Playing in the Grey Area of Proofs

Kryštof Hoder
University of Manchester
krystof.hoder@gmail.com

Laura Kovács
TU Vienna
lkovacs@complang.tuwien.ac.at

Andrei Voronkov
University of Manchester
andrei@voronkov.com

Abstract

Interpolation is an important technique in verification and static analysis of programs. In particular, interpolants extracted from proofs of various properties are used in invariant generation and bounded model checking. A number of recent papers studies interpolation in various theories and also extraction of smaller interpolants from proofs. In particular, there are several algorithms for extracting of interpolants from so-called local proofs. The main contribution of this paper is a technique of minimising interpolants based on transformations of what we call the "grey area" of local proofs. Another contribution is a technique of transforming, under certain common conditions, arbitrary proofs into local ones.

Unlike many other interpolation techniques, our technique is very general and applies to *arbitrary* theories. Our approach is implemented in the theorem prover Vampire and evaluated on a large number of benchmarks coming from first-order theorem proving and bounded model checking using logic with equality, uninterpreted functions and linear integer arithmetic. Our experiments demonstrate the power of the new techniques: for example, it is not unusual that our proof transformation gives more than a tenfold reduction in the size of interpolants.

Categories and Subject Descriptors D.2.4 [*Software Engineering*]: Formal Methods; F.3.1 [*Logic and Meanings of Programs*]: Assertions; I.2.3 [*Artificial Intelligence*]: Deduction, inference engines, resolution

General Terms Theory, Verification, Experimentation

Keywords Program verification, theorem proving, interpolation

1. Introduction

Interpolants extracted from proofs have several applications in verification and static analysis, see e.g. [3, 6, 12, 15, 21]. Although interpolants are guaranteed to exist in some theories (for example, those having quantifier elimination), interpolants extracted from proofs turn out to be smaller and more useful than those obtained by general interpolation algorithms, see, e.g. [16]. For this reason, recent papers [5, 9, 11, 14, 18, 19, 21] consider the problem of obtaining small interpolants for various theories.

In this paper we consider two related problems: extracting interpolants from proofs and minimising such interpolants. Papers [18, 20] define algorithms for extracting interpolants from so-called *local proofs*. Roughly, in local proofs some symbols are colored in the red or blue colors and others are uncolored. Uncolored symbols are said to be grey. A local proof cannot contain an inference that uses both red and blue symbols. In other words, colors cannot be mixed within the same inference.

However, building local proofs may require substantial changes to a first-order theorem prover or an SMT solver. In addition, local proofs do not necessarily exist. One of the contributions of this paper is a technique for changing proofs into local ones under some conditions. The ideas of this technique can be traced to an observation made in [17, 18] that existential quantification of constants results in an interpolant. We prove a simple result showing that this technique is correct and can be applied to translate non-local proofs with colored constants into local proofs.

When we already have a local proof, one can extract an interpolant from it. This interpolant is a boolean combination of (some) formulas occurring in the proof, if one uses the algorithm of [18]. More exactly, the interpolant is obtained as a boolean combination of conclusions of some *symbol-eliminating inferences*: an inference having at least one colored premise and a grey conclusion. The interpolation extraction theorem of [18] is not restricted to any particular theory. Essentially the only condition on proofs is *inference soundness*, that is, the conclusion of any inference is a logical consequence of its premises. This generality gives one a lot of freedom since one does not have to follow rules of any specific calculus (such as resolution and superposition) in building local proofs.

In this paper, we exploit the generality of [18] by considering proof transformations that preserve both inference soundness and locality. It is interesting that such transformations can drastically change the shape and the size of the extracted interpolant. The transformations we consider are always applied to grey formulas in the proof, which inspired the title of this paper.

While the class transformations we consider (cutting off a grey formula) obviously preserve inference soundness, they can violate locality. To preserve locality, we create a SAT problem whose solutions encode all local proofs obtained by a sequence of cut-offs. Further, we create a linear expression over the variables of the SAT problem that expresses some numeric characteristics of the interpolant, for example, the number of different atoms in it. Thus, we are interested in the solutions of the problem that minimise the linear expression: any such solution can be used to build a proof giving a smaller (in some sense) interpolant. These solutions can be found using an SMT solver or a pseudo-boolean optimisation tool.

The main contributions of our paper are summarised below.

▷ We present a new method of producing smaller interpolants from local proofs. The methods is based on transformation of the "grey area" of proofs. It uses the idea that proof locality can be expressed by a set of propositional formulas whose models represent all local proofs obtained by such transformations (Sections 5.1-5.2).

POPL'12, January 25–27, 2012, Philadelphia, PA, USA.
Copyright © 2012 ACM 978-1-4503-1083-3/12/01...$10.00

▷ We present a method for changing proofs into local ones. This method is applicable to all proofs in which all colored symbols are uninterpreted constants (Section 4).

▷ We define a transformation of interpolant minimisation problems into the problem of solving pseudo-boolean constraints (Section 5.4). Minimality is defined with respect to various measures of the size of interpolants.

▷ We implemented our minimisation algorithm in the Vampire theorem prover [24]. It uses the Yices SMT solver [10] for solving pseudo-boolean constraints (Section 6.1). As Vampire cannot yet efficiently handle the combination of various theories, we generate proofs over SMT problems using Z3 [8].

▷ We show experimentally that our method improves [18] by generating considerably better/smaller interpolants in the size, the total weight and the number of quantifiers (Section 6).

The rest of this paper is structured as follows. Section 2 overviews relevant definitions and properties of first-order logic and interpolation. In Section 3 the notion of colored and local proofs are introduced. Our result on translating non-local proofs into local ones is formulated in Section 4. Section 5 details our approach to minimising interpolants. We present experimental results in Section 6 and overview related work in Section 7. Section 8 concludes the paper.

2. Interpolation

We consider the standard first-order predicate logic with equality. We allow all standard boolean connectives and quantifiers in the language. We assume that the language contains the logical constants $\top$ for always true and $\bot$ for always false formulas.

Throughout this paper, we denote formulas by A, B, C, D, G, R, terms by r, s, t, variables by x, y, z, constants by a, b, c and function symbols by f, g, possibly with indices. Let A be a formula with free variables $\bar{x}$, then $\forall A$ (respectively, $\exists A$) denotes the formula $(\forall \bar{x})A$ (respectively, $(\exists \bar{x})A$). A formula is called *closed*, or a *sentence*, if it has no free variables. We call a *symbol* a predicate symbol, a function symbol or a constant. Thus, variables are not symbols. We consider equality $=$ part of the language, that is, equality is not a symbol. A formula or a term is called *ground* if it has no occurrences of variables. A formula is called *universal* (respectively, *existential)* if it has the form $(\forall \bar{x})A$ (respectively, $(\exists \bar{x})A$), where A is quantifier-free. We write $C_1, \ldots, C_n \vdash C$ to denote that the formula $C_1 \wedge \ldots \wedge C_n \rightarrow C$ is a tautology. Note that $C_1, \ldots, C_n, C$ may contain free variables.

A *signature* is any finite set of symbols. The *signature of a formula A* is the set of all symbols occurring in this formula. For example, the signature of $b = g(z)$ is $\{g, b\}$. The *language of a formula A*, denoted by $\mathcal{L}_A$, is the set of all formulas built from the symbols occurring in A, that is formulas whose signatures are subsets of the signature of A.

We recall the following theorem from [7].

THEOREM 2.1 (Craig's Interpolation Theorem). *Let A, B be closed formulas and let $A \vdash B$. Then there exists a closed formula $I \in \mathcal{L}_A \cap \mathcal{L}_B$ such that $A \vdash I$ and $I \vdash B$.*

In other words, every symbol occurring in I also occurs in both A and B. Every formula I satisfying this theorem will be called an *interpolant* of A and B.

We call a *theory* any set of closed formulas. If T is a theory, we write $C_1, \ldots, C_n \vdash_T C$ to denote that the formula $C_1 \wedge \ldots \wedge C_1 \rightarrow C$ holds in all models of T. In fact, our notion of theory corresponds to the notion of *axiomatisable theory* in logic. When we work with a theory T, we call symbols occurring in T *interpreted* while all other symbols *uninterpreted*.

As proved in [18], Craig's interpolation also holds for theories in the following sense:

THEOREM 2.2. *Let A, B be formulas and let $A \vdash_T B$. Then there exists a formula I such that*

1. $A \vdash_T I$ and $I \vdash B$;
2. every uninterpreted symbol of I occurs both in A and B;
3. every interpreted symbol of I occurs in B.

Likewise, there exists a formula I such that

1. $A \vdash I$ and $I \vdash_T B$;
2. every uninterpreted symbol of I occurs both in A and B;
3. every interpreted symbol of I occurs in A.

The proof of this theorem in [18] uses compactness, which is guaranteed when T is axiomatisable.

In the sequel we will sometimes be interested in the interpolation property with respect to a given theory T. We will use $\vdash_T$ instead of $\vdash$ and relativise all definitions to T. To be precise, we call an *interpolant* of A and B any formula I such that $A \vdash_T I$, $I \vdash_T B$, and every uninterpreted symbol of I occurs both in A and B.

If E is a set of expressions (for example, formulas) and constants $c_1, \ldots, c_n$ do not occur in E, then we say that $c_1, \ldots, c_n$ are *fresh* for E. We will less formally simply say *fresh constants* when E is the set of all expressions considered in the current context.

We call a *reverse interpolant* of A and B any formula I such that $A \vdash_T I$, $I, B \vdash_T \bot$, and every uninterpreted symbol of I occurs both in A and B.

Reverse interpolants for A and B are exactly interpolants of A and $\neg B$. Moreover, when B is closed, reverse interpolants are exactly interpolants in the sense of [20, 21]. Reverse interpolants are convenient when we use a refutation-based inference system, such as resolution, for finding a proof of $A \rightarrow B$ that can give us an interpolant: in this case one can search for a refutation from the set of formulas $A, \neg B$ instead.

3. Local Proofs

In this section we recall some terminology related to inference systems. Inference systems are commonly used in the theory of resolution and superposition [1, 22]; however we do not restrict ourselves to the superposition calculus. The material of this section is based on [18], adapting the terminology of [18] to our setting.

