Interpolant Generation for UTVPI [⋆]

Alessandro Cimatti[1], Alberto Griggio[2], and Roberto Sebastiani[2]

[1] FBK-Irst, Trento, Italy. cimatti@fbk.eu
[2] DISI, Università di Trento, Italy. {griggio,rseba}@disi.unitn.it

Abstract. The problem of computing Craig interpolants in SMT has recently received a lot of interest, mainly for its applications in formal verification. Efficient algorithms for interpolant generation have been presented for some theories of interest –including that of equality and uninterpreted functions ($\mathcal{EUF}$), linear arithmetic over the rationals ($\mathcal{LA}(\mathbb{Q})$), and some fragments of linear arithmetic over the integers ($\mathcal{LA}(\mathbb{Z})$)– and they are successfully used within model checking tools.
In this paper we address the problem of computing interpolants in the theory of Unit-Two-Variable-Per-Inequality ($\mathcal{UTVPI}$). This theory is a very useful fragment of $\mathcal{LA}(\mathbb{Z})$, since it is expressive enough to encode many hardware and software verification queries while still admitting a polynomial time decision procedure.
We present an efficient graph-based algorithm for interpolant generation in $\mathcal{UTVPI}$, which exploits the power of modern SMT techniques. We have implemented our new algorithm within the MathSAT SMT solver. Our experimental evaluation demonstrates both the efficiency and the usefulness of the new algorithm.

1 Motivations and goals

Given two formulas A and B such that $A \wedge B$ is inconsistent, a *Craig interpolant* (simply "interpolant" hereafter) for (A, B) is a formula I s.t. A entails I, $I \wedge B$ is inconsistent, and all uninterpreted symbols of I occur in both A and B. Since the seminal work of McMillan [17], interpolation has been recognized to be a substantial tool for formal verification. For instance, in the context of software model checking based on counter-example-guided-abstraction-refinement (CEGAR), interpolants of quantifier-free formulas in suitable theories are computed for automatically refining abstractions in order to rule out spurious counterexamples (see, e.g. [8, 11, 19]). This technique is used by state-of-the-art software model checkers like, e.g., Blast [2] and Impact [19]. Consequently, the problem of computing interpolants in SMT has received a lot of interest in the last years (e.g., [18, 26, 12, 22, 13, 5, 10, 3, 14]).

In the recent years, efficient algorithms and tools for interpolant generation for quantifier-free formulas in SMT have been presented for some theories of

⋆ The first author is partly supported by the European Commission under project FP7-2007-IST-1-217069 COCONUT. The second and third authors are partly supported by SRC under GRC Custom Research Project 2009-TJ-1880 WOLFLING, and by MIUR under PRIN project 20079E5KM8_002.

interest, including that of equality and uninterpreted functions ($\mathcal{EUF}$) [18, 14], linear arithmetic over the rationals ($\mathcal{LA}(\mathbb{Q})$) [18, 22, 5], and for their combination [26, 25, 5, 3]. In many applications, however, the domain of rational numbers is often inadequate for representing variables, which could be represented much more precisely in the integer domain (see, e.g., [7, 15]). Unfortunately, the computation of interpolants in the theory of linear arithmetic over the integers ($\mathcal{LA}(\mathbb{Z})$) raises many more problems than $\mathcal{LA}(\mathbb{Q})$, and in fact there is no known and efficient algorithm for computing interpolants in $\mathcal{LA}(\mathbb{Z})$. The only known algorithm is based on quantifier elimination, which is typically prohibitively expensive, and also requires the introduction of divisibility predicates. Therefore, it is useful to investigate interpolation for *fragments* of $\mathcal{LA}(\mathbb{Z})$, simple enough to be treated efficiently, although general enough to allow for encoding a significant amount of verification problems. To this extent, Jain, Clarke and Grunberg [10] proposed efficient algorithms for computing interpolants for conjunctions of linear Diophantine equations and disequations and for conjunctions of linear modular equations, and showed that these algorithms enabled the verification of simple programs which could not be previously checked by CEGAR-based model checkers.

In this paper, we move along the same track of Jain et al. [10], and we tackle the problem of computing interpolants in another important fragment of $\mathcal{LA}(\mathbb{Z})$, the theory of Unit-Two-Variable-Per-Inequality ($\mathcal{UTVPI}$). In $\mathcal{UTVPI}$ a formula is a Boolean combination of atoms in the form $(0 \leq ax_1 + bx_2 + k)$, where x_i are variables over $\mathbb{Z}$, k is an integer constant, and $a, b \in \{-1, 0, 1\}$. $\mathcal{UTVPI}$ is a very interesting theory: it generalizes the well-known Difference Logic ($\mathcal{DL}(\mathbb{Z})$) (where the a and b coefficient are forced to the 1 and -1 values, respectively), and it is one of the most expressive fragments of $\mathcal{LA}(\mathbb{Z})$ with a polynomial decision procedure [9]. (In fact, it is sufficient to extend the fragment to contain three unit variables, or to add non-unit coefficients, to make the decision problems NP-complete.) $\mathcal{UTVPI}$ is also a very useful fragment of $\mathcal{LA}(\mathbb{Z})$, since it allows to naturally express the queries occurring in many hardware and software verification problems [1, 24].

The problem of satisfiability modulo $\mathcal{UTVPI}$ can be tackled following the approach proposed in [20, 16], where the consistency check of a conjunction of $\mathcal{UTVPI}$ constraints is based on an encoding into $\mathcal{DL}$. This allows for the use of very efficient graph-based decision procedures for $\mathcal{DL}$ [21, 6]: these algorithms have a $O(n \cdot m)$ time complexity for problems with n variables and m constraints, and are extremely fast in practice. In addition, they have all the features required for a tight integration within a modern SMT solver: incrementalitly and backtrackability, construction of minimal conflict sets, and deduction of unassigned literals (see [23] for a survey).

