in \alpha$, $t \in \gamma$

$$\dfrac{\dfrac{\beta : stC \qquad \gamma : \bar{s}tD}{\delta : tCD}\,var(s) \qquad \alpha : s\bar{t}E}{\eta : sCDE}\,var(t) \quad \Rightarrow \quad \dfrac{\beta : stC \qquad \alpha : s\bar{t}E}{\eta' : sCE}\,var(t)$$

Case $B2$: $s \in \alpha$, $t \notin \gamma$

$$\dfrac{\dfrac{\beta : stC \qquad \gamma : \bar{s}D}{\delta : tDC}\,var(s) \qquad \alpha : s\bar{t}E}{\eta : sCDE}\,var(t) \quad \Rightarrow \quad \dfrac{\dfrac{\beta : stC \qquad \alpha : s\bar{t}E}{\delta' : sCE}\,var(t) \qquad \gamma : \bar{s}D}{\eta' : CDE}\,var(s)$$

Case $B3$: $\bar{s} \in \alpha$, $t \notin \gamma$

$$\dfrac{\dfrac{\beta : stC \qquad \gamma : \bar{s}D}{\delta : tCD}\,var(s) \qquad \alpha : \bar{s}\bar{t}E}{\eta : \bar{s}CDE}\,var(t) \quad \Rightarrow \quad \eta' : \bar{s}D$$

Figure 3: Local transformation rules.

Step 1: application of a $B2$ rule

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{pq} \qquad \mathbf{\bar{p}r}}{\mathbf{qr}}\,\mathbf{p} \qquad \mathbf{p\bar{q}}}{\mathbf{pr}}\,\mathbf{q} \qquad \bar{p}u}{ru}\,p \qquad \bar{r}v}{uv}\,r \quad \Rightarrow \quad \dfrac{\dfrac{\dfrac{\dfrac{\mathbf{pq} \qquad \mathbf{p\bar{q}}}{\mathbf{p}}\,\mathbf{q} \qquad \mathbf{\bar{p}r}}{\mathbf{r}}\,\mathbf{p} \qquad \bar{p}u}{ru}\,p \qquad \bar{r}v}{uv}\,r$$

Step 2: elimination of an unnecessary resolution step (case 2)

$$\dfrac{\dfrac{\dfrac{\dfrac{pq \qquad p\bar{q}}{p}\,q \qquad \bar{p}r}{\mathbf{r}}\,p \qquad \mathbf{\bar{p}u}}{\mathbf{ru}}\,\mathbf{p} \qquad \bar{r}v}{uv}\,r \quad \Rightarrow \quad \dfrac{\dfrac{\dfrac{pq \qquad p\bar{q}}{p}\,q \qquad \bar{p}r}{\mathbf{r}}\,p \qquad \bar{r}v}{uv}\,r$$

Step 3: update of a resolving clause (case 1)

$$\dfrac{\dfrac{\dfrac{pt \qquad p\bar{t}}{p}\,t \qquad \bar{p}r}{\mathbf{r}}\,p \qquad \mathbf{\bar{r}v}}{\mathbf{uv}}\,\mathbf{r} \quad \Rightarrow \quad \dfrac{\dfrac{\dfrac{pt \qquad p\bar{t}}{p}\,t \qquad \bar{p}r}{\mathbf{r}}\,p \qquad \mathbf{\bar{r}v}}{\mathbf{v}}\,\mathbf{r}$$

Figure 4: Example of rule application and subsumption propagation.

