

# **A Novel Technique for Computing Craig Interpolants in Satisfiability Modulo the Theory of Integer Linear Arithmetic**

Thi Thieu Hoa Le

November 2010

Technical Report # DISI-10-056 Thiieu Hoa



UNIVERSITÀ DEGLI STUDI DI TRENTO  
Facoltà di Scienze Matematiche Fisiche e Naturali  
Corso di Laurea Specialistica in Informatica

Tesi dal Titolo

**A Novel Technique for Computing Craig  
Interpolants in Satisfiability Modulo the  
Theory of Integer Linear Arithmetic**

Relatore:  
Roberto Sebastiani

Correlatore:  
Alberto Griggio

Laureando:  
Le, Thi Thieu Hoa

Anno Accademico 2009/2010

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>I</b>	<b>BACKGROUND AND STATE-OF-THE-ART</b>	<b>5</b>
<b>2</b>	<b>Background</b>	<b>6</b>
2.1	Satisfiability Modulo Theory - SMT . . . . .	6
2.2	Interpolation definition . . . . .	7
2.3	Interpolation in SMT . . . . .	8
2.4	Interpolant-based model checking . . . . .	11
<b>II</b>	<b>CONTRIBUTIONS</b>	<b>13</b>
<b>3</b>	<b>Interpolants for conjunctions of <math>\mathcal{LA}(\mathbb{Z})</math>-literals</b>	<b>14</b>
3.1	Objective . . . . .	14
3.2	Approach overview . . . . .	16
3.3	Division elimination . . . . .	16
3.4	Interpolant from an $\mathbf{R}'$ -proof of unsatisfiability . . . . .	18
3.4.1	Partially interpolating rules . . . . .	18
3.4.2	Fully interpolating rules . . . . .	19
3.5	Soundness of partially interpolating rules . . . . .	19
3.6	Soundness of fully interpolating rules . . . . .	20
3.6.1	1-CONN . . . . .	20
3.6.2	1-CONN* . . . . .	21
3.6.3	k-CONN . . . . .	22
3.7	Examples . . . . .	24
3.7.1	With one STRENGTHEN application . . . . .	24
3.7.2	With two or more STRENGTHEN applications . . . . .	25
<b>4</b>	<b>Implementation and Empirical results</b>	<b>28</b>

<b>5</b>	<b>Conclusions</b>	<b>29</b>
5.1	Conclusion and future work . . . . .	29
	<b>Bibliography</b>	<b>30</b>

# Chapter 1

## Introduction

Given two formulas  $\mathbf{A}$  and  $\mathbf{B}$  such that  $\mathbf{A} \wedge \mathbf{B}$  is unsatisfiable, a *Craig interpolant* is a formula  $\mathbf{I}$  with the following properties: (i)  $\mathbf{A}$  entails  $\mathbf{I}$  (ii)  $\mathbf{I} \wedge \mathbf{B}$  is unsatisfiable (iii) all uninterpreted symbols of  $\mathbf{I}$  occur in both  $\mathbf{A}$  and  $\mathbf{B}$ . Ever since the seminal work of McMillan [11], interpolation has seen a variety of applications in formal verification. For instance, in interpolation-based model-checking [11, 12] which is a counterexample-guided abstraction refinement (CEGAR) framework, interpolants are used for finding sets of predicates needed to rule out spurious counterexamples. The task of finding such interpolants is carried out by an interpolating theorem prover. There have been efficiently-interpolating provers available for various theories including propositional logic [15, 13], linear arithmetic over the rationals ( $\mathcal{LA}(\mathbb{Q})$ ) [12, 6, 4, 16] with uninterpreted function ( $\mathcal{EUF}$ ) [20, 19, 6, 2]. Many applications in model checking and software verification, however, require variable representation in the interger domain [14, 10]. As a result, many attempts have been made on investigating interpolation for integer arithmetic, specifically for *fragments* of integer linear arithmetic ( $\mathcal{LA}(\mathbb{Z})$ ) (such as Theory of Unit-Two-Variable-Per-Inequality ( $\mathcal{UTVPI}(\mathbb{Z})$ ) [5] and linear equality [9]) and for *full* quantifier-free *Presburger arithmetic* ( $\mathcal{QFPA}$ ) with uninterpreted predicates [3].

Moving in the same direction of exploring interpolation for  $\mathcal{LA}(\mathbb{Z})$ , in this thesis we present a solution to the interpolation problem of  $\mathcal{LA}(\mathbb{Z})$  based on interpolant generation for conjunctions of  $\mathcal{LA}(\mathbb{Z})$  equalities and weak inequalities. Although our approach was inspired by Brillout et al.'s *interpolating sequent calculus* [3], it differs from the latter in that our interpolation method is based on *cutting-plane* proofs of unsatisfiability rather than *sequent-calculus* proofs and consists of interpolation rules built to reflect the cutting plane rules. The proposed algorithm has been implemented within MATHSAT SMT solver and tested for simple cases...

The thesis is organized as follows. In chapter 2, we present some background on interpolation in SMT. We then go into the details of our interpolation method for conjunctions of  $\mathcal{LA}(\mathbb{Z})$  equalities and weak inequalities in chapter 3. We also give several examples to illustrate our algorithm and present some preliminary evaluation of our prototype tool on the RINGS benchmark. Finally, we conclude and outline our future work in chapter 4.

# Part I

## BACKGROUND AND STATE-OF-THE-ART

# Chapter 2

## Background

In this chapter we provide the necessary background knowledge on SMT, interpolation generation in SMT and model checking based on interpolation.

### 2.1 Satisfiability Modulo Theory - SMT

The material of this section is taken from [6] to which we would refer the readers for more details.

Our setting is standard first order logic. A 0-ary function symbol is called a *constant*. A *term* is a first-order term built out of function symbols and variables. A *linear term* is a linear combination  $c_1x_1 + \dots + c_nx_n + c$ , where  $c$  and  $c_i$  are numeric constants and  $x_i$  are variables. Linear terms where all constants and variables are constrained to be integers are said to be (purely) integer linear arithmetic or  $\mathcal{LA}(\mathbb{Z})$ -terms, where  $\mathbb{Z}$  is the set of integers. If  $t_1, \dots, t_n$  are terms and  $p$  is a predicate symbol, then  $p(t_1, \dots, t_n)$  is an *atom*. A *literal* is either an atom or its negation and a disjunction of literals is called a *clause*. A *formula*  $\phi$  is built in the usual way out of the universal and existential quantifiers, Boolean connectives, and atoms. We call a formula *quantifier-free* if it does not contain quantifiers, and *ground* if it does not contain free variables. A formula is said to be in *conjunctive normal form* (CNF) if it is a conjunction of clauses. Without loss of generality, we assume that formulas are in CNF. We also assume the standard first-order notions of interpretation, satisfiability, validity, logical consequence. We denote formulas with  $\phi, \psi, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{I}$ , variables with  $x, y, z$ , and numeric constants with  $a, b, c, d, i, j, k$ .

We call Satisfiability Modulo (the) Theory  $\mathcal{T}$ ,  $\text{SMT}(\mathcal{T})$ , the problem of deciding the satisfiability of quantifier-free formulas wrt. a background theory  $\mathcal{T}$ . Given a theory  $\mathcal{T}$ , we write  $\phi \models_{\mathcal{T}} \psi$  (or simply  $\phi \models \psi$ ) to denote

that the formula  $\psi$  is a logical consequence of  $\phi$  in the theory  $\mathcal{T}$ . We call  $\mathcal{T}$ -solver any procedure that decides the satisfiability of a conjunction of ground atomic formulas and their negations in  $\mathcal{T}$ . With  $\phi \preceq \psi$  we denote that all uninterpreted (in  $\mathcal{T}$ ) symbols of  $\phi$  appear in  $\psi$ . If  $\mathbf{C}$  and  $\mathbf{B}$  are clauses,  $\mathbf{C} \downarrow \mathbf{B}$  is the clause obtained by removing all the literals whose atoms do not occur in  $\mathbf{B}$ , and  $\mathbf{C} \setminus \mathbf{B}$  that obtained by removing all the literals whose atoms do occur in  $\mathbf{B}$ .

A standard technique for solving the SMT( $\mathcal{T}$ ) problem is to integrate a DPLL-based SAT solver and a  $\mathcal{T}$ -solver in a lazy manner (see, e.g., [17, 18] for a detailed description). DPLL is used as an enumerator of truth assignments for the propositional abstraction of the input formula. At each step, the set of  $\mathcal{T}$ -literals  $S$  corresponding to the current assignment is sent to the  $\mathcal{T}$ -solver to be checked for consistency in  $\mathcal{T}$ . If  $S$  is inconsistent, the  $\mathcal{T}$ -solver returns a conflict set of  $\mathcal{T}$ -literals  $\eta$  and the corresponding  $\mathcal{T}$ -lemma  $\neg\eta$  is added as a blocking clause in DPLL, and used to drive the backjump mechanism.

## 2.2 Interpolation definition

**Definition 1 - Craig Interpolant.** Given an ordered pair  $(\mathbf{A}, \mathbf{B})$  such that  $\mathbf{A} \wedge \mathbf{B} \models_{\mathcal{T}} \perp$ , a *Craig interpolant* (simply "interpolant" hereafter) is a formula  $\mathbf{I}$  with the following properties:

- (i)  $\mathbf{A} \models_{\mathcal{T}} \mathbf{I}$
- (ii)  $\mathbf{I} \wedge \mathbf{B} \models_{\mathcal{T}} \perp$
- (iii)  $\mathbf{I} \preceq \mathbf{A}$  and  $\mathbf{I} \preceq \mathbf{B}$

Intuitively,  $\mathbf{I}$  is an abstraction of  $\mathbf{A}$  which summarizes and expresses in the shared language (i.e. over the common variables of  $\mathbf{A}$  and  $\mathbf{B}$ ) *why*  $\mathbf{A}$  is inconsistent with  $\mathbf{B}$ .

**Definition 2 - Resolution Proof.** Given a set of clauses  $S \stackrel{def}{=} \{\mathbf{C}_1, \dots, \mathbf{C}_n\}$  and a clause  $\mathbf{C}$ , we call a resolution proof of  $\bigwedge_i \mathbf{C}_i \models_{\mathcal{T}} \mathbf{C}$  a *directed acyclic graph*  $G$  that satisfies properties (i)-(iii). Moreover, if  $\mathbf{C}$  is the empty clause (denoted with  $\perp$ ), then  $G$  is a *resolution proof of unsatisfiability* for  $\bigwedge_i \mathbf{C}_i$ .

