Reduction of Resolution Refutations and Interpolants via Subsumption

Roderick Bloem[1⋆], Sharad Malik[2⋆⋆], Matthias Schlaipfer[3⋆], and Georg Weissenbacher[3⋆⋆⋆]

[1] Graz University of Technology, Austria
[2] Princeton University
[3] Vienna University of Technology, Austria

Abstract. Propositional resolution proofs and interpolants derived from them are widely used in automated verification and circuit synthesis. There is a broad consensus that "small is beautiful" — small proofs and interpolants lead to concise abstractions in verification and compact designs in synthesis. Contemporary proof reduction techniques either minimise the proof during construction, or perform a *post-hoc* transformation of a given resolution proof. We focus on the latter class and present a subsumption-based proof reduction algorithm that extends existing single-pass analyses and relies on a *meet-over-all-paths* analysis to identify redundant resolution steps and clauses. We show that smaller refutations do not necessarily entail smaller interpolants, and use labelled interpolation systems to generalise our reduction approach to interpolants. Experimental results support the theoretical claims.

1 Introduction

Resolution proofs and interpolants are an integral part of many verification-related techniques such as abstraction [24] and model checking [17], vacuity detection [29], synthesis [18, 20], and patch generation [32]. These techniques take advantage of the fact that refutations and interpolants direct the focus to the *core* of the problem instance (literally and metaphorically). In practice, small refutations provide concise abstractions in model checking [24], and small interpolants enable precise refinement and compact designs in synthesis [20].

Consequently, proof reduction as well as the minimisation of unsatisfiable cores has received ample attention. We roughly group the resulting reduction approaches into two categories: techniques that minimise the proof during construction, and techniques that rely on a *post-hoc* proof transformation. Algorithms for the extraction of minimal unsatisfiable subsets (such as [25, 4]) typically fall into the former category and rely on iterative calls to a SAT solver.

⋆ Supported by the Austrian Science Fund (FWF) through grants S11403-N23 (National Research Network RiSE) and W1255-N23 (LogiCS doctoral programme).
⋆⋆ Funded by C-FAR, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP), a Semiconductor Research Corporation program sponsored by MARCO and DARPA.
⋆⋆⋆ Funded by grant VRG11-005 of the Vienna Science and Technology Fund (WWTF).

Representatives of the latter class reduce the proof size by identifying and eliminating redundancies (e.g. [3, 16, 28, 7]). The focus of these reduction algorithms is not on minimality but scalability, which is why they avoid additional SAT calls. Naturally, there are also hybrid approaches: Gershman et al. [14], for instance, rely on a solver to detect redundancies in a given proof.

The focus of our work is on *post-hoc* proof transformations. The motivation for this decision is two-fold. Firstly, while small proofs and interpolants are desirable, minimality is often not necessary and comes at the cost of scalability [3]. Secondly, it is possible to harvest information from a complete proof that is not available during proof construction. This idea is very explicit in [16], where a *meet-over-all-paths* analysis identifies redundant literals and resolution steps (discussed in §3.1). Other authors [3, 28] deploy a richer set of transformation rules (including pivot and unit recycling), but fail to exploit the information readily available in the proof. In §3, we cast pivot and unit recycling [3, 16, 13, 28] more generally as *subsumption* and generalise them in a single concise transformation rule (Theorem 1). Subsumption has been successfully deployed during proof construction (in [33], for instance). We use subsumption as a post-processing step and carry forward the idea of [16] to use proof analysis to identify redundancies that were not eliminated during proof construction. proof construction (in [33], for instance, and implicitly in [15]). We use subsumption as a post-processing step and carry forward the idea of [16] to use proof analysis to identify redundancies not eliminated during proof construction.

Interpolation is often an after-thought to proof reduction. It is common practice to extract interpolants from a reduced proof [27] and to subsequently compact the result by removing structural redundancy [8]. We show in §4 that pivot and unit recycling can actually *increase* the number of variables in an interpolant. In §4, we lift the results from §3 to *labelled* clauses in the framework of labelled interpolation systems [12], thus avoiding transformations that introduce nonessential [10] (or peripheral [29]) variables.

Contributions. In §3, we present a single concise transformation rule (Theorem 1) which, based on subsumption, generalises existing proof reduction techniques [3, 13, 16]. We show in §4 that careless transformations may increase interpolant size, and lift the results from §3 to labelled clauses [12] to rule out detrimental reductions (Theorem 2). §5 covers our implementation and provides an experimental evaluation that demonstrates a small but consistent improvement over [13, 16].

2 Notation and Preliminaries

This section introduces our notation and restates some prior results on proof restructuring [11] in §2.1, and labelled interpolation systems [12, 10] in §2.2.

2.1 Formulae, Proofs, and Transformations

Propositional Formulae. We work in the standard setting of propositional logic. Formulas are defined over a set X of propositional variables, the logical con-

stants T and F (denoting true and false, respectively), and the standard logical connectives $\wedge$, $\vee$, $\Rightarrow$, and $\neg$ (denoting conjunction, disjunction, implication, and negation, respectively).

$\mathtt{Lit}_X = \{x, \overline{x} \,|\, x \in X\}$ is the set of literals over X, where $\overline{x}$ is short for $\neg x$. We write $\mathrm{var}(t)$ for the variable occurring in the literal $t \in \mathtt{Lit}_X$, and $\mathrm{Var}(F)$ to denote the variables occurring in a formula F. A clause C is a set of literals interpreted as a disjunction. A clause C *subsumes* a clause D if $C \subseteq D$. The empty clause $\square$ contains no literals and is interpreted as F. The disjunction of two clauses C and D is their union, denoted $C \vee D$, which is further simplified to $C \vee t$ if D is the singleton $\{t\}$. A propositional formula in Conjunctive Normal Form (CNF) is a conjunction of clauses, also represented as a set of clauses.

Resolution Proofs. The resolution rule is an inference rule deriving a new clause from two clauses containing complementary literals. The clauses $C \vee x$ and $D \vee \overline{x}$ are the *antecedents*, x is the *pivot*, and $C \vee D$ is the *resolvent*. $\mathrm{Res}(C, D, x)$ denotes the resolvent of C and D with the pivot x.

Definition 1 (Resolution Proof, Refutation). *A* resolution proof R *is a directed acyclic graph* $(V_R, E_R, piv_R, \ell_R, \mathtt{s}_R)$*, where* V_R *is a set of vertices,* E_R *is a set of edges,* piv_R *is a function mapping vertices to pivot variables,* ℓ_R *is a function mapping vertices to formulae, and* $\mathtt{s}_R \in V_R$ *is a designated sink vertex. An* initial vertex *has in-degree* 0*. All other vertices are* internal *and have in-degree* 2*. The sink* $\mathtt{s}_R$ *has out-degree* 0*. For every internal vertex* v *with* $(v_1, v), (v_2, v) \in E_R$*, we have* $\ell_R(v) = \mathrm{Res}(\ell_R(v_1), \ell_R(v_2), piv_R(v))$*. A resolution proof* R *is a* resolution refutation *if* $\ell_R(\mathtt{s}_R) = \square$*.*

The subscripts above are dropped if clear from the context. A vertex $v_i \in R$ is a *parent* of v_j if $(v_i, v_j) \in E_R$. Let v^+ and v^- be the parents of v such that $piv(v) \in \ell(v^+)$ and $\neg piv(v) \in \ell(v^-)$. A vertex v_i is an *ancestor* of v_j if there is a path from v_i to v_j. A vertex v_i dominates v_j if all paths from v_j to $\mathtt{s}_R$ visit v_i. The substitution $R[v_1 \leftarrow v_2]$ replaces the sub-proof rooted at v_1 with the sub-proof rooted at v_2:

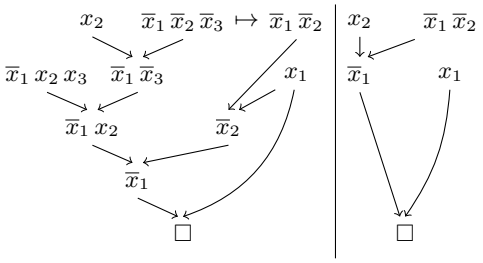
Definition 2. *Let* $R = (V_R, E_R, piv_R, \ell_R, \mathtt{s}_R)$*, and let* $v_1, v_2 \in V_R$ *(*$v_1 \neq v_2$*) such that* v_1 *is not an ancestor of* v_2*. The* substitution *of* v_1 *with* v_2 *in* R*, denoted by* $R[v_1 \leftarrow v_2]$*, is the directed acyclic graph* $G = (V_G, E_G, piv_G, \ell_G, \mathtt{s}_G)$*, where* $V_G = V_R \setminus \{v_1\}$*,* $E_G = (E_R \setminus \{(u, v) |\, u = v_1 \vee v = v_1\}) \cup \{(v_2, v) | (v_1, v) \in E_R\}$*,* $\ell_G(v) = \ell_R(v)$ *and* $piv_G(v) = piv_R(v)$ *for all* $v \neq v_1$*, and* $\mathtt{s}_G$ *is* $\mathtt{s}_R$ *if* $v_1 \neq \mathtt{s}_R$ *and* v_2 *otherwise.*

The transformation $R[v_1 \leftarrow v_2]$ does not necessarily yield a valid resolution proof. The transformation $\mathrm{RestoreRes}(G, v)$ as defined below restores the validity of the single resolution step at vertex v.

