Interpolation Modulo Theories

Dissertation

zur Erlangung des Doktorgrads
der technischen Fakultät
der Albert-Ludwigs-Universität Freiburg im Breisgau
vorgelegt von

Jürgen Christ

Tag der Disputation:
26.10.2015

Dekan:
Prof. Georg Lausen, *Albert-Ludwigs-Universität Freiburg*

Referenten:
1. Gutachter: Prof. Andreas Podelski, *Albert-Ludwigs-Universität Freiburg*
2. Gutachter: Prof. David Monniaux, *CNRS / VERIMAG*
Beisitz: Prof. Peter Thiemann, *Albert-Ludwigs-Universität Freiburg*
Vorsitz: Prof. Rolf Backofen, *Albert-Ludwigs-Universität Freiburg*

Abstract

We present new concepts and techniques to generate interpolants from the proofs produced by SMT solvers. The overall result is to expand the scope of interpolation to the wealth of theories that are supported by SMT solvers. SMT solvers can prove satisfiability for an ever growing number of theories (and for their combinations; theories are often combined to account for rich data as commonly used in complex systems). However, according to the present state of the art, SMT solvers can compute interpolants only for a few selected theories (and it is unknown how two existing methods can be combined to compute interpolants in the combination of the theories). The notion of *interpolant* goes back to Craig's interpolation theorem for first-order logic. An increasing range of software analysis tools use techniques based on interpolants.

We present a new algorithm to compute Craig interpolants from the proof produced by an SMT solver, for a wide range of theories and for their combination. The algorithm uses the proof produced by the SMT solver but it does not interfere with the intermediate steps of producing the proof and it does not manipulate the proof. As a consequence, the algorithm is generic in the theory and it can be put on top of an existing SMT solver without impeding its generality and without impeding its efficiency.

We also present an extension of the algorithm to tree interpolants. Tree interpolants are used whenever the behaviour of a software is not represented as a linear sequence (for example, to account for the return of a function call in the execution of a program). To our knowledge, this is the first algorithm that computes (provably correct) tree interpolants from SMT proofs. Motivated by the fact that interpolation algorithms without an existing correctness proof are notoriously wrong, we present a proof for the correctness of the tree interpolation algorithm.

We show how one can instantiate the algorithm for the quantifier-free fragment of the theory of uninterpreted functions and the theory of linear arithmetic over the integers or the reals, and we have implemented the resulting algorithm. The implementation is part of the interpolating SMT solver SMTInterpol. It is freely available under LGPL. Feedback from users is encouraging.

Zusammenfassung

Wir präsentieren neue Konzepte und Techniken, um Interpolanten aus Beweisen zu generieren, die von SMT Solvern generiert wurden. Das Resultat dieser Arbeit erweitert die Anwendbarkeit von Interpolation auf die Fülle an Theorien, die von SMT Solvern unterstützt werden. SMT Solver können Erfüllbarkeit für eine immer größer werdende Anzahl an Theorien und deren Kombinationen beweisen (Theorien werden oft kombiniert, um die reichhaltigen Datenstrukturen komplexer Programme zu unterstützen). Allerdings können SMT Solver derzeit nur für eine relativ kleine Auswahl an Theorien auch Interpolanten generieren (ohne dass bekannt wäre, wie zwei bestehende Methoden kombiniert werden können, um Interpolanten in der Kombination der Theorien zu generieren). Der Begriff *Interpolant* stammt von Craig's Interpolationstheorem für Logik der ersten Stufe. Eine wachsende Anzahl an Programmen zur Analyse von Software verwendet Techniken, die auf Interpolanten aufbauen.

Wir präsentieren einen neuen Algorithmus, um Craig Interpolanten aus von einem SMT Solver produzierten Beweisen zu berechnen. Der Algorithmus kann für eine große Klasse von Theorien und deren Kombination verwendet werden. Er verwendet die Beweise, die ein SMT Solver produziert, ohne in die Zwischenschritte der Beweisproduktion einzugreifen, oder den Beweis zu verändern. Daher ist der Algorithmus generisch in der verwendeten Theorie und kann mit einem SMT Solver kombiniert werden, ohne dessen Effizienz oder Allgemeingültigkeit zu beeinträchtigen.

Des weiteren präsentieren wir eine Erweiterung des Algorithmus, die Bauminterpolanten berechnen kann. Bauminterpolanten werden immer dann verwendet, wenn das Verhalten von Software nicht als lineare Sequenz beschrieben werden kann (z. B. um den Rücksprung eines Funktionsaufrufes zu modellieren). Unseres Wissens nach ist dies der erste Algorithmus, der (bewiesenermaßen korrekte) Bauminterpolanten aus SMT Beweisen generiert. Motiviert durch die Tatsache, dass Interpolationsalgorithmen, die nicht als korrekt bewiesen wurden, dann auch oft fehlerhafte Interpolanten produzieren, geben wir einen Korrektheitsbeweis für den Algorithmus zur Bauminterpolation.

Wir zeigen, wie der Algorithmus für das quantorenfreie Fragment der Theorie der uninterpretierten Funktionen und der Theorie der linearen Arithmetik über den ganzen und den rationalen Zahlen instantiiert werden kann. Der resultierende Algorithmus ist Teil des interpolierenden SMT Solvers SMTInterpol. Dieser ist frei verfügbar unter LGPL. Das Feedback von Benutzern ist ermutigend.

Acknowledgements

Foremost I thank Andreas Podelski for accepting me as a PhD student. He gave me the opportunity and space to develop my own research ideas. Furthermore, he supported me throughout my PhD time and arranged an internship at SRI International. Moreover I thank David Monniaux for serving as second reviewer of this thesis.

Next I want to thank Jochen Hoenicke for his invaluable contributions to my research and this thesis. His ingenuity paved the way to this thesis. I enjoyed developing the results together with him as well as many conversations over lunch.

I am grateful for the working environment provided by my colleagues. In addition to Andreas and Jochen I want to thank Rebecca Albrecht, Stephan Arlt, Jelena Barth, Sergiy Bogomolov, Berit Brauer, Daniel Dietsch, Evren Ermis, Sergio Feo-Arenis, Marius Greitschus, Matthias Heizmann, Christian Herrera, Marlis Jost, Vincent Langenfeld, Alexander Malkis, Stefan Maus, Martin Mehlmann, Corina Mitrohin, Marco Muñiz, Alexander Nutz, Amalinda Post, Martin Preen, Martin Schäf, Christian Schilling, Nassim Seghir, Martin Wehrle, Bernd Westphal, and Thomas Wies.

I thank SRI International, foremost Natarajan Shankar, Bruno Dutertre, and Sam Owre for a very pleasant internship in 2011. I enjoyed working with the people at SRI and getting to experience a different working environment.

I especially thank the German Research Foundation (DFG) for providing financial support for the research centre “Automated Verification and Analysis of Complex Systems” (AVACS).

Last but not least I want to thank my family for their continuous support. I dedicate this thesis to my mother.

Contents

Chapter 1
Introduction

Many recent advances in the field of computer science are backed by the advances in automated theorem provers like Satisfiability (SAT) solvers [8] and Satisfiability modulo Theories (SMT) solvers [5]. These tools take as input a formula in some logic and check for satisfiability of this formula with respect to this logic. The input formula represents an encoding of another problem. In model checking [69, 72, 53, 54, 36, 40], for example, a formula represents a path from an initial state to an error state in an abstraction of the model. If this formula is satisfiable, the path can be concretised. Otherwise it is only contained in the abstraction and not in the concrete system. A refinement loop is used to remove that so-called spurious path from the abstraction of the model. One way to refine a model is to use *interpolation* to compute a tighter abstraction of the model that excludes the spurious path, but might still contain other spurious paths. Similarly, state space abstraction [56, 72, 17] uses interpolation to over-approximate the set of "good states" while excluding all "bad states".

One way to automate interpolation is to use *Craig interpolants* [29]. Given an inconsistent conjunction $A \wedge B$, a Craig interpolant is a formula I that over-approximates A (i. e., $A \rightarrow I$ is valid), is inconsistent with B (i. e., $I \wedge B$ is unsatisfiable), and only uses vocabulary common to A and B (i. e., I contains only variables shared between A and B). Craig showed in his seminal work on interpolation how to construct an interpolant. His approach introduces quantifiers to eliminate all symbols that are not shared between A and B. Unfortunately, adding quantifiers to a formula makes satisfiability checking of this formula much harder or even undecidable. To alleviate this downside, we are interested in *quantifier-free* interpolants.

For some logics, we could use quantifier elimination [67]. But, in general, quantifier elimination is expensive. Methods to avoid it exist. They were pioneered for equational theories by Huang [58], for propositional logic by Pudlák [80], and for some theories in the context of Satisfiability modulo Theories (SMT) by McMillan [71]. All three approaches use proofs to derive an interpolant. Essentially, the techniques separate the proof in order to generate an interpolant. They use syntactic rules to compute the interpolant based on the inference step done in the proof. The result is a quantifier-free interpolant for quantifier-free input.

For propositional logic, a SAT solver typically produces resolution-based proofs that show the unsatisfiability of the input formula. Extracting Craig interpolants from such proofs is a well understood and easy task that can be accomplished, e. g., using the algorithms of Pudlák [80] or McMillan [71]. An essential property of the proofs generated by SAT solvers is that every proof step only involves literals that occur in the input. This property allows us to cut the proof into two parts at inference boundaries.

This property does not hold for proofs produced by SMT solvers [5] for formulae in a combination of first order theories. Such solvers produce new literals for different reasons. First, to combine two or more theory solvers, SMT solvers exchange (dis-)equalities between the symbols common to these two theories in a Nelson-Oppen-style theory combination [75]. Second, various techniques dynamically generate new literals to simplify proof generation. Third, new literals are introduced in the context of a branch-and-bound or branch-and-cut search for non-convex theories. The theory of linear integer arithmetic for example is typically solved by searching a model for the relaxation of the formula to linear rational arithmetic and then using branch-and-cut with Gomory cuts or *extended branches* [35] to remove the current non-integer solution from the solution space of the relaxation. The literals produced by either of these techniques only contain symbols that are already present in the input. However, a literal produced by one of these techniques may be *mixed*[1] in the sense that it may contain symbols occurring only in A and symbols occurring only in B. These literals pose the major difficulty when extracting interpolants from proofs produced by SMT solvers. We cannot directly apply the techniques developed in the context of SAT solvers since these techniques cannot deal with mixed literals.

Two different approaches were proposed to solve this problem. First, the solver can be restricted to generate proofs of a special shape [24]. This approach only solves the problem of mixed literals generated by theory combination. It does not provide a solution for non-convex theories like linear integer arithmetic. Second, the proof tree can be manipulated in a post-processing step to move mixed literals towards the leaves of the proof tree [15]. Once a sub-proof only uses mixed literals, a new lemma can be extracted that replaces all inferences on mixed literals. This technique requires an algorithm to compute interpolants for these new lemmas. Since the lemmas might combine multiple theories, the approach needs an algorithm to compute interpolant in the conjunctive fragment of a combination of theories. Thus, proof tree manipulation does not solve the problem. Instead, it reduces it to another problem that is almost as hard as the original problem. We start a new research direction for interpolation.

In this thesis, we present a generic algorithm for interpolation in SMT that is able to compute interpolants even if mixed literals are present in the proof. The novelty of the algorithm lies in its verbatim use of a proof generated by an SMT solver. It does not interfere with proof production and it does not manipulate the proof. The algorithm reuses existing techniques known from SAT for the propositional steps in the proof tree as long as no mixed literal is involved. We devise specialised rules for inference steps on mixed literals. We call the resulting algorithm *proof tree preserving interpolation*. Furthermore, we show how to use this algorithm to compute interpolants in the combination of the theory of uninterpreted functions with the theory of linear arithmetic over reals or integers.

A side-effect of the verbatim use of a proof is that our algorithm can be combined with different proof tree manipulation techniques [28, 44, 81, 9] to compute different interpolants. This allows us to easily vary the logical strength of the interpolants by manipulation of the proof. Correctness of the interpolants generated by our algorithm is proven independent of a specific shape of the proof. Thus, correctness of the interpolants computed from a manipulated proof is proven as long as the manipulation of the proof is correct, i. e., the proof remains a proof of unsatisfiability.

[1] Mixed literals sometimes are called *uncolourable*.

Recent advances in model checking led to the definition of more elaborate forms of interpolation. We consider the case of tree interpolants which is a generalisation of sequence interpolants [69]. Input to tree interpolation is a tree where each vertex is labelled with a formula. If the conjunction of these labels is unsatisfiable, a tree interpolant can be computed. This tree interpolant is another labelling of the tree that assigns to each vertex a formula that follows from the interpolants labelling the children and the input formula labelling the current vertex. Furthermore, a symbol condition similar to binary interpolation is used. In principle, tree interpolation can be reduced to multiple binary interpolation problem. This reduction, however, requires solving of different binary interpolation problems in order to guarantee correctness of the generated tree interpolant. To achieve better performance in general, we opt for a method to compute a tree interpolant from one proof, i. e., we show unsatisfiability of the conjunction of the labels of the input tree and use the generated proof to compute a tree interpolant.

Tree interpolants arise from model-checking recursive and concurrent programs in a natural way. An execution of the program with procedures can be represented as a nested trace, where the statement after a procedure call has two predecessors, the return statement of the called procedure and the procedure call itself. To reason about correctness in a modular way requires combining the function summary with the intermediate assertion before the procedure call. This leads naturally to a tree-like structure [54, 56]. Tree interpolants are also useful to approximate function summaries for incremental update checking [83]. Similarly, modular reasoning about concurrent programs needs interference free proofs or assume-guarantee reasoning. The proof of an intermediate assertion can depend on the previous assertion of the same thread and the guarantees provided by the other threads. Thus, an unfolding of the parallel program has again a tree-like shape. Other uses of interpolants in model-checking are data-flow graph based method [43] which compute tree interpolants for an unfolded data-flow tree.

These tools use efficient SMT solvers to reason about verification conditions and generate interpolants. SMT solvers generate interpolants from proofs. Although tree interpolants are widely used, only a few SMT solver are able to produce them without the need for repeated applications of binary interpolation to different interpolation problems. The techniques used by these tools to ensure correctness of inductive sequences and trees of interpolants are not well documented.

We extend our generic algorithm in order to extract tree interpolants from a proof produced by an SMT solver. The extension is conservative in the sense that the rules used to extract binary interpolants are preserved. We call the resulting algorithm *proof tree preserving tree interpolation*. Proving correctness of algorithms that compute tree interpolants is a tedious and error-prone task. Most existing algorithms are not proven correct. We show correctness of our algorithm and of an instantiation to the combination of the theory of uninterpreted functions with the theory of linear arithmetic over reals or integers. To our knowledge, this is the first algorithm that is proven correct for this combination of theories.

Contribution

This thesis makes the following contributions.

- We developed a generic algorithm to compute interpolants from proofs generated by SMT solvers. This is the first algorithm to compute interpolants without restricting the inference steps done by the solver or manipulating the proof tree in order to prevent mixed literals. The algorithm is parameterised by two functions. We demonstrate the algorithm for the combination of the theory of uninterpreted functions with the theory of linear arithmetic over reals or integers. We call this algorithm *proof tree preserving interpolation*.
- We extend this algorithm to *tree interpolation*. The extension allows us to extract a tree interpolant from a proof generated by a state of the art SMT solver without the need for repeated binary interpolation, proof tree manipulation, or any restrictions on the inferences done by the solver. We show how to adjust the conditions on the two functions that parameterise proof tree preserving interpolation to ensure correctness of the generated tree interpolant. We demonstrate the algorithm for the combination of the theory of uninterpreted functions with the theory of linear arithmetic over reals or integers.
- We demonstrate the usefulness of the aforementioned algorithms in the interpolating SMT solver *SMTInterpol*. The solver is freely available under LGPL and used by numerous model checkers (CPAchecker, Goanna, Ultimate Automizer, Ultimate Kojak, to name just a few) and various tools that do not require interpolants.

Proof Tree Preserving Interpolation

To tackle the problem of mixed literals we presented proof tree preserving interpolation at TACAS 2013 [22] and published an extended version as AVACS technical report [23]. The algorithm is, to our knowledge, the only complete interpolation technique based on proofs generated by SMT solvers for quantifier-free interpolation in the combination of the theories of uninterpreted functions and linear integer arithmetic, i. e., it is the only algorithm that can compute a quantifier-free interpolant for quantifier-free input if an interpolant exists. We present proof tree preserving interpolation in Chapter 3.

Proof Tree Preserving Tree Interpolation

We extended proof tree preserving interpolation to extract tree interpolants from one proof tree. We presented this extension at the SMT workshop 2013 [20]. A journal version is still under review [18]. To our knowledge, this is the first algorithm to compute tree interpolants from proofs generated by a state of the art SMT solver. We prove correctness of the algorithm, i. e., we show that the compute tree interpolant is correct for the given tree interpolation problem. Furthermore, the algorithm inherits completeness from proof tree preserving interpolation. We present proof tree preserving tree interpolation in Chapter 4.

SMTInterpol

We implemented both algorithms in our state of the art SMT solver SMTInterpol which was presented at SPIN 2012 [21] and regularly participated in the annual SMT competitions [3, 26] and evaluations [27]. During the FLOC Olympic Games 2014, it was awarded a Gödel medal for second place in the competition. Furthermore, SMTInterpol participates in the application track of the SMT competition since it started in 2012. This track [14] simulates the usage of an SMT solver as a core component inside another tool like, e. g., a model checker.

SMTInterpol is primarily designed to compute interpolants. Various tools use SMTInterpol for this purpose. Among them are CPAchecker [6], Goanna [16], Ultimate Automizer [52], and Ultimate Kojak [41, 40]. Additionally, interpolants generated by SMTInterpol are used to localise errors in programs via error invariants [42, 17]. Nevertheless, SMTInterpol is also used in various other scenarios [55, 65, 57, 87] that do not require interpolation. These scenarios also show the usefulness of SMTInterpol as an SMT solver even though it is not as highly optimised as competing SMT solvers.

We present SMTInterpol in Chapter 5. SMTInterpol can solve SMT problems in the quantifier-free fragment of the theories of uninterpreted functions and linear arithmetic over Integers and Reals and the combination thereof (in SMTLIB [4] notation: QF_UF, QF_LIA, QF_LRA, QF_UFLIA, and QF_UFLRA). SMTInterpol is able to compute interpolants in all these fragments, too.

Related Work

The work presented in this thesis combines interpolation methods for multiple theories. We split the related work according to the different parts covered by this thesis.

Interpolation

Craig [29] shows in his seminal work on interpolation that for every inconsistent pair of first order formulae an interpolant can be derived. In the proof of the corresponding theorem he shows how to construct interpolants without proofs by introducing quantifiers in the interpolant. In his first definition, he defines an interpolant as a formula I that splits the valid implication $A \rightarrow B$ such that the implications $A \rightarrow I$ and $I \rightarrow B$ are valid and I contains only the symbols common to A and B. He calls the version of binary interpolants used throughout this thesis *reverse interpolant*. A reverse interpolant splits an unsatisfiability proof of $A \wedge B$ into two parts: a valid implication $A \rightarrow I$ and an unsatisfiable conjunction $I \wedge B$. The symbol condition remains the same. Note that, if I is an interpolant for A and B, the it is a reverse interpolant for A and $\neg B$.

For Boolean circuits, Pudlák [80] shows how to construct quantifier-free interpolants from resolution proofs of unsatisfiability. We computes for every clause used in the derivation of the empty clause, i. e., the proof of unsatisfiability, a *partial interpolant*. The partial interpolants of the leaves depend on the origin of the clause, i. e., if the clause is part of A, part of B, or of both. Similarly, for resolution steps, he gives rules based on the origin of the pivot literal. In this thesis, we build up on this work and extend it to the SMT case where input clauses might stem from theories and newly created literals might act as pivots of resolution steps.

Almost missed in the literature on interpolation[2] is the work by Huang [58]. He presented an algorithm to construct interpolants in equational theories. When restricted to Boolean circuits, the algorithm resembles the algorithm presented by Pudlák.

Interpolation in Satisfiability Modulo a Combination of Theories

A different proof-based interpolation system is given by McMillan [70] in his seminal paper on interpolation for SMT. The presented method combines the theory of equality and uninterpreted functions with the theory of linear rational arithmetic. As in the algorithm developed by Pudlák, interpolants are computed from partial interpolants by annotating every proof step. McMillan gives different syntactic rules for the input clauses. In this thesis, we will combine the systems from Pudlák and McMillan in a way similar to labelled interpolation systems [37]. In our combined system, the systems from Pudlák and McMillan only differ in the instantiation of a projection function used to describe the origin of a literal.

The partial interpolants in McMillan's algorithm have a specific form that carries information needed to combine the theories. The proof system is incomplete for linear integer arithmetic as it cannot deal with arbitrary cuts and mixed literals introduced by these cuts. In this

[2] I thank Armin Biere for pointing me to this paper.

thesis, we resolve this deficiency by extending his general method to arbitrary mixed literals including those generated by cuts [35].

Yorsh and Musuvathi [88] show how to combine interpolants generated by an SMT solver based on Nelson-Oppen combination [75]. They define the concept of *equality-interpolating theories*. Such theories provide a shared term t for a mixed literal $a = b$ that is derivable from an interpolation problem. A troublesome mixed interface equality $a = b$ is rewritten into the conjunction $a = t \wedge t = b$. They show that both, the theory of uninterpreted functions and the theory of linear rational arithmetic are equality-interpolating. The method presented in this thesis does not explicitly split the proof. Additionally, our method can handle the theory of linear integer arithmetic without any restriction on the solver. The method of Yorsh and Musuvathi, however, cannot deal with cuts used by most modern SMT solvers to decide linear integer arithmetic since they introduce mixed literals that cannot be split into one binary conjunction.

Goel et al. [47] present a generalisation of equality-interpolating theories. They define the class of *almost-colourable proofs* and an algorithm to generate interpolants from such proofs. A proof is almost-colourable if it can be transformed into a colourable proof where every literal that occurs as pivot of a resolution step stems from A or B or both. They describe a restricted DPLL system to generate almost-colourable proofs. This system does not restrict the search if convex theories are used. Their procedure is incomplete for non-convex theories like linear arithmetic over integers since it prohibits the generation of mixed branches and cuts. The interpolation system described in this thesis does not restrict the DPLL solver or the theory solvers. Especially, we allow mixed branches and cuts in the proof tree.

Bruttomesso et al. [12] extend the notion of equality-interpolating theories to non-convex theories like the theory of linear integer arithmetic. They split an equality into a disjunction of equalities. The method presented in this thesis can be seen as an efficient implementation of equality-interpolating theories for linear integer arithmetic without actually manipulating the proof tree. Instead, we use new rules to compute partial interpolants if the pivot literal is mixed. These rules simulate splitting the proof tree using the results from Bruttomesso et al.

Cimatti et al. [24] present a method to compute interpolants for linear rational arithmetic and difference logic. The method presented in this thesis builds upon their interpolation technique for linear rational arithmetic. For theories combined via delayed theory combination, they show how to compute interpolants by transforming a proof into a so-called *ie-local* proof. In these proofs, mixed equalities are close to the leaves of the proof tree and splitting them is cheap since the proof trees that have to be duplicated are small. A variant of this restricted search strategy is used by MathSAT [49] and CSIsat [7]. The method presented in this thesis does not split and duplicate parts of the proof tree. Furthermore, the techniques presented in this thesis do not require of ie-local proofs.

Recently, techniques to transform proofs gained a lot of attention. Bruttomesso et al. [15] present a framework to lift resolution steps on mixed literals into the leaves of the resolution tree. Once a subproof only resolves on mixed literals, they replace this subproof with the conclusion removing the mixed inferences. The newly generated lemmas, however, are mixed between different theories and require special interpolation procedures. Even though these procedures only have to deal with conjunctions of literals in the combined theories it is not obvious how to compute interpolants in this setting. Similar to our algorithm, they do not restrict or interact with the SMT solver but take the proof as produced by the solver. In contrast to our approach, they manipulate the proof in a way that is worst-case exponential and rely on an interpolant generator for the conjunctive fragment of the combined theories. The interpolation system proposed in this thesis could be used as black-box by their system to compute partial interpolants for leaves containing multiple theories.

McMillan [74] presents a technique to compute interpolants from Z3 proofs. He exploits the proof tree manipulation technique from Bruttomesso et al. [15]. Whenever a sub-proof contains mixed literals, he extracts lemmas from the proof tree and delegates them to a second (possibly slower) interpolating solver. Again, the techniques presented in this thesis could be used as black-box to compute interpolants for these lemmas.

Brillout et al. [10] present an interpolating sequent calculus that can compute interpolants for the combination of uninterpreted functions and linear integer arithmetic. The interpolants computed using their method might contain quantifiers that can be reformulated to integer divisibility. They restrict the derivations allowed in the sequent calculus to ensure that the introduced quantifiers represent integer divisibility.

Rybalchenko and Sofronie–Stokkermans [82] present a method to compute interpolants in the combination of the theory of uninterpreted functions and the theory of linear rational arithmetic. The peculiarity of their approach is that they do not produce interpolants from proofs, but use off-the-shelf linear programming solvers to compute interpolants. They use a hierarchical approach [84] to deal with uninterpreted functions. In this approach, the problem of computing an interpolant in the combination of the theories is reduced to the problem of computing an interpolant in the theory of linear rational arithmetic. To prevent mixed literals in the computation of an interpolant, they compute a set of separating terms up-front. The methods presented in this thesis compute the separation on the fly. They use constraint solving techniques to solve an axiomatisation of the existence of an interpolant according to Motzkin's transposition theorem. In essence, their method is similar to the interpolation method used in this thesis. They compute coefficients to combine linear inequalities with a constraint solver after transforming the input problem.

Interpolation for Uninterpreted Functions

In his seminal work in interpolation in SMT [70], McMillan shows how to compute interpolants in the theory of uninterpreted functions. He presents a set of rules similar to those presented in Section 4.2.1 of this thesis. We extend the rules to tree interpolation and to mixed literals in the input in order to compute interpolants in a combination of theories.

Fuchs et al. [45] give an interpolation method for the theory of uninterpreted functions. Their method is based on the congruence closure algorithm [76, 77] to represent a set of equalities. Given a disequality $a \neq b$ that is in conflict with the equalities represented by the current congruence closure graph, they first colour the edges of this graph according to the origin of the equality. Then, they summarise paths to compute an interpolant. Compared to our method, Fuchs et al. do not support mixed equalities. Thus, their method is not applicable to the setting of theory combination without additional methods to remove mixed literals.

Interpolation for Linear Arithmetic over Reals

In his seminal work on interpolation in SMT [70], McMillan shows how to compute interpolants for the theory of linear arithmetic over the reals. The interpolation method sums up the contribution of the A part of a given interpolation problem to the proof of unsatisfiability. In this thesis, we extend the method pioneered by McMillan to mixed literals.

Cimatti et al. [24] show how to compute interpolants for the theory of linear arithmetic over the reals from unsatisfiability proofs generated by a state of the art SMT solver for $LA(\mathbb{Q})$. They project the literals contributing to the proof of unsatisfiability onto the A part of the given interpolation problem. In this thesis, we extend their method to include mixed literals in the proof of unsatisfiability.

Albarghouthi and McMillan present a method to derive "beautiful interpolants" [2]. They gradually refine a candidate interpolant I by excluding valuations that violate $A \models I$ or $B \wedge I \models \bot$. Such a valuation is then added to a set of violations for the corresponding formula. Then, Farkas' lemma is used to compute one half-space that separates the two sets of valuations. If no separating half-space can be found, their method tries to find multiple half-spaces to separate the two sets. The method produces interpolants in disjunctive normal form (DNF) (or, alternatively, in conjunctive normal form (CNF)). The interpolants computed by the techniques presented in this thesis produce interpolants in negation normal form (NNF) which can be seen as a preliminary step in the conversion of a formula into CNF or DNF.

Interpolation for Linear Arithmetic over Integers

For the theory of linear integer arithmetic $LA(\mathbb{Z})$ a lot of different techniques were proposed. Lynch et al. [68] present a method that produces interpolants as long as no mixed Gomory cuts are introduced. In the presence of such cuts, their interpolants might contain symbols that violate the symbol condition of Craig interpolants.

For linear Diophantine equations and linear modular equations, Jain et al. [59] present a method to compute linear modular equations as interpolants. Given an interpolation problem consisting only of equalities $\sum c_{ij}x_j = c_i$ or a set of modulo equalities $(\sum c_{ij}x_j) \equiv c_i \mod m$, they compute an interpolant as a modulo equality. Their method however is limited to equations and, thus, not suitable for the whole theory $LA(\mathbb{Z})$.

Kroening et al. [64] describe a method to compute interpolants in $LA(\mathbb{Z})$. They consider four cases to compute an interpolant. First, if the formula is unsatisfiable due to equalities over integers (i. e., linear diophantine equations), they use a Smith Normal Form decomposition to compute a proof and project the proof onto the contributions from A. The resulting interpolant is a divisibility predicate. Second, if the inequalities in the interpolation problem are unsatisfiable over the rationals, they generate an interpolant like in $LA(\mathbb{Q})$ and lift it to integer coefficients by multiplying the resulting inequality with the least common multiple of all denominators. Third, they check for conflicting predicates. If conflicting predicates exist, either $\top$ or $\bot$ is a valid interpolant depend on the origin of the unsatisfiable predicate. Last, if neither of the first three methods proved unsatisfiability and generated an interpolant (and no satisfying assignment to the original problem has been found), they strengthen the problem by splitting an inequality into three parts. If all three parts are unsatisfiable, they create an interpolant as a combination of the results of recursive calls to the interpolation procedure.

Compared to the method presented in this thesis, we do not use a dedicated module to solve linear diophantine equations but use the cuts form proofs algorithm. This algorithm uses a Hermite Normal Form to solve a system of linear diophantine equations which is similar to the Smith Normal Form decomposition to solve the system. The techniques described in this thesis are based on cuts and branches generated by the cuts from proofs algorithm. The branches introduced by this algorithm only introduce two disjuncts instead of three as done by Kroening et al.

Griggio [50] shows how to compute interpolants for $LA(\mathbb{Z})$ based on the $LA(\mathbb{Z})$-solver from MathSAT [49]. This solver uses branch-and-bound and the cuts from proofs [35] technique. Similar to the technique presented by Kroening et al. [64] the algorithm prevents generating mixed cuts and, hence, restricts the inferences done by the solver.

Tree Interpolation

Only a few publications describe how to compute tree interpolants. Gupta et al. [51] describe how to solve a set of recursion-free Horn clauses over the theories of uninterpreted functions and linear real arithmetic. This corresponds directly to the tree interpolation problem for a conjunctive formula that does not contain negated equalities. They have stricter syntactic restrictions for the partial solutions and a rule for combining partial interpolants , which is similar to our combination rule for partial interpolants. Our algorithm computes the same solutions when working on this fragment. However, we allow more input problems and our method is complete even for linear integer arithmetic.

The interpolating version of Z3 (iZ3) [1] can extract tree interpolants although there is no publication describing how it computes tree interpolants. iZ3 poses additional restrictions on the occurrence of symbols in the input and treats every non-constant function symbol as global symbol. In contrast to the method presented in this thesis, iZ3 manipulates the proof trees generated by Z3 [33] based on the technique of Bruttomesso et al. [15]. We are not aware of a publication showing correctness of the generated tree interpolants.

Chapter 2
Preliminaries

Logic, Theories, and SMT.

We assume standard first-order logic. We operate within the quantifier-free fragments of the theory of equality with uninterpreted functions *EUF* and the theories of linear arithmetic over rational numbers $LA(\mathbb{Q})$ and integers $LA(\mathbb{Z})$. The quantifier-free fragment of $LA(\mathbb{Z})$ is not closed under interpolation. Therefore, we augment the signature with division by constant functions $\lfloor \frac{\cdot}{k} \rfloor$ (rounding to $-\infty$) for all integers $k \geq 1$.

We use the standard notations $\models_T, \bot, \top$ to denote entailment in the theory T, contradiction, and tautology. In the following, we drop the subscript T as it always corresponds to the combined theory of *EUF*, $LA(\mathbb{Q})$, and $LA(\mathbb{Z})$.

The literals in $LA(\mathbb{Z})$ are of the form $s \leq c$, where c is an integer constant and s a linear combination of variables. For $LA(\mathbb{Q})$ we use constants $c \in \mathbb{Q}_\varepsilon$, $\mathbb{Q}_\varepsilon := \mathbb{Q} \cup \{q - \varepsilon | q \in \mathbb{Q}\}$ where the meaning of $s \leq q - \varepsilon$ is $s < q$. For better readability we use, e. g., $s \leq t$ resp. $s > t$ to denote $s - t \leq 0$ resp. $t - s \leq -\varepsilon$. In the integer case we use $s > t$ to denote $t - s \leq -1$. In general, we denote constant symbols by a, b, terms by s, t, numerical constants by c, function symbols by f, g, Boolean variables by p, set-valued variables by X, and other variables by x. We abbreviate a set of variables by $\mathbf{x}$.

A literal is an atom or its negation. A formula is an arbitrary Boolean combination of literals. We denote formulas by A, B, F, G, I. A clause is a disjunction of literals. A formula is in conjunctive normal form (CNF) if it is a conjunction of clauses. We assume in the remainder of this thesis that every input formula is in CNF. A formula is in negation normal form (NNF) if all negation symbols appear in front of atoms. The *propositional skeleton* of a formula is obtained by replacing theory literals by (fresh) propositional literals according to a bijective mapping between theory literals and propositional literals.

Symbol Sets.

For a formula F, we use $symb(F)$ to denote the set of non-theory symbols occurring in F. Formally the set $symb(F)$ is defined by structural induction over F:

$$symb(a) = \{a\} \qquad symb(c) = \emptyset \qquad symb(x) = \{x\} \qquad symb(X) = \{X\}$$

$$symb(\sum_{i=1}^{n} c_i t_i) = \bigcup_{i=1}^{n} symb(t_i) \qquad symb(f(t_1,\ldots,t_n)) = \{f\} \cup \bigcup_{i=1}^{n} symb(t_i)$$

$$symb(s \bowtie t) = symb(s) \cup symb(t) \text{ where } \bowtie\ \in \{<,\leq,=,\geq,>\} \qquad symb\left(\left\lfloor \frac{t}{c} \right\rfloor\right) = symb(t)$$

$$symb(\neg\ell) = symb(\ell) \qquad symb(\ell_1 \bowtie \ell_2) = symb(\ell_1) \cup symb(\ell_2) \text{ where } \bowtie\ \in \{\wedge,\vee,\rightarrow,\oplus\}$$

Similarly, we define the set $symb(C) := \bigcup_{\ell \in C} symb(\ell)$ of all symbols of a clause C as the set of the symbols occurring in its literals.

Substitution in Formulae and Monotonicity.

By $F[G_1]\ldots[G_n]$ we denote a formula in negation normal form with sub-formulae $G_1,\ldots,G_n$ that occur positively in the formula. Substituting these sub-formulae by formula $G'_1,\ldots,G'_n$ is denoted by $F[G'_1]\ldots[G'_n]$. By $F(t)$ we denote a formula with a sub-term t that can appear anywhere in F. The substitution of t with a term t' is denoted by $F(t')$.

The following lemma is important for the correctness proofs in the remainder of this thesis. It also represents a concept that is important for the understanding of the proposed procedure.

Lemma 2.1 (Monotonicity). *Given a formula $F[G_1]\ldots[G_n]$ in negation normal form with sub-formulae $G_1,\ldots,G_n$ occurring only positively in the formula and formulae $G'_1,\ldots,G'_n$, it holds that*

$$\left(\bigwedge_{i\in\{1,\ldots,n\}} (G_i \rightarrow G'_i) \right) \rightarrow (F[G_1]\ldots[G_n] \rightarrow F[G'_1]\ldots[G'_n])$$

Proof. We prove the claim by induction over the number of $\wedge$ and $\vee$ connectives in $F[\cdot]\ldots[\cdot]$. If $F[G_1]\ldots[G_n]$ is a literal different from $G_1,\ldots,G_n$ the implication holds trivially. Also for the other base case $F[G_1]\ldots[G_n] \equiv G_i$ for some $i \in \{1,\ldots,n\}$ the property holds. For the induction step observe that if $F_1[G_1]\ldots[G_n] \rightarrow F_1[G'_1]\ldots[G'_n]$ and $F_2[G_1]\ldots[G_n] \rightarrow F_2[G'_1]\ldots[G'_n]$, then

$$F_1[G_1]\ldots[G_n] \wedge F_2[G_1]\ldots[G_n] \rightarrow F_1[G'_1]\ldots[G'_n] \wedge F_2[G'_1]\ldots[G'_n] \text{ and}$$
$$F_1[G_1]\ldots[G_n] \vee F_2[G_1]\ldots[G_n] \rightarrow F_1[G'_1]\ldots[G'_n] \vee F_2[G'_1]\ldots[G'_n].$$

□

This lemma can be generalised to combine multiple formulas. We use this combination technique later to prove correctness of our approach.