- (i)  $\mathbf{C}$  is the root of  $G$
- (ii) the leaves of  $G$  are either elements of  $S$  or  $\mathcal{T}$ -lemmas
- (iii) each non-leaf node  $\mathbf{C}' = \phi_1 \vee \phi_2$  has two parents  $\mathbf{C}_{p_1}$  and  $\mathbf{C}_{p_2}$  such that  $\mathbf{C}_{p_1} = p \vee \phi_1$ ,  $\mathbf{C}_{p_2} = \neg p \vee \phi_2$ . The atom  $p$  is called the pivot of  $\mathbf{C}_{p_1}$  and  $\mathbf{C}_{p_2}$ .

An interpolant can be efficiently derived from such resolution proofs in many theories of interest including propositional logic,  $\mathcal{LA}(\mathbb{Q})$  and  $\mathcal{EUF}$ , etc.

## 2.3 Interpolation in SMT

Interpolation has been introduced for propositional logic [11],  $\mathcal{LA}(\mathbb{Q})$  and  $\mathcal{EUF}$  [12]. The technique is based on earlier work by Pudlák [15] where two interpolation algorithms are described: one for computing interpolants for propositional formulas from resolution proofs of unsatisfiability, and one for generating interpolants for conjunctions of (weak) linear inequalities in  $\mathcal{LA}(\mathbb{Q})$ . An interpolant for a pair  $(\mathbf{A}, \mathbf{B})$  can be constructed from its resolution proof of unsatisfiability as follows:

---

**Algorithm 1** : Interpolant generation for  $\text{SMT}(\mathcal{T})$

---

- 1: Generate a resolution proof of unsatisfiability  $G$  for  $\mathbf{A} \wedge \mathbf{B}$ .
  - 2: For every  $\mathcal{T}$ -lemma  $\neg\eta$  occurring in  $G$ , generate an interpolant  $\mathbf{I}_{\neg\eta}$  for  $(\eta \setminus \mathbf{B}, \eta \downarrow \mathbf{B})$ .
  - 3: For every input clause  $\mathbf{C}$  in  $G$ , set  $\mathbf{I}_{\mathbf{C}} \stackrel{\text{def}}{=} \mathbf{C} \downarrow \mathbf{B}$  if  $\mathbf{C} \in \mathbf{A}$  and  $\mathbf{I}_{\mathbf{C}} \stackrel{\text{def}}{=} \top$  if  $\mathbf{C} \in \mathbf{B}$ .
  - 4: For every inner node  $\mathbf{C}$  of  $G$  obtained by resolution from  $\mathbf{C}_1 \stackrel{\text{def}}{=} p \vee \phi_1$  and  $\mathbf{C}_2 \stackrel{\text{def}}{=} \neg p \vee \phi_2$ , set  $\mathbf{I}_{\mathbf{C}} \stackrel{\text{def}}{=} \mathbf{I}_{\mathbf{C}_1} \vee \mathbf{I}_{\mathbf{C}_2}$  if  $p$  does not occur in  $\mathbf{B}$ , and  $\mathbf{I}_{\mathbf{C}} \stackrel{\text{def}}{=} \mathbf{I}_{\mathbf{C}_1} \wedge \mathbf{I}_{\mathbf{C}_2}$  otherwise.
  - 5: Output  $\mathbf{I}_{\perp}$  as an interpolant for  $(\mathbf{A}, \mathbf{B})$ .
- 

In Algorithm 1, step 2 is the only part which depends on the theory  $\mathcal{T}$ , so the interpolation problem in  $\text{SMT}(\mathcal{T})$  is reduced to finding interpolants for  $\mathcal{T}$ -lemmas. To this extent, McMillan presents a set of rules for interpolation construction of  $\mathcal{T}$ -lemmas in  $\mathcal{LA}(\mathbb{Q})$ ,  $\mathcal{EUF}$  and their combination [12]. Moving in this direction, in order to deal with the interpolation problem in  $\text{SMT}(\mathcal{LA}(\mathbb{Z}))$ , we propose our set of rules to generate interpolants for  $\mathcal{LA}(\mathbb{Z})$ -lemmas of which the details is deferred until chapter 3. Here we give an example to illustrate Algorithm 1.

*Example 1: Consider the following two formulas in  $\mathcal{LA}(\mathbb{Z})$ :*

$$\mathbf{A} \stackrel{\text{def}}{=} (p \vee (-x_1 + 3x_2 - 1 \leq 0)) \wedge (-x_1 - x_2 \leq 0) \wedge (\neg q \vee \neg(-x_1 - x_2 \leq 0))$$

$$\mathbf{B} \stackrel{\text{def}}{=} (\neg(-x_3 + 2x_1 + 3 \leq 0) \vee (2x_3 - 1 \leq 0)) \wedge (\neg p \vee q) \wedge (p \vee (-x_3 + 2x_1 + 3 \leq 0))$$

*Figure 2.1(a) shows a resolution proof of unsatisfiability for  $\mathbf{A} \wedge \mathbf{B}$ , in which the clauses from  $\mathbf{A}$  have been underlined. The proof contains the following*

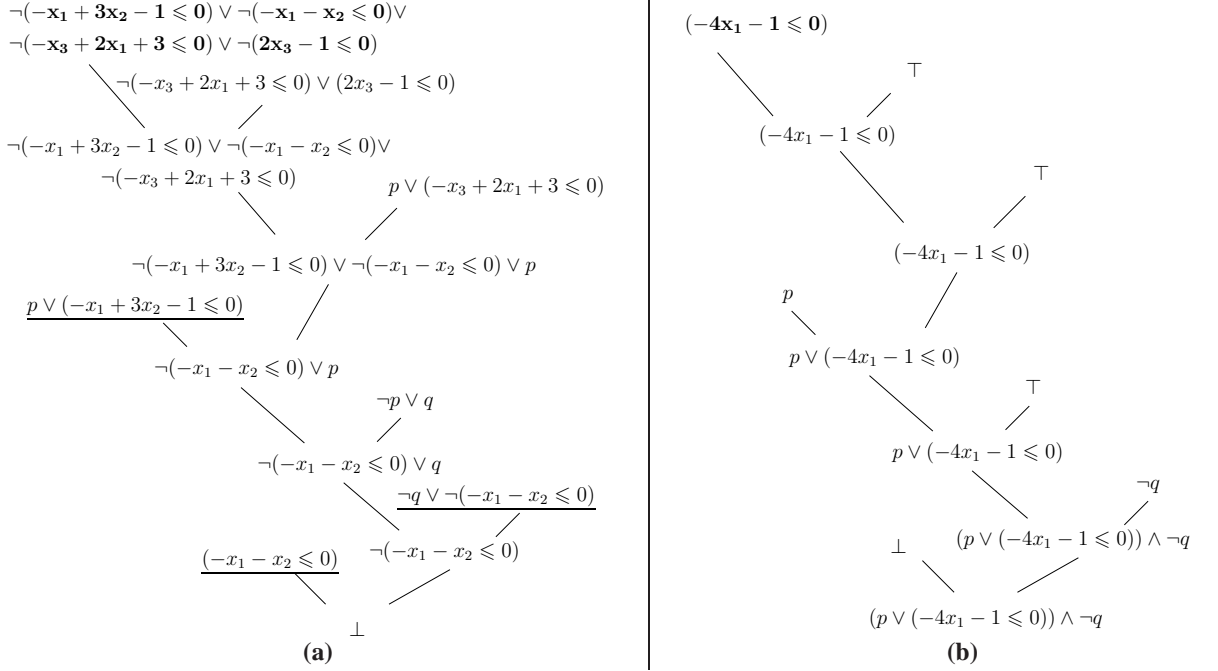


Figure 2.1: Resolution proof of unsatisfiability (a) and interpolant (b) for the pair  $(A, B)$  of formulas of Example 1.

$\mathcal{LA}(\mathbb{Z})$ -lemma (displayed in boldface):

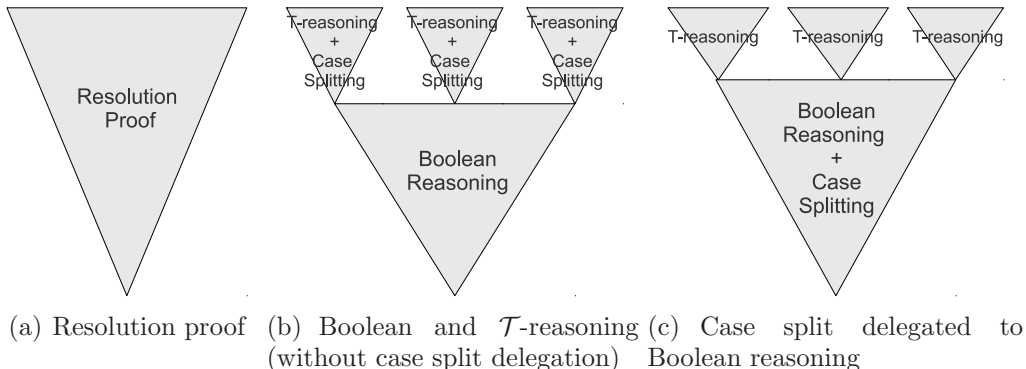
$$\neg(-x_1 + 3x_2 - 1 \leq 0) \vee \neg(-x_1 - x_2 \leq 0) \vee \neg(-x_3 + 2x_1 + 3 \leq 0) \vee \neg(2x_3 - 1 \leq 0)$$

Figure 2.1(b) shows, for each clause  $\Theta_i$  in the proof, the formula  $I_{\Theta_i}$  generated by Algorithm 1. As for the  $\mathcal{LA}(\mathbb{Z})$ -lemma, it is easy to see that  $(-4x_1 - 1 \leq 0)$  is an interpolant for

$$(-x_1 + 3x_2 - 1 \leq 0) \wedge (-x_1 - x_2 \leq 0) \wedge (-x_3 + 2x_1 + 3 \leq 0) \wedge (2x_3 - 1 \leq 0)$$

as required by Step 2 of Algorithm 1. (We will show how to obtain this interpolant in Example 2). Therefore,  $\mathbf{I}_{\perp} \stackrel{\text{def}}{=} (p \vee (-4x_1 - 1 \leq 0)) \wedge \neg q$  is an interpolant for  $(\mathbf{A}, \mathbf{B})$ .

