Splitting Proofs for Interpolation

Bernhard Gleiss, Laura Kovács, and Martin Suda

TU Wien, Austria

Abstract. It is known that Craig interpolants can be efficiently computed from special kind of proofs, called local or split proofs. In this paper we focus on local proofs and describe a novel algorithm for computing first-order interpolants from such proofs. Compared to previous approaches to interpolation, our work generates interpolants in full first-order logic with theories and produces interpolants that are "smaller" than the ones generated before. By smaller we mean, for example, that the interpolant contains less symbols and/or literals. Our algorithms uses splitting functions on local proofs and computes optimized interpolants, possibly with quantifiers, wrt. to these functions. We implemented our approach in the first-order theorem prover VAMPIRE and evaluated on a large number of examples coming from the first-order proving community. Our experiments give practical evidence that our work improves the state-of-the-art in first-order interpolation.

1 Introduction

Starting with the pioneering work of McMillan [14], interpolation became a powerful approach in verification thanks to its use in predicate abstraction and model checking [15, 1, 18]. To prove program properties over a combination of data structures, such as integers, arrays and pointers, several approaches based on theory-specific reasoning have been proposed, see e.g. [13, 5, 4]. While powerful, these techniques are limited to quantifier-free fragments of first-order logic. Addressing reasoning in full first-order theories, quantified interpolants are computed in [16, 10, 3, 22] and further optimized with respect to various measures in [8].

In this paper, we address interpolation in full first-order logic and introduce a novel approach to generate interpolants, possibly with quantifiers. Our approach improves and simplifies the aforementioned techniques, in particular [10, 8]. In [10, 8], the size of computed interpolants is in the worst case quadratic in the size of the proof and the generated interpolants may contain redundant subformulas. Our work addresses these issues and infers interpolants that are linear in the size of the proof and are much simpler than in [10, 8]. We proceed as follows. We separate the requirements on a formula being an interpolant into a part restricting the logical strength of an interpolant and a part restricting which symbols are allowed to be used in an interpolant. This way, we first handle formulas, called intermediants, satisfying the requirements on the logical strength of interpolants, and only then we restrict the generated space of intermediants to the ones that satisfy the restriction on the interpolants signature.

The work of [10] relies on so-called local proofs (or split proofs) and constructs interpolants by splitting local proofs into (maximal) subproofs. Splitting proofs is determined by the signature of formulas used in the proofs. We observed however that there are many ways to split a proof, resulting in interpolants that are different in size and strength. We therefore propose a general framework for splitting proofs and using the boundaries of the resulting subproofs to construct the intermediants. The key feature of our work is that the interpolants inferred from our various proof splits are linear in the size of the proof. When constructing interpolants from proof splits, we note that local proofs are exactly the ones that ensure that proof splits yield intermediants that satisfy the requirements of interpolants. Using local proofs and proof splits, we then describe a powerful heuristic and an optimality criterion how to choose the "best" proof split, and hence the resulting interpolant.

Contributions. The main contributions of this paper are as follows.

- We present a new algorithm for first-order-interpolation using local proofs in arbitrary sound inference systems. That is, our work can be used in any sound calculus and derives interpolants, possibly with quantifiers, in arbitrary first-order theories.
- Our interpolation algorithm is the first algorithm ensuring that the size of the interpolant is linear in the size of the proof while working with an arbitrary sound logical calculus. This result improves [10] and generalises the work of [16] to any sound inference system.
- We implemented our work in the VAMPIRE theorem prover [12] and evaluated our method on a large number of examples coming from the TPTP library [21]. Our experimental results confirm that our work improves the state-of-the-art in first-order interpolation.

The rest of this paper is structured as follows. The background notation on proofs and interpolation is covered in Section 2. We then show how to construct linear sized interpolants in Section 3 and present optimisations to the procedure in Section 4. We compare to related work in Section 5, describe our experimental results in Section 6, and conclude in Section 7.

2 Preliminaries

This section introduces the relevant theoretical notions to our work.

Formulas. We deal with standard first-order predicate logic with equality. We allow all standard boolean connectives and quantifiers in the language and, in addition, assume that it contains the logical constants $\top$, $\bot$ for true and false, respectively. Without loss of generality, we restrict ourselves to closed formulas, i.e. we do not allow formulas to contain free variables. The non-logical symbols of a formula F, denoted by $\mathcal{N}(F)$, are all the predicate symbols and function symbols (including constants) occurring in F. Note that this excludes (quantified) variables and the equality symbol.

An *axiomatisable theory*, or simply a *theory* is any set of formulas. For example, we can use the theory of linear integer arithmetic or the theory of lists. We will from now on restrict ourself to a fixed theory $\mathcal{T}$ and give all definitions relative to $\mathcal{T}$. This includes that we write $F_1, \ldots, F_n \vDash F$ (instead of $F_1, \ldots, F_n \vDash_{\mathcal{T}} F$) to denote that every model of $\mathcal{T}$ which satisfies each $F_1, \ldots, F_n$ also satisfies F.

Definition 1. Let $F_1, \ldots, F_n, F$ be formulas, $n \geq 0$. An *inference rule* R is a tuple $(F_1, \ldots, F_n, F)$. An *inference system* $\mathfrak{S}$ is a set of inference rules.

Definition 2. An inference rule $R = (F_1, \dots, F_n, F)$ is called *sound*, if $F_1, \dots, F_n \models F$. An inference system $\mathfrak{S}$ is called *sound*, if it only consists of *sound* inference rules.

From now on, we further restrict ourselves to a fixed inference system $\mathfrak{S}$ which is sound (relative to $\mathcal{T}$) and give all definitions relative to that system.

Derivations and proofs. We model logical proofs as directed hypergraphs in which vertices are associated with formulas and (hyper-)edges with inferences. Because an inference always has exactly one conclusion, we only need hypergraphs where each edge has exactly one end vertex. Moreover, because the order of premises of an inference may be important, we use tuples to model the edges. We will from now on refer to such (hyper-)edges simply as *inferences*.

Definition 3. Let G be a formula and $\mathcal{F}$ a set of formulas. A *proof of G from axioms* $\mathcal{F}$ is a finite acyclic labeled directed hypergraph $P = (V, E, L)$, where V is a set of vertices, E a set of inferences, and L is a labelling function mapping each vertex $v \in V$ to a formula $L(v)$. For an inference $r \in E$ of the form $(v_1, \dots, v_n, v)$, where $n \geq 0$, we call $v_1, \dots, v_n$ the *premises of* r and v the *conclusion of* r.