We also introduce the notion of *local proofs* and recall results on extracting interpolants from local proofs as proved in [18].

DEFINITION 3.1. An *inference rule* is an n-ary relation on formulas, where $n \geq 0$. The elements of such a relation are called *inferences* and usually written as

$$\frac{A_1 \quad \ldots \quad A_n}{A} \; .$$

The formulas $A_1, \ldots, A_n$ are called the *premises* of this inference, whereas the formula A is the *conclusion* of the inference.

An *inference system* $\mathfrak{I}$ is a set of inference rules. An *axiom* of an inference system is any conclusion of an inference with 0 premises. Any inferences with 0 premises and a conclusion A will be written without the bar line, simply as A.

A *derivation* in an inference system $\mathfrak{I}$ is a tree built from inferences in $\mathfrak{I}$. If the root of this derivation is A, then we say it is a *derivation of* A. A derivation of A is called a *proof* of A if it is finite and all leaves in the derivation are axioms. A formula A is called *provable* in $\mathfrak{I}$ if it has a proof. We say that a derivation of A is *from assumptions* $A_1, \ldots, A_m$ if the derivation is finite and every leaf in it is either an axiom or one of the formulas $A_1, \ldots, A_m$. A formula A is said to be *derivable from* assumptions $A_1, \ldots, A_m$ if there exists a derivation of A from $A_1, \ldots, A_m$. A *refutation* is a derivation of $\bot$. □

Note that a proof is a derivation from the empty set of assumptions. Any derivation from a set of assumptions S can be considered as a derivation from any larger set of assumptions $S' \supseteq S$.

Let us now fix two sentences R (red) and B (blue). In the sequel we assume R and B to be fixed and give all definitions relative to R and B. Denote by $\mathcal{L}$ the intersection of the languages of R and B, that is the set $\mathcal{L}_R \cap \mathcal{L}_B$. We call signature symbols occurring both in R and B *grey*, symbols occurring only in R *red* and symbols occurring only in B *blue*. A symbol that is either red or blue is also called *colored*. For a formula C, we say that C is *grey* if $C \in \mathcal{L}$, otherwise we say that C is *colored*. In other words, grey formulas contain only grey symbols and every colored formula contains at least one red or blue symbol. A colored formula that only contains red and grey symbols, is called a *red* formula. Similarly, a *blue* formula is a colored formula containing only blue and grey symbols. In the rest of this paper, red formulas will be denoted by R, blue formulas by B, and grey formulas by G, possibly with indices.

DEFINITION 3.2 (RB-derivation). Let us call an *RB-derivation* any derivation Π satisfying the following conditions.

(RB1) For every leaf C of Π one of the following conditions holds:
1. $R \vdash_T \forall C$ and $C \in \mathcal{L}_R$ or
2. $B \vdash_T \forall C$ and $C \in \mathcal{L}_B$.

(RB2) For every inference

$$\frac{C_1 \quad \ldots \quad C_n}{C}$$

of Π we have $\forall C_1, \ldots, \forall C_n \vdash_T \forall C$.

We will refer to property (RB2) as *soundness*. □

We will be interested in finding reverse interpolants of R and B. The case $\mathcal{L}_R \subseteq \mathcal{L}_B$ is obvious, since in this case R is a reverse interpolant of R and B. Likewise, if $\mathcal{L}_B \subseteq \mathcal{L}_R$, then $\neg B$ is a reverse interpolant of R and B. For this reason, in the sequel we assume that $\mathcal{L}_R \not\subseteq \mathcal{L}_B$ and $\mathcal{L}_B \not\subseteq \mathcal{L}_R$, that is, *both R and B contain at least one colored symbol*.

We are mostly interested in a special kind of derivation introduced in [14] and called *local* (or sometimes called *split-proofs*). The definition of a local derivation is relative to formulas R and B.

DEFINITION 3.3 (Local RB-derivation). An inference

$$\frac{C_1 \quad \ldots \quad C_n}{C}$$

in an RB-derivation is called *local* if the following two conditions hold.

(L1) Either $\{C_1, \ldots, C_n, C\} \subseteq \mathcal{L}_R$ or $\{C_1, \ldots, C_n, C\} \subseteq \mathcal{L}_B$.
(L2) If all of the formulas $C_1, \ldots, C_n$ are grey, then C is grey, too.

A derivation is called *local* if so is every inference of this derivation.
□

In other words, (L1) says that inferences cannot mix colors: no inference contains both red and blue symbols. Condition (L2) is natural (inferences should not introduce irrelevant symbols) but it is absent in other works. Condition (L2) is however essential for us since without it the proof of Theorem 3.4 does not go through [18]. Note that standard derivations produced by theorem provers often contain inferences violating (L2), especially, in instantiation rules:

$$\frac{(\forall x)A(x)}{A(r)} ,$$

where r is a red term. However, such inferences can be removed from derivations without violating (L1).

We will now formulate one of the main theorems of [18] on the extraction of interpolants from local proofs and explain the structure of interpolants obtained by the algorithm of [18].

Consider any RB-derivation Π. Note that by the soundness condition (RB2) we can replace every formula C occurring in this derivation by its universal closure $\forall C$ and obtain an RB-derivation Π' where inferences are only performed on closed formulas. We will call such derivations Π' *closed* and assume, for simplicity, that we are dealing only with closed derivations.

We call a *symbol-eliminating inference* any inference that is

1. either a grey leaf G of Π such that $R \vdash_T G$.
2. or has the form

$$\frac{A_1 \quad \cdots \quad A_n}{G} ,$$

such that G is grey and and at least one of the formulas $A_1, \ldots, A_n$ is colored.

Any such inference "eliminates" at least one colored symbol. One could also call such inferences *color-eliminating*. The following theorem is proved in [18]:

THEOREM 3.4. *Let Π be a closed local RB-refutation. Then one can extract from Π a reverse interpolant I of R and B. This reverse interpolant is a boolean combination of conclusions of symbol-eliminating inferences of Π.* □

The proof of Theorem 3.4 in [18] gives an algorithm for extracting an interpolant from a refutation.

By a close inspection of the algorithm of [18], we noted that not all conclusions of symbol-eliminating inferences occur in the extracted interpolant. To characterise the set of all formulas occurring in the interpolant, in this paper we introduce a new notion, called the *digest* of a refutation, as given below.

DEFINITION 3.5 (Digest). Consider any conclusion G of a symbol-eliminating inference.

If the inference eliminates a red symbol, then it has the form:

$$\frac{\cdots \quad R_0 \quad \cdots}{G}$$

Consider the path from G to the bottom formula of the refutation:

$$\begin{array}{c} \underline{\cdots \quad R_0 \quad \cdots} \\ G \\ \vdots \\ \bot \end{array}$$

We say that G belongs to the *digest* of the refutation if either all formulas on the path are grey *or* the first (closest to G) colored formula on the path is blue.

Likewise, for a blue symbol eliminating inference:

$$\begin{array}{c} \underline{\cdots \quad B_0 \quad \cdots} \\ G \\ \vdots \\ \bot \end{array}$$

G belongs to the *digest* of the refutation if at least one formula on the path is colored *and* the first (closest to G) colored formula on the path is red. □

Note the slight asymmetry in Definition 3.5 between red and blue symbol eliminating inferences, which is due to the interpolant generation algorithm of [18]. Using the notion of digest, we can now refine Theorem 3.4 as follows:

THEOREM 3.6. *Let Π be a closed local RB-refutation. Then one can extract from Π a reverse interpolant I of R and B. This reverse interpolant is a boolean combination of the formulas in the digest of Π.* $\square$

In what follows we will refer to the interpolant obtained from a refutation as described in Theorem 3.6 as the *interpolant extracted from* Π.

4. Proof Localisation

Extracting interpolants from proof requires a special interpolating prover, or a prover producing local proofs. While, as reported in [13], the theorem prover Vampire can search for local proofs only and hence the algorithm of [18] can be used in first-order resolution proofs, most provers and SMT solvers do not necessarily generate local proofs.

One of the main motivations of this paper was to check how our minimisation technique works on real-life examples taken from static analysis of software. Although such benchmarks exist, they can only be solved using an SMT solver, which in general produces non-local proofs.

In is interesting that in real-life examples, especially those taken from bounded model checking, all the colored symbols are normally uninterpreted constants representing state variables from intermediate states. In this section we show that for such examples one can transform arbitrary proofs into local ones, at the cost of quantifying some formulas in the proof. This idea has already appeared in [17, 18], see Lemma 4.1 below.

The downside of this approach is that a ground refutation can become a non-ground one, thus, the extracted interpolant may contain quantifiers. Once we have a local proof, the number of such quantifiers can be reduced using the technique of Section 5 (line 18 of Algorithm 1).

LEMMA 4.1. *[17, 18] Consider two formulas $A_1(a)$ and A_2 such that $A_1(a) \vdash_T A_2$ and a is an uninterpreted constant not occurring in A_2. Then, $A_1(a) \vdash (\exists x)A_1(x)$ and $(\exists x)A_1(x) \vdash A_2$.*

This lemma can be used to localise non-local derivations by quantifying away colored constants that result in mixing colors.

