Restructuring Resolution Refutations for Interpolation

Vijay D'Silva[1,2], Daniel Kroening[1,2], Mitra Purandare[2], and Georg Weissenbacher[1,2*]

[1] Computing Laboratory, Oxford University
[2] Computer Systems Institute, ETH Zurich

Abstract. Interpolants are the cornerstone of several approximate verification techniques. Current interpolation techniques restrict the search heuristics of the underlying decision procedure to compute interpolants, incurring a negative impact on performance, and apply primarily to the lazy proof explication framework. We bridge the gap between fast decision procedures that aggressively use propositional reasoning and slower interpolating decision procedures by extending the scope of the latter to non-lazy approaches and relaxing the restrictions on search heuristics. Both are achieved by combining a simple set of transformations on resolution refutations. Our experiments show that this method leads to speedups when computing interpolants and to reductions in proof size.

1 Introduction

The bottle-neck in model checking is usually steps that involve quantifier elimination such as computing an image, constructing an abstract transition relation, and fixed point detection. For a suitably constructed formula, a Craig interpolant provides a quantifier-free approximation, with respect to logical implication, of a quantified formula. This enables a family of approximate verification techniques that obviate the need for quantifier elimination [1–3].

Program verification using interpolants requires decision procedures for first order theories that can compute interpolants for unsatisfiable conjunctions of theory literals. Such an *interpolating decision procedure* must deal with a logic used in program verification and incur a low overhead as compared to a state-of-the-art decision procedure. These concerns can be at odds with existing techniques for computing interpolants. Nearly all available techniques compute interpolants in the lazy explication paradigm [4] and are not applicable if some constraints from the theory are propositionally encoded. An extreme case of the latter is implemented in nearly all competitive solvers for bit-vector arithmetic.

Further, interpolants are typically constructed from proofs of unsatisfiability and require proofs to have a certain structure. Interpolating decision procedures often restrict the search techniques used in order to ensure that the proof of

* Supported by Microsoft Research's European PhD Scholarship Programme.

unsatisfiability which is obtain has this structure. Such restrictions may unfortunately have an adverse effect on the performance of the decision procedure. For example, a restriction is imposed in [5] to construct interpolants in combinations of theories. The authors themselves point out that "*the restrictions on the branching and learning heuristics needed to generate [interface equality] local proofs might have a negative impact in performance.*" We present techniques that *a posteriori* transform a resolution refutation produced by a decision procedure (not necessarily interpolating) to obtain a refutation satisfying such restrictions.

A consequence of the methods presented in this article is that one can use a fast decision procedure that does not compute interpolants to decide if a formula is satisfiable and then use a slower interpolating decision procedure to construct interpolants. The only restrictions we make are that the fast decision procedure should generate a resolution refutation for unsatisfiable formulae, and that the propositional formula for which the refutation is generated has a certain structure (described in § 3.1). Our approach has two advantages. First, one can decide satisfiability, usually the most computationally expensive step in the verification methods we consider, using the most efficient methods available as long as they satisfy the two constraints above. Second, the resolution proof only includes the parts of the initial formula required to show unsatisfiability, so the interpolating decision procedure has to work with a smaller formula.

The typical restrictions on proof structure imposed in the literature are tantamount to imposing a *partitioned partial order* on the resolution proof. That is, the variables occurring in the proof are partitioned and a restriction defines a partial order over resolution steps on variables depending on the block of the partition in which they occur. Restrictions that can be described in this manner occur in methods for constructing interpolants in theory combinations [5], and interpolant strengthening [2].

We present an algorithm that given a resolution proof and a partitioned partial order, produces a proof satisfying this order. This approach is independent of the decision procedure that produces the resolution proof and has a low overhead in practice. The contributions of this paper build upon this algorithm. We highlight that our algorithms are devised following an extensive survey of *proofs* concerning resolution systems in the automated deduction and proof complexity literature. Proofs by induction on the structure of a resolution refutation, in particular in [6, 7], are a fertile source of ideas for tranformations on refutations.

Contributions. We make two contributions. First, we extend the applicability of current interpolation methods beyond the lazy proof explication setting. Given an interpolating decision procedure for conjunctions of literals in a ground theory, we obtain an interpolation method for any decision procedure that combines lazy explication with propositionally encoded constraints. A direct consequence of our method is an interpolating decision procedure for a fragment of bit-vector logic. No such decision procedure exists in the literature. Second, we present several tools for transforming resolution proofs. These have applications not explored in the paper, such as interpolant strengthening [2] and proof compression [8, 9].

2 Resolution Proofs and Transformations

We introduce the notation, basic terminology and background on resolution in § 2.1, followed by our transformations on resolution proofs in § 2.2, which lead to our algorithm for reordering proofs in § 2.3.

2.1 Resolution Proofs

Let A be a set of atomic propositions (atoms), $\mathcal{L}_A = \{a, \overline{a} | a \in A\}$ be the corresponding set of literals, where $\overline{l}$ or equivalently $\neg l$ denotes the negation of a literal l. A *clause* C is a set of literals and $\wp(\mathcal{L}_A)$, where $\wp(S)$ denotes the powerset of the S, is the set of clauses over $\mathcal{L}_A$. The empty clause is denoted $\Box$ and contains no literals. The disjunction of two clauses C and D is their union, denoted $C \vee D$, which is further simplified to $C \vee l$ if D is a singleton $\{l\}$. A clause C *subsumes* D if $C \subseteq D$. Observe that the empty clause subsumes every clause. A propositional formula is defined as usual with $\bot$ denoting false, and $\top$ denoting true. A formula F is in Conjunctive Normal Form (CNF) if it is a conjunction of clauses.

The *resolution principle* states that any assignment that satisfies the clauses $C \vee x$ and $D \vee \overline{x}$ also satisfies $C \vee D$. It is formally described by the inference rule:

$$\frac{C \vee x \qquad D \vee \overline{x}}{C \vee D} \quad [\mathsf{Res}]$$

The clauses $C \vee x$ and $D \vee \overline{x}$ are the *antecedents*, the atomic proposition x is the *pivot*, and $C \vee D$ is the *resolvent*. Let $\mathrm{Res}(C, D, x)$ denote the resolvent of C and D with pivot x. A clause C is *derived from F by resolution*, where F is a formula in CNF, if it is the resolvent of two clauses that either occur in F or have been derived from F by resolution.

Definition 1. *A* resolution proof *is a labelled DAG, $R = (V_R, E_R, \ell_R, \mathsf{s}_R)$, where V_R is a set of vertices, E_R is a set of edges, ℓ_R is a labelling function that maps each vertex to a clause and $\mathsf{s}_R \in V_R$ is the sink vertex. An* initial vertex *has in-degree 0 and all other vertices are* internal *and have in-degree 2. The sink has out-degree 0 and the clause labelling it is a* conclusion. *If v is an internal vertex and $(v_1, v), (v_2, v) \in E_R$, then $\ell_R(v)$ is the resolvent of $\ell_R(v_1)$ and $\ell_R(v_2)$.*

The subscripts above are dropped if clear. A resolution proof R is a proof of a clause C from a CNF formula F if $\ell_R(\mathsf{s}_R) = C$ and for each initial vertex $v \in V_R$, $\ell_R(v)$ occurs in F. A vertex v_1 in R is a *parent* of v_2 if $(v_1, v_2) \in E_R$. If a vertex v has parents v_1 and v_2, let $\mathit{piv}(v)$ be the pivot for inferring $\ell(v)$ from $\ell(v_1)$ and $\ell(v_2)$ by resolution. Further, let v^+ denote the parent of v labelled with the antecedent of $\ell_R(v)$ containing $\mathit{piv}(v)$ and let v^- be the parent labelled with the antecedent containing $\neg\mathit{piv}(v)$.

A vertex v_1 is an *ancestor* of v_2 if there is a path from v_1 to v_2. A set W of vertices in a proof is *ancestor-free* if no vertex in W has an ancestor in W.

Transformation	Notation	Effect
Resolution	$\text{ProofRes}(P_1, P_2, x)$	Resolve the conclusion of P_1 and P_2 on x.
Substitution	$P[v_1 \mapsto P_1, \ldots, v_k \mapsto P_k]$	Set $\pi(v_i)$ to P_i and propagate changes.
Restriction	$P[l \leftarrow \bot]$	Set l to $\bot$ in P propagate changes.
Projection	$P\lfloor l$	Eliminate resolutions on l in P

Table 1. Transformations on Resolution Proofs

We note in passing, that the ancestor relation defines a partial order over the vertices in a proof and that an ancestor-free set is an anti-chain with respect to this partial order. The subgraph of proof that includes all ancestors of a vertex v is denoted $\pi(v)$. Observe that $\pi(v)$ is a proof.

A *resolution refutation* is a resolution proof in which the sink is labelled with the empty clause. The words proof and refutation, if not further qualified connote resolution proofs and resolution refutations. A *refinement* of resolution is a restriction on the structure of the resolution proof. A *tree resolution* proof is a tree. A *regular resolution* proof satisfies that on every path from an initial vertex to the conclusion, each pivot occurs at most once [10].

2.2 Refactoring Resolution Proofs

A *proof transformation* is an algorithm that maps a resolution proof to another. We introduce four simple proof transformations, summarised in Table 1. Restricted forms of these transformations exist in the literature. Our contribution is to generalise them and use them to derive algorithms for restructuring proofs. Restructured proofs are used to construct interpolants in quantifier-free theories.