Lemma 2.2 (Deep Substitution). *Given formulae $F_1[G_{11}]\ldots[G_{1n}]$ and $F_2[G_{21}]\ldots[G_{2m}]$ with sub-formulae G_{1i} for $1 \leq i \leq n$ and G_{2j} for $1 \leq j \leq m$ occurring positively in F_1 and F_2.*

If $\bigwedge_{i\in\{1,\ldots,n\}} \bigwedge_{j\in\{1,\ldots,m\}} G_{1i} \wedge G_{2j} \rightarrow G_{3ij}$ holds, then

$$F_1[G_{11}]\ldots[G_{1n}] \wedge F_2[G_{21}]\ldots[G_{2m}] \rightarrow$$
$$F_1[F_2[G_{311}]\ldots[G_{31m}]]\ldots[F_2[G_{3n1}]\ldots[G_{3nm}]].$$

Proof.

$$
\begin{aligned}
& \bigwedge_{i\in\{1,\dots,n\}} \bigwedge_{j\in\{1,\dots,m\}} ((G_{1i} \wedge G_{2j}) \rightarrow G_{3ij}) \\
\Leftrightarrow\ & \bigwedge_{i\in\{1,\dots,n\}} \bigwedge_{j\in\{1,\dots,m\}} (G_{1i} \rightarrow (G_{2j} \rightarrow G_{3ij})) \\
\Leftrightarrow\ & \bigwedge_{i\in\{1,\dots,n\}} (G_{1i} \rightarrow \bigwedge_{j\in\{1,\dots,m\}} (G_{2j} \rightarrow G_{3ij})) \\
\{\text{monotonicity}\} \Rightarrow\ & \bigwedge_{i\in\{1,\dots,n\}} (G_{1i} \rightarrow (F_2[G_{21}]\dots[G_{2m}] \rightarrow F_2[G_{3i1}]\dots[G_{3im}])) \\
\Leftrightarrow\ & \bigwedge_{i\in\{1,\dots,n\}} (F_2[G_{21}]\dots[G_{2m}] \rightarrow (G_{1i} \rightarrow F_2[G_{3i1}]\dots[G_{3im}])) \\
\Leftrightarrow\ & (F_2[G_{21}]\dots[G_{2m}] \rightarrow \bigwedge_{i\in\{1,\dots,n\}} (G_{1i} \rightarrow F_2[G_{3i1}]\dots[G_{3im}])) \\
\{\text{monotonicity}\} \Rightarrow\ & (F_2[G_{21}]\dots[G_{2m}] \rightarrow (F_1[G_{11}]\dots[G_{1n}] \rightarrow \\
& \quad F_1[F_2[G_{311}]\dots[G_{31m}]]\dots[F_2[G_{3n1}]\dots[G_{3nm}]])) \\
\Leftrightarrow\ & (F_1[G_{11}]\dots[G_{1n}] \wedge F_2[G_{21}]\dots[G_{2m}]) \rightarrow \\
& \quad F_1[F_2[G_{311}]\dots[G_{31m}]]\dots[F_2[G_{3n1}]\dots[G_{3nm}]]))
\end{aligned}
$$

□

Proofs.

We assume a DPLL($\mathscr{T}$)-based approach [46] to SMT where a SAT solver enumerates truth assignments to the propositional skeleton of the input formula. Theory specific solvers (also called $\mathscr{T}$-solvers) check consistency of a conjunction of $\mathscr{T}$-literals. If an inconsistency is detected, a $\mathscr{T}$-lemma is produced. A $\mathscr{T}$-lemma is a clause that is not satisfied by the current model. It is used to block the current assignment to the propositional skeleton similarly to propositional conflicts in the context of DPLL. If for a model of the skeleton no theory detects a conflict, the formula is satisfiable. If all models of the skeleton are in conflict with at least one theory, the formula is unsatisfiable.

When multiple theories are combined using a variant of the Nelson–Oppen combination procedure [75], additional clauses are generated to ensure consistent interpretation of the equality symbol shared between all theories. Then combining uninterpreted functions and linear arithmetic, for example, clauses are added to translate inequalities into equalities. These clauses include the trichotomy clause $t_1 = t_2 \vee t_1 < t_2 \vee t_2 < t_1$. We call such clauses *theory combination clauses*.

Additionally, theories can delay case splits to the SAT solver by adding a *case split lemma*. The trichotomy lemma stated before can be seen as a case split lemma introduced by the linear arithmetic solver to split a disequality in one of the possible inequalities that satisfy the disequality. Another source for case split lemmas are lemmas from non-convex theories like the theory of linear integer arithmetic. Solvers for linear integer arithmetic typically solve the input problem by first solving a relaxation. In this relaxation, all variables are interpreted over the Reals. If no solution to the relaxation is found, the original problem is unsatisfiable.

Otherwise, if the solution found for the constraints happens to be integral, the solver found a solution to the input problem. If the solution is not integral, a branch or a cut is added to the system to remove the spurious solution from the relaxation. Assume x is the component of the solution that is assigned the non-integral value v. Then, a branch of the form $x \leq \lfloor v \rfloor \vee \lceil v \rceil \leq x$ can be added to remove the current non-integral assignment.

Given two clauses $C_1 \vee \ell$ and $C_2 \vee \neg\ell$, the resolution rule concludes the clause $C_1 \vee C_2$. If C_1 and C_2 share literals, the resulting clause might be simplified by *merging* identical literals. Thus, in general, the resolution rule can be written as

$$\textsc{Res}\ \frac{C_1 \vee D \vee \ell \qquad C_2 \vee D \vee \neg\ell}{C_1 \vee C_2 \vee D}$$

where C_1 and C_2 are disjoint. We call the literals in D *merge literals* and say the literals in D are *merged* in this resolution step. Note that merges on literals can be prevented by duplicating parts of the proof tree. In the remainder of this thesis, we assume no such manipulation of a proof tree is performed and deal with merge literals directly in our interpolation procedure. To simplify presentation, however, we omit D from the rule since mostly it is not needed.

A proof is a derivation of the empty clause from input clauses, theory lemmas, theory combination clauses, and case split clauses using only the resolution rule. Each theory lemma contains a sub-proof of its own. This proof is specific to the theory that detected the corresponding theory conflict. We present proof systems for *EUF* and linear arithmetic later.

Chapter 3
Binary Interpolants from Proofs

This chapter presents proof tree preserving interpolation. This technique takes as input a proof tree generated by a state of the art SMT solver and a partitioning A and B of the input formula. It then generates a Craig interpolant without manipulating the proof. We first review existing algorithms for proof tree based interpolation known from propositional logic. Then, we extend these techniques to SMT. This extension adds interpolation rules for mixed literals that are common in proofs generated by SMT solvers. We instantiate proof tree preserving interpolation with the theory of uninterpreted functions, and the theory of linear arithmetic both of reals and integers. We conclude this chapter with the combination of the aforementioned theories.

3.1 Binary Interpolation

A binary interpolation problem is given by two formulae A and B. If $A \wedge B \models \bot$, a binary interpolant exists. It is a formula I such that

- $A \models I$,
- $B \wedge I \models \bot$, and
- $symb(I) \subseteq symb(A) \cap symb(B)$.

This definition is typically used in the context of SMT where interpolants are extracted from proofs of unsatisfiability. Craig [29] refers to interpolants satisfying this definition as *reverse interpolants*. In the original definition by Craig, an interpolant for $A \rightarrow B$ is a formula I such that $A \rightarrow I \rightarrow B$ and I satisfies the symbol condition mentioned above (i. e., the last condition of the list). For the remainder of this thesis, we consider binary interpolant as synonym for reverse interpolant. Furthermore, by binary interpolation we refer to the extraction of a reverse interpolant from a proof of unsatisfiability.

We call a symbol $s \in symb(A) \cup symb(B)$ *shared* if $s \in symb(A) \cap symb(B)$, *A-local* if $s \in symb(A) \setminus symb(B)$, and *B-local* if $s \in symb(B) \setminus symb(A)$. Similarly, we call a term *A-local* (*B-local*) if it contains at least one A-local (B-local) and no B-local (A-local) symbols. We call a term *(AB-)shared* if it contains only shared symbols and *(AB-)mixed* if it contains A-local as well as B-local symbols. The same terminology applies to formulae.

3.2 Proof Tree Based Interpolation

Binary interpolants can be computed from proofs of unsatisfiability as Pudlák [80] and McMillan [71] have already shown. In this section we will introduce their algorithms. Then, we will discuss the changes necessary to handle mixed literals introduced, e. g., by theory combination.

3.2.1 A Generic Interpolation Algorithm

Pudlák's and McMillan's algorithms assume that the pivot literals are not mixed. We will remove this restriction later. We define a common framework that is more general and can be instantiated to obtain Pudlák's or McMillan's algorithm to compute interpolants. For this, we use two projection functions on literals $\cdot \downharpoonright A$ and $\cdot \downharpoonright B$ as defined below. They have the properties (i) $symb(\ell \downharpoonright A) \subseteq symb(A)$, (ii) $symb(\ell \downharpoonright B) \subseteq symb(B)$, and (iii) $\ell \iff (\ell \downharpoonright A \wedge \ell \downharpoonright B)$. Other projection functions are possible and this allows for varying the strength of the resulting interpolant as shown in [37]. We extend the projection function to conjunctions of literals component-wise.

	Pudlák		McMillan	
	$\ell \downharpoonright A$	$\ell \downharpoonright B$	$\ell \downharpoonright A$	$\ell \downharpoonright B$
ℓ is A-local	ℓ	$\top$	ℓ	$\top$
ℓ is B-local	$\top$	ℓ	$\top$	ℓ
ℓ is shared	ℓ	ℓ	$\top$	ℓ

Given an interpolation problem A and B, a *partial interpolant* of a clause C is an interpolant of the formulae $A \wedge (\neg C \downharpoonright A)$ and $B \wedge (\neg C \downharpoonright B)$[1]. Partial interpolants can be computed inductively over the structure of the proof tree. A partial interpolant of a theory lemma C can be computed by a theory-specific interpolation routine as an interpolant of $\neg C \downharpoonright A$ and $\neg C \downharpoonright B$. Note that the conjunction is equivalent to $\neg C$ and therefore unsatisfiable. For an input clause C from the formula A (resp. B), a partial interpolant is $\neg(\neg C \setminus A)$ (resp. $\neg C \setminus B$) where $\neg C \setminus A$ is the conjunction of all literals of $\neg C$ that are not in $\neg C \downharpoonright A$ and analogously for $\neg C \setminus B$. For a resolution step, a partial interpolant can be computed using (rule-res), which is given below. For this rule, it is easy to show that I_3 is a partial interpolant of $C_1 \vee C_2$ given that I_1 and I_2 are partial interpolants of $C_1 \vee \ell$ and $C_2 \vee \neg\ell$, respectively. Note that the "otherwise" case never triggers in McMillan's algorithm.

$$\frac{C_1 \vee \ell : I_1 \quad C_2 \vee \neg\ell : I_2}{C_1 \vee C_2 : I_3} \quad \text{where } I_3 = \begin{cases} I_1 \vee I_2 & \text{if } \ell \downharpoonright B = \top \\ I_1 \wedge I_2 & \text{if } \ell \downharpoonright A = \top \\ (I_1 \vee \ell) \wedge \\ (I_2 \vee \neg\ell) & \text{otherwise} \end{cases} \qquad \text{(rule-res)}$$

[1] Note that $\neg C$ is a conjunction of literals. Thus, $\neg C \downharpoonright A$ is well defined.

As the partial interpolant of the root of the proof tree (which is labelled with the clause $\bot$) is an interpolant of the input formulae A and B, this algorithm can be used to compute interpolants.

Theorem 3.1. *The above-given partial interpolants are correct, i. e., if I_1 is a partial interpolant of $C_1 \vee \ell$ and I_2 is a partial interpolant of $C_2 \vee \neg\ell$ then I_3 is a partial interpolant of the clause $C_1 \vee C_2$.*

Proof. The third property, i. e., $symb(I_3) \subseteq symb(A) \cap symb(B)$, clearly holds if we assume it holds for I_1 and I_2. Note that in the "otherwise" case, ℓ is shared. We prove the other two partial interpolant properties separately.

Inductivity.

We have to show
$$A \wedge \neg C_1 \downharpoonleft A \wedge \neg C_2 \downharpoonleft A \models I_3.$$
For this we use the inductivity of I_1 and I_2:
$$A \wedge \neg C_1 \downharpoonleft A \wedge \neg\ell \downharpoonleft A \models I_1 \qquad \text{(ind1)}$$
$$A \wedge \neg C_2 \downharpoonleft A \wedge \ell \downharpoonleft A \models I_2 \qquad \text{(ind2)}$$

Assume A, $\neg C_1 \downharpoonleft A$, and $\neg C_2 \downharpoonleft A$. Then, (ind1) simplifies to $\neg\ell \downharpoonleft A \rightarrow I_1$ and (ind2) simplifies to $\ell \downharpoonleft A \rightarrow I_2$. We show that I_3 holds under these assumptions.

Case $\ell \downharpoonleft B = \top$.

Then by the definition of the projection function, $\ell \downharpoonleft A = \ell$ and $\neg\ell \downharpoonleft A = \neg\ell$ hold. If ℓ holds, (ind2) gives us I_2, otherwise (ind1) gives us I_1, thus $I_3 = I_1 \vee I_2$ holds in both cases.

Case $\ell \downharpoonleft A = \top$.

Then (ind1) gives us I_1 because $\neg\ell \downharpoonleft A = \top$ (the negation of ℓ is still not in A), and (ind2) gives us I_2. So $I_3 = I_1 \wedge I_2$ holds.

Case "otherwise".

By the definition of the projection function $\ell \downharpoonleft A = \ell \downharpoonleft B = \ell$ and $\neg\ell \downharpoonleft A = \neg\ell \downharpoonleft B = \neg\ell$. If ℓ holds, the left conjunct $(I_1 \vee \ell)$ of I_3 holds and the right conjunct $(I_2 \vee \neg\ell)$ of I_3 is fulfilled because (ind2) gives us I_2. If $\neg\ell$ holds, (ind1) gives us I_1 and both conjuncts of I_3 hold.

Contradiction.

We have to show:
$$B \wedge \neg C_1 \downharpoonleft B \wedge \neg C_2 \downharpoonleft B \wedge I_3 \models \bot$$

We use the contradiction properties of I_1 and I_2:

$$B \land \neg C_1 \downharpoonright B \land \neg\ell \downharpoonright B \land I_1 \models \bot \qquad \text{(cont1)}$$
$$B \land \neg C_2 \downharpoonright B \land \ell \downharpoonright B \land I_2 \models \bot \qquad \text{(cont2)}$$

If we assume B, $\neg C_1 \downharpoonright B$, and $\neg C_2 \downharpoonright B$, (cont1) simplifies to $\neg\ell \downharpoonright B \land I_1 \rightarrow \bot$ and (cont2) simplifies to $\ell \downharpoonright B \land I_2 \rightarrow \bot$. We show $I_3 \rightarrow \bot$.

Case $\ell \downharpoonright B = \top$.

Then (cont1) and $\neg\ell \downharpoonright B = \top$ give us $I_1 \rightarrow \bot$, and (cont2) and $\ell \downharpoonright B = \top$ give us $I_2 \rightarrow \bot$. Thus $I_3 \equiv I_1 \lor I_2$ is contradictory.

Case $\ell \downharpoonright A = \top$.

Then $\ell \downharpoonright B = \ell$ and $\neg\ell \downharpoonright B = \neg\ell$. Then, if ℓ holds, (cont2) gives us $I_2 \rightarrow \bot$. If $\neg\ell$ holds, (cont1) gives us $I_1 \rightarrow \bot$ analogously. In both cases, $I_3 \equiv I_1 \land I_2$ is contradictory.

Case "otherwise".

By the definition of the projection function $\ell \downharpoonright A = \ell \downharpoonright B = \ell$ and $\neg\ell \downharpoonright A = \neg\ell \downharpoonright B = \neg\ell$ hold. Assuming $I_3 \equiv (I_1 \lor \ell) \land (I_2 \lor \neg\ell)$ holds, we prove a contradiction. If ℓ holds, the second conjunct of I_3 implies I_2. Then, (cont2) gives us a contradiction. If $\neg\ell$ holds, the first conjunct of I_3 implies I_1 and (cont1) gives us a contradiction. □

3.2.2 Projection of Mixed Literals

The proofs generated by state-of-the-art SMT solvers may contain mixed literals. We tackle them by extending the projection functions to these literals. The problem here is that there is no projection function that satisfies the conditions stated in the previous section. Therefore, we relax the conditions by allowing fresh auxiliary variables to occur in the projections.

We consider two different kinds of mixed literals: First, (dis-)equalities of the form $a = b$ or $a \neq b$ for an A-local variable a and a B-local variable b are introduced, e. g., by theory combination [75] or Ackermannization [11]. Second, inequalities of the form $a + b \leq c$ are introduced, e. g., by extended branches [35] or bound propagation. Here, a is a linear combination of A-local variables, b is a linear combination of B-local and shared variables, and c is a constant. Adding the shared variables to the B-part is an arbitrary choice. One gets interpolants of different strengths by assigning some shared variables to the A-part. It is only important to keep the projection of each literal consistent throughout the proof.

We split mixed literals using auxiliary variables, which we denote by x or p in the following. As subscript to the variable, we use the literal whose projection introduced this variable. The variable p has the type Boolean, while x has the same type as the variables in the literal. One or two fresh variables are introduced for each mixed literal. We count these variables as shared between A and B. The purpose of the auxiliary variable x is to capture the shared value that needs to be propagated between A and B. When splitting a literal ℓ into A- and B-part, we require that $\ell \Leftrightarrow \exists x, p.(\ell \downharpoonright A) \wedge (\ell \downharpoonright B)$. We need the additional Boolean variable p to split the literal $a \neq b$ into two (nearly) symmetric parts. This is achieved by the definitions below.

$$\begin{aligned}
(a = b) \downharpoonright A &:= (a = x_{a=b}) & (a = b) \downharpoonright B &:= (x_{a=b} = b)\\
(a \neq b) \downharpoonright A &:= (p_{a\neq b} \oplus a = x_{a\neq b}) & (a \neq b) \downharpoonright B &:= (\neg p_{a\neq b} \oplus x_{a\neq b} = b)\\
(a + b \leq c) \downharpoonright A &:= (a + x_{a+b\leq c} \leq 0) & (a + b \leq c) \downharpoonright B &:= (-x_{a+b\leq c} + b \leq c)
\end{aligned}$$

Since the mixed variables are considered to be shared, we allow them to occur in the partial interpolant of a clause C. However, a variable may only occur if C contains the corresponding literal. This is achieved by a special interpolation rule for resolution steps where the pivot literal is mixed. The rules for the different mixed literals are the core of our proposed algorithm and will be introduced in the following sections.

Lemma 3.1 (Partial Interpolation). *Given a mixed literal ℓ with auxiliary variable(s)* $\mathbf{x}$ *and clauses $C_1 \vee \ell$ and $C_2 \vee \neg\ell$ with corresponding partial interpolants I_1 and I_2. Let $C_3 = C_1 \vee C_2$ be the result of a resolution step on $C_1 \vee \ell$ and $C_2 \vee \neg\ell$ with pivot ℓ. If a partial interpolant I_3 satisfies the symbol condition, and*

$$(\forall \mathbf{x}. (\neg\ell \downharpoonright A \to I_1) \wedge (\ell \downharpoonright A \to I_2)) \to I_3 \qquad \text{(ind)}$$
$$I_3 \to (\exists \mathbf{x}. (\neg\ell \downharpoonright B \wedge I_1) \vee (\ell \downharpoonright B \wedge I_2)) \qquad \text{(cont)}$$

then I_3 is a partial interpolant of C_3.

Proof. We need to show inductivity and contradiction for the partial interpolants.

Inductivity.

For this we use inductivity of I_1 and I_2:

$$A \wedge \neg C_1 \downharpoonright A \wedge \neg\ell \downharpoonright A \models I_1$$
$$A \wedge \neg C_2 \downharpoonright A \wedge \ell \downharpoonright A \models I_2$$

Since $\mathbf{x}$ does not appear in $C_1 \downharpoonright A$, $C_2 \downharpoonright A$ nor A, we can conclude

$$A \wedge \neg C_1 \downharpoonright A \models \forall \mathbf{x}. \neg\ell \downharpoonright A \to I_1$$
$$A \wedge \neg C_2 \downharpoonright A \models \forall \mathbf{x}. \ell \downharpoonright A \to I_2$$

Combining these and pulling the quantifier over the conjunction gives

$$A \wedge \neg C_1 \downharpoonright A \wedge \neg C_2 \downharpoonright A \models \forall \mathbf{x}. (\neg\ell \downharpoonright A \to I_1) \wedge (\ell \downharpoonright A \to I_2)$$

Using (ind), this shows that inductivity for I_3 holds:

$$A \wedge \neg C_1 \downharpoonright A \wedge \neg C_2 \downharpoonright A \models I_3.$$

Contradiction.

First, we show the contradiction property for I_3:

$$B \wedge \neg C_1 \lfloor B \wedge \neg C_2 \lfloor B \wedge I_3 \models \bot.$$

Assume the formulae on the left-hand side hold. From (cond) we can conclude that there is some **x** such that

$$(\neg \ell \lfloor B \wedge I_1) \vee (\ell \lfloor B \wedge I_2)$$

If the first disjunct is true we can derive the contradiction using the contradiction property of I_1:

$$B \wedge \neg C_1 \lfloor B \wedge \neg \ell \lfloor B \wedge I_1 \models \bot$$

Otherwise, the second disjunct holds and we can use the contradiction property of I_2

$$B \wedge \neg C_2 \lfloor B \wedge \ell \lfloor B \wedge I_2 \models \bot$$

This shows the contradiction property for I_3. □

It is important to state here that the given purification of a literal into two new literals is not a modification of the proof tree or any of its nodes. The proof tree would no longer be well-formed if we replaced a mixed literal by the disjunction or conjunction of the purified parts. The purification is only used to define partial interpolants of clauses. In fact, it is only used in the correctness proof of our method and is not even done explicitly in the implementation.

3.3 Uninterpreted Functions

In this section we will present the part of our algorithm that is specific to the theory *EUF*. The only mixed atom that is considered by this theory is $a = b$ where a is A-local and b is B-local.

3.3.1 Leaf Interpolation

The *EUF* solver is based on the congruence closure algorithm [34]. The theory lemmas are generated from conflicts involving a single disequality that is in contradiction to a path of equalities. Thus, the clause generated from such a conflict consists of a single equality literal and several disequality literals.

When computing partial interpolants of such theory lemmas, we internally split the mixed literals according to Section 3.2.2. Then we use an algorithm similar to [45] to compute an interpolant. This algorithm basically summarises the A-equalities that are adjacent on the path of equalities.

If the theory lemma contains a mixed equality $a = b$ (without negation), it corresponds to the single disequality in the conflict. This disequality is split into $p_{a \neq b} \oplus a = x_{a \neq b}$ and

$\neg p_{a\neq b} \oplus x_{a\neq b} = b$ and the resulting interpolant depends on the value of $p_{a\neq b}$. If $p_{a\neq b} = \bot$, the disequality is part of the B-part and $x_{a\neq b}$ is the end of an equality path summing up the equalities from A. Thus, the computed interpolant contains a literal of the form $x_{a\neq b} = s$. If $p_{a\neq b} = \top$, then the A-part of the literal is $a \neq x_{a\neq b}$, and the resulting interpolant contains the literal $x_{a\neq b} \neq s$ instead. Thus, the resulting interpolant can be put into the form $I[p_{a\neq b} \oplus x_{a\neq b} = s]$. Note that the formula $p_{a\neq b} \oplus x_{a\neq b} = s$ occurs positively in the interpolant and is the only part of the interpolant containing $x_{a\neq b}$ and $p_{a\neq b}$. We define

$$EQ(x_{a\neq b}, s) := (p_{a\neq b} \oplus x_{a\neq b} = s)$$

and require that the partial interpolant of a clause containing the literal $a = b$ always has the form $I[EQ(x_{a\neq b}, s)]$ where $x_{a\neq b}$ and $p_{a\neq b}$ do not occur anywhere else.

For theory lemmas containing the literal $a \neq b$, the corresponding auxiliary variable $x_{a=b}$ may appear anywhere in the partial interpolant, even under a function symbol. A simple example is the theory conflict $s \neq f(a) \land a = (x_{a=b} =)b \land f(b) = s$, which has the partial interpolant $s \neq f(x_{a=b})$. In general the partial interpolant of such a clause has the form $I(x_{a=b})$.

When two partial interpolants for clauses containing $a = b$ are combined using (rule-res), i. e., the pivot literal is a non-mixed literal but the mixed literal $a = b$ occurs in C_1 and C_2, the resulting partial interpolant may contain $EQ(x_{a\neq b}, s_1)$ and $EQ(x_{a\neq b}, s_2)$ for different shared terms s_1, s_2. In general, we allow the partial interpolants to have the form $I[EQ(x, s_1)] \ldots [EQ(x, s_n)]$.

3.3.2 Pivoting of Mixed Equalities

We require that every clause C containing $a = b$ with auxiliary variables $x_{a\neq b}, p_{a\neq b}$ is always labelled with a formula of the form $I[EQ(x_{a\neq b}, s_1)] \ldots [EQ(x_{a\neq b}, s_n)]$. As discussed above, the partial interpolants computed for conflicts in the congruence closure algorithm are of the form $I[EQ(x_{a\neq b}, s_1)] \ldots [EQ(x_{a\neq b}, s_n)]$. This property is also preserved by (rule-res), and by Theorem 3.1 this rule also preserves the property of being a partial interpolant. On the other hand, a clause containing the literal $a \neq b$ is labelled with a formula of the form $I(x_{a=b})$, i. e., the auxiliary variable $x_{a=b}$ can occur at arbitrary positions. Again, the form $I(x_{a=b})$ and the property of being a partial interpolant is also preserved by (rule-res).

We use the following rule to interpolate the resolution step on the mixed literal $a = b$.

$$\frac{C_1 \lor a = b : I_1[EQ(x_{a\neq b}, s_1)] \ldots [EQ(x_{a\neq b}, s_n)] \qquad C_2 \lor a \neq b : I_2(x_{a=b})}{C_1 \lor C_2 : I_1[I_2(s_1)] \ldots [I_2(s_n)]} \quad \text{(rule-eq)}$$

The rule replaces every literal $EQ(x_{a\neq b}, s_i)$ in I_1 with the formula $I_2(s_i)$, in which every $x_{a=b}$ is substituted by s_i. Therefore, the auxiliary variables introduced for the mixed literal $a = b$ and its negation are removed.

Theorem 3.2 (Soundness of (rule-eq)). *Let $a = b$ and $a \neq b$ be mixed literals with auxiliary variables $x_{a\neq b}$, $p_{a\neq b}$ and $x_{a=b}$. If $I_1[EQ(x_{a\neq b}, s_1)] \ldots [EQ(x_{a\neq b}, s_n)]$ is a partial interpolant of $C_1 \lor a = b$ and $I_2(x_{a=b})$ a partial interpolant of $C_2 \lor a \neq b$ then $I_1[I_2(s_1)] \ldots [I_2(s_n)]$ is a partial interpolant of the clause $C_1 \lor C_2$.*

Proof. The symbol condition for $I_1[I_2(s_1)]\dots[I_2(s_n)]$ clearly holds if we assume that it holds for $I_1[EQ(x_{a\neq b},s_1)]\dots[EQ(x_{a\neq b},s_n)]$ and $I_2(x_{a=b})$. Hence, after we show (ind) and (cont), we can apply Lemma 3.1.

Inductivity.

We assume

$$\begin{aligned}\forall x,p.\ ((p\oplus a=x)\rightarrow I_1[p\oplus x=s_1]\dots[p\oplus x=s_n])\\ \wedge(a=x\rightarrow I_2(x))\end{aligned}$$

and show $I_1[I_2(s_1)]\dots[I_2(s_n)]$. Instantiating $x:=s_i$ for all $i\in\{1,\dots,n\}$ and taking the second conjunct gives $\bigwedge_{i\in\{1,\dots,n\}}(a=s_i\rightarrow I_2(s_i))$. Instantiating $p:=\bot$ and $x:=a$ and taking the first conjunct gives $I_1[a=s_1]\dots[a=s_n]$. With monotonicity we get $I_1[I_2(s_1)]\dots[I_2(s_n)]$ as desired.

Contradiction.

We have to show

$$\begin{aligned}&I_1[I_2(s_1)]\dots[I_2(s_n)]\rightarrow\\ &\quad\exists x,p.\ (((\neg p\oplus x=b)\wedge I_1[p\oplus x=s_1]\dots[p\oplus x=s_n])\\ &\quad\quad\vee(x=b\wedge I_2(x)))\end{aligned}$$

We show the implication for $p:=\top$ and $x:=b$. It simplifies to

$$I_1[I_2(s_1)]\dots[I_2(s_n)]\rightarrow I_1[b\neq s_1]\dots[b\neq s_n]\vee I_2(b)$$

If $I_2(b)$ holds the implication is true. If $I_2(b)$ does not hold, we have

$$\bigwedge_{i\in\{1,\dots,n\}}(I_2(s_i)\rightarrow b\neq s_i)$$

With monotonicity we get $I_1[I_2(s_1)]\dots[I_2(s_n)]\rightarrow I_1[b\neq s_1]\dots[b\neq s_n]$. □

3.4 Linear Real and Integer Arithmetic

Our solver for linear arithmetic is based on a variant of the Simplex approach [39]. A theory conflict is a conjunction of literals ℓ_j of the form $\sum_i a_{ij}x_i\leq b_j$. The proof of unsatisfiability is given by Farkas coefficients $k_j\geq 0$ for each inequality ℓ_j. These coefficients have the properties $\sum_j k_ja_{ij}=0$ and $\sum_j k_jb_j<0$. In the following we use the notation of adding inequalities (provided the coefficients are positive). Thus, we write $\sum_j k_j\ell_j$ for $\sum_i(\sum_j k_ja_{ij})x_i\leq\sum_j k_jb_j$.

With the property of the Farkas coefficients we get a contradiction ($0 < 0$) and this shows that the theory conflict is unsatisfiable.

A conjunction of literals may have rational but no integer solutions. In this case, there are no Farkas coefficients that can prove the unsatisfiability. So for the integer case, our solver may introduce extended branches [35], which are just branches of the DPLL engine on newly introduced literals. In the proof tree this results in resolution steps with these literals as pivots.

Example 3.1. The formula $t \leq 2a \leq r \leq 2b+1 \leq t$ has no integer solution but a rational solution. Introducing the branch $a \leq b \vee b < a$ leads to the theory conflicts $t \leq 2a \leq 2b \leq t-1$ and $r \leq 2b+1 \leq 2a-1 \leq r-1$ (note that $\neg(a \leq b) \equiv b < a$ is equivalent to $b+1 \leq a$). The corresponding proof tree is given below. The Farkas coefficients in the theory lemmas are given in parenthesis. Note that the proof tree shows the clauses, i. e., the negated conflicts. A node with more than two parents denotes that multiple applications of the resolution rule are taken one after another.

$$\begin{tikzcd}
\begin{array}{r}\neg(r \leq 2b+1)\ (\cdot 1)\\ \neg(b+1 \leq a)\ (\cdot 2)\\ \neg(2a \leq r)\ (\cdot 1)\end{array} \arrow[dr, no head] & r \leq 2b+1 \arrow[d, no head] & 2a \leq r \arrow[dl, no head] & \begin{array}{r}\neg(t \leq 2a)\ (\cdot 1)\\ \neg(a \leq b)\ (\cdot 2)\\ \neg(2b+1 \leq t)\ (\cdot 1)\end{array} \arrow[dr, no head] & t \leq 2a \arrow[d, no head] & 2b+1 \leq t \arrow[dl, no head] \\
 & a \leq b \arrow[dr, no head] & & & \neg(a \leq b) \arrow[dll, no head] & \\
 & & \perp & & &
\end{tikzcd}$$

Now consider the problem of deriving an interpolant between $A \equiv t \leq 2a \leq r$ and $B \equiv r \leq 2b+1 \leq t$. We can obtain an interpolant by annotating the above resolution tree with partial interpolants. To compute a partial interpolant for the theory lemma $\neg(r \leq 2b+1) \vee \neg(b+1 \leq a) \vee \neg(2a \leq r)$, we project the *negated* clause according to the definition in Section 3.2.2, which gives

$$r \leq 2b+1 \wedge x_{\neg(a \leq b)} \leq a \wedge -x_{\neg(a \leq b)} + b + 1 \leq 0 \wedge 2a \leq r.$$

Then, we sum up the A-part of the conflict (the second and fourth literal) multiplied by their corresponding Farkas coefficients. This yields the interpolant $2x_{\neg(a \leq b)} \leq r$. Similarly, the negation of the theory lemma $\neg(t \leq 2a) \vee \neg(a \leq b) \vee \neg(2b+1 \leq t)$ is purified to

$$t \leq 2a \wedge x_{a \leq b} + a \leq 0 \wedge -x_{a \leq b} \leq b \wedge 2b+1 \leq t,$$

which yields the partial interpolant $2x_{a \leq b} + t \leq 0$. Note, that we have to introduce different variables for each literal. Intuitively, the variable $x_{\neg(a \leq b)}$ stands for a and $x_{a \leq b}$ for $-a$. Using Pudlák's algorithm we can derive the same interpolants for the clause $a \leq b$ resp. $\neg(a \leq b)$.

For the final resolution step, the two partial interpolants $2x_{\neg(a \leq b)} \leq r$ and $2x_{a \leq b} + t \leq 0$ are combined into the final interpolant of the problem. Summing up these inequalities with $x_{\neg(a \leq b)} = -x_{a \leq b}$ we get $t \leq r$. While this follows from A, it is not inconsistent with B. We need an additional argument that, given $r = t$, r has to be an even integer. This also follows from the partial interpolants when setting $x_{\neg(a \leq b)} = -x_{a \leq b}$: $t \leq -2x_{a \leq b} = 2x_{\neg(a \leq b)} \leq r$. The final interpolant computed by our algorithm is $t \leq 2\left\lfloor \frac{r}{2} \right\rfloor$.

In general, we can derive additional constraints on the variables if the constraint resulting from summing up the two partial interpolants holds very tightly. We know implicitly that $x_{\neg(a \leq b)} = -x_{a \leq b}$ is an integer value between $t/2$ and $r/2$. If t equals r or almost equals r there are only a few possible values which we can explicitly express using the division function as in the example above. We assume that the (partial) interpolant F always has a certain property. There is some term s and some constant k, such that for $s > 0$ the interpolant is always false and for $s < -k$ the interpolant is always true (in our case $s = t - r$ and $k = 0$). For a partial interpolant that still contains auxiliary variables $\mathbf{x}$, we additionally require that s contains them with a positive coefficient and that F is monotone on $\mathbf{x}$, i. e., $\mathbf{x} \geq \mathbf{x}'$ implies $F(\mathbf{x}) \rightarrow F(\mathbf{x}')$. ⌟

To mechanise the reasoning used in the example above, our resolution rule for mixed inequality literals requires that the interpolant patterns that label the clauses have a certain shape. An auxiliary variable of a mixed inequality literal may only occur in the interpolant pattern if the negated literal appears in the clause. Let $\mathbf{x}$ denote the set of auxiliary variables that occur in the pattern. We require that these variables only occur inside a special sub-formula of the form $LA(s(\mathbf{x}), k, F(\mathbf{x}))$. The first parameter s is a linear term over the variables in $\mathbf{x}$ and arbitrary other terms not involving $\mathbf{x}$. The coefficients of the variables $\mathbf{x}$ in s must all be positive. The second parameter $k \in \mathbb{Q}_\varepsilon$ is a constant value. In the real case we only allow the values 0 and $-\varepsilon$. In the integer case we allow $k \in \mathbb{Z}, k \geq -1$. To simplify the presentation, we sometimes write $-\varepsilon$ for -1 in the integer case. The third parameter $F(\mathbf{x})$ is a formula that contains the variables from $\mathbf{x}$ at arbitrary positions. We require that F is monotone, i. e., $\mathbf{x} \geq \mathbf{x}'$ implies $F(\mathbf{x}) \rightarrow F(\mathbf{x}')$. Moreover, $F(\mathbf{x}) = \bot$ for $s(\mathbf{x}) > 0$ and $F(\mathbf{x}) = \top$ for $s(\mathbf{x}) < -k$. We refer to these two conditions as *range condition*. The sub-formula $LA(s(\mathbf{x}), k, F(\mathbf{x}))$ stands for $F(\mathbf{x})$ and it is only used to remember the values of s and k.

The intuition behind the formula $LA(s(\mathbf{x}), k, F(\mathbf{x}))$ is that $s(\mathbf{x}) \leq 0$ summarises the inequality chain that follows from the A-part of the formula. On this chain there may be some constraints on intermediate values. In the example above the A-part contains the chain $t \leq 2a \leq r$, which is summarised to $s \leq 0$ (with $s = t - r$). Furthermore the A-part implies that there is an even integer value between t and r. If $s < -k$ (with $k = 0$ in this case), t and r are distinct, and there always is an even integer between them. However, if $-k \leq s \leq 0$, the truth value of the interpolant depends on whether t is even.

In the remainder of the section, we will give the interpolants for the leaves produced by the linear arithmetic solver and for the resolvent of the resolution step where the pivot is a mixed linear inequality.