THEOREM 4.2. *Given two formulas R and B such that $R \rightarrow B$ and all the colored symbols of R and B are uninterpreted constant symbols. Then any proof Π of $R \rightarrow B$ can be translated into a local proof Π_l.*

PROOF. Let us take a non-local refutation Π of $R \rightarrow B$. This means, that Π contains at least one inference that violates conditions (L1)-(L2) of Definition 3.3. The proof is by induction on Π. We will eliminate all color conflicts one by one, starting from the bottom of the proof. Thus, for every conflicting inference, we can assume that the derivation below it is already local. In particular, the conclusion of the violating inference cannot mix colors. Consider the case when the conclusion is blue (other cases are similar). Then the violating inference has the form

$$\frac{R_1 \quad \cdots \quad R_n \quad A_1 \quad \cdots \quad A_m}{A},$$

where $A, A_1, \ldots, A_m$ are either grey or blue and $R_1, \ldots, R_n$ are red. Let $r_1, \ldots, r_k$ be all the red constants occurring in this inference and formulas R'_i are obtained from R_i by replacing $r_1, \ldots, r_k$ by fresh variables $x_1, \ldots, x_k$. Note that all of the R'_i are either grey or blue. The above non-local inference can then be replaced by:

$$\frac{(\exists x_1 \ldots x_k)(R'_1 \wedge \ldots \wedge R'_n) \quad A_1 \quad \cdots \quad A_m}{A},$$

This inference does not contain the red color, and we are done. Note that the premises of the formula $(\exists x_1 \ldots x_k)(R'_1 \wedge \ldots \wedge R'_n)$ are given by the union of the premises of $R_1, \ldots, R_n$. The correctness of the transformation is guaranteed by Lemma 4.1.

The above transformation can also be applied on inferences where a premise contains both a red and a blue symbol. The non-local inference is replaced by a local inference at the cost of using existential quantifiers over the premise with colored symbols.

$\square$

This theorem gives us an algorithm for changing any non-local refutation to a local one, provided that the condition on colored symbols is satisfied.

Figure 1 illustrates how the non-local proof given in Figure 1(a) is translated into the local proof listed in Figure 1(b).

5. Playing in the Grey Area

This section presents the main idea of this paper. It is based on the following observation. One can change, sometimes considerably, the grey areas (that is, areas consisting of grey formulas) of the proof without violating locality. In addition, such proof transformations can change the extracted interpolant.

We will only consider one kind of proof transformations, called here *grey slicing*. Other proof transformations can be proposed as well, but are beyond the scope of this paper.

DEFINITION 5.1 (Grey slicing). Consider any derivation Π containing a subderivation Δ of the form

$$\frac{A_1 \quad \cdots \quad A_n \quad \dfrac{A_{n+1} \quad \cdots \quad A_m}{A}}{A_0}, \tag{1}$$

where $n \geq 0$.

We say that a derivation Π' is obtained from Π by *slicing off* A *in* Δ (or simply, *slicing off* A) if Π' is obtained from Π by replacing the subderivation Δ by

$$\frac{A_1 \quad \cdots \quad A_n \quad A_{n+1} \quad \cdots \quad A_m}{A_0} \tag{2}$$

When A is a grey formula, we will refer to this transformation as *grey slicing*. $\square$

Apparently, grey slicing preserves properties (RB1)-(RB2) of Definition 3.2, so it transforms an RB-derivation into an RB-derivation. It is also easy to see that grey slicing can violate the locality conditions (L1) of Definition 3.3. For example, slicing off G_1 in

$$\frac{B_0 \quad \dfrac{R_0}{G_1}}{G_0}$$

yields a non-local derivation

$$\frac{B_0 \quad R_0}{G_0}.$$

Consider now an example showing that grey slicing transformations can change the digest, and hence the extracted interpolant.

$$\dfrac{\dfrac{(\forall x_1)(p(x_1) \vee q(x_1, r_1)) \quad \neg p(b_1)}{q(b_1, r_1)} \qquad \dfrac{(\forall x_2)(s(x_2) \vee \neg q(x_2, r_1)) \quad \neg s(b_1)}{\neg q(b_1, r_1)}}{\bot}$$

(a)

$$\dfrac{\dfrac{\dfrac{\dfrac{(\forall x_1)(p(x_1) \vee q(x_1, r_1)) \quad (\forall x_2)s(x_2) \vee \neg q(x_2, r_1)}{(\exists y)\big((\forall x_1)(p(x_1) \vee q(x_1, y)) \wedge (\forall x_2)(s(x_2) \vee \neg q(x_2, y))\big)} \quad \neg p(b_1)}{(\exists y)\big(q(b_1, y) \wedge (\forall x_2)(s(x_2) \vee \neg q(x_2, y))\big)} \quad \neg s(b_1)}{(\exists y)\big(q(b_1, y) \wedge \neg q(b_1, y)\big)}}{\bot}$$

(b)

Figure 1. Proof localisation of proof (a) into proof (b).

EXAMPLE 5.2. *Take the following refutation* Π*:*

$$\dfrac{\dfrac{\dfrac{R_3 \qquad \dfrac{\dfrac{\dfrac{R_1 \quad G_1}{G_3} \quad \dfrac{B_1 \quad G_2}{G_4}}{G_5}}{G_6}}{R_4}}{G_7}}{\bot}$$

The digest of this refutation is $\{G_4, G_7\}$ *and the extracted reverse interpolant is* $G_4 \rightarrow G_7$*. Slicing off* G_4 *in* Π *results in the refutation* Π_1*:*

$$\dfrac{\dfrac{\dfrac{R_3 \qquad \dfrac{\dfrac{\dfrac{R_1 \quad G_1}{G_3} \quad B_1 \quad G_2}{G_5}}{G_6}}{R_4}}{G_7}}{\bot}$$

with the digest $\{G_5, G_7\}$ *and the extracted reverse interpolant* $G_5 \rightarrow G_7$*. Slicing off now* G_5 *in* Π_1 *results in* Π_2*:*

$$\dfrac{\dfrac{\dfrac{R_3 \qquad \dfrac{\dfrac{R_1 \quad G_1}{G_3} \quad B_1 \quad G_2}{G_6}}{R_4}}{G_7}}{\bot}$$

with the digest $\{G_6, G_7\}$ *and the extracted reverse interpolant* $G_6 \rightarrow G_7$*. We can slice off* G_7 *in* Π_2 *and obtain the refutation:*

$$\dfrac{\dfrac{R_3 \qquad \dfrac{\dfrac{R_1 \quad G_1}{G_3} \quad B_1 \quad G_2}{G_6}}{R_4}}{\bot}$$

with the digest $\{G_6\}$*, and the reverse interpolant* $\neg G_6$*.*

However, if we slice off G_3 *in the original derivation* Π*, we obtain the refutation:*

$$\dfrac{\dfrac{\dfrac{R_3 \qquad \dfrac{\dfrac{R_1 \quad G_1 \quad \dfrac{B_1 \quad G_2}{G_4}}{G_5}}{G_6}}{R_4}}{G_7}}{\bot}$$

in which slicing off G_4 *would violate the locality of the resulting refutation.*

Example 5.2 gives us the following observations:

1. grey slicing can change the extracted interpolant, and sometimes considerably (compare $G_4 \rightarrow G_7$ and $\neg G_6$).
2. a grey slicing step can prevent other grey slicing steps, thus preventing previously possible interpolants.

The main question we are going to answer in this section is how to use grey slicing to obtain smaller, and even minimal, in some sense, interpolants. To this end we will use the following ideas. First, we will introduce a set V_Π of propositional variables expressing some properties of refutations obtained by grey slicing from a given proof Π. Next, we will define propositional formulas P_Π of the variables V_Π that express locality. Thus, every refutation obtained from Π by grey slicing is local if and only if it satisfies P_Π. This means we can use a SAT solver to "compute" all local refutations that can be obtained from Π by grey slicing. Finally, we introduce propositional formulas expressing the digest of refutations. This set of propositional formulas allows us to use an SMT solver or a pseudo-boolean optimisation tool to find refutations minimising the digest in various ways.

Let us now formalise this idea. In the rest of this section, when we speak about a formula from a derivation, we will normally mean a concrete node in the derivation containing this formula (note that a tree-like derivation may contain more than one node with the same formula). Later we will also discuss derivations in the dag form. Nonetheless, for simplicity, for the moment we prefer to deal with trees instead of dags.

The first thing to note is that every derivation is also a set of nodes occurring in it and slicing off simply removes one node from this set. This means that a sequence of slicing off transformations removes a subset of nodes. Every removed node G, at the point of removal, is replaced by a set of other nodes occurring in the derivation (namely, the premises of G at that point). Each of the nodes in this set can in turn be removed (and replaced by other nodes) etc., so eventually the place of any removed node will be taken by a set of nodes occurring in the final derivation. We will call this set a *trace* of F and define it formally below.

DEFINITION 5.3 (Trace). Let $S = \Pi_0, \dots, \Pi_k$ be a sequence of derivations such that each member in the sequence except Π_0 is obtained by slicing off a single grey node from the previous one. For every grey node G in Π_0 we define a set of formulas, call *trace of* G (with respect to S), as follows:

1. If G was never sliced off, that is, it occurs in Π_k, then $\mathit{trace}(G) \overset{\text{def}}{=} \{G\}$.
2. Suppose G was sliced off at some point, that is, G is the formula A as in Definition 5.1. Then $\mathit{trace}(G) \overset{\text{def}}{=} \mathit{trace}(A_{n+1}) \cup \ldots \cup \mathit{trace}(A_m)$. □

Denote any sequence S of slicing off transformations with the initial derivation Π and final derivation Π' by $\Pi \dashrightarrow \Pi'$. It is not hard to argue that the following lemma holds.

LEMMA 5.4. *The trace of a node does not depend on the sequence of transformations* S *but only depends on the initial and the final* *derivation in S. That is, for every two derivations of the form $\Pi \dashrightarrow \Pi'$ with the same initial derivation Π and final derivation Π', and for every grey node G in Π, the trace of G is the same in both derivations.* □

In the rest of this section we will normally assume a fixed initial derivation Π and various sequences $\Pi \dashrightarrow \Pi'$. In view of this lemma we will simply speak about the *trace of G in Π'*.

Suppose $\Pi \dashrightarrow \Pi'$ is a sequence of transformations. Let us introduce some propositions characterising the behaviour of grey nodes in Π on this sequence.

- ▷ $s(G)$: G was sliced off;
- ▷ $r(G)$: the trace of G contains a red formula;
- ▷ $b(G)$: the trace of G contains a blue formula;
- ▷ $g(G)$: the trace of G contains only grey formulas;
- ▷ $d(G)$: G belongs to the digest of Π'.

We define the set V_Π of propositional variables as consisting of all the variables $s(G), r(G), b(G), g(G), d(G)$ denoting these propositions. Later we will add to V_Π more variables.