Additionally, we require the following:

1. Each vertex $v \in V$ is a conclusion of exactly one inference $r \in E$.
2. There is exactly one vertex $v_0 \in V$ that is not a premise of any inference $r \in E$ and $L(v_0) = G$.
3. each $r \in E$ is either
   - an inference of the form (v) and $L(v) \in \mathcal{F}$, or
   - an inference of the form $(v_1, \dots, v_n, v)$ and $(L(v_1), \dots, L(v_n), L(v)) \in \mathfrak{S}$.

In the first case, we call r an *axiom inference*, in the second case, r is called a *proper inference*. A *refutation* from axioms $\mathcal{F}$ is a proof of the formula $\bot$ from $\mathcal{F}$.

Note that in order to support multiple occurrences of the same formula in a proof, one needs to distinguish between vertices and the formulas assigned to them via the labelling function L. However, because this generality is orthogonal to the ideas we want to present, we will from now on identify each node $v \in V$ with its formula $L(v)$ and stop referring to the labelling function explicitly.

In the above definition, condition 1 ensures that any formula of the proof is justified by exactly one inference. Later on we will look at subgraphs of a proof, which are not necessarily proofs themselves and in particular do not satisfy condition 1, since they contain formulas, which are not justified by any inference of the subgraph. We call such a subgraph a derivation and call the formulas which are not justified by any inference the premises of the derivation. We can see a proof as a derivation having no premises.

Definition 4. The definition of a *derivation of G from axioms* $\mathcal{F}$ is the same as that of a proof $P = (V, E, L)$ of G from $\mathcal{F}$, except that condition 1 is generalised to:

1. Each formula $F \in V$ is a conclusion of *at most one* inference $r \in E$.

The *set of premises of a derivation* P, denoted by $\mathrm{Prem}(P)$, consists of all formulas $F \in V$, such that there exists no inference $r \in E$ with conclusion F.

The definition of a derivation is not natural as it distinguishes between axioms and premises. This distinction is, however, very important for us, as it enables a succinct presentation of the results in Sect. 3.

Lemma 5 (Soundness). Let P be a derivation of G from axioms $\mathcal{F}$. Then we have

$$\mathcal{F} \models (\bigwedge_{F_k \in \mathrm{Prem}(P)} F_k) \rightarrow G.$$

To formalise the idea of a proof traversal in which the inferences are considered one by one from axioms to the final formula G, we make use of topological orderings.

Definition 6. Let $P = (V, E, L)$ be a derivation. A topological ordering $<^T$ for P is a linear ordering $<^T$ on E such that for any two inferences $r_1, r_2 \in E$ if the conclusion of r_1 is a premise of r_2 then $r_1 <^T r_2$.

A topological ordering exists for every derivation, because proofs, and thus also derivations, are required to be acyclic.

Interpolation. We now recall the notion of a logical interpolant.

Definition 7. Let A and B be formulas.

1. A non-logical symbol $s \in \mathcal{N}(A \rightarrow B)$ is called *A-local*, if $s \in \mathcal{N}(A) \setminus \mathcal{N}(B)$, *$B$-local*, if $s \in \mathcal{N}(B) \setminus \mathcal{N}(A)$, and *global* otherwise.
2. An *interpolant* for A, B is a formula I such that $\models A \rightarrow I$, $\models I \rightarrow B$ and all non-logical symbols of I are global.

Craig's interpolation theorem [6] guarantees the existence of an interpolant for any pair of formulas A, B for which $\models A \rightarrow B$. In the sequel, we assume A and B to be fixed and give all definitions relative to A and B.

Refutational theorem proving. To prove a first-order formula F in practice, a refutational theorem prover proceeds by negating the input formula, applying a normal form transformation, such as the Conjunctive Normal Form transformation, to the negation, and deriving a contradiction $\perp$ from the obtained set of formulas $\mathcal{C}_{\neg F} = \mathit{CNF}(\neg F)$. More specifically, in the case of proving the implication $A \rightarrow B$, the prover starts with axioms $\mathit{CNF}(A \wedge \neg B)$.

This is relevant for our work, because we rely on refutations as input for our method. However, a complication arises, because the normal form transformations *CNF* typically involves steps like sub-formula naming and Skolemisation [17, 19], which 1) introduce new non-logical symbols, 2) in general do not preserve logical equivalence.

To deal with 1) we impose a restriction on *CNF* which dictates that the symbols newly introduced on behalf of A and $\neg B$ do not overlap. Formally, we require

$$\mathcal{N}(A) \cap \mathcal{N}(\neg B) = \mathcal{N}(\mathit{CNF}(A)) \cap \mathcal{N}(\mathit{CNF}(\neg B)), \tag{1}$$

which is a very natural condition, because the newly introduced symbols are invariably required to be fresh.[1]

To deal with 2), let us first recall that steps like sub-formula naming and Skolemisation, although they do not preserve logical equivalence, do preserve satisfiability. While this is sufficient to guarantee soundness of refutational theorem proving, it is not enough for the purposes of interpolation. Fortunately, a stronger property, which is rarely stated explicitly, usually holds for the normal form transformation, namely the *preservation of models over the common symbols*. Formally, we require for every formula F that

[1] This could potentially be violated by an advanced transformation based on formula sharing. In particular, the case would need to involve a common sub-formula of A and $\neg B$.

– every model $\mathcal{M}'$ of $CNF(F)$ is also a model of F, and
– every model $\mathcal{M}$ of F can be extended to $\mathcal{M}'$ which is a model of $CNF(F)$,

where extended means that $\mathcal{M}'$ restricted to $\mathcal{N}(F)$ equals $\mathcal{M}$.

Equipped with a transformation CNF satisfying the above requirements, the general approach to interpolation from refutations consists of the following steps:

1. Given formulas A and B, compute the respective normal forms $\mathcal{C}_A = CNF(A)$ and $\mathcal{C}_{\neg B} = CNF(\neg B)$.
2. Find a refutation P from axioms $\mathcal{C}_A \cup \mathcal{C}_{\neg B}$.
3. Extract from P a formula I such that $\mathcal{C}_A \models I$, $\mathcal{C}_{\neg B}, I \models \bot$, and all non-logical symbols of I are global.[2]

Lemma 8. The formula I obtained in the last step is an interpolant for A and B.