The first transformation, denoted $\text{ProofRes}(P_1, P_2, x)$, applies to two proofs P_1 and P_2 and produces a new proof P. The conclusion of P is the resolvent of the conclusions of P_1 and P_2, if $x \in \ell_{P_1}(\mathtt{s}_{P_1})$ and $\overline{x} \in \ell_{P_2}(\mathtt{s}_{P_2})$. If $x \notin \ell_{P_1}(\mathtt{s}_{P_1})$, then P is just P_1, and if $x \in \ell_{P_1}(\mathtt{s}_{P_1})$ but $\overline{x} \notin \ell_{P_2}(\mathtt{s}_{P_2})$, then P is P_2. The intuition is that the proofs P_1 and P_2 are to be extended by resolving on x, assuming that the conclusion of P_1 contains x and that of P_2 contains $\overline{x}$. If the conclusion of P_1 does not contain x, then one can assume that x has already been eliminated and use P_1. The other case is symmetric. A formal definition follows.

Definition 2. *Let $P_1 = (V_1, E_1, \ell_1, \mathtt{s}_1)$ and $P_2 = (V_2, E_2, \ell_2, \mathtt{s}_2)$ be resolution proofs. The* proof resolution *transformation* $\text{ProofRes}(P_1, P_2, x)$ *results in a proof P, where P is P_1 if $x \notin \ell_1(\mathtt{s}_1)$, and P is P_2 if $\overline{x} \notin \ell_2(\mathtt{s}_2)$, otherwise $P = (V, E, \ell, \mathtt{s})$ where $\mathtt{s}$ is a new vertex not in V_1 or V_2, $V = V_1 \cup V_2 \cup \{\mathtt{s}\}$, $E = E_1 \cup E_2 \cup \{(\mathtt{s}_1, \mathtt{s}), (\mathtt{s}_2, \mathtt{s})\}$, and for $v \in V$, $\ell(v)$ is $\ell_1(v)$ if $v \in V_1$, and is $\ell_2(v)$ if $v \in V_2$, and is* $\text{Res}(\ell_1(\mathtt{s}_1), \ell_2(\mathtt{s}_2), x)$ *if $v = \mathtt{s}$.*

A similar transformation appears in the function Resolve in Figure 6 in [11]. However, the result of $\text{Resolve}(P_1, P_2, x)$ is defined to be P_2 if the conclusion of P_1 does not contain x and is P_1 if the conclusion of P_2 does not contain $\overline{x}$. This

```
REFACTOR($v, R, \rho$)
Input: Vertex $v$, Proof $R = (V_R, E_R, \ell_R, \mathtt{s})$,
           Substitution mapping $\rho = \{v_1 \mapsto P_1, \ldots, v_k \mapsto P_k\}$
 1: if $visited(v)$ then return
 2: end if
 3: $visited(v) \leftarrow$ True
 4: if $v \in dom(\rho)$ then $\pi(v) \leftarrow \rho(v)$; return
 5: else if $v$ has no parents then return
 6: else
 7:     REFACTOR($v^+, R, \rho$)
 8:     REFACTOR($v^-, R, \rho$)
 9:     $\pi(v) \leftarrow$ ProofRes($\pi(v^+), \pi(v^-), piv(v)$)
10:     return
11: end if
```

Fig. 1. Replace vertices in a proof and propagate changes

order does affect the correctness of the algorithms we propose, so we cannot use RESOLVE in place of ProofRes.

The next three transformations in Table 1 change the subproof at a vertex (or set of vertices) and propagate the change along the proof graph. We first define a *replacement* operation that replaces the proof at a vertex v in a proof P_1 with a proof P_2. This operation is not a proof transformation because the result may not be a proof.

Definition 3. *Let $P_1 = (V_1, E_1, \ell_1, \mathtt{s}_1)$ and $P_2 = (V_2, E_2, \ell_2, \mathtt{s}_2)$ be resolution proofs. The* replacement*, of the proof at a vertex v in P_1 with P_2, denoted $\pi(v) \leftarrow P_2$, is the DAG $G = (V, E, \ell, \mathtt{s}_1)$, where $V = V_1 \cup V_2$, $E = (E_1 \setminus \{(v^+, v), (v^-, v)\}) \cup (E_2 \setminus \{(\mathtt{s}_2^+, \mathtt{s}_2), (\mathtt{s}_2^-, \mathtt{s}_2)\} \cup \{(\mathtt{s}_2^+, v), (\mathtt{s}_2^-, v)\}$, and the labelling function ℓ is defined such that $\ell(v) = \ell_2(\mathtt{s}_2)$, and for all $v' \neq v$, $\ell(v')$ is $\ell_1(v')$ if $v' \in V_1$ and is $\ell_2(v')$ otherwise.*

The DAG G in Definition 3 is not a resolution proof because the the successors of v may no longer be labelled with resolvents of their parent vertices. To make G a resolution proof, we need to recursively relabel the descendants of v. Replacement and relabelling are combined in the algorithm REFACTOR in Figure 1. Given a mapping ρ of vertices to resolution proofs, REFACTOR replaces $\pi(v)$ with $\rho(v)$ if v is in the domain of ρ. The proofs at vertices not in the domain of ρ are replaced with ProofRes$(\pi(v^+), \pi(v^-), piv(v))$. This second step ensures that each vertex is labelled with the resolvent of its parents. The algorithm as presented admits a simple optimisation. The tranformation ProofRes$(\pi(v^+), \pi(v^-), piv(v))$ introduces a new vertex and replacing $\pi(v)$ with this proof Disconnects the newly introduced vertex from the proof. Thus, one can directly replace $\pi(v)$ with the resolvent of $\pi(v^+)$ and $\pi(v^-)$.

The substitution transformation on proofs is defined using REFACTOR. Let $dom(f)$ denote the domain of a mapping f. A mapping $\rho = \{v_1 \mapsto P_1, \ldots, v_k \mapsto$

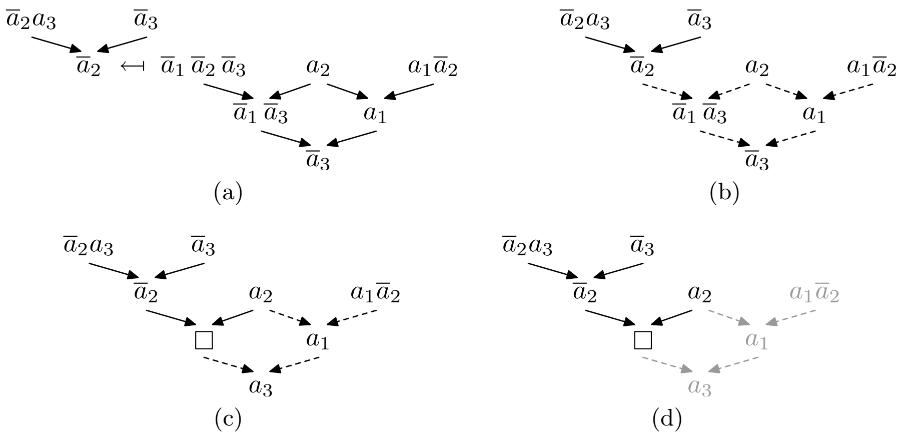

Fig. 2. Refactoring proofs.

$P_k\}$ of vertices to proofs is *ancestor-free* if the sets $\{v_1, \ldots, v_k\}$ and $\{\mathtt{s}_{P_1}, \ldots, \mathtt{s}_{P_k}\}$ are ancestor free. Substitution is defined for ancestor-free mappings.

Definition 4. *Let $R = (V_R, E_R, \ell_R, \mathtt{s}_R)$ be a resolution proof and $\rho = \{v_1 \mapsto P_1, \ldots, v_k \mapsto P_k\}$ be an ancestor free mapping. The* substitution $R[v_1 \mapsto P_1, \ldots, v_k \mapsto P_k]$*, or equivalently $R[\rho]$, is the proof returned by* $\textsc{Refactor}(\mathtt{s}_R, R, \rho)$.

A restricted form of substitution has been used in the proof complexity literature to construct regular, tree resolution proofs from tree resolution proofs. For an example, see [12, Lemma 5.1]. A similar method has more recently appeared in the RepairProof algorithm in [8]. The RepairProof algorithm is used to reduce proof size by reusing singleton clauses and by making tree-shaped proofs regular. In contrast, in our transformation, the proof substituting a vertex can be *arbitrary* as illustrated below. In the examples in this paper, if each vertex in a proof is labelled with a unique clause, and a vertex v is labelled with the clause C, we directly refer to "the vertex C" to avoid having to name vertices. Further, a clause $\{a_1, \overline{a}_2\}$ is written as $a_1\overline{a}_2$ in figures to avoid clutter. Lemma 1, which follows Example 1 states that the result of substitution is a resolution proof.

Example 1. Consider the resolution proof R in Figure 2(a). Let v be the vertex labelled with the clause $\overline{a}_1\overline{a}_2\overline{a}_3$ in Figure 2(a) and P be the proof shown alongside with the conclusion $\overline{a}_2$. We demonstrate how $\textsc{Refactor}$ is used to obtain the substitution $R[v \mapsto P]$. First, the proof $\pi(v)$ is replaced with P in 2(b) by line 4 in $\textsc{Refactor}$. The dashed arrows in Figure 2(b) indicate the parts of the proof along which the change has to be propagated. Next, as per line 9 in $\textsc{Refactor}$, the child of v is labelled with the resolvent of $\overline{a}_2$ and a_2 in 2(c). This step results in the empty clause which is propagated to the sink to obtain the proof in 2(d). The faded subgraph is no longer included in the transformed proof. $\triangleleft$

Lemma 1. *If $R = (V, E, \ell, \mathtt{s})$ is a resolution proof and ρ is an ancestor-free mapping of vertices to resolution proofs, then $R[\rho]$ is a resolution proof.*

Proof: By induction on the number of ancestors of $\mathtt{s}$. If $\mathtt{s}$ has no ancestors, then $\pi(v) \leftarrow \rho(v)$ results in a proof because $\rho(v)$ is a proof. Assume that Refactor returns a proof if $\mathtt{s}$ has at most n ancestors. Consider the case where $\mathtt{s}$ has $n+1$ ancestors. If $\mathtt{s} \in dom(\rho)$, we are done because $\rho(\mathtt{s})$ is a proof and ρ is ancestor-free. Otherwise, each parent of $\mathtt{s}$ has at most n ancestors, so by the induction hypothesis Refactor(v^+, R, ρ) and Refactor(v^-, R, ρ) are valid proofs. It remains to show that line 9 in Refactor results in a valid proof. This holds because ProofRes is a proof transformation and after replacement, $\mathtt{s}$ is labelled with the resolvent of its parents. ■

The restriction that ρ should be ancestor free is much weaker than requiring that the proofs in the range of ρ be disjoint. This allows the proofs to share subproofs, which is essential for efficiency. The following lemma is shows how one can use substitution to strengthen the conclusion of a proof. It is useful for proving the correctness of the two remaining transformations.