Then, for every sequence of transformations $\Pi \dashrightarrow \Pi'$ and every grey node G in Π, each of the above propositions is either true or false on this sequence. Therefore, if we take any propositional formula built from these propositions, it is also either true or false on this sequence.

5.1 Expressing the Digest

Our next aims are to write down a propositional formula that expresses that Π' is local, and also represent the digest of any local refutation. To this end we will first introduce propositional variables and formulas over grey nodes, then write down further formulas of these propositions that are satisfied when Π' is local, and finally show that satisfiability of these propositions implies locality of Π'.

Propositions rc and bc. Take a local derivation Π with $\Pi \dashrightarrow \Pi'$. For each grey node G in Π we first introduce the *propositions $rc(G)$ and $bc(G)$* expressing that G is not sliced off and is a conclusion of a symbol-eliminating inference in Π with at least one red (respectively, blue) premise. The propositional variables $rc(G)$ and $bc(G)$ are added to V_Π.

We will only define $rc(G)$, since the case of bc is symmetric.

Consider the following cases depending on the inference introducing G in Π.

1. G is introduced by an inference with only grey premises:
$$\frac{G_1 \quad \cdots \quad G_m}{G} \ ,$$
We then write:
$$rc(G) \leftrightarrow (\neg s(G) \wedge (r(G_1) \vee \ldots \vee r(G_m))). \tag{3}$$
The conditions on the traces of $G_1, \ldots, G_m$ ensure that G can be written as the conclusion of a symbol eliminating inference with at least one red premise. Namely, if $r(G_i)$ holds, then by slicing off G_i and some of the grey nodes from its derivation, G becomes the conclusion of a symbol eliminating inference with at least one red premise.
2. G is introduced by an inference with at least one red premise:
$$\frac{R_1 \quad \cdots \quad R_n \quad G_1 \quad \cdots \quad G_m}{G} \ .$$
We then have:
$$rc(G) \leftrightarrow \neg s(G). \tag{4}$$
3. G is introduced by an inference with at least one blue premise
$$\frac{B_1 \quad \cdots \quad B_n \quad G_1 \quad \cdots \quad G_m}{G} \ .$$
Due to the locality of derivations, we write:
$$\neg rc(G). \tag{5}$$

Equations (3)-(5) are added to the set of propositional formulas P_Π over V_Π.

Propositions rf and bf. We introduce the propositions $rf(G)$ and $bf(G)$ for every grey node G, and add the corresponding variables to V_Π. These propositions are closely related to the definition of digest. The proposition $rf(G)$ holds iff on the path from G to the root of Π either (i) all nodes are grey, or (ii) the first colored node is a blue one. Likewise, the proposition $bf(G)$ expresses that on the path from G to the root of Π, there exists a colored node and the first colored node is a red one.

We will only write down properties of rf, the case of bf is similar. We define rf "inductively", starting from the root (the bottom formula) of the derivation Π.

1. If the successor of G in Π is a red formula, then we write
$$\neg rf(G). \tag{6}$$
2. If the successor of G in Π is a blue formula, then we write
$$rf(G). \tag{7}$$
3. Finally, if the successor of G in Π is a grey node G_1, then we write
$$rf(G) \leftrightarrow (rf(G_1) \vee bc(G_1)) \wedge \neg rc(G_1). \tag{8}$$

Equations (6)-(8) are added to P_Π.

Proposition d. By straightforward inspection of the definition of digest, it is not hard to argue that $d(G)$ can be expressed as follows:
$$d(G) \leftrightarrow (rc(G) \wedge rf(G)) \vee (bc(G) \wedge bf(G)). \tag{9}$$
We add (9) to P_Π.

5.2 Expressing Locality

Take a local derivation Π and a grey node G in it. Depending on the inference introducing G, there are four possible cases:

1. G is a leaf of Π;
2. G is introduced by an inference with grey premises;
3. G is introduced by an inference with at least one red premise;
4. G is introduced by an inference with at least one blue premise. In this case, due to the locality of Π, all premises in the derivation tree of G are either blue or grey.

For each of these cases, we will show how to write down formulas expressing that $\Pi \dashrightarrow \Pi'$ results in a local derivation, that is, Π' is local. Each below listed propositional formulas is added to P_Π.
General properties of grey nodes. Note that, if a node G is not sliced off, then its trace is $\{G\}$, so we have $g(G)$:
$$\neg s(G) \rightarrow g(G). \tag{10}$$
We also know that a node which is sliced off cannot belong to the digest:
$$s(G) \rightarrow \neg d(G). \tag{11}$$

Observe that equations (10)-(11) do not make use of the assumptions that Π is local. That is, (10)-(11) hold for *arbitrary* derivations.

Further, note that for local derivations the properties b, r and g are mutually exclusive. Therefore, for every grey node node G we add the following properties expressing mutual exclusion:

$$\begin{array}{lll} \mathit{color}(G) & \overset{\text{def}}{=} & (b(G) \lor r(G) \lor g(G)) \land \\ & & (b(G) \to \neg r(G) \land \neg g(G)) \land \\ & & (r(G) \to \neg b(G) \land \neg g(G)) \land \\ & & (g(G) \to \neg r(G) \land \neg b(G)). \end{array} \tag{12}$$

G is a leaf. In this case, G cannot be sliced off and we have:

$$\mathit{leaf}(G) \overset{\text{def}}{=} \neg s(G) \land g(G) \tag{13}$$

G is introduced by an inference with grey premises:

$$\frac{G_1 \quad \cdots \quad G_m}{G}\ .$$

The locality of $\Pi \dashrightarrow \Pi'$ implies that if the trace of any $G_1, \ldots, G_m$ contains a red (respectively, blue) formula, then the traces of $G_1, \ldots, G_m$ cannot contain a blue (respectively, red) formula. To further reason about the trace of G, consider the following cases.

(i) If G is never sliced off in $\Pi \dashrightarrow \Pi'$, then the trace of G is clearly grey. Whether G is a conclusion of a symbol eliminating inference only depends on whether the trace of some of the $G_1, \ldots, G_m$ contains either a blue or a red formula.

(ii) If G is sliced off, then the color of the formulas in the trace of G depend on the color of the formulas from the traces of $G_1, \ldots, G_m$.

Based on the above reasoning, we introduce the following formula capturing the properties of the trace of G:

$$\begin{array}{ll} \mathit{grey}(G) \overset{\text{def}}{=} & (r(G_1) \lor \ldots \lor r(G_m) \to \neg b(G_1) \land \ldots \land \neg b(G_m)) \land \\ & (b(G_1) \lor \ldots \lor b(G_m) \to \neg r(G_1) \land \ldots \land \neg r(G_m)) \land \\ & (s(G) \land (r(G_1) \lor \ldots \lor r(G_m)) \to r(G)) \land \\ & (s(G) \land (b(G_1) \lor \ldots \lor b(G_m)) \to b(G)) \land \\ & (s(G) \land g(G_1) \land \ldots \land g(G_m) \to g(G)) \land \\ & (\neg s(G) \to g(G)). \end{array} \tag{14}$$

G is introduced by an inference with at least one red premise:

$$\frac{R_1 \quad \cdots \quad R_n \quad G_1 \quad \cdots \quad G_m}{G}\ .$$

In this case the locality of Π implies that the trace of G can contain only red and grey formulas. Moreover, the color of the formulas from the trace of G only depends on whether G is sliced off, as follows.

(i) If G is sliced off, then the trace of G depends on the traces of $R_1, \ldots, R_n$, $G_1, \ldots, G_m$, and hence the trace of G contains at least one red formula. Also note, that if G is sliced off, then G cannot belong to the digest of Π'.

(ii) If G is not sliced off, then $\mathit{trace}(G) = \{G\}$. Hence, the trace of G only contains grey formulas. Moreover, note that G is the conclusion of symbol eliminating inference. Thus, G also belongs to the digest of Π'.

We therefore introduce the below formula for G, capturing the properties of the trace of G:

$$\begin{array}{lll} \mathit{red}(G) & \overset{\text{def}}{=} & \neg b(G_1) \land \ldots \land \neg b(G_m) \land \\ & & (s(G) \to r(G)) \land \\ & & (\neg s(G) \to g(G)). \end{array} \tag{15}$$

G is introduced by an inference with at least one blue premise:

$$\frac{B_1 \quad \cdots \quad B_n \quad G_1 \quad \cdots \quad G_m}{G}\ .$$

Similarly to the previous case, we introduce the following formula:

$$\begin{array}{lll} \mathit{blue}(G) & \overset{\text{def}}{=} & \neg r(G_1) \land \ldots \land \neg r(G_m) \land \\ & & (s(G) \to b(G)) \land \\ & & (\neg s(G) \to g(G)). \end{array} \tag{16}$$

This completes our construction of the propositional variables and formulas explained in the beginning of this section. Namely, the set of variables V_Π consists of all variable $s(G)$, $r(G)$, $b(G)$, $g(G)$, $rc(G)$, $bc(G)$, $\mathit{rf}(G)$, $\mathit{bf}(G)$ and $d(G)$, and the set P_Π of formulas are all formulas (3)–(16).

Our construction clearly implies the following result.

THEOREM 5.5. *Let Π be a local derivation. Then a sequence $\Pi \dashrightarrow \Pi'$ satisfies all formulas (8)-(16) from P_Π if and only if Π' is local. Moreover, the propositions $r(G)$, $b(G)$, $g(G)$, $rc(G)$, $bc(G)$, $\mathit{rf}(G)$, $\mathit{bf}(G)$ and $d(G)$ have their intended meaning, in particular, in every model Π' of these formulas G belongs to the digest of Π' if and only if $d(G)$ holds on Π'.*

5.3 Derivations as Dags

Refutations found by theorem provers are normally dags. Transforming a dag to a tree can result in an exponential growth in size. Therefore, it is desirable to change our technique to deal with dags. The modification is quite simple: we allow a formula in a dag to be sliced off only if *all* the tree derivations corresponding to the resulting dag are local. Note that this may result in a smaller choice of grey slicing transformations as compared to refutations as trees and hence larger interpolants. Nonetheless, expanding dags to trees may turn to be unfeasible. Therefore, our implementation uses dags.