Definition 3. *Let* G *be the directed acyclic graph* $(V_G, E_G, piv_G, \ell_G, \mathtt{s}_G)$*. The transformation* $\mathrm{RestoreRes}(G, v)$ *yields* G *if* v *is an initial vertex of* G*. For an internal vertex* $v \in V_G$*,*

```
proc ReconstructProof (R, v)
  if v ∉ visited then
    visited ← visited ∪ {v}
    foreach (u, v) ∈ E_R
      ReconstructProof (R, u)
    R ← RestoreRes (R, v)
```

(a) ReconstructProof



(b) Reducing proof size

Fig. 1. Reconstructing proofs

– *if* $\exists (v^+, v), (v^-, v) \in E_G$ *with* $piv_G(v) \in \ell_G(v^+) \wedge \overline{piv(v)} \in \ell_G(v^-)$ *then*

$$\text{RestoreRes}(G, v) = (V_G, E_G, piv_G, \ell, \mathtt{s}_G)$$
$$\text{with } \ell(u) \stackrel{\text{def}}{=} \begin{cases} \text{Res}(\ell_G(v^+), \ell_G(v^-), piv(v)) & \text{if } u = v \\ \ell_G(u) & \text{otherwise} \end{cases}$$

– *otherwise, let* u *and* w *be the parents of* v*, and* $\text{RestoreRes}(G, v) = G[v \leftarrow u]$*, where* u *is chosen such that*

$$(piv(v) \in \ell(u) \Rightarrow piv(v) \in \ell(w)) \wedge (\overline{piv(v)} \in \ell(u) \Rightarrow \overline{piv(v)} \in \ell(w)).$$

The second case in Definition 3 affords us a choice for u if neither parent contains the pivot or both parents contain the pivot literal in the same phase. The latter situation can arise in proofs that contain tautological clauses.

The algorithm ReconstructProof in Figure 1(a) (introduced in [3]) performs a linear time post-order (parents first) traversal of the graph, applying RestoreRes to re-establish $\forall (v_1, v), (v_2, v) \in E \,.\, \ell(v) = \text{Res}(\ell(v_1), \ell(v_2), piv(v))$.

The following lemma is an adaptation of Lemma 2 from [11] to our setting.

Lemma 1. *Let* R *be a resolution proof, and let* $\pi = \{v_1 \mapsto u_1, \ldots, v_k \mapsto u_k\}$ *be a mapping such that* v_i *is not an ancestor of* u_j *for* $1 \leq i, j \leq k$*. If* $\ell_R(u_i) \subseteq \ell_R(v_i)$ *for* $1 \leq i \leq k$*, then the proof* P *obtained by applying* ReconstructProof *to* $R[v_1 \leftarrow u_1] \ldots [v_k \leftarrow u_k]$ *has sink* $\mathtt{s}_P$ *with* $\ell_P(\mathtt{s}_P) \subseteq \ell_R(\mathtt{s}_R)$*.*

Proof: By induction on the number of ancestors of $\mathtt{s}_R$ (cf. the more general proof of Theorem 1 on page 7). ■

Example 1. Consider the left proof in Figure 1(b), in which the mapping π from Lemma 1 is indicated by $\mapsto$. The refutation on the right of Figure 1(b) shows the result of ReconstructProof after substituting $\overline{x}_1\overline{x}_2$ for $\overline{x}_1\overline{x}_2\overline{x}_3$. ◁

2.2 Interpolation Systems and Labelling Functions

The following variant of Craig interpolants [9] has been introduced by McMillan [22] and is commonly used in the context of verification.

Definition 4 (Propositional Interpolant). *An* interpolant *for a pair of propositional formulae (A, B), where $A \wedge B$ is unsatisfiable, is a formula I such that $A \Rightarrow I$, $B \Rightarrow \neg I$, and $\mathrm{Var}(I) \subseteq \mathrm{Var}(A) \cap \mathrm{Var}(B)$ holds.*

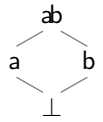
Let A and B be formulae in CNF. A refutation R is an (A, B)-*refutation* of an unsatisfiable formula $A \wedge B$ if $\ell_R(v)$ is a clause in A or a clause in B for each initial vertex $v \in V_R$.

An interpolation system Itp is a function that given an (A, B)-refutation R yields a function, denoted $\mathsf{Itp}(R, A, B)$, from vertices in R to formulae over $\mathrm{Var}(A) \cap \mathrm{Var}(B)$. An interpolation system is *correct* if for every (A, B)-refutation R with sink s, it holds that $\mathsf{Itp}(R, A, B)(\mathsf{s})$ is an interpolant for (A, B). We write $\mathsf{Itp}(R)$ for $\mathsf{Itp}(R, A, B)(\mathsf{s})$ when A and B are clear. Let v be a vertex in an (A, B)-refutation R. The pair $(\ell(v), \mathsf{Itp}(R, A, B)(v))$ is an *annotated clause* and is written $\ell(v)$ $[\mathsf{Itp}(R, A, B)(v)]$ in accordance with [23].

In the following, we review the labelled interpolation systems introduced in [12], which generalise the propositional interpolation algorithms presented by Huang [19], Krajíček [21] and Pudlák [26], and McMillan [22]. A distinguishing feature of a labelled interpolation system is that it assigns an individual label $\mathsf{c} \in \{\bot, \mathsf{a}, \mathsf{b}, \mathsf{ab}\}$ to *each literal* in the resolution refutation.

Definition 5 (Labelling Function). *Let $(\mathcal{S}, \sqsubseteq, \sqcap, \sqcup)$ be the lattice below, where $\mathcal{S} = \{\bot, \mathsf{a}, \mathsf{b}, \mathsf{ab}\}$ is a set of symbols and $\sqsubseteq$, $\sqcap$ and $\sqcup$ are defined by the Hasse diagram to the right. A* labelling function *$L_R : V_R \times \mathtt{Lit} \to \mathcal{S}$ for a refutation R over a set of literals $\mathtt{Lit}$ satisfies that for all $v \in V_R$ and $t \in \mathtt{Lit}$:*

1. *$L_R(v, t) = \bot$ iff $t \notin \ell_R(v)$*
2. *$L_R(v, t) = L_R(v^+, t) \sqcup L_R(v^-, t)$ for an internal vertex v, its parents v^+ and v^-, and literal $t \in \ell_R(v)$.*



Definition 6 (Locality). *A literal t is* A-local *if* $\mathrm{var}(t) \in \mathrm{Var}(A) \setminus \mathrm{Var}(B)$. *Conversely, t is* B-local *if* $\mathrm{var}(t) \in \mathrm{Var}(B) \setminus \mathrm{Var}(A)$. *All other literals are* shared. *A labelling function L is* locality preserving *if for any initial vertex $v \in V_R$ and $t \in \ell(v)$, $L(v, t) = \mathsf{a}$ if t is A-local and $L(v, t) = \mathsf{b}$ if t is B-local.*

Shared literals may be labelled a, b, or ab. Given a labelling function L, the downward *projection* of a clause at a vertex v with respect to $\mathsf{c} \in \mathcal{S}$ is $\ell(v)\downarrow_{\mathsf{c},L} \stackrel{\text{def}}{=} \{t \in \ell(v) \mid L(v, t) \sqsubseteq \mathsf{c}\}$. The subscript L is omitted if clear from the context.

