FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Interpolation and Local Proofs

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Bernhard Gleiss
Matrikelnummer 0904987

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Univ.Prof. Dr. Laura Kovács

Wien, 3. August 2016

Bernhard Gleiss	Laura Kovács

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Interpolation and Local Proofs

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computational Intelligence

by

Bernhard Gleiss
Registration Number 0904987

to the Faculty of Informatics
at the TU Wien

Advisor: Univ.Prof. Dr. Laura Kovács

Vienna, 3rd August, 2016

Bernhard Gleiss	Laura Kovács

Erklärung zur Verfassung der Arbeit

Bernhard Gleiss
Kasten 123, 3072 Kasten

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 3. August 2016

Bernhard Gleiss

Abstract

Propositional interpolation has become the corner stone of many contemporary model checking algorithms. Consequently, the construction of interpolants has received ample of attention, resulting in a range of interpolation algorithms which derive interpolants from proofs. One such approch is to generate interpolants from proofs of a special form, i.e. from local proofs.
As first contribution of this thesis, we give a new interpolation algorithm for local proofs in an arbitrary sound proof system in first-order logic, which improves the state of the art by generating smaller and simpler interpolants using an arguably more intuitive approach than existing algorithms.
Afterwards we look at propositional interpolation: As second contribution, we state a proof transformation, which transforms an arbitrary propositional resolution into a special local proof, from which we can trivially extract an interpolant. Finally we derive an interpolation system as abstraction of the transformation, which enables us to efficiently compute the interpolant by avoiding to explicitly perform the proof transformation.

Kurzfassung

In den letzten Jahren hat sich Propositionale Interpolation als einer der Hauptbestandteile der Modellprüfung etabliert. Aus diesem Grund wurde der Konstruktion von Interpolanten große Aufmerksamkeit gewidmet, wodurch eine Vielzahl an Algorithmen, welche Interpolanten aus Beweisen extrahieren, entstand.
Einer dieser Ansätze beruht auf der Idee, Interpolanten aus lokalen Beweisen zu konstruieren.
Der erste Beitrag dieser Masterarbeit besteht darin, dass wir einen neuen Interpolationsalgorithmus beschreiben, der Interpolanten aus lokalen Beweisen, die in beliebigen korrekten Beweissystemen in Prädikatenlogik formuliert sind, extrahiert. Der neue Algorithmus verbessert den Stand der Technik, indem er kleinere und strukturell einfachere Interpolanten als bestehende Algorithmen generiert und dabei einen intuitiveren Ansatz verwendet.
Danach wenden wir uns der Interpolation in Propositionaler Logik zu: Als zweiten Beitrag beschreiben wir eine Beweistransformation, die einen propositionalen Resolutionsbeweis in einen derartigen lokalen Beweisen transformiert, dass wir auf triviale Weise einen Interpolanten extrahieren können. Weiters leiten wir aus der Transformation ein Interpolationssystem ab, das es ermöglicht, den Interpolanten effizient zu berechnen, indem durch Abstraktion die explizite Berechnung der Transformation vermieden wird.

Contents

CHAPTER 1

Introduction

1.1 Problem Description

Starting with the pioneering work of McMillan [1], interpolation became a powerful approach in verification thanks to its use in predicate abstraction and model checking [2, 3, 4, 5]. A number of techniques have been proposed to compute interpolants in various logical theories, such as propositional logic [6, 7, 8, 9], quantifier-free fragments of first-order logic [10, 11, 12, 13] or first-order logic with theories [14, 15].

Despite the big interest in interpolation, the connections between those algorithms are not well understood. This causes several problems:

- it is not clear which interpolation algorithm to use in applications of model checking, since comparing the existing interpolation algorithms is at least very involved.
- reading into the rich literature on interpolation is complicated, since similarities and differences are not worked out already.
- many optimizations on proofs and interpolants are discovered independently multiple times, since knowledge transfer is limited due to the different representations of interpolants and proof calculi.
- the state-of-the-art stagnates, since improvements become very involved due to the overcomplicated presentation of the interpolation algorithms.

It is therefore desirable to express the current state-of-the-art algorithms using an intuitive, simple, but nonetheless general framework. The claim of this master thesis is that so called local proofs [15, 3] can be used as such a framework. In order to substantiate this claim, we give a new interpolation algorithm, which implicitly constructs local proofs and show afterwards how to derive an interpolation system from it. We have strong evidence that the same idea can be applied to McMillan's interpolation algorithm [1],

which implies that we can merge the previously separated research lines of interpolation systems and local proofs into a unified theory of interpolation.
In order to yield a strong and useful framework, the extraction of interpolants from local proofs should be both efficient and intuitive. This master thesis therefore also improves the state-of-the-art by proposing a new interpolation algorithm, which is conceptually simpler than the current state-of-the-art-algorithm from [15], but nonetheless yields smaller interpolants.

1.2 Structure of the thesis

This thesis is organised as follows: We start with recalling background material and fixing notations in chapter 2. Then we study interpolation from local proofs in chapter 3, where we present an intuitive introduction, our interpolation algorithm, an optimization of the stated algorithm and finally a general framework based on local proofs. We turn to propositional interpolation in chapter 4 and state a proof transformation which transforms arbitrary propositional resolution refutations into a special local proof, from which we can trivially extract the interpolant. Afterwards we use an abstraction to derive an interpolation system from the transformation, which efficiently computes the resulting interpolant of the algorithm. We discuss related work in chapter 5 and finally conclude in chapter 6.

CHAPTER 2

Preliminaries

2.1 Graphs

2.1.1. Definition A *labelled directed graph* G is a pair (V, E), where V is a finite set of objects and E is a set of triples (e_1, e_2, l), such that $e_1 \in V$, $e_2 \in V$ and $l \in \mathbb{N}$.
For an edge (e_1, e_2, i), we call e_1 the *start node* of e, e_2 the *end node* of e and l the *labelling* of e.
A *source* is a node v such that there is no edge which has v as end node, a *sink* is a node v such that there is no edge which has v as start node.
A *labelled directed acyclic graph (LDAG)* is a labelled directed graph without cycles.

2.2 Formulas

We work in the standard setting of propositional logic over a set V of propositional variables. We allow the standard boolean connectives and assume that the language contains the logical constants $\top$ and $\bot$. A *literal* is either a variable $v \in V$ or its negation $\overline{v}$. A *clause* is a disjunction of literals.
We denote literals by small letters a, b, x, y, z and clauses by C, possibly with indices. We write clauses as set of literals. Throughout the thesis, arbitrary formulas will be denoted by F, G, possibly with indices.
We write $F_1 \wedge \cdots \wedge F_n \vDash F$ to denote that the formula $F_1 \wedge \cdots \wedge F_n \Rightarrow F$ is a tautology.
Given formulas F, G_1, G_2, we define the result of substituting G_1 in F by G_2 recursively by

$$F[G_1 \leftarrow G_2] = \begin{cases} G_2 & \text{if } F = G_1 \\ \bigwedge_{i \in I}(F_i[G_1 \leftarrow G_2]) & \text{if } F \neq G_1 \wedge F = \bigwedge_{i \in I} F_i \\ \bigvee_{i \in I}(F_i[G_1 \leftarrow G_2]) & \text{if } F \neq G_1 \wedge F = \bigvee_{i \in I} F_i \\ F & \text{otherwise} \end{cases}$$

2.3 Inferences

2.3.1. Definition An n-ary inference r (where $n \geq 0$) is a tuple $(F_1, \ldots, F_n, F, id)$, where $F_1, \ldots, F_n, F$ are formulas and $id \in \mathbb{N}$. It is usually written as

$$\frac{F_1 \qquad \ldots \qquad F_n}{F}\, id\ .$$

The formulas $F_1, \ldots, F_n$ are called the *premises*, F is called the *conclusion* and id is called the *identifier* of r. Note that we use the identifier in order to being able to distinguish between two inferences having the same premises and conclusion.
An *axiom* is an inference with zero premises. An *inference rule* is a set of inferences. An *inference system* is a set of inference rules.

2.3.2. Definition Let IS be an inference system. Let further $\mathcal{T}$ be a set. An *IS-derivation P in $\mathcal{T}$* is a nonempty LDAG with the following properties:

- Each node is an inference, which is contained in IS or an axiom with conclusion $F \in \mathcal{T}$.
- For each edge $e = (r, r', i)$ the conclusion of r is equal to the i-th premise of r'
- For each $r' \in V$ and $i \in \mathbb{N}$, there is at most one edge with both end node r' and labelling i.
- F is contained in A iff there is a node r' with i-th premise F, such that there is no edge which has both end node r' and labelling i.
- F is contained in B iff there is a sink with conclusion F.

Let A be the set of formulas, such that F is contained in A iff there is a node r' with i-th premise F, such that there is no edge which has both end node r' and labelling i. Then A is called the set of *leaves* of P.
Let further B consist of all formulas F, such that there is a sink with conclusion F. Then B is called the set of *roots* of P.
An *IS-proof of F in $\mathcal{T}$* is an IS-derivation having no leaves and a single root F. An *IS-refutation* in $\mathcal{T}$ is a proof of $\bot$ in $\mathcal{T}$.
We write $A \vdash_{\mathcal{T}} B$ to denote that there is a derivation P in $\mathcal{T}$ with leaves A and roots B. In such case we call P a witness for $A \vdash_{\mathcal{T}} B$.

2.3.3. Remark From now on, we use the following notational conventions:

- We omit to denote IS, if it is clear from the context.
- We omit to denote the identifiers of edges if they are clear from the context.
- We use the usual presentation of derivations.
- If $A = \emptyset$, we write $\vdash_{\mathcal{T}} B$ instead of $\emptyset \vdash_{\mathcal{T}} B$ and if $A = \{A_0\}$, we write $A_0 \vdash_{\mathcal{T}} B$ instead of $\{A_0\} \vdash_{\mathcal{T}} B$. Analogously for B and $\mathcal{T}$.

2.3.4. Example Let $A := \{x_1 \vee x_2, \overline{x_1}, \overline{x_2}\}$. Consider the following derivation $P_1 = (V_1, E_1)$, which is a witness for $A \vdash \bot$: Let $V_1 = \{v_1, v_2\}$, where $v_1 = \dfrac{x_1 \vee x_2 \qquad \overline{x_1}}{x_2}$ and $v_2 = \dfrac{x_2 \qquad \overline{x_2}}{\bot}$.

Let $E_1 = \{e\}$, where $e = (v_1, v_2, 1)$.