3.4.1 Leaf Interpolation

As mentioned above, our solver produces for a clause $C \equiv \neg\ell_1 \vee \cdots \vee \neg\ell_m$ some Farkas coefficients $k_1, \ldots, k_m \geq 0$ such that $\sum_j k_j \ell_j$ yields a contradiction $0 < 0$. A partial interpolant for a theory lemma can be computed by summing up the A-part of the conflict: I is defined as $\sum_j k_j(\ell_j \downharpoonright A)$ (if $\ell_j \downharpoonright A = \top$ we regard it as $0 \leq 0$, i. e., it is not added to the sum). It is a valid interpolant as it clearly follows from $\neg C \downharpoonright A \iff \ell_1 \downharpoonright A \wedge \cdots \wedge \ell_m \downharpoonright A$. Moreover, we have that $I + \sum_j k_j(\ell_j \downharpoonright B)$ yields $0 < 0$, since for every literal, even for mixed literals, $\ell_j \downharpoonright A + \ell_j \downharpoonright B = \ell_j$ holds[2]. This shows that $I \wedge \neg C \downharpoonright B$ is unsatisfiable.

The linear constraint $\sum_j k_j(\ell_j \downharpoonright A)$ can be expressed as $s(\mathbf{x}) \leq 0$. Thus, we can equivalently write this interpolant in our pattern as $LA(s(\mathbf{x}), -\varepsilon, s(\mathbf{x}) \leq 0)$. Since the Farkas coefficients

[2] Strictly speaking this does not hold for shared literals, where $\ell \downharpoonright A = \ell \downharpoonright B = \ell$. In that case use $k_j = 0$ in $I + \sum_j k_j(\ell_j \downharpoonright B)$ to see that I is indeed a partial interpolant.

are all positive and the auxiliary variables introduced to define $\ell \downharpoonright A$ for mixed literals contain x positively, the resulting term $s(\mathbf{x})$ will also always contain x with a positive coefficient.

Theory combination lemmas.

As mentioned in the preliminaries, we use theory combination clauses to propagate equalities from and to the Simplex core of the linear arithmetic solver. These clauses must also be labelled with partial interpolants. In the following we give interpolants for those theory combination lemmas. We will start with the case where no mixed literals occur, and treat lemmas containing mixed literals afterwards.

Interpolation of Non-Mixed Theory Combination Lemmas.

If a theory combination lemma $t = u \vee t < u \vee t > u$ or $t \neq u \vee t \leq u$ contains no mixed literal, we can compute partial interpolants as follows. If all literals in the clause are A-local, the formula $\bot$ is a partial interpolant. If all literals are B-local, the formula $\top$ is a partial interpolant. These are the same interpolants Pudlák's algorithm would give for input clauses from A resp. B.

Otherwise, one of the literals belongs to A and one to B. The symbols t and u have to be shared between A and B since they appear in all literals. We can derive a partial interpolant by conjoining the negated literals projected to the A partition.

$$I \equiv (t \neq u) \downharpoonright A \wedge (t \geq u) \downharpoonright A \wedge (t \leq u) \downharpoonright A. \qquad \text{for } t = u \vee t < u \vee t > u$$
$$I \equiv (t = u) \downharpoonright A \wedge (t > u) \downharpoonright A \qquad \text{for } t \neq u \vee t \leq u$$

Since we defined I as $\neg C \downharpoonright A$, the first property of the partial interpolant holds trivially. Also $I \wedge \neg C \downharpoonright B$ is equivalent to $\neg C$ and therefore false. The symbol condition is satisfied as t and u are shared symbols.

Interpolation of AB-Mixed Theory Combination Lemmas.

If we are in the mixed case, all three literals are mixed. One of the two terms must be A-local (in the following we denote this term by a) the other term B-local (which we denote by b). To purify the literals, we introduce a fresh auxiliary variable for each literal. Table 3.1 depicts all possible mixed theory lemmas together with the projections $\neg C \downharpoonright A$ and $\neg C \downharpoonright B$ and a partial interpolant of the clause.

Lemma 3.2. *The interpolants shown in Table 3.1 are correct partial interpolants of their respective clauses.*

Proof. First, we convince ourselves that these interpolants are of the right form: The variables $x_{a \leq b}$, $x_{b \leq a}$, $x_{\neg(a \leq b)}$, and $x_{\neg(b \leq a)}$ appear in the first parameter of LA with positive coefficients. For the first two clauses that contain the literal $a \neq b$, the interpolant is allowed to contain

Clause C: $a \neq b \lor a \leq b$	Clause C: $a \neq b \lor b \leq a$
$\neg C \downharpoonright A$: $a = x_{a=b} \land -a + x_{\neg(a \leq b)} \leq 0$	$\neg C \downharpoonright A$: $a = x_{a=b} \land a + x_{\neg(b \leq a)} \leq 0$
$\neg C \downharpoonright B$: $x_{a=b} = b \land -x_{\neg(a \leq b)} + b < 0$	$\neg C \downharpoonright B$: $x_{a=b} = b \land -x_{\neg(b \leq a)} - b < 0$
Interpolant: $LA(-x_{a=b} + x_{\neg(a \leq b)}, -\varepsilon, x_{\neg(a \leq b)} \leq x_{a=b})$	Interpolant: $LA(x_{a=b} + x_{\neg(b \leq a)}, -\varepsilon, x_{a=b} \leq -x_{\neg(b \leq a)})$

Clause C: $a = b \lor a < b \lor b < a$
$\neg C \downharpoonright A$: $(p_{a \neq b} \oplus a = x_{a \neq b}) \land -a + x_{b \leq a} \leq 0 \land a + x_{a \leq b} \leq 0$
$\neg C \downharpoonright B$: $(\neg p_{a \neq b} \oplus x_{a \neq b} = b) \land -x_{b \leq a} + b \leq 0 \land -x_{a \leq b} - b \leq 0$
Interpolant: $LA(x_{b \leq a} + x_{a \leq b}, 0, x_{b \leq a} \leq -x_{a \leq b} \land (x_{b \leq a} \geq -x_{a \leq b} \to EQ(x_{a \neq b}, x_{b \leq a})))$

Table 3.1 Interpolation of mixed theory combination clauses. We assume a is A-local and b is B-local. The subscripts of the auxiliary variables indicate the literal whose projection introduced this variable. For readability, we use $a < b$ and $b < a$ in the trichotomy clause instead of $\neg(b \leq a)$ and $\neg(a \leq b)$.

$x_{a=b}$ at arbitrary positions. Note that in the first interpolant $x_{\neg(a \leq b)} \leq x_{a=b}$ is false for $-x_{a=b} + x_{\neg(a \leq b)} > 0$ and true for $-x_{a=b} + x_{\neg(a \leq b)} < \varepsilon$, i. e., $-x_{a=b} + x_{\neg(a \leq b)} \leq 0$. Also, $x_{\neg(a \leq b)} \geq x$ implies $x_{\neg(a \leq b)} \leq x_{a=b} \to x \leq x_{a=b}$. Similarly, for the second interpolant.

In the third clause, $F(x_{b \leq a}, x_{a \leq b}) = x_{b \leq a} \leq -x_{a \leq b} \land (x_{b \leq a} \geq -x_{a \leq b} \to EQ(x_{a \neq b}, x_{b \leq a}))$ is false for $x_{b \leq a} + x_{a \leq b} > 0$ (because of the first conjunct) and true for $x_{b \leq a} + x_{a \leq b} < 0$ (because the implication holds vacuously). Also, $x_{b \leq a} \geq x_1$ and $x_{a \leq b} \geq x_2$ implies $F(x_{b \leq a}, x_{a \leq b}) \to F(x_1, x_2)$. To see this, note that $F(x_{b \leq a}, x_{a \leq b})$ is false if $x_1 \geq -x_2$ and $x_1 \neq x_{b \leq a}$. The variable $x_{a \neq b}$ appears only in an EQ-term which occurs positively in the partial interpolant.

Next we show

$$\neg C \downharpoonright A \models I \qquad \text{(Inductivity)}$$
$$\neg C \downharpoonright B \land I \models \bot \qquad \text{(Contradiction)}$$

Inductivity.

For the clause $a \neq b \lor a \leq b$, the interpolant follows from $\neg C \downharpoonright A$, as $a = x_{a=b}$ and $-a + x_{\neg(a \leq b)} \leq 0$ imply $x_{\neg(a \leq b)} \leq x_{a=b}$. Similarly for the clause $a \neq b \lor a \geq b$, $\neg C \downharpoonright A$ contains $a = x_{a=b}$ and $a + x_{\neg(b \leq a)} \leq 0$, which implies $x_{a=b} \leq -x_{\neg(b \leq a)}$.

Now consider the clause $a = b \lor a < b \lor b < a$. Here, $\neg C \downharpoonright A$ implies $x_{b \leq a} \leq -x_{a \leq b}$ and that if $x_{b \leq a} \geq -x_{a \leq b}$ holds, then $x_{b \leq a} = a = -x_{a \leq b}$. Hence, $x_{b \leq a} \leq -x_{a \leq b} \land x_{b \leq a} \geq -x_{a \leq b} \to EQ(x_{a \neq b}, x_{b \leq a})$ holds.

Contradiction.

Again we only show the first and third case. For the clause $C \equiv a \neq b \lor a \leq b$, note that $\neg C \downharpoonright B$ and $LA(-x_{a=b} + x_{\neg(a \leq b)}, -\varepsilon, x_{\neg(a \leq b)} \leq x_{a=b})$ give the contradiction $x_{\neg(a \leq b)} > b = x_{a=b} > x_{\neg(a \leq b)}$. For the clause $C \equiv a = b \lor a < b \lor b < a$, $\neg C \downharpoonright B$ implies $x_{b \leq a} \geq b \geq -x_{a \leq b}$. With $x_{b \leq a} \leq -x_{a \leq b}$ from the interpolant this gives $x_{b \leq a} = b$. Also, $x_{b \leq a} \geq -x_{a \leq b} \to EQ(x_{a \neq b}, x_{b \leq a})$ from the interpolant gives $p_{a \neq b} \oplus x_{a \neq b} = b$. This is in contradiction with $\neg p_{a \neq b} \oplus x_{a \neq b} = b$ from $\neg C \downharpoonright B$. □

3.4.2 Pivoting of Mixed Literals

In this section we give the resolution rule for a step involving a mixed inequality $a+b \leq c$ as pivot element.We use the auxiliary variables $x_{a+b\leq c}$ and $x_{\neg(a+b\leq c)}$ during projection of the mixed literals. The intuition is that $x_{\neg(a+b\leq c)}$ and $-x_{a+b\leq c}$ correspond to the same value between a and $c-b$.Since the clause contain a mixed literal, the partial interpolants contain terms of the form $LA(s(\mathbf{x}),k,F(\mathbf{x}))$. The partial interpolant of the result of a resolution step on the mixed literal $a+b \leq c$ is shown below. We will give details about s_3, k_3 and F_3 later.

$$\frac{\begin{array}{c}C_1 \vee a+b \leq c : I_1[LA(c_1 x_{\neg(a+b\leq c)} + s_1(\mathbf{x}), k_1, F_1(x_{\neg(a+b\leq c)}, \mathbf{x}))] \\ C_2 \vee \neg(a+b \leq c) : I_2[LA(c_2 x_{a+b\leq c} + s_2(\mathbf{x}), k_2, F_2(x_{a+b\leq c}, \mathbf{x}))]\end{array}}{C_1 \vee C_2 : I_1[I_2[LA(s_3(\mathbf{x}), k_3, F_3(\mathbf{x}))]]} \quad \text{(rule-la)}$$

The basic idea is to find for $\exists y. F_1(y,\mathbf{x}) \wedge F_2(-y,\mathbf{x})$ an equivalent quantifier-free formula $F_3(\mathbf{x})$. To achieve this we note that we only have to look on the value of F_1 for $-k_1 \leq c_1 x_{\neg(a+b\leq c)} + s_1(\mathbf{x}) \leq 0$, since outside of this interval F_1 is guaranteed to be true resp. false. The formula F_3 must also be monotone and satisfy the range condition. We choose

$$s_3(\mathbf{x}) = c_2 s_1(\mathbf{x}) + c_1 s_2(\mathbf{x}),$$

and then F_3 will be false for $s_3(\mathbf{x}) > 0$, since either $F_1(x_{\neg(a+b\leq c)}, \mathbf{x})$ or $F_2(-x_{\neg(a+b\leq c)}, \mathbf{x})$ is false. The value of k_3 must be chosen such that $s_3(\mathbf{x}) < -k_3$ guarantees the existence of a value y with $c_1 y + s_1(\mathbf{x}) < -k_1$ and $-c_2 y + s_2(\mathbf{x}) < -k_2$. Hence, in the integer case, the gap between $\frac{s_2(\mathbf{x})+k_2}{c_2}$ and $\frac{-s_1(\mathbf{x})-k_1}{c_1}$ should be bigger than one. Then, $c_1 c_2 < c_2(-s_1(\mathbf{x}) - k_1) - c_1(s_2(\mathbf{x}) + k_2)$. So if we define

$$k_3 = c_2 k_1 + c_1 k_2 + c_1 c_2,$$

then there is a suitable y for $s_3(\mathbf{x}) < -k_3$. For F_3 we can then use a finite case distinction over all values where the truth value of F_1 is not determined. This suggests defining

$$F_3(\mathbf{x}) :\equiv \bigvee_{i=0}^{\left\lceil \frac{k_1+1}{c_1} \right\rceil} F_1\left(\left\lfloor \frac{-s_1(\mathbf{x})}{c_1} \right\rfloor - i, \mathbf{x}\right) \wedge F_2\left(i - \left\lfloor \frac{-s_1(\mathbf{x})}{c_1} \right\rfloor, \mathbf{x}\right) \quad \text{(int case)}$$

In the real case, if $k_1 = -\varepsilon$, the best choice is $y = \frac{-s_1(\mathbf{x})}{c_1}$, for which $F_1(y)$ is guaranteed to be true. If $k_1 = 0$, we need to consider two cases:

$$\begin{aligned} k_3 &:= \begin{cases} k_2 & \text{if } k_1 = -\varepsilon \\ 0 & \text{if } k_1 = 0 \end{cases} \\ F_3(\mathbf{x}) &:= \begin{cases} F_2\left(\frac{s_1(\mathbf{x})}{c_1}, \mathbf{x}\right) & \text{if } k_1 = -\varepsilon \\ s_3(\mathbf{x}) < 0 \vee \left(F_1\left(-\frac{s_1(\mathbf{x})}{c_1}, \mathbf{x}\right) \wedge F_2\left(\frac{s_1(\mathbf{x})}{c_1}, \mathbf{x}\right)\right) & \text{if } k_1 = 0 \end{cases} \end{aligned} \quad \text{(real case)}$$

Note that the formula of the integer case is asymmetric. If $\left\lceil \frac{k_2+1}{c_2} \right\rceil < \left\lceil \frac{k_1+1}{c_1} \right\rceil$ we can replace $-s_1$ by s_2, k_1 by k_2, and c_1 by c_2. This leads to a fewer number of disjuncts in F_3. Also note that we can remove F_1 from the last disjunct of F_3, as it will always be true.

With these definitions we can state the following lemma.

Lemma 3.3. *Let for $i = 1,2$, $s_i(\mathbf{x})$ be linear terms over $\mathbf{x}$, $c_i \geq 0$, $k_i \in \mathbb{Z}_{\geq -1}$ (integer case) or $k_i \in \{0, -\varepsilon\}$ (real case), $F_i(x_i, \mathbf{x})$ monotone formulas with $F_i(x_i, \mathbf{x}) = \bot$ for $c_i x_i + s_i(\mathbf{x}) > 0$ and $F_i(x_i, \mathbf{x}) = \top$ for $c_i x_i + s_i(\mathbf{x}) < -k_i$. Let s_3, k_3, F_3 be as defined above. Then F_3 is monotone, $F_3(\mathbf{x}) = \bot$ for $s_3(\mathbf{x}) > 0$ and $F_3(\mathbf{x}) = \top$ for $s_3(\mathbf{x}) < -k_i$.*

Proof. Since F_1 and F_2 are monotone and they occur only positively in F_3, F_3 must also be monotone. If $s_3(\mathbf{x}) > 0$, then $\frac{-s_1(\mathbf{x})}{c_1} < \frac{s_2(\mathbf{x})}{c_2}$. Hence, for every $x \leq \frac{-s_1(\mathbf{x})}{c_1}$, $F_2(-x, \mathbf{x})$ is false since $-c_2 x + s_2(\mathbf{x}) > 0$. By definition, every disjunct of F_3 (except $s_3(\mathbf{x}) < 0$) contains $F_2(-x, \mathbf{x})$ for such an x, so $F_3(\mathbf{x})$ is false.

Now assume $s_3(\mathbf{x}) < -k_3$. For $k_1 = -\varepsilon$ in the real case, $F_3(\mathbf{x}) = F_2(-\frac{s_1(\mathbf{x})}{c_1})$ is true since $s_1(\mathbf{x}) + s_2(\mathbf{x}) < -k_2$. For $k_1 = 0$, F_3 is true by definition. In the integer case define $y := \left\lfloor \frac{-s_1(\mathbf{x})}{c_1} \right\rfloor - \left\lceil \frac{k_1+1}{c_1} \right\rceil$. This implies $c_1 y \leq -s_1(\mathbf{x}) - k_1 - 1$, hence $F_1(y, \mathbf{x})$ holds. Also $c_1 y \geq -s_1(\mathbf{x}) - k_1 - c_1$, hence

$$c_1 c_2 y + c_1 s_2(\mathbf{x}) \geq -s_3(\mathbf{x}) - c_2 k_1 - c_1 c_2 > k_3 - c_2 k_1 - c_1 c_2 = c_1 k_2.$$

Therefore, $F_2(-y, \mathbf{x})$ holds. Since y is included in the big disjunction of F_3, $F_3(\mathbf{x})$ is true. □

Lemma 3.4. *Let for $i = 1,2$, $s_i(\mathbf{x})$ be linear terms over $\mathbf{x}$, $c_i \geq 0$, $k_i \in \mathbb{Z}_{\geq -1}$ (integer case) or $k_i \in \{0, -\varepsilon\}$ (real case), $F_i(x_i, \mathbf{x})$ monotone formulas with $F_i(x_i, \mathbf{x}) = \bot$ for $c_i x_i + s_i(\mathbf{x}) > 0$ and $F_i(x_i, \mathbf{x}) = \top$ for $c_i x_i + s_i(\mathbf{x}) < -k_i$. Let $s_3(\mathbf{x})$ and F_3 be as defined above. Then*

$$F_3(\mathbf{x}) \leftrightarrow (\exists y. F_1(y, \mathbf{x}) \wedge F_2(-y, \mathbf{x}))$$

Proof (for LA$(\mathbb{Z})$). Since F_3 is a disjunction of $F_1(y, \mathbf{x}) \wedge F_2(-y, \mathbf{x})$ for different values of y, the implication from left to right is obvious. We only need to show the other direction. For this, choose y such that $F_1(y, \mathbf{x}) \wedge F_2(-y, \mathbf{x})$ holds. We show $F_3(\mathbf{x})$. We define $z := \left\lfloor \frac{-s_1(\mathbf{x})}{c_1} \right\rfloor - \left\lceil \frac{k_1+1}{c_1} \right\rceil$. This implies $z \leq \frac{-s_1(\mathbf{x}) - k_1 - 1}{c_1}$. We show F_3 by a case split on $y < z$.

Case $y < z$.

Since F_2 is monotone and $-y > -z$, we have $F_2(-z, \mathbf{x})$. Also $F_1(z, \mathbf{x})$ holds since $c_1 z + s_1(\mathbf{x}) < -k_1$. This implies $F_3(\mathbf{x})$, since $F_1(z, \mathbf{x}) \wedge F_2(-z, \mathbf{x})$ is a disjunct of F_3.

Case $z \leq y$.

Since $F_1(y, \mathbf{x})$ holds, $c_1 y + s_1(\mathbf{x}) \leq 0$, hence $y \leq \left\lfloor \frac{-s_1(\mathbf{x})}{c_1} \right\rfloor$. Thus, y is one of the values $\left\lfloor \frac{-s_1(\mathbf{x})}{c_1} \right\rfloor - i$ for $0 \leq i \leq \left\lceil \frac{k_1+1}{c_1} \right\rceil$. This means the disjunction $F_3(\mathbf{x})$ includes $F_1(y, \mathbf{x}) \wedge F_2(-y, \mathbf{x})$. □

Proof (for $LA(\mathbb{Q})$). In the case $k_1 = -\varepsilon$, $F_1(\frac{-s_1(\mathbf{x})}{c_1}, \mathbf{x})$ is true. From the definition of F_3, we get the implication $F_3(\mathbf{x}) \to \exists y. F_1(y,\mathbf{x}) \wedge F_2(-y,\mathbf{x})$ for $y = \frac{-s_1(\mathbf{x})}{c_1}$. If $k_1 = 0$ and $s_3(\mathbf{x}) < 0$, then $\frac{s_2(\mathbf{x})}{c_2} < \frac{-s_1(\mathbf{x})}{c_1}$ and for any value y in between, $F_1(y,\mathbf{x}) \wedge F_2(-y,\mathbf{x})$ are true.

For the other direction assume that $F_1(y,\mathbf{x}) \wedge F_2(-y,\mathbf{x})$ holds. Since F_1 is not false, $y \leq \frac{-s_1(\mathbf{x})}{c_1}$ holds. If $y = \frac{-s_1(\mathbf{x})}{c_1}$ then F_3 holds by definition. In the case $k_1 = 0$ where $y < \frac{-s_1(\mathbf{x})}{c_1}$, we have $s_3(\mathbf{x}) < 0$, since $F_2(-y,\mathbf{x})$ is not false. In the case $k_1 = -\varepsilon$, we need to show that $F_2(\frac{s_1(\mathbf{x})}{c_1}, \mathbf{x})$ holds. This follows from $y \leq \frac{-s_1(\mathbf{x})}{c_1}$ and monotonicity of F_2. □

This lemma can be used to show that (rule-la) is correct.

Theorem 3.3 (Soundness of (rule-la)). *Let $a+b \leq c$ be a mixed literal with the auxiliary variable $x_{a+b\leq c}$, and $x_{\neg(a+b\leq c)}$ be the auxiliary variable of the negated literal. If $I_1[LA(c_1 x_{\neg(a+b\leq c)} + s_1, k_1, F_1)]$ is a partial interpolant of $C_1 \vee a+b \leq c$ and $I_2[LA(c_2 x_{a+b\leq c} + s_2, k_2, F_2)]$ is a partial interpolant of $C_2 \vee \neg(a+b \leq c)$ then $I_1[I_2[LA(s_3,k_3,F_3)]]$ is a partial interpolant of the clause $C_1 \vee C_2$.*

To ease the presentation, we gave the rule (rule-la) with only one LA term per partial interpolant. The generalised rule requires the partial interpolants of the premises to have the shapes $I_1[LA_{11}]\ldots[LA_{1n}]$ and $I_2[LA_{21}]\ldots[LA_{2m}]$. The resulting interpolant is

$$I_1[I_2[LA_{311}]\ldots[LA_{31m}]]\ldots[I_2[LA_{3n1}]\ldots[LA_{3nm}]]$$

where LA_{3ij} is computed from LA_{1i} and LA_{2j} as explained above.

Proof. The symbol condition holds for I_3 if it holds for I_1 and I_2, which can be seen as follows. The only symbol that is allowed to occur in I_1 resp. I_2 but not in I_3 is the auxiliary variable introduced by the literal, i. e., x_1 resp. x_2. This variable may only occur inside the LA_1 resp. LA_2 terms as indicated and, by construction, $x_{\neg(a+b\leq c)}$ and $x_{a+b\leq c}$ do not occur in LA_3. Furthermore, the remaining variables from $\mathbf{x}$ occur in $s_3(\mathbf{x})$ with a positive coefficient as required by our pattern and occur only inside the LA pattern in s_3 and F_3. Thus I_3 has the required form. We will use Lemma 3.1 to show that I_3 is a partial interpolant. For this we need to show inductivity (ind) and contradiction (cont).

In this proof we will use $I_1[LA_{1i}(x_{\neg(a+b\leq c)})]$ to denote the first interpolant

$$I_1[LA(s_{11} + c_{11}x_{\neg(a+b\leq c)}, k_{11}, F_{11})]\ldots[LA(s_{1n} + c_{1n}x_{\neg(a+b\leq c)}, k_{1n}, F_{1n})]$$

and similarly $I_2[LA_{2j}(x_{a+b\leq c})]$ and $I_1[I_2[LA_{3ij}]]$, the latter standing for

$$I_1[I_2[LA_{311}]\ldots[LA_{31m}]]\ldots[I_2[LA_{3n1}]\ldots[LA_{3nm}]]$$

where

$$LA_{3ij} = LA(c_{2j}s_{1i} + c_{1i}s_{2j}, k_{3ij}, F_{3ij}).$$

Inductivity.

We apply Lemma 3.4 on $x_{\neg(a+b\leq c)} = a$, which gives us

$$\bigwedge_{ij} LA_{1i}(a) \land LA_{2j}(-a) \to LA_{3ij}$$

Using the deep substitution lemma, we obtain

$$I_1 \left[LA_{1i}(a)\right] \land I_2 \left[LA_{2j}(-a)\right] \to I_1 \left[I_2 \left[LA_{3ij}\right]\right]. \qquad (*)$$

Now assume the left-hand-side of (ind), which in this case is

$$\forall x_1, x_2. (-a + x_1 \leq 0 \to I_1[LA_{1i}(x_1)]) \land (a + x_2 \leq 0 \to I_2[LA_{2j}(x_2)]).$$

Instantiating x_1 with a and x_2 with $-a$ gives us $I_1[LA_{1i}(a)]$ and $I_2[LA_{2j}(-a)]$. Thus by $(*)$, $I_3 \equiv I_1[I_2[LA_{3ij}]]$ holds as desired.

Contradiction.

We assume $I_1[I_2[LA_{3ij}]]$ and show

$$\exists x_1, x_2. (-x_1 - b < -c \land I_1[LA_{1i}(x_1)]) \lor (-x_2 + b \leq c \land I_2[LA_{2j}(x_2)]) \qquad (*)$$

We do a case distinction on

$$\bigwedge_{i} (I_2[LA_{3ij}] \to \exists x_1. x_1 > c - b \land LA_{1i}(x_1))$$

If it holds, then we may get a different value for x_1 for every i. However, if $LA_{1i}(x_1)$ holds for some value, it also holds for any smaller value of x_1. Take x as the minimum of these values (or $x = c - b + 1$ if the implication holds vacuously for every i). Then, $-x - b < -c$ and $\bigwedge_i (I_2[LA_{3ij}] \to LA_{1i}(x))$. With monotonicity we get from $I_1[I_2[LA_{3ij}]]$ that $I_1[LA_{1i}(x)]$ holds. Hence, the left disjunct of formula $(*)$ holds.

In the other case there is some i with

$$I_2[LA_{3ij}] \land (\forall x_1. x_1 > c - b \to \neg LA_{1i}(x_1)). \qquad (**)$$

The second part of Lemma 3.4 gives us

$$\bigwedge_{j} (LA_{3ij} \to \exists x_1. LA_{1i}(x_1) \land LA_{2j}(-x_1))$$

Then, $x_1 \leq c - b$ by $(**)$. But if $LA_{2j}(-x_1)$ holds, then LA_{2j} also holds for the smaller value $b - c$. This gives us

$$\bigwedge_{j} (LA_{3ij} \to LA_{2j}(b - c))$$

We obtain $I_2[LA_{2j}(b - c)]$ by applying monotonicity on the left conjunct of formula $(**)$. Thus the right disjunct of formula $(*)$ holds for $x_2 = b - c$. □

3.5 An Example for the Combined Theory

The previous examples showed how to use our technique to compute an interpolant in the theory of uninterpreted functions, or the theory of linear arithmetic. We will now present an example in the combination of these theories by applying our scheme to a proof of unsatisfiability of the interpolation problem

$$A \equiv t \le 2a \wedge 2a \le s \wedge f(a) = q$$
$$B \equiv s \le 2b \wedge 2b \le t+1 \wedge \neg(f(b) = q)$$

where a, b, s, and t are integer constants, q is a constant of the uninterpreted sort U, and f is an uninterpreted function from integer to U.

We derive the interpolant using Pudlák's algorithm and the rules shown in this chapter. Note that the formula is already in conjunctive normal form. Since we use Pudlák's algorithm, every input clause is labelled with $\bot$ if it is an input clause from A, and $\top$ if it is an input clause from B. We simplify partial interpolants by removing neutral elements of Boolean connectives.

Since the variables a and b are shared between the theory of uninterpreted functions and the theory of linear arithmetic, we get some theory combination clauses for a and b. The only theory combination clause needed to prove unsatisfiability of $A \wedge B$ is the trichotomy clause $a = b \vee \neg(b \le a) \vee \neg(a \le b)$ which has the partial interpolant $LA(x_{b\le a} + x_{a\le b}, 0, F[EQ(x_{a\neq b}, x_{b\le a})])$ where $F[G] \equiv x_{b\le a} \le -x_{a\le b} \wedge (x_{b\le a} \ge -x_{a\le b} \rightarrow G)$. The subscripts of the variables indicate the literal whose projection introduced this variable.

We get two lemmas from $LA(\mathbb{Z})$: The first one, $\neg(2a \le s) \vee \neg(s \le 2b) \vee a \le b$, states that we can derive $a \le b$ from $2a \le s$ and $s \le 2b$. We project the literals in the conflict, i. e., the negation of the lemma, onto A. Summing up the projections yields the partial interpolant $I_{a\le b} :\equiv LA(2x_{\neg(a\le b)} - s, -1, 2x_{\neg(a\le b)} \le s)$. We resolve this lemma with the unit clauses from the input to get $a \le b$.

$$\dfrac{s \le 2b : \top \qquad \dfrac{2a \le s : \bot \qquad \neg(2a \le s) \vee \neg(s \le 2b) \vee a \le b : I_{a\le b}}{\neg(s \le 2b) \vee a \le b : I_{a\le b}}}{a \le b : I_{a\le b}}$$

The second $LA(\mathbb{Z})$-lemma, $\neg(t \le 2a) \vee \neg(2b \le t+1) \vee b \le a$, states that we can derive $b \le a$ from $t \le 2a$ and $2b \le t+1$. We annotate the lemma with the partial interpolant $I_{b\le a} :\equiv LA(2x_{\neg(b\le a)} + t, -1, 2x_{\neg(b\le a)} + t \le 0)$ and propagate this partial interpolant to the unit clause $b \le a$ by resolution with input clauses.

$$\dfrac{2b \le t+1 : \top \qquad \dfrac{t \le 2a : \bot \qquad \neg(t \le 2a) \vee \neg(2b \le t+1) \vee b \le a : I_{b\le a}}{\neg(2b \le t+1) \vee b \le a : I_{b\le a}}}{b \le a : I_{b\le a}}$$

Additionally, we get one lemma from EUF, $f(b) = q \vee \neg(f(a) = q) \vee \neg(a = b)$, that states that, given $f(a) = q$ and $a = b$, by congruence, $f(b) = q$ has to hold. We annotate this lemma with the partial interpolant $f(x_{a=b}) = q$. Note that this interpolant has the form $I(x_{a=b})$ as required by our interpolation scheme. We propagate this partial interpolant to the unit clause

$\neg(a=b)$ by resolving the lemma with the input clauses.

$$\dfrac{f(a)=q:\bot \qquad \dfrac{f(b)=q:\top \qquad f(b)=q\vee\neg(f(a)=q)\vee\neg(a=b):f(x)=q}{\neg(f(a)=q)\vee\neg(a=b):f(x)=q}}{\neg(a=b):f(x)=q}$$

From the theory combination clause $a=b\vee\neg(b\leq a)\vee\neg(a\leq b)$ and the three unit clauses derived above, we show a contradiction. We start by resolving with the unit clause $a=b$ using (rule-eq) and produce the partial interpolant $LA(x_{b\leq a}+x_{a\leq b},0,F[f(x_{b\leq a})=q])$.

$$\dfrac{\begin{array}{c}a=b\vee\neg(b\leq a)\vee\neg(a\leq b):LA(x_{b\leq a}+x_{a\leq b},0,F[EQ(x_{a\neq b},x_{b\leq a})])\\ \neg(a=b):f(x_{a=b})=q\end{array}}{\neg(b\leq a)\vee\neg(a\leq b):LA(x_{b\leq a}+x_{a\leq b},0,F[f(x_{b\leq a})=q])}$$

The next step resolves on $b\leq a$ using (rule-la). The variables $x_{\neg(b\leq a)}$ and $x_{b\leq a}$ will be removed from the resulting partial interpolant. From the partial interpolants of the antecedents, $LA(2x_{\neg(b\leq a)}+t,-1,2x_{\neg(b\leq a)}+t\leq 0)$ and $LA(x_{b\leq a}+x_{a\leq b},0,F[f(x_{b\leq a})=q])$, we get the following components:

$c_1=2$	$s_1=t$	$k_1=-1$	$F_1(x)\equiv 2x+t\leq 0$
$c_2=1$	$s_2=x_{a\leq b}$	$k_2=0$	$F_2(x)\equiv F[f(x)=q]$

These components yield $k_3=1\cdot(-1)+2\cdot 0+2\cdot 1=1$. Furthermore, $\left\lceil\frac{k_1+1}{c_1}\right\rceil=0$ leads to one disjunct in F_3. The corresponding values are $\left\lfloor\frac{-t}{2}\right\rfloor$, resp. $-\left\lfloor\frac{-t}{2}\right\rfloor$. $F_1(\left\lfloor\frac{-t}{2}\right\rfloor)$ is always true and can be omitted. The resulting formula $G(x):=F_3(\mathbf{x})$ is

$$G(x)\equiv-\left\lfloor\frac{-t}{2}\right\rfloor\leq -x\wedge\left(\left\lfloor\frac{-t}{2}\right\rfloor\geq -x\rightarrow f\left(-\left\lfloor\frac{-t}{2}\right\rfloor\right)=q\right).$$

The partial interpolant for the clause $\neg(a\leq b)$ is $LA(t+2x_{a\leq b},1,G(x_{a\leq b}))$.

$$\dfrac{\begin{array}{c}b\leq a:LA(2x_{\neg(b\leq a)}+t,-1,2x_{\neg(b\leq a)}+t\leq 0)\\ \neg(b\leq a)\vee\neg(a\leq b):LA(x_{b\leq a}+x_{a\leq b},0,F[f(x_{b\leq a})=q])\end{array}}{\neg(a\leq b):LA(t+2x_{a\leq b},1,G(x_{a\leq b}))}$$

In the final resolution step, we resolve $a\leq b$ labelled with partial interpolant $LA(2x_{\neg(a\leq b)}-s,-1,2x_{\neg(a\leq b)}\leq s)$ against $\neg(a\leq b)$ labelled with $LA(t+2x_{a\leq b},1,G(x_{a\leq b}))$. We get the components

$c_1=2$	$s_1=-s$	$k_1=-1$	$F_1(x)\equiv 2x\leq s$
$c_2=2$	$s_2=t$	$k_2=1$	$F_2(x)\equiv G(x).$

We get $k_3=2\cdot(-1)+2\cdot 1+2\cdot 2=4$. Again, $\left\lceil\frac{k_1+1}{c_1}\right\rceil=0$ yields one disjunct in F_3 with the values $\left\lfloor\frac{s}{2}\right\rfloor$, and $-\left\lfloor\frac{s}{2}\right\rfloor$, respectively. Again, $F_1(\left\lfloor\frac{s}{2}\right\rfloor)$ is always true and can be omitted. The resulting formula is

$$\begin{aligned}H &\equiv G\left(-\left\lfloor\frac{s}{2}\right\rfloor\right)\\ &\equiv -\left\lfloor\frac{-t}{2}\right\rfloor \leq \left\lfloor\frac{s}{2}\right\rfloor \wedge \left(\left\lfloor\frac{-t}{2}\right\rfloor \geq \left\lfloor\frac{s}{2}\right\rfloor \rightarrow f\left(-\left\lfloor\frac{-t}{2}\right\rfloor\right) = q\right).\end{aligned}$$

The final resolution step yields an interpolant for this problem.

$$\frac{a \leq b : LA(2x_{\neg(a\leq b)} - s, -1, 2x_{\neg(a\leq b)} - s \leq 0) \qquad \neg(a \leq b) : LA(t + 2x_{a\leq b}, 1, G(x_{a\leq b}))}{\bot : LA(-2s + 2t, 4, H)}$$

Thus H is the final interpolant. Now we argue validity of this interpolant.

Interpolant follows from the A-part.

The A-part contains $2a \leq s$, which implies $a \leq \left\lfloor\frac{s}{2}\right\rfloor$. From $t \leq 2a$ we get $-\left\lfloor\frac{-t}{2}\right\rfloor \leq a$. Hence, $-\left\lfloor\frac{-t}{2}\right\rfloor \leq \left\lfloor\frac{s}{2}\right\rfloor$. Moreover, $-\left\lfloor\frac{-t}{2}\right\rfloor \geq \left\lfloor\frac{s}{2}\right\rfloor$ implies $-\left\lfloor\frac{-t}{2}\right\rfloor = a$. So with the A-part we get $f(-\left\lfloor\frac{-t}{2}\right\rfloor) = q$.

Interpolant is inconsistent with the B-part.