To build propositional formulas expressing locality on dags, one should only modify the propositions $\mathit{rf}(G)$ and $\mathit{bf}(G)$.

Propositions rf and bf for dags. The proposition $\mathit{rf}(G)$ holds iff on *all paths* from G to the root of Π either (i) all nodes are grey, or (ii) the first colored node is a blue one. Likewise, the proposition $\mathit{bf}(G)$ expresses that on *all paths* from G to the root of Π, the first colored node is a red one. The axiomatisation of these propositions is given below, and (6)-(8) are replaced by the below formulas in P_Π.

We will only define rf, since the axiomatisation of bf is similar. It is defined "inductively" starting from the root (the bottom formula) of the derivation Π.

1. Suppose at least one of the successors of G is a red formula. In this case we write:

$$\neg \mathit{rf}(G). \tag{17}$$

2. Otherwise, all the successors of G are either grey or blue:

$$\frac{G}{G_1 \quad \cdots \quad G_m \quad B_1 \quad \cdots \quad B_k}\ .$$

In this case we write

$$\begin{array}{l} \mathit{rf}(G) \leftrightarrow ((\mathit{rf}(G_1) \lor bc(G_1)) \land \ldots \land (\mathit{rf}(G_m) \lor bc(G_m)) \land \\ \qquad \neg rc(G_1) \land \ldots \land \neg rc(G_m)). \end{array} \tag{18}$$

5.4 Minimising Interpolants in Local Proofs

Theorem 5.5 shows how one expresses locality and digest using the propositional formulas P_Π. This allows us to introduce various measures of "quality" of interpolants and use these measures, together with an SMT solver, to find local proofs giving interpolants that are better in these measures.

As usual, we define a clause to be a disjunction, possibly empty, of literals, that is, atomic formulas and their negations. Since most theorem provers and SMT solvers present proofs as dags of clauses, apart from some preprocessing deriving a set of clauses from R, B and the theory, we assume that the digest of a proof is a set of clauses. If such a clause contains free variables, it is assumed to be implicitly universally quantified. We know that the interpolant extracted from a proof is a propositional combination of clauses occurring in this proof. If a particular clause is a propositional combination of smaller formulas, then the interpolant can be considered a propositional combination of these smaller formulas. The smallest formulas of this form are well-studied in the automated deduction community and called *components*. We will take a slightly modified definition of components from [23].

DEFINITION 5.6 (Component). A *component* of a clause C is either

- ▷ a ground atomic formula occurring in C, or
- ▷ a non-ground clause C_1 such that C has the form $C_1 \vee C_2$, both C_1 and C_2 are non-empty, and the only component of C_1 is C_1 itself.

A clause C is said to be a *g-atom* if C is the only component of itself.

For example, the clause $p(x) \vee a \neq 2 \vee q(x)$ has two components: $p(x) \vee q(x)$ and $a = 2$. Note that we have the following equivalence:

$$\forall x(p(x) \vee a \neq 2 \vee q(x)) \;\equiv\; \forall x(p(x) \vee q(x)) \vee \neg(a = 2).$$

In general, the universal closure of every clause is a boolean combination of the universal closures of its components. Therefore, the extracted interpolant is a boolean combinations of g-atoms, which are components of the formulas in the digest.

The problem of generating minimal interpolants can be thus reduced to the problem of minimising, in some sense, the set of g-atoms used in the interpolant. As minimality of interpolants is not well-understood, we introduce various measures for minimising the size of interpolants. Namely, we are interested in minimising interpolants with respect to (i) the number of g-atoms and (ii) the total weight of g-atoms, counted as a number of symbols. One can also argue that ground interpolants are more useful than those containing quantifiers, so in addition, when the refutation is non-ground, we can also minimise (iii) the number of quantifiers in the g-atoms.

For doing so, we use the fact that the digest of a derivation can be expressed using propositional variables $d(G)$ over grey nodes G and transform the minimisation problem to solving a pseudo-boolean optimisation problem over V_Π as explained below.

We consider a local refutation Π. For every component g of a grey clause G of Π, we introduce a distinct propositional variable $v(g)$. Intuitively, this variable will denote that g occurs in the digest of the transformed proof Π'. For every grey node G in Π, let $g_1, \ldots, g_k$ be all g-atoms of G. We then introduce the following axiom:

$$d(G) \rightarrow v(g_1) \wedge \ldots \wedge v(g_k). \tag{19}$$

In what follows, let $g_1, \ldots, g_m$ be all g-atoms occurring in all grey nodes of Π. Let $w_1, \ldots, w_m$ be the total weights of these atoms, respectively. We denote by $q_1, \ldots, q_m$ the number of quantifiers used, respectively, in $g_1, \ldots, g_m$.

The problem of minimising interpolants is then reduced to the problem of minimising (one of) the following sums:

$$\text{atom}_{\text{cost}} \overset{\text{def}}{=} v(g_1) + \ldots + v(g_m). \tag{20}$$
$$\text{weight}_{\text{cost}} \overset{\text{def}}{=} w_1 v(g_1) + \ldots + w_m v(g_m). \tag{21}$$
$$\text{quantifier}_{\text{cost}} \overset{\text{def}}{=} q_1 v(g_1) + \ldots + q_m v(g_m). \tag{22}$$

Each of these sums is expressed as a pseudo-boolean constraint over the g-atoms $g_1, \ldots . g_m$. A solution to the minimisation problem of the left-hand side of (20)-(22) gives us a subset of $\{g_1, \ldots, g_m\}$, such that the interpolant constructed from the boolean combinations of the formulas in this subset is a smallest interpolant among all interpolants that can be extracted from the various local Π' resulting from grey slicing.

Minimisation of $\text{atom}_{\text{cost}}$ gives the smallest interpolant in the number of distinct g-atoms. Likewise, the minimal values of $\text{weight}_{\text{cost}}$ and $\text{quantifier}_{\text{cost}}$ correspond to the interpolant with the smallest total weight and the smallest number of quantifiers, respectively. Algorithm 1 puts together the algorithm for minimising interpolants.

ALGORITHM 1. ***Minimising Interpolants***
Input: *Closed formulas R and B such that $R \rightarrow \neg B$, and a refutation Π from R, B.*
Output: *Minimised interpolants I_{atom}, I_{weight}, I_{quant} of R and $\neg B$*
Assumption: *All colored symbols of R and B are uninterpreted constants*

```
begin
                I. Proof Localisation.
Compute local proof $\Pi_l$ from $\Pi$, using Theorem 4.2.
                II. Expressing locality.
$\mathcal{G} := \{\}$, $P_\Pi := \{\}$
for each grey node $G$ in $\Pi_l$ do
    Express $d(G)$. Let $P_\Pi := P_\Pi \cup \{(3), (4), (5), (17), (18), (9)\}$.
    Express general properties. Let $P_\Pi := P_\Pi \cup \{(10), (11), (12)\}$.
    If $G$ is a leaf, $P_\Pi := P_\Pi \cup \{(13)\}$.
    If $G$ is introduced by an inference with only grey premises,
            $P_\Pi := P_\Pi \cup \{(14)\}$.
    If $G$ is introduced by an inference with a red premise,
            $P_\Pi := P_\Pi \cup \{(15)\}$.
    If $G$ is introduced by an inference with a blue premise,
            $P_\Pi := P_\Pi \cup \{(16)\}$.
    Compute $G = g_1 \vee \cdots \vee g_k$, where $g_i$ are g-atoms.
    $\mathcal{G} = \mathcal{G} \cup \{g_1, \ldots, g_k\}$
    endfor
    $P_\Pi := P_\Pi \cup \{(19)\}$
endfor
                III. Minimising Interpolants.
$min\_atom_{cost} := \{g_{i_1}, \ldots, g_{i_n}\}$
        $:= min_{\{g_{i_1},\ldots,g_{i_n}\}} \left( \sum_{g_i \in \mathcal{G}} v(g_i) \;\wedge\; P_\Pi \right)$
$min\_weight_{cost} := \{g_{i_1}, \ldots, g_{i_n}\}$
        $:= min_{\{g_{i_1},\ldots,g_{i_n}\}} \left( \sum_{g_i \in \mathcal{G}} w_i v(g_i) \;\wedge\; P_\Pi \right)$
        where $w_i$ denotes the weight of $g_i$
```

```
18 $\mathit{min\_quant}_{cost} := \{g_{i_1}, \dots, g_{i_n}\}$
        $:= \mathtt{min}_{\{g_{i_1},\dots,g_{i_n}\}} \left( \sum_{g_i \in \mathcal{G}} q_i v(g_i) \ \wedge \ P_\Pi \right)$
        where $q_i$ denotes the number of quantifiers uses in $g_i$
19 $I_{atom} = \mathit{Interpolant}_{R,B}(\mathit{min\_atom}_{cost})$
20 $I_{weight} = \mathit{Interpolant}_{R,B}(\mathit{min\_weight}_{cost})$
21 $I_{quant} = \mathit{Interpolant}_{R,B}(\mathit{min\_quant}_{cost})$
22 return $\{I_{atom}, I_{weight}, I_{quant}\}$.
23 end
```

Algorithm 1 uses the result of Theorem 4.2 and starts with translating the input refutation Π of R, B into a local one Π_l (line 2). Note that this step is only applied when Π is non-local, more precisely, when the non-local steps of Π contain colored constants. Further, the set $\mathcal{G}$ of g-atoms from Π_l and the set P_Π of (pseudo-boolean) constraints expressing locality of Π_l are initialised (line 3). Next, for each grey node G in Π_l the constraints expressing locality conditions over the digest and inferences of Π_l are constructed, (lines 5-10). Note that the propositional formulas $\mathit{rf}(G)$ and $\mathit{bf}(G)$ are expressed based on the dag-representation of proofs. The set of g-atoms of G is extracted and added to $\mathcal{G}$ (lines 11-12). Then, the property whether G is in the digest of Π_l is expressed in terms of g-atoms and added to the constraint set P_Π (line 14). As a result of these steps, at the end of line 15 of Algorithm 1, the constraint set P_Π is expressed as a set of clauses ensuring the locality of Π_l (Theorem 5.5). Next, minimal interpolants wrt to the number of g-atoms (line 16), the total weight of g-atoms (line 17), and the number of quantifiers in the g-atoms (line 18) are derived by solving a pseudo-boolean optimisation problem over g-atoms. To this end, the interpolation procedure $\mathtt{Interpolant}_{R,B}(\dots)$ of [18] is called to generate interpolants as boolean combinations of the derived minimal set of g-atoms (lines 19-21).