We will usually present P_1 by $\dfrac{\dfrac{x_1 \vee x_2 \qquad \overline{x_1}}{x_2} \qquad \overline{x_2}}{\bot}$.

Now consider the following proof $P_2 = (V_2, E_2)$, which is a witness for $\vdash_A B$: Let $V_2 = \{v_1, v_2, v_3, v_4, v_5\}$, where v_1 is an axiom with conclusion $x_1 \vee x_2$, v_2 is an axiom with conclusion $\overline{x_1}$, v_3 is an axiom with conclusion $\overline{x_2}$, $v_4 = \dfrac{x_1 \vee x_2 \qquad \overline{x_1}}{x_2}$ and $v_5 = \dfrac{x_2 \qquad \overline{x_2}}{\bot}$.

Let $E_2 = \{e_1, e_2, e_3, e_4\}$, where $e_1 = (v_1, v_4, 1)$, $e_2 = (v_2, v_4, 2)$, $e_3 = (v_4, v_5, 1)$, $e_4 = (v_3, v_5, 2)$.

We will usually present P_1 by $\dfrac{\dfrac{\overline{x_1 \vee x_2} \qquad \overline{\overline{x_1}}}{x_2} \qquad \overline{\overline{x_2}}}{\bot}$.

If we compare those two derivations, we see that we can transform a witness P_1 of $A \vdash_{\mathcal{T}} B$ to a witness of $\vdash_{\mathcal{T} \cup A} B$ by adding for each leaf A_0 an axiom with conclusion A_0 and the corresponding edges to P_1. Vice versa, we can transform a witness P_2 of $\vdash_{\mathcal{T} \cup A} B$ into a witness of $A \vdash_{\mathcal{T}} B$ by deleting all those axioms and corresponding edges from P_2.

2.3.5. Definition An inference $(F_1, \ldots, F_n, F)$ is called *sound*, iff $F_1 \wedge \cdots \wedge F_n \vDash F$. An inference rule is called *sound*, iff it consists only of sound inferences, and an inference system is called *sound*, iff it consists only of sound inference rules.

2.3.6. Theorem Let IS be a sound inference system. Let P be a witness for $A \vdash_{\mathcal{T}} B$. Then we have

$$\mathcal{T} \vDash \bigwedge_{F \in A} \to \bigwedge_{F \in B}.$$

2.3.7. Remark There are two possible ways to show $\vDash A \to B$: Either use

$$\begin{aligned} & (A \wedge \neg B) \vdash \bot \\ \text{implies } & \vDash (A \wedge \neg B) \to \bot \\ \text{implies } & \vDash A \to B \end{aligned}$$

or alternatively use

$$\begin{aligned} & \vdash_{(A \wedge \neg B)} \bot \\ \text{implies } & (A \wedge \neg B) \vDash \bot \\ \text{implies } & \vDash (A \wedge \neg B) \to \bot \\ \text{implies } & \vDash A \to B \end{aligned}$$

Note the usage of the deduction theorem in the second variant.

2.3.8. Definition We use a strict ordering $\prec$ on inferences, which is induced by the LDAG, as follows: for any inferences r, r', we define $r \prec r'$ iff there is a non-empty path in the DAG from r to r'. If $r \prec r'$ we say that r is *above* r' and r' is *below* r. We will traverse proofs topologically and while doing so we will sometimes identify a conclusion of an inference with the inference itself, resulting in visiting the conclusions topologically. Note that this identification is possible, since each formula in a proof is a conclusion of an inference (in contrast to arbitrary derivations).

2.3.9. Definition Let $P = (V, E)$ be an IS-derivation in $\mathcal{T}$. Let $P' = (V', E)$ be an LDAG, such that $V' \subseteq V$ and let $E' = \{(v_1, v_2, i) \in E \mid v_1 \in V' \wedge v_2 \in V'\}$. Then the derivation P' in $\mathcal{T}$ is called a *subderivation of* P.

2.3.10. Remark Note that a subderivation $P' = (V', E')$ of a proof $P = (V, E)$ is not necessarily a proof itself (if not all axioms of V are included in V').

2.3.11. Definition Let $P = (V, E)$ be a derivation and let S be a unary predicate on inferences. Then the set of maximal subderivations $M(P, S)$ of P with respect to S consists of the smallest set of subderivations, such that we have:

- For each inference $v \in V$ with $S(v) = 1$, there exists a subderivation $M_0 = (V_0, E_0) \in M(V, V')$ such that $v \in V_0$
- For each subderivation $M_0 = (V_0, E_0) \in M(V, V')$ we have:
  - For each inference $v_0 \in V_0$, we have $S(v_0) = 1$.
  - For all $v_1 \in V$ we have $v_0 \in V_0 \wedge (v_0, v_1) \in E \wedge S(v_1) = 1$ implies $v_1 \in V_0$.

2.3.12. Definition Let $P = (V, E)$ be a derivation containing sinks $R = (r_1, \ldots, r_n)$ with conclusions $F_1, \ldots, F_n$ and let r be an inference with premises $F_1, \ldots, F_n$.
The *result of adding* r *to* P, written $P +^R r$, is defined as (V', E'), where

$$V' = V \cup \{r\}$$
$$E' = E \cup \{(r_1, r, 1), \ldots, (r_n, r, n)\}$$

2.3.13. Definition Let $P_1 = (V_1, E_1)$ be a proof with roots B_1 and let $P_2 = (V_2, E_2)$ be a derivation with roots B_2. Let P_1 contain sinks $R = \{r_1, \ldots, r_n\}$ with conclusions $G = \{G_1, \ldots, G_n\}$. Let P_2 have leaves $G_1, \ldots, G_n$. For each such leaf G_i, let $e_i = (v_1, v_2, j)$ be a triple, such that $v_1 \in R$, the conclusion of v_1 is G_i, $v_2 \in V_2$ and v_2 has G_i as j-th leaf.
The *concatenation of* P_1 *and* P_2 *on* G, written as $P_1 +^G P_2$ is defined as $P = (V, E)$, where

$$V = V_1 \cup (V_2)$$
$$E = E_1 \cup E_2 \cup \{e_i | G_i \in G\}$$

Note that $P_1 +^R P_2$ is a proof with roots $(B_1 \setminus G) \cup B_2$.

2.3.14. Definition Let $P = (V, E)$ be a derivation containing inferences $R = r_1, \ldots, r_n$ whose conclusions are respectively $F_1, \ldots, F_n$, such that the elements of R are pairwise incomparable regarding $\prec$. The subderivation *ending* in R is defined as the derivation $P' = (V', E')$, where

$$V' = R \cup \{j \mid \exists r_i \in R : j \prec r_i\}$$
$$E' = \{(j, j', i) \mid j \in V' \wedge j' \in V' \wedge (j, j', i) \in E\}$$

The subderivation *starting* in R is defined as the derivation $P' = (V', E')$, where

$$V' = \{j \mid \exists r_i \in R : r_i \prec j\}$$
$$E' = \{(j, j') \mid j \in V' \wedge j' \in V' \wedge (j, j') \in E\}$$

2.3.15. Definition Let P be a derivation. We write $P[G_1 \leftarrow G_2]$ to mean the derivation obtained from P by replacing each inference

$$\frac{F_1 \qquad \ldots \qquad F_n}{F}$$

in P with a new inference

$$\cdot\frac{F_1[G_1 \leftarrow G_2] \qquad \ldots \qquad F_n[G_1 \leftarrow G_2]}{F[G_1 \leftarrow G_2]}$$

2.4 Normal Forms and Resolution

A formula F is in *conjunctive normal form* (CNF) if it is a conjunction of clauses. We represent CNF formulas as a set of clauses. A formula F is in *negation normal form* (NNF) if it contains only conjunctions, disjunctions, and negations, and negation in F is only applied on literals. For simplicity, throughout this thesis we assume that conjunctions (respectively, disjunctions) have arity greater than 2 and no conjunction (respectively, disjunction) in an NNF formula F has a conjunction (respectively, disjunction) as an immediate subformula. Given a formula F in NNF, the *alternation level* of F is $1 + max_{\wedge,\vee}$, where $max_{\wedge,\vee}$ is the number of maximal alternations between conjunctions and disjunctions in F. Note that every formula F can be translated into an equivalent formula F' such that F' is in CNF, respectively in NNF.

We next introduce the resolution inference system first for formulas in CNF, and then for NNF formulas.

CNF resolution. The *resolution inference system on propositional CNF formulas*, referred to as *CNF resolution*, consists of the following inference rule:

$$\frac{C_1 \vee v \qquad C_2 \vee \overline{v}}{C_1 \vee C_2}\, v \text{ [resolution]}$$

Given a clause C and a propositional variable $v \in V$, we write Cv to denote the clause $C \vee v$. Let $F = \{C_1 v, \ldots, C_n v\}$ be a set of clauses containing v and $F' = \{C'_1 \overline{v}, \ldots, C'_m \overline{v}\}$ be a set of clauses containing $\overline{v}$. We define the *pairwise resolution* of F and F', denoted as $F \otimes^v F'$, as the formula:

$$\bigwedge \{C_i \vee C'_j : 1 \leq i \leq n, 1 \leq j \leq m\}.$$

We now introduce a modification of the CNF resolution inference system, called *CNF resolution with redundancy (rCNF)*. For doing so, we replace the resolution rule of CNF resolution with inference rules whose conclusions are redundant but ensure that rCNF is closed under substitution of clauses. The (modified resolution) rules of the rCNF resolution system are as follows:

$$\frac{C_1 \qquad C_2}{(C_1 \setminus \{v\}) \vee (C_2 \setminus \{\overline{v}\})}\, v \qquad \frac{C_1 \qquad C_2}{(C_1 \setminus \{v\}) \vee C_2}\, v$$

$$\frac{C_1 \qquad C_2}{C_1 \vee (C_2 \setminus \{\overline{v}\})}\, v \qquad \frac{C_1 \qquad C_2}{C_1 \vee C_2}\, v$$

Note that rCNF is sound and the following holds.

2.4.1. Lemma Let a be a literal and C be an arbitrary clause. Let further P be some derivation in the rCNF resolution system, which contains no resolution inference on a. Then $P[a \leftarrow C]$ is also a derivation in the rCNF resolution system.