The B-part implies $s \leq 2b \leq t + 1$. Hence, we have $\left\lfloor\frac{s}{2}\right\rfloor \leq b \leq \left\lfloor\frac{t+1}{2}\right\rfloor$. A case distinction on whether t is even or odd yields $\left\lfloor\frac{t+1}{2}\right\rfloor = -\left\lfloor\frac{-t}{2}\right\rfloor$. Therefore, $\left\lfloor\frac{s}{2}\right\rfloor \leq b \leq -\left\lfloor\frac{-t}{2}\right\rfloor$ holds. The interpolant guarantees $f(-\left\lfloor\frac{-t}{2}\right\rfloor) = q$ and $-\left\lfloor\frac{-t}{2}\right\rfloor \leq \left\lfloor\frac{s}{2}\right\rfloor$. Hence, $b = -\left\lfloor\frac{-t}{2}\right\rfloor$ and with $f(b) \neq q$ from the B-part we get a contradiction.

Symbol condition is satisfied.

The symbol condition is trivially satisfied since $symb(A) = \{a,t,s,f,q\}$ and $symb(B) = \{b,t,s,f,q\}$. The shared symbols are t, s, f, and q which are exactly the symbols occurring in the interpolant.

Chapter 4
Tree Interpolants from Proofs

In this chapter, we extend proof tree preserving interpolation to extract tree interpolants from one proof of unsatisfiability of the input tree. This technique allows us to generate a tree interpolant without the need for repeated binary interpolation or repeated manipulation of a proof tree.

4.1 Tree Interpolation

A tree interpolation problem is given by a tree (V,E) where V is a set of vertices, and $E \subseteq V \times V$ is a set of edges, and a labelling function F assigning one formula to every vertex in V. The edges span a tree pointing to the root vertex, i. e., every vertex except the unique root vertex has exactly one outgoing edge and the graph has no cycles. Let E^* be the reflexive transitive closure of E. We define $st(v) := \{w \mid (w,v) \in E^*\}$ as the subtree rooted at v and $\overline{st(v)} := V \setminus st(v)$ as the complement of this subtree. For the remainder of this chapter, we fix a tree interpolation problem $(V,E),F$. If the conjunction of the labels of all vertices $\bigwedge_{v \in V} F(v)$ is unsatisfiable, a *tree interpolant* exists. A tree interpolant is a labelling function I that assigns a formula to every vertex in V. These formulas are constrained in a way similar to binary interpolation. Let $v_r \in V$ be the (unique) root of the tree. Then, the labelling function I has to satisfy (i) $I(v_r)$ is the formula $\bot$, (ii) for every vertex v, $(\bigwedge_{(v_c,v) \in E} I(v_c)) \wedge F(v) \models I(v)$, and (iii) for every vertex v, all symbols in $I(v)$ occur both inside the subtree rooted at v and outside this subtree, i. e., $symb(I(v)) \subseteq (\bigcup_{w \in st(v)} symb(F(w))) \cap (\bigcup_{w \notin st(v)} symb(F(w)))$. Let v_1, v_2 be two vertices in a tree. By $lca(v_1,v_2)$ we denote the *least common ancestor* v of v_1 and v_2, i. e., the vertex v with $v_1, v_2 \in st(v)$ that spans the smallest subtree. The function lca is associative and we allow it to be applied to a set of vertices.

Tree interpolation can be reduced to repeated binary interpolation. In this case a new binary interpolation problem is created for every vertex of the interpolation tree. Another possibility is to extract a tree interpolant from one proof using a form of “sliding window” approach. Instead of producing new binary interpolation problems, the proof is *coloured* according to the current vertex. Essentially, the proof is reinterpreted for every vertex as a proof for the corresponding binary interpolation problem. However, a naïve approach does not guarantee the tree interpolant property of the produced interpolants as the following example shows.

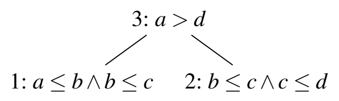


Fig. 4.1 Tree interpolation problem with a literal occurring in multiple vertices.

Example 4.1. Consider the tree interpolation problem shown in Figure 4.1. When deriving a tree interpolant by simply considering different partitioning of the same proof, possible interpolants include $a \leq b$ for vertex 1 and $c \leq d$ for vertex 2. The conjunction of these two interpolants and the label of vertex 3 ($a > d$) however is still consistent. Thus, these interpolants cannot be used to create a valid tree interpolant.

While the previous example can be seen as minimal summaries of the vertices 1 and 2, maximal summaries yielding $a \leq c$ and $b \leq d$ cannot be used to form a valid tree interpolant, too. The problem in both approaches is the literal $b \leq c$. It gets included in the interpolants either not at all or twice. Only including this literal exactly once, say in vertex 1 but not in vertex 2, yields a valid tree interpolant in this case with interpolants $a \leq c$ for vertex 1 and $c \leq d$ for vertex 2. The conjunction of these two interpolants with $a > d$ is inconsistent. Thus, we have a valid tree interpolant. ⌟

In the remainder of this chapter, we show how to extract a tree interpolant from a proof of unsatisfiability of the conjunction of the labels. We focus on the theories of uninterpreted functions and linear arithmetic. Furthermore we show how to compute tree interpolants in the combination of these theories. The basis is to carefully use the interpolation rules for binary interpolation for every vertex of the interpolation tree.

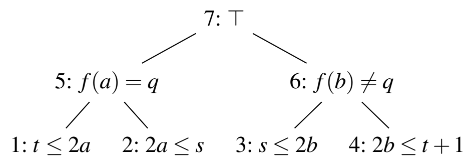


Fig. 4.2 Tree interpolation problem used in the remainder of this chapter.

Example 4.2. Consider the tree interpolation problem depicted in Figure 4.2. The tree consists of 7 vertices (numbered 1 through 7). The labelling function F is shown in the picture, i. e., vertex number 7 is labelled with $\top$. All constants are of integral sort and f is a function from integers to integers.

The conjunction of all the labels is inconsistent. Thus, a tree interpolant exists. Key to showing unsatisfiability of the conjunction of the labels is the case splitting on $a = b$. From vertices 5 and 6 we know that $a \neq b$ due to the congruence $a = b \rightarrow f(a) = f(b)$. From vertices 1 and 2 get know that $2a$ is an even integer between t and s. Furthermore, vertices 3 and 4 state that $2b$ is an even integer between s and $t+1$. From $t \leq 2a \leq s \leq 2b \leq t+1$ we get that $2a = 2b$ and thus $a = b$ since there is exactly one even integer between t and $t+1$. This leads to a contradiction and proves unsatisfiability of the conjunction of the labels. ⌟

4.1.1 Generality of Tree Interpolation

In the literature about interpolation, binary interpolation and sequence interpolation dominate. Tree interpolation, however, is more general. Thus, we show how to simulate binary interpolation and sequence interpolation by tree interpolation.

4.1.1.1 Binary Interpolation Through Tree Interpolation

Given a binary interpolation problem consisting of the formulas A and B, we generate a tree consisting of two vertices as shown in Figure 4.3.

Fig. 4.3 Tree interpolation problem corresponding to the binary interpolation problem (A,B).

It is easy to see that a tree interpolant exists if and only if a binary interpolant exists. To compute a binary interpolant using tree interpolation, we compute a tree interpolant and use the interpolant label of the vertex labelled with A.

Lemma 4.1. *Let (A,B) be a binary interpolation problem and let T,F be the corresponding tree interpolation problem defined above. Let v_A be the vertex labelled with A, v_B be the vertex labelled with B, and I be a tree interpolant for T. Then, $I(v_A)$ is a binary interpolant for (A,B).*

Proof. We have to show that $I(v_A)$ satisfies the conditions for a binary interpolant. From the conditions for tree interpolants we know that $F(v_A) \equiv A \models I(v_A)$ and that $F(v_B) \wedge I(v_A) \equiv B \wedge I(v_A) \models I(v_B) \equiv \bot$. Furthermore, the symbol condition of tree interpolation specifies that the symbols occurring in $I(v_A)$ are shared between A and B. Thus, $I(v_A)$ is a binary interpolant of (A,B). $\square$

4.1.1.2 Sequence Interpolation Through Tree Interpolation

A sequence interpolation problem is a generalisation of a binary interpolation problem. It consists of n formulas $F_1,\dots,F_n$. A sequence interpolant exists if $\bigwedge_{i=1}^{n} F_i \models \bot$. Then, a sequence interpolant is a sequence of $n+1$ formulas $I_0,\dots,I_n$ such that

- $I_0 \equiv \top$ and $I_n \equiv \bot$,
- for all $0 \leq i < n$, $I_i \wedge F_{i+1} \models I_{i+1}$, and
- for all $0 \leq i \leq n$, $symb(I_i) \subseteq (\bigcup_{j=1}^{i} symb(F_j)) \cap (\bigcup_{j=i+1}^{n} symb(F_j))$, i. e., the symbols occurring in the interpolant formula I_i are the symbols that appear both in the prefix up to i and in the remainder of the sequence of input formulas $F_1,\dots,F_n$.

A sequence interpolation problem can be seen as a special form of a tree interpolation problem where the tree degenerates to a linear list. However, we need to add a new vertex to cover the interpolant I_0. We simply label this vertex with $\top$ as shown in Figure 4.4.

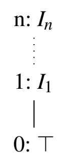


Fig. 4.4 Tree interpolation problem corresponding to a sequence interpolation problem.

It is easy to see that a sequence interpolant exists if and only if a tree interpolant exists.

Lemma 4.2. *Let $F_1,\dots,F_n$ be a sequence interpolation problem and T,F be the corresponding tree interpolation problem. Let I be a tree interpolant for T,F. Then, $I(v_0),\dots,I(v_n)$ is a sequence interpolant.*

Proof. For $0 \le i \le n$, we set $I_i \equiv I(v_i)$. We immediately get $I_n \equiv I(v_n) \equiv \bot$ from the condition of tree interpolation. Furthermore, since $F(v_0) \equiv \top \models I(v_0)$, we get $I_0 \equiv I(v_0) \equiv \top$. Furthermore, since vertex i is the only child of vertex $i+1$, the condition for tree interpolation gives us $I(v_i) \wedge F(v_{i+1}) \models I(v_{i+1})$ which is equivalent to $I_i \wedge F_{i+1} \models I_{i+1}$. The symbol condition is trivial. □

4.1.2 Locality of Symbols and Literals in a Tree

We define for every vertex $v \in V$ the set of symbols $symb(v)$ containing all symbols that can occur in the labelling of v according to the symbol condition in the definition of tree interpolation. Initially we define $symb(v) = symb(F(v))$. For a symbol a occurring in two vertices $v_1,v_2 \in V$, i. e., $a \in symb(F(v_1)) \cap symb(F(v_2))$, we add a to $symb(w)$ for every vertex w on the paths from v_1 resp. v_2 to the least common ancestor $lca(v_1,v_2)$.

Lemma 4.3. *Replacing $symb(F(w))$ with $symb(w)$ does not change the symbol condition of tree interpolants. In particular*

$$\bigcup_{w \in st(v)} symb(F(w)) = \bigcup_{w \in st(v)} symb(w) \qquad \textit{and} \qquad \bigcup_{w' \notin st(v)} symb(F(w')) = \bigcup_{w' \notin st(v)} symb(w').$$

Proof. Assume we added a to $symb(w)$ for some $w \in st(v)$. For w there are $v_1,v_2 \in V$ with $a \in symb(F(v_i))$. At least one of them is a descendent of w, hence there is another vertex $v_i \in st(v)$ with $a \in symb(F(v_i))$. On the other hand, assume we added a to $symb(w')$ for some $w' \notin st(v)$. Then again there are $v_1,v_2 \in V$ with $a \in symb(F(v_i))$ and w' lies on the path of v_1 or v_2 to $lca(v_1,v_2)$. If both v_1 and v_2 would lie in $st(v)$, then v would be a common ancestor of v_1 and v_2 which contradicts $w' \notin st(v)$. Hence, there is a vertex $v_i \notin st(v)$ with $a \in symb(F(v_i))$. □

For a term t we write $t \in symb(v)$ if all symbols of t occur in v, i. e., $symb(t) \subseteq symb(v)$, similarly for a literal ℓ. Furthermore, for a symbol a a term t or a literal ℓ we overload *lca* to denote the least common ancestor of all vertices v such that a resp. t resp. $\ell \in symb(F(v))$. Note that $lca(t)$ is only defined if $t \in symb(v)$ for some vertex v. By the definition of $symb(v)$ we have $a \in symb(lca(a))$ and $a \notin symb(v)$ for all $v \notin st(lca(a))$. Intuitively, $lca(a)$ is the least vertex containing all occurrences of a in its subtree. Having $a \in symb(lca(a))$ is the main reason why we defined $symb(v)$ as above. This also carries over to terms and literals, e. g., $\ell \in symb(lca(\ell))$. Note that $lca(a_1 = a_2)$ can be different from $lca(a_1, a_2)$. The former is the *lca* of all vertices that contain a_1 *and* a_2, while the latter is the *lca* of all vertices that contain a_1 *or* a_2.

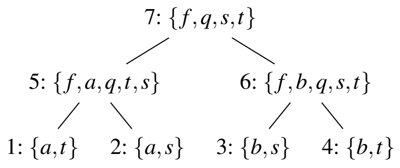


Fig. 4.5 The sets *symb* for the tree interpolation problem shown in Figure 4.2.

Example 4.3. Consider again the tree interpolation problem shown in Figure 4.2. Figure 4.5 shows for each vertex v the set $symb(v)$. We initialise the sets with the sets $symb(L(v))$. For the leaves (vertices 1, 2, 3, and 4), we are already done. Now consider the symbol t. It occurs in vertices 1 and 4. Therefore, $lca(t) = 7$ and we add t to the sets for vertices 5, 6, and 7 since they are on the paths from vertex 1 resp. 4 to vertex 7. Distribution of s is similar. Now consider f and q. They both occur in vertices 5 and 6. Thus, their least common ancestor is 7 and we add them to the set for vertex 7. Then, all symbols are distributed as far as possible and the resulting sets are shown in Figure 4.5. ⌟

4.1.3 Projection Function for Tree Interpolation

Similar to the binary case, we introduce a projection function $\ell \downharpoonleft v$ that projects a literal ℓ onto a vertex v of the tree interpolation problem. If all symbols of ℓ occur in a single partition, the literal is not mixed. In that case, we demand that for every partition v, $\ell \downharpoonleft v = \ell$ or $\ell \downharpoonleft v = \top$. Moreover, the projection must satisfy the symbol condition, i. e., $symb(\ell \downharpoonleft v) \subseteq symb(v)$. We say ℓ projects to v, if $\ell \downharpoonleft v = \ell$. Note that if ℓ projects to v, all symbols of ℓ occur in v but not vice versa. We pose another condition on our projection function. If ℓ projects to v_1 and ℓ projects to v_2, then ℓ also projects to every vertex v on the path from v_i $(i = 1,2)$ to $lca(v_1, v_2)$. This is similar to the saturation of *symb* we introduced above. Pudlák style interpolation can be achieved with the projection $\ell \downharpoonleft v = \ell$, if $symb(\ell) \subseteq symb(v)$, and $\ell \downharpoonleft v = \top$, otherwise. For McMillan's interpolation algorithm set $\ell \downharpoonleft lca(\ell) = \ell$ and $\ell \downharpoonleft v = \top$ for $v \neq lca(\ell)$.

Due to theory combination or extended cuts and branches, new literals (built from existing symbols) are generated to refute a given formula. These literals may also be mixed, which means that no vertex contains all symbols in this literal. If a literal ℓ contains symbols local to the subtree of v and symbols local to the remainder of the tree, we say ℓ *is mixed in* v. We denote the set of vertices in which ℓ is mixed by $\mathsf{mixed}(\ell)$. If this set is non-empty we denote by $\mathsf{mixedparent}(\ell)$ the least common ancestor of the parents of the vertices in $\mathsf{mixed}(\ell)$ (this implies that ℓ is not mixed in $\mathsf{mixedparent}(\ell)$). Then we can exactly characterise the set $\mathsf{mixed}(\ell)$ as follows:

Lemma 4.4. *Let ℓ be a literal that is mixed in some vertices and contains the symbols $a_1, \dots, a_n$. Then*

$$\mathsf{mixed}(\ell) = \left\{ v \in V \mid \begin{array}{l} \exists i.\ 1 \le i \le n.\, lca(a_i) \in st(v) \textit{ and} \\ \mathsf{mixedparent}(\ell) \textit{ is a proper ancestor of } v \end{array} \right\}$$

Proof. We first show "$\subseteq$". Let $v \in \mathsf{mixed}(\ell)$. Since ℓ is mixed in v, there is at least one symbol a_i that occurs only inside the subtree of v. Hence, $lca(a_i) \in st(v)$ for some i. Moreover, the $\mathsf{mixedparent}(\ell)$ is an ancestor of the parent of v, hence it is a proper ancestor of v.

For the other direction take a vertex v from the set on the right-hand side. Then there is an i such that $lca(a_i) \in st(v)$, i. e., a_i occurs only inside the subtree of v. It remains to show that there is another symbol that occurs only outside the subtree of v. There must be a vertex $w \in \mathsf{mixed}(\ell)$ such that v is not a proper ancestor of w (otherwise v would be an ancestor of $\mathsf{mixedparent}(\ell)$).

Case 1: w is an ancestor of v.

There is a symbol a_j that only occurs outside of the subtree of w. Thus, this symbol occurs only outside of the subtree of v, so ℓ is mixed in v.

Case 2: w and v have disjoint subtrees.

There is a symbol a_j that only occurs inside of w. Thus, this symbol occurs only outside of the subtree of v, so ℓ is mixed in v. □

Example 4.4. Consider again the tree interpolation problem shown in Figure 4.2 on page 38. Let $a = b$ be a mixed literal. We have $lca(a) = 5$ and $lca(b) = 6$. The mixed parent of $a = b$ is vertex 7. Thus, $\mathsf{mixed}(a = b) = \{5, 6\}$. Note that $a = b$ is not mixed in the leaves since all symbols occur outside of the respective subtrees, i. e., no leaf is a least common ancestor of one of the symbols. Since the definition of $\mathsf{mixed}(\ell)$ only depends on the symbols occurring in ℓ and the least common ancestors of these symbols, we have

$$\begin{aligned} &\mathsf{mixed}(a = b) = \mathsf{mixed}(a \neq b) \\ =&\mathsf{mixed}(a \le b) = \mathsf{mixed}(\neg a \le b) \\ =&\mathsf{mixed}(b \le a) = \mathsf{mixed}(\neg b \le a). \end{aligned}$$ ⌟

We extend the projection function to cope with mixed literals. For every literal ℓ and every vertex $v_j \in \mathsf{mixed}(\ell)$, an auxiliary variable x_j is introduced. If ℓ is not mixed in v, $\ell \downharpoonleft v$ is either $\top$ or ℓ.

Definition 4.1. Let $\downharpoonleft$ be a projection function. The projection function is *correct*, if and only if for all literals ℓ

$$\ell \iff \exists \mathbf{x}.\ \bigwedge_{v \in V} \ell \downharpoonright v$$

where $\mathbf{x} = \{x_j \mid v_j \in \mathsf{mixed}(\ell)\}$ is the set of all auxiliary variables introduced for the literal ℓ.

To ease notation we extend the projection function to sets of vertices. Hence, for $V' \subseteq V$, we define $\ell \downharpoonright V' := \bigwedge_{v \in V'} \ell \downharpoonright v$. There must be at least one vertex $v \in V$ with $\ell \downharpoonright v = \ell$ unless ℓ is mixed in some vertices.

If ℓ is not mixed in any vertices, we instantiate $\ell \downharpoonright v = \ell$ if and only if $\ell \in symb(v)$ for Pudláks algorithm, or $\ell \downharpoonright v = \ell$ only if v is the least common ancestor of the vertices v' such that $\ell \in symb(v')$. Different projection functions are possible [37] yielding interpolants of different strength.

4.1.4 Simultaneous Binary Interpolation

To extract tree interpolants from one proof tree we apply (rule-res) per vertex of the interpolation tree. If a literal occurs in the labelling of more than one vertex, we uniquely assign this literal to one vertex to solve the problem shown in Example 4.1. Thus, for a literal ℓ occurring in the input, there is exactly one vertex v_0 such that $\ell \downharpoonright v_0 = \ell$. For all other vertices $v \neq v_0$ we have $\ell \downharpoonright v = \top$. To extract tree interpolants we use the following modified resolution rule.

$$\frac{C_1 \vee \ell : I_1 \qquad C_2 \vee \neg\ell : I_2}{C_1 \vee C_2 : I_3} \qquad \text{(rule-tres)}$$

$$\text{where } I_3(v) = \begin{cases} I_1(v) \vee I_2(v) & \text{if } \ell \downharpoonright v' = \top \text{ for all } v' \notin st(v) \\ I_1(v) \wedge I_2(v) & \text{if } \ell \downharpoonright v' = \top \text{ for all } v' \in st(v) \\ \mathsf{mixcomb}(\ell, I_1(v), I_2(v)) & \text{if } \ell \text{ is mixed} \\ (I_1(v) \vee \ell) \wedge (I_2(v) \vee \neg\ell) & \text{otherwise} \end{cases}$$

Similarly to the correctness proof of binary interpolation we use *partial tree interpolants*. Partial tree interpolants are computed recursively over the proof tree starting from the leaves of the tree to the root containing the empty clause.

Definition 4.2 (Partial Tree Interpolant). A *partial tree interpolant* for a clause C is a tree interpolant for T and F' where $F'(v) = F(v) \wedge (\neg C \downharpoonright v)$ for $v \in V$.

Since the case of mixed literals is different for every theory we delay proving the mixed case until we instantiate mixcomb. To prove the remaining cases, we first note that if $\ell \downharpoonright st(v) = \top$ for some $v \in V$, then $\ell \downharpoonright st(v_c) = \top$ for every child v_c of v. Furthermore, if $\ell \downharpoonright st(v_c) = \ell$ for some child v_c of v, then $\ell \downharpoonright st(v) = \ell$, too. We use these observations to prove the following lemma.

Lemma 4.5. *Assume no literal is mixed. Then, rule (rule-tres) produces partial tree interpolants, i. e., if I_1 and I_2 are partial tree interpolants for $C_1 \vee \ell$ resp. $C_2 \vee \neg\ell$ then I_3 is a partial tree interpolant for $C_1 \vee C_2$.*

Proof. The symbol condition of tree interpolation trivially holds assuming it holds for I_1 and I_2. We fix a vertex v_p and show $\bigwedge_{(v_c,v_p)\in E} I_3(v_c) \wedge F(v_p) \wedge (\neg C_1 \wedge \neg C_2) \downharpoonright v_p \models I_3(v_p)$ under the following assumptions:

$$\bigwedge_{(v_c,v_p)\in E} I_1(v_c) \wedge F(v_p) \wedge (\neg C_1 \wedge \neg \ell) \downharpoonright v_p \models I_1(v_p) \tag{4.1}$$

$$\bigwedge_{(v_c,v_p)\in E} I_2(v_c) \wedge F(v_p) \wedge (\neg C_2 \wedge \ell) \downharpoonright v_p \models I_2(v_p) \tag{4.2}$$

We assume $\bigwedge_{(v_c,v_p)\in E} I_3(v_c) \wedge F(v_p) \wedge (\neg C_1 \wedge \neg C_2) \downharpoonright v_p$ and show $I_3(v_p)$ and split cases on the remaining cases of rule (rule-tres).

Case 1.

There is an edge $(v_c, v_p) \in E$ with $\ell \downharpoonright v = \top$ for all $v \notin st(v_c)$. Then $I_3(v_c) = I_1(v_c) \vee I_2(v_c)$ and $I_3(v_p) = I_1(v_p) \vee I_2(v_p)$. Also for all other edges $(v_c', v_p) \in E$, $\ell \downharpoonright v = \top$ for all $v \in st(v_c')$, hence $I_3(v_c') = I_1(v_c') \wedge I_2(v_c')$.

Consider the assumption $\bigwedge_{(v_c,v_p)\in E} I_3(v_c)$. For v_c, we get $I_1(v_c) \vee I_2(v_c)$ and for all other $v_{c'}$ we get $I_1(v_{c'}) \wedge I_2(v_{c'})$. Since $\ell \downharpoonright v = \top$ for all $v \notin st(v_c)$, we have $\ell \downharpoonright v_p = \neg\ell \downharpoonright v_p = \top$. If $I_1(v_c)$ holds, we get $I_1(v_p)$ from (4.1). If $I_2(v_c)$ holds, we get $I_2(v_p)$ from (4.2). In both cases, we get $I_1(v_p) \vee I_2(v_p)$.

Case 2.

Assume $\ell \downharpoonright v = \top$ for all $v \in st(v_p)$. Then $I_3(v_p) = I_1(v_p) \wedge I_2(v_p)$ and $I_3(v_c) = I_1(v_c) \wedge I_2(v_c)$ for all $(v_c, v_p) \in E$. From $\bigwedge_{(v_c,v_p)\in E} I_3(v_c)$ we conclude that $\bigwedge_{(v_c,v_p)\in E} I_1(v_c)$ and $\bigwedge_{(v_c,v_p)\in E} I_2(v_c)$ hold. Using the induction hypothesis (again $\ell \downharpoonright v_p = \top$) we derive $I_1(v_p)$ and $I_2(v_p)$ thus $I_3(v_p)$.

Case 3.

Otherwise $I_3(v_c) \implies (I_1(v_c) \vee \ell) \wedge (I_2(v_c) \vee \neg\ell)$ for all $(v_c, v_p) \in E$ since we are not in Case 1. With the induction hypothesis and $\ell \implies \ell \downharpoonright v_p$ we derive from $\bigwedge_{(v_c,v_p)\in E} I_3(v_c)$ that $(I_1(v_p) \vee \ell) \wedge (I_2(v_p) \vee \neg\ell)$ holds. Since we are not in Case 2, this implies $I_3(v_p)$. □

Rule (rule-tres) allows us to compute partial tree interpolants for resolution steps. To compute partial tree interpolants for the leaves of the resolution tree, we distinguish between input clauses, theory lemmas, and theory combination lemmas. We first consider computation of partial tree interpolants for input clauses. Every input clause C comes from (at least) one vertex of the interpolation tree, which we call v_C. Since this vertex contains all symbols of all literals in the clause, the literals cannot be mixed. For a conjunction $\bigwedge \ell_i$ where every ℓ_i is not mixed in any vertex and a set of vertices V we define $\bigwedge \ell_i \setminus V$ as the conjunction of all literals ℓ such that $\ell \downharpoonright V = \top$. Note that for an arbitrary conjunction F, $F \setminus V = \top$ if V contains all vertices in the interpolation tree. With these definitions we can compute partial tree interpolants for input clauses.

Lemma 4.6. *Let C be an input clause of a vertex $v_C \in V$. Then I with*

$$I(v) = \begin{cases} \neg(\neg C \setminus st(v)) & \textit{if } v_C \in st(v) \\ \neg C \setminus \overline{st(v)} & \textit{otherwise} \end{cases}$$

is a partial tree interpolant for C.

Proof. The symbol condition is satisfied in both cases since we always remove the literals that are local to the (complement of the) subtree. It is also clear that the root vertex is mapped to $\neg\top = \bot$. Remains to show inductivity, i. e., $\bigwedge_{(v_c,v_p)\in E} I(v_c) \wedge \neg C \downharpoonleft v_p \wedge F(v_p) \models I(v_p)$.

We first observe that if v_c is a child of v_p that does not contain the clause vertex v_C in its subtree and if some $\ell \in C$ projects to some vertex in $st(v_c)$ then the assumption of inductivity contains $\neg\ell$. To see this, assume $I(v_c) \equiv \neg C \setminus \overline{st(v_c)}$ does not contain $\neg\ell$. This means that ℓ projects to some vertex v_1 in $st(v_c)$ and to some vertex v_2 in $\overline{st(v_c)}$. Then $lca(v_1,v_2)$ must be an ancestor of v_p and v_p occurs on the path from v_1 to $lca(v_1,v_2)$. By the conditions of the projection function, ℓ also projects to v_p. Hence $\neg C \downharpoonleft v_p$ contains $\neg\ell$.

We now show inductivity, by case distinction.

Case $v_C = v_p$.

Since $C \in F(v_p)$, one of the literals $\ell \in C$ is true. If ℓ projects to some vertex in $st(v_c)$ for some child v_c of v_p, then $\neg\ell$ occurs in the assumption. Likewise, $\neg\ell$ occurs in $\neg C \downharpoonleft v_p$ if ℓ projects to v_p. Thus, since ℓ is true, ℓ does not project to any v in $st(v_p)$. Hence, $I(v_p) \equiv \neg(\neg C \setminus st(v_p))$ is true.

Case there exists a child v_{c^*} such that $v_C \in st(v_{c^*})$.

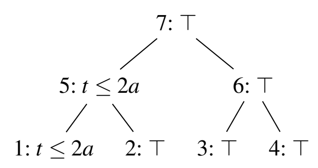
If C occurs in the child v_{c^*} of v_p, then $I(v_{c^*}) = \neg(\neg C \setminus st(v_{c^*}))$. If this assumption holds, there is a true literal ℓ in C that does not project to any vertex in $st(v_{c^*})$. Similar reasoning as in the previous case shows that it does not project to any vertex in $st(v_c)$ for the other children v_c of v_p and it does not project to v_p. Hence, $I(v_p) \equiv \neg(\neg C \setminus st(v_p))$ is true.

Case v_C does not occur inside the subtree.

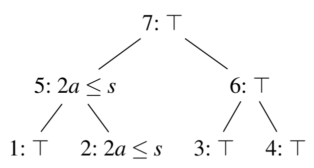
Then v_C does not occur in $st(v_c)$ for all children. By the observation all literals $\ell \in C$ that project to some vertex in $st(v_c)$ occur negated in the assumption. This also holds if ℓ projects to v_p. Hence, all $\ell \in C$ that project to some vertex in $st(v_p)$ are false. Thus, $I(v_p) \equiv \neg C \setminus \overline{st(v_p)}$ is true. □

Example 4.5. Consider again the tree interpolation problem shown in Figure 4.2. We have six input clauses: $t \leq 2a$, $2a \leq s$, $s \leq 2b$, $2b \leq t+1$, $f(a) = q$, and $f(b) \neq q$. The projection of the individual input clauses onto the tree are shown in Figure 4.6. We use the projection function that simulates Pudláks algorithm. Hence, whenever $symb(\ell) \subseteq symb(v)$, we add ℓ to v. Note that the definition of $symb$ implies that $(\neg\ell) \downharpoonleft v = \neg(\ell \downharpoonleft v)$ when using this projection function.

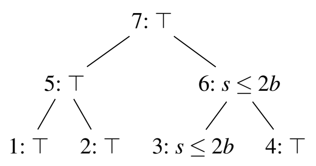
The interested reader might check the projections against the symbol sets shown in Figure 4.5 on page 41. Since all clauses are unit clauses, the projection of the clauses onto the tree is equivalent to the projection of the literal onto the tree. It is easy to verify that the projections are correct, i. e., that for every literal ℓ, the conjunction of the projection of ℓ onto the tree is equivalent to ℓ.



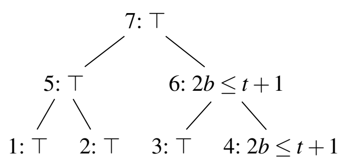
(a) Projection of the input clause $t \leq 2a$.



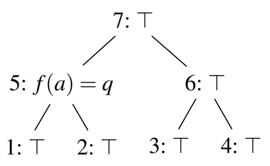
(b) Projection of the input clause $2a \leq s$.



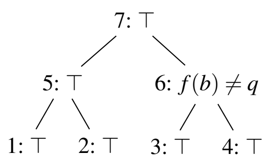
(c) Projection of the input clause $s \leq 2b$.



(d) Projection of the input clause $2b \leq t+1$.



(e) Projection of the input clause $f(a) = q$.

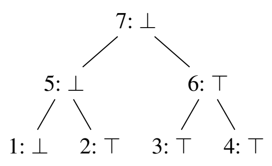


(f) Projection of the input clause $f(b) \neq q$.

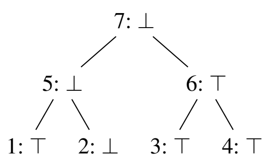
Fig. 4.6 Projections of the input clauses of the tree interpolation problem shown in Figure 4.2 using the projection function for Pudláks algorithm.

We derive a partial tree interpolant for every input clause according to Lemma 4.6. The resulting partial tree interpolants are shown in Figure 4.7. Consider the input clause $t \leq 2a$ that labels vertex 1, i. e., v_c is vertex 1. In vertex 1, the first case of Lemma 4.6 applies since $v_c \in st(v_c)$. Thus, we have to compute the conjunction of the projection of $\neg(t \leq 2a)$ onto the interpolation tree without v_c. We get $\neg(t \leq 2a)$ since the projection onto vertex 5 is $\neg(t \leq 2a)$ and the projection onto all other vertices is $\top$. Thus, $I(1) = t \leq 2a$. Now consider vertex 5. Again, we are in the first case since vertex 1 is a child of vertex 5 and, therefore, in the subtree. When we remove the subtree rooted at vertex 5 and project $\neg(t \leq 2a)$ onto the resulting tree, we get $\top$. Thus, $I(5) = \top$. The labelling for the root $I(7) = \bot$ is trivial. For all remaining vertices, v_c occurs outside of the subtree and the second case of Lemma 4.6 applies. But for all these nodes v, we have $v_c \in \overline{st(v)}$ and therefore $(\neg(t \leq 2a)) \downharpoonright \overline{st(v)} = \neg t \leq 2a$. Thus, $I(v) = \bot$ in all these cases. The labels for the other input clauses are computed in a similar way. ⌟

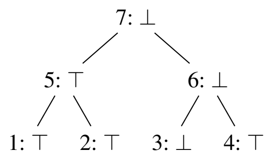
In the remainder of this chapter we show how to construct partial tree interpolants for different theories and how to combine them.



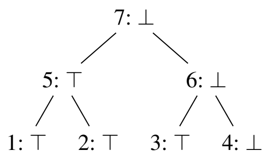
(a) Partial tree interpolant for the input clause $t \leq 2a$.



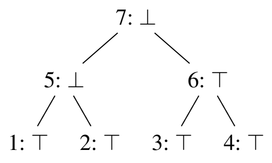
(b) Partial tree interpolant for the input clause $2a \leq s$.



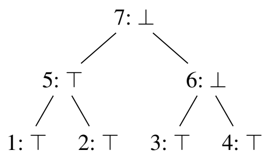
(c) Partial tree interpolant for the input clause $s \leq 2b$.



(d) Partial tree interpolant for the input clause $2b \leq t+1$.



(e) Partial tree interpolant for the input clause $f(a) = q$.



(f) Partial tree interpolant for the input clause $f(b) \neq q$.

Fig. 4.7 Partial tree interpolants for the input clauses of the tree interpolation problem shown in Figure 4.2.

4.2 Theory Specific Interpolation

Proof trees generated by SMT solvers contain some theory lemmas. These theory lemmas are clauses that are valid according to the theory that created these lemmas. Thus, if C is a theory lemma, then $\neg C$ is a *theory conflict* since it is unsatisfiable according to the theory.

To compute a tree interpolant from a refutation containing theory lemmas, theory specific methods to compute partial tree interpolants for theory lemmas are needed. Given a theory lemma C, we distribute the literals in $\neg C$ onto the interpolation tree according to the projection function. Unfortunately, unlike input clauses, theory lemmas might contain mixed literals. Thus, besides interpolation for theory lemmas, a theory specific interpolation method must also provide an instantiation of mixcomb to deal with resolution steps on the mixed literals of this theory.

We now show how to compute partial tree interpolants for the theories of uninterpreted functions and the theory of linear arithmetic. Furthermore, we show how to combine the partial tree interpolants if the pivot of a resolution step is a mixed literal. We show how to combine the theories and conclude this chapter with the computation of a tree interpolant for the tree interpolation problem shown in Example 4.2.

4.2.1 Theory of Uninterpreted Functions

The Theory of Uninterpreted Functions can be solved by saturating equalities over the existing sub-terms in the formula using the usual rules of reflexivity, symmetry, transitivity, and congruence, see Figure 4.8. Thus, the theory lemmas are instances of these rules, where the terms a, b, c, $f(a_1,\ldots,a_n)$, and $f(b_1,\ldots,b_n)$ occur in the input formula. The latter is important, as it guarantees that there is always a vertex of the tree interpolation problem that contains the term and thus all its symbols. Thus the terms occurring on one side of an equality are never mixed. However, the equality literal can be mixed in the sense that there is no vertex that contains all symbols on the left- and on the right-hand-side of the equality.

$$a = a \quad (reflexivity) \qquad a \neq b \vee b = a \quad (symmetry) \qquad a \neq b \vee b \neq c \vee a = c \quad (transitivity)$$
$$a_1 \neq b_1 \vee \cdots \vee a_n \neq b_n \vee f(a_1,\ldots,a_n) = f(b_1,\ldots,b_n) \quad (congruence)$$

Fig. 4.8 The lemmas that are generated by the Theory of Uninterpreted Functions. Each term occurring on one side of an equality is guaranteed to occur in the input formula, i. e. in one partition of the input problem.