Lemma 2. *Let $R_1 = (V_1, E_1, \ell_1, \mathtt{s}_1)$ and $R[\rho] = (V_2, E_2, \ell_2, \mathtt{s}_2)$ be resolution proofs, where ρ is an ancestor-free mapping, and C_v be the conclusion of the proof $\rho(v)$ for each $v \in dom(\rho)$. If $C_v \subseteq \ell(v)$ for each $v \in dom(\rho)$, then $\ell_2(\mathtt{s}_2) \subseteq \ell_1(\mathtt{s}_1)$.*

Proof: By induction on the number of ancestors of $\mathtt{s}_1$. If $\mathtt{s}_1$ has no ancestors, then $C_{\mathtt{s}_1} \subseteq \ell_1(\mathtt{s}_1)$ by the condition of the lemma. Assume that the lemma holds if $\mathtt{s}_1$ has at most n ancestors. If $\mathtt{s}_1$ has $n+1$ ancestors, $\mathtt{s}_1^+$ and $\mathtt{s}_1^-$ have at most n ancestors each and by the induction hypothesis, the lemma holds for $\pi(\mathtt{s}_1^+)[\rho]$ and $\pi(\mathtt{s}_1^-)[\rho]$. Let $\mathtt{s}_2^+$ be the sink of $\pi(\mathtt{s}_1^+)[\rho]$ labelled with C^+ and $\mathtt{s}_2^-$ be the sink of $\pi(\mathtt{s}_1^-)[\rho]$ labelled with C^-. We need to show that the lemma holds for ProofRes$(\pi(\mathtt{s}_2^+), \pi(\mathtt{s}_2^-), piv(\mathtt{s}_1))$, which is equivalent to $R[\rho]$. If $piv(\mathtt{s}_1) \notin C^+$ then $\ell_2(\mathtt{s}_2) = C^+$ by the definition of ProofRes, and $\ell_2(\mathtt{s}_2) \subseteq \ell_1(\mathtt{s}_1)$ by the induction hypothesis and resolution. If $piv(\mathtt{s}_1) \in C^+$ but $\neg piv(\mathtt{s}_1) \notin C^-$ then $\ell_2(\mathtt{s}_2) = C^-$ by the definition of ProofRes, and $C^- \subseteq \ell_1(\mathtt{s}_1)$ by the induction hypothesis and resolution. Finally, if $piv(\mathtt{s}_1) \in C^+$ and $\neg piv(\mathtt{s}_1) \in C^-$, then $\text{Res}(C^+, C^-, piv(\mathtt{s}_1)) \subseteq \text{Res}(\ell_1(\mathtt{s}_1^+), \ell_1(\mathtt{s}_1^-), piv(\mathtt{s}_1))$, by the induction hypothesis and resolution. In all cases, we have that $\ell_2(\mathtt{s}_2) \subseteq \ell_1(\mathtt{s}_1)$. ■

A particular case of subsumption as above arises from strengthening the conclusion of a proof by assigning truth values to atoms. For simplicity, we identify a proof with a single vertex with the clause labelling that vertex. That is, for a proof $P = (\{\mathtt{s}_P\}, \emptyset, \ell_P, \mathtt{s}_P)$ and a clause C such that $\ell_P(\mathtt{s}_P) = C$, we say $P = C$ and denote the replacement $\pi(v) \leftarrow P$ by $\pi(v) \leftarrow C$. The restriction of a proof is defined next.

Definition 5. *Let $R = (V_R, E_R, \ell_R, \mathtt{s}_R)$ be a resolution proof and l be a literal occurring in R. Define the mapping $\rho_l = \{v \mapsto P | v \text{ is an initial vertex}, l \in \ell(v), \text{ and } P = \ell(v) \setminus \{l\}\}$. The* restriction *of R with l set to $\bot$ is defined by $R[\rho_l]$.*

The restriction of R with l set to $\bot$ is, by abuse of notation, denoted $R[l \leftarrow \bot]$. The restriction with l set to $\top$ is $R[\bar{l} \leftarrow \bot]$. Restriction A statement about the correctness of restriction follows.

Lemma 3. *Let $R_1 = (V_1, E_1, \ell_1, \mathtt{s}_1)$ be a resolution proof, l be a literal occurring in R_1 and $R_2 = (V_2, E_2, \ell_2, \mathtt{s}_2)$ be $R_1[l \leftarrow \bot]$. The literal l does not occur in any clause in R_2, and $\ell_2(\mathtt{s}_2) \subseteq \ell_1(\mathtt{s}_1)$.*

Proof: We first prove by induction on the number of ancestors of $\mathtt{s}_1$ that l does not occur in R_2. If $\mathtt{s}_1$ has no ancestors, then $\ell_2(\mathtt{s}_2) = \ell_1(\mathtt{s}_1) \setminus \{l\}$, so l does not occur in R_2. Assume that l does not occur in R_2 if $\mathtt{s}_1$ has at most n ancestors. By the definition of restriction, $R_2 = R_1[l \leftarrow \bot] = R_1[\rho_l]$ for ρ_l as in Definition 5. From the definition of substitution, we have that $R_2 = \text{ProofRes}(\pi(\mathtt{s}_1^+)[\rho_l], \pi(\mathtt{s}_1^-)[\rho_l], piv(\mathtt{s}_1))$. The induction hypothesis applies to $\pi(\mathtt{s}_1^+)[\rho_l]$ and $\pi(\mathtt{s}_1^-)[\rho_l]$, so l does not occur in these two proofs. We conclude that l does not occur in R_2 because ProofRes does not introduce literals.

For the second part, observe that for each $v \in dom(\rho_l)$, where ρ_l is as before, $\rho_l(v)$ is a single vertex labelled with $\ell_1(v) \setminus \{l\}$. It follows from Lemma 2 that $\ell_2(\mathtt{s}_2) \subseteq \ell_1(\mathtt{s}_1)$. ■

The last transformation we introduce, which is essential for changing the order of resolution steps in a proof is *projection*. Restriction eliminates a certain literal from a proof. In contrast, we use projection to eliminate resolution steps on a specified literal, which ensures that the literal is present in the conclusion of the proof. Projection is used to prove lower bounds on the size of resolution proofs in [7, Lemma 3.2].

Consider the proof $R[l \leftarrow \bot]$. It is obtained from R by removing l from the labels of some initial vertices in R and propagating this change along the structure of the proof. By Lemma 3, the literal l does not occur in the proof $R[l \leftarrow \bot]$. If we now add the literal l back to the clauses from which it was removed, l is guaranteed to be present in the conclusion of the proof because it is never resolved away. This transformation is defined below.

Definition 6. *Let $R_1 = (V_1, E_1, \ell_1, \mathtt{s}_1)$ be a resolution proof, and $R_2 = R_1[l \leftarrow \bot] = R_1[\rho_1] = (V_2, E_2, \ell_2, \mathtt{s}_2)$, be the restriction of R_1 with l set to $\bot$, obtained by substitution with the mapping ρ_l. Define the mapping $\gamma = \{v \mapsto P | v \in dom(\rho_l) \cap V_2$ and $P = \ell_2(v) \cup \{l\}\}$. The* projection *of l from R_1, denoted $R_1 \lfloor l$ is the proof $(R_1[l \leftarrow \bot])[\gamma]$.*

The definition above is not strictly formal because the initial vertices in $dom(\rho_l)$ may not be the initial vertices in R_2 after refactoring. A purely formal definition would require defining a total, surjective function mapping vertices from R_1 to R_2 and defining the domain of ρ_2 to be those vertices in R_2 which are the image of an initial vertex in R_1. We hope this definition suffices to define projection. Lemma 4 states a basic property of projection.

Lemma 4. *Let $R_1 = (V_1, E_1, \ell_1, \mathtt{s}_1)$ be a resolution proof, $l \in \ell_1(v)$ for some initial vertex $v \in V_1$ and $R_1 \lfloor l = (V_2, E_2, \ell_2, \mathtt{s}_2)$. It holds that $l \in \ell_2(\mathtt{s}_2)$ and that $\ell_2(\mathtt{s}_2) \subseteq \ell_1(\mathtt{s}_1) \cup \{l\}$.*

Proof: First, we show by induction on the number of ancestors of $\mathtt{s}_1$ that $l \in \ell_2(\mathtt{s}_2)$. If $\mathtt{s}_1$ has no ancestors, l is not in the clause labelling the sink of $R_1[l \leftarrow \bot]$ and by definition l is contained in $R_1\lfloor l$. Suppose this holds if the sink of a proof has at most n ancestors. Consider the case of $n+1$ ancestors. By definition $R_1\lfloor l$ is the proof $(R_1[l \leftarrow \bot])[\gamma]$, where γ is a mapping as in Definition 6. $R_1[l \leftarrow \bot]$ is in turn a proof of the form $\text{ProofRes}(\pi(\mathtt{s}_1^+), \pi(\mathtt{s}_1^-), \mathit{piv}(\mathtt{s}_1))[l \leftarrow \bot]$. By the definition of restriction, this proof is equivalent to $\text{ProofRes}(\pi(\mathtt{s}_1^+)[l \leftarrow \bot], \pi(\mathtt{s}_1^-)[l \leftarrow \bot], \mathit{piv}(\mathtt{s}_1))$. If $\mathit{piv}(\mathtt{s}_1)$ is the atom in l, this proof is equivalent to $\pi(\mathtt{s}_1^+)[l \leftarrow \bot]$, because by Lemma 3, l does not occur in $\pi(\mathtt{s}_1^+)[l \leftarrow \bot]$. If $\mathit{piv}(\mathtt{s}_1)$ is not the atom in l, $R_1[l \leftarrow \bot]$ is as above.