The contribution of this paper is the first interpolation algorithm for $\mathcal{UTVPI}$. The algorithm follows the decision procedure for $\mathcal{UTVPI}$, working in two phases. In the first phase, it checks whether the conjunction of $\mathcal{UTVPI}$ constraints is inconsistent in the rational domain ($\mathcal{UTVPI}(\mathbb{Q})$). If so, an interpolant is obtained with a generalization of the graph-based algorithm for $\mathcal{DL}$ [5]. The second phase is entered if the problem is consistent in the rational domain, but inconsistent in the integer domain. The second phase is also based on the analysis of the graph resulting from the encoding into $\mathcal{DL}$. However, unlike with $\mathcal{DL}(\mathbb{Q})$ and $\mathcal{DL}(\mathbb{Z})$, the problem of interpolant generation on $\mathcal{UTVPI}(\mathbb{Z})$ is by no means a straightforward variant of that in $\mathcal{UTVPI}(\mathbb{Q})$, and several cases must be covered.

The proposed algorithm has the following merits. First, it generates interpolants that are within $\mathcal{UTVPI}$. This is important for applications where the computed interpolants are iteratively combined with the original problem, such as in interpolation-based bounded model checking [17]. Second, the approach can be easily implemented on top of a modern SMT procedure for $\mathcal{UTVPI}$, and runs with very limited overhead.

We have implemented our new algorithm within the MathSAT SMT solver [4], and performed experiments in order to evaluate both its efficiency and its usefulness. Our results demonstrate not only that our specialized $\mathcal{UTVPI}(\mathbb{Q})$ algorithm is faster and generates smaller interpolants than general $\mathcal{LA}(\mathbb{Q})$ interpolation procedures (and thus is interesting in itself), but also that our $\mathcal{UTVPI}(\mathbb{Z})$ algorithm can be useful for the verification of software model checking problems which require reasoning on the integers, and which could not be proved before by the Blast software model checker [2], due to the approximation resulting from its use of $\mathcal{LA}(\mathbb{Q})$ interpolation procedures.

Content of the paper. In §2 we provide the necessary background knowledge on SMT and interpolant generation in SMT. In §3 we present our novel graph-based interpolant technique for $\mathcal{UTVPI}(\mathbb{Q})$, whilst in §4 we show how to extend it to the case of $\mathcal{UTVPI}$ over $\mathbb{Z}$. In §5 we report some empirical results. In §6 we draw some conclusions, and outline directions for future research.

2 Background

Satisfiability Modulo Theory – SMT. Our setting is standard first order logic. We use the standard notions of theory, satisfiability, validity, logical consequence. We call *Satisfiability Modulo (the) Theory* $\mathcal{T}$, *SMT*($\mathcal{T}$), the problem of deciding the satisfiability of quantifier-free formulas wrt. a background theory $\mathcal{T}$. [3] Given a theory $\mathcal{T}$, we write $\phi \models_{\mathcal{T}} \psi$ (or simply $\phi \models \psi$) to denote that the formula ψ is a logical consequence of ϕ in the theory $\mathcal{T}$. With $\phi \preceq \psi$ we denote that all uninterpreted (in $\mathcal{T}$) symbols of ϕ appear in ψ. Without loss of generality, we also assume that the formulas are in Conjunctive Normal Form (CNF). If C is a clause, $C \downarrow B$ is the clause obtained from C by removing all the literals whose atoms do not occur in B, and $C \setminus B$ that obtained by removing all the literals whose atoms do occur in B. With a little abuse of notation, we might sometimes denote conjunctions of literals $l_1 \wedge \ldots \wedge l_n$ as sets $\{l_1, \ldots, l_n\}$ and vice versa. If η is a the set $\{l_1, \ldots, l_n\}$, we might write $\neg\eta$ to mean $\neg l_1 \vee \ldots \vee \neg l_n$.

[3] The general definition of SMT deals also with quantified formulas. Nevertheless, in this paper we restrict our interest to quantifier-free formulas.

We call $\mathcal{T}$-solver a procedure that decides the consistency of a conjunction of literals in $\mathcal{T}$. If S is a set of literals in $\mathcal{T}$, we call *$\mathcal{T}$-conflict set w.r.t. S* any subset η of S which is inconsistent in $\mathcal{T}$. We call $\neg\eta$ a *$\mathcal{T}$-lemma* (notice that $\neg\eta$ is a $\mathcal{T}$-valid clause). Given a set of clauses $S \stackrel{\text{def}}{=} \{C_1, \ldots, C_n\}$ and a clause C, we call a *resolution proof* that $\bigwedge_i C_i \models_{\mathcal{T}} C$ a DAG $\mathcal{P}$ such that:

1. C is the root of $\mathcal{P}$;
2. the leaves of $\mathcal{P}$ are either elements of S or $\mathcal{T}$-lemmas;
3. each non-leaf node C' has two parents C_{p_1} and C_{p_2} such that C_{p_1} is in the form $p \vee \phi_1$, C_{p_2} is in the form $\neg p \vee \phi_2$, and C' is $\phi_1 \vee \phi_2$. The atom p is called the *pivot* of C_{p_1} and C_{p_2}.

If C is the empty clause (denoted with $\bot$), then $\mathcal{P}$ is a *resolution proof of unsatisfiability* (or *resolution refutation*) for $\bigwedge_i C_i$.

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., [23] 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 in 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 η, and the corresponding $\mathcal{T}$-lemma $\neg\eta$ is added as a blocking clause in DPLL, and used to drive the backjump mechanism. With a small modification of the embedded DPLL engine, a lazy SMT solver can also be used to generate a resolution proof of unsatisfiability, where the leaf $\mathcal{T}$-lemmas are (some of) those returned by the $\mathcal{T}$-solver.