3 Interpolants from refutations

We can separate the properties of an interpolant into two parts, the logical part and the restriction to the global symbols. Instead of considering only interpolants, we now want to look more generally at the formulas, which satisfy the logical part of the properties of interpolants, but not necessarily the restriction to the global symbols. We call such formulas intermediants.

Definition 9. Let A, B be two formulas. An *intermediant* for A, B is a formula I such that we have both $\models A \to I$ and $\models I \to B$.

In the first part of this section, we want to investigate the space of intermediants, which is induced by a given refutation. In the second part, we look at the subspace of those intermediants which also respect the restriction on the global symbols, i.e. the formulas which are interpolants.

3.1 Splitting refutations

Let us now show how to use a refutation of $A \to B$ to construct intermediants. Intuitively, we want to split the refutation into two parts and construct a formula which describes the boundaries between the parts.

In the light of the discussion at the end of the last section, we assume the formulas A and $\neg B$ have been transformed to sets of axioms $\mathcal{C}_A$ and $\mathcal{C}_{\neg B}$. It is also natural to extend the notion of an intermediant to axiom sets:

Definition 10. Let $\mathcal{C}_A$ and $\mathcal{C}_{\neg B}$ be two sets of axioms. An *intermediant* for $\mathcal{C}_A, \mathcal{C}_{\neg B}$ is a formula I such that we have both $\mathcal{C}_A \models I$ and $\mathcal{C}_{\neg B}, I \models \bot$.

Splitting a proof into two parts for us means mapping each inference to one of the two parts. Formally, we introduce a two element set $\{\mathcal{A}, \mathcal{B}\}$ to serve as a co-domain of such mapping, where $\mathcal{A}$ denotes the A-part and $\mathcal{B}$ the B-part. It is natural to map the axioms from $\mathcal{C}_A$ to $\mathcal{A}$ and the axioms from $\mathcal{C}_{\neg B}$ to $\mathcal{B}$, therefore we only consider mappings of this form. All other inferences can be mapped to any part.

[2] Note that the symbols are global with respect to A and B if and only if they are global with respect to $\mathcal{C}_A$ and $\mathcal{C}_{\neg B}$ thanks to the requirement (1).

Definition 11. Let P be a refutation from axioms $\mathcal{C}_A \cup \mathcal{C}_{\neg B}$. A *splitting function* $\mathcal{S}$ is a function assigning each inference of P to either $\mathcal{A}$ or $\mathcal{B}$, such that for each axiom inference $r = (F)$, if $\mathcal{S}(r) = \mathcal{A}$ then $F \in \mathcal{C}_A$ and r is called an *A-axiom*, and if $\mathcal{S}(r) = \mathcal{B}$ then $F \in \mathcal{C}_{\neg B}$ and r is called a *B-axiom*.

A given splitting function $\mathcal{S}$ splits a proof into several maximal subderivations. We now want to caption this intuitive notion formally. We start with the concept of In-formulas (resp. Out-formulas) of P and $\mathcal{S}$. Intuitively, these are the formulas which occur at the boundary between the subderivations.

Definition 12. Let $P = (V, E, L)$ be a refutation from axioms $\mathcal{C}_A \cup \mathcal{C}_{\neg B}$ and let $\mathcal{S}$ be a splitting function on P. The *set of in-formulas*, which is denoted $\mathrm{In}(P, \mathcal{S})$, consists of every formula $F \in V$, which has the following properties:

- There exists an inference $r_1 \in E$ with conclusion F and $\mathcal{S}(r_1) = \mathcal{B}$.
- There exists an inference $r_2 \in E$ with premise F and $\mathcal{S}(r_2) = \mathcal{A}$.

The *set of out-formulas*, which is denoted $\mathrm{Out}(P, \mathcal{S})$, consists of every formula $F \in V$, which has the following properties:

- There exists an inference $r_1 \in E$ with conclusion F and $\mathcal{S}(r_1) = \mathcal{A}$.
- Either there exists an inference $r_2 \in E$ with premise F and $\mathcal{S}(r_2) = \mathcal{B}$, or $F = \bot$.

We are now able to formally introduce the maximal subderivations.

Definition 13. Let $P = (V, E, L)$ be a refutation from axioms $\mathcal{C}_A \cup \mathcal{C}_{\neg B}$ and let $\mathcal{S}$ be a splitting function on P. Let $r \in E$ be an inference and let $\{r_1, \ldots, r_l\}$ be the set of those inferences which derive a premise of r and are mapped by $\mathcal{S}$ to the same part as r, i.e. $\mathcal{S}(r) = \mathcal{S}(r_i)$ for $i = 1, \ldots l$. Then we define $\mathrm{Sub}(r)$ recursively as

$$\mathrm{Sub}(r) = \{r\} \cup \mathrm{Sub}(r_1) \cup \cdots \cup \mathrm{Sub}(r_l).$$

Now let $F \in \mathrm{Out}(P, \mathcal{S})$ (resp. $F \in \mathrm{In}(P, \mathcal{S})$) be a formula and r be the inference deriving F. We define the *maximal A-subderivation (resp. B-subderivation) of* F, denoted by $\mathrm{Sub}(F)$, as the induced derivation $(V', \mathrm{Sub}(r), L)$, where V' contains every vertex which is either a premise or a conclusion of an inference in $\mathrm{Sub}(F)$. We call F the *conclusion* of $\mathrm{Sub}(F)$.

The dependencies of F, written $\mathrm{Dep}(F)$, are defined as the premises of $\mathrm{Sub}(F)$.

We can observe that the In-formulas (resp. Out-formulas) are the premises (resp. conclusions) of all maximal A-subderivations. Dually, the In-formulas (resp. Out-formulas) are the conclusions (resp. premises) of all maximal B-subderivations. The use of the introduced concepts is demonstrated in Fig. 3.1.

Note that the A-subderivations contain all A-axioms, but no B-axiom. Therefore the A-axioms's contribution to the derivation is captured by the A-subderivations. The key idea of this subsection is that encoding the contribution of the A-subderivations as a formula therefore yields the formula I we are looking for. The following lemma tells use how to describe the contribution of an A-subderivation.

Lemma 14. Let P be a refutation from axioms $\mathcal{C}_A \cup \mathcal{C}_{\neg B}$ and let $\mathcal{S}$ be a splitting function on P.