Example 4.6. To prove unsatisfiability of the labelling of the interpolation tree depicted in Figure 4.2, we use the theory lemma $a \neq b \vee f(a) \neq q \vee f(b) = q$. This lemma is derived using one application of symmetry, one application of transitivity, one application of congruence, and two resolution steps.

We first apply congruence to a and b to get $f(a) = f(b)$ and then use symmetry to swap the sides of the equality. A resolution step is used to combine the applications of congruence and symmetry.

$$\text{RES}\ \frac{\text{CONG}\ \dfrac{}{a \neq b \vee f(a) = f(b)} \qquad \text{SYM}\ \dfrac{}{f(a) \neq f(b) \vee f(b) = f(a)}}{a \neq b \vee f(b) = f(a)}$$

Now, we use transitivity to derive the lemma.

$$\text{RES}\ \frac{\dfrac{\vdots}{a \neq b \vee f(b) = f(a)} \qquad \text{TRANS}\ \dfrac{}{f(b) \neq f(a) \vee f(a) \neq q \vee f(b) = q}}{a \neq b \vee f(a) \neq q \vee f(b) = q}$$

⌟

4.2.1.1 Projection of Mixed Literals

We start by giving the projection function for an equality literal $\ell :\equiv a_1 = a_2$. By Lemma 4.4, every vertex $v_p \in \mathsf{mixed}(\ell)$ lies on a path between $lca(a_i)$ and $\mathsf{mixedparent}(\ell)$ (for some $i \in \{1,2\}$). The a_i is unique, since v_p is not mixed if $lca(a_i) \in st(v_p)$ for both $i = 1,2$. For each vertex $v_p \in \mathsf{mixed}(\ell)$ we introduce an auxiliary variable $x^p_{a_1=a_2}$ that captures the value of this a_i. The projection of ℓ achieves this by fixing the value $x^p_{a_1=a_2}$ of a mixed vertex v_p to the value $x^c_{a_1=a_2}$ of the (uniquely defined) child v_c that lies on the path to the unique $lca(a_i)$, or to the value of a_i if $v_p = lca(a_i)$. The projection of the vertex $\mathsf{mixedparent}(\ell)$ ensures that $a_1 = a_2$ by making the auxiliary variables of the corresponding children equal.

$$a_1 = a_2 \downharpoonright v_p = \begin{cases} x^{c_1}_{a_1=a_2} = x^{c_2}_{a_1=a_2} & \text{if } (v_{c_1}, v_p), (v_{c_2}, v_p) \in E \text{ and } v_{c_1}, v_{c_2} \in \mathsf{mixed}(\ell) \\ a_i = x^c_{a_1=a_2} & \text{if } (v_c, v_p) \in E,\ v_c \in \mathsf{mixed}(\ell), \text{ and } a_i \in symb(v_p) \\ x^c_{a_1=a_2} = x^p_{a_1=a_2} & \text{if } (v_c, v_p) \in E,\ v_c, v_p \in \mathsf{mixed}(\ell) \\ a_i = x^p_{a_1=a_2} & \text{if } lca(a_i) = v_p,\ v_p \in \mathsf{mixed}(\ell) \text{ for some } i \in \{1,2\} \\ \top & \text{otherwise} \end{cases}$$

To show correctness of the projection in the sense of Definition 4.1, we show by induction that for a subtree $st(v_p)$ the projection $\exists \mathbf{x}.\ (a_1 = a_2) \downharpoonright st(v_p)$ where $\mathbf{x}$ ranges over all auxiliary variables except $x^p_{a_1=a_2}$ is equivalent to (1) $a_i = x^p_{a_1=a_2}$ if $a_1 = a_2$ is mixed in v_p and $lca(a_i) \in st(v_p)$ and (2) $a_1 = a_2$ if both a_1 and a_2 occur in $st(v_p)$. The induction step can be seen by distinguishing the cases in the definition of $(a_1 = a_2) \downharpoonright v_p$ as follows.

In the first two cases, we observe that a_1, a_2 occur inside the subtree of v_p. Hence, v_p is not mixed but has at least one mixed child. By Lemma 4.4, $v_p = \mathsf{mixedparent}(\ell)$. Usually, this means that there are exactly two child vertices v_{c_1} and v_{c_2} in which ℓ is mixed, one an ancestor of $lca(a_1)$ and one an ancestor of $lca(a_2)$ (first case). However, it is also possible that $a_i \in symb(v_p)$ for one of the two symbols a_1, a_2 (second case). In both cases, the corresponding projection ensures that $a_1 = a_2$, provided that the projection on the subtree of v_{c_i} ensures $a_i = x^{c_i}_{a_1=a_2}$.

When ℓ is mixed in v_p, the third or the fourth case applies. Then, for exactly one $i \in \{1,2\}$, $lca(a_i)$ occurs in the subtree of v_p. If already $v_p = lca(a_i)$, we are in the forth case and the projection is $x^p_{a_1=a_2} = a_i$. Otherwise, the third case applies and v_c is the child containing $lca(a_i)$. By the induction hypothesis, the projection to the subtree already ensures $x^c_{a_1=a_2} = a_i$, hence $x^p_{a_1=a_2} = a_i$ is ensured for the value a_i that occurs in the subtree of v_p. The last case only applies if ℓ is not mixed in v_p and $v_p \neq \mathsf{mixedparent}(\ell)$. If a_1 and a_2 occur in $st(v_p)$ then v_p must be an ancestor of $\mathsf{mixedparent}$ and the projection of the subtree that contains $\mathsf{mixedparent}$ is equivalent to $a_1 = a_2$.

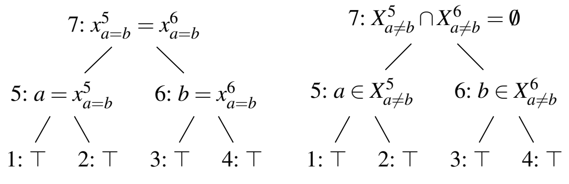
The projection of a disequality $a_1 \neq a_2$ is tricky. Instead of a plain auxiliary variable $x^p_{a_1 \neq a_2}$ we introduce a set-valued auxiliary variable $X^p_{a_1 \neq a_2}$ for every vertex v_p where the literal is mixed. For such a vertex v_p one a_i $(i = 1,2)$ occurs only in the subtree of v_p and the other only outside the subtree. The projections of the literal enforce that $X^p_{a_1 \neq a_2}$ contains the value a_i that occurs in the subtree of vertex v_i and does not contain the other value. It may contain other values different from a_1 and a_2 when the exact value of a_i cannot be expressed using only symbols shared between the subtree of v_i and its complement. The projections of $a_1 \neq a_2$ are defined as follows.

$$a_1 \neq a_2 \downharpoonright v_p = \begin{cases} X^{c_1}_{a_1 \neq a_2} \cap X^{c_2}_{a_1 \neq a_2} = \emptyset & \text{if } (v_{c_1}, v_p), (v_{c_2}, v_p) \in E \text{ and } v_{c_1}, v_{c_2} \in \mathsf{mixed}(\ell) \\ a_i \notin X^c_{a_1 \neq a_2} & \text{if } (v_c, v_p) \in E,\ v_c \in \mathsf{mixed}(\ell), \text{ and } a_i \in symb(v_p) \\ X^c_{a_1 \neq a_2} \subseteq X^p_{a_1 \neq a_2} & \text{if } (v_c, v_p) \in E,\ v_c, v_p \in \mathsf{mixed}(\ell) \\ a_i \in X^p_{a_1 \neq a_2} & \text{if } lca(a_i) = v_p,\ v_p \in \mathsf{mixed}(\ell) \text{ for some } i \in \{1,2\} \\ \top & \text{otherwise} \end{cases}$$

Although the projections are different, the cases are exactly the same as for equality. The fourth and third formulae ensure that $X^p_{a_1 \neq a_2}$ contains the value a_i that occurs in the subtree of v_p. With this property, each of the first two formulae ensures that $a_1 \neq a_2$.

Despite the definition of the projection function, we do not need set-theoretic reasoning in our solver. The projections are only used to prove the correctness of the resolution rule and the theory specific interpolation rules.

Example 4.7. In our running example, the literals $a = b$ and $a \neq b$ occur in the refutation of the labelling of the vertices. As seen in Example 4.4, $\mathsf{mixed}(a = b) = \mathsf{mixed}(a \neq b) = \{5,6\}$. Furthermore, the mixed parent of these literals is vertex 7, $lca(a)$ is vertex 5, and $lca(b)$ is vertex 6.



(a) Projection of $a = b$. (b) Projection of $a \neq b$.

Fig. 4.9 Projections of mixed equality literals onto the interpolation tree from Figure 4.2 on page 38.

The projections of $a = b$ resp. $a \neq b$ are shown in Figure 4.9. It is easy to see that the literal $a = b$ is preserved if we conjoin the labelling of the vertices in Figure 4.9(a) and existentially quantify over $x^5_{a=b}$ and $x^6_{a=b}$ which are integer-valued auxiliary variables (since a and b are integer-valued variables). The projection of $a \neq b$ introduces the variables $X^5_{a\neq b}$ and $X^6_{a\neq b}$ which represent sets of integers. If we conjoin the labelling of the vertices and existentially quantify over $X^5_{a\neq b}$ and $X^6_{a\neq b}$, we get $\exists X^5_{a\neq b}, X^6_{a\neq b}.\ a \in X^5_{a\neq b} \wedge b \in X^6_{a\neq b} \wedge X^5_{a\neq b} \cap X^6_{a\neq b} = \emptyset$ which states that a is different from b. ⌟

To facilitate the mixed resolution rule, we pose a syntactic restriction on the partial invariants containing $X^p_{a\neq b}$, namely that the only occurrences are in a positively occurring literals $s \in X^p_{a\neq b}$, where s is an arbitrary term (not containing a set-valued variable). In particular, $s \in X^p_{a\neq b}$ may occur only positively. To get a similar notation as in the previous chapter, we define $EQ(X,s) :\equiv s \in X$ and write the interpolant as $I[EQ(X,s_1)] \ldots [EQ(X,s_n)]$. On the other hand, a variable $x^p_{a=b}$ introduced by a mixed equality may occur anywhere in the partial interpolant, even under a function application or in the s-part of an $EQ(X,s)$ term.

4.2.1.2 Mixed Combination for Resolution

The theory lemmas are combined using only the resolution rule, thus the interpolation procedure presented in Section 4.1.4 can be used. We have to instantiate the function $\mathsf{mixcomb}$ for mixed equality literals. For a mixed equality literal $a = b$, the interpolant $I_1(v)$ is the interpolant of a conflict $\neg C_1 \wedge a \neq b$, thus is of the form $I_1[EQ(X^v_{a\neq b}, s_1)] \ldots [EQ(X^v_{a\neq b}, s_n)]$. The interpolant $I_2(v)$ is of the form $I_2(x^v_{a=b})$ where $x^v_{a=b}$ is the auxiliary variable of $a = b$ for the vertex v. We define

$$\mathsf{mixcomb}(a = b, I_1[EQ(X,s_1)]\dots[EQ(X,s_n)], I_2(x)) := I_1[I_2(s_1)]\dots[I_2(s_n)]$$

Lemma 4.7. *The rule (rule-tres) produces a partial tree interpolants in the case where ℓ is an equality $a = b$ that is mixed in some of the vertices: Provided that I_1 and I_2 are partial tree interpolants for $C_1 \vee \ell$ resp. $C_2 \vee \neg\ell$ then I_3 is a partial tree interpolant for $C_1 \vee C_2$.*

Proof. The symbol condition carries over (note that the auxiliary variables are eliminated by mixcomb) and since we always have $I_3 = I_1 \vee I_2$ for the root vertex its interpolant is still $\perp$.

We do a case split over the five cases of the projection function for $a = b$ and $a \neq b$ (two children are mixed, one child is mixed, the parent and one child is mixed, only the parent is mixed, and neither parent nor any child is mixed). The last case does not involve mixcomb, thus the proof given in Section 4.1.4 applies. We now assume that we are in the first four cases.

In the first two cases the parent is $\mathsf{mixedparent}(\ell)$ and $I_3(v_p) = I_1(v_p) \vee I_2(v_p)$ since all symbols occur in $st(v_p)$. In the third and forth case the parent is mixed and $I_3(v_p) = \mathsf{mixcomb}(\ell, I_1(v_p), I_2(v_p))$. For every child v_c, where ℓ is mixed, the interpolants $I_1(v_c)$ and $I_2(v_c)$ are of the form $I^c_1[EQ(X^c_{a\neq b}, s_1)]\dots[EQ(X^c_{a\neq b}, s_n)]$ and $I^c_2(x^c_{a=b})$. For the other children, all symbols in ℓ occur outside of the subtree; thus, $I_3(v_c) = I_1(v_c) \wedge I_2(v_c)$.

So the inductivity rule for I_3 becomes:

$$\bigwedge_{(v_c,v_p)\in E, v_c\in\mathsf{mixed}(\ell)} I^c_1[I^c_2(s_1)]\dots[I^c_2(s_n)] \wedge \bigwedge_{(v_c,v_p)\in E, v_c\notin\mathsf{mixed}(\ell)} (I_1(v_c) \wedge I_2(v_c)) \wedge (\neg C_1) \downharpoonright v_p \wedge (\neg C_2) \downharpoonright v_p \models I_3(v_p)$$

If we assume $(\neg C_2) \downharpoonright v_p$ and $I_2(v_c)$ for all children that are not mixed, we can simplify the inductivity rule for I_2 (which holds by induction) to

$$\bigwedge_{(v_c,v_p)\in E, v_c\in\mathsf{mixed}(\ell)} I^c_2(x^c_{a=b}) \wedge (a = b) \downharpoonright v_p \models I_2(v_p)$$

Similarly, the inductivity rule for I_1 simplifies to

$$\bigwedge_{(v_c,v_p)\in E, v_c\in\mathsf{mixed}(\ell)} I^c_1[EQ(X^c_{a\neq b}, s_1)]\dots[EQ(X^c_{a\neq b}, s_n)] \wedge (a \neq b) \downharpoonright v_p \models I_1(v_p)$$

The main trick is to instantiate in the last formula the variable $X^c_{a\neq b}$ with the set $\{x | I^c_2(x)\}$. Then $EQ(X^c_{a\neq b}, s_i)$ is $s_i \in \{x | I^c_2(x)\}$ and simplifies to $I^c_2(s_i)$. Likewise, we instantiate the variable $X^p_{a\neq b}$ with $\{x | I^p_2(x)\}$. The induction hypothesis for I_1 contains $X^c_{a\neq b}$ and $X^p_{a\neq b}$ also in $(a \neq b) \downharpoonright v_p$, so we should investigate what the substitution does to this part of the formula.

In the first case $(a \neq b) \downharpoonright v_p$ is $X^{c1}_{a\neq b} \cap X^{c2}_{a\neq b} \neq \emptyset$, i. e., $I^{c1}_2(x) \wedge I^{c2}_2(x) \models \perp$. This is implied by the induction hypothesis for I_2, if we assume that $I_2(v_p)$ does not hold (otherwise inductivity would hold as $I_3(v_p) = I_1(v_p) \vee I_2(v_p)$). Similarly in the second case $(a \neq b) \downharpoonright v_p$ is $b \notin X^c_{a\neq b}$, i. e., $I^c_2(b) \models \perp$, which is again implied by the induction hypothesis for I_2. Also in the third and fourth case $(a \neq b) \downharpoonright v_p$ is implied by the induction hypothesis for I_2.

To show inductivity, we can assume $I_1^c[I_2^c(s_1)]\dots[I_2^c(s_n)]$ for the mixed children. Together with $(a \neq b) \downharpoonleft v_p$ these are the assumption in the instantiated induction hypothesis for I_1. Thus we can conclude that the instantiated $I_1(v_p)$ holds. In the first two cases this implies $I_3(v_p) = I_1(v_p) \vee I_2(v_p)$. In the third and forth cases this is exactly $I_3(v_p) = I_1^p[I_2^p(s_1)]\dots[I_2^p(s_n)]$ after instantiating. Thus we have shown $I_3(v_p)$ and inductivity holds. □

4.2.1.3 Leaf Interpolation

It only remains to show that there is a partial tree interpolant for every theory lemma C. If all symbols occur outside the subtree, we interpolate it like an input clause that occurs outside the subtree. Otherwise, if all symbols occur in the subtree, we interpolate it like an input clause that occurs in some node in $st(v)$. Thus all interpolants have the form

$$I(v) = \begin{cases} \neg C \setminus \overline{st(v)} & \text{if } symb(C) \subseteq symb(\overline{st(v)}) \\ \neg(\neg C \setminus st(v)) & \text{if } symb(C) \subseteq symb(st(v)) \\ \dots & \dots \end{cases}$$

The inductivity follows in the same way as in Lemma 4.6 if both parent and child interpolants fall into the first two cases. Only if the clause is mixed in the parent or a child vertex we still have to show inductivity. Note that if a theory lemma is mixed, at least one of its literals must be mixed, since for every pair of symbols in a clause there is a literal that contains both symbols.

Interpolation of Reflexivity

The reflexivity axiom is never mixed and a partial interpolant is I with

$$I(v) = \begin{cases} \neg C \setminus \overline{st(v)} & \text{if } a \notin symb(st(v)), \\ \neg(\neg C \setminus st(v)) & \text{otherwise.} \end{cases}$$

Interpolation of Symmetry

For symmetry we have to find a partial interpolant of $a = b \wedge b \neq a$. Here we need to consider the case where the literals are mixed, e. g., $lca(a) \in st(v)$ and b occurs only in $\overline{st(v)}$. The partial interpolant is

$$I(v) = \begin{cases} \neg C \setminus \overline{st(v)} & \text{if } a,b \in symb(\overline{st(v)}), \\ \neg(\neg C \setminus st(v)) & \text{otherwise if } a,b \in symb(st(v)) \\ x_{a=b} \in X_{b \neq a} & \text{if } a = b \text{ is mixed} \end{cases}$$

We show inductivity: $\bigwedge I(v_c) \wedge (\neg C) \downharpoonleft v_p \to I(v_p)$.

Case $a = b$ is mixed in v_p.

W. l. o. g. $lca(a) \in st(v)$ and $b \notin st(v)$. If $lca(a) \in st(v_c)$ for some child v_c, then $I(v_c) \wedge (\neg C) \downharpoonright v_p$ is $x^c_{a=b} \in X^c_{b \neq a} \wedge x^c_{a=b} = x^p_{a=b} \wedge X^c_{b \neq a} \subseteq X^p_{b \neq a}$, which implies $I(v_p) = x^p_{a=b} \in X^p_{b \neq a}$. Otherwise $lca(a) = v_p$ and $(\neg C) \downharpoonright v_p$ is $a = x^p_{a=b} \wedge a \in X^p_{b \neq a}$, which implies $I(v_p)$.

Case $a, b \in symb(st(v_p))$ and $a = b$ is mixed in some child.

Then the mixed child contains either $lca(a)$ or $lca(b)$, w. l. o. g., $lca(a)$ (the other case is symmetric). We call the mixed child v_{ca}.

If $lca(b)$ is in the subtree of another child v_{cb}, that child is also mixed because it contains b only in the subtree and a only outside the subtree. We have $I(v_{ca}) \wedge I(v_{cb}) \wedge \neg C \downharpoonright v_b \leftrightarrow x^{ca}_{a=b} \in X^{ca}_{b \neq a} \wedge x^{cb}_{a=b} \in X^{cb}_{b \neq a} \wedge x^{ca}_{a=b} = x^{cb}_{a=b} \wedge X^{ca}_{b \neq a} \cap X^{cb}_{b \neq a} = \emptyset$. This leads to a contradiction. Hence, inductivity holds.

Otherwise, $b \in symb(v_p)$ and $I(v_{ca}) \wedge \neg C \downharpoonright v_p \leftrightarrow x^c_{a=b} \in X^c_{b \neq a} \wedge x^c_{a=b} = b \wedge b \notin X^c_{b \neq a}$, which is a contradiction. Thus, inductivity holds.

Interpolation of Transitivity

For transitivity, we first consider the case where $a \neq c$ is mixed. W. l. o. g., a occurs only in $st(v)$ and c only in $\overline{st(v)}$. We have to find a shared term that is equal to a and c. If b is shared it can be used. Otherwise either $a = b$ or $b = c$ is mixed and the corresponding auxiliary variable can be used. We define

$$aux1(a,b,c)(v) = \begin{cases} x^v_{a=b} & \text{if } a = b \text{ is mixed in } v, \\ x^v_{b=c} & \text{if } b = c \text{ is mixed in } v, \\ b & \text{otherwise.} \end{cases}$$

If $a = c$ is not mixed but the clause is mixed, then $a = b$, $b = c$ or both are mixed. If only one of them, say $b = c$, is mixed then a is shared. We define

$$aux2(a,b,s)(v) = \begin{cases} x^v_{a=b} & \text{if } a = b \text{ is mixed in } v, \\ s & \text{otherwise.} \end{cases}$$

Finally, the partial interpolant for transitivity is

$$I(v) = \begin{cases} \neg C \setminus \overline{st(v)} & \text{if } a,b,c \in symb(\overline{st(v)}), \\ \neg(\neg C \setminus st(v)) & \text{otherwise if } a,b,c \in symb(st(v)), \\ aux1(a,b,c)(v) \in X^v_{a \neq c} & \text{if } a \neq c \text{ is mixed,} \\ aux2(a,b,a)(v) = aux2(b,c,c)(v) & \text{if } a,c \in symb(\overline{st(v)}), \\ aux2(a,b,a)(v) \neq aux2(b,c,c)(v) & \text{if } a,c \in symb(st(v)). \end{cases}$$

Again we only need to consider the cases where the clause is mixed in the parent or in one child. The proof of inductivity requires to consider a lot of different cases depending which literals are mixed in v_p and v_c. However, each case is trivial.

Interpolation of Congruence

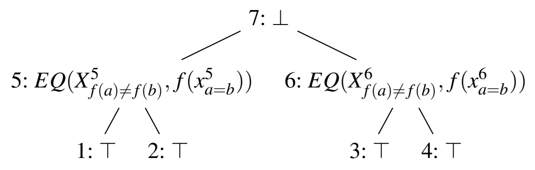
In the congruence clause $a_1 \neq b_1 \vee \cdots \vee a_n \neq b_n \vee f(a_1,\ldots,a_n) = f(b_1,\ldots,b_n)$ the clause is only mixed if the last literal, which contains all symbols, is mixed. We assume w. l. o. g. that $lca(f(a_1,\ldots,a_n)) \in st(v)$. The symbol f is shared and $f(b_1,\ldots,b_n)$ occurs in $\overline{st(v)}$. We find for each equality $a_i = b_i$ a shared term s_i as follows. We choose $s_i = a_i$ if a_i occurs in $\overline{st(v)}$, $s_i = b_i$ if b_i occurs in $st(v)$ and otherwise $s_i = x_{a_i=b_i}$ (then, $a_i = b_i$ is mixed). The interpolant is

$$I(v) = \begin{cases} \neg C \setminus \overline{st(v)} & \text{if } a_i, b_i \in symb(\overline{st(v)}) \text{ for } 1 \leq i \leq n, \\ \neg(\neg C \setminus st(v)) & \text{otherwise if } a_i, b_i \in symb(st(v)) \text{ for } 1 \leq i \leq n, \\ f(s_1,\ldots,s_n) \in X^v_{f(a_1,\ldots,a_n) \neq f(b_1,\ldots,b_n)} & \text{otherwise.} \end{cases}$$

Again, the proof requires a lot of case splits. The cases, however, are trivial.

4.2.1.4 Example

We now show how to derive a partial tree interpolant for the theory lemma $a \neq b \vee f(a) \neq q \vee f(b) = q$ used to refute the labelling of the tree interpolation problem shown in Figure 4.2 on page 38. We already presented a derivation of this lemma using the proof rules described above in Example 4.6 on page 48.



7: $\bot$
5: $EQ(X^5_{f(a)\neq f(b)}, f(x^5_{a=b}))$
6: $EQ(X^6_{f(a)\neq f(b)}, f(x^6_{a=b}))$
1: $\top$
2: $\top$
3: $\top$
4: $\top$

Fig. 4.10 Partial tree interpolant for the congruence clause $a \neq b \vee f(a) = f(b)$.

We first compute a partial tree interpolant for the congruence clause $a \neq b \vee f(a) = f(b)$. We simply use $\bot$ to label the root. For all leaves, the symbols a, b, and f occur outside of the subtree. Thus, the first case applies. Considering the projection of $a = b$ shown in Figure 4.9(a) on page 50, we conclude that $\top$ is a valid label for the leaves. Note that the projection of $f(a) \neq f(b)$ is similar to the one of $a \neq b$ only with a resp. b replaced by $f(a)$ resp. $f(b)$. For vertices 5 and 6, the last case applies since $a = b$ and $f(a) \neq f(b)$ are mixed in these vertices. Thus, we get the labels $f(x^i_{a=b}) \in X^i_{f(a)\neq f(b)}$ for $i = 5, 6$. These labels can be expressed as $EQ(X^i_{f(a)\neq f(b)}, f(x^i_{a=b}))$. The resulting partial tree interpolant for the congruence clause is shown in Figure 4.10.

Next we compute a partial tree interpolant for the symmetry clause $f(a) \neq f(b) \vee f(b) = f(a)$. Again, the labels for the root and the leaves are trivial. The labels for vertices 5 and 6 correspond to the third case. For $i = 5, 6$, we get $x^i_{f(a)=f(b)} \in X^i_{f(b)\neq f(a)}$. These labels can again be expressed as $EQ(X^i_{f(b)\neq f(a)}, x^i_{f(a)=f(b)})$. The resulting partial tree interpolant is shown in Figure 4.11.

$$\begin{tikzcd}[column sep=tiny]
 & & & 7\colon \bot \arrow[dll, dash] \arrow[drr, dash] & & & \\
 & 5\colon EQ(X^5_{f(a)\neq f(b)}, x^5_{f(a)=f(b)}) \arrow[dl, dash] \arrow[dr, dash] & & & & 6\colon EQ(X^6_{f(a)\neq f(b)}, x^6_{f(a)=f(b)}) \arrow[dl, dash] \arrow[dr, dash] & \\
1\colon \top & & 2\colon \top & & 3\colon \top & & 4\colon \top
\end{tikzcd}$$

Fig. 4.11 Partial tree interpolant for the symmetry clause $f(a) \neq f(b) \vee f(b) = f(a)$.

Now we resolve the two clauses on the pivot literal $f(a) = f(b)$. The labels for the root and the leaves are trivial. For the vertices 5 and 6 we use mixcomb to combine the labels. Consider vertex 5. We have $I_1(5) = EQ(X^5_{f(a)\neq f(b)}, f(x^5_{a=b}))$ and $I_2(5) = EQ(X^5_{f(a)\neq f(b)}, x^5_{f(a)=f(b)})$ which can be seen as $I(x^5_{f(a)=f(b)})$. If we apply the instantiation of mixcomb to uninterpreted functions we get $EQ(X^5_{f(a)\neq f(b)}, f(x^5_{a=b}))$ which corresponds to $I_1(5)$. The same reasoning leads to a label for vertex 6. The partial tree interpolant after this resolution step is the same as the partial interpolant for the congruence clause and is shown in Figure 4.10.

$$\begin{tikzcd}[column sep=tiny]
 & & & 7\colon \bot \arrow[dll, dash] \arrow[drr, dash] & & & \\
 & 5\colon x^5_{f(b)=f(a)} = q \arrow[dl, dash] \arrow[dr, dash] & & & & 6\colon x^6_{f(b)=f(a)} \neq q \arrow[dl, dash] \arrow[dr, dash] & \\
1\colon \top & & 2\colon \top & & 3\colon \top & & 4\colon \top
\end{tikzcd}$$

Fig. 4.12 Partial tree interpolant for the transitivity clause $f(b) \neq f(a) \vee f(a) \neq q \vee f(b) = q$.

Next we compute a partial tree interpolant for the transitivity clause $f(b) \neq f(a) \vee f(a) \neq q \vee f(b) = q$. The labels for the root and the leaves are trivial. For the other cases, we have to match the rules against our instantiation, i. e., the variables a, b, and c in the rule are replaced by $f(b)$, $f(a)$, and q, respectively. Since $f(b) \neq q$ occurs in vertex 6, only the last two cases can apply. Now consider vertex 5. We use the fourth case to derive $aux2(f(b), f(a), f(b)) = aux2(f(a), q, q)$ which evaluates to $x^5_{f(b)=f(a)} = q$ since $f(b) = f(a)$ is mixed in vertex 5, but $f(a) = q$ is not. For vertex 6 we are in the last case. We derive $aux2(f(b), f(a), f(b)) \neq aux2(f(a), q, q)$ which evaluates to $x^5_{f(b)=f(a)} \neq q$.

$$\begin{tikzcd}[column sep=tiny]
 & & & 7\colon \bot \arrow[dll, dash] \arrow[drr, dash] & & & \\
 & 5\colon f(x^5_{a=b}) = q \arrow[dl, dash] \arrow[dr, dash] & & & & 6\colon f(x^6_{a=b}) \neq q \arrow[dl, dash] \arrow[dr, dash] & \\
1\colon \top & & 2\colon \top & & 3\colon \top & & 4\colon \top
\end{tikzcd}$$

Fig. 4.13 Partial tree interpolant for the theory lemma $a \neq b \vee f(a) \neq q \vee f(b) = q$.

Finally, we resolve the result of the resolution of congruence and symmetry with transitivity on the pivot $f(b) = f(a)$. The labels for the root and the leaves are trivial. For vertices 5 and 6 we use mixcomb where $I_1(i) = EQ(X^i_{f(a)\neq f(b)}, f(x^i_{a=b}))$. Thus, we replace $x^i_{f(b)=f(a)}$

with $f(x^i_{a=b})$ and get the labels depicted in Figure 4.13. Note that this partial tree interpolant is also a partial tree interpolant for the clause $a \neq b$ created by resolving against the input clauses $f(a) = q$ and $f(b) \neq q$. We skip the derivation of the partial tree interpolants since all steps are trivial given the partial tree interpolants for the input clauses from Figure 4.7.

4.2.2 Theory of Linear Arithmetic

Our solver for linear arithmetic is based on a variant of the Simplex approach [39, 62]. A theory conflict is a conjunction of literals ℓ_j of the form $\sum_i c_{ij}a_i \leq c_j$. The proof of unsatisfiability is given by Farkas coefficients $k_j \geq 0$ for each inequality ℓ_j. These coefficients have the properties $\sum_j k_j c_{ij} = 0$ and $\sum_j k_j c_j < 0$. In the following we use the notation of adding inequalities (provided the coefficients are positive). Thus, we write $\sum_j k_j \ell_j$ for $\sum_i (\sum_j k_j c_{ij}) a_i \leq \sum_j k_j c_j$. With the property of the Farkas coefficients we get a contradiction ($0 < 0$) and this shows that the theory conflict is unsatisfiable.

A conjunction of literals may have rational but no integer solutions. In this case, there are no Farkas coefficients that can prove the unsatisfiability. So for the integer case, our solver may introduce extended branches [35], which are just branches of the DPLL engine on newly introduced literals. In the proof tree this results in resolution steps with these literals as pivots. We first show how to project mixed literals, then show the general form of partial interpolants, explain interpolation of theory conflicts (i. e., negated theory lemmas), and finally explain how to combine partial interpolants in a resolution step on a mixed literal.

4.2.2.1 Projection of Mixed Literals

Literals generated by extended branches or theory combination might be mixed. To define the projection of a mixed linear inequality $\ell :\equiv c_1a_1 + \ldots + c_na_n \leq c$ we introduce an auxiliary variable for every vertex in $\mathsf{mixed}(\ell)$. First we define a helper projection function

$$c_1a_1 + \cdots + c_na_n \downharpoonright v_p = \sum_{i \mid lca(a_i) = v_p} c_ia_i$$

Thus, the projection of the sum to v_p is the sum of all terms that occur in v_p for the last time. We extend the projection to sets of vertices by summing up the projections onto the individual vertices $\sum c_ia_i \downharpoonright V := \sum_{v \in V} (\sum c_ia_i \downharpoonright v)$. Then, the projection of ℓ to v_p is defined as the projection of the sum minus the auxiliary variables for the mixed vertices.

$$\ell \downharpoonright v_p = \begin{cases} (c_1a_1 + \dots + c_na_n \downharpoonright v_p) + \\ \sum\limits_{c \mid (v_c,v_p)\in E \wedge v_c \in \mathsf{mixed}(\ell)} -x_\ell^c + & \text{if } v_p = \mathsf{mixedparent}(\ell) \\ \sum\limits_{i \mid v_p \in st(lca(a_i)) \wedge v_p \neq lca(a_i)} c_ia_i \le c \\ (c_1a_1 + \dots + c_na_n \downharpoonright v_p) + \\ \sum\limits_{c \mid (v_c,v_p)\in E \wedge v_c \in \mathsf{mixed}(\ell)} -x_\ell^c + x_\ell^p \le 0 & \text{if } v_p \in \mathsf{mixed}(\ell) \\ \top & \text{otherwise} \end{cases}$$

Lemma 4.8. *The projection above is* correct *for mixed literals. Let ℓ be a literal with $\mathsf{mixed}(\ell) \neq \emptyset$, then $\ell \iff \exists\{x_\ell^j \mid v_j \in \mathsf{mixed}(\ell)\}.\ \bigwedge_{v\in V} \ell \downharpoonright v$.*

Proof. Let $\ell :\equiv a_1c_1 + \ldots + c_na_n \le c$ be a literal such that $\mathsf{mixed}(\ell) \neq \emptyset$. For every $v_j \in \mathsf{mixed}(\ell)$ we introduce an auxiliary variable x_ℓ^j. Let $\mathbf{x} :\equiv \{x_\ell^j \mid v_j \in (st(v_p) \setminus \{v_p\}) \cap \mathsf{mixed}(\ell)\}$. We first show by induction over v_p

$$\exists \mathbf{x}.\ \bigwedge_{v_j \in st(v_p)} \ell \downharpoonright v_j \iff \begin{cases} \top & \text{if } \mathsf{mixed}(\ell) \cap st(v_p) = \emptyset \\ \sum\limits_{i \mid lca(a_i) \in st(v_p)} c_ia_i + x_\ell^p \le 0 & \text{if } v_p \in \mathsf{mixed}(\ell) \\ \sum_i c_ia_i \le c & \text{if } \mathsf{mixedparent}(\ell) \in st(v_p) \end{cases}.$$

v_p is a leaf.

Note that the last case is impossible since if $v_p = lca(symb(\ell))$ then ℓ cannot be mixed in v_p and the first case would apply. The remaining cases are trivial.

v_p is an inner vertex.

If ℓ is not mixed in any vertex in $st(v_p)$ the proof is trivial.

If $v_p \in \mathsf{mixed}(\ell)$ we get $\exists \mathbf{x}.\ \bigwedge_{v_j \in st(v_c)} \ell \downharpoonright v_j = \sum\limits_{i \mid lca(a_i) \in st(v_c)} c_ia_i + x_\ell^c \le 0$ per induction hypothesis for every child $v_c \in \mathsf{mixed}(\ell)$, and $\bigwedge_{v_j \in st(v_c)} \ell \downharpoonright v_j = \top$ for all other children. Furthermore, $\ell \downharpoonright v_p = \sum\limits_{(v_c,v_p)\in E \wedge v_c \in \mathsf{mixed}(\ell)} -x_\ell^c + \sum\limits_{i \mid lca(a_i) = v_p} c_ia_i + x_\ell^p \le 0$. Combining the projections onto the subtrees of the children and the projection onto v_p we get $\exists \mathbf{x}.\ \bigwedge_{v_j \in st(v_p)} \ell \downharpoonright v_j \to \sum\limits_{i \mid lca(a_i) \in st(v_p)} c_ia_i + x_\ell^p \le 0$. To prove the other direction we set $x_\ell^c = -\sum\limits_{i \mid lca(a_i) \in st(v_c)} c_ia_i$.

If $v_p = \mathsf{mixedparent}(\ell)$ we get for every child $v_c \in \mathsf{mixed}(\ell)$ from the induction hypothesis $\exists \mathbf{x}.\ \bigwedge_{v_j \in st(v_c)} \ell \downharpoonright v_j \iff \sum\limits_{i \mid lca(a_i) \in st(v_c)} c_ia_i + x_\ell^c \le 0$. Furthermore, we have $\ell \downharpoonright v_p = \sum_{c \mid (v_c,v_p)\in E \wedge v_c \in \mathsf{mixed}(\ell)} -x_\ell^c + \sum_{i \mid lca(a_i) = v_p} c_ia_i + \sum\limits_{i \mid v_p \in st(lca(a_i)) \wedge v_p \neq lca(a_i)} c_ia_i \le c$. The remaining steps for this case are the same as in the previous case.