Algorithm 1: Subsumption propagation

```
Input: A proof perturbated by a $B$-rule
Output: A reconstructed proof
Data: $R$: set of nodes reachable from $n_{\eta'}$, $V$: set of
      visited nodes
begin
    $V \leftarrow \emptyset$;
    Determine $R$, for example through a visit from $n_{\eta'}$;
    while $R \setminus V \neq \emptyset$ do
        Choose $n \in R \setminus V$ such that:
        ($n^{ant1} \in R \cap V$ or $n^{ant1} \notin R$) and
        ($n^{ant2} \in R \cap V$ or $n^{ant2} \notin R$);
        $V \leftarrow V \cup \{n\}$;
        if $n_{piv} \in n_{cl}^{ant1}$ and $n_{piv} \in n_{cl}^{ant2}$ then
            $n_{cl} \leftarrow Res(n_{cl}^{ant1}, n_{cl}^{ant2})$
        else if $n_{piv} \notin n_{cl}^{ant1}$ and $n_{piv} \in n_{cl}^{ant2}$ then
            Substitute $n$ with $n^{ant1}$;
        else if $n_{piv} \in n_{cl}^{ant1}$ and $n_{piv} \notin n_{cl}^{ant2}$ then
            Substitute $n$ with $n^{ant2}$;
        else if $n_{piv} \notin n_{cl}^{ant1}$ and $n_{piv} \notin n_{cl}^{ant2}$ then
            Choose an antecedent $n^{ant}$;
            Substitute $n$ with $n^{ant}$;
    end
end
```

2. it analyzes, one by one, all the reachable nodes; it is necessary that the antecedents of a node n have already been visited (and possibly modified), in order to guarantee a correct propagation of the modifications to n_{cl};

3. due to the potential vanishing of literals from clauses, it might happen that in some resolution step the pivot is not present in both antecedents anymore; if that is the case, we delete such resolution step, by replacing the resolvent with the antecedent devoid of the pivot (if the pivot is missing in both antecedents, either of them is arbitrarily chosen), otherwise, we keep the step while updating the resolvent clause. At the graph level, we substitute n with n^{ant1} or n^{ant2}, assigning the resolvents of n (if any) to it.

Notice that this algorithm deals with the general case of *proof graphs*, where the propagation process could affect an arbitrarily large subgraph instead of a single path leading to the empty clause.

3.3 Pivot Reordering

We stated in §2.2 that the main purpose of introducing the set of rules is to obtain a transformation of a proof P into a proof P' such that all light variables are above heavy variables in the proof.

In order to achieve this goal it is sufficient to consider only *unordered* contexts, i.e. those in which $var(t)$ is a light variable and $var(s)$ is a heavy variable. Therefore a simple non-deterministic algorithm can be derived as follows:

Algorithm 2: Pivot reordering

```
Input: A proof
Output: A proof without unordered contexts
Data: $U$: set of unordered contexts
begin
    Determine $U$, for example through a visit of the
    proof tree;
    while $U \neq \emptyset$ do
        Choose a context in $U$;
        Apply the associated rule, and the propagation
        algorithm if necessary;
        Update $U$;
    end
end
```

The algorithm terminates. Notice in fact that each iteration strictly decreases the distance of an occurrence of a heavy pivot w.r.t. $\perp$, until no more unordered contexts are left.

3.4 Proof Graphs

Up until now the discussion was related to proof trees, where each node had two incoming edges (from its antecedents) and exactly one outgoing edge (to its resolvent); there was no restriction on the number of graph occurrences of a clause, which in fact could appear more than once, associated with different nodes.

In order to avoid this redundancy while achieving a more compact data structure, it is usual to represent a proof with a *proof graph*. The only significant difference from a proof tree consists in the possibility for a node n to have more than one outgoing edge, since it is linked to all the nodes whose clauses are the resolvents of n_{cl} in the proof.

A few remarks are in order:

1. if the clause δ participates as antecedent in more than one resolution step (i.e. the associated node has more than one outgoing edge) then a rule cannot now be directly applied. In fact, the modification of δ would indirectly affect other steps besides the one present in the current context. We use a simple solution that consists of creating a copy $n_{\delta'}$ of n_δ, to which we assign all the edges of n_δ, except $n_\delta \to n_\eta$, before applying a rule.

2. Having multiple outgoing edges for n_β, n_γ, n_α does not add any difficulty: the application of a rule will take into account a single edge corresponding to an occurrence of a clause as antecedent, leaving the other edges untouched.

3. As for the propagation of the effect of B rules, the set of nodes reachable from n_η is likely to be a subgraph rather than a simple path. Our algorithm takes into account the fact that the propagation might happen along multiple paths.

