Proof Tree Preserving Interpolation*

Jürgen Christ, Jochen Hoenicke, and Alexander Nutz

Chair of Software Engineering, University of Freiburg

Abstract. Craig interpolation in SMT is difficult because, e. g., theory combination and integer cuts introduce mixed literals, i. e., literals containing local symbols from both input formulae. In this paper, we present a scheme to compute Craig interpolants in the presence of mixed literals. Contrary to existing approaches, this scheme neither limits the inferences done by the SMT solver, nor does it transform the proof tree before extracting interpolants. Our scheme works for the combination of uninterpreted functions and linear arithmetic but is extendable to other theories. The scheme is implemented in the interpolating SMT solver SMTInterpol.

1 Introduction

A Craig interpolant for a pair of formulae A and B whose conjunction is unsatisfiable is a formula I that follows from A and whose conjunction with B is unsatisfiable. Furthermore, I only contains symbols common to A and B. Model checking and state space abstraction [13,15] make intensive use of interpolation to achieve a higher degree of automation. This increase in automation stems from the ability to fully automatically generate interpolants from proofs produced by modern theorem provers.

For propositional logic, a SAT solver typically produces resolution-based proofs that show the unsatisfiability of an error path. Extracting Craig interpolants from such proofs is a well understood and easy task that can be accomplished, e. g., using the algorithms of Pudlák [19] or McMillan [14]. 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 does not hold for proofs produced by SMT solvers for formulae in a combination of first order theories. Such solvers produce new literals for different reasons. First, to combine two theory solvers, SMT solvers exchange (dis-)equalities between the symbols common to these two theories in a Nelson-Oppen-style theory combination. 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

* This work is supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR14 AVACS).

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* [7] 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.

In this paper, we present a scheme to compute Craig interpolants in the presence of mixed literals. Our interpolation scheme is based on syntactical restrictions of *partial interpolants* and specialized rules to interpolate resolution steps on mixed literals. This enables us to compute interpolants in the context of a state-of-the-art SMT solver without manipulating the proof tree or restricting the solver in any way. We base our presentation on the quantifier-free fragment of the combined theory of uninterpreted functions and linear arithmetic over the rationals or the integers. The interpolation scheme is used in the interpolating SMT solver SMTInterpol [4]. Proofs for the theorems in this paper are given in the technical report [3].

Related Work. For Boolean circuits, Pudlák [19] shows how to construct quantifier-free interpolants from resolution proofs of unsatisfiability. A different proof-based interpolation system is given by McMillan [14] 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. Interpolants are computed from partial interpolants by annotating every proof step. The partial interpolants 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.

Brillout et al. [1] 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 quantifier since they do not use divisibility predicates. Furthermore their method limits the generation of Gomory cuts in the integer solver to prevent some mixed cuts. The method presented in this paper combines the two theories without quantifiers and, furthermore, does not restrict any component of the solver.

Yorsh and Musuvathi [20] show how to combine interpolants generated by an SMT solver based on Nelson-Oppen combination. They define the concept of *equality-interpolating theories*. These are theories that can 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. We do not

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

explicitly split the proof. Additionally, our method can handle the theory of linear integer arithmetic without any restriction on the solver.

Cimatti et al. [5] present a method to compute interpolants for linear rational arithmetic and difference logic. The method presented in this paper 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.

Goel et al. [11] present a generalization of equality-interpolating theories. They define the class of *almost-colorable proofs* and an algorithm to generate interpolants from such proofs. Furthermore they describe a restricted DPLL system to generate almost-colorable 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.

Recently, techniques to transform proofs gained a lot of attention. Bruttomesso et al. [2] 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. 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.

McMillan [16] presents a technique to compute interpolants from Z3 proofs. Whenever a sub-proof contains mixed literals, he extracts lemmas from the proof tree and delegates them to a second (possibly slower) interpolating solver.

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 $\mathcal{EUF}$ and the theories of linear arithmetic on rationals $\mathcal{LA}(\mathbb{Q})$ and integers $\mathcal{LA}(\mathbb{Z})$. The quantifier-free fragment of $\mathcal{LA}(\mathbb{Z})$ is not closed under interpolation. Therefore, we augment the signature with division by constant functions $\lfloor \frac{\cdot}{k} \rfloor$ for all integers $k \geq 1$.

We use the standard notations $\models_T, \perp, \top$ 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 $\mathcal{EUF}$, $\mathcal{LA}(\mathbb{Q})$, and $\mathcal{LA}(\mathbb{Z})$.