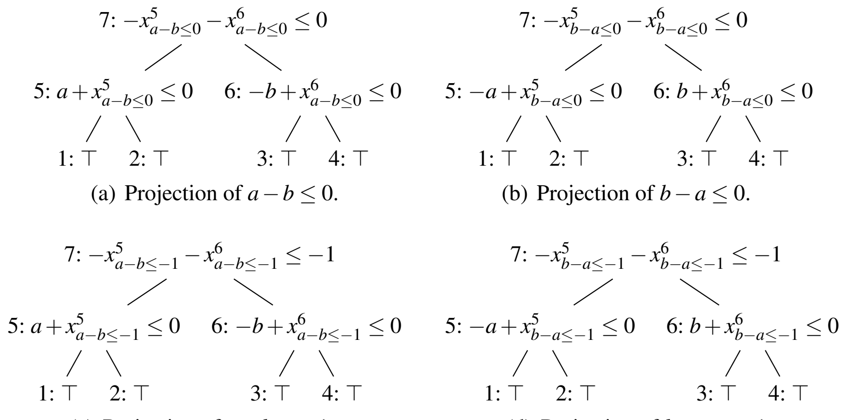
Finally, if $\mathsf{mixedparent}(\ell) \in st(v_p)$, but $v_p \neq \mathsf{mixedparent}(\ell)$, we can only have one child v_c such that $\mathsf{mixedparent}(\ell) \in st(v_c)$. In the subtrees of all other children, ℓ is not mixed. By induction hypotheses we get $\exists \mathbf{x}.\ \sum_i c_ia_i \le c$ for v_c and $\top$ for the remaining children. Additionally, $\ell \downharpoonright v_p = \top$. Thus, the conjunction yields the goal formula.

Since all vertices are contained in the subtree of the root and no literal can be mixed in the root, the last case applies. Thus, the projection function is correct. □

Example 4.8. To refute the conjunction of the labels of the interpolation tree shown in Figure 4.2 on page 38, we use the literals $a \leq b$, $b \leq a$, and their negations $a < b$ and $b < a$. We first normalise the literals.

$$\begin{aligned}
a \leq b &\equiv a - b \leq 0\\
b \leq a &\equiv b - a \leq 0\\
a < b &\equiv a - b \leq -1\\
b < a &\equiv b - a \leq -1
\end{aligned}$$

Note that the last two normalisations use integer reasoning to transform a strict inequality into a non-strict one. Since these literals occur in the resolution proof of the conjunction of the labels of the input problem, we need to compute purifications.



7: $-x^5_{a-b\leq 0} - x^6_{a-b\leq 0} \leq 0$
5: $a + x^5_{a-b\leq 0} \leq 0$
6: $-b + x^6_{a-b\leq 0} \leq 0$
1: $\top$
2: $\top$
3: $\top$
4: $\top$
7: $-x^5_{b-a\leq 0} - x^6_{b-a\leq 0} \leq 0$
5: $-a + x^5_{b-a\leq 0} \leq 0$
6: $b + x^6_{b-a\leq 0} \leq 0$
1: $\top$
2: $\top$
3: $\top$
4: $\top$
7: $-x^5_{a-b\leq -1} - x^6_{a-b\leq -1} \leq -1$
5: $a + x^5_{a-b\leq -1} \leq 0$
6: $-b + x^6_{a-b\leq -1} \leq 0$
1: $\top$
2: $\top$
3: $\top$
4: $\top$
7: $-x^5_{b-a\leq -1} - x^6_{b-a\leq -1} \leq -1$
5: $-a + x^5_{b-a\leq -1} \leq 0$
6: $b + x^6_{b-a\leq -1} \leq 0$
1: $\top$
2: $\top$
3: $\top$
4: $\top$

(a) Projection of $a - b \leq 0$.

(b) Projection of $b - a \leq 0$.

(c) Projection of $a - b \leq -1$.

(d) Projection of $b - a \leq -1$.

Fig. 4.14 Projection of the mixed literals from linear arithmetic for the tree interpolation problem shown in Figure 4.2.

The purifications are shown in Figure 4.14. In all cases, the literals are only mixed in vertices 5 and 6. Thus, the mixed parent is vertex 7. We use the last case of the projection for the leaves resulting in the formula $\top$. For the remaining cases we need $lca(a) = 5$ and $lca(b) = 6$. With these observations we compute the projections of the literals onto the remaining vertices. Note that the sum of the projection onto the vertices (where we treat $\top$ as $0 \leq 0$) yields the projected literal and cancels out every auxiliary variable. ⌟

4.2.2.2 Form of Partial Interpolants

With the projection function extended to mixed literals we now show how to compute partial interpolants. We start with an example to motivate the structure of our partial interpolants.

Example 4.9. The formula $t \le 2a \le r \le 2b+1 \le t$ has no integer solution but a rational solution. Introducing the branch $a \le b \vee b < a$ leads to the theory conflicts $t \le 2a \le 2b \le t-1$ and $r \le 2b+1 \le 2a-1 \le r-1$ (note that $\neg(a \le b) \equiv b < a$ is equivalent to $b+1 \le a$). The corresponding proof tree is given below. The Farkas coefficients in the theory lemmas are given in parenthesis. Note that the proof tree shows the clauses, i. e., the negated conflicts. A node with more than two parents denotes that multiple applications of the resolution rule are taken one after another.

$$\begin{tikzcd}
\begin{array}{r}\neg(r \le 2b+1)\ (\cdot 1)\\ \neg(b+1 \le a)\ (\cdot 2)\\ \neg(2a \le r)\ (\cdot 1)\end{array} \arrow[dr, dash] & r \le 2b+1 \arrow[d, dash] & 2a \le r \arrow[dl, dash] & & \begin{array}{r}\neg(t \le 2a)\ (\cdot 1)\\ \neg(a \le b)\ (\cdot 2)\\ \neg(2b+1 \le t)\ (\cdot 1)\end{array} \arrow[dr, dash] & t \le 2a \arrow[d, dash] & 2b+1 \le t \arrow[dl, dash] \\
 & a \le b \arrow[drr, dash] & & & & \neg(a \le b) \arrow[dll, dash] & \\
 & & & \bot & & &
\end{tikzcd}$$

Now consider the problem of deriving an interpolant between $A \equiv t \le 2a \le r$ and $B \equiv r \le 2b+1 \le t$. We can obtain an interpolant by annotating the above resolution tree with partial interpolants. To compute a partial interpolant for the theory lemma $\neg(r \le 2b+1) \vee \neg(b+1 \le a) \vee \neg(2a \le r)$, we project the *negated* clause according to the definition in Section 3.2.2, which gives

$$r \le 2b+1 \wedge x_{\neg(a\le b)} \le a \wedge -x_{\neg(a\le b)} + b + 1 \le 0 \wedge 2a \le r.$$

Then, we sum up the A-part of the conflict (the second and fourth literal) multiplied by their corresponding Farkas coefficients. This yields the interpolant $2x_{\neg(a\le b)} \le r$. Similarly, the negation of the theory lemma $\neg(t \le 2a) \vee \neg(a \le b) \vee \neg(2b+1 \le t)$ is purified to

$$t \le 2a \wedge x_{a\le b} + a \le 0 \wedge -x_{a\le b} \le b \wedge 2b+1 \le t,$$

which yields the partial interpolant $2x_{a\le b} + t \le 0$. Note, that we have to introduce different variables for each literal. Intuitively, the variable $x_{\neg(a\le b)}$ stands for a and $x_{a\le b}$ for $-a$. Using Pudlák's algorithm we can derive the same interpolants for the clause $a \le b$ resp. $\neg(a \le b)$.

For the final resolution step, the two partial interpolants $2x_{\neg(a\le b)} \le r$ and $2x_{a\le b} + t \le 0$ are combined into the final interpolant of the problem. Summing up these inequalities with $x_{\neg(a\le b)} = -x_{a\le b}$ we get $t \le r$. While this follows from A, it is not inconsistent with B. We need an additional argument that, given $r = t$, r has to be an even integer. This also follows from the partial interpolants when setting $x_{\neg(a\le b)} = -x_{a\le b}$: $t \le -2x_{a\le b} = 2x_{\neg(a\le b)} \le r$. The final interpolant computed by our algorithm is $t \le 2\left\lfloor \frac{r}{2} \right\rfloor$.

In general, we can derive additional constraints on the variables if the constraint resulting from summing up the two partial interpolants holds very tightly. We know implicitly that $x_{\neg(a\le b)} = -x_{a\le b}$ is an integer value between $t/2$ and $r/2$. If t equals r or almost equals r there are only a few possible values which we can explicitly express using the division function as in the example above. We assume that the (partial) interpolant F always has a certain property. There is some term s and some constant k, such that for $s > 0$ the interpolant is always false and for $s < -k$ the interpolant is always true (in our case $s = t - r$ and $k = 0$). For a partial interpolant that still contains auxiliary variables $\mathbf{x}$, we additionally require that s contains them with a positive coefficient and that F is monotone on $\mathbf{x}$, i. e., $\mathbf{x} \ge \mathbf{x}'$ implies $F(\mathbf{x}) \to F(\mathbf{x}')$. ⌟

To mechanise the reasoning used in the example above, our resolution rule for mixed inequality literals requires that the individual labels of the partial tree interpolant are patterns of a certain shape. An auxiliary variable of a mixed inequality literal may only occur in the pattern if the negated literal appears in the clause. Let $\mathbf{x}$ denote the set of auxiliary variables that occur in the pattern. We require that these variables only occur inside a special sub-formula of the form $LA(s(\mathbf{x}),k,F(\mathbf{x}))$. The first parameter s is a linear term over the variables in $\mathbf{x}$ and arbitrary other terms not involving $\mathbf{x}$. The coefficients of the variables $\mathbf{x}$ in s must all be positive. The second parameter $k \in \mathbb{Q}_\varepsilon$ is a constant value. In the real case we only allow the values 0 and $-\varepsilon$. In the integer case we allow $k \in \mathbb{Z}, k \geq -1$. The third parameter $F(\mathbf{x})$ is a formula that contains the variables from $\mathbf{x}$ at arbitrary positions. We require that F is monotone, i. e., $\mathbf{x} \geq \mathbf{x}'$ implies $F(\mathbf{x}) \rightarrow F(\mathbf{x}')$. Moreover, $F(\mathbf{x}) = \bot$ for $s(\mathbf{x}) > 0$ and $F(\mathbf{x}) = \top$ for $s(\mathbf{x}) < -k$. The sub-formula $LA(s(\mathbf{x}),k,F(\mathbf{x}))$ stands for $F(\mathbf{x})$ and it is only used to remember what the values of s and k are.

The intuition behind the formula $LA(s(\mathbf{x}),k,F(\mathbf{x}))$ is that $s(\mathbf{x}) \leq 0$ summarises the inequality chain that follows from the subtree of the current vertex. On this chain there may be some constraints on intermediate values. In the example above the A-part contains the chain $t \leq 2a \leq r$, which is summarised to $s \leq 0$ (with $s = t - r$). Furthermore the A-part implies that there is an even integer value between t and r. If $s < -k$ (with $k = 0$ in this case), t and r are distinct, and there always is an even integer between them. However, if $-k \leq s \leq 0$, the truth value of the interpolant depends on whether t is even.

In the remainder of the section, we give the interpolants for the leaves produced by the linear arithmetic solver and for the resolvent of the resolution step if the pivot is a mixed linear inequality.

4.2.2.3 Leaf Interpolation

As mentioned above, our solver produces for a clause $C \equiv \neg\ell_1 \vee \cdots \vee \neg\ell_m$ some Farkas coefficients $k_1,\ldots,k_m \geq 0$ such that $\sum_j k_j\ell_j$ yields a contradiction $0 < 0$. A partial interpolant for a theory lemma can be computed by summing up the projection of the conflict onto the respective subtrees: $I(v)$ is defined as $\sum_j k_j(\ell_j \downharpoonleft st(v))$ (if $\ell_j \downharpoonleft st(v) = \top$ we regard it as $0 \leq 0$, i. e., it is not added to the sum).

Lemma 4.9. *Summing up the projections onto the individual subtrees yields a partial tree interpolant for a lemma $C \equiv \neg\ell_1 \vee \cdots \vee \neg\ell_m$.*

Proof. Let $k_1,\ldots,k_m$ be the Farkas coefficients used to prove unsatisfiability of $\neg C$. To see that the symbol condition holds, we first note that all symbols that only occur inside of the subtree have to be removed from the interpolant since $\sum k_i\ell_i$ leads to a contradiction. If a symbol local to the subtree was not removed when summing up the subtree, it would not get removed at all and $\sum k_i\ell_i$ would not be contradictory.

To show inductivity, we split cases on the current vertex v.

v is a leaf.

Since all k_i are positive, we can sum up all literals $\ell_i \downharpoonleft v$ and get $I(v) = \sum_{i=1}^m k_i(\ell_i \downharpoonleft v)$ since in this case, $st(v) = \{v\}$. This sum trivially follows from $\neg C \downharpoonleft v$.

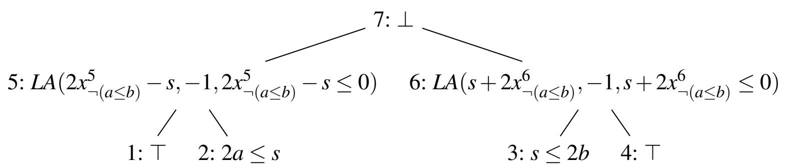
v is an inner vertex.

We have to show tree inductivity, i. e., $\bigwedge_{(v_c,v)\in E} I(v_c) \wedge F(v) \wedge (\neg C \downharpoonright v) \models \sum_{i=1}^{m} k_i(\ell \downharpoonright st(v))$. For every child v_c we have $I(v_c) = \sum_{i=1}^{m} k_i(\ell \downharpoonright st(v_c))$. Since for every literal ℓ, the projection $\ell \downharpoonright v'$ equals ℓ for exactly one vertex v', no literal ℓ is summed up in $I(v_c)$ more than one. Additionally, if ℓ is summed up in one $I(v_c)$, then $\ell \downharpoonright v = \top$ and we will not add it again. Since all k_i are positive, we get $\bigwedge_{(v_c,v)\in E} \sum_{i=1}^{m} k_i(\ell_i \downharpoonright st(v_c)) \wedge (\neg C \downharpoonright v) \models \sum_{i=1}^{m} k_i(\ell \downharpoonright st(v))$ sicne $\neg C \downharpoonright v$ contains $\ell_i \downharpoonright v$ for all i.

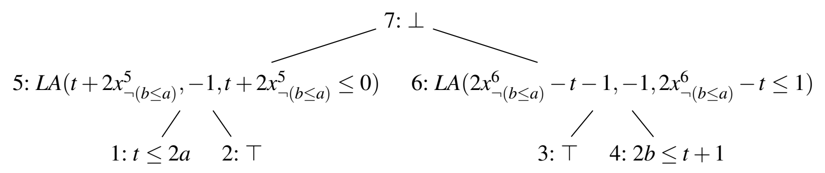
Additionally, if v is the root, we have $\neg C \downharpoonright v = \neg C$. In this case we get $I(v) = \sum_{i=1}^{m} k_i \ell_i$ which corresponds to the proof of unsatisfiability of $\neg C$. Thus, $I(v) = \bot$. □

The linear constraint $\sum_j k_j(\ell_j \downharpoonright st(v))$ can be expressed as $s(\mathbf{x}) \leq 0$. Thus, we can equivalently write this interpolant in our pattern as $LA(s(\mathbf{x}), -\varepsilon, s(\mathbf{x}) \leq 0)$. Since the Farkas coefficients are all positive and the auxiliary variables introduced to define $\ell \downharpoonright st(v)$ for mixed literals contain x positively for the current vertex, the resulting term $s(\mathbf{x})$ will also always contain x with a positive coefficient.

Example 4.10. We use the theory lemmas $\neg(2a \leq s) \vee \neg(s \leq 2b) \vee a \leq b$ and $\neg(t \leq 2a) \vee (2b \leq t+1) \vee b \leq a$ to refute the conjunction of the labels of the tree interpolation problem shown in Figure 4.2 on page 38. To compute the labels for the partial tree interpolants for these clauses, we sum up for every node the projections onto the subtree (see Figure 4.6 on page 46 and Figure 4.14 on page 58 for the purification of the individual literals). The resulting partial tree interpolants are shown in Figure 4.15.


7: $\bot$
5: $LA(2x^5_{\neg(a\leq b)} - s, -1, 2x^5_{\neg(a\leq b)} - s \leq 0)$
6: $LA(s + 2x^6_{\neg(a\leq b)}, -1, s + 2x^6_{\neg(a\leq b)} \leq 0)$
1: $\top$
2: $2a \leq s$
3: $s \leq 2b$
4: $\top$

(a) Partial tree interpolant for the clause $\neg(2a \leq s) \vee \neg(s \leq 2b) \vee a \leq b$


7: $\bot$
5: $LA(t + 2x^5_{\neg(b\leq a)}, -1, t + 2x^5_{\neg(b\leq a)} \leq 0)$
6: $LA(2x^6_{\neg(b\leq a)} - t - 1, -1, 2x^6_{\neg(b\leq a)} - t \leq 1)$
1: $t \leq 2a$
2: $\top$
3: $\top$
4: $2b \leq t+1$

(b) Partial tree interpolant for the clause $\neg(t \leq 2a) \vee \neg(2b \leq t+1) \vee b \leq a$

Fig. 4.15 Partial tree interpolants for the lemmas from linear arithmetic used to refute the conjunction of the labels of the tree interpolation problem from Figure 4.2.

Note that the partial tree interpolants do not change when resolving these lemmas with the input clauses $2a \leq s$, $s \leq 2b$, $t \leq 2a$, and $2b \leq t+1$ to generate the clauses $a \leq b$ and $b \leq a$ and computing the interpolants using Pudláks algorithm. To see this, we first remark that the last case of rule (rule-tres) applies only if the pivot of the resolution is the vertex containing this clause. For every vertex on the path from this leaf to root, we the first case since all vertices outside that subtree project the pivot literal to $\top$. The remaining cases use the second case. Second, we remark that, besides the usual simplification and absorption of neutral elements, i. e., $F \wedge \top \equiv F$, $F \wedge \bot \equiv \bot$, $F \vee \bot \equiv F$, and $F \vee \top \equiv \top$, we also simplify $(\bot \vee F) \wedge (F \vee \neg F) \equiv F$ in the leaves where the fourth case of rule (rule-tres) applies. ⌟

4.2.2.4 Pivoting of Mixed Literals

Now, we give the resolution rule for a step involving a mixed inequality $a+b \leq c$ as pivot element. To simplify presentation, we fix a vertex v and omit the dependency onto v. I. e., we write I to denote $I(v)$ and x for an auxiliary variable x^v. Furthermore, we assume the mixed literal was never merged during resolution. In the following we denote the auxiliary variable of the negated literal $\neg(a+b \leq c)$ with $x_{\neg(a+b\leq c)}$ and the auxiliary variable of $a+b \leq c$ with $x_{a+b\leq c}$. The intuition here is that $x_{\neg(a+b\leq c)}$ and $-x_{a+b\leq c}$ correspond to the same value between a and $c-b$. The partial interpolants combined by a resolution step on pivot $a+b \leq c$ have form $I_1[LA(c_1 x_{a+b\leq c} + s_1(\mathbf{x}), k_1, F_1(x_1, \mathbf{x}))]$ and $I_2[LA(c_2 x_{\neg(a+b\leq c)} + s_2(\mathbf{x}), k_2, F_2(x_2, \mathbf{x}))]$. The result of the combination should have form $I_1[I_2[LA(s_3(\mathbf{x}), k_3, F_3(\mathbf{x}))]]$. We now show how to derive s_3, k_3 and F_3 for both integer and real arithmetic.

The basic idea is to find for $\exists x. F_1(x, \mathbf{x}) \wedge F_2(-x, \mathbf{x})$ (note that $x_{a+b\leq c} = -x_{\neg(a+b\leq c)}$) an equivalent quantifier-free formula $F_3(\mathbf{x})$. To achieve this we note that we only have to look on the value of F_1 for $-k_1 \leq c_1 x_{\neg(a+b\leq c)} + s_1(\mathbf{x}) \leq 0$, since outside of this interval F_1 is guaranteed to be true resp. false. The formula F_3 must also be monotone and satisfy the range condition. We choose

$$s_3(\mathbf{x}) = c_2 s_1(\mathbf{x}) + c_1 s_2(\mathbf{x}),$$

and then F_3 will be false for $s_3(\mathbf{x}) > 0$, since either $F_1(x, \mathbf{x})$ or $F_2(-x, \mathbf{x})$ is false. The value of k_3 must be chosen such that $s_3(\mathbf{x}) < -k_3$ guarantees the existence of a value x with $c_1 x + s_1(\mathbf{x}) < -k_1$ and $-c_2 x + s_2(\mathbf{x}) < -k_2$. Hence, in the integer case, the gap between $\frac{s_2(\mathbf{x})+k_2}{c_2}$ and $\frac{-s_1(\mathbf{x})-k_1}{c_1}$ should be bigger than one. Then, $c_1 c_2 < c_2(-s_1(\mathbf{x}) - k_1) - c_1(s_2(\mathbf{x}) + k_2)$. So if we define

$$k_3 = c_2 k_1 + c_1 k_2 + c_1 c_2,$$

then there is a suitable x for $s_3(\mathbf{x}) < -k_3$. For F_3 we can then use a finite case distinction over all values where the truth value of F_1 is not determined. This suggests defining

$$F_3(\mathbf{x}) :\equiv \bigvee_{i=0}^{\left\lceil \frac{k_1+1}{c_1} \right\rceil} F_1\left(\left\lfloor \frac{-s_1(\mathbf{x})}{c_1} \right\rfloor - i, \mathbf{x}\right) \wedge F_2\left(i - \left\lfloor \frac{-s_1(\mathbf{x})}{c_1} \right\rfloor, \mathbf{x}\right) \qquad \text{(int case)}$$

In the real case, if $k_1 = -\varepsilon$, the best choice is $x = \frac{-s_1(\mathbf{x})}{c_1}$. Then, $c_1 x + s_1(\mathbf{x}) = 0 < \varepsilon$ guarantees $F_1(x)$ is true. If $k_1 = 0$, we need to consider two cases:

$$k_3 := \begin{cases} k_2 & \text{if } k_1 = -\varepsilon \\ 0 & \text{if } k_1 = 0 \end{cases}$$
$$F_3(\mathbf{x}) := \begin{cases} F_2\left(\frac{s_1(\mathbf{x})}{c_1}, \mathbf{x}\right) & \text{if } k_1 = -\varepsilon \\ s_3(\mathbf{x}) < 0 \vee \left(F_1\left(-\frac{s_1(\mathbf{x})}{c_1}, \mathbf{x}\right) \wedge F_2\left(\frac{s_1(\mathbf{x})}{c_1}, \mathbf{x}\right)\right) & \text{if } k_1 = 0 \end{cases} \qquad \text{(real case)}$$

Note that the formula of the integer case is asymmetric. If $\left\lceil \frac{k_2+1}{c_2} \right\rceil < \left\lceil \frac{k_1+1}{c_1} \right\rceil$ we can replace $-s_1$ by s_2, k_1 by k_2, and c_1 by c_2. This leads to a fewer number of disjuncts in F_3. Also note that we can remove F_1 from the last disjunct of F_3, as it will always be true.

With these definitions we can state the following lemmas.

Lemma 4.10. *Let for $i = 1,2$, $s_i(\mathbf{x})$ be linear terms over $\mathbf{x}$, $c_i \geq 0$, $k_i \in \mathbb{Z}_{\geq -1}$ (integer case) or $k_i \in \{0, -\varepsilon\}$ (real case), $F_i(x_i, \mathbf{x})$ monotone formulas with $F_i(x_i, \mathbf{x}) = \bot$ for $c_i x_i + s_i(\mathbf{x}) > 0$ and $F_i(x_i, \mathbf{x}) = \top$ for $c_i x_i + s_i(\mathbf{x}) < -k_i$. Let s_3, k_3, F_3 be as defined above. Then F_3 is monotone, $F_3(\mathbf{x}) = \bot$ for $s_3(\mathbf{x}) > 0$ and $F_3(\mathbf{x}) = \top$ for $s_3(\mathbf{x}) < -k_i$.*

Proof. Since F_1 and F_2 are monotone and they occur only positively in F_3, F_3 must also be monotone. If $s_3(\mathbf{x}) > 0$, then $\frac{-s_1(\mathbf{x})}{c_1} < \frac{s_2(\mathbf{x})}{c_2}$. Hence, for every $x \leq \frac{-s_1(\mathbf{x})}{c_1}$, $F_2(-x, \mathbf{x})$ is false since $-c_2 x + s_2(\mathbf{x}) > 0$. By definition, every disjunct of F_3 (except $s_3(\mathbf{x}) < 0$) contains $F_2(-x, \mathbf{x})$ for such an x, so $F_3(\mathbf{x})$ is false.

Now assume $s_3(\mathbf{x}) < -k_3$. For $k_1 = -\varepsilon$ in the real case, $F_3(\mathbf{x}) = F_2(-\frac{s_1(\mathbf{x})}{c_1})$ is true since $s_1(\mathbf{x}) + s_2(\mathbf{x}) < -k_2$. For $k_1 = 0$, F_3 is true by definition. In the integer case define $y := \left\lfloor \frac{-s_1(\mathbf{x})}{c_1} \right\rfloor - \left\lceil \frac{k_1+1}{c_1} \right\rceil$. This implies $c_1 y \leq -s_1(\mathbf{x}) - k_1 - 1$, hence $F_1(y, \mathbf{x})$ holds. Also $c_1 y \geq -s_1(\mathbf{x}) - k_1 - c_1$, hence

$$c_1 c_2 y + c_1 s_2(\mathbf{x}) \geq -s_3(\mathbf{x}) - c_2 k_1 - c_1 c_2 > k_3 - c_2 k_1 - c_1 c_2 = c_1 k_2.$$

Therefore, $F_2(-y, \mathbf{x})$ holds. Since y is included in the big disjunction of F_3, $F_3(\mathbf{x})$ is true. □

Lemma 4.11. *Let for $i = 1,2$, $s_i(\mathbf{x})$ be linear terms over $\mathbf{x}$, $c_i \geq 0$, $k_i \in \mathbb{Z}_{\geq -1}$ (integer case) or $k_i \in \{0, -\varepsilon\}$ (real case), $F_i(x_i, \mathbf{x})$ monotone formulas with $F_i(x_i, \mathbf{x}) = \bot$ for $c_i x_i + s_i(\mathbf{x}) > 0$ and $F_i(x_i, \mathbf{x}) = \top$ for $c_i x_i + s_i(\mathbf{x}) < -k_i$. Let F_3 be as defined above. Then*

$$F_3(\mathbf{x}) \leftrightarrow (\exists x. F_1(x, \mathbf{x}) \wedge F_2(-x, \mathbf{x}))$$

Proof (for LA$(\mathbb{Z})$). Since F_3 is a disjunction of $F_1(x, \mathbf{x}) \wedge F_2(-x, \mathbf{x})$ for different values of x, the implication from left to right is obvious. We only need to show the other direction. For this, choose x such that $F_1(x, \mathbf{x}) \wedge F_2(-x, \mathbf{x})$ holds. We show $F_3(\mathbf{x})$. We define $y := \left\lfloor \frac{-s_1(\mathbf{x})}{c_1} \right\rfloor - \left\lceil \frac{k_1+1}{c_1} \right\rceil$. This implies $y \leq \frac{-s_1(\mathbf{x}) - k_1 - 1}{c_1}$. We show F_3 by a case split on $x_1 < y$.

Case $x < y$.

Since F_2 is monotone and $-x > -y$, we have $F_2(-y, \mathbf{x})$. Also $F_1(y, \mathbf{x})$ holds since $c_1 y + s_1(\mathbf{x}) < -k_1$. This implies $F_3(\mathbf{x})$, since $F_1(y, \mathbf{x}) \wedge F_2(-y, \mathbf{x})$ is a disjunct of F_3.

Case $y \leq x$.

Since $F_1(x,\mathbf{x})$ holds, $c_1 x + s_1(\mathbf{x}) \leq 0$, hence $x \leq \left\lfloor \frac{-s_1(\mathbf{x})}{c_1} \right\rfloor$. Thus, x is one of the values $\left\lfloor \frac{-s_1(\mathbf{x})}{c_1} \right\rfloor - i$ for $0 \leq i \leq \left\lceil \frac{k_1+1}{c_1} \right\rceil$. This means the disjunction $F_3(\mathbf{x})$ includes $F_1(x,\mathbf{x}) \wedge F_2(-x,\mathbf{x})$. □

Proof (for $LA(\mathbb{Q})$). In the case $k_1 = -\varepsilon$, $F_1(\frac{-s_1(\mathbf{x})}{c_1},\mathbf{x})$ is true. From the definition of F_3, we get the implication $F_3(\mathbf{x}) \to \exists x. F_1(x,\mathbf{x}) \wedge F_2(-x,\mathbf{x})$ for $x = \frac{-s_1(\mathbf{x})}{c_1}$. If $k_1 = 0$ and $s_3(\mathbf{x}) < 0$, then $\frac{s_2(\mathbf{x})}{c_2} < \frac{-s_1(\mathbf{x})}{c_1}$ and for any value x_1 in between, $F_1(x,\mathbf{x}) \wedge F_2(-x,\mathbf{x})$ are true.

For the other direction assume that $F_1(x,\mathbf{x}) \wedge F_2(-x,\mathbf{x})$ holds. Since F_1 is not false, $x \leq \frac{-s_1(\mathbf{x})}{c_1}$ holds. If $x = \frac{-s_1(\mathbf{x})}{c_1}$ then F_3 holds by definition. In the case $k_1 = 0$ where $x < \frac{-s_1(\mathbf{x})}{c_1}$, we have $s_3(\mathbf{x}) < 0$, since $F_2(-x,\mathbf{x})$ is not false. In the case $k_1 = -\varepsilon$, we need to show that $F_2(\frac{s_1(\mathbf{x})}{c_1},\mathbf{x})$ holds. This follows from $x \leq \frac{-s_1(\mathbf{x})}{c_1}$ and monotonicity of F_2. □

If the mixed pivot has been merged during resolution steps, the partial interpolants contain multiple occurrences of LA terms which we abbreviate by LA_i, i.e., they have form $I[LA_1(s_1,k_1,F_1)]\dots[LA_n(s_n,k_n,F_n)]$ which we write as $I[LA_i]$. We abbreviate the formula $I_1[I_2[LA_{311}]\dots[LA_{31m}]]\dots[I_2[LA_{3n1}]\dots[LA_{3nm}]]$ by $I_1[I_2[LA_{3ij}]]$. With this notation we get

$$\mathsf{mixcomb}(t \leq c, I_1[LA_{1i}(x)], I_2[LA_{2j}(-x)]) :\equiv I_1[I_2[LA_{3ij}]].$$

We need the following lemma to prove correctness of the mixed combination.

Lemma 4.12. *Let $I_1[LA_{1i}]$, $I_2[LA_{2j}]$, and $I_1[I_2[LA_{3ij}]]$ as defined above. Then,*

$$I_1[I_2[LA_{3ij}]] \iff ((\exists x.\ I_1[LA_{1i}(x)] \wedge I_2[LA_{2j}(-x)]) \vee \forall x.\ I_1[LA_{1i}(x)]).$$

Proof. We split the proof into three parts. First, we show $\forall x.\ I_1[LA_{1i}(x)] \to I_1[I_2[LA_{3ij}]]$. For each i, there exists x_i such that $LA_{1i}(x)$ is false since $s_{1i}(x) > 0$. Furthermore, the coefficient of x in $LA_{1i}(x)$ is positive and this formula is monotone on x. We choose $x = \max(x_i)$ and conclude $(\forall x.\ I_1[LA_{1i}(x)]) \iff I_1[\bot]$ since for the chosen x, all $LA_{1i}(x)$ are $\bot$. Furthermore we have $\bigwedge_{i,j} \bot \to I_2[LA_{3ij}]$. With Lemma 2.1 we get $I_1[I_2[LA_{3ij}]]$.

Next, we use Lemma 4.11 to compute for each i,j an x_{ij} with

$$LA_{3ij} \to LA_{1i}(x_{ij}) \wedge LA_{2j}(-x_{ij})$$

For every i, set $x_i = \min\{x_{ij} | LA_{3ij}\}$. If this set is empty, we choose x_i such that $LA_{1i}(x_i)$ holds. Then, $LA_{1i}(x_i)$ and $LA_{3ij} \to LA_{2j}(-x_i)$ hold for all i,j. We choose $x = \max\{x_i | I_2[LA_{3ij}]\}$. If the set is empty, we choose an arbitrary value for x_i. Then, $I_2[LA_{3ij}] \to LA_{1i}(x)$. Therefore, $I_1[I_2[LA_{3ij}]] \to I_1[LA_{1i}(x)]$. If the set was not empty, then $I_2[LA_{3ij}]$ was true for i where $x = x_i$. Hence, $I_2[LA_{2j}(-x)]$ is true, since $LA_{3ij} \to LA_{2j}(-x)$. If the set was empty, then $I_1[\bot]$ is true, hence $\forall x I_1[LA_{1i}(x)]$ holds. □

This lemma can be used to show that our instantiation of $\mathsf{mixcomb}$ for mixed inequalities is correct.

Theorem 4.1 (Soundness of $\mathsf{mixcomb}$ instantiation for inequalities). *Let $t \leq c$ be a mixed literal. If I_1 is a partial tree interpolant of $C_1 \vee t \leq c$ and I_2 is a partial tree interpolant of*

$C_2 \vee \neg(t \leq c)$, then I_3 computed according to rule (rule-tres) is a partial tree interpolant for $C_1 \vee C_2$.

Proof. We only consider the cases where $t \leq c$ is mixed in vertex v_p or where $v_p = \mathsf{mixedparent}(t \leq c)$. The remaining cases have proofs similar to those of Lemma 4.5. Furthermore, we partition the children of v_p into the sets V_m of children in which $t \leq c$ is mixed, and V_r of children v_c such that $t \leq c \downharpoonright v_c = \top$ since all symbols occur outside of the subtree rooted at v_c.

The symbol condition holds in both cases since, in the mixcomb rule, we remove the auxiliary variables of the literal created for v_p. All other variables and symbols are still allowed assuming I_1 and I_2 satisfy the symbol condition.

If $t \leq c$ is mixed in v_p, we assume

$$\bigwedge_{v \in V_m} I_1(v)[I_2(v)[LA^v_{3ij}]] \wedge \bigwedge_{v \in V_r} I_1(v) \wedge I_2(v) \wedge F(v_p) \wedge \neg(C_1 \vee C_2) \downharpoonright v_p$$

and show $I_1(v_p)[I_2(v_p)[LA^{v_p}_{3ij}]]$ under the assumptions

$$\bigwedge_{v \in V_m} I_1(v)[LA^v_{1i}(x_v)] \wedge \bigwedge_{v \in V_r} I_1(v) \wedge F(v_p) \wedge \neg(C_1 \vee t \leq c) \downharpoonright v_p \models I_1(v_p)[LA^{v_p}_{1i}(x_p)] \tag{4.3}$$

$$\bigwedge_{v \in V_m} I_2(v)[LA^v_{2j}(-x_v)] \wedge \bigwedge_{v \in V_r} I_2(v) \wedge F(v_p) \wedge \neg(C_2 \vee \neg(t \leq c)) \downharpoonright v_p \models I_2(v_p)[LA^{v_p}_{2j}(-x_p)] \tag{4.4}$$

Note that $t \leq c \downharpoonright v_p$ and $(\neg(t \leq c)) \downharpoonright v_p$ contain auxiliary variables for every vertex in v_m and the auxiliary variable $x^p_{t \leq c}$ resp. $x^p_{\neg(t \leq c)}$ for the current vertex. In the remainder of this proof, we use x^p to denote $x^p_{\neg(t \leq c)}$.

From Lemma 4.12 and $I_1(v)[I_2(v)[LA^v_{3ij}]]$ we either get for each $v \in V_m$ a x^v such that $I_1(v)[LA^v_{1i}(x^v)] \wedge I_2(v)[LA^v_{2j}(-x^v)]$, or there is at least one $v \in V_m$ such that $\forall x.\ I_1(v)[LA^v_{1i}(x)]$ holds.

In the first case we instantiate the variables for the children in the projections to the witnesses of the existential quantifier and v_p such that $t \leq c \downharpoonright v_p$ and $(\neg t \leq c) \downharpoonright v_p$ hold, i. e., we set $x^p = -\sum_{i | lca(a_i) = v_p} c_i a_i + \sum_{v \in V_m} x^v$. Then, we get $I_1(v_p)[LA^{v_p}_{1i}(x^p)]$ from (4.3) and $I_2(v_p)[LA^{v_p}_{2j}(-x^p)]$ from (4.3). Using Lemma 4.12 we get $I_1(v_p)[I_2(v_p)[LA^p_{3ij}]]$.

If for some children $\forall x.\ I_1(v)[LA^v_{1i}(x)]$ holds, it holds for all $v \in V_m$. Using Lemma 4.12 we get $I_1(v_p)[I_2(v_p)[LA^p_{3ij}]]$.