4. APPLICATIONS

In this section we show a number of techniques currently employed in state-of-the-art SMT-Solvers that can potentially introduce AB-mixed predicates during the solving phase. If these predicates become part of the proof of unsatisfiability, the proof-reordering algorithm described in §3.3 can be applied to produce an AB-pure proof.

Id	Clauses	Propositional abstraction
1	$x = wr(y,i,e)$	p_1
2	$rd(x,j) \neq rd(y,j)$	$\overline{p_2}$
3	$rd(x,k) \neq rd(y,k)$	$\overline{p_3}$
4	$j \neq k$	$\overline{p_4}$
5	$(i = j \vee rd(wr(y,i,e),j) = rd(y,j))$	$p_5\, p_6$
6	$(i = k \vee rd(wr(y,i,e),k) = rd(y,k))$	$p_7\, p_8$
7	$(x \neq wr(y,i,e) \vee rd(x,j) = rd(y,j) \vee rd(wr(y,i,e),j) \neq rd(y,j))$	$\overline{p_1}\, p_2\, \overline{p_6}$
8	$(x \neq wr(y,i,e) \vee rd(x,k) = rd(y,k) \vee rd(wr(y,i,e),k) \neq rd(y,k))$	$\overline{p_1}\, p_3\, \overline{p_8}$
9	$(j = k \vee i \neq j \vee i \neq k)$	$p_4\, \overline{p_5}\, \overline{p_7}$

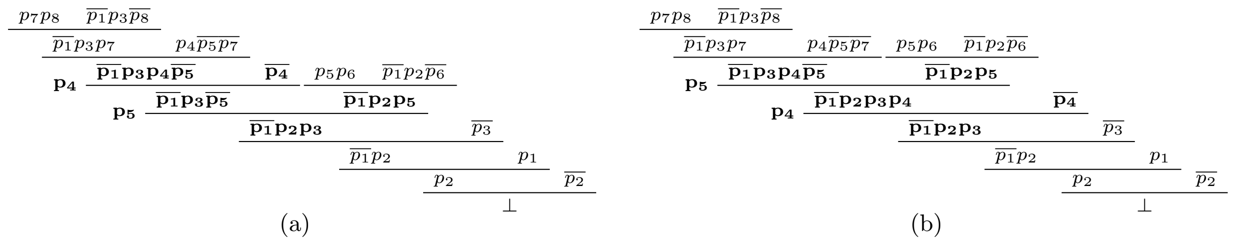

Figure 5: Clauses from Example 2. $\varphi \equiv \{1,2,3,4\}$, $\psi \equiv \{5,6\}$. Clauses 7-9 are theory lemmata discovered by the $\mathcal{EUF}$ solver. (a) is a possible proof obtained by the SMT-Solver (for $\mathcal{EUF}$) on $\varphi \wedge \psi$. (b) is a proof after swapping p_4 and p_5 by means of rule A2; in the resulting proof all mixed literals (p_5-p_8) appear in the upper part of the proof in an AB-mixed proof subtree. The root of the AB-mixed subtree $\overline{p_1}p_2p_3p_4$ is a valid theory lemma in $\mathcal{AX}$.

4.1 Reduction Techniques

Let $\mathcal{T}_k$ and $\mathcal{T}_j$ be two quantifier-free decidable theories such that $\mathcal{T}_k$ is weaker (less expressive) than $\mathcal{T}_j$. Given a $\mathcal{T}_j$-formula φ, and a decision procedure SMT($\mathcal{T}_k$) for quantifier-free formulæ in $\mathcal{T}_k$, it is often possible to obtain a decision procedure SMT($\mathcal{T}_j$) for quantifier-free formulæ in $\mathcal{T}_j$ by augmenting φ with a finite set of $\mathcal{T}_k$-lemma ψ. These lemmata (or axioms) explicitly encode the necessary knowledge such that $\mathcal{T}_k \models \varphi \wedge \psi$ if and only if $\mathcal{T}_j \models \varphi$. Therefore a decision procedure for $\mathcal{T}_j$ is simply:

Algorithm 3: A reduction approach for SMT_j

```
Input: $\varphi$ for $\mathcal{T}_j$
begin
    $\psi$ = generateLemmata($\varphi$);
    return $SMT(\mathcal{T}_k)(\varphi \wedge \psi)$;
end
```

In practice the lemmata generation function can be made lazy by plugging it inside the SMT-Solver directly. This paradigm is known as Lemma on Demand [21] or Splitting on Demand [3][1]. This feature does not affect our method. We show some reduction techniques as follows.