2.5 Interpolation

2.5.1. Definition Given an arbitrary propositional formula F, let $\mathrm{Var}(F)$ denote the set of propositional variables occuring in F. Let A, B be two propositional formulas such that $A \Rightarrow B$. In the sequel we assume A and B to be fixed and give all definitions relative to A and B. A variable $v \in \mathrm{Var}(A \Rightarrow B)$ is called *A-local*, iff $v \in \mathrm{Var}(A) \setminus \mathrm{Var}(B)$, *$B$-local*, iff $v \in \mathrm{Var}(B) \setminus \mathrm{Var}(A)$ and *global* otherwise. In the following, we denote A-local variables by a, B-local variables by b and global variables by x, possibly with indices.

We now recall the definition of an interpolant:

2.5.2. Definition (Interpolant) An *interpolant* for A, B is a formula I such that $A \Rightarrow I$, $I \Rightarrow B$ and I contains only global variables.

Craig's interpolation theorem [16] guarantees the existence of an interpolant.

2.5.3. Definition (A-axiom, B-axiom) Let A and $\neg B$ be given as set of clauses C_A and C_B. For a refutation in $C_A \cup C_B$, an *A-axiom* (respectively, *B-axiom*) is an axiom C such that $C \in C_A$ (respectively, $C \in C_B$).

CHAPTER 3

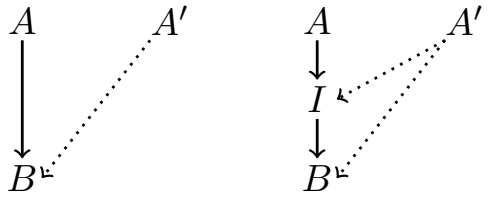
Interpolants from refutations

In this chapter, we want to motivate interpolants and show how to derive them from refutations. We will see that we are actually interested in splitting a given proof into two parts and that interpolants are exactly the formulas corresponding to the boundaries induced by splittings of a special form, which arise very naturally.

3.1 Splits

We start with intuitively motivating splitting a proof: Consider the situation where we repeatedly want to show the validity of implications, e.g. model checking. Assume we first want to show $A \to B$. We do this by asking a prover to find a proof of $A \to B$. If the prover finds such a proof, it returns the fact that there is a proof and we conclude that the implication is valid. Note that by doing so we actually only remember the existence of the proof, but discard the proof itself. When we later want to prove an implication $A' \to B$, we do the same procedure again.

The main idea of proof-splitting is now that we could do better in such a situation by remembering some part of the proof of the first implication in order to simplify the proof of the second implication: If we split the proof of $A \to B$ into two lemmas, i.e we prove $A \to I$ and $I \to B$ for some I, which corresponds to an intermediate stage in the proof (which we call split), forget about the proofs itself, but remember their existence and furthermore I, then proving $A' \to B$ is simplified, since we either can prove $A' \to B$ or prove $A' \to I$, which we can then concatenate with the second half of the first proof in order to get a proof of $A' \to B$. Actually it is even sufficient to prove $A' \to (I \vee B)$ and this is usually easier than proving $A' \to B$.



Note that a similar approach is very common in mathematics: after generating any proof one tries to factor out useful parts of the proof into lemmas in order to be able to use them in later proofs.

We now show how to combine this ideas with refutations: For the rest of this chapter, let again A and B be two fixed formulas with $A \Rightarrow B$ and let C_A and C_B denote clause sets corresponding to A and $\neg B$. Let further P be a refutation in $C_A \cup C_B$.
It is important to note that splitting a proof into two parts means assigning each inference to one of the parts. In the other direction, each such assignment induces a splitting of the proof into two parts.

It is natural to assign the A-axioms to the A-part and the B-axioms to the B-part, therefore we only consider assignments of this form from now. All other inferences can be performed by both parts, so we can choose freely how to assign them.

3.1.1. Definition Let P be a sound refutation in $C_A \cup C_B$.

- A *splitting function* is a function assigning each of the inferences of P to either the A-part or the B-part, such that each A-axiom is assigned to A-part and each B-axiom is assigned to the B-part.
- For a given splitting function f, let S denote the predicate which returns 1 iff $f(r) = A$. Now let $M(P, S)$ denote the set of maximal subderivations and call each element of $M(P, S)$ an *A-subderivation*.

Each splitting function naturally induces a formula describing the boundaries between the A-part and the B-part, the so called splitting formula: for each A-subderivation, we construct the formula 'the conjunction of the premises implies the conjunction of the conclusions' and then conjoin all those formulas.

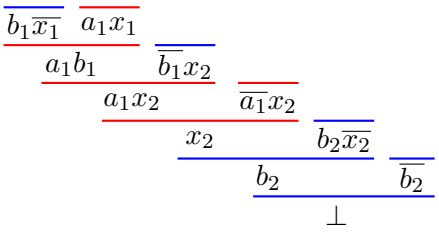
3.1.2. Definition Let P be some refutation in a sound proof system and let f be a splitting function for P. For each A-subderivation M_i, let In_i denote the set of leaves of M_i[1] and Out_i denote the set of roots of M_i. Let

$$I := \bigwedge_{i=1}^{n} ((\bigwedge_{C \in In_i} C) \rightarrow (\bigwedge_{C \in Out_i} C)).$$

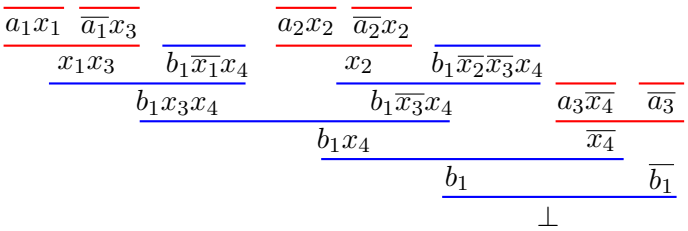
Then I is called *splitting formula* of P.

[1] note again that axioms don't have premises!

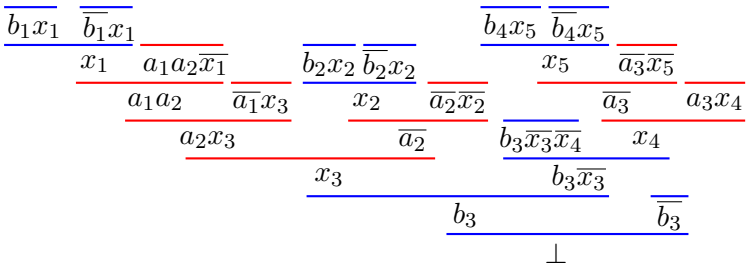
3.1.3. Example Consider the following proof: We choose the splitting function, which is induced by the colors of the inferences: red inferences are assigned to the A-part and blue inferences are assigned to the B-part. We have only one A-subderivation, with leaves $b_1 \vee \overline{x_1}$, $\overline{b_1} \vee x_2$ and root x_2. Therefore the induced split is $((b \vee \overline{x}) \wedge (\overline{b_1} \vee x_2)) \to x_2$.



3.1.4. Example In this example we have three A-subderivations. The A-subderivation in the left part of the proof has no premises and conclusion $x_1 \vee x_3$, the A-subderivation in the middle has no premises and conclusion x_2 and the A-subderivation at the right has no premises and conclusion $\overline{x_4}$. Therefore the induced split is $(\bot \to (x_1 \vee x_3)) \wedge (\bot \to x_2) \wedge (\bot \to \overline{x_4})$, which is equivalent to $(x_1 \vee x_3) \wedge x_2 \wedge \overline{x_4}$



3.1.5. Example In this example we have two A-subderivations. The left A-subderivation has premises x_1, x_2 and conclusion x_3, the right A-subderivation has x_5 as premise and conclusion x_4. Therefore the induced split is $((x_1 \wedge x_2) \to x_3) \wedge (x_5 \to x_4)$



Note that the splitting formula I encodes the use of the A-subderivations in the proof, so on the one hand A implies I and on the other hand we can use I to replace the A-subderivations, so we know that using I and B we can proof $\bot$:

3.1.6. Theorem Let P be a refutation in $C_A \cup C_B$ in a sound proof system and let f be a splitting function for P. Let further I be the splitting formula induced by f. Then we have both

$$A \models I \text{ and } I \models B.$$

Proof :

- We start by proving $A \models I$: For each $M_i \in M(P, f)$ we know that all axioms of M_i are A-axioms, so we have $In_i \vdash_{C_A} Out_i$. Since the proof system is sound, we therefore get $C_A \models (\bigwedge_{C \in In_i} C) \to (\bigwedge_{C \in Out_i} C)$, which by definition of C_A implies $A \models (\bigwedge_{C \in In_i} C) \to (\bigwedge_{C \in Out_i} C)$. Repeating this argument for each $M_i \in M(P, f)$, we get $A \models \bigwedge_{i=1}^{n} (\bigwedge_{C \in In_i} C) \to (\bigwedge_{C \in Out_i} C)$.
- We now prove $I \models B$: We enrich the proof system by modus ponens (and get again a sound proof system), and then replace each M_i by an application of modus ponens with premises $\bigwedge_{C \in In_i} C$, $(\bigwedge_{C \in In_i} C) \to (\bigwedge_{C \in Out_i} C)$ and conclusion $\bigwedge_{C \in Out_i} C$. If we replace all M_i in this way, we get a proof using only B-axioms and axioms of the form $(\bigwedge_{C \in In_i} C) \to (\bigwedge_{C \in Out_i} C)$ for some M_i. In other words, we have $\vdash_{C_B \cup \{(\bigwedge_{C \in In_i} C) \to (\bigwedge_{C \in Out_i} C) | 1 \leq i \leq n\}} \bot$. Using the soundness of the enriched proof system, we get $(\bigwedge_{C \in C_B} C) \wedge \bigwedge_{i=1}^{n} ((\bigwedge_{C \in In_i} C) \to (\bigwedge_{C \in Out_i} C)) \models \bot$, which by construction of C_B implies $\neg B \wedge \bigwedge_{i=1}^{n} ((\bigwedge_{C \in In_i} C) \to (\bigwedge_{C \in Out_i} C)) \models \bot$. From this we finally get $\bigwedge_{i=1}^{n} ((\bigwedge_{C \in In_i} C) \to (\bigwedge_{C \in Out_i} C)) \models B$ by using the deduction theorem.

3.1.7. Algorithm Assume we have a proof and a corresponding splitting function. We now want to find the subderivations which are maximal with respect to the property that each inference is assigned to A. We traverse the proof bottom up and use the idea of union-find to efficiently compute those maximal subderivations. When we have found the subderivations, we again traverse the proof in order to compute the split-formulas associated with each A-subderivation. Implementing this idea results in Algorithm 1.