Step 1 of Algorithm 1 requires generating resolution proofs of unsatisfiability which can be performed by a *lazy* SMT solver [17, 18]. The lazy approach is based on the integration of a DPLL-based SAT solver and one (or more)  $\mathcal{T}$ -solvers, respectively handling the Boolean and the theory-specific components of reasoning: the SAT solver enumerates truth assignments which satisfy the Boolean abstraction of the input formula while the  $\mathcal{T}$ -solvers



checks the consistency in  $\mathcal{T}$  of the set of literals corresponding to the assignments enumerated. In addition, a  $\mathcal{T}$ -solver may be based on case splitting or splitting-on-demand [1] either because the solver is bound to resort to some form of case splitting when deciding the  $\mathcal{T}$ -consistency of a conjunction of  $\mathcal{T}$ -literals is NP-hard, or because of simplicity and convenience when the  $\mathcal{T}$ -consistency problem is polynomial.

Unfortunately, the restriction put on step 2 of Algorithm 1 (which requires all atomic predicates occurring in  $\neg\eta$  to occur either in  $\mathbf{A}$  or in  $\mathbf{B}$ , see [12] for more details) prevents us from splitting cases on atomic predicates neither present in  $\mathbf{A}$  nor  $\mathbf{B}$ . In other words, we cannot make arbitrary cuts. However, we can effectively split cases on any atomic predicate  $\phi$  as long as  $\phi \preceq \mathbf{A}$  or  $\phi \preceq \mathbf{B}$ . For instance, assuming that  $\phi \preceq \mathbf{A}$ , we can add the  $\mathcal{T}$ -lemma  $\phi \vee \neg\phi$  to  $\mathbf{A}$  and send the lemma back to the DPLL engine. In this way, we can introduce the predicate  $\phi$  into the proof and split cases on it while preserving the validity of any interpolant that we may obtain [12]. Moreover, the implementation benefits greatly from exploiting the DPLL engine for the exploration of the branches introduced by splitting cases since it can take advantage for free of all the advanced techniques (e.g. conflict-driven backjumping, learning, etc.) for search-space pruning implemented in modern DPLL engines. In this thesis, we advocate delegating these case splits to the DPLL engines instead of performing them within the  $\mathcal{T}$ -solver (or the  $\mathcal{LA}(\mathbb{Z})$ -solver in our case) itself. Figure 2.2(a) shows the resolution proof as a whole while figure 2.2(b) and 2.2(c) depicts the proof as an integration of Boolean and theory specific reasoning ( $\mathcal{T}$ -reasoning) without and with the case split delegation.

For rational and real linear arithmetic, there have been many approaches to the interpolation problems. Rybalchenko and Sofronie-Stokkermans [16] show how to compute interpolants in combined  $\mathcal{LA}(\mathbb{Q})$ ,  $\mathcal{EUF}$  and real linear arithmetic  $\mathcal{LA}(\mathbb{R})$  by using linear programming solvers in a black-box

fashion. The method allows both weak and strict inequalities. Cimatti et al. [6] show how to compute interpolants in for  $\mathcal{LA}(\mathbb{Q})$ , rational difference logic fragment and  $\mathcal{EUF}$ . By utilizing state-of-the-art SMT algorithms they obtain significant improvements over existing interpolation tools for  $\mathcal{LA}(\mathbb{Q})$  and  $\mathcal{EUF}$ . Yorsh and Musuvathi [20] present a Nelson-Oppen style method for generating interpolants for combined theories by using the interpolation procedures for individual theories. This technique is further improved in [6, 4].

For integer linear arithmetic, the problem of interpolant generation still remains open. In fact, there is no known and efficient algorithm for computing interpolants in  $\mathcal{LA}(\mathbb{Z})$ . Quantifier elimination is one known-interpolation method, however, it has exponential complexity and does not immediately yield efficient algorithms for computing interpolants. Brillout et al. [3] shows how to compute interpolants for  $\mathcal{QFPA}$ , i.e., linear arithmetic over the integers, using interpolating sequent calculus. This approach, however, does not provide any guarantee on the size of an interpolating proof and its interpolant. While they proposed to *lift* proofs of unsatisfiability to *interpolating* proofs to avoid the well-known disadvantages of quantifier-elimination-based interpolation, they risked increasing the size of the proofs and their interpolants exponentially in general. So far, efficient interpolating solvers have been reported only for *fragments* of  $\mathcal{LA}(\mathbb{Z})$ . Jain et al. [9] proposed efficient interpolation algorithms for conjunctions of linear Diophantine (dis)equations and for conjunction of linear modular equations. Moving in the same track of Jain et al., Cimatti et al. [6] address effectively the interpolation problem for  $\mathcal{UTVPI}(\mathbb{Z})$  which is another important fragment of  $\mathcal{LA}(\mathbb{Z})$ .

## 2.4 Interpolant-based model checking

One of the important applications of interpolation in Formal Verification is interpolant-based model checking which is a combination of bounded model checking and interpolation to produce an over-approximate image operator that can be used in symbolic model checking. This application have been discussed in literature, for example [11, 7]. For the sake of completeness, we report here a brief summary of that application, referring the interested readers to the cited papers for further details.

A transition system  $M = (S, T)$  consists a finite set of states  $S$  and a transition relation  $T \subseteq S \times S$ . A bounded model checking(BMC) problem consists of a set of constraints: initial constraints( $I$ ), transition constraints( $T$ ) and final constraints( $F$ ). These constraints are encoded in propositional logic and

translated to CNF. For instance, the propositional encoding of  $T \subseteq S \times S$  is  $T(x, x')$  where  $x$  and  $x'$  are vectors of propositional variables representing system states. An instance of BMC is a formula  $A(x_0, x_1) \wedge B(x_1, \dots, x_k)$  where:

- $A(x_0, x_1) \stackrel{def}{=} I(x_0) \wedge T(x_0, x_1)$
- $B(x_1, \dots, x_k) \stackrel{def}{=} T(x_1, x_2) \wedge \dots \wedge T(x_{k-1}, x_k) \wedge (F(x_1) \vee \dots \vee F(x_k))$

The formula  $P(x_1) \stackrel{def}{=} \exists x_0. A(x_0, x_1)$  encodes the forward image of  $I$  and using  $P(x_1)$  we can repeatedly compute images until we obtain a formula encoding all reachable states from  $I$ . However, the formula  $P(x_1)$  is quantified and quantifier elimination is very expensive, so computing the set of reachable states from  $I$  in this manner is not feasible.

Instead, using a SAT solver, we prove that  $A(x_0, x_1) \wedge B(x_1, \dots, x_k)$  is unsatisfiable (i.e., there are no counterexamples of length  $k$ ). From the proof, we derive an interpolant  $p$  for  $A(x_0, x_1) \wedge B(x_1, \dots, x_k)$ . By definition of interpolant, we have  $A(x_0, x_1) \models p$  meaning  $p$  is implied by the initial condition and the first transition constraint, it follows that  $p$  is true in every state reachable from the initial state in one step. That is,  $p$  is an *over-approximation* of the forward image of  $I$ . Further,  $p \wedge B(x_1, \dots, x_k) \models \perp$  meaning that no state satisfying  $p$  can reach a final state in  $k - 1$  steps.

This over-approximation is useful in computing approximately reachable states and may make us falsely conclude that  $F$  is reachable. However, by increasing  $k$ , we must eventually find a true counterexample (a path from  $I$  to  $F$ ) or prove that  $F$  is not reachable.

Part II

**CONTRIBUTIONS**

# Chapter 3

## Interpolants for conjunctions of $\mathcal{LA}(\mathbb{Z})$ -literals

We show in section 2.3 that the interpolation problem in  $\text{SMT}(\mathcal{LA}(\mathbb{Z}))$  could be reduced to finding interpolants for  $\mathcal{LA}(\mathbb{Z})$ -lemmas. Moreover, we prefer delegating the case split to the DPLL engine to handling it within the  $\mathcal{LA}(\mathbb{Z})$ -solver. Therefore, we have shrunk the problem of finding interpolants for  $\mathcal{LA}(\mathbb{Z})$ -lemmas to the possible smallest extent at which it is addressed fully by our interpolation method presented in this chapter.

### 3.1 Objective

We first recall the algorithm of McMillan [12] for computing  $\mathcal{LA}(\mathbb{Q})$ -interpolants from unsatisfiable proofs for conjunctions of  $\mathcal{LA}(\mathbb{Q})$ -equalities and weak inequalities.

- An  $\mathcal{LA}(\mathbb{Q})$ -*proof rule* for such a conjunction is either an element of  $\Gamma$  or it has the form  $\frac{P}{\phi}$  where  $\phi$  is an equality or a weak inequality and  $P$  is a sequence of proof rules, called the *premises* of the rule.
- An  $\mathcal{LA}(\mathbb{Q})$ -*proof of unsatisfiability* for such a conjunction is simply a rule in which  $\phi \equiv (c \leq 0)$  where  $c$  is a positive numerical constant.

Given an  $\mathcal{LA}(\mathbb{Q})$ -proof of unsatisfiability for a conjunction  $\Gamma$  of equalities and weak inequalities partitioned into  $(A, B)$ , an interpolant  $I$  can be computed simply by replacing every atom  $t \leq 0$  occurring in  $B$  (resp.  $t = 0$ ) with  $0 \leq 0$  (resp.  $0 = 0$ ) in each leaf sub-rule of the proof, and propagating the results. The interpolant is then the single weak inequality  $t \leq 0$  at the proof root [12].

Similarly to McMillan's rules, we use the following proof rules for a conjunction of  $\mathcal{LA}(\mathbb{Z})$ -equalities and weak inequalities: **LEQEQ** for deriving inequalities from equalities, and **COMB** for performing linear combinations. In this thesis, we assume that all inequalities have the form of  $\sum_i c_i v_i + c \leq 0$ .

$$1. \text{ LEQEQ } \frac{t \circ 0}{t \leq 0}$$

where  $\circ \in \{=, \leq\}$  and  $t = \sum_i c_i v_i + c$  ( $c_i, v_i$  denote integer constant and variable)

$$2. \text{ COMB } \frac{t_1 \leq 0 \quad t_2 \leq 0}{c_1 t_1 + c_2 t_2 \leq 0} \text{ where } c_1, c_2 > 0$$

Furthermore, the notion of an  $\mathcal{LA}(\mathbb{Z})$ -proof of unsatisfiability for such a conjunction is similar to that of an  $\mathcal{LA}(\mathbb{Q})$ -proof of unsatisfiability except that  $c$  must be an integer constant. However, relying solely on the above two rules would make our proof system incomplete. Thus, we have augmented our proof system with the *cutting plane* rules for integer inequalities [15]. Cutting plane proof systems were proved to be complete by Gomory [8]. In particular, we make use of the following rule:

$$3. \text{ DIV } \frac{\sum_i c_i v_i + c \leq 0}{\sum_i \frac{c_i}{d} v_i + \lceil \frac{c}{d} \rceil \leq 0} \text{ where } d > 0 \text{ is the common divisor of } c_i$$

We call  $\mathbf{R}$  the set of the above three  $\mathcal{LA}(\mathbb{Z})$  rules. Given an  $\mathcal{LA}(\mathbb{Z})$ -proof of unsatisfiability  $P$  for a conjunction  $\Gamma$  of  $\mathcal{LA}(\mathbb{Z})$ -equalities and weak inequalities partitioned into  $(\mathbf{A}, \mathbf{B})$  built from  $\mathbf{R}$  (called an  $\mathbf{R}$ -proof hereafter), our objective is to construct an interpolation algorithm for  $(\mathbf{A}, \mathbf{B})$ . Our algorithm works exactly in the same way as McMillan's interpolation algorithm for proofs composed of only rule **LEQEQ** and rule **COMB** as can be demonstrated in the example below. Moreover, our algorithm provides a mechanism to compute an interpolant for proofs including also the cutting plane rules (i.e. the **DIV** rule).