THEOREM 5.7. *Algorithm 1 is correct. That is, given two formulas R and B and a refutation Π, Algorithm 1 returns the minimal interpolants of R and $\neg B$ wrt the imposed measures (20)-(22) among all interpolants extracted from proofs obtained by grey slicing of* Π.

We next show that finding minimal interpolants by Algorithm 1 is NP-hard.

THEOREM 5.8. *Given two formulas R and B and a refutation Π of $R, B \to \bot$. Extracting a minimal reversed interpolant from Π by grey slicing is an NP-hard optimisation problem.*

PROOF. We use a reduction from finding a maximal independent set of a graph $G(V, E)$ with a set of vertices $V = \{v_1, \dots, v_m\}$ and a set of edges $E = \{(u_1, w_1), \dots, (u_n, w_n)\}$, which is known to be NP-hard.

To fulfil conditions of Theorem 2.2, we first fix a background theory T. For each vertex $v \in V$ we introduce a propositional grey variable, also denoted by v, of weight 1. Further, for each edge $(u, v) \in E$ we add $u \to v$ to the theory T.

Introduce also a blue propositional variable b and a red propositional variable r. Define B to be the blue formula $v_1 \wedge \dots \wedge v_m \wedge b$ and R to be the red formula $\neg v_1 \wedge \dots \wedge \neg v_m \wedge r$.

Further, for each edge $(u, w) \in E$ we introduce the following derivation $\Pi_{u,w}$:

$$\cfrac{\cfrac{B}{u}}{w}$$

Note that this derivation is sound in the underlying theory T. We next construct the proof tree Π to be:

$$\cfrac{\cfrac{\cfrac{B}{u_1}}{w_1} \quad \dots \quad \cfrac{\cfrac{B}{u_n}}{w_n} \quad R}{\bot}$$

where the weight of $\bot$ is considered to be zero. Observe that Π is a valid refutation of R, B. Also note that building a minimal interpolant from Π reduces to finding a derivation Π' obtained from Π by grey slicing with a minimal number of g-atoms.

Let Π' be any derivation obtained from Π by grey slicing. Denote by D' the digest of Π'. For every edge $(u, w) \in E$, the subderivation $\Pi_{u,w}$ either remains a subderivation of Π', or u gets sliced off. In the first case we have $u \in D'$, in the second case $w \in D'$. Therefore, either $u \notin V - D'$, or $w \notin V - D'$, which implies that $V - D'$ is an independent set.

Using similar arguments, one can prove that every independent set S of vertices is a subset of $V - D'$ for some digest D' of a derivation obtained from Π by grey slicing. As each set $V - D'$ is an independent set as well, for every maximal independent set S there exists a digest D' such that S is equal to $V - D'$. Therefore finding a digest of the minimal size is equivalent to finding a maximal independent set. □

Let us finally note that our method can be extended with other proof transformations and optimisation criteria (e.g., [9, 16]), to improve the quality of interpolants. For example, many approaches use templates to identify predicates desirable to be used in invariants or interpolants. Algorithm 1, thanks to its generality, can easily be modified to give preference to predicates matching predefined templates. We therefore believe that our method is of an independent value, since one can first minimise the interpolant and then try to make it semantically better using other methods. Another important feature of our algorithm is that it optimises the proof globally: that is, the optimal solution is not necessarily a sum of optimal solutions given by local proof transformations. We believe this a very essential property of the algorithm not shared by other known approaches to minimising interpolants.

	Interpolant size decrease					
	some	$2-4$	$4-6$	$6-8$	>8	to 0
Symbols	1369	369	55	24	20	386
g-atoms	912	248	37	16	7	386

Table 1. Minimisation results on 6577 TPTP problems with non-trivial interpolants.

6. Experimental Results

6.1 Implementation

We implemented our interpolant minimisation method in C++ and integrated it in version 1.8 of the Vampire theorem prover [24]. For minimising the set of pseudo-boolean constraints we used version 1.0.29 of the SMT solver Yices [10].

Due to the lack of realistic verification benchmarks, that is examples coming from some industrial project, we evaluated our method on the following two classes of problems. First, we took a collection of examples over first-order logic with equality from the TPTP library [25]. We minimised interpolants in the first-order proofs generated by Vampire. Second, we considered SMT benchmarks from the SMT-Lib library [2] that come from bounded model checking. We analysed proofs generated by the Z3 SMT solver [8]. We used version 2.19 of Z3 without any modification. However, for minimising interpolants from Z3 proofs, we implemented a parser for processing and translating Z3 proofs into local proofs. To this end, we used our algorithm for proof localisation (see proof of Theorem 4.2).

	0	< 5	$5-9$	$10-19$	$20-49$	$50-99$	$100-199$	≥ 200
Before	3055	530	1099	1578	2035	991	260	84
After	3441	522	1225	1557	1882	766	166	73

Table 2. Number of symbols in TPTP benchmark interpolants, before and after minimisation.

All experiments reported in this paper were carried out using a 64-bit 2.33 GHz quad core Dell server with 12 GB RAM.

6.2 First-Order Problems

For this part of the experiments, we took a collection of first-order problems from the TPTP library. We started with annotating these problems with coloring information, using the following coloring strategies.

(1) We order symbols by the number of their occurrences in the problem, and starting with the symbols occurring the least number of times, we attempt to assign colors to them. A color can be assigned to a symbol if the symbol does not occur in a formula with a symbol that was already assigned with the opposite color. The colors are being assigned in an alternating manner. If the last assigned color was red, we first attempt assigning blue to the next symbol, and try to assign red only if this the blue color ended in an unsuccessful assignment (i.e. an input formula with two different colored symbols is obtained). We stop when we have attempted to assign a color to all the symbols.

(2) The previous assignment strategy is more or less random. To use interpolants in a more logical way, we used the following idea. Typically, TPTP problems come with annotations classifying formulas from a problem into axioms, conjectures and hypotheses. We have to prove the conjecture from the axioms and hypotheses. It is commonly the case that axioms axiomatise some theory, so we have to prove that the hypotheses imply the conjecture in the theory given by the axioms. This gives us the following way of coloring the problem symbols. We assign blue color to symbols appearing only in the formulas of the conjecture (i.e. formula B), and red color to symbols occurring only in hypothesis (i.e. formula R). The symbols shared by the conjecture and the hypotheses are considered grey, as well as the symbols occurring only in the axioms.

Local proofs for the colored TPTP problems were generated using the interpolation mode of Vampire [13]. Altogether, we evaluated our minimisation method on 9632 colored TPTP examples. Out of the 9632 problem instances, 3055 problems had trivial interpolants, that is the interpolant was either $\top$ or $\bot$. This left us with 6577 problems with non-trivial interpolants. We were interested to see how our minimisation algorithm performs on these problems. To this end, for each of the 6577 problems, our minimisation method took the corresponding local proof generated by Vampire and derived the smallest interpolants by minimising (i) the number of symbols (i.e total weight) and (ii) the number of g-atoms in the interpolant. Table 1 gives a summary on how the size of the interpolant decreases after minimisation, as compared to the interpolant extracted from the original proof. Rows 1 and 2 of Table 1 show respectively the changes in the interpolant size after minimising the number of symbols, respectively the number of g-atoms in the interpolant. For each imposed measure, column 1 of Table 1 lists the number of examples where the size of the interpolants has decreased only by a small amount. The numbers shown in column 2 (resp. in column 3, column 4, and column 5) correspond to the number of those examples whose interpolants became 2-4 (resp. 4-6, 6-8, and more than 8) times smaller after minimisation. Column 6 gives the number of examples whose interpolants became trivial after minimisation, even though the non-minimised interpolants were non-trivial.

	0	<5	5-9	10-19	20-49	50-99	100-199	≥200
$\text{Symbols}_{\text{pre}}$	112	3	149	150	82	90	321	1216
$\text{Symbols}_{\text{post}}$	112	5	168	158	56	87	323	1214
g-atoms	112	1558	303	114	9	0	0	0
Quantifiers	464	279	291	367	394	157	123	48

Table 3. Minimisation results on 2123 SMT benchmarks.

In Table 2 we report on the number of symbols in the interpolants before (row 1) and after (row 2) minimisation. Each column of Table 2 gives the number of interpolants whose number of symbols satisfy the numeric constraint given in the first cell of the column. That is, column 1 gives the number of trivial interpolants (the number of symbols is 0), column 2 shows the number of interpolants with less than 5 symbols, column 3 reports shows the number of interpolants that contain between 5 and 9 symbols etc.

By analysing the results of Table 1, we note that for 854 (respectively 694) examples the number of symbols (respectively, the number of g-atoms) of the interpolant decreased by at least a factor of 2. However, we also note that for 4354 (respectively 4971) problems out of the 6577 examples we tried minimisation did not improve the size: these examples are omitted in Table 1. As the qualitative studies of interpolants is a challenging topic, we believe that the experimental results reported in Table 1 show the potential of our method in generating better interpolants.

In Figures 2 and 3 we show a colored proof of a TPTP problem before and after minimization. Formulas appearing in the interpolant are given in bold, while other consequences of symbol eliminating inferences in italic. Red symbols in the proof are underlined, whereas blue symbols are underbraced. Figures 4 and 5 show the proof before and after minimisation in a tree form. As mentioned, formulas denoted by R (resp. by B or G) refer to red (resp. blue or grey formulas). The formulas G_{814} and G_{41} appear in the original interpolant, but when G_{815}, G_{45} and G_{41} are sliced off by the minimisation algorithm, the new interpolant formulas are G_{853} and G_{86}. This is because the formula G_{853} is eliminating red symbols from the premises it received as a result of the slicing. The formula G_{86} now appears in the interpolant because it is an ancestor of a red symbol eliminating formula. Even though we still have two formulas in the interpolant, its size decreased because G_{853} is a trivial formula $\bot$. When compared to Figure 4, note that the number of grey formulas in Figure 5 has decreased due to grey slicing.