The literals in $\mathcal{LA}(\mathbb{Z})$ are of the form $s \leq c$, where c is an integer constant and s a linear combination of variables. For $\mathcal{LA}(\mathbb{Q})$ 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., $x \leq y$ resp. $x > y$ to denote $x - y \leq 0$ resp. $y - x \leq -\varepsilon$. In the integer case we use $x > y$ to denote $y - x \leq -1$.

Our algorithm operates on a proof of unsatisfiability generated by an SMT solver based on DPLL(T) [18]. Such a proof is a resolution tree with the $\bot$-clause at its root. The leaves of the tree are either clauses from the input formulae[2] or theory lemmas that are produced by one of the theory solvers. The negation of a theory lemma is called a *conflict*.

The theory solvers for $\mathcal{EUF}$, $\mathcal{LA}(\mathbb{Q})$, and $\mathcal{LA}(\mathbb{Z})$ are working independently and exchange (dis-)equality literals through the DPLL engine in a Nelson-Oppen style [17]. Internally, the solver for linear arithmetic uses only inequalities in theory conflicts. In the proof tree, the (dis-)equalities are related to inequalities by the (valid) clauses $x = y \lor x < y \lor x > y$, and $x \neq y \lor x \leq y$. We call these leaves of the proof tree *theory combination clauses.*

Interpolants and Symbol Sets. For a formula F, we use $symb(F)$ to denote the set of non-theory symbols occurring in F. An interpolation problem is given by two formulae A and B such that $A \land B \models \bot$. An interpolant of A and B is a formula I such that (i) $A \models I$, (ii) $B \land I \models \bot$, and (iii) $symb(I) \subseteq symb(A) \cap symb(B)$.

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.

Substitution in Formulae and Monotonicity. By $F[G]$ we denote a formula in negation normal form with a sub-formula G that occurs positively in the formula. Substituting this sub-formula by a formula G' is denoted by $F[G']$. 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 of our interpolation scheme.

Lemma 1 (Monotonicity). *Given a formula $F[G_1][G_2]\ldots[G_n]$ in negation normal form with sub-formulae $G_1, G_2, \ldots, G_n$ occurring only positively in the formula and formulae $G'_1, \ldots, G'_n$, it holds that*

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

3 Proof Tree-Based Interpolation

Interpolants can be computed from proofs of unsatisfiability as Pudlák and McMillan 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.

[2] W. l. o. g. we assume input formulae are in conjunctive normal form.

3.1 Pudlák's and McMillan's Interpolation Algorithms

Pudlák's and McMillan's algorithms assume that the 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 $\cdot \downharpoonleft A$ and $\cdot \downharpoonleft B$ as defined below. They have the properties (i) $\mathit{symb}(\ell \downharpoonleft A) \subseteq \mathit{symb}(A)$, (ii) $\mathit{symb}(\ell \downharpoonleft B) \subseteq \mathit{symb}(B)$, and (iii) $\ell \iff (\ell \downharpoonleft A \wedge \ell \downharpoonleft B)$. Other projection functions are possible and this allows for varying the strength of the resulting interpolant as shown in [8]. We extend the projection function to conjunctions of literals component-wise.

	Pudlák		McMillan	
	$\ell \downharpoonleft A$	$\ell \downharpoonleft B$	$\ell \downharpoonleft A$	$\ell \downharpoonleft B$
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 $A \wedge (\neg C \downharpoonleft A)$ and $B \wedge (\neg C \downharpoonleft B)$[3]. 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 $\neg C \downharpoonleft A$ and $\neg C \downharpoonleft B$. 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 $\neg C \downharpoonleft A$ 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.

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

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 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 $C_1 \vee C_2$.*

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

3.2 Purification 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 in the previous section. Therefore, we relax the conditions by allowing fresh auxiliary variables to occur in the projections.

In our setting there are 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 or Ackermannization. Second, inequalities of the form $a+b \leq c$ are introduced, e. g., by extended branches [7] 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.

We split mixed literals using auxiliary variables, which we denote by x, x_a, or x_b in the following. 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 variables 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 $\ell \Leftrightarrow \exists x, x_a, x_b.(\ell \downharpoonleft A) \wedge (\ell \downharpoonleft B)$. We need two variables x_a and x_b to split the literal $a \neq b$ into two symmetric parts. For symmetry we split the literal $a = b$ in the same fashion instead of introducing only a single auxiliary variable. This is achieved by the definitions below.

$$\begin{aligned}
(a = b) \downharpoonleft A &:= (a = x_a \wedge x_a = x_b) & (a = b) \downharpoonleft B &:= (x_a = x_b \wedge x_b = b)\\
(a \neq b) \downharpoonleft A &:= (a = x_a \wedge x_a \neq x_b) & (a \neq b) \downharpoonleft B &:= (x_a \neq x_b \wedge x_b = b)\\
(a + b \leq c) \downharpoonleft A &:= (a + x \leq 0) & (a + b \leq c) \downharpoonleft B &:= (-x + b \leq c)
\end{aligned}$$

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.