*Example 2: Consider the following sets of  $\mathcal{LA}(\mathbb{Z})$  atoms:*

$$\mathbf{A} \stackrel{\text{def}}{=} \{(-x_1 + 3x_2 + 1 \leq 0), (-x_1 - x_2 \leq 0)\}$$

$$\mathbf{B} \stackrel{\text{def}}{=} \{(-x_3 + 2x_1 + 3 \leq 0), (-1 + 2x_3 \leq 0)\}$$

*An  $\mathbf{R}$ -proof for  $\mathbf{A} \wedge \mathbf{B}$  is the following:*

$$\frac{\frac{-x_1 + 3x_2 + 1 \leq 0 \quad 3 * (-x_1 - x_2 \leq 0)}{-4x_1 - 1 \leq 0} \quad \frac{2x_3 - 1 \leq 0 \quad 2 * (-x_3 + 2x_1 + 3 \leq 0)}{4x_1 + 5 \leq 0}}{4 \leq 0}$$

By replacing inequalities in  $\mathbf{B}$  with  $0 \leq 0$ , we obtain the interpolation proof:

$$\frac{\frac{-x_1 + 3x_2 + 1 \leq 0 \quad 3 * (-x_1 - x_2 \leq 0)}{-4x_1 - 1 \leq 0} \quad \frac{0 \leq 0 \quad 2 * (0 \leq 0)}{0 \leq 0}}{-4x_1 - 1 \leq 0}$$

Thus, the interpolant obtained is  $(-4x_1 - 1 \leq 0)$ .

## 3.2 Approach overview

In order to extract interpolants from proofs of unsatisfiable conjunctions  $\mathbf{A} \wedge \mathbf{B}$ , we will first need to annotate formulae introduced along the proof with *partial interpolants* (PIs) that record the  $\mathbf{A}$ -contribution to a formula obtained jointly from  $\mathbf{A}$  and  $\mathbf{B}$ . Then, from those PIs we can construct an interpolant for  $(\mathbf{A}, \mathbf{B})$ . Formulae introduced by **LEQEQ** and **COMB** can always be annotated with a PI while those introduced by **DIV** can not. For instance, an inequality  $2x - 1 \leq 0[3x - y \leq 0]$  could not derive the corresponding PI for  $x \leq 0$  because 2 is not the common divisor of the  $3x - y \leq 0$ . As a solution, we will perform a multiplication by  $d$  right after the division by  $d$  is done in the proof. In other words, we replace the **DIV** rule with the **STRENGTHEN** rule combining both division by  $d$  and multiplication by  $d$  into one single rule. The new set containing proof rules 1, 2 and 4 is called  $\mathbf{R}'$ .

$$4. \text{ STRENGTHEN } \frac{\sum_i c_i v_i + c \leq 0}{\sum_i c_i v_i + d \lceil \frac{c}{d} \rceil \leq 0} \text{ where } d \text{ is the common divisor of } c_i$$

The approach to our aforementioned problem consists of two main steps:

- First, replace the **DIV** rule and rewrite  $P$  using the new set of proof rules  $\mathbf{R}'$ , the re-written proof is called an  $\mathbf{R}'$ -proof of unsatisfiability or  $P'$ .
- Second, extract an interpolant for  $(\mathbf{A}, \mathbf{B})$  from  $P'$ .

The first step is addressed in section 3.3 and the second step in section 3.4.

## 3.3 Division elimination

**Theorem 1.** For any  $\mathbf{R}$ -proof of unsatisfiability  $P$ , there exists an  $\mathbf{R}'$ -proof of unsatisfiability  $P'$  that proves the same property. In addition, if  $P$  is viewed

as a tree whose nodes are formulae and edges are annotated with integer coefficients used in the rules, then  $P$  and  $P'$  have exactly the same number of nodes and edges. The only difference between them is that formulae in  $P'$  are multiplications of formulae in  $P$  and some factor  $f$ .

**PROOF.** by induction on the depth  $k$  of the proof tree.

In order to easily identify which node is the result of which rule, we will tag the initial letter of the rule on each node. That is an **L/C/D/S**-node is a node introduced by rule **LEQEQ/COMB/DIV/STRENGTHEN** respectively. **C**-nodes have two parental nodes and edges annotated with integer coefficients while other nodes have only one parental node and annotated edge.

- $k = 1$ : If the root of  $P$  is either an **L**-node or a **C**-node, then  $P' \equiv P$  and nodes in  $P'$  are multiplications of nodes in  $P$  and 1. If the root of  $P$  is a **D**-node introduced by a division by  $d$ , then  $P'$  is obtained from  $P$  by replacing the **D**-node with its multiplication by  $d$ . The root of  $P'$  is actually an **S**-node with the gaining factor of  $d$ , meaning that the  $P'$ -root formula is a multiplication of  $d$  and the  $P$ -root formula.
- Now assume that the theorem is true for  $k = n$ . Then for an **R**-proof tree  $P$  of depth  $n + 1$ , if its root is a **C**-node with two parental edges annotated with  $c_L$  and  $c_R$ , we can always change its left and right subtree into **R'**-proof trees with gaining factor of  $f_L$  and  $f_R$  respectively. To finish structuring  $P'$ , we create a **C**-root and annotate the two edges that connect it to the new left subtree with  $c_L * f_R$  and to the new right subtree with  $c_R * f_L$ . The gaining factor of the new root is then  $f_L * f_R$ . If  $P$  instead has a **D**-root, we similarly change its subtree into an **R'**-proof tree with a gaining factor  $f$  and then create an **S**-root to connect to it. The gaining factor at this new **S**-root is  $f * d$ .

The induction steps show that we can always transform any **R**-proof  $P$  into an **R'**-proof  $P'$  whose nodes are multiplications of nodes in  $P$  and some factor. Particularly, the  $P'$ -root formula is a multiplication of the  $P$ -root formula which is  $\perp$ . Therefore, this results in  $P'$  being also a proof of unsatisfiability and completes the proof of the theorem.

## 3.4 Interpolant from an $\mathbf{R}'$ -proof of unsatisfiability

Generating interpolants for  $\mathbf{R}'$ -proofs consists of two phases:

- Compute PIs for formulae introduced along the proof tree using rules given in section 3.4.1.
- Derive the final interpolant from the PIs using rules given in section 3.4.2.

### 3.4.1 Partially interpolating rules

**Definition 3 - Partial Interpolant.** A *partial interpolant* (PI) [3] is a formula  $\theta_{pi}$  that record the  $\mathbf{A}$ -contribution to a formula  $\theta$  obtained jointly from  $\mathbf{A}$  and  $\mathbf{B}$  such that: (i)  $\mathbf{A} \vdash \theta_{pi}$  (ii)  $\mathbf{B} \wedge \theta_{pi} \vdash \theta$

To derive PIs for formulae obtained from rules in  $\mathbf{R}'$ , we use the following partially interpolating rules:

1.  $\text{LEQEQ}_L \quad \frac{t \circ 0}{t \leq 0 [t \leq 0]} \quad t \circ 0 \in \mathbf{A}$
2.  $\text{LEQEQ}_R \quad \frac{t \circ 0}{t \leq 0 [0 \leq 0]} \quad t \circ 0 \in \mathbf{B}$
3.  $\text{COMB} \quad \frac{t_1 \leq 0 [t_{pi_1} \leq 0] \quad t_2 \leq 0 [t_{pi_2} \leq 0]}{c_1 t_1 + c_2 t_2 \leq 0 [c_1 t_{pi_1} + c_2 t_{pi_2} \leq 0]} \quad \text{where } c_1, c_2 > 0$
4.  $\text{STRENGTHEN} \quad \frac{\sum_i c_i v_i + c \leq 0 [t_{pi} \leq 0]}{\sum_i c_i v_i + d \lceil \frac{c}{d} \rceil \leq 0 [t_{pi} + j \leq 0]}$

where  $d$  is the common divisor of  $c_i$   
 $0 \leq j \leq d \lceil \frac{c}{d} \rceil - c$

In the last rule, the derived PI formula ( $t_{pi} + j \leq 0$ ) records the  $\mathbf{A}'$ -contribution to a formula obtained jointly from  $\mathbf{A}'$  and  $\mathbf{B}'$  where  $\mathbf{A}' = \mathbf{A} \cup \{t_{pi} + j \leq 0\}$  and  $\mathbf{B}' = \mathbf{B} \cup \{\sum_i c_i v_i + d \lceil \frac{c}{d} \rceil - (t_{pi} + j) \leq 0\}$ . With this rule, we are branching on  $j$ , fixing it to a different constant value between  $[0, k]$  in each branch where  $k = d \lceil \frac{c}{d} \rceil - c$ . At the end, we will have  $k$  interpolants from  $k$  branches which are then combined together to produce the real interpolant for  $(\mathbf{A}, \mathbf{B})$ .

### 3.4.2 Fully interpolating rules

It is noticeable that in  $\mathbf{R}'$ , only rule **STRENGTHEN** introduces additional premises. As a result, we might end up obtaining  $(\mathbf{A}', \mathbf{B}') \vdash \perp [t_{pi} \leq 0]$  where  $\mathbf{A}'/\mathbf{B}'$  is the union of all formulae originally in  $\mathbf{A}/\mathbf{B}$  and additional premises added along the unsatisfiability proof. Therefore,  $t_{pi} \leq 0$  serves as the interpolant for  $(\mathbf{A}', \mathbf{B}')$  and not for  $(\mathbf{A}, \mathbf{B})$ .