Thus, $(R_1[l \leftarrow \bot])[\gamma]$ is either (1) $(\pi(\mathtt{s}_1^+)[l \leftarrow \bot])[\gamma]$ or (2) $\text{ProofRes}((\pi(\mathtt{s}_1^+)[l \leftarrow \bot])[\gamma], (\pi(\mathtt{s}_1^-)[l \leftarrow \bot])[\gamma], \mathit{piv}(\mathtt{s}_1))$, where $\mathit{piv}(\mathtt{s}_1)$ is not the atom in l. In the first case, $\mathtt{s}_1^+$ has at most n ancestors, so the induction hypothesis applies. In the second case, at least one of $\pi(\mathtt{s}_1^+)$ or $\pi(\mathtt{s}_1^-)$ has an initial vertex labelled with a clause containing l. Without loss of generality, let $\pi(\mathtt{s}_1^+)$ be this proof. The induction hypothesis now applies, so the conclusion of $P_1 = (\pi(\mathtt{s}_1^+)[l \leftarrow \bot])[\gamma]$ contains l. Finally, because $\mathit{piv}(\mathtt{s}_1)$ is not the atom in l, the conclusion of $\text{ProofRes}(P_1, (\pi(\mathtt{s}_1^-)[l \leftarrow \bot])[\gamma], \mathit{piv}(\mathtt{s}_1))$ contains l. It follows that $l \in \ell_2(\mathtt{s}_2)$.

For the second part, we have from Lemma 3 that the conclusion of $R_1[l \leftarrow \bot]$, say C, subsumes $\ell_1(\mathtt{s}_1)$. From the first part, we have that the conclusion of $R_1\lfloor l$, $\ell_2(\mathtt{s}_2) = C \cup \{l\}$. It follows that $\ell_2(\mathtt{s}_2) \subseteq \ell_1(\mathtt{s}_1) \cup \{l\}$. ■

2.3 Reordering Resolution Proofs

We use transformation to design an algorithm for reordering resolution proofs. Though reordered proofs have several applications, we restrict our attention to the construction of theory-specific interpolants. A proof is reordered with respect to a partition of the pivot variables in the proof, and a partial oder over these partitions. We define this notion next.

Definition 7. *A* partially ordered partition *of a set S is a tuple $(\beta_S, \sqsubseteq_S)$, where β_S is a partition of S and $\sqsubseteq_S$ is a partial order over the blocks of β_S.*

Let $\beta_S(v)$ denote the block of $v \in S$ in β_S. A sequence $v_0, \ldots, v_k$ from S *respects* $(\beta_S, \sqsubseteq_S)$ if for any v_i, v_j in the sequence with $0 \leq i \leq j \leq k$, if $\beta_S(v_i)$ and $\beta_S(v_j)$ are comparable, then $\beta_S(v_i) \sqsubseteq_S \beta_S(v_j)$. Let A be the set of atoms appearing in a resolution proof R and $(\beta_A, \sqsubseteq_A)$ be a partially ordered partition of A. The proof R *respects* $(\beta_A, \sqsubseteq_A)$ if on every path $v_0, \ldots, v_k$ from the sink to an initial vertex, the sequence of pivots $\mathit{piv}(v_0), \ldots, \mathit{piv}(v_k)$ respects $(\beta_A, \sqsubseteq_A)$. To reduce notational burden, we write $\beta_A(v_0)$ for $\beta_A(\mathit{piv}(v_0))$ and say $v_0, \ldots, v_k$ respects $(\beta_A, \sqsubseteq_A)$ if $\mathit{piv}(v_0), \ldots, \mathit{piv}(v_k)$ respects $(\beta_A, \sqsubseteq_A)$. The subscripts are dropped if clear.

The idea of our reordering algorithm is as follows. Let $p = v_0, \ldots, v_k$ be a path from the sink to a vertex in the proof that respects $(\beta, \sqsubseteq)$. If v is the next vertex on this path and $v_k \not\sqsubseteq v$, we identify a vertex v_i such that the sequences $v_0, \ldots, v_i, v$ and $v, v_{i+1}, \ldots, v_k$ respect $(\beta, \sqsubseteq)$. The resolutions on $\mathit{piv}(v)$ and

$\neg piv(v)$ in $\pi(v_{i+1})$ are projected out and resolved after the vertex v_{i+1}. Observe that the proof is now closer to being partially ordered.

Some vertices in $\pi(v_{i+1})$ may have decendants that do not lie on a path to $\pi(v_{i+1})$. Such vertices have to be copied because the literals that are projected out may be required in other parts of the proof. We reduce the number of copies required using dominators (dominators are defined shortly). Despite copying, the new proof at v_{i+1} may be smaller because $piv(v)$ occurs only once as a pivot variable. The transformations above (projection, resolution and copying on-demand) are repeatedly applied until every path respects $(\beta, \sqsubseteq)$. The Reorder algorithm appears in Figure 3. We begin the description of this algorithm by defining dominators.

Definition 8. *A vertex v in a proof R is* dominated *by v' if v' occurs on every path from v to s_R.*

For example, in Figure 2(a), the vertex labelled with a_2 is dominated by $\overline{a}_3$ but not by a_1. If v_1 is dominated by v_2, replacing $\pi(v_2)$ by $\text{Res}(\pi(v_2){\downharpoonright}l, \pi(v_2){\downharpoonright}\overline{l}, l)$, where $l = piv(v_1)$, will not weaken the conclusion of the proof (this can be proved using Lemma 4). If v_1 is not dominated by v_2, we need to propagate the change to the label of v_1 to other parts of the proof, which may affect the order of pivot variables. Instead, we copy nodes on a path from v_1 to v_2 that are not dominated by v_2. The method DominatedCopy(v_1, v_2) in Figure 3 creates a copy v_3 of a vertex v_1 if v_1 *is not* dominated by v_2 and sets children of v_1 not dominated by v_2 as children of v_3. A different proof transformation using dominators appears in [9].

We now consider the procedure Reorder. The procedure follows a path in the proof DAG until it encounters a pivot that breaks the order. Resolution on this pivot is deferred to a position on the path that respects the order. When multiple positions exist, the choice can be made heuristically. Choosing a vertex close to the sink can result in reduction of proof size, while a higher vertex may require fewer vertices to be copied. We illustrate a run of Reorder next.

Example 2. Consider the proof in Figure 4(a). Define the partitioned partial order $(\beta, \sqsubseteq)$, where β is $\{\{a_1\}, \{a_2\}, \{a_3\}\}$ and $\beta(a_1) \sqsubseteq \beta(a_3) \sqsubseteq \beta(a_2)$. The path labelled with the sequence of clauses $\Box, a_1, a_2, a_1$, does not respect $(\beta, \sqsubseteq)$ because $\beta(a_2) \not\sqsubseteq \beta(a_1)$. We choose the lowest vertex labelled with a_1 and reorder the proof. That is, we consider two proofs:$\pi(\Box){\downharpoonright}a_1$ and $\pi(\Box){\downharpoonright}\overline{a}_1$.

The proof $\pi(\Box){\downharpoonright}a_1$ has only one vertex labelled a_1. The proof $\pi(\Box){\downharpoonright}\overline{a}_1$ is shown in Figure 4(b). The resolvent of these proofs on a_1 produces the proof in Figure 4(c), which respects $(\beta, \sqsubseteq)$. Observe that this proof is regular and is smaller than the original proof. $\lhd$

Our reordering method amounts to sorting the order of pivots in proof and may induce changes in several parts of the proof. A seemingly more local transformation appears in [2]. We briefly discuss the difference between this method and Reorder.

Let $(\beta, \sqsubseteq)$ be a partitioned partial order as before and x and y be two atoms such that $\beta(x) \sqsubseteq \beta(y)$. Consider a pair of vertices with pivots that do not respect

```
Reorder($v, p = v_0, \ldots, v_k$)
Input: vertex $v$, path $v_0, \ldots, v_k$ to $v$ in $R$ with $\mathsf{s}_R = v_0$
 1: if $v$ is a leaf then return
 2: else if $\beta(v_k) \sqsubseteq \beta(v)$ then
 3:     Reorder($v^+, (v_0, \ldots, v_k, v)$)
 4:     Reorder($v^-, (v_0, \ldots, v_k, v)$); return
 5: else
 6:     find $v_i$ such that $v_0, ..v_i, v$ and $v, v_{i+1}, \ldots, v_k$ respect $(\beta, \sqsubseteq)$
 7:     for all $v'$ between $v$ and $v_{i+1}$ do
 8:         DominatedCopy($v', v_{i+1}$)
 9:     end for
10:     Let $P$ be Res($\pi(v_{i+1})\lfloor piv(v), \pi(v_{i+1})\lfloor\neg piv(v), piv(v)$)
11:     Refactor(this proof, $\{v_i \mapsto P\}$)
12:     Reorder($v_i, p = v_0, \ldots, v_i$)
13: end if
```

```
DominatedCopy($v_1, v_2$)
Input: Vertex $v_1$ to copy, vertex $v_2$
 1: if $v_1$ is dominated by $v_2$ then return
 2: else
 3:     Create new vertex $v_3$, add edges $(v_1^+, v_3), (v_1^-, v_3)$
 4:     Set $\ell(v_3)$ to $\ell(v_1)$
 5:     for all edges $(v_1, v_4)$ in $P$ do
 6:         if $v_4$ is not dominated by $v_2$ then
 7:             Delete $(v_1, v_4)$, add $(v_3, v_4)$
 8:         end if
 9:     end for
10: end if
```

Fig. 3. Reorder a proof given a partitioned partial order.

this order. More formally, let v be a vertex with pivot y, and v^+ have pivot x. The proof at $\pi(v^+)$ is of the form $\mathrm{Res}(P_1, P_2, x)$ for two proofs P_1 and P_2, and let P_3 be the proof $\pi(v^-)$. The method of [2] essentially swaps the order of the two resolution steps using the substitution $R[v \mapsto P]$, where P is the proof $\mathrm{Res}(\mathrm{Res}(P_1, P_3, y), \mathrm{Res}(P_2, P_3, y), x)$.