3.1.8. Theorem Let P be some refutation. Then Algorithm 1 computes a splitting formula.

Proof

We only have to show that the algorithm correctly computes the In- and Out-sets of the maximal A-subproofs:

- In the first traversel, the algorithm computes for each subproof a unique root-clause by using standard ideas of union-find.
- In the second traversel, it adds to each subproof all the In- and Out-clauses: A clause is in the In-set of a maximal A-subproof, if it both is the premise of an inference assigned to the A-subproof and has a parent-inference, which is assigned to the B-subproof.

  A clause is in the Out-set of a maximal A-subproof, if it is the conclusion of an inference having a child-inference, which is assigned to the B-part or if it is $\bot$. For the first case, it suffices to go through all inferences assigned to the B-part and for

Algorithm 1 extractSplit(Proof P)

```
init int-array parent with parent[i] = i
init mapping int → List of ints called In
init mapping int → List of ints called Out
v = partitionizeProof(P)
for each inference i of P(bottom-up) do
    if v[i] then
        for each parent j of i do
            if v[j] then
                rep = root of parent[j] (optimization: apply path compression)
                parent[rep] = i (optimization: weighted)
            end if
        end for
    end if
end for
for each inference i of P (bottom up) do
    if v[i] then
        rep = root of parent[i]
        for each parent j of i do
            if ¬v[j] then
                append conclusion[j] to In[rep] if not already there
            end if
        end for
    else
        for each parent j of i do
            if v[j] then
                rep = root of parent[j]
                append conclusion[j] to Out[rep] if not already there
            end if
        end for
    end if
    if conclusion[i] = ⊥ then
        if v[i] then
            rep = root of i
            append ⊥ to Out[rep]
        end if
    end if
end for
init empty List of formulas called Interpolant
for each entry i of In/Out do
    F1 = makeAnd(In)
    F2 = makeAnd(Out)
    F3 = makeImplication(F1, F2)
    add F3 to interpolant
end for
return makeAnd(Interpolant)
```

each such inference to check if it has a parent inference, which is assigned to the A-part, in which case we add the conclusion of that parent inference to the Out-set of the corresponding A-subproof. For the second case, we trivially have to check if the unique inference with conclusion $\bot$ is assigned to A and in that case add $\bot$ to the Out-set of the corresponding A-subproof.

3.2 Interpolants as nice splits

We now know how to construct splits given a splitting function. Since there are usually many possible splitting functions, the question is which of them to use. We want the split to be as useful as possible for further proofs, so we want to assign all inferences which are done 'because of A' to the A-part and all inferences which are done 'because of B' to the B-part. Note that this is a very vague description. On the other hand, there are inferences which can syntactically be determined to correspond to A or B: We already added the axioms to the corresponding parts, since they naturally belong to them. The main idea is now that we should also add those inferences which contain local symbols to the corresponding parts. This implies that we need proofs, where no inference contains both A-local and B-local symbols, since otherwise we would be forced to assign an inference to two parts, which is not possible. We call proofs of this form local proofs:

3.2.1. Definition (Local Proof) A proof P in $C_A \cup C_B$ is called a *local proof* if the following condition[2]. hold:

(L) For every inference

$$\frac{C_1 \dots C_n}{C}$$

we have either:

- $Var(C_1) \cup \dots \cup Var(C_n) \cup Var(C) \subseteq Var(A)$ or
- $Var(C_1) \cup \dots \cup Var(C_n) \cup Var(C) \subseteq Var(B)$

Note that not all proofs are local, for instance example 3.1.3 is not local, since there exists an inference step containing both a_1 and b_1.

The definition of local proofs ensures that we can get an assignment, such that all A-local symbols are in the A-part and all B-local symbols are in the B-part: assign all A-axioms and inferences containing A-local symbols to the A-part, all B-axioms and inferences containing B-local symbols to the B-part and all other inferences arbitrarily:

[2] note that we don't need the second condition from [15]

Algorithm 2 partitionizeProofSimple(Proof P)

```
for each inference i of P (any order) do
    if i is an A-axiom or i contains an A-local symbol then
        v[i] = 1
    else
        v[i] = 0
    end if
end for
return v
```

3.2.2. Definition A local splitting function is a splitting function f, such that $f(r) = A$ ($f(r) = B$) for all inferences r having as premise or conclusion a formula containing an A-local (a B-local) symbol.

3.2.3. Lemma Let P be a refutation in $C_A \cup C_B$, let f be a local splitting function and let I be the corresponding splitting formula.

i) Each subproof $M_i \in M(P, S)$ has only clauses without local symbols as leaves and roots.
ii) The splitting formula contains only global symbols.

Proof

i) Let C be a leaf of some M_i and let r be the inference of M_i with premise M_i. Since r is part of M_i, we can conclude $f(r) = A$. By the locality of f we then get that C contains no B-local symbol. By the maximality of M_i, we know that there is an inference r' with conclusion C, such that $f(r') = B$. By the locality of f we then get that C contains no A-local symbol. This concludes the proof that no leaf contains a local symbol. Now let C be a root of some M_i. If C is $\bot$, it trivially contains no local symbol, otherwise we can use an argument analogous to the leaf-case.
ii) Follows immediately from i) and the definition of the splitting formula.

3.2.4. Corollary Let P be a refutation, let f be a local splitting function for P and let I be the corresponding splitting formula.

i) I is an interpolant for A, B.
ii) Algorithm 1 parameterised by algorithm 2 yields an interpolant for A, B.

Proof

i) Follows directly from theorem 3.1.6 and lemma 3.2.3 ii).
ii) Follows directly from i) and theorem 3.1.8.

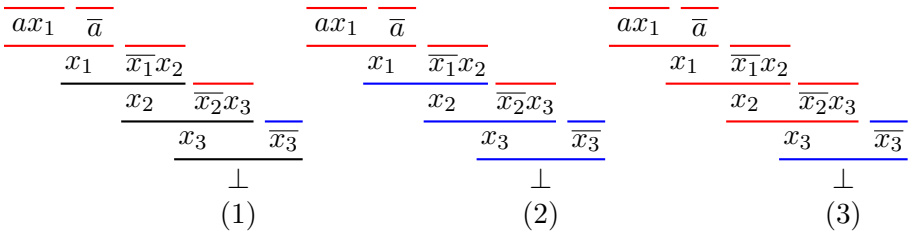
3.3 Splitting the grey area

In practice it does make a difference how to assign the inferences, which are neither axioms nor contain local symbols, so we have to find a good strategy how to split them. There are three approaches:

1. assign all A-axioms and inferences containing A-local symbols to the A-part and assign all other inferences to the B-part. This is usually not optimal, but simple and therefore used by many interpolation algorithms.
2. also assign those inferences to A, which only have parent-inferences, which are assigned to A. The intuition for this approach is that splits tend to be smaller at the bottom of a proof.
3. encode the problem of finding an optimal assignment as minimization problem and pass it to a solver. This yields the optimal assignment, but is computationally expensive.

The following example shows that the approach 1 is not always a good choice:

3.3.1. Example We want to find a splitting for the given refutation. In (1) we have colored the inferences which have to be in the corresponding partitions because they are axioms or contain local symbols. The question is how to assign the other inferences. In (2) we use approach 1 to assign those inferences, i.e. we assign all of them to the B-part. In (3), we use approach 3 to find an optimal splitting (in this example, we use the size of the split as quality measurement). The induced split of (2) is $x_1 \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee x_3)$, the induced split of (3) is x_3, which is much smaller.



We now give a detailed description of approach 3:

Let w_i be weights based on some quality measurement of the clauses (e.g. size, number of quantifiers, ...). We use propositional variables x_i to denote that inference i is assigned to A and use propositional variables L_i to denote that the conclusion of i occurs in the interpolant. Note that a conclusion $C \neq \bot$ of an inference i will be added to the interpolant, iff there exists an inference j, which has C as premise and is assigned to a different part than i. Therefore, by the definition of A-subproofs, each clause will be added to the interpolant at most once, so we get interpolants linear in the size of the proof.

Algorithm 3 partitionizeProofOptimized(Proof P)

for each inference i of P (bottom-up) **do**
  if i is an A-axiom or i contains an A-local symbol **then**
    assert x_i
  else if i is a B-axiom or i contains a B-local symbol **then**
    assert $\neg x_i$
  end if
  for each parent inference j of i **do**
    assert $(x_i \not\leftrightarrow x_j) \rightarrow L(j)$
  end for
end for
minimize $w_1 \cdot L_1 + \cdots + w_n \cdot L_n$
return vector of values of x_i

3.4 Forward local proofs

We now state a simple special case of local proofs, called forward local proof, which was introduced in [17]. Those are interesting, since they are implicitly used by many propositional interpolation algorithms.

3.4.1. Definition

- A proof is called forward local, if it is local and for any inference r_1, which is an A-axiom or contains an A-local symbol and for any inference r_2, which is a B-axiom or contains a B-local symbol, we have $r_2 \not\prec r_1$.
- A splitting function f of a proof P is called forward local, if it is local and for any two inferences r_1, r_2 of P with $f(r_1) = A$ and $f(r_2) = B$, we have $r_2 \not\prec r_1$.

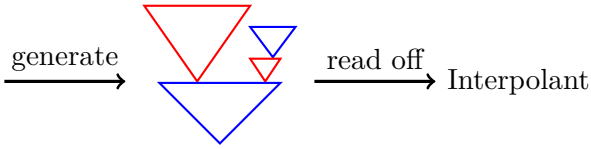
3.4.2. Corollary Let P be a forward local refutation, let f be a forward local splitting function for P and let I be the corresponding splitting formula. Let furthermore $M(P, S) = A_1, \ldots, A_n$ be the A-maximal subproofs and let $Out_{i,j}$ be the j-th output clause of A_i. Then

$$\bigwedge_{i=1}^{n} (\bigwedge_{j} Out_{i,j})$$

is an interpolant of A, B.

3.4.3. Example We already encountered a forward local proof in example 3.1.4.

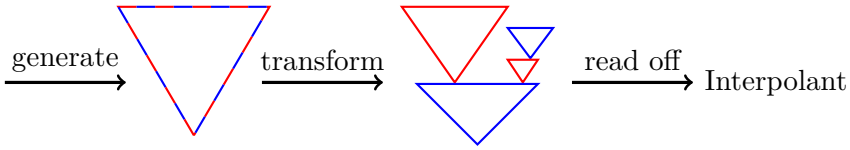
3.5 Framework for interpolation



We now propose the following approach to interpolation:

1. Construct a local refutation in any sound proof system using some proof generating algorithm.
2. Extract an interpolant from this local refutation using the ideas from above.