Interpolation in SMT. We consider the *SMT*($\mathcal{T}$) problem for some background theory $\mathcal{T}$. Given an ordered pair (A, B) of formulas such that $A \wedge B \models_{\mathcal{T}} \bot$, a *Craig interpolant* (simply "interpolant" hereafter) is a formula I s.t. (i) $A \models_{\mathcal{T}} I$, (ii) $I \wedge B$ is $\mathcal{T}$-inconsistent, and (iii) $I \preceq A$ and $I \preceq B$.

Following [18], an interpolant for (A, B) is constructed from a resolution proof of unsatisfiability of $A \wedge B$, generated as outlined above. The algorithm works by computing a formula I_C for each clause in the resolution refutation, such that the formula $I_\bot$ associated to the empty root clause is the computed interpolant. The algorithm can be described as follows:

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

1. Generate a proof of unsatisfiability $\mathcal{P}$ for $A \wedge B$.
2. For every $\mathcal{T}$-lemma $\neg\eta$ occurring in $\mathcal{P}$, generate an interpolant $I_{\neg\eta}$ for $(\eta \setminus B, \eta \downarrow B)$.
3. For every input clause C in $\mathcal{P}$, set I_C to $C \downarrow B$ if $C \in A$, and to $\top$ if $C \in B$.
4. For every inner node C of $\mathcal{P}$ obtained by resolution from $C_1 \stackrel{\text{def}}{=} p \vee \phi_1$ and $C_2 \stackrel{\text{def}}{=} \neg p \vee \phi_2$, set I_C to $I_{C_1} \vee I_{C_2}$ if p does not occur in B, and to $I_{C_1} \wedge I_{C_2}$ otherwise.
5. Output $I_\bot$ as an interpolant for (A, B).

Notice that Step 2. of the algorithm is the only part which depends on the theory $\mathcal{T}$, so that the problem of interpolant generation in $SMT(\mathcal{T})$ reduces to that of finding interpolants for negations of $\mathcal{T}$-lemmas, that is, for conjunctions of $\mathcal{T}$-literals in the given theory $\mathcal{T}$. Therefore, in the rest of the paper, we shall present algorithms for conjunctions/sets of literals only, which can be extended to general formulas by simply "plugging" them into the above algorithm. Moreover, we shall assume that when computing an interpolant for an inconsistent conjunction $A \wedge B$, neither A nor B is inconsistent in itself. In such cases, in fact, the interpolant would simply be $\perp$ and $\top$ respectively.

Graph-based Interpolation for Difference Logic. The theory of Difference Logic ($\mathcal{DL}$) is a fragment of the theory of linear arithmetic in which all atoms are inequalities of the form $(0 \leq y - x + c)$, where x and y are variables and c is an integer constant. [4] Many SMT solvers use dedicated, graph-based algorithms for checking the consistency of a set of $\mathcal{DL}$ atoms [6, 21]. Intuitively, a set ϕ of $\mathcal{DL}$ atoms induces a graph $G(\phi)$ whose vertices are the variables of the atoms, and there exists an edge $x \xrightarrow{c} y$ for every $(0 \leq y - x + c) \in \phi$. ϕ is inconsistent if and only if $G(\phi)$ has a cycle of negative weight.

In [5] we extended the graph-based approach to generate interpolants. Consider the interpolation problem (A, B) where A and B are sets of inequalities as above, and let C be (the set of atoms in) a negative cycle in the graph corresponding to $A \cup B$. Since we are assuming that neither A nor B is inconsistent, the edges in the cycle can be partitioned in subsets of A and B. We call a maximal A-path of C a path $x_1 \xrightarrow{c_1} \ldots \xrightarrow{c_{n-1}} x_n$ such that (I) $x_i \xrightarrow{c_i} x_{i+1} \in A$, and (II) C contains $x' \xrightarrow{c'} x_1$ and $x_n \xrightarrow{c''} x''$ that are in B. The variables x_1 and x_n of a maximal A-path $x_1 \rightsquigarrow x_n$ are called end-point variables.

Let the *summary constraint* of a maximal A-path $x_1 \xrightarrow{c_1} \ldots \xrightarrow{c_{n-1}} x_n$ be the inequality $0 \leq x_n - x_1 + \sum_{i=1}^{n-1} c_i$. The conjunction of summary constraints of the maximal A-paths of C is an interpolant for (A, B) [5].

3 Graph-based interpolation for $\mathcal{UTVPI}(\mathbb{Q})$

We analyze first the simpler case of $\mathcal{UTVPI}(\mathbb{Q})$. Miné [20] showed that it is possible to encode a set of $\mathcal{UTVPI}(\mathbb{Q})$ constraints into a $\mathcal{DL}(\mathbb{Q})$ one in a satisfiability-preserving way. The encoding works as follows. We use x_i to denote variables in the $\mathcal{UTVPI}(\mathbb{Q})$ domain and u, v for variables in the $\mathcal{DL}(\mathbb{Q})$ domain. For every variable x_i, we introduce two distinct variables x_i^+ and x_i^-. We introduce a mapping Υ from $\mathcal{DL}(\mathbb{Q})$ variables to $\mathcal{UTVPI}(\mathbb{Q})$ signed variables, such that $\Upsilon(x_i^+) = x_i$ and $\Upsilon(x_i^-) = -x_i$. Υ extends to (sets of) constraints in the natural way. We say that $(x_i^+)^- = x_i^-$ and $(x_i^-)^- = x_i^+$. We say that the

[4] Notice that a conjunction of non-strict difference inequalities has a solution in the integer domain if and only if it has a solution in the rational domain, so that in this context we make no distinction between $\mathcal{DL}(\mathbb{Q})$ and $\mathcal{DL}(\mathbb{Z})$.