6.3 Experiments with SMT Problems

We used a set of SMT-Lib benchmarks coming from bounded model checking. Variables in these problems correspond to state variables representing various unrolling steps of loops. These variables are typically indexed by integer constants, where the integer index expresses the unrolling step to which the state variable belongs to.

Translating and localising Z3 proofs. We generated proofs of SMT problems by using Z3. Z3 proofs are expressed in the sequent calculus, while our proof localisation and minimisation algorithms work with resolution-style proofs. We therefore parsed and translated Z3 proofs into our framework by using a simple Lisp parser. To this end, we replaced conditionalised Z3 formulas of the form $A_1, \ldots, A_n \vdash F$ by implications $(A_1 \vee \ldots \vee A_n) \rightarrow F$.

The coloring strategy we used for the SMT benchmarks was as follows. Except the state variables, all other symbols were colored grey. We divided the set of state variables into three parts. State variables corresponding to the middle loop unrolling step were left grey, variables from earlier states were marked red and those from later states were colored blue. In our experiment this coloring strategy turned out to be successful, in 95% of all examples we

853. ⊥ [815,86]
815. ¬ udl(sK_0,rl(sK_0)) [814,45]
814. ¬ udl(x_0,rl(x_1)) ∨ ¬ udl(x_0,x_1) [813,17]
813. ¬ udl(x_0,x_1) ∨ $\underline{\mathit{lcl}}$(x_0,x_1) ∨ ¬ udl(x_0,rl(x_1)) [15,17]
86. udl(x_0,rl(x_0)) [79,49]
79. udl(x_9,x_7) ∨ ¬ udl($\underbrace{\mathit{ptp}}$($x_7$,$x_8$),$x_9$) [61,42]
61. udl(x_7,$\underbrace{\mathit{ptp}}$($x_6$,$x_8$)) ∨ ¬ udl($x_7$,$x_5$) ∨ udl($x_5$,$x_6$) [57,33]
57. ¬ udl(x_5,$\underbrace{\mathit{ptp}}$($x_6$,$x_7$)) ∨ udl($x_5$,$x_6$) [33,43]
49. udl($\underbrace{\mathit{ptp}}$(rl($x_3$),$x_4$),$x_3$) [38,43]
45. udl(sK_0,rl(rl(sK_0))) [30,41]
43. ¬ udl($\underbrace{\mathit{ptp}}$($x_1$,$x_2$),$x_1$) [25,24]
42. ¬ udl(x_0,x_0) [25,27]
41. udol(sK_0,rl(sK_0)) [6,7]
38. udl(x_0,rl(x_1)) ∨ udl(x_0,x_1) [input]
33. udl(x_1,x_2) ∨ ¬ udl(x_0,x_1) ∨ udl(x_0,x_2) [input]
30. ¬ udol(x_0,x_1) ∨ udl(x_0,rl(x_1)) [input]
27. eld(x_0,x_0) [input]
25. ¬ udl(x_0,x_1) ∨ ¬ eld(x_0,x_1) [input]
24. eld($\underbrace{\mathit{ptp}}$($x_1$,$x_0$),$x_1$) [input]
17. ¬ $\underline{\mathit{lcl}}$(x_0,x_1) [input]
15. ¬ udl(x_0,x_1) ∨ $\underline{\mathit{lcl}}$(x_0,x_1) ∨ ¬ udl(x_0,rl(x_1)) ∨ $\underline{\mathit{lcl}}$(x_0,rl(x_1)) [input]
7. ¬ $\underline{\mathit{edol}}$($sK_0$,rl($sK_0$)) [input]
6. udol(x_0,x_1) ∨ $\underline{\mathit{edol}}$($x_0$,$x_1$) [input]

Figure 2. Proof of the GEO269+3 problem from the TPTP library.

853. ⊥ [814,86,30,6,7]
814. ¬ udl(x_0,rl(x_1)) ∨ ¬ udl(x_0,x_1) [813,17]
813. ¬ udl(x_0,x_1) ∨ $\underline{\mathit{lcl}}$(x_0,x_1) ∨ ¬ udl(x_0,rl(x_1)) [15,17]
86. udl(x_0,rl(x_0)) [79,49]
79. udl(x_9,x_7) ∨ ¬ udl($\underbrace{\mathit{ptp}}$($x_7$,$x_8$),$x_9$) [61,42]
61. udl(x_7,$\underbrace{\mathit{ptp}}$($x_6$,$x_8$)) ∨ ¬ udl($x_7$,$x_5$) ∨ udl($x_5$,$x_6$) [57,33]
57. ¬ udl(x_5,$\underbrace{\mathit{ptp}}$($x_6$,$x_7$)) ∨ udl($x_5$,$x_6$) [33,43]
49. udl($\underbrace{\mathit{ptp}}$(rl($x_3$),$x_4$),$x_3$) [38,43]
43. ¬ udl($\underbrace{\mathit{ptp}}$($x_1$,$x_2$),$x_1$) [25,24]
42. ¬ udl(x_0,x_0) [25,27]
38. udl(x_0,rl(x_1)) ∨ udl(x_0,x_1) [input]
33. udl(x_1,x_2) ∨ ¬ udl(x_0,x_1) ∨ udl(x_0,x_2) [input]
30. ¬ udol(x_0,x_1) ∨ udl(x_0,rl(x_1)) [input]
27. eld(x_0,x_0) [input]
25. ¬ udl(x_0,x_1) ∨ ¬ eld(x_0,x_1) [input]
24. eld($\underbrace{\mathit{ptp}}$($x_1$,$x_0$),$x_1$) [input]
17. ¬ $\underline{\mathit{lcl}}$(x_0,x_1) [input]
15. ¬ udl(x_0,x_1) ∨ $\underline{\mathit{lcl}}$(x_0,x_1) ∨ ¬ udl(x_0,rl(x_1)) ∨ $\underline{\mathit{lcl}}$(x_0,rl(x_1)) [input]
7. ¬ $\underline{\mathit{edol}}$($sK_0$,rl($sK_0$)) [input]
6. udol(x_0,x_1) ∨ $\underline{\mathit{edol}}$($x_0$,$x_1$) [input]

Figure 3. Transformed proof of Figure 2 by slicing off formulas using weight minimization.

tried input formulas have been translated into formulas colored by at most one color.

However, the usage of colors yielded non-local Z3 proofs in more than 90% of all examples we tried. We translated non-local Z3 proofs into local ones by applying our proof localisation algorithm. To this end, we always used existential quantification to eliminate red symbols from non-local inferences. As the size of generated interpolants depends on the introduced quantified formulas, we believe that a dynamic analysis over the colored symbols to be eliminated, for example eliminate blue symbol instead of a red one, is an interesting topic for further examination.

The result of proof localisation was further used to minimise interpolants.

Minimising local SMT proofs. Altogether, we used 4347 SMT benchmarks. Out of these 4347 examples, we generated interpolants for 2123 problems. We analyse these interpolants below and summarize our results in Table 3.

The remaining 2224 SMT problems we could not fully process. This was due to a 60s time limit which we imposed as the processing time of one problem, including proof translation, coloring and localisation. In 102 cases the run was terminated by reaching the time limit in translating and coloring Z3 proofs, in 1580 cases the timeout was reached in the proof localisation phase, and for 542 benchmarks the time limit was reached during the minimization phase (in constructing and minimising the set of propositional formulas).

Among the 2123 interpolants, 112 interpolants were trivial, and 1659 contained existential quantifiers introduced by proof localisation. The number of symbols in the interpolants was decreased by our minimisation algorithm for 331 interpolants, out of which 14 interpolants had a decrease by more than two times. The number of g-atoms in the interpolant decreased for 83 interpolants, whereas the number of quantified variables was minimised for 7 interpolants. Table 3 shows the distribution of the number of symbol occurring in interpolants before (row 1) and after minimization (row 2). The distribution of the number of g-atoms (row 3) and quantifiers (row 4) in interpolants is shown only before minimization, because the effect of minimisation on these values was not significant. Each column of Table 3 gives the number of interpolants whose number of size/g-atoms/quantifiers are bounded by the numeric value given in the first cell of the column. That is, for examples, the number of symbols in 168 (row 2, column 3) interpolants is between 5-9 after minimisation. The numbers given in column 1 of Table 3 correspond to the number of trivial interpolants.

The experiments show that our minimisation algorithm is not very efficient on this benchmark suite compared to the first-order benchmarks. We believe that the problem is not in the method but in the way Z3 produces proofs (since the produced proofs were not intended for interpolation). It was often the case that the proofs contained very large formulas, sometimes mixing colors in these formulas. The formulas are then quantified by other algorithm and cannot further be removed from the proof, thus spoiling the minimisation statistics. These formulas are normally large conjunctions or if-then-else expressions, which can also be represented as conjunctions and could have been split into smaller ones. This would not only replace large formulas by smaller one, but also improve coloring of proofs and reduce (or eliminate) the necessity to quantify formulas in them. We believe that our technique will work very well if SMT solvers are modified to obtain proofs of a better quality. Moreover, once a proof is found, post processing can also be done and one may try to change non-local parts of the proof again by theorem proving.

7. Related Work

Interpolation has a number of application in formal verification, ranging from approximating the set of reachable sets in predicate abstraction [14, 16] to invariant generation of loops [21]. Formal verification thus crucially depends to which extent “good” interpolants can be automatically generated.

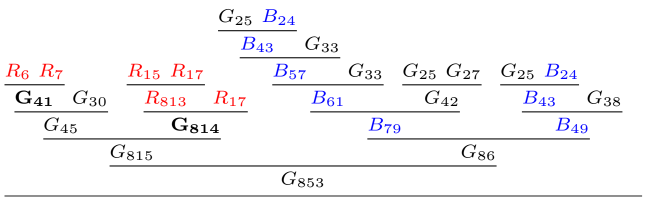

Figure 4. Proof tree for Figure 2.