If $v_p = \mathsf{mixedparent}(t \leq c)$, we have to show $I_1(v_p) \vee I_2(v_p)$ from the assumptions

$$\bigwedge_{v \in V_m} I_1(v)[I_2(v)[LA^v_{3ij}]] \wedge \bigwedge_{v \in V_r} I_1(v) \wedge I_2(v) \wedge F(v_p) \wedge \neg(C_1 \vee C_2) \downharpoonright v_p \tag{4.5}$$

$$\bigwedge_{v \in V_m} I_1(v)[LA^v_{1i}(x_v)] \wedge \bigwedge_{v \in V_r} I_1(v) \wedge F(v_p) \wedge \neg(C_1 \vee t \leq c) \downharpoonright v_p \models I_1(v_p) \tag{4.6}$$

$$\bigwedge_{v \in V_m} I_2(v)[LA^v_{2j}(-x_v)] \wedge \bigwedge_{v \in V_r} I_2(v) \wedge F(v_p) \wedge \neg(C_2 \vee \neg(t \leq c)) \downharpoonright v_p \models I_2(v_p) \tag{4.7}$$

Again, we split $I_1(v)[I_2(v)[LA^v_{3ij}]]$ according to Lemma 4.12. If for all $v \in V_m\ \forall x.\ I_1(v)[LA^v_1(x)]$ holds, we choose an x such that $(\neg t \leq c) \downharpoonright v_p$ holds. Then, we get $I_1(v_p)$ from (4.6). Otherwise, we get an x^v for every $v \in V_m$. With these values, either $(\neg t \leq c) \downharpoonright v_p$ or $t \leq c \downharpoonright v_p$ holds yielding either $I_1(v_p)$ or $I_2(v_p)$ by (4.6) or (4.7). □

4.3 Combining Theories

Typical problems in SMT contain literals from various theories. To combine these theories, the Nelson–Oppen [75] combination method is typically used. In the remainder of this chapter we show how to compute partial tree interpolants in the context of Nelson–Oppen based theory combination. We focus the presentation on the combination of the theory of uninterpreted functions with the theory of linear arithmetic.

4.3.1 Nelson–Oppen-based Theory Combination

In the Nelson–Oppen combination methods, theories are required to only share the equality symbol. Furthermore, theories need to be stably infinite. A theory is stably infinite if and only if every finite model embeds into an infinite one. This property is satisfied by most theories. The theory of uninterpreted functions and the theory of linear arithmetic considered in this thesis are two prominent examples of stably infinite theories.

If both requirements mentioned above are satisfied, two theories can be combined using a procedure consisting of three steps. First, the input formula is *purified* such that every literal only contains symbols interpreted by one or both theories. This is achieved by adding new symbols to the formula. Consider the formula $f(x+1) = y$ where f is an uninterpreted function, and x and y are integers. Then, purification creates the equisatisfiable formula $f(s) = y \land s = x+1$. The atom $f(s) = y$ is then interpreted by the theory of uninterpreted functions while $s = x+1$ is interpreted by the theory of linear arithmetic. Purification is saturated until completion yielding a formula that can be separated w. r. t. the different theories.

Then, the algorithm guesses an *arrangement* of the new symbols introduced during purification. This arrangement is a set of equalities and disequalties between these symbols. Nelson and Oppen showed that sharing only equalities between the theories is sufficient to amalgamate the models generated by every theory for the symbols interpreted by this theory into a model for the original input formula. For convex theories, the arrangement is typically deduced from the formula since entailed equalities can cheaply be propagated.

Finally, solving is preformed of the purified formula and the arrangement. If the solver returns unsatisfiability, a new arrangement is tested until either a satisfiable arrangement is found or all arrangements are proven unsatisfiable.

Various improvements to the procedure have been proposed in recent years. These improvements reduce the number of assignments guessed during the second phase of the algorithm. Regardless of the improvements of the combination procedure, only equalities between terms shared by multiple theories need to be propagated. If for two new symbols equality between these symbols is not propagated, the individual theory solvers generate (partial) models where these symbols are assigned different values.

In the context of interpolation, the equalities of the arrangement might mix symbols produced by the purification of the labelling of different vertices. Then, the equality literal might be mixed. Furthermore, this literal might occur in theory lemmas among with other mixed literals and non-mixed literals. Note however that theory combination only creates equalities for symbols created during purification. These symbols represent terms already present in the input. Thus, theory combination does not create new terms.

Additionally, theories might introduce *theory combination clauses* to relate (dis-)equalities to the operators used by the theory. Consider, e. g., the theory of linear arithmetic. Equality is expressed by the clauses $s \neq t \vee s \leq t$ and $s \neq t \vee s \geq t$ where only $s \leq t$ and $s \geq t$ are interpreted by the theory of linear arithmetic. We use the trichotomy clause $s = t \vee \neg(s \leq t) \vee \neg(s \geq t)$ to express disequality. These clauses can be seen as a way to propagate (dis-)equalities from resp. to the theory of linear arithmetic. Note that, in the context of interpolation, these clauses might be mixed.

4.3.2 Interpolating the Combination of Uninterpreted Functions and Linear Arithmetic

To combine linear arithmetic with the theory of uninterpreted functions, we use trichotomy clauses $s = t \vee \neg(s \leq t) \vee \neg(s \geq t)$, and clauses $s \neq t \vee s \leq t$ and $s \neq t \vee s \geq t$ interpreting equality in the vocabulary of linear arithmetic. These clauses might occur in a proof tree. Thus, we need to compute partial tree interpolants for them. We distinguish two different cases.

In the first case we assume both literals are not mixed in any vertex of the interpolation tree. Then, we can compute an interpolant for every vertex by conjoining the projections of the negated literals onto v, i. e., we set

$$I(v) = (s \neq t) \downharpoonright v \wedge (s \geq t) \downharpoonright v \wedge (s \leq t) \downharpoonright v$$
$$I(v) = (s = t) \downharpoonright v \wedge (s > t) \downharpoonright v$$

for $s = t \vee \neg(s \leq t) \vee \neg(s \geq t)$ resp. $s \neq t \vee s \leq t$.

In the second case, the literals are mixed. Note however that we only consider a limited form of mixed literals. In the Nelson–Oppen combination method, we only generate new equalities between terms that are shared between the two theories. Since in all combination clauses, an equality is created, the left hand side and the right hand side must already exist in the input. These equalities might contain shared symbols, but the left hand side and the right hand side have to contain at least one symbol local to a subtree. We will denote these symbols a and b for the symbol containing a part local to the current subtree, resp. local to outside of the current subtree.

To compute partial tree interpolants we first note that one of the literals is mixed if and only it the other literals of the clauses are mixed, too. Thus, the $\mathsf{mixedparent}$ vertices of all literals are the same. Furthermore, for all vertices where $\mathsf{mixedparent}$ is not a descendent, all symbols occur outside of the subtree. This already gives us two important cases. Two more cases are given by the fact that the least common ancestor of a might be different from $\mathsf{mixedparent}$. If $\mathsf{mixedparent}$ is in the subtree of the current vertex, both symbols occur inside the subtree. These cases lead to the following definition.

Definition 4.3. Let $\delta \in \{-1,1\}$, $a = b$ be a mixed literal, and $v_* = \mathsf{mixedparent}(\cdot)$ be the mixed parent of the mixed literals. Then,

$$I(v) = \begin{cases} \bot & \text{if } v_* \in st(v) \\ -\delta y_{a=b} + x_{\neg(\delta a - \delta b \leq 0)} \leq 0 & \text{if } a \in symb(v) \wedge v \in \mathsf{mixed}(\cdot) \\ \delta y_{a=b} + x_{\neg(\delta a - \delta b \leq 0)} \leq 0 & \text{if } b \in symb(v) \wedge v \in \mathsf{mixed}(\cdot) \\ \top & \text{otherwise} \end{cases}$$

$$I(v) = \begin{cases} \bot & \text{if } v_* \in st(v) \\ x_{\neg(a-b\leq 0)} + x_{\neg(b-a\leq 0)} \leq 0 \wedge & \\ (x_{\neg(a-b\leq 0)} + x_{\neg(b-a\leq 0)} \geq 0 \to x_{\neg(a-b\leq 0)} \in Y_{a\neq b}) & \text{if } a \in symb(v) \wedge v \in \mathsf{mixed}(\cdot) \\ x_{\neg(a-b\leq 0)} + x_{\neg(b-a\leq 0)} \leq 0 \wedge & \\ (x_{\neg(a-b\leq 0)} + x_{\neg(b-a\leq 0)} \geq 0 \to -x_{\neg(a-b\leq 0)} \in Y_{a\neq b}) & \text{if } b \in symb(v) \wedge v \in \mathsf{mixed}(\cdot) \\ \top & \text{otherwise} \end{cases}$$

are partial tree interpolants for the theory combination clauses $a \neq b \vee \delta a - \delta b \leq 0$ resp. $a = b \vee a - b < 0 \vee b - a < 0$. To ease presentation we omitted the reference to the current vertex.

Lemma 4.13. *The formulas given in Definition 4.3 satisfy the conditions of partial tree interpolants.*

Proof. We first note that the formulas satisfy the restrictions of our interpolation scheme. The formulas could easily be folded into LA and EQ form. Then, all variables created by projection of arithmetic literals occur positively in the sum, the set-valued variable $Y_{a\neq b}$ occurs only if $a \neq b$ occurs in the negation of the trichotomy clause and only in the correct position for our pattern.

Next we have to show tree inductivity. Consider C_δ. If all symbols occur outside the subtree of v, the projection of the literals is $\top$. Thus, $\top$ satisfies the conditions for partial tree interpolants in this case. Now assume $a \in symb(v)$ and $v \in \mathsf{mixed}(\cdot)$. We get two cases. If $v = lca(a)$, the projection of $\neg C_\delta$ onto v yields $a = y_{a=b} \wedge -\delta a + x_{\neg(\delta a - \delta b \leq 0)} \leq 0$. Note that the clause is not mixed in the children since both symbols occur outside of the subtree rooted at a child. Hence, the interpolants of the children are $\top$ by induction hypothesis. We conclude that $-\delta y_{a=b} + x_{\neg(\delta a - \delta b \leq 0)} \leq 0$ follows from the interpolants of the children and the projection of the negated clause.

If $v = lca(b)$, we get $y_{a=b} = b \wedge \delta b + x_{\neg(\delta a - \delta b \leq 0)} \leq 0$ when projecting $\neg C_\delta$ onto v. The interpolant $\delta y_{a=b} + x_{\neg(\delta a - \delta b \leq 0)} \leq 0$ follows from this formula and the interpolants of the children which are again $\top$.

Now let v be a vertex such that the literals are mixed in v, but v is not v_*. Since the argumentation is symmetric, we assume b only occurs outside of the subtree. Then, we get the projection $y^c_{a=b} = y^p_{a=b} \wedge -x^c_{\neg(\delta a - \delta b \leq 0)} + x^p_{\neg(\delta a - \delta b \leq 0)} \leq 0$. With the interpolant $\delta y^c_{a=b} + x^c_{\neg(\delta a - \delta b \leq 0)} \leq 0$ we get $\delta y^p_{a=b} + x^p_{\neg(\delta a - \delta b \leq 0)} \leq 0$.

Finally consider $v = v_*$. Let v_1 and v_2 be the children of v_* in which the literals are mixed. Then, the projection of the negated clause is $y^{v_1}_{a=b} = y^{v_2}_{a=b} \wedge -x^{v_1}_{\neg(\delta a - \delta b \le 0)} - x^{v_2}_{\neg(\delta a - \delta b \le 0)} \le -\varepsilon$ since v_* is neither $lca(a)$ nor $lca(b)$. Together with the interpolants $-\delta y^{v_1}_{a=b} + x^{v_1}_{\neg(\delta a - \delta b \le 0)} \le 0$ and $\delta y^{v_2}_{a=b} + x^{v_2}_{\neg(\delta a - \delta b \le 0)} \le 0$ from the two mixed children we get $0 \le -\varepsilon$ which is contradictory. Thus, $\perp$ is a correct interpolant for v and, by induction, all vertices above v.

Now consider trichotomy. Again, if all symbols occur outside the subtree of v, all projections are $\top$ and $\top$ is a valid interpolant. If $v = lca(a)$, we get the projection $a \in Y_{a\neq b} \wedge -a + x_{\neg(a<b)} \le 0 \wedge a + x_{\neg(b<a)} \le 0$. Summing up the inequalities we get $x_{\neg(a<b)} + x_{\neg(b<a)} \le 0$. Furthermore, if $x_{\neg(a<b)} = -x_{\neg(b<a)}$ holds, $x_{\neg(a<b)} = a$ has to hold. Thus, in this case, $x_{\neg(a<b)} \in Y_{a\neq b}$ follows from the projection.

If $v = lca(b)$ we get $b \in Y_{a\neq b} \wedge b + x_{\neg(a<b)} \le 0 \wedge -b + x_{\neg(b<a)} \le 0$. Reasoning similar than before yields the desired interpolant. Note that in this case, if $x_{\neg(a<b)} = -x_{\neg(b<a)}$, then $x_{\neg(a<b)} = -b$.

Now assume v is a vertex between $lca(a)$ and v_*. Then, the projection of trichotomy onto v gives $Y^c_{a\neq b} \subseteq Y^p_{a\neq b} \wedge -x^c_{\neg(a<b)} + x^p_{\neg(a<b)} \le 0 \wedge -x^c_{\neg(b<a)} + x^p_{\neg(b<a)} \le 0$. Furthermore, by induction hypothesis, we get for all non-mixed children the interpolant $\top$ and for the mixed children $x^c_{\neg(a<b)} + x^c_{\neg(b<a)} \le 0 \wedge (x^c_{\neg(a<b)} + x^c_{\neg(b<a)} \ge 0 \to x^c_{\neg(a<b)} \in Y^c_{a\neq b}$. Combining the linear constraints we get $x^p_{\neg(a<b)} + x^p_{\neg(b<a)} \le 0$. If $x^p_{\neg(a<b)} = -x^p_{\neg(b<a)}$, we get $x^c_{\neg(a<b)} + x^c_{\neg(b<a)} \ge 0$ and $x^c_{\neg(a<b)} = x^p_{\neg(a<b)}$. Thus, in this case, $x^p_{\neg(a<b)} \in Y^c_{a\neq b} \subseteq Y^p_{a\neq b}$ yields the desired formula.

Finally, let $v = v_*$. Let v_1 and v_2 be the two children in which the clauses are mixed. Then, the projection is $Y^{v_1}_{a\neq b} \cap Y^{v_2}_{a\neq b} = \emptyset \wedge -x^{v_1}_{\neg(a<b)} - x^{v_2}_{\neg(a<b)} \le 0 \wedge -x^{v_1}_{\neg(b<a)} - x^{v_2}_{\neg(b<a)} \le 0$. From the induction hypothesis we get for the non-mixed children the interpolant $\top$, for the child whose subtree contains a we get (w. l. o. g. we assume this child has index 1) $x^{v_1}_{\neg(a<b)} + x^{v_1}_{\neg(b<a)} \le 0 \wedge (x^{v_1}_{\neg(a<b)} + x^{v_1}_{\neg(b<a)} \ge 0 \to x^{v_1}_{\neg(a<b)} \in Y^{v_1}_{a\neq b})$ and for the other mixed child we get $x^{v_2}_{\neg(a<b)} + x^{v_2}_{\neg(b<a)} \le 0 \wedge (x^{v_2}_{\neg(a<b)} + x^{v_2}_{\neg(b<a)} \ge 0 \to -x^{v_2}_{\neg(a<b)} \in Y^{v_2}_{a\neq b})$. Combining the linear constraints we get $x^{v_1}_{\neg(a<b)} = -x^{v_2}_{\neg(a<b)}$. Thus we get $x^{v_1}_{\neg(a<b)} \in Y^{v_1}_{a\neq b}$ and $x^{v_1}_{\neg(a<b)} = -x^{v_2}_{\neg(a<b)} \in Y^{v_2}_{a\neq b}$. This is contradictory with $Y^{v_1}_{a\neq b} \cap Y^{v_2}_{a\neq b} = \emptyset$. Thus, $\perp$ is an interpolant for v_* and all vertices above it. $\square$

Example 4.11. In our running example shown in Figure 4.2 on page 38, we use the trichotomy clause $a = b \vee \neg(b \le a) \vee \neg(a \le b)$ to refute the conjunction of the labels of the vertices in the input tree. We now compute a partial tree interpolant for this clause.

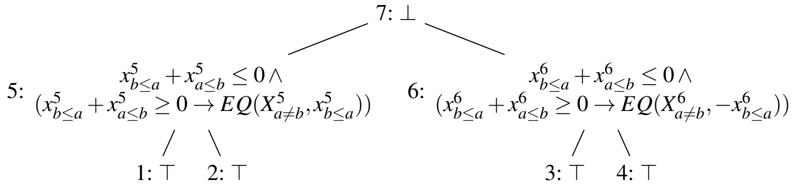

7: $\perp$
5: $x^5_{b\le a} + x^5_{a\le b} \le 0 \wedge (x^5_{b\le a} + x^5_{a\le b} \ge 0 \to EQ(X^5_{a\neq b}, x^5_{b\le a}))$
6: $x^6_{b\le a} + x^6_{a\le b} \le 0 \wedge (x^6_{b\le a} + x^6_{a\le b} \ge 0 \to EQ(X^6_{a\neq b}, -x^6_{b\le a}))$
1: $\top$
2: $\top$
3: $\top$
4: $\top$

Fig. 4.16 Partial tree interpolant for the clause $a = b \vee \neg(b \le a) \vee \neg(a \le b)$.

Since the literals are only mixed in vertices 5 and 6, the labels for the other vertices are trivial. In vertex 5, we have $a \in symb(v_5)$, but not b. Thus, we have the second case of Definition 4.3. We use the variables $x^5_{b\leq a}$, $x^5_{a\leq b}$, and $X^5_{a\neq b}$ in the projection of the literals $b \leq a$, $a \leq b$, and $a \neq b$, respectively. Similarly, we compute the label for vertex 6. But this time, $b \in symb(v_6)$, while a is not. Thus, the third case applies. Note that there is a small but significant difference between case two and three in the EQ term. The auxiliary variable $x^5_{b\leq a}$ occurs positively in the EQ term in vertex 5.

We can write the labels in LA form. Let x^i abbreviate $x^i_{b\leq a} + x^i_{a\leq b}$. Then, we can write for $i = 5, 6$ the label of vertex i as $LA(x^i, 0, x^i \leq 0 \wedge (x^i \geq 0 \rightarrow EQ(X^i_{a\neq b}, (-1)^{\delta_{i6}} x^i_{b\leq a}))$ where δ_{i6} is the Kronecker delta, i. e., it is 1 if and only if $i = 6$ and 0 in all other cases. ⌟

4.4 An Example for the Combined Theory

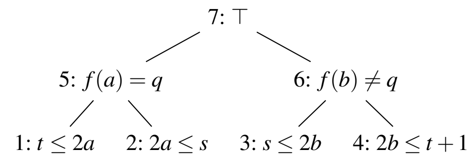


Fig. 4.17 Tree Interpolation Problem from Example 4.2

We now show how to compute a tree interpolant for the tree interpolation problem from Example 4.2 (the interpolation tree is shown again in Figure 4.17). We first show how to derive a tree interpolant for this problem. Then, we prove correctness of the generated tree interpolant.

4.4.1 Derivation of a Tree Interpolant

The complete proof of unsatisfiability of the conjunction of the labels of the vertices uses a lot of resolution steps. We already considered in previous examples how to use resolution on theory lemmas and input clauses to create the unit clauses $a \neq b$, $b \leq a$, and $a \leq b$ and the corresponding partial tree interpolants. We first restate the partial tree interpolants in Figure 4.18 before combining them with the interpolant for the trichotomy clause presented in Example 4.11. To simplify presentation, let $F_i[t] := x^i_{b\leq a} + x^i_{a\leq b} \leq 0 \wedge (x^i_{b\leq a} + x^i_{a\leq b} \geq 0 \rightarrow t)$.

We compute the final interpolant from the partial tree interpolants of these clauses. For this, we annotate the resolution proof

$$\begin{tikzcd}[column sep=small]
 & & & 7\colon \bot \arrow[dll, dash] \arrow[drr, dash] & & & \\
 & 5\colon f(x^5_{a=b}) = q \arrow[dl, dash] \arrow[dr, dash] & & & & 6\colon f(x^6_{a=b}) \neq q \arrow[dl, dash] \arrow[dr, dash] & \\
1\colon \top & & 2\colon \top & & 3\colon \top & & 4\colon \top
\end{tikzcd}$$

(a) Clause $a \neq b$.

$$\begin{tikzcd}[column sep=small]
 & & & 7\colon \bot \arrow[dll, dash] \arrow[drr, dash] & & & \\
 & 5\colon LA(2x^5_{\neg(a\leq b)} - s, -1, 2x^5_{\neg(a\leq b)} - s \leq 0) \arrow[dl, dash] \arrow[dr, dash] & & & & 6\colon LA(s + 2x^6_{\neg(a\leq b)}, -1, s + 2x^6_{\neg(a\leq b)} \leq 0) \arrow[dl, dash] \arrow[dr, dash] & \\
1\colon \top & & 2\colon 2a \leq s & & 3\colon s \leq 2b & & 4\colon \top
\end{tikzcd}$$

(b) Clause $a \leq b$.

$$\begin{tikzcd}[column sep=small]
 & & & 7\colon \bot \arrow[dll, dash] \arrow[drr, dash] & & & \\
 & 5\colon LA(t + 2x^5_{\neg(b\leq a)}, -1, t + 2x^5_{\neg(b\leq a)} \leq 0) \arrow[dl, dash] \arrow[dr, dash] & & & & 6\colon LA(2x^6_{\neg(b\leq a)} - t - 1, -1, 2x^6_{\neg(b\leq a)} - t \leq 1) \arrow[dl, dash] \arrow[dr, dash] & \\
1\colon t \leq 2a & & 2\colon \top & & 3\colon \top & & 4\colon 2b \leq t + 1
\end{tikzcd}$$

(c) Clause $b \leq a$.

$$\begin{tikzcd}[column sep=small]
 & & & 7\colon \bot \arrow[dll, dash] \arrow[drr, dash] & & & \\
 & 5\colon F_5[EQ(X^5_{a\neq b}, x^5_{b\leq a})] \arrow[dl, dash] \arrow[dr, dash] & & & & 6\colon F_6[EQ(X^6_{a\neq b}, -x^6_{b\leq a})] \arrow[dl, dash] \arrow[dr, dash] & \\
1\colon \top & & 2\colon \top & & 3\colon \top & & 4\colon \top
\end{tikzcd}$$

(d) Clause $a = b \vee \neg(b \leq a) \vee \neg(a \leq b)$.

Fig. 4.18 Partial tree interpolants for the clauses containing mixed literals. Derivation of the labels were shown in Example 4.10 and Example 4.11.

$$\dfrac{\dfrac{\vdots}{a \leq b} \qquad \dfrac{\dfrac{\vdots}{b \leq a} \qquad \dfrac{\dfrac{}{a = b \vee \neg(b \leq a) \vee \neg(a \leq b)} \qquad \dfrac{\vdots}{a \neq b}}{\neg(b \leq a) \vee \neg(a \leq b)}}{\neg(a \leq b)}}{\bot}$$

with partial tree interpolants. The final interpolant will be the annotation of the empty clause.

In the first resolution step, we combine the partial tree interpolant for trichotomy with the partial tree interpolant for the clause $a \neq b$. Figure 4.19 shows the resulting partial tree interpolant after the resolution step with the pivot $a = b$. The labels for the vertices 1 to 4 and 7 are trivial since the pivot is not mixed in these vertices. In vertices 5 and 6 we use the instantiation of $\mathsf{mixcomb}$ for uninterpreted functions. For vertex 5, we identify the components $I_1 \equiv F_5$ and $I_2(x) \equiv f(x) = q$. We apply $\mathsf{mixcomb}$ and compute $I_1[I_2(x^5_{b\leq a})] \equiv$

$$\begin{tikzcd}
& & & 7\colon \bot \arrow[dll, dash] \arrow[drr, dash] & & & \\
& 5\colon F_5[f(x^5_{b\leq a}) = q] \arrow[dl, dash] \arrow[dr, dash] & & & & 6\colon F_6[f(-x^6_{b\leq a}) \neq q] \arrow[dl, dash] \arrow[dr, dash] & \\
1\colon \top & & 2\colon \top & & 3\colon \top & & 4\colon \top
\end{tikzcd}$$

Fig. 4.19 Partial tree interpolant for the clause $\neg(b \leq a) \vee \neg(a \leq b)$.

$F_5[f(x^5_{b\leq a}) = q]$. We compute the label for vertex 6 as $I_1[I_2(-x^6_{b\leq a})] \equiv F_6[f(-x^6_{b\leq a}) \neq q]$, were $I_1 \equiv F_6$ and $I_2(x) \equiv f(x) \neq q$.

The next resolution step has pivot $b \leq a$. Since this literal is mixed, we use the instantiation of mixcomb to mixed inequalities. Again, the labels for vertices 1 to 4 and 7 are trivial given that $b \leq a$ is not mixed in these vertices. To compute the label for vertex 5, we combine $LA(2x^5_{b\leq a} - s, -1, 2x^5_{b\leq a} - s \leq 0)$ with $LA(x^5_{b\leq a} + x^5_{a\leq b}, 0, F[f(x^5_{b\leq a}) = q])$. From these patterns we extract the components

$$\begin{array}{llll}
c_1 = 2 & s_1 = t & k_1 = -1 & F_1(x) \equiv t + 2x \leq 0 \\
c_2 = 1 & s_2 = x^5_{a\leq b} & k_2 = 0 & F_2(x) \equiv F[f(x) = q]
\end{array}$$

which lead to the components $s_3 = t + 2x^5_{a\leq b}$ and $k_3 = 2 \cdot 0 + 1 \cdot (-1) + 2 \cdot 1 = 1$. Since $k_1 = -1$, we get one disjunction in the resulting formula. We instantiate the definition of F_3 from Section 4.2.2.4 and compute

$$G_5(x) = -\left\lfloor \frac{-t}{2} \right\rfloor + x \leq 0 \wedge \left(-\left\lfloor \frac{-t}{2} \right\rfloor + x \geq 0 \rightarrow f\left(-\left\lfloor \frac{-t}{2} \right\rfloor \right) = q \right).$$

Similarly, we combine the patterns $LA(2x_6 - t - 1, -1, 2x_6 - t \leq 1)$ with $LA(x^6_{b\leq a} + x^6_{a\leq b}, 0, F_6[f(-x^6_{b\leq a}) \neq q])$ to compute the label for vertex 6. From the components

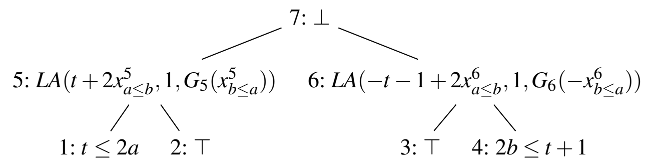
$$\begin{array}{llll}
c_1 = 2 & s_1 = -t - 1 & k_1 = -1 & F_1(x) \equiv 2x - t \leq 1 \\
c_2 = 1 & s_2 = x^6_{a\leq b} & k_2 = 0 & F_2(x) \equiv F[f(-x) \neq q]
\end{array}$$

we compute the components $s_3 = -t - 1 + 2x^6_{a\leq b}$ and $k_3 = 1$. Again, since $k_1 = -1$, we only get one disjunct in the resulting formula which we will call G_6. We compute this formula from the components above and the definition in Section 4.2.2.4:

$$G_6(x) = -\left\lfloor \frac{t+1}{2} \right\rfloor + x \leq 0 \wedge \left(-\left\lfloor \frac{t+1}{2} \right\rfloor + x \geq 0 \rightarrow f\left(\left\lfloor \frac{t+1}{2} \right\rfloor \right) \neq q \right)$$

Note that we simplified the argument of f by removing the double negation that would occur when instantiating $x^6_{b\leq a}$ in $f(-x^6_{b\leq a}) \neq q$ with $-\lfloor \frac{t+1}{2} \rfloor$. The partial tree interpolant computed for the clause $\neg(a \leq b)$ is shown in Figure 4.20.

In the final resolution step, we resolve on $a \leq b$ to generate the empty clause and, thus, refuting the conjunction of the labels of the input interpolation problem. Again, the literal is mixed and we use the instantiation of mixcomb to inequalities. Since $a \leq b$ is only mixed in



7: $\bot$
5: $LA(t+2x^5_{a\le b}, 1, G_5(x^5_{b\le a}))$
6: $LA(-t-1+2x^6_{a\le b}, 1, G_6(-x^6_{b\le a}))$
1: $t \le 2a$
2: $\top$
3: $\top$
4: $2b \le t+1$

Fig. 4.20 Partial tree interpolant for the clause $\neg(a \le b)$.

vertices 5 and 6, the labels for the other vertices a computed as a disjunction (for vertex 7) or a conjunction (for vertices 1 to 4) of the labels of the partial tree interpolants of the antecedents.

For vertex 5, we combine the patterns $LA(2x^5_{a\le b} - s, -1, 2x^5_{a\le b} - s \le 0)$ and $LA(t + 2x^5_{a\le b}, 1, G_5(x^5_{b\le a}))$. From the components

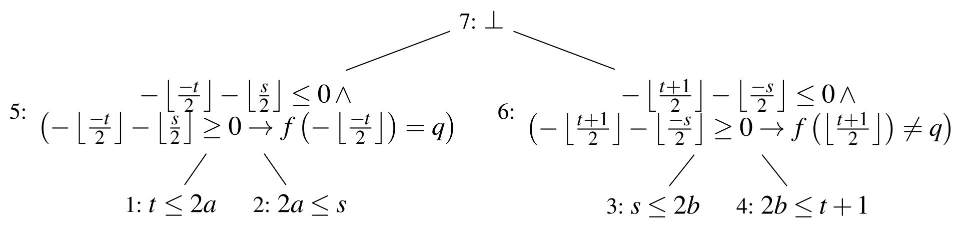
$$\begin{array}{llll} c_1 = 2 & s_1 = -s & k_1 = -1 & F_1(x) \equiv 2x - s \le 0 \\ c_2 = 2 & s_2 = t & k_2 = 1 & F_2(x) \equiv G_5(x) \end{array}$$

we get $s_3 = 2t - 2s$, $k_3 = -1\cdot 2 + 1\cdot 2 + 2\cdot 2 = 4$. Since $k_1 = -1$, we get only one disjunction in the resulting formula. The corresponding instantiation for $x^5_{a\le b}$ is $-\lfloor \frac{s}{2} \rfloor$. The resulting formula can be expressed as $G_5\left(-\lfloor \frac{s}{2} \rfloor\right)$.

For vertex 6, we combine the patterns $LA(s + 2x^6_{a\le b}, -1, s + 2x^6_{a\le b} \le 0)$ and $LA(-t-1+2x^6_{a\le b}, 1, G_6(-x^6_{b\le a}))$. The components

$$\begin{array}{llll} c_1 = 2 & s_1 = s & k_1 = -1 & F_1(x) \equiv s + 2x \le 0 \\ c_2 = 2 & s_2 = -t-1 & k_2 = 1 & F_2(x) \equiv G_6(x) \end{array}$$

give us $s_3 = 2s + 2(-t-1) = 2s - 2t - 2$ and $k_3 = 1\cdot 2 - 1\cdot 2 + 2\cdot 2 = 4$. Again, $k_1 = -1$ leads to only one disjunct in the resulting formula. The corresponding instantiation for $x^6_{a\le b}$ is $-\lfloor \frac{-s}{2} \rfloor$ and the resulting formula is $G_6\left(-\lfloor \frac{-s}{2} \rfloor\right)$.



7: $\bot$
5: $-\lfloor \frac{-t}{2} \rfloor - \lfloor \frac{s}{2} \rfloor \le 0 \wedge$ $(-\lfloor \frac{-t}{2} \rfloor - \lfloor \frac{s}{2} \rfloor \ge 0 \rightarrow f(-\lfloor \frac{-t}{2} \rfloor) = q)$
6: $-\lfloor \frac{t+1}{2} \rfloor - \lfloor \frac{-s}{2} \rfloor \le 0 \wedge$ $(-\lfloor \frac{t+1}{2} \rfloor - \lfloor \frac{-s}{2} \rfloor \ge 0 \rightarrow f(\lfloor \frac{t+1}{2} \rfloor) \neq q)$
1: $t \le 2a$
2: $2a \le s$
3: $s \le 2b$
4: $2b \le t+1$

Fig. 4.21 Tree interpolant for the tree interpolation problem from Example 4.2.

Since we derived the empty clause, no resolution step has to be done on this clause. Thus, the *LA* pattern is no longer needed and we simply expand it to its third component. The resulting interpolant is shown in Figure 4.21.

4.4.2 Correctness of the Generated Interpolant

To show correctness of the generated tree interpolant shown in Figure 4.21, we have to proof that the tree interpolant satisfies the conditions for tree interpolants given in Section 4.1.

Label of the root is $\bot$.

This condition is trivially satisfied by the root (vertex 7) which is labelled with $\bot$.

For all vertices, the interpolant label of the children and the input label of the vertex imply the interpolant label of the vertex.

This condition is trivial for the leaves (vertices 1 to 4) where the interpolant label equals the input label.

Next we consider vertex 5. We have $t \leq 2a \leq s \wedge f(a) = q$ from the interpolant labels of the children and the input label of vertex 5. We first show $-\left\lfloor\frac{-t}{2}\right\rfloor - \left\lfloor\frac{s}{2}\right\rfloor \leq 0$. Using the simplification $-\lfloor -x \rfloor = \lceil x \rceil$, we get $\left\lceil\frac{t}{2}\right\rceil \leq \left\lfloor\frac{s}{2}\right\rfloor$. If $t < s$, this formula holds trivially. If $t = s$, we have to show that t is even. This follows directly from $t \leq 2a \leq s$ and $t = s$. Second, we show $f\left(\left\lceil\frac{t}{2}\right\rceil\right) = q$ if $\left\lceil\frac{t}{2}\right\rceil = \left\lfloor\frac{s}{2}\right\rfloor$. If t is even, there is some i such that $t = 2i$. Then, $\left\lceil\frac{t}{2}\right\rceil = i = \left\lfloor\frac{s}{2}\right\rfloor$ gives us two possible values for s: $2i$ and $2i+1$. In both cases, the only even value between t and s is $2i$. Thus, $i = a$ and $\left\lceil\frac{t}{2}\right\rceil = a$. With $f(a) = q$ we get $f\left(\left\lceil\frac{t}{2}\right\rceil\right) = q$. Finally we notice that $-\left\lfloor\frac{-t}{2}\right\rfloor - \left\lfloor\frac{s}{2}\right\rfloor \leq 0 \wedge \left(-\left\lfloor\frac{-t}{2}\right\rfloor - \left\lfloor\frac{s}{2}\right\rfloor \geq 0 \rightarrow f\left(-\left\lfloor\frac{-t}{2}\right\rfloor\right) = q\right)$ is equivalent to the formula $\left\lceil\frac{t}{2}\right\rceil \leq \left\lfloor\frac{s}{2}\right\rfloor \wedge \left(\left\lceil\frac{t}{2}\right\rceil = \left\lfloor\frac{s}{2}\right\rfloor \rightarrow f\left(\left\lceil\frac{t}{2}\right\rceil\right) = q\right)$ which was used in the argumentation above.

Next we consider vertex 6. The argumentation is similar than in the previous case. From $-\left\lfloor\frac{t+1}{2}\right\rfloor - \left\lfloor\frac{-s}{2}\right\rfloor \leq 0$ we get $\left\lceil\frac{s}{2}\right\rceil \leq \left\lfloor\frac{t+1}{2}\right\rfloor$ which is implied by $s \leq 2b \leq t+1$. If equality holds, we conclude $\left\lfloor\frac{t+1}{2}\right\rfloor = b$. With $f(b) \neq q$ we get $f\left(\left\lfloor\frac{t+1}{2}\right\rfloor\right) \neq q$.

Finally, we consider vertex 7. From $-\left\lfloor\frac{-t}{2}\right\rfloor - \left\lfloor\frac{s}{2}\right\rfloor \leq 0$ we get $\left\lceil\frac{t}{2}\right\rceil \leq \left\lfloor\frac{s}{2}\right\rfloor$ and from $-\left\lfloor\frac{t+1}{2}\right\rfloor - \left\lfloor\frac{-s}{2}\right\rfloor \leq 0$ we get $\left\lceil\frac{s}{2}\right\rceil \leq \left\lfloor\frac{t+1}{2}\right\rfloor$. If t is the even integer $2i$, we have $\left\lceil\frac{t}{2}\right\rceil = \left\lceil\frac{2i}{2}\right\rceil = i$ and $\left\lfloor\frac{t+1}{2}\right\rfloor = \left\lfloor\frac{2i+1}{2}\right\rfloor = i$. Thus, $\left\lceil\frac{t}{2}\right\rceil = \left\lfloor\frac{t+1}{2}\right\rfloor$. If t is the odd integer $2i-1$, we have $\left\lceil\frac{t}{2}\right\rceil = \left\lceil\frac{2i-1}{2}\right\rceil = i = \left\lfloor\frac{2i}{2}\right\rfloor = \left\lfloor\frac{t+1}{2}\right\rfloor$. In both cases we have $\left\lceil\frac{t}{2}\right\rceil = \left\lfloor\frac{t+1}{2}\right\rfloor = i$. Thus, we have $i \leq \left\lfloor\frac{s}{2}\right\rfloor$ and $\left\lceil\frac{s}{2}\right\rceil \leq i$. To satisfy both conditions at the same time, s has to be the even integer $2i$. Thus, we have $-\left\lfloor\frac{-t}{2}\right\rfloor - \left\lfloor\frac{s}{2}\right\rfloor = 0$ and $-\left\lfloor\frac{t+1}{2}\right\rfloor - \left\lfloor\frac{-s}{2}\right\rfloor = 0$. Since $\left\lceil\frac{t}{2}\right\rceil = \left\lfloor\frac{t+1}{2}\right\rfloor$ we derive unsatisfiability from $f\left(-\left\lfloor\frac{-t}{2}\right\rfloor\right) = q$ and $f\left(\left\lfloor\frac{t+1}{2}\right\rfloor\right) \neq q$.