4.1.1 Reduction of $\mathcal{AX}$ to $\mathcal{EUF}$

We consider the case where $\mathcal{T}_k \equiv \mathcal{EUF}$, the theory of equality with uninterpreted functions, and $\mathcal{T}_j \equiv \mathcal{AX}$, the theory of arrays with extensionality. The axioms of $\mathcal{EUF}$ are the ones of equality (reflexivity, symmetry, and transitivity) plus the congruence axioms $\forall x, y.\ x = y \Rightarrow f(x) = f(y)$, for any functional symbol of the language.

[1]The second is often used to indicate the case where some branching, that is supposed to occur inside a theory-solver, is delegated to the underlying Boolean layer.

The theory of arrays $\mathcal{AX}$ is instead axiomatized by:

$$\forall x, i, e. \quad rd(wr(x,i,e),i) = e \tag{1}$$

$$\forall x, i, j, e. \quad i = j \vee rd(wr(x,i,e),j) = rd(x,j) \tag{2}$$

$$\forall x, y. \quad x = y \Leftrightarrow (\forall i.\ rd(x,i) = rd(y,i)) \tag{3}$$

State-of-the-art approaches for $\mathcal{AX}$ implemented in SMT-Solvers [5,7,11,14] are all based on reduction to $\mathcal{EUF}$. Instances of the axioms of $\mathcal{AX}$ are added to the formula in a lazy manner until either the formula is proven unsatisfiable or saturation is reached. The addition of new lemmata may require the creation of AB-mixed predicates when a partitioned formula is considered.

Example 2. *Let $\varphi \equiv A \wedge B$, where $A \equiv x = wr(y,i,e)$, and $B \equiv rd(x,j) \neq rd(y,j) \wedge rd(x,k) \neq rd(y,k) \wedge j \neq k$. Variables $\{i,e\}$ are A-local, $\{j,k\}$ are B-local, and $\{x,y\}$ are AB-common. To prove φ unsatisfiable with a reduction to $\mathcal{EUF}$, we need to instantiate axiom (2) twice as $\psi \equiv (i = j \vee rd(wr(y,i,e),j) = rd(y,j)) \wedge (i = k \vee rd(wr(y,i,e),k) = rd(y,k))$. Notice that we introduced four AB-mixed predicates. Now we can send $\varphi \wedge \psi$ to a SMT-Solver for $\mathcal{EUF}$ to produce the proof of unsatisfiability. Figure 5 shows a possible resolution proof generated by the SMT-Solver, and how it can be transformed into a proof without AB-mixed predicates.*

4.1.2 Reduction of $\mathcal{LIA}$ to $\mathcal{LRA}$

Decision procedures for $\mathcal{LIA}$ (linear integer arithmetic) often rely on iterated calls to a decision procedure for $\mathcal{LRA}$ (linear rational arithmetic). An example is the well-known method of "branch-and-bound": given a feasible rational region R for a set of variables $\vec{x} = (x_1, \ldots, x_n)$, and a non-integer point $\vec{c} \in R$ for $\vec{x}$, then one step of "branch-and-bound" generates the two subproblems $R \cup \{x_i \leq \lfloor c_i \rfloor\}$ and $R \cup \{x_i \geq \lceil c_i \rceil\}$. These are again recursively explored until an integer point $\vec{c}$ is found.

Notice that the splitting on the bounds can be delegated to the Boolean engine by adding the lemma $((x_i \leq \lfloor c_i \rfloor) \vee (x_i \geq \lceil c_i \rceil))$. In order to obtain a faster convergence of the algorithm, it is possible to split on *cuts*, i.e. linear constraints, rather than on simple bounds. However cuts may add AB-mixed predicates if A-local and B-local variables are mixed into the same cut.