$\mathcal{UTVPI}(\mathbb{Q})$ constraints	$\mathcal{DL}(\mathbb{Q})$ constraints
$(0 \leq x_1 - x_2 + k)$	$(0 \leq x_1^+ - x_2^+ + k), (0 \leq x_2^- - x_1^- + k)$
$(0 \leq -x_1 - x_2 + k)$	$(0 \leq x_1^- - x_2^+ + k), (0 \leq x_2^- - x_1^+ + k)$
$(0 \leq x_1 + x_2 + k)$	$(0 \leq x_1^+ - x_2^- + k), (0 \leq x_2^+ - x_1^- + k)$
$(0 \leq -x_1 + k)$	$(0 \leq x_1^- - x_1^+ + 2 \cdot k)$
$(0 \leq x_1 + k)$	$(0 \leq x_1^+ - x_1^- + 2 \cdot k)$

Fig. 1. The conversion map from $\mathcal{UTVPI}(\mathbb{Q})$ to $\mathcal{DL}(\mathbb{Q})$ [20, 16].

constraints $(0 \leq u - v)$ and $(0 \leq (v)^- - (u)^-)$ s.t. $u, v \in \{x_i^+, x_i^-\}_i$ are *dual.* We encode each $\mathcal{UTVPI}$ constraint into the conjunction of two dual $\mathcal{DL}(\mathbb{Q})$ constraints, as represented in Figure 1. For each $\mathcal{DL}(\mathbb{Q})$ constraint $(0 \leq v - u + k)$, $(0 \leq \Upsilon(v) - \Upsilon(u) + k)$ is the corresponding $\mathcal{UTVPI}(\mathbb{Q})$ constraint. Notice that the two dual $\mathcal{DL}(\mathbb{Q})$ constraints are just different representations of the original $\mathcal{UTVPI}(\mathbb{Q})$ constraint. (The two dual constraints encoding a single-variable constraint are identical, so that their conjunction is collapsed into one constraint only.) The resulting set of constraints is satisfiable in $\mathcal{DL}(\mathbb{Q})$ if and only if the original one is satisfiable in $\mathcal{UTVPI}(\mathbb{Q})$ [20, 16].

Consider the pair (A, B) where A and B are sets of $\mathcal{UTVPI}(\mathbb{Q})$ constraints. We apply the map of Figure 1 and we encode (A, B) into a $\mathcal{DL}(\mathbb{Q})$ pair (A', B'), and build the constraint graph $G(A' \wedge B')$. If $G(A' \wedge B')$ has no negative cycle, we can conclude that $A' \wedge B'$ is $\mathcal{DL}(\mathbb{Q})$-consistent, and hence that $A \wedge B$ is $\mathcal{UTVPI}(\mathbb{Q})$-consistent. Otherwise, $A' \wedge B'$ is $\mathcal{DL}(\mathbb{Q})$-inconsistent, and hence $A \wedge B$ is $\mathcal{UTVPI}(\mathbb{Q})$-inconsistent. In fact, we observe that for any set of $\mathcal{DL}(\mathbb{Q})$ constraints $\{C_1, \ldots, C_n, C\}$ resulting from the encoding of some $\mathcal{UTVPI}(\mathbb{Q})$ constraints, if $\bigwedge_{i=1}^n C_i \models_{\mathcal{DL}(\mathbb{Q})} C$ then $\bigwedge_{i=1}^n \Upsilon(C_i) \models_{\mathcal{UTVPI}(\mathbb{Q})} \Upsilon(C)$.

When $A \wedge B$ is inconsistent, we can generate an $\mathcal{UTVPI}(\mathbb{Q})$-interpolant by extending the graph-based approach used for $\mathcal{DL}(\mathbb{Q})$. We consider a negative-weight cycle of $G(A' \wedge B')$, and we build an interpolant I' in $\mathcal{DL}(\mathbb{Q})$ by means of the graph-based interpolation technique described in §2. We claim that the $\mathcal{UTVPI}(\mathbb{Q})$ formula $I \overset{\text{def}}{=} \Upsilon(I')$ is an interpolant for (A, B). The proof is as follows. (i) I' is a conjunction of summary constraints, so it is in the form $\bigwedge_i C_i$. Therefore $A' \models_{\mathcal{DL}(\mathbb{Q})} C_i$ for all i, and so by the observation above $A \models_{\mathcal{UTVPI}(\mathbb{Q})} \Upsilon(C_i)$. Hence, $A \models_{\mathcal{UTVPI}(\mathbb{Q})} I$. (ii) From the $\mathcal{DL}(\mathbb{Q})$-inconsistency of $I' \wedge B'$ we immediately derive that $I \wedge B$ is $\mathcal{UTVPI}(\mathbb{Q})$-inconsistent. (iii) $I \preceq A$ and $I \preceq B$ derive from $I' \preceq A'$ and $I' \preceq B'$ by the definitions of Υ and the map of Figure 1.

Example 1. Consider the following sets of $\mathcal{UTVPI}(\mathbb{Q})$ constraints:

$$\begin{aligned} A = \{&(0 \leq -x_2 - x_1 + 3), (0 \leq x_1 + x_3 + 1), \\ &(0 \leq -x_3 - x_4 - 6), (0 \leq x_5 + x_4 + 1)\} \end{aligned}$$

$$B = \{(0 \leq x_2 + x_3 + 3), (0 \leq x_6 - x_5 - 1), (0 \leq x_4 - x_6 + 4)\}$$

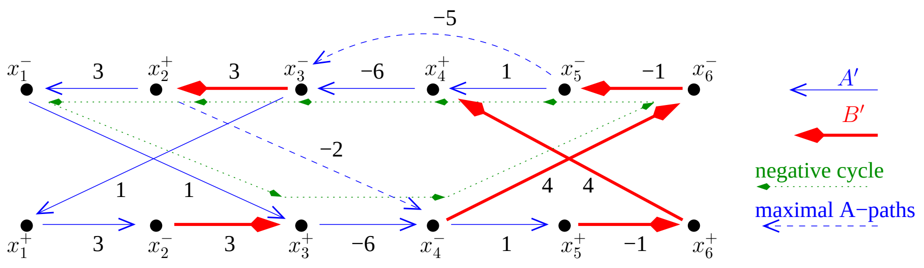

Fig. 2. The constraint graph of Example 1. (We represent only one negative cycle with its corresponding A-paths, because the other is dual.)