Since there are cases, where it is not known how to efficiently construct local proofs (e.g. propositional resolution based on Conflict-Driven Clause Learning), we need to expand this approach to the following one:



1. Construct a suitable (not necessarily local) refutation using some proof generating algorithm.
2. Transform the refutation into a local refutation in any sound proof system.
3. Extract an interpolant from this local refutation using the ideas from above.

It is very important to note that the proof system, in which the proof is constructed, is not necessarily the same as the proof system, in which the transformed proof is in. Usually the first system is very restricted in order to efficiently find a proof. This property is not needed for the second proof system, which in contrast should be very expressive in order to being able to transform the first proof into a local proof, which is not much bigger than the first proof.

Since proof transformations are usually computationally expensive, explicitly constructing the transformed proof is problematic. On the other hand, we actually don't need the whole local proof, but only enough information in order to extract the interpolant from it. We therefore can use the idea of only implicitly performing the proof transformation by merging the last two steps together. There is strong evidence, that this idea is applied by the most used propositional interpolation algorithms [1, 18]. We call algorithms, which merge the last two steps, interpolation systems, reusing the name from [19].

The stated approach has the following main advantages:

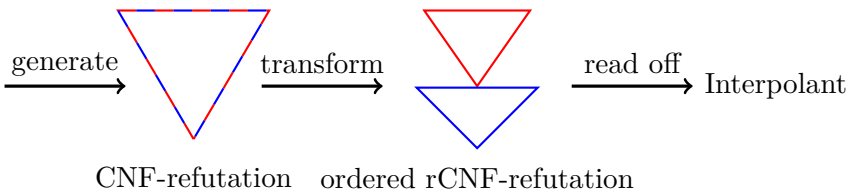
- it is very modular, i.e. we can separately optimize the generation of local proofs and the extraction of interpolants from local proofs.
- We can compare different interpolation algorithms by the local proofs they construct and not only by the interpolants they compute.
- there is strong evidence that every proof-based state-of-the-art interpolation algorithm is expressible using this framework.
- correctness proofs of interpolation algorithms reduce to showing that the proofs, which are generated in the second step, are valid and local.

CHAPTER 4

Propositional Interpolation - literal-reordering

4.1 Ordered resolution proofs

We now apply the ideas from chapter 3 to propositional logic: At first we will introduce the notion of ordered proofs and show that ordered refutations are always forward local. Afterwards we show how to iteratively transform an arbitrary CNF-resolution refutation into an ordered refutation. At the end we show how to perform the transformation implicitly by only keeping track of the so called frontiers while iteratively transforming the proof.



We start with the notion of ordered proofs:

4.1.1. Definition (Ordered rCNF Proof)
Let P be a derivation in $C_A \cup C_B$ in the rCNF-resolution inference system. We say that P is *ordered* if the following condition holds:

(CO) For any resolution inference r_a on an A-local variable a, there exists no resolution inference r on a B-local or global variable such that $r \prec r_a$.

In other words, condition (CO) imposes that no resolution inference on a B-local or global variable happens above any resolution inference on an A-local variable. We now introduce some further vocabulary on ordered derivations:

4.1.2. Definition (Frontier)
Let P be an ordered derivation. An inference of P is called an *A-inference* if it is an A-axiom or a resolution inference on an A-local variable.
Now let S_A be the following set of A-inferences of P:

$$S_A = \left\{ r \middle| \begin{array}{c} r \text{ is an } A\text{-inference and} \\ \text{for every inference } r' \text{ in } P \text{ with } r \prec r': \\ r' \text{ is not an } A\text{-inference} \end{array} \right\}.$$

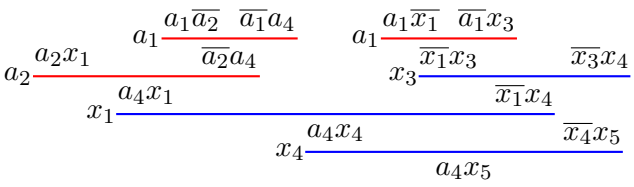
We call $\{C \mid C \text{ is the conclusion of } r \in S_A\}$ the *frontier of P* and denote it by $frontier(P)$.

In other words, $frontier(P)$ contains the conclusions of the last A-inferences of P.

4.1.3. Definition (A-part, B-part)
We define the largest subderivation P' of P ending in the clauses of $frontier(P)$ to be the *A-part* of P, and denote P' by $\mathcal{A}(P)$. We furthermore define the largest subderivation P' of P starting in the clauses of $frontier(P)$ to be the *B-part* of P, and denote P' by $\mathcal{B}(P)$.

Let us illustrate the structure of an ordered proof in Figure 4.1. Note that the A-part of the proof contains only resolution inferences over the A-local variables a_1, a_2, a_3. The B-part of the proof contains resolution inferences over B-local variables b_1, b_2 and global variables x_1, x_2. The frontier of the proof consists of the clauses $C_1, \ldots, C_n$, separating the A-part and B-part of the proof.

4.1.4. Example Let $\{a_2\overline{x_1}, a_1\overline{a_2}, \overline{a_1}a_4, a_1\overline{x_1}, \overline{a_1}x_3, \overline{x_3}x_4\}$ be A-axioms and $\{\overline{x_4}x_5\}$ a B-axiom. The following proof is an ordered proof in the rCNF inference system:



For each resolution step, the proof also displays the resolved variable. The frontier of the proof consists of $a_4x_1, \overline{x_1}x_3, \overline{x_3}x_4$. Note that clauses of the frontier can contain A-local variables (which then also appear in the root of the proof).

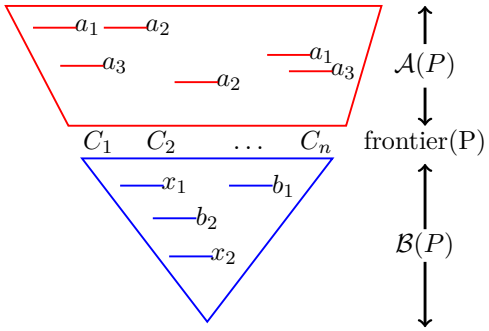


Figure 4.1: Structure of an Ordered Proof.

We will now show that in the propositional resolution inference system, condition (CO) implies that inferences in ordered refutations do not use both A-local and B-local variables. This means, that ordered refutations are (forward) local proofs.

4.1.5. Theorem Let P be an ordered rCNF refutation, such that for each resolution inference on an A-local symbol a both premises contain a. Then P is forward local.

Proof

i) We show that any resolution-inference i with a conclusion containing an A-local symbol a must already have as premises only clauses containing an A-local symbol: Consider any resolution-inference i with a conclusion containing an A-local symbol. We make a case distinction on the resolution literal of i:
   - i resolves on an A-local literal: By the assumptions, both premises must contain an A-local symbol.
   - i resolves on a global or B-local literal: Since the conclusion of i contains an A-local literal a, there must be some other inference j on the way to the root, which resolves on a (since a is not contained in the root, which is $\bot$), so $i \prec j$, which contradicts the fact that all resolution steps on A-local literals occur before any other steps.

ii) Any clause, which contains an A-local symbol, is derived solely from A: We can conclude this fact by a simple induction, the base case for conclusions of axioms is trivial and the inductive step trivially follows from i).

iii) By the definition of resolution-inferences we know that only clauses which are derived using at least one B-axiom contain a B-local symbol. Therefore we can use the invariant from (ii) to conclude that no clause, which contains an A-local symbol, contains also a B-local symbol. Therefore P is local.

iv) Since all clauses containing A-local symbols are derived solely from A, we know that also the conclusions of A-symbol elimininating clauses are derived solely from A. Therefore all maximal A-subproofs are derived without any B-parents, so P is not only local, but even forward local.

As shown in [3], interpolants constructed from local proofs are boolean combinations of clauses in the proof. The following corollary improves this result when working with ordered rCNF-refutations instead of arbitrary local refutations by showing that the interpolant is a conjunction of clauses from the *frontier* of the proof.

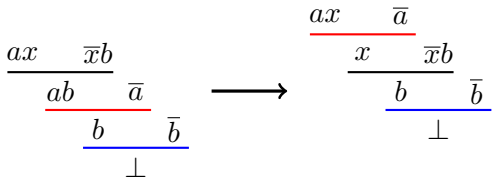
4.1.6. Corollary Let P be an ordered rCNF refutation in $C_A \cup C_B$, such that for each resolution inference on an A-local symbol a both premises contain a. Then $\bigwedge_{C \in frontier(P)} C$ is an interpolant for A, B.

4.2 Transforming Arbitrary Refutations into Ordered Refutations

In the following section, we present a proof transformation, which allows us to reorder the literals by shifting an inference above one or both parent inferences. In [20] it was shown that such a transformation is always possible[1] for propositional resolution refutations. We give a description of a quite similar transformation (which already implicitly occured in [21]). Note that all the different cases described in [20](Fig.5) can be obtained as special cases of our transformation. Furthermore, our transformation can perform multiple reordering steps at once, which allows us to preserve more of the structure of the original proof (cf. the following examples) and also enables our intuitive connection to interpolation systems. Finally our transformation also highlights the connection to the idea of variable elimination, which was introduced in the seminal paper of Putnam & Davis [22].

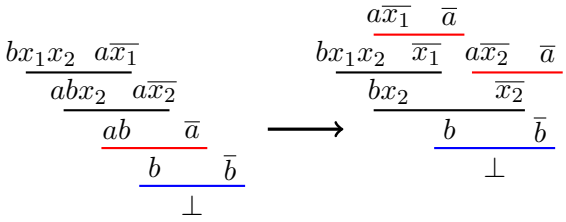
Before stating the transformation, we give some examples, which highlight the challenges of designing such a proof transformation.

4.2.1. Example Consider the following proofs: The left proof corresponds to the original proof and the right one to the proof after the transformation. Since the resolution on a happens below the resolution on x, we have to shift the resolution on a upwards: We move the resolution with the clause $\overline{a}$ to the top, such that the resolution on a happens before the resolution on x.