Let  $\mathbf{A}^*/\mathbf{B}^*$  denote the sets of premises before applying rule **STRENGTHEN**,  $\mathbf{A}_k^j/\mathbf{B}_k^j$  denote the union of  $\mathbf{A}^*/\mathbf{B}^*$  and additional premises added by rule **STRENGTHEN**,  $t \leq 0$  denote the premise used in rule **STRENGTHEN**,  $k$  denote  $d \lceil \frac{c}{d} \rceil - c$ . In order to derive the interpolant for  $(\mathbf{A}^*, \mathbf{B}^*)$ , we need a rule to connect the interpolants of  $(\mathbf{A}_k^j, \mathbf{B}_k^j)$  and  $(\mathbf{A}^*, \mathbf{B}^*)$  as follows:

$$\text{k-CONN} \frac{\begin{array}{c} (\mathbf{A}^*, \mathbf{B}^*) \vdash t \leq 0 [t_{pi} \leq 0] \\ (\mathbf{A}_k^0, \mathbf{B}_k^0) \vdash \perp [\mathbf{I}_k^0] \\ \dots \\ (\mathbf{A}_k^k, \mathbf{B}_k^k) \vdash \perp [\mathbf{I}_k^k] \end{array}}{(\mathbf{A}^*, \mathbf{B}^*) \vdash \perp [\bigvee_{0 \leq j < k} (\mathbf{I}_k^j \wedge \mathbf{E}_k^j) \vee \mathbf{I}_k^k]}$$

where  $\mathbf{E}_k^j = \{t_{pi} + j = 0\};$   
 $\mathbf{A}_k^j = \{\mathbf{A}^*, t_{pi} + j \leq 0\};$   
 $\mathbf{B}_k^j = \{\mathbf{B}^*, t + k - (t_{pi} + j) \leq 0\}$

Note that  $t_{pi} + j = 0$  can still contain local symbols that occur only in  $\mathbf{A}^*$ . And when the formula does contain such symbols, it is necessary to introduce existential quantifiers to quantify  $\mathbf{A}^*$ -local symbols before combining the formula with other PIs. From now on, let's assume that  $t_{pi} + j = 0$  does not contain  $\mathbf{A}^*$ -local symbols, for the sake of brevity.

## 3.5 Soundness of partially interpolating rules

The soundness of rule **LEQEQ<sub>L</sub>**, **LEQEQ<sub>R</sub>** and **COMB** has been proved by [McMillan 2005]. To conclude the soundness of our partially interpolating rules, we only need to prove that rule **STRENGTHEN** is also sound.

In the **STRENGTHEN** rule, we have expanded  $\mathbf{A}$  to include the assumption of  $t_{pi} + j \leq 0$  and  $\mathbf{B}$  to include  $t + k - (t_{pi} + j) \leq 0$  where  $t = \sum_i c_i v_i + c$ , therefore  $t_{pi} + j \leq 0$  is obviously a partial interpolant for  $t + k \leq 0$  which is obtained jointly from the newly-expanded sets of premises.

## 3.6 Soundness of fully interpolating rules

### 3.6.1 1-CONN

Rule 1-CONN is the simplest instance of rule k-CONN when  $k = 1$ .

$$\text{1-CONN} \frac{\begin{array}{c} (\mathbf{A}^*, \mathbf{B}^*) \vdash t \leq 0 [t_{pi} \leq 0] \\ (\mathbf{A}_1^0, \mathbf{B}_1^0) \vdash \perp [\mathbf{I}_1^0] \\ (\mathbf{A}_1^1, \mathbf{B}_1^1) \vdash \perp [\mathbf{I}_1^1] \end{array}}{(\mathbf{A}^*, \mathbf{B}^*) \vdash \perp [(\mathbf{I}_1^0 \wedge \mathbf{E}_1^0) \vee \mathbf{I}_1^1]}$$

$$\begin{array}{l} \mathbf{E}_1^0 = \{t_{pi} = 0\}; \\ \text{where } \mathbf{A}_1^j = \{\mathbf{A}^*, t_{pi} + j \leq 0\}; \\ \mathbf{B}_1^j = \{\mathbf{B}^*, t + 1 - (t_{pi} + j) \leq 0\} \end{array}$$

We have  $(\mathbf{A}_e, \mathbf{B}_e) \vdash \perp [\mathbf{E}_1^0]$  where  $\mathbf{A}_e = \{\mathbf{A}^*, t_{pi} = 0\}$  and  $\mathbf{B}_e = \{\mathbf{B}^*, t - t_{pi} = 0\}$  because:

- (i)  $\mathbf{A}_e \vdash \mathbf{E}_1^0$
- (ii) Combining  $\mathbf{E}_1^0$  and  $t - t_{pi} = 0$  (from  $\mathbf{B}_e$ ), we obtain  $t = 0$  which is unsatisfiable as it is the premise of rule **STRENGTHEN** where  $k = 1$ . Hence,  $\mathbf{B}_e, \mathbf{E}_1^0 \vdash \perp$

Then  $\mathbf{A}^* \vdash \mathbf{I}_1^0 \wedge \mathbf{E}_1^0, \mathbf{I}_1^1$  is valid because  $\mathbf{A}^* \vdash \mathbf{E}_1^0, \mathbf{I}_1^1$  is valid:

$$\frac{\frac{\frac{\mathbf{A}_e \vdash \mathbf{E}_1^0}{\mathbf{A}_e \vdash \mathbf{E}_1^0, \mathbf{I}_1^1} \quad \frac{\mathbf{A}_1^1 \vdash \mathbf{I}_1^1}{\mathbf{A}_1^1 \vdash \mathbf{E}_1^0, \mathbf{I}_1^1}}{\mathbf{A}_e \vee \mathbf{A}_1^1 \vdash \mathbf{E}_1^0, \mathbf{I}_1^1}}{\frac{\{\mathbf{A}^*, t_{pi} = 0\} \vee \{\mathbf{A}^*, t_{pi} + 1 \leq 0\} \vdash \mathbf{E}_1^0, \mathbf{I}_1^1}{\mathbf{A}^*, t_{pi} \leq 0 \vdash \mathbf{E}_1^0, \mathbf{I}_1^1}}{\frac{\mathbf{A}^* \vdash t_{pi} \leq 0}{\mathbf{A}^* \vdash t_{pi} \leq 0, \mathbf{E}_1^0, \mathbf{I}_1^1}}}{\mathbf{A}^* \vdash \mathbf{E}_1^0, \mathbf{I}_1^1}$$

and  $\mathbf{A}^* \vdash \mathbf{I}_1^0, \mathbf{I}_1^1$  is valid:

$$\frac{\frac{\frac{\mathbf{A}_1^0 \vdash \mathbf{I}_1^0}{\mathbf{A}^*, t_{pi} \leq 0 \vdash \mathbf{I}_1^0} \quad \frac{\mathbf{A}^* \vdash t_{pi} \leq 0}{\mathbf{A}^* \vdash t_{pi} \leq 0, \mathbf{I}_1^0}}{\mathbf{A}^* \vdash \mathbf{I}_1^0}}{\mathbf{A}^* \vdash \mathbf{I}_1^0, \mathbf{I}_1^1}$$

Similarly  $\mathbf{B}^*, (\mathbf{I}_1^0 \wedge \mathbf{E}_1^0) \vee \mathbf{I}_1^1 \vdash \perp$  is valid because  $\mathbf{B}^*, \mathbf{E}_1^0, \mathbf{I}_1^0 \vdash \perp$  is valid:

$$\frac{\frac{\frac{\mathbf{B}_e, \mathbf{E}_1^0 \vdash \perp}{\mathbf{B}_e, \mathbf{E}_1^0, \mathbf{I}_1^0 \vdash \perp} \quad \frac{\mathbf{B}_1^0, \mathbf{I}_1^0 \vdash \perp}{\mathbf{B}_1^0, \mathbf{E}_1^0, \mathbf{I}_1^0 \vdash \perp}}{\mathbf{B}_e \vee \mathbf{B}_1^0, \mathbf{E}_1^0, \mathbf{I}_1^0 \vdash \perp}}{\frac{\{\mathbf{B}^*, t - t_{pi} = 0\} \vee \{\mathbf{B}^*, t + 1 - t_{pi} \leq 0\}, \mathbf{E}_1^0, \mathbf{I}_1^0 \vdash \perp}{\mathbf{B}^*, t - t_{pi} \leq 0, \mathbf{E}_1^0, \mathbf{I}_1^0 \vdash \perp}}{\mathbf{B}^*, \mathbf{E}_1^0, \mathbf{I}_1^0 \vdash t - t_{pi} \leq 0}} \frac{\mathbf{B}^* \vdash t - t_{pi} \leq 0}{\mathbf{B}^*, \mathbf{E}_1^0, \mathbf{I}_1^0 \vdash t - t_{pi} \leq 0}}{\mathbf{B}^*, \mathbf{E}_1^0, \mathbf{I}_1^0 \vdash \perp}$$

And  $\mathbf{B}^*, \mathbf{I}_1^1 \vdash \perp$  is valid:

$$\frac{\frac{\mathbf{B}_1^1, \mathbf{I}_1^1 \vdash \perp}{\{\mathbf{B}^*, t - t_{pi} \leq 0\}, \mathbf{I}_1^1 \vdash \perp}}{\mathbf{B}^*, \mathbf{I}_1^1 \vdash \perp}}{\frac{\mathbf{B}^* \vdash t - t_{pi} \leq 0}{\mathbf{B}^* \vdash t - t_{pi} \leq 0, \perp}}{\mathbf{B}^*, \mathbf{I}_1^1 \vdash \perp}}$$

As a result, we can affirm the soundness of rule 1-CONN.

### 3.6.2 1-CONN\*

From the premise  $(\mathbf{A}_1^0, \mathbf{B}_1^0) \vdash \perp [\mathbf{I}_1^0]$  used in rule 1-CONN, we can infer:

$$\langle \{\mathbf{A}_1^0, t_{pi} = 0\} \{\mathbf{B}_1^0\} \rangle \vdash \perp [\mathbf{I}_1^0]$$

Replacing the original premise with this slightly modified premise, we obtain rule 1-CONN\* which produces the same conclusion. This rule is introduced for later use in proving the soundness of the k-CONN rule in the next section.

$$1\text{-CONN}^* \frac{\begin{array}{l} (\mathbf{A}^*, \mathbf{B}^*) \vdash t \leq 0 [t_{pi} \leq 0] \\ \langle \{\mathbf{A}_1^0, t_{pi} = 0\}, \{\mathbf{B}_1^0\} \rangle \vdash \perp [\mathbf{I}_1^0] \\ (\mathbf{A}_1^1, \mathbf{B}_1^1) \vdash \perp [\mathbf{I}_1^1] \end{array}}{(\mathbf{A}^*, \mathbf{B}^*) \vdash \perp [(\mathbf{I}_1^0 \wedge \mathbf{E}_1^0) \vee \mathbf{I}_1^1]}$$

$$\begin{array}{l} \mathbf{E}_1^0 = \{t_{pi} = 0\}; \\ \text{where } \mathbf{A}_1^j = \{\mathbf{A}^*, t_{pi} + j \leq 0\}; \\ \mathbf{B}_1^j = \{\mathbf{B}^*, t + 1 - (t_{pi} + j) \leq 0\} \end{array}$$

The proof for the soundness of this rule follows the same line of that for rule 1-CONN in section 3.6.1.