To reorder a proof using this transformation, one can repeatedly swap the pivots in a proof until the proof respects the desired partitioned partial order. Just as in Reorder, one may also have to copy vertices and propagate changes. Copying is discussed in [2]. We provide an example to demonstrate that refactoring is also required after swapping two pivots.

Example 3. Consider the proof in Figure 5(a). Let $(\beta, \sqsubseteq)$ be a partitioned partial order with $\beta(a_1) \sqsubseteq \beta(a_2)$. There is a path in Figure 5(a) which does not respect this order. We can swap the order of these two resolution steps as described above to obtain the proof in Figure 5(b). This proof has a stronger conclusion.

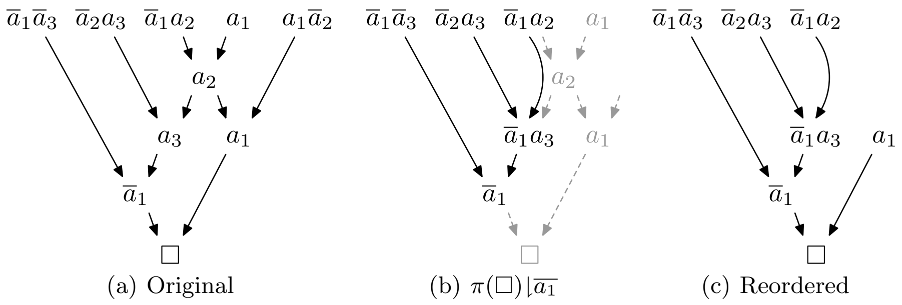
(a) Original (b) $\pi(\Box)\lfloor_{\overline{a_1}}$ (c) Reordered

Fig. 4. Reordering a Proof

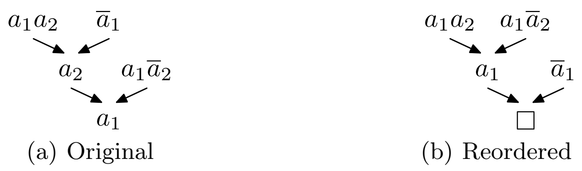
(a) Original (b) Reordered

Fig. 5. Reordering a Proof using the method of [2]

In general one may not have the empty clause, so the change will have to be propagated along the proof. $\lhd$

Thus, whether one uses the method in [2] or the algorithm REORDER, vertices have to be copied and changes have to be propagated along the proof. In the next section, we show how theory-specific interpolants can be constructed using a method for reordering proofs. Any method is applicable.

3 Interpolants by Reordering Refutations

In this section, we study the problem of constructing theory-specific interpolants using resolution proofs generated by a SAT solver. In § 3.1, we introduce the notation and review the basic concepts of the class of decision procedures we consider. The problem of constructing interpolants is examined in § 3.3.

3.1 SAT-Based Decision Procedures

We work in the standard setting of a first order theory. Consider a theory T over the signature Σ_T with variables, function symbols, and predicate symbols, where constants are nullary functions and atomic propositions are nullary predicates. If P is an n-ary predicate and $r_1, \ldots r_n$ are terms, $P(r_1, \ldots, r_n)$ is an *atom*. Literals, clauses and CNF formulae are defined over atoms as in Section 2.1 and formulae are defined as usual. We consider only quantifier-free formulae. We write $\phi \models_T \psi$ to denote that the formula ϕ entails the formula ψ in the theory T. Validity and satisfiability in T are defined as usual. A solver for T, or T-solver, is an algorithm that decides if ϕ is satisfiable. Let $\mathrm{Var}(\phi)$ be the set of variables in ϕ, $\mathrm{Atoms}(\phi)$ be the atoms in ϕ, and $\phi \preceq \psi$ denote that the variables and atomic propositions occurring in ϕ occur in ψ.

The *propositional skeleton* of a formula ϕ is a formula $\mathsf{sk}(\phi)$ obtained by replacing each atom in ϕ by an atomic proposition not in ϕ. Observe that we can define a bijective mapping $\lambda : \mathcal{L}_{\mathrm{Atoms}(\mathsf{sk}(\phi))} \to \mathcal{L}_{\mathrm{Atoms}(\phi)}$ between the literals in $\mathsf{sk}(\phi)$ and ϕ. A satisfying assignment to $\mathsf{sk}(\phi)$ is a conjunction of literals $l_1 \wedge \cdots \wedge l_k$ over atoms in $\mathsf{sk}(\phi)$ that satisfies $\mathsf{sk}(\phi)$.

Contemporary decision procedures construct a formula ψ so that ϕ is satisfiable if and only if $\mathsf{sk}(\phi) \wedge \psi$ is. *Lazy* approaches use a SAT solver to generate truth assignments to $\mathsf{sk}(\phi)$ and iteratively strengthen ψ. If $l_1 \wedge \cdots \wedge l_k$ satisfies $\mathsf{sk}(\phi)$, a dedicated solver is used to check the satisfiability of $\lambda(l_1) \wedge \cdots \wedge \lambda(l_k)$ in T. If this formula is unsatisfiable, $l_1 \wedge \cdots \wedge l_k$ is an assignment that the SAT solver should not consider. T-solvers include algorithms to return conjunction $\eta_1 = t_1 \wedge \cdots \wedge t_n$ of literals in $\lambda(l_1) \wedge \cdots \wedge \lambda(l_k)$ that lead to unsatisfiability in T. Observe that $\neg\eta_1$ is valid in T.

The search space of the SAT solver is not restricted using the formula $\mathsf{sk}(\phi) \wedge \psi_1$, where ψ_1 is the *blocking clause* $\neg\lambda^{-1}(t_1) \vee \cdots \vee \neg\lambda^{-1}(t_n)$. If ϕ is unsatisfiable, the lazy decision procedure eventually generates a resolution refutation for a formula of the form $\mathsf{sk}(\phi) \wedge \psi_1 \cdots \psi_k$ where each ψ_i is a blocking clause.

Example 4. Consider the formula $\phi \equiv (y \leq 4 \vee x = 5) \wedge (x = y) \wedge (x - 6 \geq 0)$. The propositional skeleton $\mathsf{sk}(\phi)$ for ϕ is $(e_1 \vee e_2) \wedge e_3 \wedge e_4$. One possible unsatisfiable conjunction derived by interaction with a solver for this theory is $\mathsf{sk}(\phi) \wedge (\overline{e}_1 \vee \overline{e}_3 \vee \overline{e}_4) \wedge (\overline{e}_2 \vee \overline{e}_3 \vee \overline{e}_4)$. $\lhd$

See [4] for an overview of techniques for generating small blocking clauses and using information deduced by the solver. Another approach to deciding formulae in a theory is to propositionally encode constraints on the literals in $\mathsf{sk}(\phi)$ and delegate all reasoning to the SAT solver. *Eager* approaches generate a formula $\mathsf{sk}(\phi) \wedge \Theta$, where Θ is a conjunction of constraints of the form $e_i \Leftrightarrow \theta_i$ in which e_i is an atom from the propositional skeleton and θ_i is a propositional formula that constrains e_i. The constraints ensure that every truth assignment to $\mathsf{sk}(\phi) \wedge \Theta$ corresponds to a truth assignment to ϕ in the theory T and vice versa.

Example 5. Consider the formula $\phi \equiv (x = y) \wedge ((x \& 2) = 2)$ in which the variables x and y are interpreted as bit-vectors of width 2. This formula can be propositionally encoded because each bit in x can be represented by the propositional variables x_0 and x_1. The same applies for y.

The propositional skeleton $\mathsf{sk}(\phi)$ is $e_1 \wedge e_2$. The constraints Θ are encoded by the formula $(e_1 \Leftrightarrow (x_0 \Leftrightarrow y_0 \wedge x_1 \Leftrightarrow y_1))$. The propositional formula $\mathsf{sk}(\phi) \wedge \Theta$ is checked by a SAT solver to determine satisfiability of ϕ. $\lhd$

Decision procedures may combine both approaches by propositionally encoding some constraints and using a T-solver to determine if satisfying assignments generated by the SAT solver can be satisfied in the theory T. Thus, we assume that for an unsatisfiable formula ϕ in theory T, a T-solver produces a resolution refutation for a CNF encoding of a formula of the form

$$\Phi = \mathsf{sk}(\phi) \wedge \bigwedge_{i=1}^{k} \psi_i \wedge \bigwedge_{j=1}^{n} e_j \Rightarrow \alpha_j \wedge \bigwedge_{j=1}^{n} \alpha_j \Rightarrow e_j \tag{1}$$

where ψ_i is a blocking clause over skeleton literals, e_j is an atom in $\mathsf{sk}(\phi)$, and α_i is a propositional encoding of a constraint on e_i.

3.2 Interpolants in Lazy Approaches

We briefly review how interpolants are constructed using a purely lazy decision procedure.