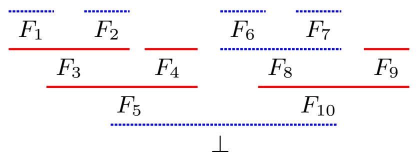

Fig. 1. Consider the proof above along with the splitting function which is denoted by drawing the inferences assigned to $\mathcal{A}$ using solid red lines and the inferences assigned to $\mathcal{B}$ using dashed blue lines. The maximal A-subderivation of F_5 has premises F_1, F_2 (and conclusion F_5), the maximal A-subderivation of F_{10} has premise F_8. The maximal B-subderivation of F_1 has no premises, the maximal B-subderivation of $\perp$ has premises F_5 and F_{10}. The In-formulas are F_1, F_2 and F_8 and the Out-formulas are F_5 and F_{10}. The induced simple splitting formula is $((F_1 \wedge F_2) \rightarrow F_5) \wedge (F_8 \rightarrow F_{10})$.

1. Let $F \in \text{Out}(P, \mathcal{S})$. Then we have $\mathcal{C}_A \vDash (\bigwedge_{F_k \in \text{Dep}(F)} F_k) \rightarrow F$.
2. Let $F \in \text{In}(P, \mathcal{S})$. Then we have $\mathcal{C}_{\neg B} \vDash (\bigwedge_{F_k \in \text{Dep}(F)} F_k) \rightarrow F$.

We therefore arrive at the following definition.

Definition 15. Let P be a refutation from axioms $\mathcal{C}_A \cup \mathcal{C}_{\neg B}$ and let $\mathcal{S}$ be a splitting function on P. The formula

$$I := \bigwedge_{F \in \text{Out}(P,\mathcal{S})} ((\bigwedge_{F_k \in \text{Dep}(F)} F_k) \rightarrow F)$$

is called the *simple splitting formula of* P *induced by* $\mathcal{S}$.

Theorem 16. Let P be a refutation from axioms $\mathcal{C}_A \cup \mathcal{C}_{\neg B}$ and let $\mathcal{S}$ be a splitting function on P. Then the simple splitting formula I induced by $\mathcal{S}$ is an intermediant.

Proof.
1. For each $F \in \text{Out}(P, \mathcal{S})$, we can use Lemma 14.1 to get $\mathcal{C}_A \vDash (\bigwedge_{F_k \in \text{Dep}(F)} F_k) \rightarrow F$. Therefore we have $\mathcal{C}_A \vDash \bigwedge_{F \in \text{Out}(P,\mathcal{S})}((\bigwedge_{F_k \in \text{Dep}(F)} F_k) \rightarrow F)$.
2. Let $<^T$ be a topological ordering for P and let $F_1, \ldots, F_n$ denote the formulas of $\text{In}(P) \cup \text{Out}(P)$ in the order induced by $<^T$. We visit the formulas from F_1 to F_n and prove by complete induction that $I, \mathcal{C}_{\neg B} \vDash F_1, \ldots, F_i$. Since $F_n = \perp$, we afterwards are able to conclude $I, \mathcal{C}_{\neg B} \vDash \perp$.
   Inductive step: Let us assume, by the induction hypothesis, that $I, \mathcal{C}_{\neg B} \vDash F_1, \ldots, F_{i-1}$. We make a case distinction on $\mathcal{S}(r)$, where r is the inference which derived F_i:
   - Case $\mathcal{S}(r) = \mathcal{A}$: By the definition of I, we know that $I \vDash (\bigwedge_{F_k \in Dep(F_i)} F_k) \rightarrow F_i$. Using both the definition of topological orderings and the definition of Dep we know that $\text{Dep}(F_i) \subseteq \{F_1, \ldots F_{i-1}\}$, so we can combine the previous facts to obtain $I, \mathcal{C}_{\neg B} \vDash F_1, \ldots, F_i$.
   - Case $\mathcal{S}(r) = \mathcal{B}$: We use Lemma 14.2 to conclude $\mathcal{C}_{\neg B} \vDash (\bigwedge_{F_k \in \text{Dep}(F_i)} F_k) \rightarrow F_i$. As in the previous case, we can use $\text{Dep}(F_i) \subseteq \{F_1, \ldots F_{i-1}\}$ to conclude $I, \mathcal{C}_{\neg B} \vDash F_1, \ldots, F_i$.

We summarise the ideas of this subsection in Simple-splitting-formula (Algorithm 1).

Algorithm 1 Simple-splitting-formula

```
choose a splitting function on $P$.
compute $\mathrm{Out}(P)$ and $\mathrm{Dep}(F)$ for all $F$ using depth first search
**return** $I$ as defined in Definition 15
```

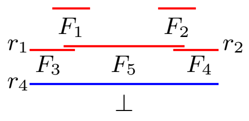

Fig. 2. Let $r_1 = (F_1, F_3)$, $r_2 = (F_2, F_4)$, $r_3 = (F_1, F_2, F_5)$, and $r_4 = (F_3, F_4, F_5, \perp)$. Let further $\mathcal{S}(r_1) = \mathcal{S}(r_2) = \mathcal{S}(r_3) = \mathcal{A}$ and $\mathcal{S}(r_4) = \mathcal{B}$. Then Algorithm 1 generates the simple splitting formula $I = (F_1 \to F_3) \wedge (F_2 \to F_4) \wedge ((F_1 \wedge F_2) \to F_5)$. There is no intermediant which is both logically equivalent to I and contains each formula of the given proof at most once.

3.2 Intermediants of linear size

Simple-splitting-formula yields an intermediant of size which is in the worst case quadratic in the size of the proof. This may be prohibitively large for large proofs. In this subsection, we describe an algorithm which yields intermediants of size which is linear in the size of the proof. Modifying Algorithm 1 to generate such an intermediant is nontrivial: there are examples, where the simple splitting formula is provably logically stronger than any intermediant which uses every formula of the refutation only once, cf. Fig. 2. We therefore need to modify the algorithm such that it produces an intermediant which is logically weaker but still sufficiently strong to be inconsistent with C_B.

The key idea for the new algorithm is contained in the following definition.

Definition 17. Let P be a refutation from axioms $\mathcal{C}_A \cup \mathcal{C}_{\neg B}$, $\mathcal{S}$ a splitting function on P, and let $<^T$ be a topological ordering for P. Furthermore let $F_1, \ldots, F_n$ denote the formulas of $\mathrm{In}(P) \cup \mathrm{Out}(P)$ ordered by $<^T$. Now let

$$I_i = \begin{cases} \top & \text{if } i = n+1 \\ F_i \to I_{i+1} & \text{if } F_i \in \mathrm{In}(P) \\ F_i \wedge I_{i+1} & \text{if } F_i \in \mathrm{Out}(P) \end{cases}$$

Then I_1 is called *linear splitting formula of P induced by $\mathcal{S}$ and $<^T$*.