By the map of Figure 1, they are converted into the following sets of $\mathcal{DL}(\mathbb{Q})$ constraints:

$$\begin{aligned} A' = \{&(0 \leq x_1^- - x_2^+ + 3), (0 \leq x_2^- - x_1^+ + 3),\\ &(0 \leq x_3^+ - x_1^- + 1), (0 \leq x_1^+ - x_3^- + 1),\\ &(0 \leq x_4^- - x_3^+ - 6), (0 \leq x_3^- - x_4^+ - 6),\\ &(0 \leq x_4^+ - x_5^- + 1), (0 \leq x_5^+ - x_4^- + 1)\} \end{aligned}$$

$$\begin{aligned} B' = \{&(0 \leq x_3^+ - x_2^- + 3), (0 \leq x_2^+ - x_3^- + 3),\\ &(0 \leq x_6^+ - x_5^+ - 1), (0 \leq x_5^- - x_6^- - 1),\\ &(0 \leq x_4^+ - x_6^+ + 4), (0 \leq x_6^- - x_4^- + 4)\} \end{aligned}$$

whose conjunction corresponds to the constraint graph of Figure 2. This graph has a negative cycle

$$C' \overset{\text{def}}{=} x_2^+ \xrightarrow{3} x_1^- \xrightarrow{1} x_3^+ \xrightarrow{-6} x_4^- \xrightarrow{4} x_6^- \xrightarrow{-1} x_5^- \xrightarrow{1} x_4^+ \xrightarrow{-6} x_3^- \xrightarrow{3} x_2^+.$$

Thus, $A \wedge B$ is inconsistent in $\mathcal{UTVPI}(\mathbb{Q})$. From the negative cycle C' we can extract the set of A'-paths $\{x_2^+ \xrightarrow{-2} x_4^-,\ x_5^- \xrightarrow{-5} x_3^-\}$, corresponding to the formula $I' \overset{\text{def}}{=} (0 \leq x_4^- - x_2^+ - 2) \wedge (0 \leq x_3^- - x_5^- - 5)$, which is an interpolant for (A', B'). I' is thus mapped back into $I \overset{\text{def}}{=} (0 \leq -x_2 - x_4 - 2) \wedge (0 \leq x_5 - x_3 - 5)$, which is an interpolant for (A, B).

4 Graph-based interpolation for $\mathcal{UTVPI}(\mathbb{Z})$

In order to deal with the more complex case of $\mathcal{UTVPI}(\mathbb{Z})$ (hereafter simply $\mathcal{UTVPI}$), we adopt a layered approach [23]. First, we check the consistency in $\mathcal{UTVPI}(\mathbb{Q})$ using the technique of [20]. If this results in an inconsistency, we compute an $\mathcal{UTVPI}(\mathbb{Q})$-interpolant as described in §3. Clearly, this is also an interpolant in $\mathcal{UTVPI}$: condition (iii) is obvious, and conditions (i) and (ii) follow immediately from the fact that if an $\mathcal{UTVPI}$-formula is inconsistent over the rationals then it is inconsistent also over the integers.

If the $\mathcal{UTVPI}(\mathbb{Q})$-procedure does not detect an inconsistency, we check the consistency in $\mathcal{UTVPI}$ using the algorithm proposed by Lahiri and Musuvathi in [16], which extends the ideas of [20] to the integer domain. In particular, it gives necessary and sufficient conditions to decide unsatisfiability by detecting particular kinds of zero-weight cycles in the induced $\mathcal{DL}$ constraint graph. This procedure works in $O(n \cdot m)$ time and $O(n+m)$ space, m and n being the number of constraints and variables respectively, which improves the previous $O(n^2 \cdot m)$ time and $O(n^2)$ space complexity of the previous procedure by Jaffar et al. [9].

We build on top of this algorithm and we extend the graph-based approach of §3 for producing interpolants also in $\mathcal{UTVPI}$. In particular, we use the following reformulation of a result of [16].

Theorem 1. *Let ϕ be a conjunction of $\mathcal{UTVPI}$ constraints s.t. ϕ is satisfiable in $\mathcal{UTVPI}(\mathbb{Q})$. Then ϕ is unsatisfiable in $\mathcal{UTVPI}$ iff the constraint graph $G(\phi)$ generated from ϕ has a cycle C of weight 0 containing two vertices x_i^+ and x_i^- s.t. the weight of the path $x_i^- \rightsquigarrow x_i^+$ along C is odd.*

The "only if" part is a corollary of lemmas 1, 2 and 4 in [16]. The "if" comes straightforwardly from the analysis done in [16], whose main intuitions we recall in what follows. Assume the constraint graph $G(\phi)$ generated from ϕ has one cycle C of weight 0 containing two vertices x_i^+ and x_i^- s.t. the weight of the path $x_i^- \rightsquigarrow x_i^+$ along C is $2k+1$ for some integer value k. (Since C has weight 0, the weight of the other path $x_i^+ \rightsquigarrow x_i^-$ along C is $-2k-1$.) Then, the paths $x_i^- \rightsquigarrow x_i^+$ and $x_i^+ \rightsquigarrow x_i^-$ contain at least two constraints, because otherwise their weight would be even (see the last two lines of Figure 1). Then, $x_i^- \rightsquigarrow x_i^+$ is in the form $x_i^- \rightsquigarrow v \xrightarrow{n} x_i^+$, for some v and n. From $x_i^- \rightsquigarrow v$, we can derive the summary constraint $(0 \leq v - x_i^- + (2k+1-n))$, which corresponds to the $\mathcal{UTVPI}$ constraint $(0 \leq \Upsilon(v) + x_i + (2k+1-n))$. (This corresponds to $l-2$ applications of the Transitive rule of [16], l being the number of constraints in $x_i^- \rightsquigarrow x_i^+$.) Then, by observing that the $\mathcal{UTVPI}$ constraint corresponding to $v \xrightarrow{n} x_i^+$ is $(0 \leq x_i - \Upsilon(v) + n)$, we can apply the Tightening rule of [16] to obtain $(0 \leq x_i + \lfloor (2k+1-n+n)/2 \rfloor)$, which is equivalent to $(0 \leq x_i + k)$. Similarly, from $x_i^+ \rightsquigarrow x_i^-$ we can obtain $(0 \leq -x_i - k - 1)$, and thus an inconsistency using the Contradiction rule of [16].