Instead of with a single partial interpolant, we label each clause with a pattern from which we can derive two partial interpolants, a strong and a weak one. The strong interpolant of a clause C implies the weak interpolant under the assumption that $\neg C \downharpoonleft A$ or $\neg C \downharpoonleft B$ holds. Having two interpolants enables us to complete the inductive proof. We show that the strong interpolant follows from the A-part of the resolvent if the strong interpolants of the premises follow from their respective A-part. On the other hand, the weak interpolant is in contradiction to the B-part in the resolvent if this is the case for the premises. Since the weak interpolant follows from the strong interpolant this shows that both are partial interpolants. The models for the strong and the weak interpolants only differ in the values of the auxiliary variable. The interpolants are needed because the "right" value for the auxiliary variable is not known when interpolating the leaves of the proof tree. The strong and the weak interpolant are identical if the clause does not contain mixed literals. Therefore, we derive only one interpolant for the bottom clause.

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.

4 Uninterpreted Functions

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

4.1 Leaf Interpolation

The $\mathcal{EUF}$ solver is based on the congruence closure algorithm [6]. 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 the partial interpolants of the theory lemmas, we internally split the mixed literals according to Section 3.2. Then we use an algorithm similar to [10] 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. The disequality is split into $a = x_a$, $x_a \neq x_b$ and $x_b = b$ and the resulting interpolant depends on whether we consider the disequality to belong to the A-part or to the B-part. If we consider it to belong to the B-part, then x_a is the end of an equality path summing up the equalities from A. Thus, the computed interpolant has the form $I[x_a = s]$. If we consider $x_a \neq x_b$ to belong to the A-part, the resulting interpolant is $I[x_b \neq s]$. Note that in both cases the literal $x_a = s$ resp. $x_b \neq s$ occurs positively in the interpolant and is the only literal containing x_a resp. x_b. To summarise, the partial interpolant computed for a theory clause $C \vee a = b$ where $a = b$ has the auxiliary variables x_a, x_b has the form $I[x_a = s]$ or $I[x_b \neq s]$ and x_a, x_b do not appear at any other place in I. Both interpolants $I[x_a = s]$ and $I[x_b \neq s]$ are partial interpolants of the clause. From $x_a \neq x_b$ we can derive the weak interpolant $I[x_b \neq s]$ from the strong interpolant $I[x_a = s]$ using Lemma 1 (monotonicity). We define

$$EQ_S(x, s) := (x_a = s), \qquad EQ_W(x, s) := (x_b \neq s)$$

and label a clause in the proof tree with $I[EQ(x,s)]$ to denote that the formulae $I[EQ_S(x,s)]$ and $I[EQ_W(x,s)]$ are the strong and weak partial interpolants.

For theory lemmas containing the literal $a \neq b$, the corresponding auxiliary variables x_a, x_b may appear anywhere in the partial interpolant, even under a function symbol. A simple example is the theory conflict $s \neq f(a) \wedge a = (x_a = x_b =)b \wedge f(b) = s$, which has the partial interpolants $s \neq f(x_a)$ and $s \neq f(x_b)$ (depending on whether $x_a = x_b$ is considered as A- or as B-literal). We simply label the corresponding theory lemma with the interpolant $s \neq f(x)$. In general the label of such a clause has the form $I(x)$. The formulae $I(x_a)$ and $I(x_b)$ are the strong and weak partial interpolants of that clause. Of course, here the interpolants are equivalent given $x_a = x_b$.

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 $EQ(x,s_1)$ and $EQ(x,s_2)$ for different shared terms s_1, s_2. In general, we allow the partial interpolants to have the form $I[EQ(x,s_1)]\dots[EQ(x,s_n)]$.

4.2 Pivoting of Mixed Equalities

We require that every clause containing $a = b$ with auxiliary variables x_a, x_b is always labelled with a formula of the form $I[EQ(x,s_1)]\dots[EQ(x,s_n)]$ and that this is a partial interpolant of the clause for both EQ_S and EQ_W. As discussed above, this is automatically the case for the theory lemmas computed from conflicts in the congruence closure algorithm. This property is also preserved by (rule-res) and this rule also preserves the property of being a strong or weak partial interpolant.

On the other hand, a clause containing the literal $a \neq b$ is labelled with a formula of the form $I(x)$, i. e., the auxiliary variable x can occur at arbitrary positions. Both $I(x_a)$ and $I(x_b)$ are partial interpolants of the clause. Again, the form $I(x)$ 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 $a = b$.