Definition 7 (Labelled Interpolation System for Resolution). *Let L be a locality preserving labelling function for an (A, B)-refutation R. The labelled interpolation system $\mathsf{Itp}(L)$ maps vertices in R to partial interpolants as defined in Figure 2.*

For an initial vertex v with $\ell(v) = C$

(A-clause) $\dfrac{}{C \quad [C\restriction_{\mathsf{b}}]}$ if $C \in A$ (B-clause) $\dfrac{}{C \quad [\neg(C\restriction_{\mathsf{a}})]}$ if $C \in B$

For an internal vertex v with $piv(v) = x$, $\ell(v^+) = C_1 \vee x$ and $\ell(v^-) = C_2 \vee \overline{x}$

$$\frac{C_1 \vee x \quad [I_1] \qquad C_2 \vee \overline{x} \quad [I_2]}{C_1 \vee C_2 \quad [I_3]}$$

(A-Res) if $L(v^+, x) \sqcup L(v^-, \overline{x}) = \mathsf{a}$, $I_3 \stackrel{\text{def}}{=} I_1 \vee I_2$
(AB-Res) if $L(v^+, x) \sqcup L(v^-, \overline{x}) = \mathsf{ab}$, $I_3 \stackrel{\text{def}}{=} (x \vee I_1) \wedge (\overline{x} \vee I_2)$
(B-Res) if $L(v^+, x) \sqcup L(v^-, \overline{x}) = \mathsf{b}$, $I_3 \stackrel{\text{def}}{=} I_1 \wedge I_2$

Fig. 2. Labelled interpolation systems

Itp yields interpolants of a highly redundant propositional structure. The structural redundancy is typically reduced in a subsequent step [8]. Therefore, we resort to the number of variables as a measure of interpolant size. Labelled interpolation systems support the elimination of *nonessential* (or *peripheral* [29]) variables from interpolants [10], as stated by the following lemma.

Lemma 2. *Let L and L' be locality preserving labelling functions for an (A, B)-refutation R, where $L(v, t) = \mathsf{a}$ if $\ell_R(v) \in A$ and $L(v, t) = \mathsf{b}$ if $\ell_R(v) \in B$ for all initial vertices of R. Then* $\mathrm{Var}(\mathsf{Itp}(L)(v)) \subseteq \mathrm{Var}(\mathsf{Itp}(L')(v))$ *for all $v \in V_R$.*

Example 2. Assume that the left refutation in Figure 1(b) is an (A, B)-refutation with $(\overline{x}_1\, x_2\, x_3), (\overline{x}_1\, \overline{x}_2\, \overline{x}_3), (x_1) \in A$ and $(x_2), (\overline{x}_1\, \overline{x}_2) \in B$, and let L be the labelling function from Lemma 2. $\mathsf{Itp}(L, R)(v) = \mathsf{F}$ for all initial vertices v in A and $\mathsf{Itp}(L, R)(v) = \mathsf{T}$ for all remaining initial vertices. The internal vertices are annotated as follows:

$$\overline{x}_1\, \overline{x}_3\, [\underbrace{(x_2 \vee \mathsf{T}) \wedge (\overline{x}_2 \vee \mathsf{F})}_{\overline{x}_2}], \quad \overline{x}_1\, x_2\, [\overline{x}_2 \vee \mathsf{F}], \quad \overline{x}_2\, [\underbrace{(x_1 \vee \mathsf{F}) \wedge (\overline{x}_1 \vee \mathsf{T})}_{x_1}]$$
$$\overline{x}_1\, [\underbrace{(x_2 \vee \overline{x}_2) \wedge (\overline{x}_2 \vee x_1)}_{x_1 \vee \overline{x}_2}], \ \square\, [(x_1 \vee \overline{x}_2) \vee \mathsf{F}]$$

Accordingly, $\mathsf{Itp}(L, R)(\mathsf{s}_R) = x_1 \vee \overline{x}_2$. For the same partition (A, B) and the right refutation P in Figure 1(b) we obtain $\mathsf{Itp}(L, P)(\mathsf{s}_P) = x_1$. $\lhd$

According to Lemma 2, the set $\mathrm{Var}(\mathsf{Itp}(L, R)(v))$ in Example 2 cannot be reduced any further by mutating L. The proof transformation in Figure 1(b), however, results in an interpolant with fewer variables.

We present a proof transformation technique aimed at reducing proof size in §3. In §4, we show that smaller proofs do not always yield interpolants with fewer variables, and specialise our reduction technique to eliminate variables (and Boolean connectives) from interpolants.

3 Proof Reduction via Subsumption

Example 1 in §2.1 demonstrates that the size of proofs can be reduced by means of clause subsumption. In general, let R be a resolution proof with vertices $u_i, v_i \in V_R$ such that $\ell_R(u_i) \subseteq \ell_R(v_i)$ for $1 \leq i \leq k$. Then the sub-proofs of R rooted at v_i can be pruned by means of substitution (see Def. 2) if no v_i is an ancestor of a u_j for $1 \leq i, j \leq k$ (cf. Lemma 1). The following example shows that the requirement $\ell_R(u_i) \subseteq \ell_R(v_i)$ is sufficient but not necessary.

Example 3. Let R be a refutation resembling the proof on the left of Figure 1(b) except that we replace the clause $\overline{x}_1\,\overline{x}_2\,\overline{x}_3$ with $\overline{x}_2\,\overline{x}_3$. In this setting, Lemma 1 does not justify the substitution proposed in Example 1 anymore, since $\overline{x}_1\,\overline{x}_2 \not\subseteq \overline{x}_2\,\overline{x}_3$. The substitution of $\overline{x}_2\,\overline{x}_3$ with $\overline{x}_1\,\overline{x}_2$ followed by ReconstructProof, however, still results in the proof on the right of Figure 1(b). The substitution is still valid because $\overline{x}_1$ is eliminated along all paths from $\overline{x}_1\,\overline{x}_2$ to $\square$ in the resulting graph. Intuitively, this situation arises since $\overline{x}_1$ is a *merge-literal* [2] of the resolution $\mathrm{Res}(\overline{x}_1\,x_2\,x_3, \overline{x}_1\,\overline{x}_3, x_3)$ in the original proof in Example 1. ◁

The set of literals eliminated along all paths from $v \in V_R$ to $\mathtt{s}_R$ can be defined as the *meet-over-all-paths* in the terminology of data-flow analysis:

$$\mathrm{rlit}(v,w) = t \text{ s.t. } t \in \ell(v), \mathrm{var}(t) = piv(w), \exists u \neq w\,.\,(u,w) \in E \wedge \mathrm{rlit}(u,w) = \bar{t}$$
$$\sigma(v) = \begin{cases} \emptyset & \text{if } v = \mathtt{s}_R \\ \bigcap_{(v,w)\in E} (\sigma(w) \cup \{\mathrm{rlit}(v,w)\}) & \text{otherwise} \end{cases} \tag{1}$$

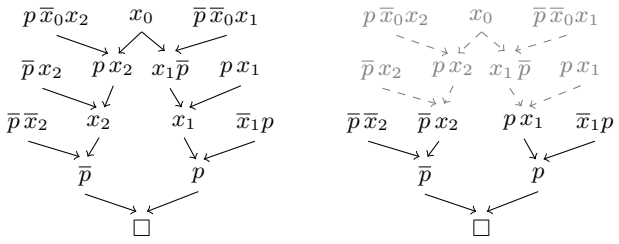
A solution to the data-flow equation 1 can be computed in linear time since the graph R is acyclic. Our definition of σ resembles the *safe literals* [13] and *expansion set* [16]. Unlike Gupta in [16] we do not rule out literals of opposing phase in $\sigma(v)$.

Given a resolution proof R and a solution of σ_R of Equation 1 for R, we call $\ell_R(v) \cup \sigma_R(v)$ the *augmented clause* of $v \in V_R$. The following theorem generalises Lemma 1 to use subsumption of augmented clauses.

Theorem 1. *Let R be a resolution proof, let σ_R be a solution of Equation 1 for R, and let $\pi = \{v_1 \mapsto u_1, \ldots, v_k \mapsto u_k\}$ be a mapping such that for all $1 \leq i \leq j \leq k$ it holds that a) no vertex v_i is an ancestor of u_j, and b) if v_j is an ancestor of u_i then $\sigma_R(u_i) \subseteq \sigma_R(v_i)$. If $\ell_R(u_i) \subseteq (\ell_R(v_i) \cup \sigma_R(v_i))$ for $1 \leq i \leq k$, then applying* ReconstructProof *to $R[v_1 \leftarrow u_1] \ldots [v_k \leftarrow u_k]$ yields a proof P with sink $\mathtt{s}_P$ such that $\ell_P(\mathtt{s}_P) \subseteq \ell_R(\mathtt{s}_R)$.*

The proof is led by nested structural induction on the number of substitutions and the number of ancestors of $\mathtt{s}_R$. The core insight is that for every sub-proof of R rooted at $\mathtt{s}_R$, ReconstructProof yields a proof P with sink $\mathtt{s}_P$ such that $\ell_P(\mathtt{s}_P) \subseteq (\ell_R(\mathtt{s}_R) \cup \sigma_R(\mathtt{s}_R))$.

The restrictions on the substitutions π in Theorem 1 are much weaker than in Lemma 1 (which corresponds to [11, Lemma 2]). Theorem 1 as well as Lemma 1 allow overlapping proofs in the range of π. In addition, Theorem 1 allows the substitution of vertices that are ancestors of preceding substitutions, and introduces



(a) A redundant proof (b) After substitution

Fig. 3. Subsumption for elimination of redundant resolution steps

a more general notion of subsumption by considering augmented clauses. In the following section, we show that Theorem 1 justifies the redundancy elimination algorithms presented in [3, 13, 16].