Consider a pair (A, B) of $\mathcal{UTVPI}$ constraints such that $A \wedge B$ is consistent in $\mathcal{UTVPI}(\mathbb{Q})$ but inconsistent in $\mathcal{UTVPI}$. By Theorem 1, the constraint graph $G(A' \wedge B')$ has a cycle C of weight 0 containing two vertices x_i^+ and x_i^- s.t. the weight of the paths $x_i^- \rightsquigarrow x_i^+$ and $x_i^+ \rightsquigarrow x_i^-$ along C are $2k+1$ and $-2k-1$ respectively, for some value $k \in \mathbb{Z}$. Our algorithm computes an interpolant for (A, B) from the cycle C. Let C_A and C_B be the subsets of the edges in C corresponding to constraints in A' and B' respectively. We have to distinguish four distinct sub-cases.

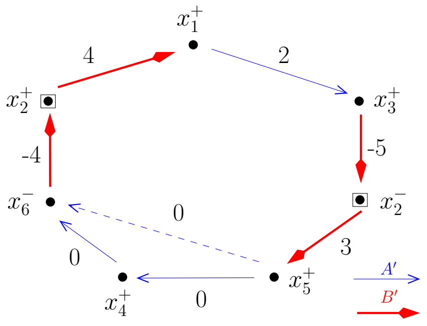

Fig. 3. $\mathcal{UTVPI}$ interpolation, Case 1.

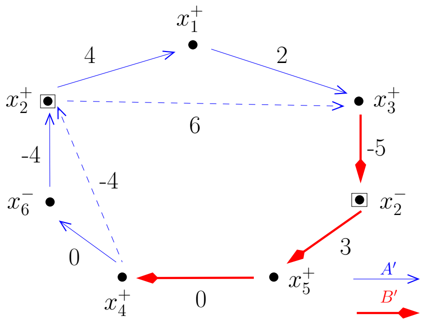

Fig. 4. $\mathcal{UTVPI}$ interpolation, Case 2.

Case 1: *x_i occurs in B but not in A.* Consequently, x_i^+ and x_i^- occur in B' but not in A', and hence they occur in C_B but not in C_A. Let I' be the conjunction of the summary constraints of the maximal C_A-paths, and let I be the conjunction of the corresponding $\mathcal{UTVPI}$ constraints. We show that I is an interpolant for (A, B). (i) By construction, $A \models_{\mathcal{UTVPI}} I$, as in §3. (ii) The constraints in I' and C_B form a cycle matching the hypotheses of Theorem 1, from which $I \wedge B$ is $\mathcal{UTVPI}$-inconsistent. (iii) We notice that every variable x_j^+, x_j^- occurring in the conjunction of the summary constraints is an end-point variable, so that $I' \preceq C_A$ and $I' \preceq C_B$, and thus $I \preceq A$ and $I \preceq B$.

Example 2. Consider the following set of constraints:

$$S = \{(0 \leq x_1 - x_2 + 4), (0 \leq -x_2 - x_3 - 5), (0 \leq x_2 + x_6 - 4), (0 \leq x_5 + x_2 + 3), \\ (0 \leq -x_1 + x_3 + 2), (0 \leq -x_6 - x_4), (0 \leq x_4 - x_5)\},$$

partitioned into A and B as follows:

$$A \begin{cases} (0 \leq x_3 - x_1 + 2) \\ (0 \leq -x_6 - x_4) \\ (0 \leq x_4 - x_5) \end{cases} \qquad B \begin{cases} (0 \leq x_1 - x_2 + 4) \\ (0 \leq -x_2 - x_3 - 5) \\ (0 \leq x_2 + x_6 - 4) \\ (0 \leq x_5 + x_2 + 3) \end{cases}$$

Figure 3 shows a zero-weight cycle C in $G(A' \wedge B')$ such that the paths $x_2^- \rightsquigarrow x_2^+$ and $x_2^+ \rightsquigarrow x_2^-$ have an odd weight (-1 and 1 resp.) Therefore, by Theorem 1, $A \wedge B$ is $\mathcal{UTVPI}$-inconsistent. The two summary constraints of the maximal C_A paths are $(0 \leq x_6^- - x_5^+)$ and $(0 \leq x_3^+ - x_1^+ + 2)$. It is easy to see that $I = (0 \leq -x_6 - x_5) \wedge (0 \leq x_3 - x_1 + 2)$ is an $\mathcal{UTVPI}$-interpolant for (A, B).

Case 2: *x_i occurs in both A and B.* Consequently, x_i^+ and x_i^- occur in both A' and B'. If neither x_i^+ nor x_i^- is such that both the incoming and outgoing edges belong to C_A, then the cycle obtained by replacing each maximal C_A-path with its summary constraint still contains both x_i^+ and x_i^-, so we can apply the same process of Case 1. Otherwise, if both the incoming and outgoing edges of x_i^+

belong to C_A, then we split the maximal C_A-path $u_1 \xrightarrow{c_1} \dots \xrightarrow{c_k} x_i^+ \xrightarrow{c_{k+1}} \dots \xrightarrow{c_n} u_n$ containing x_i^+ into the two parts which are separated by x_i^+: $u_1 \xrightarrow{c_1} \dots \xrightarrow{c_k} x_i^+$ and $x_i^+ \xrightarrow{c_{k+1}} \dots \xrightarrow{c_n} u_n$. We do the same for x_i^-. Let I' be the conjunction of the resulting summary constraints, and let I be corresponding set of $\mathcal{UTVPI}$ constraints. We show that I is an interpolant for (A, B). (i) As with Case 1, again, $A \models_{\mathcal{UTVPI}} I$. (ii) Since we split the maximal C_A paths as described above, the constraints in I' and C_B form a cycle matching the hypotheses of Theorem 1, from which $I \wedge B$ is $\mathcal{UTVPI}$-inconsistent. (iii) x_i^+, x_i^- occur in both A' and B' by hypothesis, and every other variable x_j^+, x_j^- occurring in the conjunction of the summary constraints is an end-point variable, so that $I' \preceq C_A$ and $I' \preceq C_B$, and thus $I \preceq A$ and $I \preceq B$.