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

The rule replaces every literal $EQ(x,s_i)$ in I_1 with the formula $I_2(s_i)$, in which every x is substituted by s_i. Therefore the auxiliary variable introduced for the mixed literal $a = b$ is removed.

Theorem 2 (Soundness of (rule-eq)). *Let $a = b$ be a mixed literal with auxiliary variable x. If $I_1[EQ(x,s_1)]\dots[EQ(x,s_n)]$ yields two (strong and weak) partial interpolants of $C_1 \vee a = b$ and $I_2(x)$ two partial interpolants of $C_1 \vee a \neq b$ then $I_1[I_2(s_1)]\dots[I_2(s_n)]$ yields two partial interpolants of the clause $C_1 \vee C_2$.*

4.3 Example

We demonstrate our algorithm on the following example:

$$A \equiv (\neg p \vee a = s_1) \wedge (p \vee a = s_2) \wedge f(a) = t$$
$$B \equiv (\neg p \vee b = s_1) \wedge (p \vee b = s_2) \wedge f(b) \neq t$$

The conjunction $A \wedge B$ is unsatisfiable. In this example, a is A-local, b is B-local and the remaining symbols are shared.

Assume the theory solver for $\mathcal{EUF}$ introduces the mixed literal $a = b$ and provides the lemmas (i) $f(a) \neq t \vee a \neq b \vee f(b) = t$, (ii) $a \neq s_1 \vee b \neq s_1 \vee a = b$, and (iii) $a \neq s_2 \vee b \neq s_2 \vee a = b$. Let the variable x be associated with the equality $a = b$. Then, we label the lemmas with (i) $f(x) = t$, (ii) $EQ(x, s_1)$, and (iii) $EQ(x, s_2)$.

We compute an interpolant for A and B using Pudlák's algorithm. Since the input is already in clausal form, we can directly apply resolution. From lemma (ii) and the input clauses $\neg p \vee a = s_1$ and $\neg p \vee b = s_1$ we can derive the clause $\neg p \vee a = b$. The partial interpolant of the derived clause is still $EQ(x, s_1)$, since the partial interpolants of the input clauses are $\bot$ resp. $\top$. Similarly, from lemma (iii) and the input clauses $p \vee a = s_2$ and $p \vee b = s_2$ we can derive the clause $p \vee a = b$ with partial interpolant $EQ(x, s_2)$. A resolution step on these two clauses with p as pivot yields the clause $a = b$. Since p is a shared literal, Pudlák's algorithm introduces the case distinction. Hence, we get the partial interpolant $(EQ(x, s_2) \vee p) \wedge (EQ(x, s_1) \vee \neg p)$. Note that this interpolant has the form $I_1[EQ(x, s_1)][EQ(x, s_2)]$ and, therefore, satisfies the syntactical restrictions.

From the $\mathcal{EUF}$-lemma (i) and the input clauses $f(a) = t$ and $f(b) \neq t$, we can derive the clause $a \neq b$ with partial interpolant $f(x) = t$. Note that this interpolant has the form $I_2(x)$ which also corresponds to the syntactical restrictions needed for our method.

If we apply the final resolution step on the mixed literal $a = b$ using (rule-eq), we get the interpolant $I_1[I_2(s_1)][I_2(s_2)]$ which corresponds to the interpolant $(f(s_2) = t \vee p) \wedge (f(s_1) = t \vee \neg p)$.

5 Linear Real and Integer Arithmetic

Our solver for linear arithmetic is based on a variant of the Simplex approach [9]. 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 $\sum_j k_j a_{ij} = 0$ and $\sum_j k_j b_j < 0$. 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 $\sum_i (\sum_j k_j a_{ij}) x_i \leq \sum_j k_j b_j$. 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 an extended branch [7], which is just a branch of the DPLL engine on a newly introduced literal. In the proof tree this results in a resolution step with this literal as pivot.

Example 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 $b < a$ 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.

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

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. Using the purification and summing up the contributions of the A-part we get the partial interpolants $2x_1 \leq r$ for $a \leq b$ and $2x_2 + t \leq 0$ for $\neg(a \leq b)$. Intuitively, the variable x_1 stands for a and x_2 for $-a$. Summing up the two partial interpolants with $x_1 = -x_2$ 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 A-part, more precisely from $t \leq -2x_2 = 2x_1 \leq r$. The final interpolant computed by our algorithm is $t \leq r \wedge (t \geq r \rightarrow t \leq 2\lfloor r/2 \rfloor)$.

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 $x_1 = -x_2$ 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. This leads to the general form $t - r \leq 0 \wedge (t - r \geq -k \rightarrow F)$. In our example we have $k = 0$ and F specifies that $r = t$ is even.