### 3.6.3 k-CONN

In order to prove the soundness of rule k-CONN, we first prove the validity of the following induction:

$$(\mathbf{C}_n^j, \mathbf{D}_n^j) \vdash \perp[\mathbf{I}_k^j \wedge (n = 0 \vee \mathbf{E}_k^j)]$$

This induction then enables us to prove the second induction:

$$(\mathbf{F}_n^j, \mathbf{G}_n^j) \vdash \perp[\bigvee_{0 \leq q < n} (\mathbf{I}_k^{q+j} \wedge \mathbf{E}_k^{q+j}) \vee \mathbf{I}_k^{n+j}]$$

which finally allows us to conclude the soundness of rule k-CONN.

**a) First induction:**  $(\mathbf{C}_n^j, \mathbf{D}_n^j) \vdash \perp[\mathbf{I}_k^j \wedge (n = 0 \vee \mathbf{E}_k^j)]$

$$\begin{aligned} \mathbf{C}_n^j &= \{\mathbf{A}^*, t_{pi} + j \leq 0, t_{pi} + j = 0\}; \\ \text{Where: } \mathbf{D}_n^j &= \begin{cases} \mathbf{B}^*, \\ t + k - n - (t_{pi} + j) \leq 0, \\ \dots, \\ t - (t_{pi} + j) \leq 0 \end{cases} \end{aligned}$$

For all  $n$  s.t.  $0 \leq n \leq k$ , we prove the induction is valid for all  $j$ .

**Base case:** since  $(\mathbf{A}_k^j, \mathbf{B}_k^j) \vdash \perp[\mathbf{I}_k^j]$  (which is a premise of k-CONN) we can infer:

$$\begin{aligned} &\{\mathbf{A}_k^j, t_{pi} + j = 0\}, \{\mathbf{B}_k^j, t + k - (t_{pi} + j) \leq 0, \dots, t - (t_{pi} + j) \leq 0\} \vdash \perp[\mathbf{I}_k^j] \text{ or} \\ &\{\mathbf{A}^*, t_{pi} + j \leq 0, t_{pi} + j = 0\}, \{\mathbf{B}_k^j, t + k - (t_{pi} + j) \leq 0, \dots, t - (t_{pi} + j) \leq 0\} \vdash \perp[\mathbf{I}_k^j] \\ &\text{which is } (\mathbf{C}_0^j, \mathbf{D}_0^j) \vdash \perp[\mathbf{I}_k^j] \end{aligned}$$

**Step case:** assuming that the induction is valid up to  $n < k$ , we apply the 1-CONN rule to prove the induction is also valid for  $n + 1$ .

$$\begin{array}{c} (\mathbf{C}_{n+1}^j, \mathbf{D}_{n+1}^j) \vdash t + k - n - 1 \leq 0[t_{pi} + j \leq 0] \\ (\mathbf{C}_{n1}^{j0}, \mathbf{D}_{n1}^{j0}) \vdash \perp[\mathbf{I}_{n1}^{j0}] \\ (\mathbf{C}_{n1}^{j1}, \mathbf{D}_{n1}^{j1}) \vdash \perp[\mathbf{I}_{n1}^{j1}] \\ \hline \text{1-CONN} \quad (\mathbf{C}_{n+1}^j, \mathbf{D}_{n+1}^j) \vdash \perp[(\mathbf{I}_{n1}^{j0} \wedge \mathbf{E}_{n1}^{j0}) \vee \mathbf{I}_{n1}^{j1}] \end{array}$$

$$\begin{aligned} \text{where } \mathbf{E}_{n1}^{j0} &= \{t_{pi} + j = 0\}; \\ \mathbf{C}_{n1}^{jh} &= \{\mathbf{C}_{n+1}^j, t_{pi} + j + h \leq 0\}, \quad (0 \leq h \leq 1); \\ \mathbf{D}_{n1}^{jh} &= \{\mathbf{D}_{n+1}^j, t + k - n - (t_{pi} + j + h) \leq 0\} \end{aligned}$$

Notice that  $\mathbf{E}_{n0}^{j0} \equiv \mathbf{E}_k^j \equiv (t_{pi} + j = 0)$ . In addition  $\mathbf{I}_{n1}^{j0} \equiv (\mathbf{I}_k^j \wedge (n = 0 \vee \mathbf{E}_k^j))$  because the inductive step  $n$  has been proved valid and:

- $\mathbf{C}_{n1}^{j_0} \equiv \{\mathbf{C}_{n+1}^j, t_{pi} + j \leq 0\} \equiv \mathbf{C}_n^j$
- $\mathbf{D}_{n1}^{j_0} = \{\mathbf{D}_{n+1}^j, t + k - n - (t_{pi} + j) \leq 0\} \equiv \mathbf{D}_n^j$

Also  $\mathbf{I}_{n1}^{j_1} \equiv \perp$  because  $\mathbf{C}_{n1}^{j_1}$  contains  $t_{pi} + j = 0$  and  $t_{pi} + j + 1 \leq 0$  which is a contradiction. Since  $n > 0$  in this inductive step:

$$\begin{aligned} (\mathbf{I}_{n1}^{j_0} \wedge \mathbf{E}_{n1}^{j_0}) \vee \mathbf{I}_{n1}^{j_1} &= [\mathbf{I}_k^j \wedge ((n = 0) \vee \mathbf{E}_k^j)] \wedge \mathbf{E}_k^j \\ &= \mathbf{I}_k^j \wedge \mathbf{E}_k^j \\ &= \mathbf{I}_k^j \wedge ((n + 1) = 0 \vee \mathbf{E}_k^j) \end{aligned}$$

b) **Second induction:**  $(\mathbf{F}_n^j, \mathbf{G}_n^j) \vdash \perp [\bigvee_{0 \leq q < n} (\mathbf{I}_k^{q+j} \wedge \mathbf{E}_k^{q+j}) \vee \mathbf{I}_k^{n+j}]$

$$\begin{aligned} \mathbf{F}_n^j &= \{\mathbf{A}^*, t_{pi} + j \leq 0\}; \\ \text{Where: } \mathbf{G}_n^j &= \begin{cases} \mathbf{B}^*, \\ t + k - n - (t_{pi} + j) \leq 0, \\ \dots, \\ t - (t_{pi} + j) \leq 0 \end{cases} \end{aligned}$$

For all  $n$  s.t.  $0 \leq n \leq k$ , we prove the induction is valid for all  $j$ .

**Base case:** since  $(\mathbf{A}_k^j, \mathbf{B}_k^j) \vdash \perp [\mathbf{I}_k^j]$  (which is a premise of  $k$ -CONN) we can infer:

$$\begin{aligned} &\{\mathbf{A}_k^j\}, \{\mathbf{B}_k^j, t + k - (t_{pi} + j) \leq 0, \dots, t - (t_{pi} + j) \leq 0\} \vdash \perp [\mathbf{I}_k^j] \text{ or} \\ &\{\mathbf{A}^*, t_{pi} + j \leq 0\}, \{\mathbf{B}_k^j, t + k - (t_{pi} + j) \leq 0, \dots, t - (t_{pi} + j) \leq 0\} \vdash \perp [\mathbf{I}_k^j] \text{ or} \\ &(\mathbf{F}_0^j, \mathbf{G}_0^j) \vdash \perp [\mathbf{I}_k^j] \end{aligned}$$

**Step case:** assuming that the induction is valid up to  $n < k$ , we apply the 1-CONN\* rule to prove the induction is also valid for  $n + 1$ .

$$\text{1-CONN*} \frac{\begin{array}{c} (\mathbf{F}_{n+1}^j, \mathbf{G}_{n+1}^j) \vdash t + k - (n + 1) \leq 0 [t_{pi} + j \leq 0] \\ \langle \{\mathbf{F}_{n1}^{j_0}, t_{pi} + j = 0\}, \{\mathbf{G}_{n1}^{j_0}\} \rangle \vdash \perp [\mathbf{I}_{n1}^{j_0}] \\ (\mathbf{F}_{n1}^{j_1}, \mathbf{G}_{n1}^{j_1}) \vdash \perp [\mathbf{I}_{n1}^{j_1}] \end{array}}{(\mathbf{F}_{n+1}^j, \mathbf{G}_{n+1}^j) \vdash \perp [(\mathbf{I}_{n1}^{j_0} \wedge \mathbf{E}_{n1}^{j_0}) \vee \mathbf{I}_{n1}^{j_1}]}$$

$$\begin{aligned} \text{where } \mathbf{E}_{n1}^{j_0} &= \{t_{pi} + j = 0\}; \\ \mathbf{F}_{n1}^{j_h} &= \{\mathbf{F}_{n+1}^j, t_{pi} + j + h \leq 0\}, \quad (0 \leq h \leq 1); \\ \mathbf{G}_{n1}^{j_h} &= \{\mathbf{G}_{n+1}^j, t + k - n - (t_{pi} + j + h) \leq 0\} \end{aligned}$$

Notice that  $\mathbf{E}_{n1}^{j_0} \equiv \mathbf{E}_k^j \equiv (t_{pi} + j = 0)$ . In addition  $\mathbf{I}_{n1}^{j_0} \equiv [\mathbf{I}_k^j \wedge (n = 0 \vee \mathbf{E}_k^j)]$  because of the first induction proved previously and:

- $\{\mathbf{F}_{n1}^{j_0}, t_{pi} + j = 0\} \equiv \{\mathbf{F}_{n+1}^j, t_{pi} + j \leq 0, t_{pi} + j = 0\} \equiv \mathbf{C}_n^j$
- $\mathbf{G}_{n1}^{j_0} = \{\mathbf{G}_{n+1}^j, t + k - n - (t_{pi} + j) \leq 0\} \equiv \mathbf{D}_n^j$

Also  $\mathbf{I}_{n1}^{j_1} \equiv [\bigvee_{0 \leq q < n} (\mathbf{I}_k^{q+j+1} \wedge \mathbf{E}_k^{q+j+1}) \vee \mathbf{I}_k^{n+j+1}]$  because the inductive step  $n$  has been proved valid for all  $j'$  which in this particular case is  $j + 1$ :