3.1 Eliminating Redundant Resolution Steps

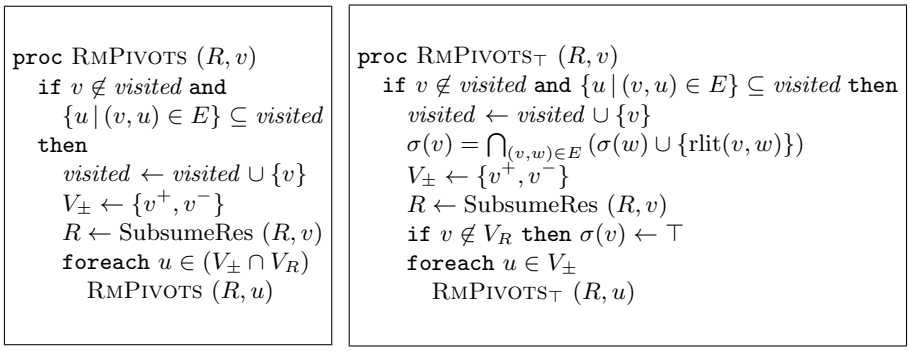
In the published version of his 1966 talk at a Leningrad seminar, Grigory Tseitin introduced the notion of regular proofs [30]. A resolution proof R is regular if, along any path from an initial vertex to the sink $\mathtt{s}_R$, every pivot occurs at most once. If proofs are represented as trees rather than directed acyclic graphs, then refutations of minimal size are always regular [31, Lemma 5.1]. Consequently, pivots that repeatedly occur along a path in tree-shaped proofs are redundant. Bar-Ilan et al. [3] introduce an algorithm (RmPivots) which eliminates such redundant resolution steps in the tree-shaped parts of a proof.

Fontaine et al. [13] generalise this algorithm to directed acyclic graphs considering all paths from a given vertex to the sink.[4] To this end, they introduce the notion of a *safe literals*, which resembles our definition of σ in Equation 1. The following example illustrates the algorithm from [13] on a redundant proof and shows that the resulting reduction is justified by Theorem 1.

Example 4. Consider the proof in Figure 3(a). Let v_1 and v_2 be the vertex for which $\ell(v_1) = x_1$ and $\ell(v_2) = x_2$, respectively. Then $\sigma(v_1) = \{p, x_1\}$ and $\sigma(v_2) = \{\overline{p}, x_2\}$, and the algorithm from [13] prunes the sub-proofs for $x_1\,\overline{p}$ and $p\,x_2$.

Now let v_3 and v_4 be the vertices such that $\ell(v_3) = p\,x_1$ and $\ell(v_4) = \overline{p}\,x_2$. Since $\sigma(v_1) = \{p, x_1\}$ and $\sigma(v_2) = \{\overline{p}, x_2\}$, we may perform the transformation $R[v_1 \leftarrow v_3][v_2 \leftarrow v_4]$ by Theorem 1. This transformation corresponds to pruning the sub-proofs as described above. Figure 3(b) shows the corresponding proof returned by ReconstructProof. ◁

[4] The resulting proofs are not necessarily regular. This is not a shortcoming of the algorithm, as minimal refutations are in general not regular [1].



(a) RmPivots (b) Optimised variant of RmPivots

Fig. 4. Removing redundant resolutions

In the following, we provide a subsumption-based formalisation of the redundancy elimination algorithm RmPivots which relies on σ to identify redundant resolution steps.

Proposition 1. *If* $piv(v) \in \sigma(v)$ *then* $\ell(v^+) \subseteq (\ell(v) \cup \sigma(v))$. *If* $\overline{piv(v)} \in \sigma(v)$ *then* $\ell(v^-) \subseteq (\ell(v) \cup \sigma(v))$.

Based on Proposition 1, the following proof transformation eliminates redundant resolution steps.

Definition 8. *Let* R *be the resolution proof* $(V_R, E_R, piv_R, \ell_R, \mathtt{s}_R)$, *and let* σ_R *be a solution of Equation 1 for* R. *We define the following transformation:*

$$\text{SubsumeRes}(R, v) \stackrel{\text{def}}{=} \begin{cases} R[v \leftarrow v^+] \text{ if } piv(v) \in \sigma_R(v) \\ R[v \leftarrow v^-] \text{ if } \overline{piv(v)} \in \sigma_R(v) \end{cases}$$

The procedure RmPivots in Figure 4(a) performs a pre-order traversal of the proof (starting at the root), which guarantees that the order of substitutions performed by SubsumeRes$(R, \mathtt{s}_R)$ satisfies condition *a)* in Theorem 1. The fact that $\sigma(v^+) = \sigma(v) \cup \{piv(v)\}$ and $\sigma(v^-) = \sigma(v) \cup \{\overline{piv(v)}\}$ in combination with the conditions of SubsumeRes in Definition 8 ($piv(v) \in \sigma(v)$ and $\overline{piv(v)} \in \sigma(v)$, respectively) establishes condition *b)*. Proposition 1 guarantees that $\ell(u_i) \subseteq (\ell(v_i) \cup \sigma(v_i))$. Therefore, applying RmPivots followed by ReconstructProof yields a valid refutation.

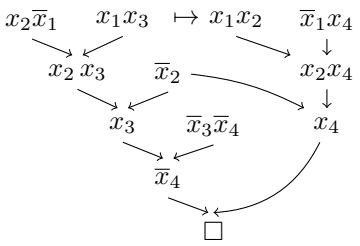
Optimisations. The definition of σ is unnecessarily restrictive in the context of RmPivots. Observe that $\sigma(v)$ is propagated even if RmPivots prunes the node v. The constraints propagated along pruned paths may result in the unnecessary exclusion of literals from ancestors of v. We amend this by setting $\sigma(v)$ to the top element $\top$ of the power set lattice if a vertex v is pruned. Figure 4(b) shows

the optimised version of RmPivots, which intertwines the computation of σ and RmPivots.

3.2 Limiting the Number of Candidates for Subsumption

RmPivots (as introduced in §3.1) only considers a subset of all feasible subsumptions. For the proof in Example 1 in §2.1, for instance, RmPivots substitutes x_2 for $\overline{x}_1\overline{x}_3$ (resulting in a different proof than the substitution suggested in Example 1). The algorithm RecycleUnits [3], on the other hand, only considers unit clauses and would substitute $\overline{x}_2$ for $\overline{x}_1\overline{x}_2\overline{x}_3$. However, RmPivots and RecycleUnits may miss valid subsumptions.

Example 5. Consider the refutation to the right. Note that no pivots are eliminated more than once along any of the paths, and none of the unit clauses are valid candidates for substitutions, since their vertices violate the ancestor requirement of Definition 2. Let v be the vertex with $\ell(v) = x_1x_3$. Since $\sigma(v) = \{x_1, x_2, x_3, \overline{x}_4\}$, v is subsumed by x_1x_2 (as indicated by $\mapsto$ in the figure). $\lhd$



The computational cost for checking all pairs of clauses satisfying the ancestor requirement in Definition 2 for subsumption is substantial. In the following, we derive a lemma that allows us to reduce the number of subsumption checks.

Proposition 2. *If v_i dominates v_j then the following subset relations hold:*

$$a) \quad (\ell(v_j) \setminus \ell(v_i)) \subseteq \sigma(v_j) \qquad \text{and} \qquad b) \quad \sigma(v_i) \subseteq \sigma(v_j)$$

Corollary 1. *If R is a refutation, then $\ell(v) \subseteq \sigma(v)$ for all $v \in V_R$ that are ancestors of $\mathtt{s}_R$.*

Corollary 2. *Let R be a resolution refutation, and let $u_i, v_i \in V_R$ be such that $\ell(u_i) \subseteq (\ell(v_i) \cup \sigma(v_i))$. Then $\ell(u_i) \subseteq \sigma(v_j)$ for any $v_j \in V_R$ dominated by v_i.*

This is a simple consequence of Proposition 2b and Corollary 1. The following lemma allows us to exclude vertices that dominate a path segment of vertices with out-degree one from our search for subsumed vertices.