Note that the size of I_1 is linear in the size of P in Definition 17.

Theorem 18. Let P be a refutation from axioms $\mathcal{C}_A \cup \mathcal{C}_{\neg B}$, let $\mathcal{S}$ be a splitting function on P and let $<^T$ be a topological ordering for P. Then the linear splitting formula I induced by $\mathcal{S}$ and $<^T$ is an intermediant.

Proof. Let

$$I' = \bigwedge_{F_i \in \mathrm{Out}(P)} \left(\left(\bigwedge_{F_k \in \mathrm{In}(P), k <^T i} F_k \right) \to F_i \right).$$

Algorithm 2 Linear-splitting-formula

```
choose a splitting function and a topological ordering on $P$.
compute $\mathrm{In}(P)$ and $\mathrm{Out}(P)$
**return** $I_1$ as defined in Definition 17
```

First note that I' is logically equivalent to I: This can be proved by a simple induction using the two facts that conjunction on the right distributes over implication and that $A \to (B \to C)$ is equivalent to $(A \wedge B) \to C$.

Now we complete the proof by showing that I' is an intermediant:

1. Using both the definition of topological orderings and the definition of Dep we know that $\mathrm{Dep}(F_i) \subseteq \{F_k \in \mathrm{In}(P) \mid k <^T i\}$, so I' is logically weaker than the simple splitting formula. Therefore $\mathcal{C}_A \models I'$ follows from Theorem 16.1.
2. We can show $\mathcal{C}_{\neg B}, I' \models \bot$ by re-using the proof of Theorem 16.2 with $<^T$ as the topological ordering and by replacing $\mathrm{Dep}(P)$ with $\{F_k \in \mathrm{In}(P) \mid k <^T i\}$.

We summarise the presented ideas in Linear-splitting-formula (Algorithm 2) and conclude this subsection by pointing out the following basic lemma, which will become useful later in the paper.

Lemma 19. Let $P = (V, E, L)$ be a refutation from axioms $\mathcal{C}_A \cup \mathcal{C}_{\neg B}$ and let $\mathcal{S}$ be a splitting function on P. Let further I be the linear splitting formula induced by $\mathcal{S}$ and let $F \in V$ be an arbitrary formula different from $\bot$. Then F occurs in I if and only if there are two inferences r_1, r_2, where r_1 derives F, F is a premise of r_2 and $\mathcal{S}(r_1) \neq \mathcal{S}(r_2)$.

3.3 Interpolants as special intermediants

In the previous subsections, we discussed how to construct intermediants given a splitting function. We now look closer at the question which splitting function to choose. While studying the intermediants induced by different choices of a splitting function is an interesting topic in general, we turn our attention to the problem of choosing a splitting function such that the induced intermediant is an interpolant, i.e. we have the additional requirement that the intermediant contains no local symbols.

Let us recall the notion of local proofs—also called split proofs—introduced by Jhala and McMillan [9]:

Definition 20 (Local Proof). A proof $P = (V, E, L)$ from axioms $\mathcal{C}_A \cup \mathcal{C}_{\neg B}$ is *local* if for every inference $(F_1, \ldots, F_n, F) \in E$ we have either:

– $\mathcal{N}(F_1) \cup \ldots \cup \mathcal{N}(F_k) \cup \mathcal{N}(F) \subseteq \mathcal{N}(\mathcal{C}_A)$ or
– $\mathcal{N}(F_1) \cup \ldots \cup \mathcal{N}(F_k) \cup \mathcal{N}(F) \subseteq \mathcal{N}(\mathcal{C}_{\neg B})$.

The definition of local proofs ensures that we can define a splitting function $\mathcal{S}$ which maps all inferences with A-local symbols to $\mathcal{A}$ and those with B-local symbols to $\mathcal{B}$.

Definition 21. Let P be a local proof. A *local splitting function on* P is a splitting function $\mathcal{S}$ on P such that $\mathcal{S}(r) = \mathcal{A}$ (resp. $\mathcal{S}(r) = \mathcal{B}$) for all inferences r having as premise or conclusion a formula containing an A-local (resp. a B-local) symbol.

The corollary of the following lemma represents the central observation of this subsection: local proofs are exactly the proofs on which we can define a splitting function which induces an intermediant which is an interpolant.

Lemma 22. Let $P = (V, E, L)$ be a refutation from axioms $C_A \cup C_B$, $\mathcal{S}$ be a local splitting function on P, and I the corresponding simple (resp. linear) splitting formula.

i) Then any formula $F \in \mathrm{In}(P, \mathcal{S}) \cup \mathrm{Out}(P, \mathcal{S})$ contains neither an A-local nor a B-local symbol.

ii) I contains neither A-local nor B-local symbols.

Proof. i) Consider any formula $F \in \mathrm{Out}(P, \mathcal{S})$. If $F = \bot$ then F trivially contains neither an A-local nor a B-local symbol. Otherwise, we know that there exists an inference $r_1 \in E$ with premise F and $\mathcal{S}(r_1) = \mathcal{B}$. By the locality of $\mathcal{S}$ we get that F contains no A-local symbol. Furthermore, we know that there exists an inference $r_2 \in E$ with conclusion F and $\mathcal{S}(r_2) = \mathcal{A}$. By the locality of $\mathcal{S}$ we get that F contains no B-local symbol.
Now consider any formula $F \in \mathrm{In}(P, \mathcal{S})$. We can use a similar argument to show that F contains neither an A-local nor a B-local symbol.

ii) Follows immediately from i) and the definition of the simple (resp. linear) splitting formula.

Corollary 23. Let P be a local refutation, let $\mathcal{S}$ be a local splitting function on P and let I be either the simple splitting formula or the linear splitting formula. Then I is an interpolant for A, B.

4 Implementing Local Splitting Functions

By the definition of a local splitting function we know that we need to assign axioms and inferences with local symbols to the corresponding part. All the other inferences—the inferences forming the so called *grey area* [8]—can be assigned freely to either part. Different choices on how to split the grey area result in different A-subproofs and therefore in different interpolants, which vary, e.g., in size, the number of contained quantifiers and in logical strength.

We want to minimize the interpolant with respect to a given weight function w, which maps each formula F to its weight $w(F)$. The task we want to solve in this section is, therefore, to be able to come up with a local splitting function which minimises the weight of the resulting interpolant.