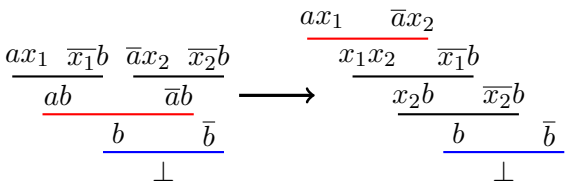


[1] contrary to the statement in [21], which wrongly attributes the claim to [19], where only a transformation for the case of non-merge-literals is proposed

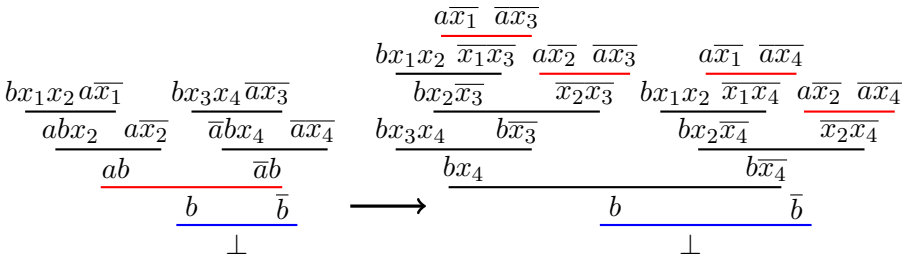
4.2.2. Example As in the previous example, we have to shift the resolution on a upwards. Note that in contrast to the previous example, there are actually two occurences of a which flow into one of the premises of the inference on a. We therefore have to push the clause $\overline{a}$ both to $a\overline{x_1}$ and to $a\overline{x_2}$. This implies that we have to copy $\overline{a}$. Note that such copying can be the source for a blowup, in the worst case the resulting proofs are exponentially bigger.



4.2.3. Example As in the previous example, we have to shift the resolution on a upwards. Note that this example is more involved than the previous one, since not only the subproof rooted in ab has to be transformed, but also the subproof rooted in $\overline{a}b$ has to be transformed. This was not necessary in the previous examples, where the second subproof consisted of the single clause $\overline{a}$. Note that the transformation should be understood as: resolve the clauses containing a and $\overline{a}$, then perform the resolution steps from the subproof rooted in ab, then perform the resolution steps from the subproof rooted in $\overline{a}b$ (the rest of the proof stays the same).



4.2.4. Example This example contains both previously discussed complications. Note again the emerging pattern: We resolve pairwise all clauses containing a with all clauses containing $\overline{a}$. Then, for each clause in the original proof, which contains $\overline{a}$, we perform the resolution steps from the subproof rooted in ab (i.e. for each of those clauses, we add a copy of the whole subproof) and finally we perform the resolution steps from the subproof rooted in $\overline{a}b$. The rest of the proof stays the same.



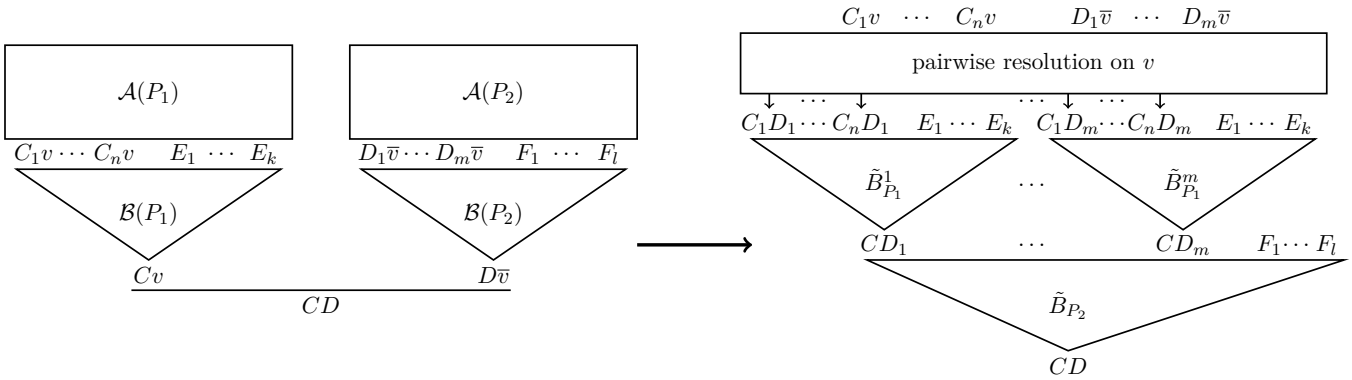
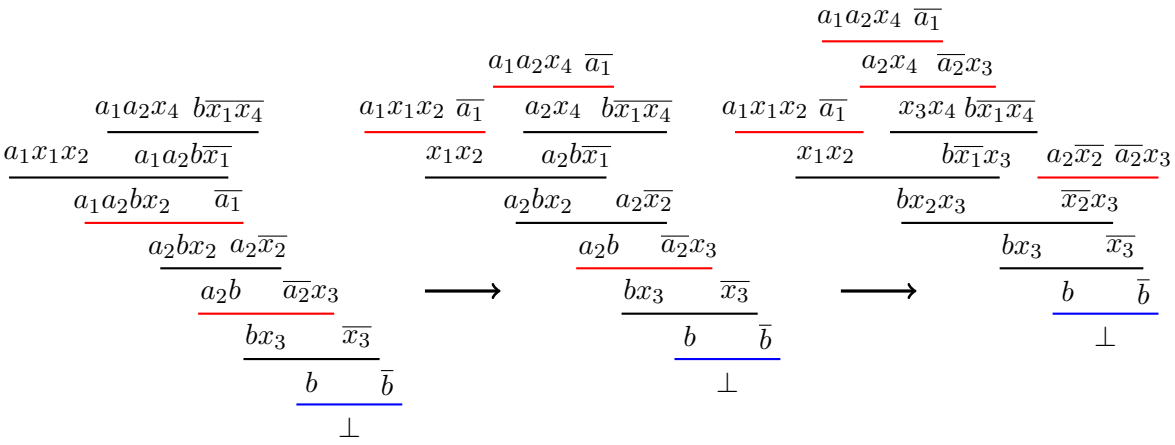


Figure 4.2: Visualization of the proof transformations of Algorithm 4 in the case when resolution on A-local variables is applied after resolution on B-local or global variables. (the A-parts of $trans(P_1)$ and $trans(P_2)$ are omitted in the transformed proof)

4.2.5. Example In this example we have to shift multiple steps. At first we have to shift a_1 to the top, afterwards we have to shift a_2 to the top.



We now formally present our construction for translating P into an ordered refutation: In a nutshell, algorithm 4 recursively goes through the refutation and whenever a resolution inference r on an A-local variable is encountered, the algorithm uses a proof transformation to shift r over all inferences above r, which resolve on non-A-local variables. Our construction therefore guarantees that, after visiting a clause C in P, we have constructed an ordered proof with root C. The proof transformation used in algorithm 4 is visualized in Figure 4.2.

Algorithm 4 Trans(P) - Transforming P into an ordered Proof

Input: Refutation P in $C_A \cup C_B$ in rCNF resolution system

Output: Ordered refutation $\tilde{P}$ of $C_A \cup C_B$ s.t. $\tilde{P} = trans(P)$

if sink of P is axiom **then**
  $trans(P) := P$
else
  sink of P is resolution inference r on v with premises Cv, $D\overline{v}$ and conclusion CD
  P_1 := largest subproof of P ending in Cv.
  P_2 := largest subproof of P ending in $D\overline{v}$.
  $\tilde{P}_1 = trans(P_1)$
  $\tilde{P}_2 = trans(P_2)$
  if v is not A-local **then**
    r_1 := root of $\tilde{P}_1$
    r_2 := root of $\tilde{P}_2$
    $\tilde{P} := (\tilde{P}_1 +^{\emptyset} \tilde{P}_2) +^{\{r_1, r_2\}} r$
    $trans(P) := \tilde{P}$
  else
    v is A-local
    $frontier(\tilde{P}_1) := \{C_1 v, \ldots, C_n v, E_1, \ldots, E_k\}$,
      where $E_1, \ldots, E_k$ do not contain v
    $frontier(\tilde{P}_2) := \{D_1 \overline{v}, \ldots, D_m \overline{v}, F_1, \ldots, F_l\}$
      where $F_1, \ldots, F_l$ do not contain $\overline{v}$
    for $i = 1$ to n **do**
      s_i := inference of $S_A(\tilde{P}_1)$ with conclusion $C_i v$
    end for
    for $j = 1$ to m **do**
      t_j := inference of $S_A(\tilde{P}_2)$ with conclusion $D_j \overline{v}$
    end for
    $R_1 := \{r \mid r \in S_A(\tilde{P}_1), r \text{ has conclusion } E_i\}$
    $R_2 := \{r \mid r \in S_A(\tilde{P}_2), r \text{ has conclusion } F_i\}$
    $\tilde{A}_{P_1} := \mathcal{A}(\tilde{P}_1)$
    $\tilde{A}_{P_2} := \mathcal{A}(\tilde{P}_2)$
    $\tilde{P} = \tilde{A}_{P_1} +^{\emptyset} \tilde{A}_{P_2}$
    for $i = 1$ to n and $j = 1$ to m **do**
      Resolution inference $r_{ij} := \frac{C_i v \qquad D_j \overline{v}}{C_i D_j}$
      $\tilde{P} = \tilde{P} +^{\{s_i, t_j\}} r_{ij}$
    end for
    for $j = 1$ to m **do**
      $\tilde{B}^j_{P_1} := \mathcal{B}(\tilde{P}_1)[v \leftarrow D_j]$
      u_j := root of $\tilde{B}^j_{P_1}$
      $\tilde{P} := \tilde{P} +^{\{r_{1j}, \ldots, r_{nj}\} \cup R_1} \tilde{B}^i_{P_1}$
    end for
    $\tilde{B}_{P_2} := \mathcal{B}(\tilde{P}_2)[\overline{v} \leftarrow C]$
    $\tilde{P} := \tilde{P} +^{\{u_1, \ldots, u_m\} \cup R_2} \tilde{B}_{P_2}$
  end if
end if