Lemma 3. *Let $v_j \to v_{j+1} \to \ldots \to v_k$ be a path in a refutation R such that all vertices v_i have out-degree 1 and $\mathrm{rlit}(v_i, v_{i+1}) \notin \sigma(v_k)$ (where $j \leq i < k$). Further, let u_k be such that $\ell(u_k) \subseteq (\ell(v_k) \cup \sigma(v_k))$ and v_j is not an ancestor of u_k. Then applying* ReconstructProof *to $R[v_k \leftarrow u_i]$ or $R[v_j \leftarrow u_i]$ yields the same refutation.*

```
if σ(v) ≠ ⊤ ∧ (V± = ∅ ∨ ∃u ∈ V± . |{w|(u, w) ∈ E}| > 1) then
  pick u ∈ {w | ℓ(w) ⊆ σ(v) ∧ v ↦ w satisfies Theorem 1}
  R ← R[v ← u]
```

Fig. 5. Subsumption-based substitution of vertices

Proof: We consider only the case that v_k is an ancestor of $\mathtt{s}_R$, since v_j and v_k are otherwise not visited by ReconstructProof. Since R is a refutation, $\ell(u_k) \subseteq \sigma(v_k)$ (Corollary 1), and therefore $\ell(u_k) \subseteq \sigma(v_j)$ (Corollary 2). Since $\mathrm{rlit}(v_{i-1}, v_i) \notin \sigma(v_k)$ for $j < i \leq k$ and $\ell(u_k) \subseteq \sigma(v_k)$, we have $\mathrm{rlit}(v_{i-1}, v_i) \notin \ell(u_k)$ and $\mathrm{rlit}(w, v_i) \in \ell(w)$ for $w \neq v_{i-1}$ and $(w, v_i) \in E$. Therefore, RestoreRes propagates vertex u_k until v_k is reached (cf. Definition 3). ∎

We claim that the restriction in Lemma 3 that v_j may not be an ancestor of u_k does not exclude viable candidates for subsumption: Every $\ell(v_i)$ $(j \leq i < k)$ contains a literal $\mathrm{rlit}(v_i, v_{i+1}) \notin \sigma(v_k)$ and therefore $\ell(v_i) \not\subseteq \sigma(v_k)$.

RmPivots establishes the condition $\mathrm{rlit}(v_i, v_{i+1}) \notin \sigma(v_k)$ $(j \leq i < k)$ on paths as defined in Lemma 3. Consequently, we only need to search for clauses subsuming vertices with either no parent or a parent with out-degree greater than one (i.e., meets for σ in Equation 1). The corresponding pseudo-code is shown in Figure 5 and can be incorporated into RmPivots$_\top$ (Figure 4(b)) before the recursive call. We present an efficient technique to detect clauses subsuming $\sigma(v_j)$ (i.e., the second line in Figure 5) in §5.

Lemma 3 reduces the computational burden, not least because contemporary SAT solvers such as PicoSAT [5] construct resolution chains whose intermediate vertices have out-degree one.

Finally, we point out that vertices v with $\{x, \overline{x}\} \subseteq (\ell(v) \cup \sigma(v))$ can be replaced with a fresh vertex $u \notin V_R$ with $\ell(u) = (x\,\overline{x})$. However, RmPivots already guarantees that $\{x, \overline{x}\} \subseteq (\ell(v) \cup \sigma(v))$ only occurs on pruned traces.

4 Interpolant Reduction via Subsumption

It is tempting to apply the techniques of §3 with the intention to reduce interpolant size. The following example demonstrates that this approach may in fact have the opposite effect.

Example 6. Consider the (A, B)-refutation R with $(\overline{x}_0), (x_0\,\overline{x}_1), (x_1\,x_2) \in A$ and $(x_1\,\overline{x}_2), (\overline{x}_1) \in B$ on the left of Figure 6. We use a labelled interpolation system (Definition 7) with the labelling L (Definition 5) from Lemma 2. Each vertex is annotated with $\ell(v)\,[\mathsf{Itp}(L, R)(v)]$ as described in §2.2, and the label $L(t)$ of each literal $t \in \ell(v)$ is indicated using a superscript. The shared variable x_1 does not occur in $\mathsf{Itp}(L, R)(\mathtt{s})$, since the literals x_1^{a} and $\overline{x}_1^{\mathsf{a}}$ are *peripheral*.[5]

[5] Intuitively, since resolution corresponds to existential quantification and x_1 is eliminated *within* the A partition $((\exists x_1 \,.\, (x_0 \vee \overline{x}_1) \wedge (x_1 \vee x_2)) \Rightarrow (x_0 \vee x_2))$, the pivot

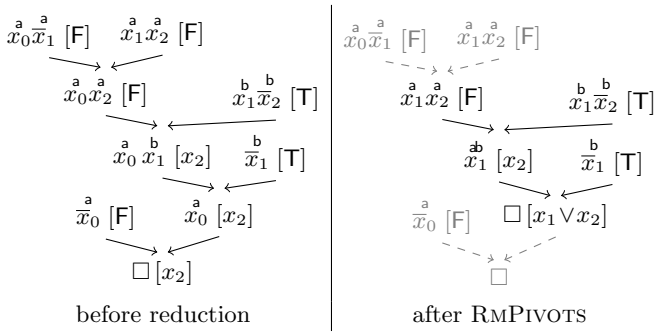


Fig. 6. Reduced proof size may increase number of variables in interpolant

We obtain the proof P on the right of Figure 6 by applying RmPivots and ReconstructProof to R. P is smaller than R, but the substitution has eliminated a peripheral resolution step and $\mathsf{Itp}(L, P)$ is forced to introduce x_1 when we resolve on x_1^{ab} and $\overline{x}_1^{\mathsf{b}}$.

Using a different interpolation technique (such as Pudlák's [26] or McMillan's [22]) or changing L does not resolve this problem. Labelled interpolation systems generalise Pudlák's and McMillan's interpolation systems [12], and according to Lemma 2, *any* labelling L would require $\mathsf{Itp}(L, P)$ to introduce x_1 at some point in P [10]. ◁

The elimination of the redundant vertex in Example 6 introduces a merge literal x_1 at the node v with $\ell(v) = x_1$ with $L(v, x_1) = \mathsf{ab}$. In order to rule out substitutions that change the label of peripheral pivots from a or b to ab, we strengthen the subsumption requirement in Theorem 1 to include labels. Given a refutation R, we compute a mapping $\varsigma : V_R \times \mathtt{Lit} \to \mathcal{S}$ in lockstep with σ:

$$\begin{aligned} \mathrm{litlab}(u, v, t) &= \begin{cases} L(v^+, \mathrm{var}(t)) \sqcup L(v^-, \overline{\mathrm{var}(t)}) & \text{if } t = \mathrm{rlit}(u, v) \\ \varsigma(v, t) & \text{otherwise} \end{cases} \\ \varsigma(v, t) &= \begin{cases} \bot & \text{if } v = \mathtt{s}_R \\ \bigsqcap_{(v,w)\in E} \mathrm{litlab}(v, w, t) & \text{otherwise} \end{cases} \end{aligned} \tag{2}$$

In analogy to Corollary 1, we observe the following relationship between the labelling L and ς:

Lemma 4. *Let R be an (A, B)-refutation, L be a locality preserving labelling function for R, and $\varsigma_{R,L}$ be a solution of Equation 2. Then $L(v, t) \sqsubseteq \varsigma_{R,L}(v, t)$ for all $v \in V_R$ and $t \in \mathtt{Lit}$.*

Proof: By structural induction. The claim holds trivially for $\mathtt{s}_R$. Let $(v, w) \in E$ and $L(v, t) \neq \bot$. If $t \neq \mathrm{rlit}(v, w)$ then $t \in \ell(w)$ and $L(v, t) \sqsubseteq L(w, t)$ by

can be "renamed" and treated as a local variable. As a side-effect, fewer logical connectives are introduced (prior to structural reduction), since the rule (AB-Res) introduces two more connectives than (A-Res) or (B-Res) (see Definition 7).

Definition 5, and $L(v,t) \sqsubseteq \varsigma(w,t)$ by the induction hypothesis. If $t = \mathrm{rlit}(v,w)$ then $L(v,t) \sqsubseteq \mathrm{litlab}(v,w,t)$. Therefore, $L(v,t) \sqsubseteq \bigsqcap_{(v,w)\in E} \mathrm{litlab}(v,w,t)$. ■

Abusing our notation, we use $\sqsubseteq$ to denote the order on $\mathcal{S}$ (Definition 5) extended point-wise to the literals $\mathtt{Lit}$. In the following, we lift Theorem 1 to labelled sets of literals using the product order $\preceq$ for the Cartesian product of the power set of $\mathtt{Lit}$ and $(\mathtt{Lit} \to \mathcal{S})$, defined as follows:

$$\langle \ell(u), L(u)\rangle \preceq \langle \sigma(v), \varsigma(v)\rangle \quad \stackrel{\text{def}}{=} \quad (\ell(u) \subseteq \sigma(v)) \wedge (L(u) \sqsubseteq \varsigma(v))$$

Based on the definition of ς (Equation 2) and the order $\preceq$, Theorem 2 disallows substitutions that may introduce additional variables in an interpolant:

Theorem 2. *Let R be an (A,B)-refutation and let σ_R, ς_R be solutions of the Equations 1 and 2 for R. Let $\pi = \{v_1 \mapsto u_1, \ldots, v_k \mapsto u_k\}$ be a mapping such that for all $1 \le i \le j \le k$ it holds that a) no vertex v_i is an ancestor of u_j, and b) if v_j is an ancestor of u_i then $\langle \sigma_R(u_i), \varsigma_R(u_i)\rangle \preceq \langle \sigma_R(v_i), \varsigma_R(v_i)\rangle$. If $\langle \ell_R(u_i), L(u_i)\rangle \preceq \langle \sigma_R(v_i), \varsigma_R(v_i)\rangle$ for $1 \le i \le k$, then applying* ReconstructProof *to $R[v_1 \leftarrow u_1]\ldots[v_k \leftarrow u_k]$ yields a proof P such that* $\mathrm{Var}(\mathsf{Itp}(L,P)) \subseteq \mathrm{Var}(\mathsf{Itp}(L,R))$.