- $\mathbf{F}_{n1}^{j_1} = \{\mathbf{F}_{n+1}^j, t_{pi} + j + 1 \leq 0\} \equiv \{\mathbf{F}_n^{j+1}, t_{pi} + j \leq 0\} \equiv \{\mathbf{F}_n^{j'}, t_{pi} + j \leq 0\}$
- $\mathbf{G}_{n1}^{j_1} = \{\mathbf{G}_{n+1}^j, t + k - n - (t_{pi} + j + 1) \leq 0\} \equiv \mathbf{G}_n^{j+1} \equiv \mathbf{G}_n^{j'}$

Since  $n > 0$  in this inductive step:

$$\begin{aligned}
(\mathbf{I}_{n1}^{j_0} \wedge \mathbf{E}_{n1}^{j_0}) \vee \mathbf{I}_{n1}^{j_1} &= [\mathbf{I}_k^j \wedge ((n = 0) \vee \mathbf{E}_k^j)] \wedge \mathbf{E}_k^j \vee \mathbf{I}_{n1}^{j_1} \\
&= (\mathbf{I}_k^j \wedge \mathbf{E}_k^j) \vee \bigvee_{0 \leq q < n} (\mathbf{I}_k^{q+j+1} \wedge \mathbf{E}_k^{q+j+1}) \vee \mathbf{I}_k^{n+j+1} \\
&= \bigvee_{0 \leq q < n+1} (\mathbf{I}_k^{q+j} \wedge \mathbf{E}_k^{q+j}) \vee \mathbf{I}_k^{(n+1)+j}
\end{aligned}$$

The soundness of rule k-CONN is proved when applying the second induction for  $n = k$  and  $j = 0$ :

$$(\mathbf{F}_k^0, \mathbf{G}_k^0) \vdash \perp [\bigvee_{0 \leq q < k} (\mathbf{I}_k^q \wedge \mathbf{E}_k^q) \vee \mathbf{I}_k^k] \text{ where } \mathbf{F}_k^0 \equiv \mathbf{A}^*, \mathbf{G}_k^0 \equiv \mathbf{B}^*$$

## 3.7 Examples

### 3.7.1 With one STRENGTHEN application

Consider the following sets of  $\mathcal{LA}(\mathbb{Z})$  atoms:

$$\begin{aligned}
\mathbf{A} &= \{-y - 4x - 1 \leq 0, y + 4x \leq 0\} \\
\mathbf{B} &= \{y + 4z - 2 \leq 0, -y - 4z + 1 \leq 0\}
\end{aligned}$$

An  $\mathbf{R}'$ -proof of unsatisfiability  $P$  for  $\mathbf{A} \wedge \mathbf{B}$  is the following:

$$\frac{\frac{\frac{-y - 4x - 1 \leq 0}{-y - 4x - 1 \leq 0} (1) \quad \frac{y + 4z - 2 \leq 0}{y + 4z - 2 \leq 0} (2)}{4z - 4x - 3 \leq 0} (3)}{4z - 4x \leq 0} (4) \quad \frac{\frac{y + 4x \leq 0}{y + 4x \leq 0} (1) \quad \frac{-y - 4z + 1 \leq 0}{-y - 4z + 1 \leq 0} (2)}{4x - 4z + 1 \leq 0} (3)}{1 \leq 0} (3)$$

The numbers in brackets denote  $\mathbf{R}'$ -rules (i.e. (1) for  $\text{LEQEQ}_L$ , (2) for

LEQEQ<sub>R</sub>, etc.). In the above proof, we use a STRENGTHEN rule with  $k = 3$  so the corresponding PIs-tree is as follows:

$$\frac{\frac{\frac{-y - 4x - 1 \leq 0}{(1)} \quad \frac{0 \leq 0}{(2)}}{-y - 4x - 1 \leq 0} \quad (3) \quad \frac{\frac{y + 4x \leq 0}{(1)} \quad \frac{0 \leq 0}{(2)}}{y + 4x \leq 0} \quad (3)}{\frac{-y - 4x - 1 + j \leq 0}{(4)} \quad \frac{y + 4x \leq 0}{(3)}}{j - 1 \leq 0} \quad (3)$$

Because we were branching on  $j$ ,  $\mathbf{E}_k^j = -y - 4x - 1 + j$  and the interpolant obtained in each branch is  $\mathbf{I}_k^j = \{j - 1 \leq 0\}$ . The final interpolant for  $(\mathbf{A}, \mathbf{B})$  is:

$$\begin{aligned} \bigvee_{0 \leq j < k} (\mathbf{I}_k^j \wedge \mathbf{E}_k^j) \vee \mathbf{I}_k^k &= (\mathbf{I}_3^0 \wedge \mathbf{E}_3^0) \vee (\mathbf{I}_3^1 \wedge \mathbf{E}_3^1) \vee (\mathbf{I}_3^2 \wedge \mathbf{E}_3^2) \vee \mathbf{I}_3^3 \\ &= (\top \wedge \mathbf{E}_3^0) \vee (\top \wedge \mathbf{E}_3^1) \vee (\perp \wedge \mathbf{E}_3^2) \vee \perp = \mathbf{E}_3^0 \vee \mathbf{E}_3^1 \\ &= (-y - 4x - 1 = 0) \vee (-y - 4x - 1 + 1 = 0) \\ &= (y + 1 = -4x) \vee (y = -4x) \\ &= \bigvee_{0 \leq j \leq 1} 4|(y + j) \end{aligned}$$

### 3.7.2 With two or more STRENGTHEN applications

Consider the following sets of  $\mathcal{LA}(\mathbb{Z})$  atoms:

$$\begin{aligned} \mathbf{A} &= \{-y - 4x - 1 \leq 0, y + 4x + 1 \leq 0\} \\ \mathbf{B} &= \{y + 4z - 2 \leq 0, -y - 4z + 1 \leq 0\} \end{aligned}$$

An  $\mathbf{R}'$ -proof of unsatisfiability  $P$  for  $\mathbf{A} \wedge \mathbf{B}$  is the following:

$$\frac{\frac{\frac{-y - 4x - 1 \leq 0}{(1)} \quad \frac{y + 4z - 2 \leq 0}{(2)}}{-y - 4x - 1 \leq 0} \quad (3) \quad \frac{\frac{y + 4x + 1 \leq 0}{(1)} \quad \frac{-y - 4z + 1 \leq 0}{(2)}}{y + 4x + 1 \leq 0} \quad (3)}{\frac{4z - 4x - 3 \leq 0}{(4)} \quad \frac{4x - 4z + 2 \leq 0}{(4)}}{4z - 4x \leq 0} \quad (4) \quad \frac{4x - 4z + 4 \leq 0}{(4)}}{1 \leq 0} \quad (3)$$

In the above proof, we use two STRENGTHEN applications with  $k = 3$  and  $k' = 2$  so the corresponding PIs-tree is as follows:

$$\frac{\frac{\frac{-y - 4x - 1 \leq 0}{(1)} \quad \frac{0 \leq 0}{(2)}}{-y - 4x - 1 \leq 0} \quad (3) \quad \frac{\frac{y + 4x + 1 \leq 0}{(1)} \quad \frac{0 \leq 0}{(2)}}{y + 4x + 1 \leq 0} \quad (3)}{\frac{-y - 4x - 1 \leq 0}{(4)} \quad \frac{y + 4x + 1 \leq 0}{(4)}}{j + j' \leq 0} \quad (3)$$

Assuming that we were branching on  $j$  first and then on  $j'$ , the interpolant obtained in every combined branch is  $\mathbf{I}_{kk'}^{jj'} = \{j + j' \leq 0\}$ . We write  $jj'$  to denote the order of applying STRENGTHEN rules on different nodes or the branching order on auxiliary variables  $j, j'$ . Moreover,  $\mathbf{E}_k^j = -y - 4x - 1 + j$  and  $\mathbf{E}_{kk'}^{jj'} = y + 4x + 1 + j'$ . We will first combine  $k'$  interpolants  $(\mathbf{I}_{kk'}^j, \dots, \mathbf{I}_{kk'}^{j'})$  obtained in every  $j$ -branch and then combine  $k$  interpolants  $(\mathbf{I}_k^0, \dots, \mathbf{I}_k^k)$  from the  $j$ -branches to produce the final interpolant for  $(\mathbf{A}, \mathbf{B})$ . Let  $(\mathbf{A}_k^j, \mathbf{B}_k^j)$  be the set of premises after branching on  $j$ , then the interpolant for  $(\mathbf{A}_k^j, \mathbf{B}_k^j)$  is:

$$\begin{aligned} \mathbf{I}_k^j &= \bigvee_{0 \leq j' < k'} (\mathbf{I}_{kk'}^{jj'} \wedge \mathbf{E}_{kk'}^{jj'}) \vee \mathbf{I}_{kk'}^{jk'} \\ &= [(j \leq 0 \wedge y + 4x + 1 = 0) \vee (j + 1 \leq 0 \wedge y + 4x + 2 = 0) \vee (j + 2 \leq 0)] \end{aligned}$$

Combining all  $\mathbf{I}_k^j$ , we obtain the interpolant for  $(\mathbf{A}, \mathbf{B})$ :

$$\begin{aligned} \bigvee_{0 \leq j < k} (\mathbf{I}_k^j \wedge \mathbf{E}_k^j) \vee \mathbf{I}_k^k &= (\mathbf{I}_3^0 \wedge \mathbf{E}_3^0) \vee (\mathbf{I}_3^1 \wedge \mathbf{E}_3^1) \vee (\mathbf{I}_3^2 \wedge \mathbf{E}_3^2) \vee \mathbf{I}_3^3 \\ &= (y + 4x + 1 = 0 \wedge \mathbf{E}_3^0) \vee (\perp \wedge \mathbf{E}_3^1) \vee (\perp \wedge \mathbf{E}_3^2) \vee \perp \\ &= (y + 4x + 1 = 0) \wedge (-y - 4x - 1 = 0) \\ &= (y + 1 = -4x) \\ &= 4|(y + 1) \end{aligned}$$

In general, assuming that we use  $n$  STRENGTHEN applications in our proof. Then our interpolant obtained in each branch is  $\mathbf{I}_{k_1 \dots k_n}^{j_1 \dots j_n}$ . To compute the interpolant for  $(\mathbf{A}, \mathbf{B})$ , we could follow the same procedure as described above. That is, we combine  $k_i$  (where  $1 \leq i \leq n$ ) interpolants  $\mathbf{I}_{k_1 \dots k_i}^{j_1 \dots 0}, \dots, \mathbf{I}_{k_1 \dots k_i}^{j_1 \dots (k_i - 1)}$  together to obtain  $\mathbf{I}_{k_1 \dots k_{i-1}}^{j_1 \dots j_{i-1}}$ :

$$\mathbf{I}_{k_1 \dots k_{i-1}}^{j_1 \dots j_{i-1}} = \bigvee_{0 \leq j_i < k_i} (\mathbf{I}_{k_1 \dots k_i}^{j_1 \dots j_i} \wedge \mathbf{E}_{k_1 \dots k_i}^{j_1 \dots j_i}) \vee \mathbf{I}_{k_1 \dots k_i}^{j_1 \dots k_i}$$