We present two different solutions, a heuristical greedy approach and one of expressing the optimal splitting as a minimisation problem. Both solutions are based on the insight from Lemma 19 of Sect. 3: A conclusion F of an inference r_1 occurs in the linear splitting formula if and only if there is an inference r_2 with F as a premise such that the splitting function maps r_1 and r_2 to different parts.

4.1 Greedy weighted sum heuristic

Consider an inference r of the grey area with premises $C_1, \ldots C_n, D_1, \ldots, D_m$ and assume that the inferences deriving $C_1, \ldots, C_n$ are already assigned to $\mathcal{A}$ and that the inferences deriving $D_1, \ldots, D_m$ are already assigned to $\mathcal{B}$. Using Lemma 19, we know

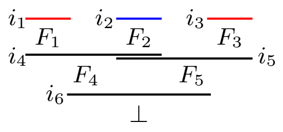

Fig. 3. Let $\mathcal{S}(i_1) = \mathcal{S}(i_3) = \mathcal{A}$ and $\mathcal{S}(i_2) = \mathcal{B}$. Let further $w(F_1) = w(F_3) = 2$ and $w(F_2) = 3$. For both inferences i_4 and i_5, the assignment of the inference to $\mathcal{A}$ is locally optimal, then causes the assignment of i_6 to $\mathcal{A}$ and finally yields an interpolant of size 4. Note that F_2 is used as a premise of both i_4 and i_5, so due to the DAG-structure we would only include it once if we assigned both i_4 and i_5 to $\mathcal{B}$. This would then cause the assignment of i_6 to $\mathcal{B}$ and finally yield a smaller interpolant of size 3.

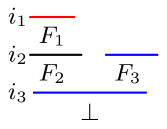

Fig. 4. Let $\mathcal{S}(i_1) = \mathcal{A}$ and $\mathcal{S}(i_3) = \mathcal{B}$. Let further $w(F_1) < w(F_2)$. Algorithm 3 would now assign i_2 to $\mathcal{A}$ and therefore include F_2 in the interpolant. It would be better to assign i_2 to $\mathcal{B}$ in order to include F_1 in the interpolant instead of F_2.

that if we assign r to $\mathcal{A}$, then $D_1, \ldots, D_m$ will be added to the interpolant and if we assign r to $\mathcal{B}$, then $C_1, \ldots, C_n$ will be added to the interpolant.

We can therefore use the following greedy strategy to locally minimize the weight of the interpolant: for any inference r of the grey area, if $\sum_{k=1}^{n} w(C_k) > \sum_{k=1}^{m} w(D_k)$, map r to $\mathcal{A}$, otherwise to $\mathcal{B}$.

This results in Top-down-weighted-sum-heuristic (Algorithm 3):

Algorithm 3 Top-down-weighted-sum-heuristic

```
for each inference $r$ of $P$ (top-down) do
    if $r$ is an $A$-axiom or $r$ contains an $A$-local symbol then
        set $\mathcal{S}(r)$ to $\mathcal{A}$
    else if $i$ is a $B$-axiom or $r$ contains a $B$-local symbol then
        set $\mathcal{S}(r)$ to $\mathcal{B}$
    else
        if $\sum_{k=1}^{n} w(C_k) > \sum_{k=1}^{m} w(D_k)$ then
            set $\mathcal{S}(r)$ to $\mathcal{A}$
        else
            set $\mathcal{S}(r)$ to $\mathcal{B}$
return $\mathcal{S}$
```

The two reasons why a locally optimal choice is not a globally optimal choice can be seen in Figures 3 and 4.

4.2 Encoding optimal splitting as a minimisation problem

Similar to the idea presented in [8], we can alternatively encode the problem of finding an optimal local splitting function as a minimisation problem and pass it to a pseudo-boolean constraint solver. This yields an optimal assignment, but is computationally more expensive.

The encoding works as follows. We use propositional variables x_i to denote that inference i is assigned to $\mathcal{A}$ and use propositional variables L_i to denote that the conclusion of i occurs in the interpolant. We again predict the size of the resulting interpolant using Lemma 19, but this time use the optimisation procedure to make globally optimal choices instead of greedily making locally optimal ones. This leads to algorithm Weighted-sum-optimal (Algorithm 4).

Algorithm 4 Weighted-sum-optimal

```
for each inference $r$ of $P$ do
    if $r$ is an $A$-axiom or $r$ contains an $A$-local symbol then
        assert $x_r$
    else if $r$ is a $B$-axiom or $r$ contains a $B$-local symbol then
        assert $\neg x_r$
    for each parent inference $r'$ of $r$ do
        assert $(\neg(x_r \leftrightarrow x_{r'})) \rightarrow L_{r'}$
compute model $M$ which minimises $\sum_{r \in P} w(concl(r)) \cdot L_r$
for each inference $r$ of $P$ do
    if $x_r$ evaluates to true in $M$ then
        set $\mathcal{S}(r)$ to $\mathcal{A}$
    else
        set $\mathcal{S}(r)$ to $\mathcal{B}$
return $\mathcal{S}$
```

5 Discussion and Related Work

The work of [16] generates quantified interpolants that are linear in the size of the proofs. The approach is however restricted to the superposition inference system. Our work generalises [16] as our interpolation algorithm can be used in any sound logical calculus. Interpolation in superposition proving is also studied in [2] where a method for computing interpolants from arbitrary proofs in first-order logic without equality is presented. While our proof splits are restricted to local proofs, in our approach we handle first-order theories with equality.

The first approach to constructing interpolants in first-order logic with equality using an arbitrary sound inference system was introduced in [11] and later improved by an optimisation technique in [8]. Let us refer to the interpolation algorithm from [11] as $\mathcal{SE}$. In a nutshell, $\mathcal{SE}$ uses two main concepts:

As a first concept, it constructs the largest subderivations containing only symbols from one of the two partitions (cf. Lemma 8 of [11]). This construction corresponds to a commitment to a specific choice of local splitting function in our framework. In contrast, both Algorithm 1 and Algorithm 2 are parametrized by an arbitrary local splitting function and different choices yield different interpolants.

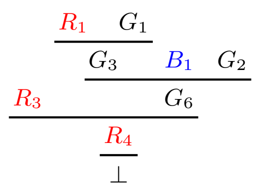

Fig. 5. Consider the proof above, taken from Example 5.2 in [8]. Let R_1, R_3 and R_4 be formulas containing A-local symbols, B_1 a formula containing B-local symbols and let G_1, G_2, G_3 and G_6 be formulas containing no local symbols. Then the digest contains only G_6, but the algorithm from [11] would construct the interpolant $G_3 \wedge \neg G_6$, which also contains G_3.