The proof is an extension of the proof of Theorem 1 to labelled clauses. For the labelling L that maps all shared literals to ab, $L(v) \sqsubseteq \varsigma(u)$ is always satisfied, allowing us to relax the labelling constraint.

In the setting of Example 6, let v be the vertex with $\ell(v) = x_0\, x_2$ and let u be the vertex with $\ell(u) = x_1\, x_2$. We obtain $\varsigma(v, x_1) = \mathsf{b}$. Accordingly, the condition $L(u) \sqsubseteq \varsigma(v)$ in Theorem 2 rules out the substitution $R[v \leftarrow u]$. The following example, however, demonstrates that this restriction is not always beneficial.

Example 7. We continue working in the setting of Example 2. Let v_1, v_2 be the vertices such that $\ell(v_1) = (\overline{x}_0 \overline{x}_2 \overline{x}_3)$ and $\ell(v_2) = \overline{x}_1$, and u_1, u_2 be the vertices with $\ell(u_1) = \overline{x}_1 \overline{x_2}$ and $\ell(u_2) = \overline{x}_2$. Recall that the substitution $R[v_1 \leftarrow u_1]$ reduced the interpolant from $x_1 \vee \overline{x}_2$ to x_1. Theorem 2 disallows $v_1 \mapsto u_1$, since $\varsigma(v_1, \overline{x}_1) = \mathsf{a}$ and $\varsigma(u_1, \overline{x}_1) = \mathsf{b}$. Detecting that it is safe to introduce x_1 at v_2 (where $\varsigma(v_2, x_1) = \mathsf{a}$) since $x_1 \in \mathrm{Var}(\mathsf{Itp}(R)(u_2))$ would require a computationally more expensive analysis. The substitution $R[v_1 \leftarrow u_2]$ is valid, however, since $\varsigma(v_1, \overline{x}_2) = \mathsf{ab}$ and $\varsigma(u_2, \overline{x}_2) = \mathsf{b}$. The corresponding interpolant is x_1. ◁

	a	b	ab
a	a	⊥	ab
b	⊥	b	ab
ab	ab	ab	ab

The conservative restrictions of Theorem 2, which enforce $\mathrm{Var}(\mathsf{Itp}(P)(v)) \subseteq \mathrm{Var}(\mathsf{Itp}(R)(v))$ for *all* $v \in V_P$, may prevent ReconstructProof from eliminating variables by pruning. One strategy to relax this restriction is to replace the meet in Equation 2 with the operation to the right. This modification effectively enables the introduction of a variable at vertex v if it is already introduced along *one* path from v to s_R. In general, the detection of variables already introduced in other parts requires a more sophisticated analysis.

synthesis	[16]	vs. T0	vs. T20	[16]	vs. T0
solved	92/133			126/133	
max size	367044			1150888	
size (%)	17.35	22.89	25.23	18.74	24.57
vars (%)	0.62	0.68	0.68	0.65	0.69
time (s)	5	5	207	18	15
mem (GB)	1.2	1.2	2.5	2.2	2.3

HWMCC	[16]	vs. T0	vs. T20	[16]	vs. T0
solved	38/131			111/131	
max size	311151			1710588	
size (%)	11.49	14.12	16.61	20.60	25.66
vars (%)	1.61	1.99	1.99	1.87	2.47
time (s)	3	3	218	30	23
mem (GB)	0.8	0.8	2.5	3.4	3.4

Table 1. We provide results for the benchmarks from synthesis and the HWMCC. We use a locality-preserving labelling function. For synthesis benchmarks, the partitions are acquired from the synthesis tool. For HWMCC benchmarks we use a random partition (A, B). We compare ALL-RMPIVOTS [16] to RMPIVOTS$_\top$ without (T0) and with (T20) a search for subsumed clauses (limited to at most 20 minutes). We chose to implement ALL-RMPIVOTS rather than the algorithm from [13] because it is not clear how pruned edges are treated. In each comparison we use the intersection of solved benchmarks (no time- or mem-out in any configuration). Max size is the size of the largest solved proof measured in vertices. Size (%) is the average reduction in proof vertices. Vars (%) is the average reduction in variables in the final interpolant (analogous to interpolant size, cf. Footnote 5). Time (s) is the average run time (without proof creation). Mem (GB) is the average memory usage after proof creation.

5 Implementation and Experiments

We implemented (in Scala) the algorithms of §3.1 and §3.2 generalised to labelled clauses as described in §4. The performance of the algorithm in §3.1 and §3.2 hinges on an efficient check for the conditions of Theorems 1 and 2 (line 2 in Figure 5):

- *Subsumption check.* To identify clauses that subsume $\sigma(v)$, we maintain a single watch literal for each clause in R. By incrementally assigning the literals in $\sigma(v)$ to F, the watch literal enables us to identify clauses $\ell(u)$ that are inconsistent with $\neg\sigma(v)$. We may terminate before all subsuming clauses are found, in which case the algorithm favours shorter clauses. By prioritising literals in $\ell(v) \subseteq \sigma(v)$, we also avoid the unnecessary introduction of merge literals. The compatibility of $L(v)$ and $\varsigma(u)$ is checked separately.
- *Ancestor check.* Our algorithm performs a pre-order traversal starting from s_R. To detect cycles, we maintain ancestor information that is restricted to initial vertices and vertices with out-degree larger than one (see Lemma 3). If a substitution $v_i \mapsto u_i$ is performed, we remove the successors of v_i from the list of watched clauses up to the point where all literals in $\ell(u_i) \setminus \ell(v_i)$ have been merged or eliminated (to avoid invalid substitutions), and mark all ancestors of u_i as tainted. We currently disallow any v_j to be an ancestor of u_i $(j \geq i)$ in our subsumption check.

We present an experimental evaluation of our algorithms in Table 1. We use benchmarks from reactive synthesis [6] obtained via [20] and single safety property examples from the 2013 Hardware Model Checking Competition (HWMCC).

We use PicoSAT [5] 957 (synthesis) and 959 (HWMCC) to obtain resolution proofs in TraceCheck format (`-t` option). We limited synthesis benchmarks to a TraceCheck file size between 100kB and 10MB (resulting in 133 benchmarks). We obtained the HWMCC proofs by unrolling until the file size grew beyond 10MB and pick the last file smaller than 10MB (resulting in 131 benchmarks). The experiments were run on an Intel Xeon E5645 2.40GHz with a 16GB JVM memory limit and a timeout of 30 minutes.

RmPivots$_\top$ provides small but consistent improvements over Gupta's algorithm [16], for proof as well as interpolant reduction. Subsumption beyond SubsumeRes yields additional proof reduction, but is significantly more expensive (in consistence with the results in [3]). Since we currently choose the first (and smallest) subsuming clause found, we believe that there is still room for improvement by adding heuristics for selecting better candidates.

6 Conclusion

We present a framework for the reduction of refutations and interpolants, generalising the proof analysis introduced in [16] to subsumption. We point out potential conflicts between the reduction of proofs and interpolants and introduce conservative criteria that prevent subsumptions that are detrimental to interpolant size. As future work, we intend to explore more sophisticated proof analyses enabling a more aggressive reduction of interpolant size.