Example 3. Consider again the set of constraints S of Example 2, partitioned into A and B as follows:

$$A \begin{cases} (0 \leq x_3 - x_1 + 2) \\ (0 \leq -x_6 - x_4) \\ (0 \leq x_2 + x_6 - 4) \\ (0 \leq x_1 - x_2 + 4) \end{cases} \qquad B \begin{cases} (0 \leq -x_2 - x_3 - 5) \\ (0 \leq x_5 + x_2 + 3) \\ (0 \leq x_4 - x_5) \end{cases}$$

and the zero-weight cycle C of $G(A' \wedge B')$ shown in Figure 4. As in the previous example, there is a path $x_2^- \rightsquigarrow x_2^+$ of weight -1 and a path $x_2^+ \rightsquigarrow x_2^-$ of weight 1. In this case there is only one maximal C_A path, namely $x_4^+ \rightsquigarrow x_3^+$. Since the cycle obtained by replacing it with its summary constraint $(0 \leq x_3^+ - x_4^+ + 2)$ does not contain x_2^+, we split $x_4^+ \rightsquigarrow x_3^+$ into two paths, $x_4^+ \rightsquigarrow x_2^+$ and $x_2^+ \rightsquigarrow x_3^+$, whose summary constraints are $(0 \leq x_2^+ - x_4^+ - 4)$ and $(0 \leq x_3^+ - x_2^+ + 6)$ respectively. By replacing the two paths above with the two summary constraints, we get a zero-weight cycle which still contains the two odd paths $x_2^- \rightsquigarrow x_2^+$ and $x_2^+ \rightsquigarrow x_2^-$. Therefore, $I \stackrel{\text{def}}{=} (0 \leq x_2 - x_4 - 4) \wedge (0 \leq x_3 - x_2 + 6)$ is an interpolant for (A, B).

Notice that the $\mathcal{UTVPI}$-formula $J \stackrel{\text{def}}{=} (0 \leq x_3 - x_4 + 2)$ corresponding to the summary constraint of the maximal C_A path $x_4^+ \rightsquigarrow x_3^+$ is not an interpolant, since $J \wedge B$ is not $\mathcal{UTVPI}$-inconsistent. In fact, if we replace the maximal C_A path $x_4^+ \rightsquigarrow x_3^+$ with the summary constraint $x_4^+ \xrightarrow{2} x_3^+$, the cycle we obtain has still weight zero, but it contains no odd path between two variables x_i^+ and x_i^-.

Case 3: *x_i occurs in A but not in B, and one of the paths $x_i^+ \rightsquigarrow x_i^-$ or $x_i^- \rightsquigarrow x_i^+$ in C contains only constraints of C_A.* In this case, x_i^+ and x_i^- occur in A' but not in B'. Suppose that $x_i^- \rightsquigarrow x_i^+$ consists only of constraints of C_A (the case $x_i^+ \rightsquigarrow x_i^-$ is analogous). Let $2k+1$ be the weight of the path $x_i^- \rightsquigarrow x_i^+$ (which is odd by hypothesis), and let $\overline{C}$ be the cycle obtained by replacing such path with the edge $x_i^- \xrightarrow{2k} x_i^+$ in C. In the following, we call such a replacement *tightening summarization.* Since C has weight zero, $\overline{C}$ has negative weight. Let C^P be the set of $\mathcal{DL}$-constraints in the path $x_i^- \rightsquigarrow x_i^+$. Let I' be the $\mathcal{DL}$-interpolant computed from $\overline{C}$ for $(C_A \setminus C^P \cup \{(0 \leq x_i^+ - x_i^- + 2k)\}, C_B)$, and let I be the corresponding $\mathcal{UTVPI}$ formula. We show that I is an interpolant for (A, B).

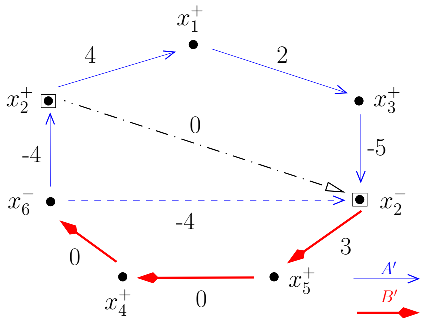

Fig. 5. $\mathcal{UTVPI}$ interpolation, Case 3.

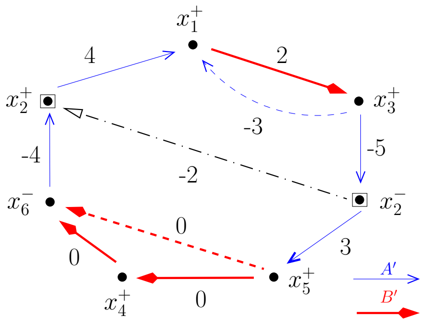

Fig. 6. $\mathcal{UTVPI}$ interpolation, Case 4.