Symbol condition is satisfied.

Figure 4.5 on page 41 shows the symbols present in a subtree. To check the symbol condition, we have to find for every vertex and every symbol occurring in the interpolant label of the vertex a vertex inside the subtree and a vertex outside the subtree such that the symbol occurs in this vertex.

The symbol conditions holds trivially for the root. The symbols occurring in the subtree rooted at vertex 5 and those occurring in the subtree rooted at vertex 6 only differ in a resp. b. Thus, in both vertices, t, s, f, q occur inside and outside of the subtree. The cases for the leaves are trivial to check.

Since all conditions for a tree interpolant are satisfied, the labelling function shown in Figure 4.21 is correct.

Chapter 5
SMTInterpol

The techniques presented in the previous chapters are implemented in our interpolating SMT solver SMTInterpol [21]. SMTInterpol proved competitive to other SMT solvers when solving SMT queries. It regularly participates in the SMT solver competitions SMT-COMP [3, 26]. Various version of SMTInterpol were evaluated during SMT-EVAL 2013 [27]. In 2014, SMTInterpol received a Gödel medal for ranking second in the SMT competition. Since its introduction in 2012, SMTInterpol also participates the application track [14]. This track simulates interaction between the solver and a user by repeatedly posing satisfiability problems to the solver. Unfortunately, this track does not cover interpolation which represents the major usage of SMTInterpol.

SMTInterpol is written in Java. It has a rich API which follows closely the extension to the SMTLIB 2 standard [4]. We restrict the presentation of SMTLIB features in this chapter to those added by SMTInterpol that are described in this thesis. For standard features We refer to the tutorial [25].

5.1 Architecture

In this section, we will shortly explain the different components of SMTInterpol and the techniques implemented by these components.

User Interaction

SMTInterpol supports the SMTLIB [4] script language and provides a Java API modelled after the commands of this language through its `Script` interface. Users can either give commands via an SMTLIB file or the standard input channel of the solver, or use the API.

CNF Conversion

Every asserted formula gets converted into *Conjunctive Normal Form* (CNF), which is a conjunction of disjunctions of literals. SMTInterpol uses a variant of the encoding proposed by Plaisted and Greenbaum [78] to convert a formula into CNF. If production of assignments to named Boolean terms contained in the input formula is enabled, SMTInterpol uses Tseitin encoding [86] for the named term. The term in the input is then replaced by a proxy literal and the corresponding definitions are added to the clause database of the DPLL core. Tseitin encoding ensures that the DPLL core propagates the correct value to the proxy literal.

Preprocessing

Various preprocessing techniques for SMT formulas exist. SMTInterpol only includes some trivial preprocessing rules. These preprocessing rules are designed to remove unnecessary parts of the formula by absorption or idempotency. For integer division and modulo, SMTInterpol creates a new variable representing the result of the division or modulo operation. Then, we add axioms to relate the new variable to the division or the modulo. For example, if $\frac{t}{c}$ for an integer-valued term t and an integral constant c occurs in the input, we generate an auxiliary variable x, replace $\frac{t}{c}$ by x and add the axioms $c \cdot x \leq t$ and $t - |c| < x$. These axioms represent a truncation of the division when considered over the reals.

SMTInterpol does not contain sophisticated preprocessing techniques to handle term if-then-else constructs [61]. Instead, we use an encoding based on a new variable that captures the value of the term if-then-else and suitable axioms.

Note that all these preprocessing steps are applied to one formula at a time. SMTInterpol does not simplify a formula with respect to the previously asserted formulas. We would also need to justify such simplifications when producing interpolants.

DPLL Core

SMTInterpol follows the DPLL($\mathscr{T}$) paradigm [46]. The DPLL engine [31, 30] serves as a truth enumerator and communicates with a set of satellite theories. It is implemented as a custom-build SAT solver that can also handle creation of new literals during solving. The core is able to log resolution proofs.

Satellite Theories

SMTInterpol currently contains two satellite solvers: one for uninterpreted functions and one for linear arithmetic. The solver for the theory of uninterpreted functions is based on congruence closure [34]. The solver for linear arithmetic implements a variant of Simplex [39]. Additionally, it uses the “cuts from proofs” [35] technique to deal with integer or mixed integer problems. Theories are combined using model-based theory combination [32]. Every satellite solver can propagate literals to the DPLL core. The core might ask the satellite solver for an explanation of the propagation in form of a clause. In this case, the satellite solvers provide a clause annotated with enough information to reconstruct the internal proof of unsatisfiability.

The solver for uninterpreted functions provides a trace of the steps used to connect the terms that should not be equal. The trace omits reflexivity and symmetry steps to reduce the overhead of proof generation.

The solver for the theory of linear arithmetic provides Farkas coefficients such that the weighted sum of the negation of the literals in the clause yields a conflict.

Models and Proofs

SMTInterpol can produce models for satisfiable formulas and resolution proofs for unsatisfiable formulas. From these proofs, SMTInterpol can extract unsatisfiable cores or Craig interpolants.

Interpolants

The architecture of the interpolation engine follows roughly the DPLL($\mathcal{T}$) paradigm: A *core interpolator* produces *partial interpolants* for the resolution steps while theory specific interpolators [70, 24] produce partial interpolants for $\mathcal{T}$-lemmas. In the presence of *mixed literals*, i. e., literals that do not occur in any block of the interpolation problem, special *mixed literal interpolators* combine partial interpolants according to the rules presented in the previous chapters.

5.2 Interpolation

To support interpolation, SMTInterpol extends the SMTLIB standard with a non-standard `get-interpolants` command. This command expects as parameters a description of the interpolation problem. To add a formula to an interpolation problem, it has to be *named*, i. e., asserted using the command `(assert (! formula :named Name))`. A binary interpolation problem is the simplest form supported by SMTInterpol. To specify a binary interpolation problem, the names of two formulas have to be given. The first name represents the A part and the second name the B part of the binary interpolation problem. All other parts, if named or not, are considered as extension to the background theory. Thus, all their symbols are assumed shared and might appear in the interpolant. SMTInterpol supports combination of multiple named formulas when specifying a part through the `and` connector.

If more than two parameters are supplied, an inductive sequence of interpolants is computed. Since a query for a sequence of interpolants can be seen as a restricted form of a query for tree interpolants, we encode trees in a similar way than sequences. The following EBNF describes our tree interpolation encoding:

$$\begin{aligned} \mathit{tree} &::= \mathit{symbol} \mid \mathit{subtrees}\,\mathit{symbol} \\ \mathit{subtrees} &::= \mathit{tree} \mid \mathit{tree}(\mathit{subtrees}) \end{aligned}$$

The non-terminal *symbol* matches a name of a named formula or the conjunction of such names. The first rule encodes a tree as either a single symbol (for a single-node tree) or as the encoding of the subtrees of the root vertex followed by the symbol labeling the root vertex. The encoding of the first child requires no parenthesis (thus yielding a simple encoding for a sequence), but its siblings need to be parenthesized to encode the tree structure. Note that the grammar above is LALR(1) and is easy to parse. The formulas appear in the same order as in a post-order traversal of the tree. E. g., to compute the interpolants for the tree depicted in Figure 5.1, the command is

$$\begin{tikzcd}
\phi_1 & \phi_2 \arrow[l] & & \phi_5 \arrow[ll] \arrow[dl] & \phi_6 \arrow[l] \\
& \phi_3 & \phi_4 \arrow[l] & &
\end{tikzcd}$$

Fig. 5.1 Example Formula Tree

```
(get-interpolants phi1 phi2 (phi3 phi4) phi5 phi6)
```

The command can be used after a satisfiability check returned *unsat* and before a command that changes the assertion stack of the solver. Interpolant computation can be redone with a different partition by calling `get-interpolants` again with different arguments.

The solver replies to this command with a parenthesized sequence of interpolants `(I1 I2 ...In-1)` that correspond to the post-order traversal of the tree. The last interpolant, which is the interpolant `false` annotating the root vertex, is omitted from the returned sequence. The tree structure is also omitted from the output. Note that this output unifies binary interpolation, sequence interpolation, and tree interpolation since they can all be expressed as tree interpolation.

Since SMTInterpol extracts interpolants from proof trees, proof production or interpolant production has to be enabled. This is done by setting the options `:produce-proofs` resp. `:produce-interpolants` to `true` before setting the logic. Note however that these options are not synonyms in SMTInterpol. Since we do not simplify input formulas across different assertions, we do not need to track the conversion from an assertion into CNF if we want to compute interpolants. This partial tracking removes some memory overhead. Thus, if only interpolants are desired, enabling only interpolants is cheaper than enabling complete proofs.

5.3 New Literals

As mentioned before, literals that do not occur in the input formulas pose a major difficulty to interpolation. We have several sources for such new literals in SMTInterpol. Some of these sources are due to common solving techniques, others are specific to SMTInterpol.

Theory Combination

SMTInterpol uses a variant of model-based theory combination. Equalities that are deduced by a solver are directly propagated to the other solver. In the case of linear arithmetic, the solver is not always able to detect which equalities to propagate. To overcome this problem, the solver only *suggest* to the DPLL core to decide an equality. This capability is needed especially in the case on non-convex theories like linear integer arithmetic. Note that, while the equalities shared between the theory solvers might not be present in the input, the symbols they equate are already present. Thus, only a new literal built from existing symbols is created.

SMTInterpol uses a special notion of shared terms. Consider the terms x, y, $x+1$, $y+1$, $5x-7$, and $5y-7$. If they occur in the congruence closure graph, they all represent different terms. If, e. g., $x = y$ is detected during solving, the equalities $x+1 = y+1$, and $5x-7 = 5y-7$ should be propagated directly. To solve this problem, SMTInterpol uses *linear offset terms* to reduce the number of variables considered by the linear arithmetic solver. A linear offset term consists of a variable v known to the linear arithmetic solver, a multiplicative constant c, and an offset o and has form $c \cdot v + o$. Then, considering the example terms from above, the linear arithmetic solver only needs to reason about x and y and the equality on the terms $x+1$ and $y+1$ corresponds to the equality $x = y$ for the linear arithmetic solver.

To propagate equalities between the solvers, we use lemmas of the form $a = b \rightarrow c \cdot a + o = c \cdot b + o$ or $c \cdot a + o = c \cdot b + o \rightarrow a = b$ where c and o are constants. The literal $a = b$ is considered by the theory solver for linear arithmetic and $c \cdot a + o = c \cdot b + o$ is a linear offset equality considered by the theory solver for the theory of uninterpreted functions.

Since equalities between shared terms can be mixed, we need specialised interpolation rules for such equalities. Even if one of the literals is interpreted by the linear arithmetic solver, we consider them as equalities in the proof tree and compute interpolants that can be used with the instantiation of mixcomb to uninterpreted functions. Since we have one mixed equality and one mixed disequality in the conflict, i. e., the negated clause, we need interpolants of the form $F[EQ(X_{\neq}, f(x_{=}))](x_{=})$ where $X_{\neq}$ is the auxiliary variable introduced by the projection of the disequality and $x_{=}$ is the variable introduced by projection of the equality. The general form indicates that the EQ term occurs positively in the formula F and the auxiliary variable $x_{=}$ occurs either in the s-part of the EQ term or at arbitrary positions in the remaining formula.

We consider two cases. First, let the linear arithmetic solver propagate an equality to the solver for uninterpreted functions. W. l. o. g. the lemma has form $a = b \rightarrow c \cdot a + o = c \cdot b + o$ where $a = b$ is an equality used by the linear arithmetic solver and $c \cdot a + o = c \cdot b + o$ is an equality used by the solver for uninterpreted functions. The corresponding conflict is $a = b \wedge c \cdot a + o \neq c \cdot b + o$. We project this conflict onto the current part and solve the projection of $a = b$ for the part local to the current part (either a or b). Then, we substitute the generated equation into the projection of the disequality. The generated interpolant has form $EQ(X_{c \cdot a + o \neq c \cdot b + o}, c \cdot c_{a=b} + o)$.

$$\begin{tikzcd}[column sep=small]
 & 3\colon \emptyset \arrow[dl, dash] \arrow[dr, dash] & \\
1\colon \{a\} & & 2\colon \{b\}
\end{tikzcd}$$

Fig. 5.2 Distribution of the relevant symbols for interpolant computation of the theory combination clauses for linear offset terms.

Example 5.1. Consider the clause $a \neq b \vee 5a + 7 = 5b + 7$ with the symbol distribution shown in Figure 5.2. Both literals are mixed in vertices 1 and 2. For vertex 1, we compute the projection of the negated clause as $a = x^1_{a=b} \wedge 5a + 7 \in Y^1_{5a+7 \neq 5b+7}$. We use the first conjunct to eliminate a from the second conjunct and compute the label $5x^1_{a=b} + 7 \in Y^1_{5a+7 \neq 5b+7}$ for vertex 1. Similarly, we compute the label $5x^2_{a=b} + 7 \in Y^2_{5a+7 \neq 5b+7}$ for vertex 2. The label for vertex 3 has to be $\perp$ to satisfy the conditions for partial tree interpolants. This label follows from the labels computed for vertices 1 and 2, and the projection $x^1_{a=b} = x^2_{a=b} \wedge Y^1_{5a+7 \neq 5b+7} \cap Y^2_{5a+7 \neq 5b+7} = \emptyset$ of the negated clause onto vertex 3. ⌟

Second, let the solver for uninterpreted functions propagate a value to the solver for linear arithmetic. W. l. o. g. the lemma has form $c \cdot a + o = c \cdot b + o \rightarrow a = b$. If a resp. b is a rational-valued term, we can solve the projection of the equality for this term and substitute it in the projection of the linear equality. If these terms are integer-valued, we need to make sure that the solution is also divisible without remainder by c. Hence, the resulting interpolant will contain a divisibility predicate. The resulting interpolant has form $EQ(X_{a=b}, \frac{x_{c \cdot a+o=c \cdot b+o} - o}{c})$ if the terms are rational-valued. For integer values, we add the conjunct $c \mid x_{c \cdot a+o=c \cdot b+o} - o$ to ensure that the value of the s term in the EQ form is not truncated.

Example 5.2. Consider the clause $5a+7 \neq 5b+7 \vee a = b$ and the symbol distribution shown in Figure 5.2. Since the terms are integer-valued, we compute for vertices 1 and 2 labels containing divisibility predicates. For vertex 1, we compute $EQ(X^1_{a=b}, x^1_{5a+7=5b+7} - 7) \wedge 5 \mid x^1_{5a+7=5b+7} - 7$. Similarly, we compute $EQ(X^1_{a=b}, x^2_{5a+7=5b+7} - 7) \wedge 5 \mid x^2_{5a+7=5b+7} - 7$ for vertex 2. The label of vertex 3 has to be $\bot$. It also follows form the labels for the vertices 1 and 2, and the projection of the clause onto vertex 3 which is $X^1_{a=b} \cap X^2_{a=b} = \emptyset \wedge x^1_{5a+7=5b+7} = x^2_{5a+7=5b+7}$. ┘

Cut Generation

To solve linear integer arithmetic, SMTInterpol first tries to solve the relaxation of the input problem to linear real arithmetic. If a model is found that contains at least one integer variable that does not have an integral value, SMTInterpol uses *extended branches* [35] to exclude the current model from the relaxation. This technique lends itself to interpolation since the newly created literals don't need to be justified in the proof tree. Essentially, they act like case splits and occur as pivots in some resolution steps.

Note however that the newly created branching literals might be mixed. We do not need to restrict the generation of these branching literals to avoid mixed literals. This is a benefit of our algorithm for proof tree preserving (tree) interpolation and a difference to other interpolating SMT solvers.

Bound Propagation

To solve linear arithmetic, SMTInterpol uses a variant of the Simplex adaptation to SMT [39]. Internally, bound propagation is integrated into the Simplex algorithm in the following two ways.

First, let c be a constant and x be a variable. If the solver already knows $x \neq c$ and the literal $x \leq c$ gets assigned, the bound is refined into $c - \varepsilon$ before continuing with the original algorithm. Since in the case of linear integer arithmetic we have $\varepsilon = 1$, the new bound might be subject to a disequality as well. Thus, in the case of linear integer arithmetic, multiple such refinement steps are possible before the actual bound gets asserted.

Second, let $x_b = \sum_{i \in \mathcal{N}^+} a_{ib} x_i + \sum_{i \in \mathcal{N}^-} a_{ib} x_i$ be a row in the Simplex tableau with $a_{ib} > 0$ for all $i \in \mathcal{N}^+$ and $a_{ib} < 0$ for all $i \in \mathcal{N}^-$. Then, if for all $i \in \mathcal{N}^+$, v_i has an upper bound, and for all $i \in \mathcal{N}^-$, v_i has a lower bound, SMTInterpol computes an upper bound for x_b and similarly for the lower bound. If the computed upper resp. lower bound refines the upper resp. lower bound known for x_b, it gets asserted as a *composite bound*. This bound might not correspond to a literal present in the input, but can be explained from literals in the input. Note that composite bounds might depend upon other composite bounds.

When explaining these newly created bounds, we sometimes prefer to create new literals to shorten the proof tree. These new literals, however, are built from an existing variable and a bound. Thus, these literals are mixed if and only if the variable is mixed[1].

5.4 Proof of Concept

SMTInterpol is used as a proof of concept implementation of the algorithms described in the Chapters 3 and 4. It is implemented in the programming language Java and distributed under LGPL version 3 or later. These conditions make SMTInterpol a platform-independent, free to use candidate for both SMT solving and interpolation. The solver can be downloaded from

`http://ultimate.informatik.uni-freiburg.de/smtinterpol/`

and its sources can be found at

`https://github.com/juergenchrist/smtinterpol/`

[1] Note that in the context of Simplex for SMT, a variable might actually represent a sum of variables.

Chapter 6
Future Work

There are many possible directions for future work on interpolation in the context of SMT. We classify them in three parts.

6.1 Theory Specific Interpolation

The algorithm presented in this thesis is parameterised by a combination function depending on a theory. We showed how to instantiate this function for the theories of uninterpreted functions and linear arithmetic. Further theories like the theory of arrays [13, 85], the theory of bit-vectors [48], or the theory of non-linear arithmetic, to just name a few, are interesting candidates for future work. Existing techniques in these fields do not integrate theory-specific interpolation into an interpolation algorithm that supports other theories. Instead, they either build interpolating rules from scratch or reduce the interpolation problem to an interpolation problem in a theory where interpolation is well studied. Integration of such theories into the algorithm for proof tree preserving interpolation from Chapter 3 and the algorithm for proof tree preserving tree interpolation form Chapter 4 will allow us to compute interpolants for almost all verification conditions produced by interpolation-based software model checkers.

6.2 Quantifiers

Interpolation in the context of quantifiers was investigated in several papers [71, 73, 63, 19]. These methods generate quantified interpolants where quantifiers are introduced to satisfy the symbol condition of the definition of Craig interpolants. A notable exception is the work by Jhala and McMillan [60]. They generate quantified array interpolants that describe common loop invariants.

An interesting direction of future work will be to integrate quantifier reasoning into the algorithms. To generate interpolants that contain quantifiers and, at the same time, are useful in the context of interpolation-based software model checking, new solving techniques are needed to produce lemmas for which quantified (partial) interpolants can be computed. The quantifiers should not just be introduced to satisfy the symbol condition. Instead, quantifiers should be used to express some property that is likely to be useful in the context of software model checking.

6.3 Strength Variation

Given an input problem to interpolation, the resulting interpolant is not unique. In fact, different interpolants for the same problem exist. These interpolants differ in logic strength. The desired strength depends on the usage scenario of the interpolant. We identify two possible directions.

First, we can try to find *the right proof*. At the moment, the first proof generated by the SMT solver is used to compute an interpolant. In some cases, however, this proof might use an argument that is not good for the current usage scenario. In interpolation-based software model checking, for example, a proof is a certificate that a given error path is spurious, i. e., not possible in the not concrete system. A model checker tries to create an invariant from this certificate by interpolation. Consider a finite unrolling of a loop that is part of the error path. If the certificate argues that it is impossible to exit the loop after only this finite number of unrollings, the generated interpolant will not generalise to a loop invariant that summarises all loop iterations. Thus, a different proof might be desirable in this scenario. Iteration of proofs can be achieved by means of iteration of unsatisfiable cores [79, 66].

Second, even if we have the right proof, we still need to extract *the right interpolant*. Strength modification can be done in various ways. D'Silva et al. [38] present a method to alter the strength of interpolants generated from resolution refutations generated by a SAT solver. In binary interpolation, if a literal occurs in A and B, we can choose to put it either in A, or in B, or in both. This leads to different cases in the interpolation algorithm when this literal acts as the pivot of a resolution step. Exposing the capabilities offered by this method to users, however, is non-trivial. Cimatti et al. [24] present a method to alter the strength of the partial interpolants generated for lemmas in the theory of linear arithmetic. The idea is to strengthen the partial interpolant by not including shared literals. This method can be exploited to generate different sequence interpolants and tree interpolants. The theory of linear arithmetic, for example, computes partial interpolants by summing up parts of a proof. The proofs generated by SMT solver, however, might have some slack, i. e., instead of producing the proof $1 \leq 0$, they might produce the proof $c \leq 0$ for $c > 1$. In this case, we have slack that can be added to the summary while preserving inconsistency with the part that is not summed up. This technique is interesting in the case of interpolation-based software model checking since the resulting interpolant should not only satisfy the conditions of sequence or tree interpolants. Some labels of the resulting interpolant should *cover* [72] other labels. Covering expresses that the set of models of the first formula is a superset of the set of models of the second formula. If the formulas computed at the loop entry point at different iterations of the loop cover, an inductive invariant for the loop is found. Hence, improving the covering relation improves the accuracy of the software model checker.

A different direction of research regarding strength variation is the effect of *proof manipulation*. SMTInterpol already contains some proof manipulation techniques [44]. One possible future research is to investigate the effect of various proof manipulations [28, 81, 9] on the strength of the generated interpolant.

Chapter 7
Conclusion

In this thesis, we presented proof tree preserving interpolation and extended it to tree interpolation. The algorithm tackles the problem of interpolation in the context of proofs generated by state of the art SMT solvers. It builds upon existing algorithm for interpolation in propositional logic and adds specialised methods to lift these algorithm to the context of SMT. Special care has to be taken for mixed literals, i. e., literals that do not occur in the input but are created during solving on various occasions. Contrary to other techniques, proof tree preserving interpolation deals with these literals without restricting the solver or manipulating the proof tree. We showed correctness of the techniques. Since we do not restrict the solver, completeness of the technique follows from completeness of the solver. In other words, for every proof generated by an SMT solver, proof tree preserving interpolation can compute an interpolant.

The contributions of this thesis can be summarised as follows.

- We presented proof tree preserving interpolation. We instantiated the algorithm for the theories of uninterpreted functions and the theories of linear arithmetic over integers or reals. We showed how to combine these theories to compute quantifier-free interpolants for formulas that are difficult for other interpolation techniques.
- We extended the algorithm to tree interpolation. This enables us to compute tree interpolants (and, thus, sequence interpolants) from one proof tree without the need to solve repeated binary interpolation problems or repeatedly manipulate a proof. Again, we instantiated this technique to the theories of uninterpreted functions and linear arithmetic of integers or reals.
- We present our proof of concept implementation SMTInterpol which is an open-source SMT solver that can compute binary interpolants and tree interpolants using the algorithms developed for proof tree preserving (tree) interpolation.

References

1. iZ3 documentation. `http://research.microsoft.com/en-us/um/redmond/projects/z3/old/iz3documentation.html`. Accessed: 2012-10-05.
2. A. Albarghouthi and K. L. McMillan. Beautiful interpolants. In *CAV*, pages 313–329. Springer, 2013.
3. C. Barrett, M. Deters, L. de Moura, A. Oliveras, and A. Stump. 6 years of SMT-COMP. *J. Autom. Reasoning*, 50(3):243–277, 2013.
4. C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB Standard: 2.0. In *SMT*, 2010.
5. C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*, pages 825–885. IOS Press, 2009.
6. D. Beyer and M. E. Keremoglu. CPAchecker: A tool for configurable software verification. In *CAV*, pages 184–190. Springer, 2011.
7. D. Beyer, D. Zufferey, and R. Majumdar. CSIsat: Interpolation for LA+EUF. In *CAV*, pages 304–308. Springer, 2008.
8. A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009.
9. J. Boudou and B. W. Paleo. Compression of propositional resolution proofs by lowering subproofs. In *Automated Reasoning with Analytic Tableaux and Related Methods - 22th International Conference, TABLEAUX 2013, Nancy, France, September 16-19, 2013. Proceedings*, pages 59–73. Springer, 2013.
10. A. Brillout, D. Kroening, P. Rümmer, and T. Wahl. Beyond quantifier-free interpolation in extensions of Presburger arithmetic. In *VMCAI*, pages 88–102. Springer, 2011.
11. R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, A. Santuari, and R. Sebastiani. To ackermann-ize or not to ackermann-ize? On efficiently handling uninterpreted function symbols in $SMT(EUF \cup T)$. In *LPAR*, pages 557–571. Springer, 2006.
12. R. Bruttomesso, S. Ghilardi, and S. Ranise. From strong amalgamability to modularity of quantifier-free interpolation. In *IJCAR*, pages 118–133. Springer, 2012.
13. R. Bruttomesso, S. Ghilardi, and S. Ranise. Quantifier-free interpolation of a theory of arrays. *Logical Methods in Computer Science*, 8(2), 2012.
14. R. Bruttomesso and A. Griggio. Broadening the scope of SMT-COMP: the application track. In *COMPARE*, pages 18–27. CEUR-WS.org, 2012.
15. R. Bruttomesso, S. Rollini, N. Sharygina, and A. Tsitovich. Flexible interpolation with local proof transformations. In *ICCAD*, pages 770–777. IEEE, 2010.
16. F. Cassez, C. Müller, and K. Burnett. Summary-based inter-procedural analysis via modular trace refinement. In *FSTTCS*, pages 545–556. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.
17. J. Christ, E. Ermis, M. Schäf, and T. Wies. Flow-sensitive fault localization. In *VMCAI*, pages 189–208. Springer, 2013.
18. J. Christ and J. Hoenicke. Proof tree preserving tree interpolation. *J. Autom. Reasoning*. submitted.
19. J. Christ and J. Hoenicke. Instantiation-based interpolation for quantied formulae. In *SMT 2010: 8th International Workshop on Satisfiability Modulo Theories*, 2010.
20. J. Christ and J. Hoenicke. Extending proof tree preserving interpolation to sequences and trees. In *SMT*, 2013.
21. J. Christ, J. Hoenicke, and A. Nutz. SMTInterpol: An interpolating SMT solver. In *SPIN*, pages 248–254. Springer, 2012.
22. J. Christ, J. Hoenicke, and A. Nutz. Proof tree preserving interpolation. In *TACAS*, pages 124–138. Springer, 2013.
23. J. Christ, J. Hoenicke, and A. Nutz. Proof tree preserving interpolation. Reports of SFB/TR 14 AVACS 89, SFB/TR 14 AVACS, February 2013. ISSN: 1860-9821, `http://www.avacs.org`.
24. A. Cimatti, A. Griggio, and R. Sebastiani. Efficient interpolant generation in satisfiability modulo theories. In *TACAS*, pages 397–412. Springer, 2008.
25. D. R. Cok. jSMTLIB: Tutorial, validation and adapter tools for SMT-LIBv2. In *NASA Formal Methods*, pages 480–486. Springer, 2011.
26. D. R. Cok, A. Griggio, R. Bruttomesso, and M. Deters. The 2012 SMT competition. In *SMT*, pages 131–142. EasyChair, 2012.
27. D. R. Cok, A. Stump, and T. Weber. The 2013 SMT evaluation. Technical Report 2014-017, Department of Information Technology, Uppsala University, July 2014.

28. S. Cotton. Two techniques for minimizing resolution proofs. In *Theory and Applications of Satisfiability Testing - SAT 2010, 13th International Conference, SAT 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, pages 306–312. Springer, 2010.
29. W. Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *J. Symb. Log.*, 22(3):269–285, 1957.
30. M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.
31. M. Davis and H. Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.
32. L. de Moura and N. Bjørner. Model-based theory combination. *Electr. Notes Theor. Comput. Sci.*, 198(2):37–49, 2008.
33. L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *TACAS*, pages 337–340. Springer, 2008.
34. D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: A theorem prover for program checking. *J. ACM*, 52(3):365–473, 2005.
35. I. Dillig, T. Dillig, and A. Aiken. Cuts from proofs: A complete and practical technique for solving linear inequalities over integers. In *CAV*, pages 233–247. Springer, 2009.
36. K. Dräger, A. Kupriyanov, B. Finkbeiner, and H. Wehrheim. SLAB: A certifying model checker for infinite-state concurrent systems. In *TACAS*, pages 271–274. Springer, 2010.
37. V. D'Silva, D. Kroening, M. Purandare, and G. Weissenbacher. Interpolant strength. In *VMCAI*, pages 129–145. Springer, 2010.
38. V. D'Silva, D. Kroening, M. Purandare, and G. Weissenbacher. Interpolant strength. In *VMCAI*, pages 129–145. Springer, 2010.
39. B. Dutertre and L. de Moura. A fast linear-arithmetic solver for DPLL(T). In *CAV*, pages 81–94. Springer, 2006.
40. E. Ermis, J. Hoenicke, and A. Podelski. Splitting via interpolants. In *VMCAI*, pages 186–201. Springer, 2012.
41. E. Ermis, A. Nutz, D. Dietsch, J. Hoenicke, and A. Podelski. Ultimate kojak - (competition contribution). In *TACAS*, pages 421–423. Springer, 2014.
42. E. Ermis, M. Schäf, and T. Wies. Error invariants. In *FM*, pages 187–201. Springer, 2012.
43. A. Farzan, Z. Kincaid, and A. Podelski. Inductive data flow graphs. In *POPL*, pages 129–142. ACM, 2013.
44. P. Fontaine, S. Merz, and B. W. Paleo. Compression of propositional resolution proofs via partial regularization. In *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, pages 237–251. Springer, 2011.
45. A. Fuchs, A. Goel, J. Grundy, S. Krstic, and C. Tinelli. Ground interpolation for the theory of equality. In *TACAS*, pages 413–427. Springer, 2009.
46. H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In *CAV*, pages 175–188. Springer, 2004.
47. A. Goel, S. Krstic, and C. Tinelli. Ground interpolation for combined theories. In *CADE*, pages 183–198. Springer, 2009.
48. A. Griggio. Effective word-level interpolation for software verification. In *FMCAD*, pages 28–36. FMCAD Inc., 2011.
49. A. Griggio. A practical approach to satisability modulo linear integer arithmetic. *JSAT*, 8(1/2):1–27, 2012.
50. A. Griggio, T. T. H. Le, and R. Sebastiani. Efficient interpolant generation in satisfiability modulo linear integer arithmetic. In *TACAS*, pages 143–157. Springer, 2011.
51. A. Gupta, C. Popeea, and A. Rybalchenko. Solving recursion-free horn clauses over LI+UIF. In *APLAS*, pages 188–203. Springer, 2011.
52. M. Heizmann, J. Christ, D. Dietsch, E. Ermis, J. Hoenicke, M. Lindenmann, A. Nutz, C. Schilling, and A. Podelski. Ultimate automizer with smtinterpol - (competition contribution). In *TACAS*, pages 641–643. Springer, 2013.
53. M. Heizmann, J. Hoenicke, and A. Podelski. Refinement of trace abstraction. In *SAS*, pages 69–85. Springer, 2009.
54. M. Heizmann, J. Hoenicke, and A. Podelski. Nested interpolants. In *POPL*, pages 471–482. ACM, 2010.
55. J. Henry, M. Asavoae, D. Monniaux, and C. Maiza. How to compute worst-case execution time by optimization modulo theory and a clever encoding of program semantics. In *LCTES*, pages 43–52. ACM, 2014.

56. T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan. Abstractions from proofs. In *POPL*, pages 232–244. ACM, 2004.
57. T. S. Hoang, S. Itoh, K. Oyama, K. Miyazaki, H. Kuruma, and N. Sato. Validating the consistency of specification rules. Technical report, Yokohama Research Laboratory, Hitachi Ltd., 2015.
58. G. Huang. Constructing craig interpolation formulas. In *COCOON*, pages 181–190. Springer, 1995.
59. H. Jain, E. M. Clarke, and O. Grumberg. Efficient craig interpolation for linear diophantine (dis)equations and linear modular equations. *Formal Methods in System Design*, 35(1):6–39, 2009.
60. R. Jhala and K. L. McMillan. Array abstractions from proofs. In *CAV*, pages 193–206. Springer, 2007.
61. H. Kim, F. Somenzi, and H. Jin. Efficient term-ite conversion for satisfiability modulo theories. In *SAT*, pages 195–208. Springer, 2009.
62. T. King, C. Barrett, and B. Dutertre. Simplex with sum of infeasibilities for SMT. In *FMCAD*, pages 189–196. IEEE, 2013.
63. L. Kovács and A. Voronkov. Interpolation and symbol elimination. In *CADE*, pages 199–213. Springer, 2009.
64. D. Kroening, J. Leroux, and P. Rümmer. Interpolating quantifier-free Presburger arithmetic. In *LPAR*, pages 489–503. Springer, 2010.
65. J. Kühn, P. Schoonbrood, A. Stollenwerk, C. Brendle, N. Wardeh, M. Walter, R. Rossaint, S. Leonhardt, S. Kowalewski, and R. Kopp. Safety conflict analysis in medical cyber-physical systems using an smt-solver. In *Gemeinsamer Tagungsband der Workshops der Tagung Software Engineering 2015, Dresden, Germany, 17.-18. März 2015.*, pages 19–23. CEUR-WS.org, 2015.
66. M. H. Liffiton and A. Malik. Enumerating infeasibility: Finding multiple muses quickly. In *CPAIOR*, pages 160–175. Springer, 2013.
67. R. Loos and V. Weispfenning. Applying linear quantifier elimination. *Comput. J.*, 36(5):450–462, 1993.
68. C. Lynch and Y. Tang. Interpolants for linear arithmetic in SMT. In *ATVA*, pages 156–170. Springer, 2008.
69. K. L. McMillan. Interpolation and SAT-based model checking. In *CAV*, pages 1–13. Springer, 2003.
70. K. L. McMillan. An interpolating theorem prover. In *TACAS*, pages 16–30. Springer, 2004.
71. K. L. McMillan. An interpolating theorem prover. *Theor. Comput. Sci.*, 345(1):101–121, 2005.
72. K. L. McMillan. Lazy abstraction with interpolants. In *CAV*, pages 123–136. Springer, 2006.
73. K. L. McMillan. Quantified invariant generation using an interpolating saturation prover. In *TACAS*, pages 413–427. Springer, 2008.
74. K. L. McMillan. Interpolants from Z3 proofs. In *FMCAD*, pages 19–27. FMCAD Inc., 2011.
75. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. Program. Lang. Syst.*, 1(2):245–257, 1979.
76. G. Nelson and D. C. Oppen. Fast decision procedures based on congruence closure. *J. ACM*, 27(2):356–364, 1980.
77. R. Nieuwenhuis and A. Oliveras. Proof-producing congruence closure. In *RTA*, pages 453–468. Springer, 2005.
78. D. A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *J. Symb. Comput.*, 2(3):293–304, 1986.
79. A. Previti and J. Marques-Silva. Partial MUS enumeration. In *AAAI*. AAAI Press, 2013.
80. P. Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *J. Symb. Log.*, 62(3):981–998, 1997.
81. S. Rollini, R. Bruttomesso, and N. Sharygina. An efficient and flexible approach to resolution proof reduction. In *Hardware and Software: Verification and Testing - 6th International Haifa Verification Conference, HVC 2010, Haifa, Israel, October 4-7, 2010. Revised Selected Papers*, pages 182–196. Springer, 2010.
82. A. Rybalchenko and V. Sofronie-Stokkermans. Constraint solving for interpolation. In *VMCAI*, pages 346–362. Springer, 2007.
83. O. Sery, G. Fedyukovich, and N. Sharygina. Incremental upgrade checking by means of interpolation-based function summaries. In *FMCAD*, pages 114–121. IEEE, 2012.
84. V. Sofronie-Stokkermans. Hierarchic reasoning in local theory extensions. In *CADE*, pages 219–234. Springer, 2005.
85. N. Totla and T. Wies. Complete instantiation-based interpolation. In *POPL*, pages 537–548. ACM, 2013.
86. G. S. Tseitin. On the complexity of derivation in propositional calculus. In *Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970*, pages 466–483. Springer, 1983.

87. S. Vijzelaar, K. Verstoep, W. Fokkink, and H. Bal. Bonsai: Cutting models down to size. In *PSI*. Springer, 2014.
88. G. Yorsh and M. Musuvathi. A combination method for generating interpolants. In *CADE*, pages 353–368. Springer, 2005.