EXAMPLE 3. *We are given three $\mathcal{LIA}$ (linear integer arithmetic) constraints: Let $\varphi \equiv A \wedge B$, where $A \equiv 5x - y \leq 1 \wedge y - 5x \leq -1$, and $B \equiv 5z - y \leq -2 \wedge y - 5z \leq 3$. The axiom $\psi \equiv ((x - z \leq 0) \vee (x - z \geq 1))$ (which contains two AB-mixed literals) is sufficient for $\varphi \wedge \psi$ to be proven unsatisfiable by a solver for $\mathcal{LRA}$, by discovering two additional theory lemmata $((5x - y \not\leq 1) \vee (y - 5z \not\leq 3) \vee (x - z \leq 0))$ and $((5x - y \not\leq -1) \vee (y - 5z \not\leq -2) \vee (x - z \geq 1))$.*

4.1.3 Ackermann's Expansion

When $\mathcal{T}_j$ is a combination of theories of the form $\mathcal{EUF} \cup \mathcal{T}_j$, Ackermann's expansion [1] can be used to reduce the reasoning from $\mathcal{T}_j$ to $\mathcal{T}_k$. The idea is to use as ψ the exhaustive instantiation of the congruence axiom $x = y \Rightarrow f(x) = f(y)$ for all pairs of variables appearing in uninterpreted functional symbols and all uninterpreted functional symbols f in φ. This instantiation generates AB-mixed predicates when x is A-local and y is B-local.

EXAMPLE 4. *Let $\mathcal{T}_k \equiv \mathcal{LRA}$. Let $\varphi \equiv A \wedge B$ and $A \equiv (a = x + y \wedge f(a) = c)$, $B \equiv (b = x + y \wedge f(b) = d \wedge c \neq d)$. The axiom $\psi \equiv ((a \neq b) \vee (f(a) = f(b))$ is sufficient for $\mathcal{LRA}$ to detect the unsatisfiability of $\varphi \wedge \psi$, by discovering two additional theory lemmata $((f(a) \neq f(b)) \vee (f(a) \neq c) \vee (f(b) \neq d) \vee (c \neq d))$ and $((a \neq x+y) \vee (b \neq x+y) \vee (a = b))$. (Notice that $f(a)$ and $f(b)$ are treated by $\mathcal{LRA}$ as if they were two variables).*

4.2 Theory Combination via DTC

A generic framework for theory combination was introduced by Nelson and Oppen in [22]. We recall it briefly as follows.

Given two signature-disjoint and stably-infinite theories $\mathcal{T}_1$ and $\mathcal{T}_2$, a decision procedure for a conjunction of constraints in the combined theory $\mathcal{T}_1 \cup \mathcal{T}_2$ can be obtained from the decision procedures for $\mathcal{T}_1$ and $\mathcal{T}_2$. First, the formula φ is *flattened*, i.e. auxiliary variables are introduced to separate terms that contain both symbols of $\mathcal{T}_1$ and $\mathcal{T}_2$. Then the idea is that the two theory-solvers for $\mathcal{T}_1$ and $\mathcal{T}_2$ are forced to exhaustively exchange *interface equalities* i.e. equalities between *interface variables* (interface variables are those that appear both in constraints of $\mathcal{T}_1$ and $\mathcal{T}_2$ after flattening)[2].

Delayed Theory Combination (DTC) implements a nondeterministic version of the Nelson-Oppen framework, in which interface equalities are not exchanged by the deciders directly, but they are guessed by the SAT-Solver. With DTC it is possible to achieve a higher level of modularity w.r.t. the classical Nelson-Oppen framework. DTC is currently implemented (with some variation) in most state-of-the-art SMT-Solvers.

If the set of shared variables contains only A-local or B-local symbols, than no AB-mixed interface equality is generated, and an interpolant can be derived with the methods already present in literature. Otherwise our method can be applied to reorder the proof, as an alternative to the techniques described in [9, 15].