Next, we explain algorithm 4 in more detail: The algorithm takes as input an arbitrary proof P of C in $C_A \cup C_B$ in the CNF resolution inference system, and returns an ordered proof $\tilde{P}$ of C in $C_A \cup C_B$ in the rCNF resolution inference system. For doing so, the algorithm recursively iterates over the size of P. In the base case, when the sink of P is an axiom, then P is already ordered (line 1). Otherwise, the sink of P is a resolution inference r on a variable v. The subderivations of P ending in the premises of r are denoted by P_1 and P_2 and the algorithm respectively computes the ordered versions $\tilde{P}_1$ and $\tilde{P}_2$ of P_1 and P_2. If v is a B-local or global variable, then r does not violate the property of ordered proofs; in this case, Algorithm 4 respectively replaces P_1 and P_2 by their ordered versions $\tilde{P}_1$ and $\tilde{P}_2$, changing P this way into an ordered proof (lines 10-13). However, if v is an A-local variable, r contradicts the property of ordered proofs. In this case, Algorithm 4 first determines the frontier clauses of the ordered subderivations $\tilde{P}_1$ and $\tilde{P}_2$. Next, the pairwise resolution inferences r_{ij} on v of the frontier clauses $\tilde{P}_1$ and $\tilde{P}_2$ are moved above all resolutions inferences over B-local and global variables (lines 15-41), as follows: Variable v is replaced by C in the B-part of $\tilde{P}_2$, resulting in an ordered derivation $\tilde{B}_{P_2}$ with leaves $CD_1, \ldots, CD_m, F_1, \ldots, F_l$; these clauses are the clauses obtained by replacing v by C in the frontier clauses of $\tilde{P}_2$. Next, the B-part of the ordered version of P is computed by replacing the leaves CD_i of $\tilde{B}_{P_2}$ with the ordered derivations $\tilde{B}^i_{P_1}$ (line 36). The leaves of $\tilde{B}^i_{P_1}$, and hence of the B-part of $trans(P)$, are the clauses $E_1, \ldots, E_k$ from the frontier of $\tilde{P}_1$ and the conclusions of the pairwise resolution inferences r_{ij} on v. The frontier of $trans(P)$ consists of the frontier clauses of $\tilde{P}_1$ and $\tilde{P}_2$.

We now turn to proving correctness of Algorithm 4 using both the following lemma and theorem:

4.2.6. Lemma Let P be an rCNF-resolution proof whose sink is given by the resolution inference

$$\frac{Cv \qquad D\overline{v}}{CD}\, v$$

on an A-local variable v.
Then applying Algorithm 4 on P returns an ordered rCNF-resolution proof $trans(P)$ whose root is CD.

Proof We use the notations of Algorithm 4. Using Lemma 2.4.1, we conclude that the derivations $\tilde{B}^i_{P_1} := \mathcal{B}(trans(P_1))[v \leftarrow D_i]$ and $\tilde{B}_{P_2} := \mathcal{B}(trans(P_2))[\overline{v} \leftarrow C]$ are rCNF-derivations and $\tilde{B}_{P_2}$ ends in CD. It remains to show that each leaf of P occurs as a conclusion of an inference in $trans(P)$ (without introducing cycles). We do as follows:

- Each premise of the pairwise resolution inferences r_{ij} is a conclusion of an inference of either $\mathcal{A}(trans(P_1))$ or $\mathcal{A}(trans(P_2))$.
- Each subderivation $\tilde{B}^i_{P_1}$ has leaves $C_1D_i, \ldots, C_nD_i$, $E_1, \ldots, E_k$ and root CD_j, since $C_1v, \ldots, C_nv$ are by definition the only leaves of $trans(P_1)$ which contain

v. Therefore each of the leaves is also a conclusion either of one of the pairwise resolution inferences or of an inference of $\mathcal{A}(P_1)$.
- The subderivation $\tilde{B}_{P_2}$ has leaves $CD_1, \ldots, CD_m$, $F_1, \ldots, F_l$ and root CD, since $D_1\overline{v}, \ldots, D_m\overline{v}$ are by definition the only leaves of $trans(P_2)$ which contain $\overline{v}$. Therefore each leaf is also a conclusion either of an inference of one of the derivations $\tilde{B}^i_{P_1}$ or of an inference of $\mathcal{A}(trans(P_2))$.

4.2.7. Theorem Algorithm 4 is correct. That is, for an arbitrary CNF-resolution refutation P in $C_A \cup C_B$, $trans(P)$ is an ordered rCNF-resolution refutation in $C_A \cup C_B$.

Proof We apply structural induction on P and use the induction hypothesis that for any subproof P with root C, $trans(P)$ is an ordered rCNF-resolution proof with root C.
Base case: If P is a proof consisting of a single axiom r, then P contains no resolution inferences. Hence, P is ordered and so is $trans(P)$.
Inductive case: If P is a proof whose sink is a resolution inference on v with premises Cv and $D\overline{v}$, let P_1 and P_2 be the subproofs ending respectively in Cv and $D\overline{v}$. By the induction hypothesis, $trans(P_1)$ and $trans(P_2)$ are ordered and rooted in Cv respectively. $D\overline{v}$. We make a case distinction over the resolution variable v of r.
- If v is not an A-local literal, then $trans(P)$ is ordered as $trans(P_1)$ and $trans(P_2)$ are ordered and r does not violate the properties of ordered proofs.
- If v is an A-local variable, then using Lemma 4.2.6 we conclude that $trans(P)$ is also an r-CNF proof, rooted in CD. Note however that, while computing $trans(P)$, Algorithm 4 removes the resolution inference r of P from $trans(P)$ and adds to $trans(P)$ resolution inferences on A-local variables only below the lowest inferences on A-local variables of P. In other words, no newly introduced inference in $trans(P)$ has an inference above that resolves on a B-local or global literal. By the induction hypothesis, we then conclude that $trans(P)$ is ordered.

4.3 Implicit Proof Transformations and Interpolation Systems

Although Algorithm 4 always yields ordered refutations and therefore interpolants, applying it to big refutations P is too expensive: the proof transformations of Algorithm 4 cause in general an exponential blowup in the size of $trans(P)$. Note however that for computing interpolants we do not need to construct the entire proof $trans(P)$; all we need is the frontier of $trans(P)$. In our interpolation approach we are therefore *interested in computing explicitly only the frontier of* $trans(P)$. Note that while computing $trans(P)$ in Algorithm 4, we push the resolution step on an A-local variable v exactly below the A-parts of the ordered subproofs of P_1 and P_2. This way, the frontier of $trans(P)$ only depends on the frontiers of the two immediate subproofs P_1 and P_2 and the resolution variable v we push upwards. We are therefore able to compute the frontiers for each transformed proof $trans(P)$ without explicitly computing the transformed proof $trans(P)$, as presented below.

4.3.1. Theorem Let P be an rCNF-resolution proof.

1. Let P consist of a single A-axiom r with conclusion C. Then
$$frontier(trans(P)) = frontier(P) = \{C\}.$$
2. Let P consist of a single B-hypothesis r, then
$$frontier(trans(P)) = frontier(P) = \emptyset.$$
3. Let P be a proof where the sink is a resolution inference r on v with premises Cv and $D\overline{v}$ and let P_1 and P_2 be the subproofs ending in Cv resp. $D\overline{v}$. Let further P' be the proof consisting of $trans(P_1), trans(P_2)$ and r.
We use a case distinction on the properties of r:
If v is a B-local or global symbol, we have
$$frontier(trans(P)) = frontier(P') = frontier(P_1) \cup frontier(P_2).$$
Otherwise v is an A-local literal, so the transformation adds at least one inference on a below each lowest inference on A-local literals of $trans(P_1)$, which has a conclusion containing a, and each lowest inference on A-local literals of $trans(P_2)$, which has a conclusion containing $\overline{a}$, but adds no other A-inferences anywhere else. We therefore can conclude that $frontier(trans(P))$ consists of the pairwise resolvents of frontier clauses of $trans(P_1)$ containing a and frontier clauses of $trans(P_2)$ containing $\overline{a}$ and furthermore of all other clauses of $frontier(trans(P_1))$ and $frontier(trans(P_2))$.

We can now use Theorem 4.3.1 to trivially extract rules which describe how to recursively compute the frontier of the transformed proof without explicitly computing the transformation.
We formulate these rules in the style of interpolation approaches where clauses in proofs are annotated by their so-called partial interpolants, see e.g. [23, 8]. Consider an arbitrary rCNF resolution proof P. For any clause C of P, let P_C denote the largest subderivation of P rooted in C. From Theorem 4.2.7, we conclude that $trans(P_C)$ is ordered. For each clause C of P we set its partial interpolant to be $\bigwedge_{C' \in frontier(trans(P_C))} C'$. Using Theorem 4.3.1, we have the following representation of our interpolation system, where clauses are labelled by their partial interpolants using [.]:

4.3.2. Definition Let P be an rCNF refutation in $C_A \cup C_B$. We define the interpolation system Itp_{CNF} to be the interpolation system that maps vertices of P to their partial interpolants as follows:

For a hypothesis with conclusion C:

$$\frac{}{C \quad [C]} \text{ if } C \in A$$

$$\frac{}{C \quad [\top]} \text{ if } C \in B$$

For a resolution inference on a B-local or global variable v:

$$\frac{Cv \quad [I_1] \qquad D\overline{v} \quad [I_2]}{CD \quad [I_1 \wedge I_2]}$$

where $I_1 = frontier\_itp(P_{I_1})$ and $I_2 = frontier\_itp(P_{I_2})$.

For a resolution inference on an A-local variable v:

$$\frac{Cv \quad [I_1] \qquad D\overline{v} \quad [I_2]}{CD \quad [I_1' \otimes^v I_2' \wedge I_1'' \wedge I_2'']}$$

where

- $I_1 = frontier\_itp(P_{I_1})$ and $I_2 = frontier\_itp(P_{I_2})$;
- I_1' and I_2' contain respectively all clauses of I_1 and I_2 which contain v, respectively $\overline{v}$;
- I_1'' and I_2'' contain respectively all clauses of I_1 and I_2 which do not contain v, respectively $\overline{v}$.

Using Theorem 4.3.1, we conclude the following result.

4.3.3. Theorem Let P be an arbitrary refutation in $C_A \cup C_B$. Then Itp_{CNF} computes an interpolant of A and B as the partial interpolant of the root of P.

CHAPTER 5

State of the art

Proof-based interpolation algorithms use the idea to extract an interpolant from a refutation in a sound proof system. There are two approaches:

For propositional interpolation, there is the idea of interpolation systems: Such algorithms require a propositional resolution refutation as input (which can be produced by SAT-solvers which implement proof-logging, e.g. MiniSat [24]) and recursively attach to each node of that proof a formula, so that the formula, which is finally attached to the root, is an interpolant. In contrast to designing an interpolation systems without further guideline, we use the approach to first design a proof transformation, which transforms the propositional resolution refutation into a local proof and then use the transformation to extract an interpolation system from it by figuring out how the splitting formulas change during the recursive computation. We therefore achieve a design which is more constructive in comparison to coming up with an interpolation system without further guideline. Furthermore, it is much easier to get variations of interpolation systems, since one can use proof theoretic ideas to vary the proof transformation and therefore easily get different interpolation systems. We think that each interpolation system is induced by some proof transformation using the stated approach.