In the end, the interpolant for  $(\mathbf{A}, \mathbf{B})$  is  $\bigvee_{0 \leq j < k} (\mathbf{I}_k^j \wedge \mathbf{E}_k^j) \vee \mathbf{I}_k^k$ . However, if one wants to generate the interpolant directly by using  $\mathbf{I}_{k_1 \dots k_n}^{j_1 \dots j_n}$ , there is another way as well which does not require indirect interpolation computation as above. We notice that:

$$\begin{aligned} \mathbf{I}_{k_1 \dots k_{i-1}}^{j_1 \dots j_{i-1}} \wedge \mathbf{E}_{k_1 \dots k_{i-1}}^{j_1 \dots j_{i-1}} &= \left[ \bigvee_{0 \leq j_i < k_i} (\mathbf{I}_{k_1 \dots k_i}^{j_1 \dots j_i} \wedge \mathbf{E}_{k_1 \dots k_i}^{j_1 \dots j_i}) \vee \mathbf{I}_{k_1 \dots k_i}^{j_1 \dots k_i} \right] \wedge \mathbf{E}_{k_1 \dots k_{i-1}}^{j_1 \dots j_{i-1}} \\ &= \bigvee_{0 \leq j_i < k_i} (\mathbf{I}_{k_1 \dots k_i}^{j_1 \dots j_i} \wedge \mathbf{E}_{k_1 \dots k_{i-1}}^{j_1 \dots j_{i-1}} \wedge \mathbf{E}_{k_1 \dots k_i}^{j_1 \dots j_i}) \vee (\mathbf{I}_{k_1 \dots k_i}^{j_1 \dots k_i} \wedge \mathbf{E}_{k_1 \dots k_{i-1}}^{j_1 \dots j_{i-1}}) \end{aligned}$$

Hence:

$$\begin{aligned}
\mathbf{I}_{k_1 \dots k_{i-2}}^{j_1 \dots j_{i-2}} &= \bigvee_{0 \leq j_{i-1} < k_{i-1}} \left[ \bigvee_{0 \leq j_i < k_i} (\mathbf{I}_{k_1 \dots k_i}^{j_1 \dots j_i} \wedge \mathbf{E}_{k_1 \dots k_{i-1}}^{j_1 \dots j_{i-1}} \wedge \mathbf{E}_{k_1 \dots k_i}^{j_1 \dots j_i}) \vee (\mathbf{I}_{k_1 \dots k_i}^{j_1 \dots k_i} \wedge \mathbf{E}_{k_1 \dots k_{i-1}}^{j_1 \dots j_{i-1}}) \right] \vee \mathbf{I}_{k_1 \dots k_{i-1}}^{j_1 \dots k_{i-1}} \\
&= \bigvee_{0 \leq j_{i-1} < k_{i-1}} \left[ \bigvee_{0 \leq j_i < k_i} (\mathbf{I}_{k_1 \dots k_i}^{j_1 \dots j_i} \wedge \mathbf{E}_{k_1 \dots k_{i-1}}^{j_1 \dots j_{i-1}} \wedge \mathbf{E}_{k_1 \dots k_i}^{j_1 \dots j_i}) \vee (\mathbf{I}_{k_1 \dots k_i}^{j_1 \dots k_i} \wedge \mathbf{E}_{k_1 \dots k_{i-1}}^{j_1 \dots j_{i-1}}) \right] \vee \\
&\quad \bigvee_{0 \leq j_i < k_i} (\mathbf{I}_{k_1 \dots k_{i-1} k_i}^{j_1 \dots k_{i-1} j_i} \wedge \mathbf{E}_{k_1 \dots k_{i-1} k_i}^{j_1 \dots k_{i-1} j_i}) \vee \mathbf{I}_{k_1 \dots k_{i-1} k_i}^{j_1 \dots k_{i-1} k_i}
\end{aligned}$$

The above formula is a disjunction of many conjunctions. Each conjunction is characterized by a set of values for  $(j_1, \dots, j_i)$ . We notice that  $\mathbf{E}_{k_1 \dots k_i}^{j_1 \dots j_i}$  does not appear in conjunctions where  $j_i = k_i$ . Similarly,  $\mathbf{E}_{k_1 \dots k_{i-1}}^{j_1 \dots j_{i-1}}$  does not appear in conjunctions where  $j_{i-1} = k_{i-1}$ . Therefore, as we goes on, we will see that neither does  $\mathbf{E}_{k_1 \dots k_{i-2}}^{j_1 \dots j_{i-2}}$  appear in conjunctions where  $j_{i-2} = k_{i-2}$ . Generalizing this fact, we would obtain the interpolant for  $(\mathbf{A}, \mathbf{B})$  as follows:

$$\mathbf{I} = \bigvee_{\substack{0 \leq j_1 \leq k_1 \\ \dots \\ 0 \leq j_n \leq k_n}} [\mathbf{I}_{k_1 \dots k_n}^{j_1 \dots j_n} \bigwedge_{1 \leq i \leq n} (\text{not\_appear}(j_1, \dots, j_i) \vee \mathbf{E}_{k_1 \dots k_i}^{j_1 \dots j_i})]$$

where:

- $\mathbf{I}_{k_1 \dots k_n}^{j_1 \dots j_n}$  is the interpolant obtained in each branch
- $\mathbf{E}_{k_1 \dots k_i}^{j_1 \dots j_i}$  is the PI equality of the conclusion of rule **STRENGTHEN**
- $\text{not\_appear}(j_1, \dots, j_i) = \begin{cases} \top & \text{if } j_i = k_i \\ \perp & \text{if } j_i < k_i \end{cases}$

## Chapter 4

# Implementation and Empirical results

In this chapter, we describe our approach in computing interpolants for conjunctions of  $\mathcal{LA}(\mathbb{Z})$ -equalities and weak inequalities. We then give several examples to illustrate our interpolation algorithm. At the end of this chapter, we present some preliminary results of our prototype tool on the RINGS benchmark.

# Chapter 5

## Conclusions

### 5.1 Conclusion and future work

# Bibliography

- [1] Clark Barrett, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Splitting on demand in sat modulo theories. In Miki Hermann and Andrei Voronkov, editors, *Proceedings of the 13<sup>th</sup> International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR '06)*, volume 4246 of *Lecture Notes in Computer Science*, pages 512–526. Springer-Verlag, nov 2006. Phnom Penh, Cambodia.
- [2] Dirk Beyer, Damien Zufferey, and Rupak Majumdar. Csisat: Interpolation for la+euf. In Aarti Gupta and Sharad Malik, editors, *CAV*, volume 5123 of *Lecture Notes in Computer Science*, pages 304–308. Springer, 2008.
- [3] Angelo Brillout, Daniel Kroening, Philipp Rümmer, and Thomas Wahl. An interpolating sequent calculus for quantifier-free presburger arithmetic. In Jürgen Giesl and Reiner Hähnle, editors, *Proceedings, International Joint Conference on Automated Reasoning (IJCAR)*, Incs. spv, 2010. To appear.
- [4] Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani. Efficient interpolant generation in satisfiability modulo theories. In Orna Grumberg and Michael Huth, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 397–412. Springer, 2008.
- [5] Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani. Interpolant generation for utvpi. In Renate A. Schmidt, editor, *CADE*, volume 5663 of *Lecture Notes in Computer Science*, pages 167–182. Springer, 2009.
- [6] Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani. Efficient generation of Craig interpolants in satisfiability modulo theories. *ACM Transactions on Computational Logic (TOCL)*, To Appear.
- [7] Vijay D'Silva, Daniel Kroening, Mitra Purandare, and Georg Weissenbacher. Interpolant strength. In Gilles Barthe and Manuel V.

- Hermenegildo, editors, *VMCAI*, volume 5944 of *Lecture Notes in Computer Science*, pages 129–145. Springer, 2010.
- [8] R.E. Gomory. An algorithm for integer solutions of linear programs. *Recent Advances in Mathematical Programming*, pages 269–302, 1963.
  - [9] Himanshu Jain, Edmund M. Clarke, and Orna Grumberg. Efficient craig interpolation for linear diophantine (dis)equations and linear modular equations. *Formal Methods in System Design*, 35(1):6–39, 2009.
  - [10] Lahiri and Bryant. Deductive verification of advanced out-of-order microprocessors. In *CAV: International Conference on Computer Aided Verification*, 2003.
  - [11] McMillan. Interpolation and sat-based model checking. In *CAV: International Conference on Computer Aided Verification*, 2003.
  - [12] McMillan. An interpolating theorem prover. *TCS: Theoretical Computer Science*, 345, 2005.
  - [13] Kenneth L. McMillan. Applications of craig interpolants in model checking. In Nicolas Halbwachs and Lenore D. Zuck, editors, *TACAS*, volume 3440 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005.
  - [14] G. Nelson. Extended static checking for java. *Lecture Notes in Computer Science*, 3125:1, 2004.
  - [15] Pavel Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *Journal of Symbolic Logic*, 62(3):981–998, sep 1997.
  - [16] Andrey Rybalchenko and Viorica Sofronie-Stokkermans. Constraint solving for interpolation. In Byron Cook and Andreas Podelski, editors, *VMCAI*, volume 4349 of *Lecture Notes in Computer Science*, pages 346–362. Springer, 2007.
  - [17] Roberto Sebastiani. Lazy satisfiability modulo theories. *JSAT*, 3(3-4):141–224, 2007.
  - [18] Roberto Sebastiani and Armando Tacchella. Sat techniques for modal and description logics. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185, chapter Frontiers in Artificial Intelligence and Applications, pages 781–824. IOS Press, 2009.

- [19] Viorica Sofronie-Stokkermans. Interpolation in local theory extensions. In Ulrich Furbach and Natarajan Shankar, editors, *IJCAR*, volume 4130 of *Lecture Notes in Computer Science*, pages 235–250. Springer, 2006.
- [20] Greta Yorsh and Madanlal Musuvathi. A combination method for generating interpolants. In Robert Nieuwenhuis, editor, *CADE*, volume 3632 of *Lecture Notes in Computer Science*, pages 353–368. Springer, 2005.