EXAMPLE 5. *Consider again φ of Example 4. Since a, b, $f(a)$, $f(b)$ appear in constraints of both theories, we need to generate two interface equalities $a = b$ and $f(a) = f(b)$. The guessing of their polarity is delegated to the SAT-Solver. The SMT-Solver will detect the unsatisfiability after the $\mathcal{EUF}$-solver discovers the two theory-lemmata $((a \neq b) \vee (f(a) = f(b))$ and $((f(a) \neq f(b)) \vee (f(a) \neq c) \vee (f(b) \neq d) \vee (c \neq d))$ and the $\mathcal{LRA}$-solver discovers the theory-lemma $((a \neq x + y) \vee (b \neq x + y) \vee (a = b))$. (Notice, again, that $f(a)$ and $f(b)$ are treated by $\mathcal{LRA}$ as if they were two variables).*

5. EXPERIMENTS

For the purpose of this experimentation we have chosen to focus on one particular application of §4, namely Ackermann's Expansion for theory combination.

We evaluated the proof transformation technique on the set of `QF_UFIDL` formulæ from the SMT-LIB[3] (`QF_UFIDL` encodes formulæ in the combined theory $\mathcal{EUF} \cup \mathcal{IDL}$). The suite contains 319 unsatisfiable instances. Each formula was split in half to obtain an artificial interpolation problem (in the same fashion as [9])[4].

The proof transformation framework is implemented in OpenSMT [8] which features a preliminary support for generation and transformation of proofs along the lines of §3. Proof transformation is applied when the proof contains AB-mixed predicates, in order to lift them up inside AB-maximal sub-proofs and replace them with their roots.

We ran the experiments on an Ubuntu server equipped with Dual-Core 2GHz Opteron 2212 CPU and 4GB of memory. The benchmarks were executed with a time-out of 60 minutes and a memory threshold of 2GB (whatever is reached first): 172 instances, of which 82 proofs contained AB-mixed predicates[5], were successfully handled within these limits. We have reported the cost of the transformation and its effect on the proof; the results are summarized in Table 5. We grouped benchmarks together following the original classification used in SMT-LIB and provided average values for each group[4].

The results in Table 5 demonstrate that our proof transformation technique induces, on average, about than 13% overhead with respect to plain solving time (recall that our implmentation is still at a rather preliminary stage – to come up with an efficient implementation of the proof transformation algorithm is part of the ongoing work). However, what is important for this experimental evaluation is to show that no blow-up in the proofs size takes place during the process.

The average increase in size is around 74%, but not all the instances experienced a growth; we observed in fact that in 42 out of 82 benchmarks the transformed proof was smaller then the original one both in the number of nodes and edges. Overall it is important to point out that we never experienced any exponential blowup in the size of the proof during the transformation.

[2]Notice that in practice flattening can be avoided. For instance in Example 5 we do not perform any flattening.

[3]`http://www.smt-lib.org`

[4] The Benchmarks and the detailed results are available at `http://verify.inf.usi.ch/opensmt/iccad2010`.

[5]Notice that in some cases AB-mixed predicates were produced during the search, but they did not appear in the proof.

Group	#	$\#AB$	$\%_{time}$	$\%_{nodes}$	$\%_{edges}$
RDS	2	7	84%	-16%	-19%
EufLaArithmetic	2	74	18%	187%	193%
pete	15	20	16%	66%	68%
pete2	52	13	6%	73%	80%
uclid	11	12	29%	87%	90%
Overall	82	16	13%	74%	79%

Table 1: The effect of proof transformation on `QF_UFIDL` benchmarks summarized per group: # — number of benchmarks in a group, $\#AB$ — average number of AB-mixed predicates in a proof, $\%_{time}$ — average time overhead induced by transformation, $\%_{nodes}$ and $\%_{edges}$ — average difference in the proof size as a result of transformation.

Another interesting result to report is the fact that only 45% of the proofs contained AB-mixed predicates and, consequently, required transformation. This is another motivation for using off-the-shelf algorithms for SMT-Solvers and have the proof transformed in a second stage, rather then tweaking (and potentially slowing down) the solver to generate clean proofs upfront.