As the main contribution of [8], the authors extend algorithm $\mathcal{SE}$ such that it also considers a space of different interpolants and optimise over this space. We can see that the extension simulates different choices of splitting function by merging proof steps. Both the algorithm from [8] and our Algorithm 4 encode the space of candidates and the minimisation objective as a pseudo-boolean constraint problem and then ask an optimising SMT-solver for an optimal solution. While encoding the space of splitting functions is trivial using Algorithm 4, encoding the space of local proofs, which are results from repeated pairwise merging of inferences, is much more involved. More critically, while we can make use of Lemma 19 to predict the size of the resulting interpolant, the approach from [8] uses a notion of so called digest to predict the size of the interpolant computed from the transformed proof. The authors claim that the interpolant is a boolean combination of formulas in the digest (Theorem 3.6, [8]). Unfortunately, this claim is wrong, which can be concluded from the counterexample presented in Fig. 5. Therefore the technique presented in [8] can potentially yield sub-optimal interpolants.

As the second concept, the algorithm $\mathcal{SE}$ from [11] relies on a recursive construction to compute the interpolant: The construction computes for each largest subderivation a formula such that the formula of the outermost call yields an interpolant (cf. Lemma 10 of [11]). We now want to hint at the relation of algorithm $\mathcal{SE}$ and Algorithm 1. Consider a subderivation with premises $F_1, \dots, F_k$ and conclusion F. Let further $I_1, \dots, I_k$ denote the recursively computed formulas. Algorithm $\mathcal{SE}$ now constructs the following formulas:

- Case A: $I = ((I_1 \vee F_1) \wedge \dots \wedge (I_k \vee F_k)) \wedge \neg(F_1 \wedge \dots \wedge F_k)$.
- Case B: $I = ((I_1 \vee F_1) \wedge \dots \wedge (I_k \vee F_k))$.

It is not difficult to see that one can reformulate the construction of $\mathcal{SE}$ as the following one, which we will refer to as $\mathcal{SE}'$:

- Case A: $I = ((I_1 \vee F_1) \wedge \dots \wedge (I_k \vee F_k)) \vee F \wedge ((F_1 \wedge \dots \wedge F_k) \rightarrow F)$.
- Case B: $I = (I_1 \wedge \dots \wedge I_k)$.

Note that although the intermediate formulas of algorithm $\mathcal{SE}$ and $\mathcal{SE}'$ are potentially different, the result of the outermost call is the same for $\mathcal{SE}$ and $\mathcal{SE}'$.

We now state a recursive presentation of our Algorithm 1 in order to compare it to $\mathcal{SE}'$. The idea is to replace the global view on the refutation, i.e. the iteration over all elements of $\mathrm{Out}(P, \mathcal{S})$, by a recursive construction which collects all the formulas describing the boundaries of maximal A-subderivations.

Let P be a local proof of a formula F and let r be the inference which derives F. Let further $\mathcal{S}$ be a local splitting function on P. We compute a formula using the following recursive construction: Let $F_1, \ldots, F_k$ denote the elements of $\mathrm{Dep}(F)$ and let I_i denote the formula computed recursively from F_i.

- Case $\mathcal{S}(r) = \mathcal{A}$: $I = (I_1 \wedge \cdots \wedge I_n) \wedge ((F_1 \wedge \cdots \wedge F_n) \rightarrow F)$.
- Case: $\mathcal{S}(e) = \mathcal{B}$: $I = (I_1 \wedge \cdots \wedge I_n)$.

If we now compare algorithm $\mathcal{SE}'$ and the recursive presentation of Algorithm 1 and see that they are the same with the exception that $\mathcal{SE}'$ contains redundant sub-formulas. More critically, since we know that Algorithm 1 yields an interpolant of size which is worst-case-quadratic in the size of the proof, we know that the same holds for $\mathcal{SE}'$ and therefore for $\mathcal{SE}$, i.e. for the interpolation algorithm of [11]. This represents the most important downside of the approach of [11] and makes it inferior to Algorithm 2.

6 Experimental Results

We implemented Linear-splitting-formula (Algorithm 2, Sect. 3) in the automated theorem prover VAMPIRE [12] and combined it with the two approaches for obtaining a local splitting function: the Top-down-weighted-sum-heuristic (Algorithm 3) and the Weighted-sum-optimal (Algorithm 4). In this experiment, we focus on evaluation the latter combination and refer to it as `LinOpt`. The aim of the experiment is to compare the performance of the new algorithm to algorithm from [11] equipped with its optimising improvement from [8], which were already implemented in a previous version of the prover. We will from now on refer to this combination as `SEOpt`. We are mainly interested in learning to which extent do the obtained interpolants differ in size, but also in the time needed to obtain an interpolant by each algorithm.

To compensate for the lack of a representative set of benchmarks explicitly focusing on first-order interpolation, we made use of the first-order problems from the TPTP library [21] (version 6.4.0). We clausified each problem using VAMPIRE and split the obtained set of clauses into halves, treating the first half as $\mathcal{C}_A$ and the the second as $\mathcal{C}_{\neg B}$. We attempted to refute each of the obtained problems using VAMPIRE (which was instructed to generate only local proofs as described in [11]) and followed up by one of `LinOpt` or `SEOpt` to compute an interpolant. We imposed a 60 s time limit on the proof search in VAMPIRE and a total limit of 100 s on each whole run. We ran the experiment on the StarExec compute cluster [20].

In total, we obtained 7442 local refutations. Out of these `SEOpt` failed to construct an interpolant in 723 cases. In contrast, `LinOpt` failed to construct an interpolant in only 16 cases. Furthermore, there were 353 cases in which `SEOpt` returned only an approximate result and 108 cases where optimisation failed and the simple unoptimized version of [11] was used as a fallback instead. Although the observed higher computational demands of `SEOpt` can be partly ascribed to reliance on a different pseudo-boolean solver, we would like to point out that the optimisation problem `SEOpt` constructs is arguably much more complex than the one stemming from Weighted-sum-optimal employed by `LinOpt`.