Definition 9. *Let $A \wedge B$ be a formula that is unsatisfiable in a theory T. A* Craig interpolant *is a formula I such that $A \models_T I$, $B \models_T \neg I$, $I \preceq A$ and $I \preceq B$.*

Craig interpolants are referred to as interpolants. An *interpolating decision procedure* is one that given a formula $A \wedge B$, decides if $A \wedge B$ is unsatisfiable and returns an interpolant if this is the case. An *interpolating decision procedure for conjunctions* requires A and B to be conjunctions of literals.

If a lazy decision procedure generates a resolution proof, interpolants can be constructed using an interpolating decision procedure for conjunctions and interpolation rules for resolution. The decision procedure is used to generate *blocking clause interpolants*, which are defined below.

Definition 10. *Let $\phi \equiv A \wedge B$ be an unsatisfiable formula in a theory T, $\mathsf{sk}(\phi)$ be of the form $\mathsf{sk}(A) \wedge \mathsf{sk}(B)$ and $\lambda : \mathcal{L}_{\mathrm{Atoms}(\mathsf{sk}(\phi))} \rightarrow \mathcal{L}_{\mathrm{Atoms}(\phi)}$. A* blocking clause interpolant *I_C for a blocking clause C over literals in $\mathsf{sk}(\phi)$ is an interpolant for the formula $A_C \wedge B_C$, where*

$$A_C = \bigwedge_{l \in C \setminus \mathcal{L}_{\mathrm{Atoms}(B)}} \lambda(l) \quad \textit{and} \quad B_C = \bigwedge_{l \in C \cap \mathcal{L}_{\mathrm{Atoms}(B)}} \lambda(l).$$

Blocking clause interpolants are then combined with propositional interpolants to obtain an interpolant for a formula. One can combine these interpolants using the rules provided either by Pudlák [13] or McMillan [3]. Both annotate the nodes of a resolution proof with *partial interpolants*. The annotation of the empty clause is the interpolant for the formula. This general scheme, using the annotation system of McMillan [3], appears in [5] and is defined below.

Definition 11. *Let $\phi \equiv A \wedge B$ be an unsatisfiable formula in a theory T and R be a resolution refutation for a CNF encoding of $\mathsf{sk}(A) \wedge \mathsf{sk}(B) \wedge_{i=1}^{k} \psi_k$, where ψ_i is a blocking clause. A* partial interpolant *is inductively defined as an annotation of the clauses in a proof as follows:*

1. *If C is a blocking clause, the annotated clause is $C \quad [I_C]$, where I_C is a blocking clause interpolant.*

2. *If C is not a blocking clause but is initial and part of the CNF encoding of $\mathsf{sk}(A)$, the annotated clause is $C \quad [C \cap \mathcal{L}_{\mathrm{Atoms}(\mathsf{sk}(B))}]$.*
3. *If C is not a blocking clause but is initial and part of the CNF encoding of $\mathsf{sk}(B)$, the annotated clause is $C \quad [\top]$.*
4. *If $x \in Var(A) \setminus Var(B)$, then*

$$\frac{C \vee x \quad [I_1] \qquad D \vee \overline{x} \quad [I_2]}{C \vee D \quad [I_1 \vee I_2]} \quad [\mathsf{A-Res}]$$

5. *If $x \in Var(B)$, then*

$$\frac{C \vee x \quad [I_1] \qquad D \vee \overline{x} \quad [I_2]}{C \vee D \quad [I_1 \wedge I_2]} \quad [\mathsf{B-Res}]$$

Theorem 1 ([14]). *Let $\phi \equiv A \wedge B$ be an unsatisfiable conjunction in the theory T, R be a resolution refutation for $\mathsf{sk}(A) \wedge \mathsf{sk}(B) \wedge \bigwedge_{i=1}^{k} \psi_i$, where each ψ_i is a blocking clause and λ be as before. Let C be a clause labelling a vertex in R and I be the partial interpolant annotating this clause. The following statements hold:*

1. *$A \models_T I \vee \bigvee_{l \in C \setminus \mathcal{L}_{\mathrm{Atoms}(\mathsf{sk}(B))}} \lambda(l)$*
2. *$B \models_T \neg I \vee \bigvee_{l \in C \cap \mathcal{L}_{\mathrm{Atoms}(\mathsf{sk}(B))}} \lambda(l)$*
3. *$I \preceq A$ and $I \preceq B$.*

It follows from this theorem that the annotation of the empty clause is an interpolant for $A \wedge B$.

3.3 Beyond Interpolants in Lazy Approaches

The method for constructing interpolants in Definition 11 is restricted to proofs of unsatisfiability generated by a purely lazy approach. We now show how proof transformations can be used to construct interpolants for approaches that combine the lazy approach with propositionally encoded constraints. A special case where this applies is decision procedures that are entirely based on propositionally encoded constraints. No interpolating decision procedure exists for such methods. We make the following assumptions:

1. Given an unsatisfiable conjunction $\phi = A \wedge B$, the T-solver used generates a resolution refutation for a CNF encoding of a formula Φ as in Equation 1.
2. Let $C_1 \wedge \ldots \wedge C_m$ be the CNF formula encoding Φ in Equation 1. We assume that there exists a function trace mapping each C_i to one of $\mathsf{sk}(\phi), \psi_i, e_j \Rightarrow \alpha_j, \alpha_j \Rightarrow e_j$ in Φ. That is, we can determine if an initial clause in the resolution proof generated by the T-solver is derived from the propositional skeleton, a blocking clause or a propositionally encoded constraint.
3. Recall that λ is a mapping from literals in the propositional skeleton of ϕ to literals over atoms inϕ. We assume that for each constraint of the form $e_i \Leftrightarrow \alpha_i$ that α_i and $\lambda(e_i)$ are equisatisfiable.

4. An interpolating decision procedure for conjunctions of theory literals exists.

Purely lazy approaches admit a two-tier interpolant construction process, where interpolants are first constructed for conjunctions of theory literals and then composed with Boolean connectives based on the structure of the proof. The challenge when dealing with a refutation of a formula Φ as in Equation 1 is that such a construction process may no longer be possible. First, there is no clear separation in the proof between theory specific and Boolean reasoning. Second, even if such a separation exists, the clauses corresponding to propositional constraints cannot be labelled with interpolants because the theory literals they encode are not necessarily unsatisfiable.

The solution we propose is to transform a resolution refutation R of Φ (from Eqn. 1) to a refutation R' for a formula $\mathsf{sk}(\phi) \wedge \Psi \wedge \Gamma$, where Ψ is the conjunction of blocking clauses in the original formula and Γ is a conjunction of blocking clauses that the SAT solver has deduced. If ϕ is unsatisfiable, there exist blocking clauses for all satisfying assignments to $\mathsf{sk}(\phi)$. Satisfying assignments that are not explicitly excluded by blocking clauses must be excluded by reasoning with propositionally encoded constraints. In other words, these constraints implicitly encode blocking clauses and by reordering the proof, we can obtain these blocking clauses. Under the assumption that an interpolating decision procedure for conjunctions of theory literals exists, we can apply the construction in Definition 11 to obtain an interpolant.

We first derive the contradiction that the SAT solver constructs in terms of theory literals and then show how reordering can be used to ensure that this contradiction is in CNF. Similar to interpolant construction, we annotate the vertices of a proof with a formula representing the combination of theory literals that eventually yield a contradiction.

Definition 12. *Let R be a resolution refutation of a CNF encoding of Φ in Equation 1. The formula $\mathsf{contra}(v)$ is defined over vertices in V_R as follows:*

1. *For an initial vertex v,*
   (a) $\mathsf{contra}(v) = \bot$ *if* $\mathsf{trace}(\ell(v))$ *is the propositional skeleton* $\mathsf{sk}(\phi)$*, and*
   (b) $\mathsf{contra}(v) = \bigwedge_{l \in \ell(v)} \neg\lambda^{-1}(l)$ *if* $\mathsf{trace}(\ell(v))$ *is a blocking clause* ψ_i*, and*
   (c) $\mathsf{contra}(v) = \lambda^{-1}(e_i)$ *if* $\mathsf{trace}(\ell(v))$ *is the implication* $e_i \Rightarrow \alpha_i$ *in a propositional constraint, and*
   (d) $\mathsf{contra}(v) = \neg\lambda^{-1}(e_i)$ *if* $\mathsf{trace}(\ell(v))$ *is the reverse implication* $\alpha_i \Rightarrow e_i$ *in a propositional constraint.*
2. *For an internal vertex v,*
   (a) *if* $piv(v) \in \mathrm{Atoms}(\mathsf{sk}(\phi))$*, then* $\mathsf{contra}(v) = \mathsf{contra}(v^+) \vee \mathsf{contra}(v^-)$*, and*
   (b) *if* $piv(v) \notin \mathrm{Atoms}(\mathsf{sk}(\phi))$*, then* $\mathsf{contra}(v) = \mathsf{contra}(v^+) \wedge \mathsf{contra}(v^-)$*.*

The construction $\mathsf{contra}(v)$ yields a formula with a useful property. If the label $\ell(v)$ of a vertex v contains only clauses from the propositional skeleton, then $\mathsf{contra}(v)$ is unsatisfiable in the theory T.

Theorem 2. *Let ϕ be a theory formula, R be a resolution refutation for a formula Φ as in Equation 1, and $S = \mathcal{L}_{\mathrm{Atoms}(\Phi)} \setminus \mathcal{L}_{\mathrm{Atoms}(\mathsf{sk}(\phi))}$. If $\ell(v) \cap S = \emptyset$, then $\mathsf{contra}(v)$ is a contradiction in T.*

Proof: Fill in details when I'm more awake. ■

Observe that for any vertex v in a proof, if $\pi(v)$ contains no pivots in $\mathrm{Atoms}(\mathsf{sk}(\phi))$, then $\mathsf{contra}(v)$ must be a conjunction of T-literals. Moreover, if $\ell(v)$ contains only literals from the propositional skeleton, $\mathsf{contra}(v)$ is an unsatisfiable conjunction and can be mapped to a blocking clause. To construct an interpolant, it remains to ensure that the refutation generated by the solver contains a sub-proof with such vertices v as initial vertices. Lemma 5 shows how Reorder can be used to obtain such a proof.