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 $\boldsymbol{x}$ denote the set of the variables that occur in the pattern. We additionally require that these variables only occur inside a special sub-formula of the form $LA(s(\boldsymbol{x}), k, F(\boldsymbol{x}))$. The first parameter s is a linear term over the variables in $\boldsymbol{x}$ and arbitrary other terms not involving $\boldsymbol{x}$. The coefficients of the variables $\boldsymbol{x}$ 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 $F(\boldsymbol{x})$ is a formula that contains the variables from $\boldsymbol{x}$ at arbitrary positions. To simplify the presentation, we treat $-\varepsilon$ as -1 in the integer case. Again we have a strong and a weak partial interpolant that are obtained by using different definitions for LA. These definitions are

$$LA_S\,(s(\boldsymbol{x}), k, F(\boldsymbol{x})) :\equiv \forall \boldsymbol{x}' \leq \boldsymbol{x} : s(\boldsymbol{x}') \leq 0 \wedge (s(\boldsymbol{x}') \geq -k \rightarrow F(\boldsymbol{x}'))$$
$$LA_W\,(s(\boldsymbol{x}), k, F(\boldsymbol{x})) :\equiv \exists \boldsymbol{x}' \geq \boldsymbol{x} : s(\boldsymbol{x}') \leq 0 \wedge (s(\boldsymbol{x}') \geq -k \rightarrow F(\boldsymbol{x}'))$$

The intuition behind the formula $LA(s(\boldsymbol{x}), k, F(\boldsymbol{x}))$ is that $s(\boldsymbol{x}) \leq 0$ 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 $t \leq r$. Furthermore the A-part implies that there is an even integer value between t and r. If t and r are distinct, this is no problem. However, if $t \geq r$ we need that t is even. Using the above pattern we can choose $k = 0$ and F as the formula that states that t is even.

To see that the strong interpolant $LA_S(s, k, F)$ implies the weak interpolant $LA_W(s, k, F)$, instantiate $\boldsymbol{x}'$ with $\boldsymbol{x}$ in both formulas. Having quantifiers in the interpolant is no problem; once all mixed literals are resolved, all auxiliary variables are removed. Then, the strong and weak interpolant are identical and have no quantifiers.

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.

5.1 Leaf Interpolation

As mentioned above, our solver produces for a clause $C = \neg\ell_1 \vee \cdots \vee \neg\ell_m$ some Farkas coefficients $k_1, \ldots, k_m \geq 0$ such that $\sum_j k_j \ell_j$ yields a contradiction $0 < 0$. The interpolant for a theory lemma can be computed by summing up the A-part of the conflict: I is defined as $\sum_j k_j(\ell_j \downharpoonleft A)$ (if $\ell_j \downharpoonleft A = \top$ 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 $\neg C \downharpoonleft A \iff \ell_1 \downharpoonleft A \wedge \cdots \wedge \ell_m \downharpoonleft A$. Moreover, we have that $I + \sum_j k_j(\ell_j \downharpoonleft B)$ yields $0 < 0$, since for every literal, even for mixed literals, $\ell_j \downharpoonleft A + \ell_j \downharpoonleft B = \ell_j$ holds. This shows that $I \wedge \neg C \downharpoonleft B$ is unsatisfiable.

The linear constraint $\sum_j k_j(\ell_j \downharpoonleft A)$ can easily be expressed as $s(\boldsymbol{x}) \leq 0$. Thus, we can equivalently write the interpolant in our pattern as $LA(s(\boldsymbol{x}), -\varepsilon, \perp)$. Since the Farkas coefficients are all positive and the auxiliary variables introduced to define $\ell \downharpoonleft A$ for mixed literals contain x positively, the resulting term $s(\boldsymbol{x})$ 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. Table 1 shows the corresponding partial interpolants. The non-mixed case is given in the technical report.

Table 1. Interpolation of mixed theory combination clauses. We assume a is A-local, b is B-local, $a-b \leq 0$ has the auxiliary variable x_1, $b-a \leq 0$ has the auxiliary variable x_2 and $a=b$ the auxiliary variables x_a and x_b.