We now give an overview of important interpolation systems:
The first interpolation system has been independently discovered in [18, 25, 26]. Then the seminal paper [1] introduced an improved version of the algorithm, which is nowadays the most used interpolation approach. We have strong evidence that there exists a proof transformation, which takes a CNF-resolution refutation as input and produces an NNF-resolution proof, which induces McMillan's algorithm as interpolation system.
Afterwards [19] generalized [18, 25, 26] and [1] to a system of interpolation algorithms and also coined the name interpolation systems. Note that any instance of the generalized algorithm from [19] generates interpolants with the property that the alternation between

conjunctions and disjunctions is unbounded. In contrast to this, our algorithm always produces CNF-interpolants.
Then [6] showed how to apply the algorithm from [1] to compressed resolution proofs (i.e. clausal / DRUP-proofs) in order to avoid an explicit representation of resolution proofs, which can be infeasible. It is easy to see that extracting interpolants from such a compressed representation is also applicable to our interpolation system and all the interpolation systems from [19].
Finally our interpolation system is most similar to the second interpolation algorithm described in [7], but our transformation always produces interpolants in the first place, while their algorithm needs further SAT-calls in some cases.

For general first-order logic, there are interpolation algorithms based on the idea of extracting interpolants from proofs of a special form, i.e. from local proofs. Such proofs can for instance be generated by the world-leading theorem prover Vampire [27].
The idea of using local proofs for interpolation already occured implicitly in [28] and was explicitly stated in [3]. Afterwards, [15] showed that one can extract interpolants from local refutations independently from the proof system. Our algorithm is arguably more intuitive and yields simpler interpolants than [15]. Furthermore, our construction guarantees that the generated interpolants are linear in the size of the proof, whereas the interpolants from [15] are quadratic in the size of the proof in the worst-case, as pointed out by [29].

The algorithm from [15] was further optimized in [30]. Note that the algorithm stated in [15] unnecessary restricts itself to always use a splitting function, which traverses the proof bottom-up and assigns as many steps as possible to the current partition. In contrast to this, our algorithm allows to freely choose how to assign inferences, which are no axioms and don't contain any local symbols.
In order to be able to generate different splitting functions even under presence of the stated restriction, [30] introduced a proof transformation, which collapses some of the inferences in the grey area. Then they express the locality of the resulting proof as a set of propositional formulas and use an SMT solver to find a model of this formula, i.e. a specific instance of the transformation, which produces a proof, such that the extracted interpolant is optimal with regard to some quality measurement. Since our approach is less restricted, we are able to encode the optimal interpolant in a much simpler way and without an implicit proof transformation.

From a more general view our framework connects the research lines of interpolation systems and local proofs by arguing that interpolation systems can be seen as implicitly constructing (forward) local proofs.

CHAPTER 6

Conclusion

At first, we provided an intuitive introduction to craig interpolation, which motivates the central notion of local proofs. From this, we derived a new interpolation algorithm for arbitrary local proofs in first-order-logic, which advances the state of the art. Furthermore we emphasized that instead of directly generating local proofs it is sometimes better to use the strategy generate-arbitrary-proof, transform-into-local-proof, read-off-interpolant. Finally, we connected the strategy to interpolation systems.

Afterwards we used the stated strategy to develop a new interpolation algorithm for propositional logic, which generates interpolants in CNF. We emphasized the idea that instead of directly developing an interpolation system it should be easier to develop a proof transformation, which generates a (forward) local proof, and use this transformation to extract rules for an interpolation systems. We therefore reduce the amount of trial and error and get a more constructive way to develop interpolation systems.

As future work we want to work out a proof transformation, which is similar to our transformation and induces as interpolation system McMillan's algorithm. Furthermore there is strong evidence that this enables us to generalize both our interpolation system and McMillan's algorithm in order to get a wide range of interpolants differing in logical strength.
Furthermore we think that the process of developing and understanding interpolation algorithms crucially depends on understanding the proofs and induced splits. Deep inference systems and especially the functorial calculus seem to be a natural choice for a clean framework, so another direction to head in would be to state our current framework in the functorial calculus.

Bibliography

[1] K. L. McMillan. „Interpolation and SAT-Based Model Checking“. In: *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*. Ed. by W. A. H. Jr. and F. Somenzi. Vol. 2725. Lecture Notes in Computer Science. Springer, 2003, pp. 1–13.

[2] T. A. Henzinger et al. „Abstractions from proofs“. In: *Principles of Programming Languages*. ACM, 2004, pp. 232–244.

[3] R. Jhala and K. L. McMillan. „A Practical and Complete Approach to Predicate Refinement“. In: *Tools and Algorithms for the Construction and Analysis of Systems, 12th International Conference, TACAS 2006 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25 - April 2, 2006, Proceedings*. Ed. by H. Hermanns and J. Palsberg. Vol. 3920. Lecture Notes in Computer Science. Springer, 2006, pp. 459–473.

[4] R. Jhala and K. L. McMillan. „Interpolant-Based Transition Relation Approximation“. In: *Logical Methods in Computer Science* 3.4 (2007).

[5] Y. Vizel, G. Weissenbacher, and S. Malik. „Boolean Satisfiability Solvers and Their Applications in Model Checking“. In: *Proceedings of the IEEE* 103.11 (2015), pp. 2021–2035.

[6] A. Gurfinkel and Y. Vizel. „Druping for interpolants“. In: *Formal Methods in Computer-Aided Design*. FMCAD Inc. 2014, pp. 99–106.

[7] Y. Vizel, V. Ryvchin, and A. Nadel. „Efficient Generation of Small Interpolants in CNF“. In: *CAV*. Vol. 8044. LNCS. Springer, 2013, pp. 330–346.

[8] V. D'Silva et al. „Interpolant Strength“. In: *VMCAI*. Vol. 5944. LNCS. Springer, 2010, pp. 129–145.

[9] M. Schlaipfer and G. Weissenbacher. „Labelled Interpolation Systems for Hyper-Resolution, Clausal, and Local Proofs“. In: *Journal of Automated Reasoning* 57 (1 2016), pp. 3–36.

[10] S. K. Lahiri and K. K. Mehra. *Interpolant based Decision Procedure for Quantifier-Free Presburger Arithmetic*. Tech. rep. MSR-TR-2005-121. Microsoft Research, 2005.

[11] A. Cimatti, A. Griggio, and R. Sebastiani. „Efficient Interpolant Generation in Satisfiability Modulo Theories“. In: *TACAS*. Vol. 4963. LNCS. Springer, 2008, pp. 397–412.

[12] D. Kroening and G. Weissenbacher. „Lifting Propositional Interpolants to the Word-Level“. In: *Formal Methods in Computer-Aided Design*. IEEE, 2007, pp. 85–89.

[13] D. Beyer, D. Zufferey, and R. Majumdar. „CSIsat: Interpolation for LA+EUF“. In: *CAV*. Vol. 5123. LNCS. Springer, 2008, pp. 304–308.

[14] K. L. McMillan. „Quantified Invariant Generation Using an Interpolating Saturation Prover“. In: *TACAS*. Vol. 4963. LNCS. Springer, 2008, pp. 413–427.

[15] 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*. Ed. by R. A. Schmidt. Vol. 5663. Lecture Notes in Computer Science. Springer, 2009, pp. 199–213.

[16] W. Craig. „Linear Reasoning. A New Form of the Herbrand-Gentzen Theorem“. In: *J. Symb. Log.* 22.3 (1957), pp. 250–268.

[17] K. L. McMillan. „Interpolation: Proofs in the Service of Model Checking“. In: (2012).

[18] P. Pudlák. „Lower Bounds for Resolution and Cutting Plane Proofs and Monotone Computations“. In: *J. Symb. Log.* 62.3 (1997), pp. 981–998.

[19] V. D'Silva et al. „Interpolant Strength“. In: *Verification, Model Checking, and Abstract Interpretation, 11th International Conference, VMCAI 2010, Madrid, Spain, January 17-19, 2010. Proceedings*. Ed. by G. Barthe and M. V. Hermenegildo. Vol. 5944. Lecture Notes in Computer Science. Springer, 2010, pp. 129–145.

[20] S. Rollini et al. „Resolution proof transformation for compression and interpolation“. English. In: *Formal Methods in System Design* 45.1 (2014), pp. 1–41.

[21] Y. Vizel, A. Nadel, and V. Ryvchin. „Efficient generation of small interpolants in CNF“. In: *Formal Methods in System Design* 47.1 (2015), pp. 51–74.

[22] M. Davis and H. Putnam. „A Computing Procedure for Quantification Theory“. In: *J. ACM* 7.3 (1960), pp. 201–215.

[23] K. L. McMillan. „Interpolation and SAT-Based Model Checking“. In: *CAV*. Vol. 2725. LNCS. Springer, 2003, pp. 1–13.

[24] N. Eén and N. Sörensson. „An Extensible SAT-solver“. In: *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*. Ed. by E. Giunchiglia and A. Tacchella. Vol. 2919. Lecture Notes in Computer Science. Springer, 2003, pp. 502–518.

[25] J. Krajíček. „Interpolation Theorems, Lower Bounds for Proof Systems, and Independence Results for Bounded Arithmetic“. In: *J. Symbolic Logic* 62.2 (1997), pp. 457–486.

[26] G. Huang. „Constructing Craig Interpolation Formulas“. In: *Computing and Combinatorics, First Annual International Conference, COCOON '95, Xi'an, China, August 24-26, 1995, Proceedings*. Ed. by D. Du and M. Li. Vol. 959. Lecture Notes in Computer Science. Springer, 1995, pp. 181–190.

[27] A. Riazanov and A. Voronkov. „The design and implementation of VAMPIRE“. In: *AI Commun.* 15.2-3 (2002), pp. 91–110.

[28] K. L. McMillan. „An interpolating theorem prover“. In: *Theor. Comput. Sci.* 345.1 (2005), pp. 101–121.

[29] E. M. Clarke, T. A. Henzinger, and H. Veith, eds. *Handbook of Model Checking*. Springer, 2017. Chap. 12, to appear.

[30] K. Hoder, L. Kovács, and A. Voronkov. „Playing in the grey area of proofs“. In: *Principles of Programming Languages*. ACM, 2012, pp. 259–272.