6. CONCLUSION AND FUTURE WORK

We have presented a technique to transform a propositional proof that isolates and removes AB-mixed predicates, in such a way that known procedures to compute interpolants can be applied. The approach enables the use of off-the-shelf techniques for SMT-Solvers that are likely to introduce AB-mixed predicates, such as Ackermann's Expansion, Lemma on Demand, Splitting on Demand, and DTC. We have shown by means of experiments that our rules can effectively transform the proof without generating any blow-up.

As future work we would like to implement interpolation techniques on top of our framework in order to experiment with different strategies of proof transformation, and compare with other approaches. Also we plan to explore the possibility of using our rules for the mere purpose of proof reduction.

7. REFERENCES

[1] W. Ackermann. *Solvable Cases of the Decision Problem.* Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 1954.

[2] O. Bar-Ilan, O. Fuhrmann, S. Hoory, O. Shacham, and O. Strichman. Linear-Time Reductions of Resolution Proofs. In *HVC*, pages 114–128, 2008.

[3] C. Barrett, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Splitting on Demand in SAT Modulo Theories. In *LPAR*, pages 512–526, 2006.

[4] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. *Handbook on Satisfiability*, volume 185, chapter Satisfiability Modulo Theories. IO Press, 2009.

[5] M. Bofill, R. Nieuwenhuis, A. Oliveras, E. Rodrguez-Carbonell, and A. Rubio. A Write-Based Solver for SAT Modulo the Theory of Arrays. In *FMCAD*, pages 101–108, 2008.

[6] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. Van Rossum, S. Ranise, and R. Sebastiani. Efficient Satisfiability Modulo Theories via Delayed Theory Combination. In *CAV'05*, pages 335–349, 2005.

[7] R. Brummayer and A. Biere. Lemmas on Demand for the Extensional Theory of Arrays. *JSAT*, 2009.

[8] R. Bruttomesso, E. Pek, N. Sharygina, and A. Tsitovich. The OpenSMT Solver. In *TACAS*, 2010.

[9] A. Cimatti, A. Griggio, and R. Sebastiani. Efficient Interpolant Generation in Satisfiability Modulo Theories. In *TACAS*, pages 397–412, 2008.

[10] W. Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *J. Symb. Log.*, pages 269–285, 1957.

[11] L. de Moura and N. Bjørner. Generalized, Efficient Array Decision Procedures. In *FMCAD*, 2009.

[12] V. D'Silva, D. Kroening, M. Purandare, and G. Weissenbacher. Restructuring Resolution Refutations for Interpolation. Technical report, ETH, 2008.

[13] V. DSilva, D. Kroening, M. Purandare, and G. Weissenbacher. Interpolant Strength. In *VMCAI*, pages 129–145, 2010.

[14] A. Goel, S. Krstić, and A. Fuchs. Deciding Array Formulas with Frugal Axiom Instantiation. In *SMT*, 2008.

[15] A. Goel, S. Krstic, and C. Tinelli. Ground Interpolation for Combined Theories. In *CADE*, 2009.

[16] T. Henzinger and K. L. McMillan R. Jhala, R. Majumdar. Abstractions from Proofs. In *POPL*, 2004.

[17] R. Jhala and K.L. McMillan. Interpolant-Based Transition Relation Approximation. In *CAV*, pages 39–51, 2005.

[18] K. L. McMillan. Interpolation and SAT-Based Model Checking. In *CAV*, pages 1–13, 2003.

[19] K. L. McMillan. An Interpolating Theorem Prover. In *TACAS*, pages 16–30, 2004.

[20] K. L. McMillan. Applications of Craig Interpolation to Model Checking. In *CSL*, pages 22–23, 2004.

[21] L. De Moura and H. Rue. Lemmas on Demand for Satisfiability Solvers. In *SAT*, 2002.

[22] G. Nelson and D. C. Oppen. Simplification by Cooperating Decision Procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–57, 1979.

[23] P. Pudlák. Lower Bounds for Resolution and Cutting Plane Proofs and Monotone Computations. *J. Symb. Log.*, 62(3):981–998, 1997.

[24] G. Yorsh and M. Musuvathi. A Combination Method for Generating Interpolants. In *CADE*, 2005.