Clause C: $a \neq b \vee a \leq b$	Clause C: $a \neq b \vee b \leq a$
$\neg C \downharpoonright A$: $a = x_a \wedge x_a = x_b \wedge -a + x_1 \leq 0$	$\neg C \downharpoonright A$: $a = x_a \wedge x_a = x_b \wedge a + x_2 \leq 0$
$\neg C \downharpoonright B$: $x_a = x_b \wedge x_b = b \wedge -x_1 + b < 0$	$\neg C \downharpoonright B$: $x_a = x_b \wedge x_b = b \wedge -x_2 - b < 0$
Interpolant I: $LA(-x + x_1, -\varepsilon, \bot)$	Interpolant I: $LA(x + x_2, -\varepsilon, \bot)$

Clause C: $a = b \vee a < b \vee a > b$
$\neg C \downharpoonright A$: $a = x_a \wedge x_a \neq x_b \wedge -a + x_1 \leq 0 \wedge a + x_2 \leq 0$
$\neg C \downharpoonright B$: $a = x_a \wedge x_a \neq x_b \wedge -x_1 + b \leq 0 \wedge -x_2 - b \leq 0$
Interpolant I: $LA(x_1 + x_2, 0, EQ(x, x_1))$

The interpolant for the clause $a = b \vee a < b \vee a > b$ deserves more explanation. This clause is used to propagate equalities from the linear arithmetic solver if it can derive $a \leq b$ and $b \leq a$. In the interpolant, x_1 is the variable with $b \leq x_1 \leq a$, and x_2 the variable with $a \leq -x_2 \leq b$. The formula $LA(x_1 + x_2, 0, EQ(x, x_1))$ basically states that $x_1 \leq -x_2$ and that if $x_1 \geq -x_2$ then x_1 equals the shared value x of the equality $a = b$. We stress that the interpolant has the required form: x_1 and x_2 only occur inside an LA and with the correct coefficients in $x_1 + x_2$ while x only occurs as first parameter of an EQ term, which appears positively in the negation normal form (by the definition of LA_S and LA_W).

5.2 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. In the following we denote the auxiliary variable of the negated literal $\neg(a + b \leq c)$ with x_1 and the auxiliary variable of $a + b \leq c$ with x_2. The intuition here is that x_1 and $-x_2$ correspond to the same value between a and $c - b$. The resolution rule for pivot element $a + b \leq c$ is as follows where the values for s_3, k_3 and F_3 are given later.

$$\frac{\begin{array}{c} C_1 \vee a + b \leq c : I_1[LA(c_1x_1 + s_1(\boldsymbol{x}), k_1, F_1(x_1, \boldsymbol{x}))] \\ C_2 \vee \neg(a + b \leq c) : I_2[LA(c_2x_2 + s_2(\boldsymbol{x}), k_2, F_2(x_2, \boldsymbol{x}))] \end{array}}{C_1 \vee C_2 : I_1[I_2[LA(s_3(\boldsymbol{x}), k_3, F_3)]]} \quad \text{(rule-la)}$$

The formula $LA(s_3, k_3, F_3)$ should hold if and only if there is some $x_1 = -x_2$ such that $LA(c_1x_1 + s_1, k_1, F_1)$ and $LA(c_2x_2 + s_2, k_2, F_2)$ hold. From $c_1x_1 + s_1(\boldsymbol{x}) \leq 0$ and $c_2x_2 + s_2(\boldsymbol{x}) \leq 0$ and $x_1 = -x_2$ we get $c_2s_1(\boldsymbol{x}) + c_1s_2(\boldsymbol{x}) \leq 0$, hence we choose

$$s_3(\boldsymbol{x}) = c_2s_1(\boldsymbol{x}) + c_1s_2(\boldsymbol{x}).$$

For the inverse direction we need to guarantee the existence of $x_1 = -x_2$ between $\frac{s_2(\boldsymbol{x})}{c_2}$ and $\frac{-s_1(\boldsymbol{x})}{c_1}$ such that the following formulae hold:

$$F_1^*(x_1) :\equiv s_1(\boldsymbol{x}) + c_1x_1 \geq -k_1 \rightarrow F_1(x_1, \boldsymbol{x}),$$
$$F_2^*(x_2) :\equiv s_2(\boldsymbol{x}) + c_2x_2 \geq -k_2 \rightarrow F_2(x_2, \boldsymbol{x}).$$

In the integer case, we can guarantee this if $c_2 s_1(\boldsymbol{x}) + c_1 s_2(\boldsymbol{x}) < -c_2 k_1 - c_1 k_2 - c_1 c_2$ by choosing $x_1 = \left\lfloor \frac{-s_1(\boldsymbol{x}) - k_1 - 1}{c_1} \right\rfloor$. Otherwise there are only finitely many candidates for $x_1 = -x_2$ between $\frac{s_2(\boldsymbol{x})}{c_2}$ and $\frac{-s_1(\boldsymbol{x})}{c_1}$. For these we can do a finite case distinction in F_3. This suggests the definitions

$$\begin{aligned} k_3 &:= c_2 k_1 + c_1 k_2 + c_1 c_2 \\ F_3(\boldsymbol{x}) &:\equiv \bigvee_{i=0}^{\left\lceil \frac{k_1+1}{c_1} \right\rceil} F_1^* \left(\left\lfloor \frac{-s_1(\boldsymbol{x})}{c_1} \right\rfloor - i \right) \wedge F_2^* \left(i - \left\lfloor \frac{-s_1(\boldsymbol{x})}{c_1} \right\rfloor \right) \end{aligned} \qquad \text{(int case)}$$