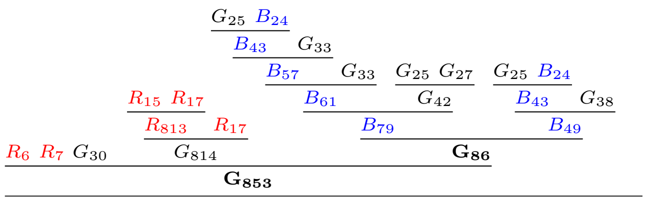

Figure 5. Proof tree for the minimized proof of Figure 3.

General criteria for comparing interpolants can be defined by the logical strength of the interpolant, see e.g. [9, 16]. The approach described in [16] reorders the sequence of resolution steps in a proof to strengthen the derived interpolants. The main heuristic used for proof transformation is to make resolution steps on red/blue variables before those on grey variables. The work of [9] extends [16] and gives a theoretical investigation on the logical strength of propositional interpolants extracted from resolution proofs. The approach uses the notion of labeling functions, which essentially label literals by red, blue or grey labels. The differences among the labeling functions come from how grey literals are labeled (red, blue, or grey). The strength of the various labeling functions is compared, and weaker or stronger interpolants are derived by changing the deployed labeling functions and swapping some nodes in the derivation.

Examples of [9] emphasise that weaker interpolants might lead to better performance, whereas experimental results of [16] show that stronger interpolants can speed up the convergence of a software model checker based on predicate abstraction. Optimising interpolants by only using the logical strength of the interpolant as a selection criteria is thus not always the best way to go in designing efficient interpolation algorithms.

The logical strength of the interpolant is also evaluated in [14, 21], in the context of verification of programs with loops. Although one can derive various program properties by unwinding loop iteration, the resulting set of program properties is a diverging sequence of non-inductive formulas. In [14] interpolants are generated by searching the proof space and avoiding divergence by deeper unwindings of loop iterations. The method is further extended in [21] to infer quantified interpolants. It is shown that by bounding the behavior of the interpolating prover (e.g. delaying inferences over colored or grey symbols), divergence is prevented and an inductive invariant is eventually produced from quantified interpolants. A somehow related approach is presented in [13, 18], where quantified interpolants are extracted from first-order local proof. These techniques generate interpolants by taking the boolean combinations of the grey conclusions of the largest colored subderivations.

The works of [4, 5, 11, 19] evaluate the quality of interpolants by using, in some sense, a different selection criteria. These methods are motivated to generate interpolants that are small in the number of their components, and describe interpolation procedures for the theory of linear integer arithmetic w/o uninterpreted function and predicate symbols. The approach of [4] computes ground interpolants that are exponential in the size of the proofs. The method is improved in [19] by restricting the logical power of the interpolating prover, and is further extended in [5] by handling uninterpreted function and predicate symbols. To this end, [5] shows that quantified interpolants are needed. However, by using guarded quantifiers and divisibility predicates, the quantified interpolants can be translated into equivalent quantifier-free formulas. A similar problem is addressed and solved in [11], where ceiling functions are used to avoid quantified interpolants and generate quantifier-free interpolants of quantifier-free formulas in linear integer arithmetic. Ceiling functions are handled in the interpolating prover by replacing every non-variable ceiling term by a fresh integer variable. Inequality constraints over the newly introduced integer variables are added to capture the semantics of ceiling terms. Whereas [4, 5, 11, 19] show good performance on experiments, due to the lack of realistic benchmarks, it is hard to draw broad conclusions whether the interpolants generated by these works are the "best" in size and expressiveness.

Contrary to all aforementioned works, we define a set of pseudo-boolean constraints over the grey formulas of the proof. Any solution to this set of constraints gives a different interpolant, and any interpolant can be expressed as a solution of the constraint set. The proof transformations carried out in our approach use only slicing off formulas that are logical consequences of other formulas. Furthermore, we evaluate the logical strength of interpolants by minimising the size, the total weight and the number of quantifiers. Unlike [4, 5, 9, 11, 14, 16, 19], our method can generate and minimise interpolants of quantified formulas. When compared to [13], our experiments show that we get better interpolants then the ones of [13] extracted from the largest colored subderivations. More generally spoken, our minimisation algorithm can be applied to any input proof, provided that the input proof can be translated into an equivalent local proof. A special case of such proofs are those whose only colored symbols are uninterpreted constants. Although such a condition might sound severe, it turns out that in practice a large class of examples satisfy this imposed restriction: interpolation benchmarks in the combined theory of uninterpreted functions, predicates and linear integer arithmetic coming from the SMT community satisfy this coloring constraint [4, 5, 11, 19].

8. Conclusion

We described how interpolants extracted from arbitrary proofs can be obtained and minimised in various ways giving smaller interpolants. Our method (1) takes an arbitrary refutation proof, (2) translates it into a local one, provided that all colored symbols are uninterpreted constants, (3) applies minimisation based on analysis of grey areas in the refutation, and (4) computes a minimal interpolant by using pseudo-boolean optimisation.

Our method is very general and can be used with any theory and in conjunction with any theorem prover that outputs refutation proofs of interpolation problems. The evaluation of our method on first-order and SMT bounded model checking benchmarks shows that, in many cases, minimisation considerably decreases the interpolant size.

We intend to integrate our method into concrete verification tools and evaluate our approach on more realistic verification benchmarks. An interesting question we plan to address in the future is how the quality of minimised interpolants effects the efficiency of interpolation-based verification methods. Using a highly optimised pseudo-boolean solver instead an SMT solver is left for further experiments.

We believe that our method opens a new avenue on research in interpolation-based methods. Indeed, other proof transformation methods can be used as well. For example, we can quantify away not only red, but sometimes also blue symbols or slice off colored formulas. In addition, as we pointed out in Section 6 better proofs can considerably improve the quality of interpolants.

Acknowledgments

We acknowledge funding from the University of Manchester and an EPSRC Pathfinder grant (Hoder), the FWF Hertha Firnberg Research grant T425-N23 and the FWF National Research Network RiSE S11410-N23 (Kovács), and an EPSRC grant (Voronkov).

References

[1] L. Bachmair and H. Ganzinger. Resolution theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 2, pages 19–99. Elsevier Science, 2001.

[2] C. Barrett, A. Stump, and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2010.

[3] D. Beyer, T. A. Henzinger, and G. Théoduloz. Lazy Shape Analysis. In *Proc. of CAV*, pages 532–546, 2006.

[4] A. Brillout, D. Kroening, P. Rümmer, and T. Wahl. An Interpolating Sequent Calculus for Quantifier-Free Presburger Arithmetic. In *Proc. of IJCAR*, pages 384–399, 2010.

[5] A. Brillout, D. Kroening, P. Rümmer, and T. Wahl. Beyond Quantifier-Free Interpolation in Extensions of Presburger Arithmetic. In *Proc. of VMCAI*, pages 88–102, 2011.

[6] A. Cimatti, A. Griggio, A. Micheli, I. Narasamdya, and M. Roveri. Kratos - A Software Model Checker for SystemC. In *Proc. of CAV*, pages 310–316, 2011.

[7] W. Craig. Three uses of the Herbrand-Gentzen Theorem in Relating Model Theory and Proof Theory. *Journal of Symbolic Logic*, 22(3):269–285, 1957.

[8] L. de Moura and N. Bjorner. Z3: An Efficient SMT Solver. In *Proc. of TACAS*, pages 337–340, 2008.

[9] V. D'Silva, D. Kroening, M. Purandare, and G. Weissenbacher. Interpolant strength. In *Proc. of VMCAI*, pages 129–145, 2010.

[10] B. Dutertre and L. de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In *Proc. of CAV*, pages 81–94, 2006.

[11] A. Griggio, T. T. H. Le, and R. Sebastiani. Efficient Interpolant Generation in Satisfiability Modulo Linear Integer Arithmetic. In *Proc. of TACAS*, pages 143–157, 2011.

[12] T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan. Abstractions from Proofs. In *Proc. of POPL*, pages 232–244, 2004.

[13] K. Hoder, L. Kovacs, and A. Voronkov. Interpolation and Symbol Elimination in Vampire. In *Proc. of IJCAR*, pages 188–195, 2010.

[14] R. Jhala and K. L. McMillan. A practical and complete approach to predicate refinement. In *Proc. of TACAS*, pages 459–473, 2006.

[15] R. Jhala and K. L. McMillan. Array Abstractions from Proofs. In *Proc. of CAV*, pages 193–206, 2007.

[16] R. Jhala and K. L. McMillan. Interpolant-Based Transition Relation Approximation. *Logical Methods in Computer Science*, 3(4), 2007.

[17] D. Kapur, R. Majumdar, and C. G. Zarba. Interpolation for Data Structures. In *SIGSOFT FSE*, pages 105–116, 2006.

[18] L. Kovacs and A. Voronkov. Interpolation and Symbol Elimination. In *Proc. of CADE*, pages 199–213, 2009.

[19] D. Kroening, J. Leroux, and P. Rümmer. Interpolating Quantifier-Free Presburger Arithmetic. In *Proc. of LPAR-17*, pages 489–503, 2010.

[20] K. L. McMillan. An Interpolating Theorem Prover. *Theor. Comput. Sci.*, 345(1):101–121, 2005.

[21] K. L. McMillan. Quantified Invariant Generation Using an Interpolating Saturation Prover. In *Proc. of TACAS*, pages 413–427, 2008.

[22] R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In *Handbook of Automated Reasoning*, volume I, chapter 7, pages 371–443. 2001.

[23] A. Riazanov and A. Voronkov. Splitting without Backtracking. In *Proc. of IJCAI*, pages 611–617, 2001.

[24] A. Riazanov and A. Voronkov. The Design and Implementation of Vampire. *AI Communications*, 15(2-3):91–110, 2002.

[25] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. *J. Autom. Reasoning*, 43(4):337–362, 2009.