Fig. 6 contains a scatter plot comparison of the sizes of obtained interpolants for `SEOpt` and `LinOpt`. When either `SEOpt` or `LinOpt` failed to provide an interpolant an artificial large value was substituted which is reflected by the data points on the

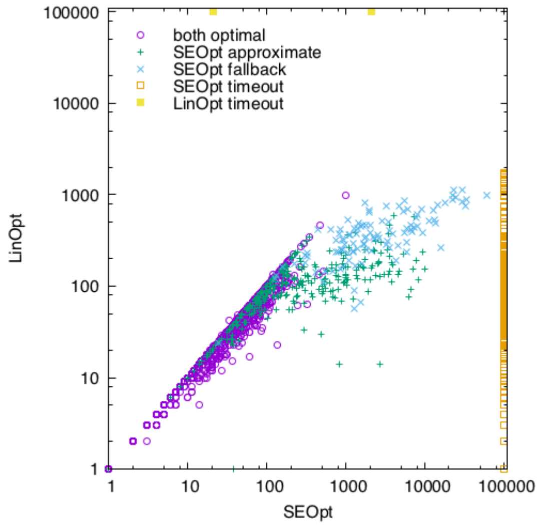

Fig. 6. Size comparison of interpolant produced by `SEOpt` and `LinOpt`. Each point corresponds to a single refutation and its position to the sizes of the respective interpolants.

right and the upper border, respectively. The plot further separates the points to categories based on the optimality guarantee provided by `SEOpt`. We can see that `LinOpt` consistently yields better results. Moreover, the improvement tends to get more pronounced with the growing size of the instances. Finally, even when just focusing on instances where `SEOpt` finished optimising, there are numerous cases where the interpolant from `LinOpt` is several times smaller than that of `SEOpt`. This is because `SEOpt` cannot avoid repeating certain formulas from the refutation many times in the interpolant and corresponds to the worst case quadratic complexity discussed in Sect. 5.

Given the encouraging results we intend to officially replace `SEOpt` by `LinOpt` in VAMPIRE and make it available with the next release of the prover.

7 Conclusion

We presented a new technique for constructing interpolants from first-order local refutations. The technique is based on an idea of proof splitting and on a novel non-inductive construction which arguably gives more insight than previous work and yields interpolants of linear size. This leads to a new interpolation algorithm which we implemented in the automated theorem prover VAMPIRE. Finally, we confirmed in an extensive experiment that the algorithm also improves over the state-of-the-art in practice.

References

1. F. Alberti, R. Bruttomesso, S. Ghilardi, S. Ranise, and N. Sharygina. Lazy abstraction with interpolants for arrays. In *Logic for Programming, Artificial Intelligence, and Reasoning - 18th International Conference, LPAR-18, Mérida, Venezuela, March 11-15, 2012. Proceedings*, vol. 7180 of *Lecture Notes in Computer Science*, pp. 46–61. Springer, 2012.

2. M. P. Bonacina and M. Johansson. On Interpolation in Automated Theorem Proving. *J. Autom. Reasoning*, 54(1):69–97, 2015.
3. J. Christ and J. Hoenicke. Instantiation-based interpolation for quantified formulae. In *Decision Procedures in Software, Hardware and Bioware, 18.04. - 23.04.2010*, vol. 10161 of *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2010.
4. J. Christ and J. Hoenicke. Proof Tree Preserving Tree Interpolation. *J. Autom. Reasoning*, 57(1):67–95, 2016.
5. A. Cimatti, A. Griggio, and R. Sebastiani. Efficient interpolant generation in satisfiability modulo theories. In *TACAS*, vol. 4963 of *LNCS*, pp. 397–412. Springer, 2008.
6. W. Craig. Linear reasoning. A new form of the herbrand-gentzen theorem. *J. Symb. Log.*, 22(3):250–268, 1957.
7. G. Gallo, G. Longo, S. Nguyen, and S. Pallottino. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2):177–201, 1993.
8. K. Hoder, L. Kovács, and A. Voronkov. Playing in the grey area of proofs. In *Principles of Programming Languages*, pp. 259–272. ACM, 2012.
9. R. Jhala and K. L. McMillan. A practical and complete approach to predicate refinement. In *TACAS*, vol. 3920 of *LNCS*, pp. 459–473. Springer, 2006.
10. L. Kovács and A. Voronkov. Interpolation and symbol elimination. In *CADE*, vol. 5663 of *LNCS*, pp. 199–213. Springer, 2009.
11. L. Kovács and A. Voronkov. Interpolation and symbol elimination. In *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings*, vol. 5663 of *Lecture Notes in Computer Science*, pp. 199–213. Springer, 2009.
12. L. Kovács and A. Voronkov. First-order theorem proving and Vampire. In *CAV 2013*, vol. 8044 of *Lecture Notes in Computer Science*, pp. 1–35, 2013.
13. S. K. Lahiri and K. K. Mehra. Interpolant based decision procedure for quantifier-free Presburger arithmetic. Technical Report MSR-TR-2005-121, Microsoft Research, 2005.
14. K. L. McMillan. Interpolation and SAT-based model checking. In *CAV*, vol. 2725 of *LNCS*, pp. 1–13. Springer, 2003.
15. K. L. McMillan. Lazy abstraction with interpolants. In *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, vol. 4144 of *Lecture Notes in Computer Science*, pp. 123–136. Springer, 2006.
16. K. L. McMillan. Quantified invariant generation using an interpolating saturation prover. In *TACAS*, vol. 4963 of *LNCS*, pp. 413–427. Springer, 2008.
17. A. Nonnengart and C. Weidenbach. Computing small clause normal forms. In *Handbook of Automated Reasoning (in 2 volumes)*, pp. 335–367. Elsevier and MIT Press, 2001.
18. A. Podelski, M. Schäf, and T. Wies. Classifying Bugs with Interpolants. In *TAP*, vol. 9762 of *LNCS*, pp. 151–168, 2016.
19. G. Reger, M. Suda, and A. Voronkov. New techniques in clausal form generation. In *GCAI 2016. 2nd Global Conference on Artificial Intelligence*, vol. 41 of *EPiC Series in Computing*, pp. 11–23. EasyChair, 2016.
20. A. Stump, G. Sutcliffe, and C. Tinelli. StarExec, a cross community logic solving service. `https://www.starexec.org`, 2012.
21. G. Sutcliffe. The TPTP problem library and associated infrastructure. *J. Autom. Reasoning*, 43(4):337–362, 2009.
22. N. Totla and T. Wies. Complete instantiation-based interpolation. In *Principles of Programming Languages*, pp. 537–548. ACM, 2013.