In the real case, we require that k_1 and k_2 are either $-\varepsilon$ and 0. Then, the only candidate for x_1 is $\frac{-s_1(\boldsymbol{x})}{c_1}$. We define

$$\begin{aligned} k_3 &:= \begin{cases} -\varepsilon & \text{if } k_1 = k_2 = -\varepsilon \\ 0 & \text{if } k_1 = 0 \vee k_2 = 0 \end{cases} \\ F_3(\boldsymbol{x}) &:\equiv F_1^* \left(\tfrac{-s_1(\boldsymbol{x})}{c_1} \right) \wedge F_2^* \left(-\tfrac{-s_1(\boldsymbol{x})}{c_1} \right) \end{aligned} \qquad \text{(real case)}$$

With these definition we can state the following lemma.

Lemma 2. *Let $s_1(\boldsymbol{x}), s_2(\boldsymbol{x})$ be linear terms over $\boldsymbol{x}$, $c_1, c_2 \geq 0$, $k_1, k_2 \in \mathbb{Z}$ (integer case) or $k_1, k_2 \in \{0, -\varepsilon\}$ (real case), $F_1(x_1, \boldsymbol{x}), F_2(x_2, \boldsymbol{x})$ arbitrary formulae and s_3, k_3, F_3 as defined above. Then*

$$\begin{gathered} (\exists x_1 . LA_S(c_1 x_1 + s_1(\boldsymbol{x}), k_1, F_1(x_1, \boldsymbol{x})) \wedge LA_S(-c_2 x_1 + s_2(\boldsymbol{x}), k_2, F_2(-x_1, \boldsymbol{x}))) \\ \rightarrow LA_S(s_3(\boldsymbol{x}), k_3, F_3(\boldsymbol{x})) \end{gathered}$$

and

$$\begin{gathered} LA_W(s_3(\boldsymbol{x}), k_3, F_3(\boldsymbol{x})) \rightarrow \\ (\exists x_1 . LA_W(c_1 x_1 + s_1(\boldsymbol{x}), k_1, F_1(x_1, \boldsymbol{x})) \wedge LA_W(-c_2 x_1 + s_2(\boldsymbol{x}), k_2, F_2(-x_1, \boldsymbol{x}))) \end{gathered}$$

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

Theorem 3 (Soundness of (rule-la)). *Let $a + b \leq c$ be a mixed literal with the auxiliary variable x_2, and x_1 be the auxiliary variable of the negated literal. If $I_1[LA(c_1 x_1 + s_1, k_1, F_1)]$ yields two partial interpolants (strong and weak) of $C_1 \vee a + b \leq c$ and $I_2[LA(c_2 x_2 + s_2, k_2, F_2)]$ yields two partial interpolants of $C_1 \vee \neg(a + b \leq c)$ then $I_1[I_2[LA(s_3, k_3, F_3)]]$ yields two partial interpolants 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 $I_1[LA_1^{(1)}] \ldots [LA_n^{(1)}]$ and $I_2[LA_1^{(2)}] \ldots [LA_m^{(2)}]$. The resulting interpolant is

$$I_1[I_2[LA_{11}^{(3)}] \ldots [LA_{1m}^{(3)}]] \ldots [I_2[LA_{n1}^{(3)}] \ldots [LA_{nm}^{(3)}]]$$

where $LA_{ij}^{(3)}$ is computed from $LA_i^{(1)}$ and $LA_j^{(2)}$ as explained above.

6 Conclusion and Future Work

We presented a novel interpolation scheme to extract Craig interpolants from resolution proofs produced by SMT solvers without restricting the solver or reordering the proofs. The key ingredients of our method are virtual purifications of troublesome mixed literals, syntactical restrictions of partial interpolants, and specialized interpolation rules for pivoting steps on mixed literals.

In contrast to previous work, our interpolation scheme does not need specialized rules to deal with extended branches as commonly used in state-of-the-art SMT solvers to solve $\mathcal{LA}(\mathbb{Z})$-formulae. Furthermore, our scheme can deal with resolution steps where a mixed literal occurs in both antecedents, which are forbidden by other schemes [5,11].

Our scheme works for resolution based proofs in the DPLL(T) context provided there is a procedure that generates partial interpolants with our syntactic restrictions for the theory lemmas. We sketched these procedures for the theory lemmas generated by either congruence closure or linear arithmetic solvers producing Farkas proofs. In this paper, we limited the presentation to the combination of the theory of uninterpreted functions, and the theory of linear arithmetic over the integers or the reals. Nevertheless, the scheme could be extended to support other theories. This requires defining the projection functions for mixed literals in the theory, defining a pattern for weak and strong partial interpolants, and proving a corresponding resolution rule.