Lemma 5. *Let ϕ be an unsatisfiable CNF formula and $S \subseteq \mathrm{Atoms}(\phi)$ be a set of atoms. Define the partitioned partial order $(\beta_S, \sqsubseteq_S)$ such that for all $x, y \in \mathrm{Atoms}(\phi)$, $\beta_S(x) \neq \beta_S(y)$ if $x \in S, y \notin S$, and $\beta_S(x) = \beta_S(y)$ otherwise, and $\beta_S(x) \sqsubseteq_S \beta_S(y)$ if $x \in S$. Let R be a refutation of ϕ that respects $(\beta_S, \sqsubseteq_S)$.*

1. *There exists a set of vertices $W \subseteq V_R$ such that (a) every path from s_R to an initial vertex traverses a vertex in W, and (b) for each $v \in W$, and vertex v' in $\pi(v)$, $piv(v') \notin S$.*
2. *Define the mapping $\rho = \{v \mapsto \ell_R(v) | v \in W\}$. Then, $R[\rho]$ is a resolution refutation for $\bigwedge_{v \in W} \ell_R(v)$.*

Let S in Lemma 5 be $\mathrm{Atoms}(\mathsf{sk}(\phi))$. Observe that W induces a cut in the proof graph and that by Theorem 2, for $v \in W$, $\mathsf{contra}(v)$ is a contradictory conjunction of theory literals. Thus, we can compute an interpolant from the proof $R[\rho]$ by annotating each $v \in W$ with a partial interpolant obtained from $\mathsf{contra}(v)$. In general, the entire proof need not be reordered. It is not necessary to move resolution steps that do not affect the propositional structure of $\mathsf{contra}(v)$. Again, this choice can be made heuristically. Moving a pivot towards the sink of the proof may require few changes, but will generate a few large unsatisfiable conjunctions. If all skeleton atoms are resolved after constraint atoms, there may be many small unsatisfiable conjunctions. A detailed example follows.

Example 6. Consider the bit-vector formula $((x = y) \Rightarrow ((x \,\&\, 2) = 2)) \wedge (y = z + z) \wedge (x = z \ll 1) \wedge ((z \,\&\, 1) = 0)$, with $\mathsf{sk}(\phi)$ being $(e_1 \Rightarrow e_2) \wedge e_3 \wedge e_4 \wedge e_5$. The constraints on each skeleton atom and the corresponding CNF clauses are shown below.

Encoding	Propositional constraint	CNF clauses
$e_1 \Leftrightarrow (x = y)$	$e_1 \Leftrightarrow (x_0 \Leftrightarrow y_0) \wedge e_1 \Leftrightarrow \dots$	$(\overline{e_1}\,\overline{x_0} y_0)(\overline{e_1} x_0 \overline{y_0})(e_1 \overline{x_0}\,\overline{y_0})(e_1 x_0 y_0) \dots$
$e_2 \Leftrightarrow ((x \& 2) = 2)$	$e_2 \Rightarrow x_1$	$(\overline{e_2} x_1)$
$e_3 \Leftrightarrow (y = z + z)$	$e_3 \Rightarrow \overline{y_0}$	$(\overline{e_3}\,\overline{y_0})$
$e_4 \Leftrightarrow (x = z \ll 1)$	$e_4 \Rightarrow (x_1 \Leftrightarrow z_0) \wedge (e_4 \Rightarrow \overline{x_0})$	$(\overline{e_4}\,\overline{x_1} z_0)(\overline{e_4} x_1 \overline{z_0})(\overline{e_4}\,\overline{x_0})$
$e_5 \Leftrightarrow ((z \& 1) = 0)$	$e_5 \Rightarrow \overline{z_0}$	$(\overline{e_5}\,\overline{z_0})$

$$\begin{tikzcd}
 & & & & & & & & & \overline{e_1}e_2 \arrow[dr] & & e_1x_0y_0 \arrow[dl] & & \\
 & & e_4 \arrow[dddr] & & & & & & \overline{e_2}x_1 \arrow[dr] & & e_2x_0y_0 \arrow[dl] & & e_3 \arrow[ddddll] & \\
 & & & & \overline{e_4}\,\overline{x_1}z_0 \arrow[ddl] & & & \overline{e_3}\,\overline{y_0} \arrow[ddr] & & x_0x_1y_0 \arrow[ddl] & & & & \\
e_5 \arrow[dr] & \overline{e_5}\,\overline{z_0} \arrow[d] & & & & & & & & & & & & \\
 & \overline{z_0} \arrow[dr] & & \overline{x_1}z_0 \arrow[dl] & & & & & \overline{e_3}\,x_0x_1 \arrow[drr] & & & \overline{e_4}\,\overline{x_0} \arrow[dr] & & e_4 \arrow[dl] \\
 & & \overline{x_1} \arrow[ddrrrr] & & & & & & & & x_0x_1 \arrow[d] & & \overline{x_0} \arrow[dll] & \\
 & & & & & & & & & & x_1 \arrow[dllll] & & & \\
 & & & & & & \Box & & & & & & &
\end{tikzcd}$$

(a) Unordered proof

$$\begin{tikzcd}
 & & e_4 \arrow[dddr] & & & & & & & & & & e_3 \arrow[ddddlll] & \\
 & & & & \overline{e_4}\,\overline{x_1}z_0 \arrow[ddl] & & & & e_1x_0y_0 \arrow[dr] & & \overline{e_3}\,\overline{y_0} \arrow[dl] & & & \\
e_5 \arrow[dr] & \overline{e_5}\,\overline{z_0} \arrow[d] & & & & & & & & e_1x_0\overline{e_3} \arrow[dd] & & & & \\
 & \overline{z_0} \arrow[dr] & & \overline{x_1}z_0 \arrow[dl] & & \overline{e_2}x_1 \arrow[dr] & & \overline{e_1}e_2 \arrow[dl] & & & & \overline{e_4}\,\overline{x_0} \arrow[dr] & & e_4 \arrow[dl] \\
 & & \overline{x_1} \arrow[drr] & & & & \overline{e_1}x_1 \arrow[dll] & & & e_1x_0 \arrow[dr] & & & \overline{x_0} \arrow[dll] & \\
 & & & & \overline{e_1} \arrow[drrr] & & & & & & e_1 \arrow[dlll] & & & \\
 & & & & & & & \Box & & & & & &
\end{tikzcd}$$

(b) Refactored and ordered proof

Fig. 6. Refactoring proofs for the purpose of computing interpolants.

A refutation over these clauses is shown in Figure 6(a). Observe that skeleton atoms are resolved before constraint atoms on all paths. The optimisation suggested above is applicable; it suffices to project e_1 to obtain contradictory conjunctions. The proofs $\pi(\Box) \downharpoonleft \overline{e}_1$, which is the left sub-proof in Figure 6(b), and $\pi(\Box) \downharpoonleft e_1$, which is the right sub-proof, are resolved to obtain $\Box$.

We now construct $\mathsf{contra}(v)$. For example, $\mathsf{contra}(\overline{e}_5\overline{z}_0) = ((z\&1) = 0)$ and $\mathsf{contra}(\overline{e}_2 x_1) = ((x\&2) = 2)$. In particular, $\mathsf{contra}(\overline{e}_1) = ((z\&1) = 0) \wedge (x_1 \Leftrightarrow z_0) \wedge ((x\&2) = 2)$, and $\mathsf{contra}(e_1) = \neg(x = y) \wedge (y = z + z) \wedge (x = z \ll 1)$ are both contradictory conjunctions.

Define a partition of the formula where A is $((x = y) \Rightarrow ((x \,\&\, 2) = 2)) \wedge (y = z + z)$, and B is$(x = z \ll 1) \wedge ((z \,\&\, 1) = 0)$. The partial interpolant (using a method explained next) for $\mathsf{contra}(\overline{e}_1)$ is $\neg(x = z + z)$, and for $\mathsf{contra}(e_1)$ is $((x\&2) = 2)$. The interpolant is the disjunction of these two. We emphasise that no existing tools can compute interpolants for such a formula. $\lhd$

Theory combinations. We briefly describe how to extended our method for theory combinations, by relaxing the restrictions on solver heuristics in [5]. We consider combination methods based on equalities deduced by individual T-solvers, where the solvers communicate equalities à la Nelson-Oppen's method, or use a SAT-solver to integrate equalities via Delayed Theory Combination. Method for computing interpolants in these settings appear in [5, 15].

In both cases, we obtain a refutation for a formula of the form $\mathsf{sk}(\phi) \wedge \Psi \wedge \Omega^{=}$, where Ψ is a conjunction of blocking clauses and the atoms in $\Omega^{=}$ encode equality constraints. We only need modifications to deal with equalities over terms from both parts of the formula for interpolation. The modifications in [5] restrict the solver to produce a refutation in which such *mixed equalities* only appear in sub-proofs and are resolved before any other atoms. An alternative is to define a partitioned partial order such that mixed equalities in $\Omega^{=}$ are in the same block, and this block is maximum with respect to the partial order. Reordering can generate a proof which respects this order. It remains to replace each mixed equality by two pure equalities and resolutions on a mixed equality atom by two resolution steps on pure equality atoms. This is achieved by substitution using the proofs indicated in [5].