References

1. M. Alekhnovich, J. Johannsen, T. Pitassi, and A. Urquhart. An exponential separation between regular and general resolution. In *STOC*. ACM, 2002.
2. P. B. Andrews. Resolution with merging. *J. ACM*, 15(3):367–381, 1968.
3. O. Bar-Ilan, O. Fuhrmann, S. Hoory, O. Shacham, and O. Strichman. Reducing the size of resolution proofs in linear time. *STTT*, 13(3):263–272, 2011.
4. A. Belov, I. Lynce, and J. Marques-Silva. Towards efficient mus extraction. *AI Communications*, 25(2):97–116, 2012.
5. A. Biere. PicoSAT essentials. *JSAT*, 4(2-4):75–97, 2008.
6. R. Bloem, R. Könighofer, and M. Seidl. Sat-based synthesis methods for safety specs. In K. McMillan and X. Rival, editors, *Verification, Model Checking, and Abstract Interpretation*, volume 8318 of *Lecture Notes in Computer Science*, pages 1–20. Springer Berlin Heidelberg, 2014.
7. J. Boudou and B. W. Paleo. Compression of propositional resolution proofs by lowering subproofs. In *TABLEAUX*, volume 8123 of *LNCS*. Springer, 2013.
8. G. Cabodi, C. Loiacono, and D. Vendraminetto. Optimization techniques for craig interpolant compaction in unbounded model checking. In *Design, Automation and Test in Europe*, pages 1417–1422. ACM, 2013.
9. W. Craig. Linear reasoning. A new form of the Herbrand-Gentzen theorem. *J. Symbolic Logic*, 22(3):250–268, 1957.
10. V. D'Silva. Propositional interpolation and abstract interpretation. In *European Symposium on Programming*, volume 6012 of *LNCS*. Springer, 2010.

11. V. D'Silva, D. Kroening, M. Purandare, and G. Weissenbacher. Restructuring resolution refutations for interpolation. Technical report, Oxford, October 2008.
12. V. D'Silva, D. Kroening, M. Purandare, and G. Weissenbacher. Interpolant strength. In *VMCAI*, volume 5944 of *LNCS*, pages 129–145. Springer, 2010.
13. P. Fontaine, S. Merz, and B. W. Paleo. Compression of propositional resolution proofs via partial regularization. In *CADE*, volume 6803 of *LNCS*. Springer, 2011.
14. R. Gershman, M. Koifman, and O. Strichman. Deriving small unsatisfiable cores with dominators. In *CAV*, volume 4144 of *LNCS*, pages 109–122. Springer, 2006.
15. E. Goldberg and Y. Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *Design, Automation and Test in Europe*, pages 886–891. IEEE, 2003.
16. A. Gupta. Improved single pass algorithms for resolution proof reduction. In *ATVA*, volume 7561 of *LNCS*, pages 107–121. Springer, 2012.
17. N. Halbwachs and L. Zuck, editors. *Applications of Craig Interpolants in Model Checking*, volume 3440 of *LNCS*. Springer, 2005.
18. G. Hofferek, A. Gupta, B. Könighofer, J.-H. R. Jiang, and R. Bloem. Synthesizing multiple boolean functions using interpolation on a single proof. In *Formal Methods in Computer-Aided Design*, pages 77–84. IEEE, 2013.
19. G. Huang. Constructing Craig interpolation formulas. In *Computing and Combinatorics*, volume 959 of *LNCS*, pages 181–190. Springer, 1995.
20. J.-H. R. Jiang, H.-P. Lin, and W.-L. Hung. Interpolating functions from large Boolean relations. In *ICCAD*, pages 779–784. ACM, 2009.
21. J. Krajíček. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *J. Symbolic Logic*, 62(2):457–486, 1997.
22. K. L. McMillan. Interpolation and SAT-based model checking. In *CAV*, volume 2725 of *LNCS*, pages 1–13. Springer, 2003.
23. K. L. McMillan. An interpolating theorem prover. *Theoretical Comput. Sci.*, 345(1):101–121, 2005.
24. K. L. McMillan and N. Amla. Automatic abstraction without counterexamples. In *TACAS*, volume 2619 of *LNCS*, pages 2–17. Springer, 2003.
25. A. Nadel, V. Ryvchin, and O. Strichman. Efficient MUS extraction with resolution. In *Formal Methods in Computer-Aided Design*, pages 197–200. IEEE, 2013.
26. P. Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *J. Symbolic Logic*, 62(3):981–998, 1997.
27. S. F. Rollini, L. Alt, G. Fedyukovich, A. E. J. Hyvärinen, and N. Sharygina. PeRIPLO: A framework for producing effective interpolants in SAT-based software verification. In *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 8312 of *LNCS*, pages 683–693. Springer, 2013.
28. S. F. Rollini, R. Bruttomesso, N. Sharygina, and A. Tsitovich. Resolution proof transformation for compression and interpolation. *The Computing Research Repository*, abs/1307.2028, 2013.
29. J. Simmonds, J. Davies, A. Gurfinkel, and M. Chechik. Exploiting resolution proofs to speed up LTL vacuity detection for BMC. *STTT*, 12(5):319–335, 2010.
30. G. Tseitin. On the complexity of derivation in propositional calculus. *Studies in Mathematics and Mathematical Logic*, Part II, 1970.
31. A. Urquhart. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 1(4):425–467, 1995.
32. B.-H. Wu, C.-J. Yang, C.-Y. Huang, and J.-H. Jiang. A robust functional ECO engine by SAT proof minimization and interpolation techniques. In *ICCAD*, 2010.
33. L. Zhang. On subsumption removal and on-the-fly CNF simplification. In *SAT*, volume 3569 of *LNCS*, pages 482–489. Springer, 2005.

A Proofs

Theorem 1. *Let R be a resolution proof, let σ_R be a solution of Equation 1 for R, and let $\pi = \{v_1 \mapsto u_1, \ldots, v_k \mapsto u_k\}$ be a mapping such that for all $1 \leq i \leq j \leq k$ it holds that a) no vertex v_i is an ancestor of u_j, and b) if v_j is an ancestor of u_i then $\sigma_R(u_i) \subseteq \sigma_R(v_i)$. If $\ell_R(u_i) \subseteq (\ell_R(v_i) \cup \sigma_R(v_i))$ for $1 \leq i \leq k$, then applying* ReconstructProof *to $R[v_1 \leftarrow u_1] \ldots [v_k \leftarrow u_k]$ yields a proof P with sink $\mathtt{s}_P$ such that $\ell_P(\mathtt{s}_P) \subseteq \ell_R(\mathtt{s}_R)$.*

Proof: Because of condition *a)*, $R[v_1 \leftarrow u_1] \ldots [v_k \leftarrow u_k]$ is cycle-free. Otherwise, there must be a substitution $v_j \mapsto u_j$ introducing a cycle through u_j. Since v_j is not an ancestor of u_j in R, the cycle must visit an edge from u_i to a successor of v_i introduced by the substitution $v_i \mapsto u_i$. This is impossible, since v_i is not an ancestor of u_j. Condition *b)* prevents that the substitution $v_i \mapsto u_i$ introduces a path from v_j through a successor of v_i to $\mathtt{s}_R$ along which not all literals in $\sigma(v_j)$ are eliminated.

The core of the proof is led by nested structural induction on the number of substitutions and the number of ancestors of $\mathtt{s}_R$:

Outer base case ($\pi = \emptyset$). Applying ReconstructProof to R trivially results in a proof P satisfying that $\ell_P(\mathtt{s}_P) \subseteq (\ell_R(\mathtt{s}_R) \cup \sigma_R(\mathtt{s}_R))$.

Outer induction step. The outer induction hypothesis is that for every vertex v in $R[v_1 \leftarrow u_1] \ldots [v_j \leftarrow u_j]$, the literals in $\sigma_R(v)$ are eliminated along every path from v to the sink, and ReconstructProof yields a proof P with $\ell_P(\mathtt{s}_P) \subseteq (\ell_R(\mathtt{s}_R) \cup \sigma_R(\mathtt{s}_R))$ if applied to $R[v_1 \leftarrow u_1] \ldots [v_j \leftarrow u_j]$.

Inner base case. Assume $\mathtt{s}_R$ has no ancestors. If $\mathtt{s}_R \neq v_{j+1}$, then $\mathtt{s}_P = \mathtt{s}_R$ and $\ell(s_P) = \ell(s_R)$. Otherwise, $\mathtt{s}_P = u_{j+1} = \pi(\mathtt{s}_R)$, where u_{j+1} is the root of a sub-proof of R such that no v_i is an ancestor of u_{j+1} for $1 \leq i \leq j+1$. Therefore, ReconstructProof leaves u_{j+1} and $\ell(u_{j+1})$ unmodified. The premise guarantees that $\ell_R(u_{i+1}) \subseteq (\ell_R(\mathtt{s}_R) \cup \sigma_R(\mathtt{s}_R))$, and therefore $\ell_R(\mathtt{s}_P) \subseteq (\ell_R(\mathtt{s}_R) \cup \sigma_R(\mathtt{s}_R))$. Condition *b)* warrants that the substitutions $\{v_i \mapsto u_i \,|\, j+1 < i \leq k\}$ remain feasible.