We plan to produce interpolants of different strengths using the technique from D'Silva et al. [8]. This is orthogonal to our interpolation scheme (particularly to the weak and strong interpolants used for mixed literals). Furthermore, we want to extend the correctness proof to show that our scheme works with inductive sequences of interpolants [15] and tree interpolants [12]. We also plan to extend this scheme to other theories including arrays and quantifiers.

References

1. Brillout, A., Kroening, D., Rümmer, P., Wahl, T.: Beyond Quantifier-Free Interpolation in Extensions of Presburger Arithmetic. In: Jhala, R., Schmidt, D. (eds.) VMCAI 2011. LNCS, vol. 6538, pp. 88–102. Springer, Heidelberg (2011)
2. Bruttomesso, R., Rollini, S., Sharygina, N., Tsitovich, A.: Flexible interpolation with local proof transformations. In: ICCAD, pp. 770–777. IEEE (2010)
3. Christ, J., Hoenicke, J., Nutz, A.: Proof tree preserving interpolation. AVACS Technical Report 89, SFB/TR 14 AVACS (October 2012) ISSN: 1860-9821, `http://www.avacs.org/paper/`
4. Christ, J., Hoenicke, J., Nutz, A.: SMTInterpol: An Interpolating SMT Solver. In: Donaldson, A., Parker, D. (eds.) SPIN 2012. LNCS, vol. 7385, pp. 248–254. Springer, Heidelberg (2012)
5. Cimatti, A., Griggio, A., Sebastiani, R.: Efficient Interpolant Generation in Satisfiability Modulo Theories. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 397–412. Springer, Heidelberg (2008)
6. Detlefs, D., Nelson, G., Saxe, J.B.: Simplify: A theorem prover for program checking. J. ACM 52(3), 365–473 (2005)

7. Dillig, I., Dillig, T., Aiken, A.: Cuts from Proofs: A Complete and Practical Technique for Solving Linear Inequalities over Integers. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 233–247. Springer, Heidelberg (2009)
8. D'Silva, V., Kroening, D., Purandare, M., Weissenbacher, G.: Interpolant Strength. In: Barthe, G., Hermenegildo, M. (eds.) VMCAI 2010. LNCS, vol. 5944, pp. 129–145. Springer, Heidelberg (2010)
9. Dutertre, B., de Moura, L.: A Fast Linear-Arithmetic Solver for DPLL(T). In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 81–94. Springer, Heidelberg (2006)
10. Fuchs, A., Goel, A., Grundy, J., Krstić, S., Tinelli, C.: Ground Interpolation for the Theory of Equality. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 413–427. Springer, Heidelberg (2009)
11. Goel, A., Krstić, S., Tinelli, C.: Ground Interpolation for Combined Theories. In: Schmidt, R.A. (ed.) CADE 2009. LNCS, vol. 5663, pp. 183–198. Springer, Heidelberg (2009)
12. Heizmann, M., Hoenicke, J., Podelski, A.: Nested interpolants. In: POPL, pp. 471–482. ACM (2010)
13. Henzinger, T.A., Jhala, R., Majumdar, R., McMillan, K.L.: Abstractions from proofs. In: POPL 2004, pp. 232–244. ACM (2004)
14. McMillan, K.L.: An Interpolating Theorem Prover. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 16–30. Springer, Heidelberg (2004)
15. McMillan, K.L.: Lazy Abstraction with Interpolants. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 123–136. Springer, Heidelberg (2006)
16. McMillan, K.L.: Interpolants from z3 proofs. In: FMCAD, pp. 19–27. FMCAD Inc. (2011)
17. Nelson, G., Oppen, D.C.: Simplification by cooperating decision procedures. ACM Trans. Program. Lang. Syst. 1(2), 245–257 (1979)
18. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Abstract DPLL and Abstract DPLL Modulo Theories. In: Baader, F., Voronkov, A. (eds.) LPAR 2004. LNCS (LNAI), vol. 3452, pp. 36–50. Springer, Heidelberg (2005)
19. Pudlák, P.: Lower bounds for resolution and cutting plane proofs and monotone computations. J. Symb. Log. 62(3), 981–998 (1997)
20. Yorsh, G., Musuvathi, M.: A Combination Method for Generating Interpolants. In: Nieuwenhuis, R. (ed.) CADE 2005. LNCS (LNAI), vol. 3632, pp. 353–368. Springer, Heidelberg (2005)