Solving Bit-vector Formulae. To compute interpolants for a fragment of bit-vector logic that appears in program properties, we use a simple instantiation rule for theory axioms to introduce tautological equalities on demand:

$$\textsc{Ax-Inst}\ \frac{\vdash \forall \overrightarrow{a}.f(\overrightarrow{a}) = g(\overrightarrow{a})}{f(\overrightarrow{x}) = g(\overrightarrow{x})}\ \dagger$$

where $\forall \overrightarrow{a}.f(\overrightarrow{a}) = g(\overrightarrow{a})$ is an axiom of the theory. The symbol † indicates the requirement that f as well as g are interpreted functions of the theory, and that $\overrightarrow{x} \preceq A$ or $\overrightarrow{x} \preceq B$, respectively. This is not sufficient to deal with full bit-vector arithmetic, but enables handling cases that occur in practice, in particular formulae as in Example 6. Useful examples are the bit-vector axioms for dealing with bit-shifts and bit-masks: (1) $\forall t.t \ll 1 = t + t$, (2) $\forall t.(t \ll c_1)\&c_2 = 0$, and (3) $\forall t.(t \,|\, c)\&c = c$, where t is a term, and c, c_1 and c_2 are constants.

Example 7. If the axiom $\forall t.t \ll 1 = t + t$ is instantiated with $z \ll 1$, we can compute an interpolant for the unsatisfiable conjunction $\neg(x = y) \wedge (y = z + z) \wedge (x = z \ll 1)$ using an interpolating decision procedure for equality logic. $\lhd$

4 Evaluation

We evaluated the feasibility of the proposed techniques by integrating them into our verification tool suite consisting of the hardware model checker EBMC and the predicate abstraction framework SATABS [16].[3] We discuss (a) the impact of Reorder on the proof size during predicate abstraction of hardware benchmarks with bit-vector terms, and (b) the performance of our interpolation technique when used as refinement algorithm in SATABS, using the DDVerify benchmarks. The change in proof size in the latter is negligible.[4]

EBMC is a hardware model checker that implements interpolation. The first column of Table 2 lists benchmarks from the Sun PicoJava II, opencores (http://www.opencores.org), Texas97 and VIS benchmark suites. Subsequent columns show the unwinding bound k, the size of the refutation before and after applying Reorder, measured as number of resolution steps (clauses in proof

[3] Both tools are available for download from http://www.verify.ethz.ch/

[4] http://www.verify.ethz.ch/ddverify/

model	k	original proof	after REORDER	time [s]	# blk. cl.
cache-coherence	10	881	913	≈0	61
	11	981	1021	≈0	68
	12	1081	1131	≈0	75
	13	1181	1241	≈0	82
	14	1281	1353	≈0	89
	15	1381	1465	≈0	96
ethernet	10	1362	1365	≈0	32
	11	9572	10192	≈0	325
	12	14180	15294	0.5	461
	13	26236	14437	43	509
Miim	10	4446	3343	0.2	358
	11	4886	3030	0.5	320
	12	5086	5337	0.3	570
	13	6025	5962	0.7	650
pj-icu	12	509	565	≈0	28
	13	617	1001	≈0	32
	14	552	568	≈0	33
	15	624	665	≈0	37
usb-phy	10	567	571	≈0	49
	11	1071	1082	≈0	85
	12	1156	1170	≈0	90
	13	1332	1352	≈0	102

Table 2. Impact of REORDER on the proof size (given as number of resolution steps)

dev. driver	iter.	CSISAT [s]	lifting [s]	dev. driver	iter.	CSISAT [s]	lifting [s]
machzwd	11	5.52	4.06	nbd	13	143.60	11.74
pcwd_pci	11	7.29	4.76	toshiba	8	0.69	0.90
umem	12	13.95	4.37	cs5535_gpio	6	5.31	1.81
efirtc	15	7.95	5.19	nwbutton	17	24.01	5.65

Table 3. Runtimes of interpolating decision procedures in DDVerify [18]

minus initial clauses), and the time for reordering. The last column shows the number of blocking clauses identified by our algorithm.

Contrary to what one may expect, though reordering involves copying, the resulting proof may be smaller. The proof size of the ethernet entry with width 13 is almost halved by reordering. The reason is that each update by REORDER makes the proof more *regular*. As Tseitin explains [10], "*The regularity condition can be interpreted as a requirement for not proving intermediate results in a form stronger than that in which they are later used.*" A tree refutation of minimal size is regular, though a regular refutation may be smaller than the minimal tree. The relationship between the sizes of regular refutations and those generated by SAT-solvers is a topic of ongoing research [17].

In Table 3, we list the runtime of using our interpolation algorithm in predicate abstraction. We measure the total time for propositionally encoding a counterexample and constructing interpolants, over all iterations till the CEGAR loop terminates. A counterexample ϕ is typically a conjunction of constraints, so the proof size remains unchanged in all examples. We use CSIsat [19] to compute the partial interpolants for the blocking clauses our algorithm derives. We compare the runtime of our tool to that of CSIsat for directly computing interpolants (without propositional encoding). In most cases, we observe a significant speedup, which we believe is because our algorithm extracts from the core only blocking clauses that lead to unsatisfiability.

5 Related Work

Transformations on resolution proofs appear in proofs about the completeness of resolution refinements in the automated deduction literature, and for deriving lower bounds in proof complexity. The idea of reordering dates back (at least) to Andrews [20], and substitution was suggested by Tseitin [10]. A full survey is beyond the scope of this paper. Proof transformations have been applied to reduce the size of unsatisfiable cores [8] and to strengthen interpolants [2].

Several linear-time algorithms for computing interpolants from resolution refutations exist. Huang [21] also considers paramodulation, Pudlák's [13] system includes bounded arithmetic, and McMillan's [14] applies to linear arithmetic and equality with uninterpreted functions. Interpolating decision procedures can be combined using the Nelson-Oppen approach as in [15] or using delayed theory combination [5]. All these methods are based on proof producing decision procedures. An technique for conjunctions of literals in linear arithmetic that does not rely on proofs appears in [22] and is implemented in the CSIsat tool [19]. This method is compatible with the techniques we present here. Finally, a first attempt to construct interpolants from proofs that include propositionally encoded equality constraints is [23]. This method incomplete, since it does not reorder the proof.

References

1. Henzinger, T.A., Jhala, R., Majumdar, R., McMillan, K.L.: Abstractions from proofs. In: Principles of Programming Languages (POPL), ACM (2004) 232–244
2. Jhala, R., McMillan, K.L.: Interpolant-based transition relation approximation. In: CAV. Volume 3576 of LNCS., Springer (2005) 39–51
3. McMillan, K.L.: Interpolation and SAT-based model checking. In: CAV. Volume 2725 of LNCS., Springer (2003) 1–13
4. Cimatti, A., Sebastiani, R.: Building efficient decision procedures on top of SAT solvers. In: Formal Methods for the Design of Computer, Communication, and Software Systems. Volume 3965 of LNCS., Springer (2006) 144–175
5. Cimatti, A., Griggio, A., Sebastiani, R.: Efficient interpolant generation in satisfiability modulo theories. In: TACAS. Volume 4963 of LNCS., Springer (2008) 397–412
6. Anderson, R., Bledsoe, W.W.: A linear format for resolution with merging and a new technique for establishing completeness. J. ACM **17** (1970) 525–534

7. Ben-Sasson, E., Wigderson, A.: Short proofs are narrow—resolution made simple. J. ACM **48** (2001) 149–169
8. Bar-Ilan, O., Fuhrmann, O., Hoory, S., Shacham, O., Strichman, O.: Linear-time reductions of resolution proofs. Technical Report IE/IS-2008-02, Technion (2008)
9. Gershman, R., Koifman, M., Strichman, O.: Deriving small unsatisfiable cores with dominators. In: CAV. Volume 4144 of LNCS., Springer (2006) 109–122
10. Tseitin, G.: On the complexity of proofs in poropositional logics. In: Automation of Reasoning: Classical Papers in Computational Logic 1967–1970. Volume 2., Springer (1983) Originally published 1970.
11. Jhala, R., McMillan, K.L.: Interpolant-based transition relation approximation. Logical Methods in Computer Science **3** (2007)
12. Urquhart, A.: The complexity of propositional proofs. Bulletin of Symbolic Logic **1** (1995) 425–467
13. Pudlák, P.: Lower bounds for resolution and cutting plane proofs and monotone computations. The Journal of Symbolic Logic **62** (1997) 981–998
14. McMillan, K.L.: An interpolating theorem prover. Theoretical Computer Science **345** (2005) 101–121
15. Yorsh, G., Musuvathi, M.: A combination method for generating interpolants. In: Computer Aided Deduction. (2005) 353–368
16. Clarke, E.M., Kroening, D., Sharygina, N., Yorav, K.: Predicate abstraction of ANSI–C programs using SAT. Formal Methods in System Design (FMSD) **25** (2004) 105–127
17. Hertel, P., Bacchus, F., Pitassi, T., Gelder, A.V.: Clause learning can effectively p-simulate general propositional resolution. In: AAAI Conference on Artificial Intelligence. (2008) 283–290
18. Witkowski, T., Blanc, N., Kroening, D., Weissenbacher, G.: Model checking concurrent Linux device drivers. In: ASE, IEEE (2007) 501–504
19. Beyer, D., Zufferey, D., Majumdar, R.: CSIsat: Interpolation for LA+EUF. In: CAV. Volume 5123 of LNCS., Springer (2008) 304–308
20. Andrews, P.B.: Resolution with merging. J. ACM **15** (1968) 367–381
21. Huang, G.: Constructing Craig interpolation formulas. In: Computing and Combinatorics (COCOON). Volume 959 of LNCS., Springer (1995) 181–190
22. Rybalchenko, A., Sofronie-Stokkermans, V.: Constraint solving for interpolation. In: VMCAI. Volume 4349 of LNCS., Springer (2007) 346–362
23. Kroening, D., Weissenbacher, G.: Lifting propositional interpolants to the word-level. In: FMCAD, IEEE (2007) 85–89