Inner induction step. Consider the case that $\mathtt{s}_R$ has $n+1$ ancestors. The case where $\mathtt{s}_R = v_{j+1}$ is equivalent to the base case above. Therefore, let $\mathtt{s}_R \neq v_{j+1}$, and let R^+ and R^- be the sub-proofs rooted at $\mathtt{s}_R^+$ and $\mathtt{s}_R^-$, respectively. Each parent of $\mathtt{s}_R^+$ has at most n ancestors, so by induction applying ReconstructProof to $R^+[v_1 \leftarrow u_1] \ldots [v_{j+1} \leftarrow u_{j+1}]$ yields a proof P^+ with sink $\mathtt{s}_P^+$ and $\ell_{P^+}(\mathtt{s}_P^+) \subseteq (\ell_R(\mathtt{s}_R^+) \cup \sigma(\mathtt{s}_R^+))$, and similarly for R^-. Since RestoreRes($\mathtt{s}_R$) eliminates the literals rlit($\mathtt{s}_R^+, \mathtt{s}_R$) and rlit($\mathtt{s}_R^-, \mathtt{s}_R$), applying RestoreRes to $\mathtt{s}_R$ results in a proof P satisfying $\ell_P(\mathtt{s}_P) \subseteq (\ell_R(\mathtt{s}_R) \cup \sigma(\mathtt{s}_R))$.

Finally, the fact that $\sigma(\mathtt{s}_R) = \emptyset$ establishes $\ell_P(\mathtt{s}_P) \subseteq \ell_R(\mathtt{s}_R)$. ∎

Theorem 2. *Let R be an (A, B)-refutation and let σ_R, ς_R be solutions of the Equations 1 and 2 for R. Let $\pi = \{v_1 \mapsto u_1, \ldots, v_k \mapsto u_k\}$ be a mapping such*

that for all $1 \leq i \leq j \leq k$ *it holds that a) no vertex* v_i *is an ancestor of* u_j*, and b) if* v_j *is an ancestor of* u_i *then* $\langle \sigma_R(u_i), \varsigma_R(u_i) \rangle \preceq \langle \sigma_R(v_i), \varsigma_R(v_i) \rangle$*. If* $\langle \ell_R(u_i), L(u_i) \rangle \preceq \langle \sigma_R(v_i), \varsigma_R(v_i) \rangle$ *for* $1 \leq i \leq k$*, then applying* ReconstructProof *to* $R[v_1 \leftarrow u_1] \ldots [v_k \leftarrow u_k]$ *yields a proof* P *such that* $\mathrm{Var}(\mathsf{ltp}(L, P)) \subseteq \mathrm{Var}(\mathsf{ltp}(L, R))$.

Proof: Given a sub-proof rooted at $\mathtt{s}_R$, applying ReconstructProof yields a sub-proof P such that $\ell(\mathtt{s}_P) \subseteq \sigma(\mathtt{s}_R)$ (by Corollary 1 and the induction hypothesis of the proof in Theorem 1). By lifting the proof of Theorem 1 to $\preceq$, we derive $L(\mathtt{s}_P) \sqsubseteq \varsigma(\mathtt{s}_R)$. Let $\mathtt{s}_R$ be a vertex with ancestors $\mathtt{s}_R^+$ and $\mathtt{s}_R^-$. By induction, $\langle \ell(\mathtt{s}_P^+), L(\mathtt{s}_P^+) \rangle \preceq \langle \sigma(\mathtt{s}_R^+), \varsigma(\mathtt{s}_R^+) \rangle$, and similarly for $\mathtt{s}_R^-$. Since $\varsigma(\mathtt{s}_P^+, \mathit{piv}(\mathtt{s}_P)) \sqsubseteq \mathrm{litlab}(\mathtt{s}_P^+, \mathtt{s}_P, \mathit{piv}(\mathtt{s}_P))$ and similarly for $\mathtt{s}_P^-$ and $\overline{\mathit{piv}(\mathtt{s}_P)}$ (by Equation 2), we have $(L(\mathtt{s}_P^+, \mathit{piv}(\mathtt{s}_P)) \sqcup L(\mathtt{s}_P^-, \overline{\mathit{piv}(\mathtt{s}_p)})) \sqsubseteq (L(\mathtt{s}_R^+, \mathit{piv}(\mathtt{s}_R)) \sqcup L(\mathtt{s}_R^-, \overline{\mathit{piv}(\mathtt{s}_R)}))$, and therefore $\mathrm{Var}(\mathsf{ltp}(R, L)(\mathtt{s}_P)) \subseteq \mathrm{Var}(\mathsf{ltp}(R, L)(\mathtt{s}_R))$ by Definition 7. ∎

B More Details on Related Work

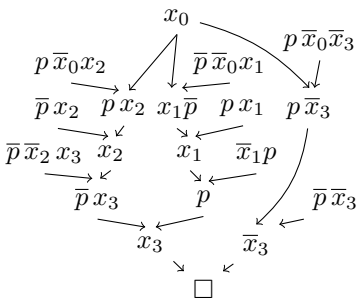
Gupta [16] introduces the notion of an *expansion set* ρ, which resembles our σ in Equation 1, but rules out literals of opposing phase:

$$\bigcap_{(v,w) \in E} (\rho(v) \cup \{\mathrm{rlit}(v, w)\}) \setminus \{\neg \mathrm{rlit}(v, w)\}\,. \tag{3}$$

According to [16], the expansion set $\rho(v)$ is the "largest set of literals" such that the "appearance of literals from $\rho(v)$ in $\ell(v)$" due to a proof transformation does not result in an invalid proof. Since the definition of ρ in Equation 3 (unlike σ) rules out literals of opposing phase, $\rho(v)$ "is a subset of the resolving literals that appear in all the paths from v to $\mathtt{s}_R$" [16]. RmPivots performs a post-order traversal of R. Whenever $\rho(v)$ contains $\mathit{piv}(v)$ ($\overline{\mathit{piv}(v)}$, respectively), Gupta's version of RmPivots prunes the sub-proof rooted at v^- (v^+, respectively).

The following example shows that $\rho(v)$ is in fact *not* the "largest set of literals" whose introduction at v preserves the validity of a proof (as claimed in [16]) and that the restrictive definition of ρ in Equation 3 is detrimental.

Example 8. Consider the extension of the refutation in Figure 3(a) to the right. Let v_1 and v_2 be the vertices for which $\ell(v_1) = p\,x_2$ and $\ell(v_2) = x_1\overline{p}$, and let v_0 be such that $\ell(v_0) = x_0$. We obtain $\rho(v_1) = \{p, x_2, x_3\}$, and $\rho(v_2) = \{\overline{p}, x_1, x_3\}$ (since ρ only propagates *last* elimination of a literal over p), and therefore $\rho(v_0) = \{x_0\}$. On the other hand, $\sigma(v_1) = \{p, \overline{p}, x_2, x_3\}$ and $\sigma(v_1) = \{p, \overline{p}, x_1, x_3\}$, and thus $\sigma(v_0) = \{p, x_0\}$. Accordingly, ρ potentially admits fewer eliminations of redundant resolution steps than σ. ◁



Thus, the algorithm in Fontaine et al. [13] and consequently our subsumption-based algorithm in Section 3.1 achieves *at least* the same reduction as Gupta's version of RmPivots.

In fact, the definition of σ is unnecessarily restrictive in the context of RmPivots. Observe that $\sigma(v_0) = \{p, x_0\}$ in the refutation in Example 8, even though RmPivots prunes all but one outgoing edge of v_0. The constraints propagated from v_1 and v_2 result in the unnecessary exclusion of $\overline{x}_3$ from $\sigma(v_0)$. We amend this by setting $\sigma(v)$ to the top element $\top$ of the power set lattice if a vertex v is pruned. Figure 4(b) shows the optimised version of RmPivots, which intertwines the computation of σ and RmPivots.

Boudou et al. [7] introduce a proof reconstruction technique called *lowering*, where resolutions are moved downwards in the proof by means of projection of literals. This technique is similar to the algorithm deployed in [11], where a similar approach is used to move resolutions on local pivots (c.f. 6) upwards in the proof. As pointed out in [7] paper, lowering can be sequentially combined with other proof reduction techniques such as our framework.

C Additional Experimental Results

synthesis	T0 vs. T0-ns		T20 vs. T20-ns	
solved	127/133		92/133	
max size	1150888		367044	
size (%)	24.74	24.76	25.23	25.25
vars (%)	0.69	0.69	0.68	0.68
time (s)	18	18	209	210
mem (GB)	2.4	2.4	2.5	2.5

HWMCC	T0 vs. T0-ns		T20 vs. T20-ns	
solved	111/131		38/131	
max size	1710588		311151	
size (%)	25.66	25.67	16.61	16.62
vars (%)	2.47	2.48	1.99	2.02
time (s)	23	23	218	215
mem (GB)	3.4	3.4	2.6	2.6

Table 2. Comparison of suppression of certain substitutions due to $\sqsubseteq$ (regular approach) and ignoring the label information (*-ns configuration).

Table 2 provides additional experimental results showing that for our benchmarks reducing the proof as much as possible yields a slight improvement of interpolant reduction as well.