(i) Let P be the set of $\mathcal{UTVPI}$ constraints in the path $x_i^- \rightsquigarrow x_i^+$. Since the weight $2k+1$ of such path is odd, we have that $P \models_{\mathcal{UTVPI}} (0 \leq x_i + k)$ (cf. page 8). Since $P \subseteq A$, therefore, $A \models_{\mathcal{UTVPI}} (0 \leq x_i + k)$. By observing that $(0 \leq x_i^+ - x_i^- + 2k)$ is the $\mathcal{DL}$-constraint corresponding to $(0 \leq x_i + k)$ we conclude that $C_A \setminus C^P \cup (0 \leq x_i^+ - x_i^- + 2k) \models_{\mathcal{DL}} I'$ implies that $A \setminus P \cup (0 \leq x_i + k) \models_{\mathcal{UTVPI}} I$, and so that $A \models_{\mathcal{UTVPI}} I$.

(ii) Since all the constraints in C_B occur in $\overline{C}$, we have that $B \wedge I$ is $\mathcal{UTVPI}$-inconsistent.

(iii) Since by hypothesis all the constraints in the path $x_i^- \rightsquigarrow x_i^+$ occur in C_A, from $I' \preceq (C_A \setminus C^P \cup \{(0 \leq x_i^+ - x_i^- + 2k)\})$ we have that $I \preceq A$. Finally, since all the constraints in C_B occur in $\overline{C}$, we have that $I \preceq B$.

Example 4. Consider again the set S of constraints of Example 2, this time partitioned into A and B as follows:

$$A \begin{cases} (0 \leq x_1 - x_2 + 4) \\ (0 \leq x_3 - x_1 + 2) \\ (0 \leq -x_2 - x_3 - 5) \\ (0 \leq x_2 + x_6 - 4) \end{cases} \qquad B \begin{cases} (0 \leq x_5 + x_2 + 3) \\ (0 \leq -x_6 - x_4) \\ (0 \leq x_4 - x_5) \end{cases}$$

Figure 5 shows a zero-weight cycle C of $G(A' \wedge B')$. The only maximal C_A path is $x_6^- \rightsquigarrow x_2^-$. Since the path $x_2^+ \rightsquigarrow x_2^-$ has weight 1, we can add the tightening edge $x_2^+ \xrightarrow{1-1} x_2^-$ to $G(A' \wedge B')$ (shown in dots and dashes in Figure 5), corresponding to the constraint $(0 \leq x_2^- - x_2^+)$. Since all constraints in the path $x_2^+ \rightsquigarrow x_2^-$ belong to A', $A' \models (0 \leq x_2^- - x_2^+)$. Moreover, the cycle obtained by replacing the path $x_2^+ \rightsquigarrow x_2^-$ with the tightening edge $x_2^+ \xrightarrow{0} x_2^-$ has a negative weight (-1). Therefore, we can generate a $\mathcal{DL}$-interpolant $I' \overset{\text{def}}{=} (0 \leq x_2^- - x_6^- - 4)$ from such cycle, which corresponds to the $\mathcal{UTVPI}$-interpolant $I \overset{\text{def}}{=} (0 \leq -x_2 + x_6 - 4)$.

Notice that, similarly to Example 3, also in this case we cannot obtain an interpolant from the summary constraint $(0 \leq x_2^- - x_6^- - 3)$ of the maximal C_A path $x_6^- \rightsquigarrow x_2^-$, as $(0 \leq -x_2 + x_6 - 3) \wedge B$ is not $\mathcal{UTVPI}$-inconsistent.

Case 4: *x_i occurs in A but not in B, and neither the path $x_i^+ \rightsquigarrow x_i^-$ nor the path $x_i^- \rightsquigarrow x_i^+$ in C consists only of constraints of C_A.* As in the previous case, x_i^+ and x_i^- occur in A' but not in B', and hence they occur in C_A but not in C_B. In this case, however, we can apply a tightening summarization neither to $x_i^+ \rightsquigarrow x_i^-$ nor to $x_i^- \rightsquigarrow x_i^+$, since none of the two paths consists only of constraints of C_A. We can, however, perform a *conditional tightening summarization* as follows. Let C_A^P and C_B^P be the sets of constraints of C_A and C_B respectively occurring in the path $x_i^- \rightsquigarrow x_i^+$, and let $\overline{C}_A^P$ and $\overline{C}_B^P$ be the sets of summary constraints of maximal paths in C_A^P and C_B^P. From $\overline{C}_A^P \cup \overline{C}_B^P$, we can derive $x_i^- \xrightarrow{2k} x_i^+$ (cf. Case 3), where $2k+1$ is the weight of the path $x_i^- \rightsquigarrow x_i^+$. Therefore, $\overline{C}_A^P \cup \overline{C}_B^P \models (0 \leq x_i^+ - x_i^- + 2k)$, and thus $\overline{C}_A^P \models \overline{C}_B^P \rightarrow (0 \leq x_i^+ - x_i^- + 2k)$. We say that $(0 \leq x_i^+ - x_i^- + 2k)$ is the summary constraint for $x_i^- \rightsquigarrow x_i^+$ *conditioned to* $\overline{C}_B^P$.

Using conditional tightening summarization, we generate an interpolant as follows. By replacing the path $x_i^- \rightsquigarrow x_i^+$ with $x_i^- \xrightarrow{2k} x_i^+$, we obtain a negative-weight cycle $\overline{C}$, as in Case 3. Let I' be the $\mathcal{DL}$-interpolant computed from $\overline{C}$ for $(C_A \setminus C_A^P \cup \{(0 \leq x_i^+ - x_i^- + 2k)\}, C_B \setminus C_B^P)$, and let I be the corresponding $\mathcal{UTVPI}$ formula. Finally, let $\overline{P}_B$ be the conjunction of $\mathcal{UTVPI}$ constraints corresponding to $\overline{C}_B^P$. We show that $(\overline{P}_B \rightarrow I)$ is an interpolant for (A, B).

(i) We know that $C_A \setminus C_A^P \cup \{(0 \leq x_i^+ - x_i^- + 2k)\} \models I'$, because I' is a $\mathcal{DL}$-interpolant. Moreover, $\overline{C}_A^P \cup \overline{C}_B^P \models (0 \leq x_i^+ - x_i^- + 2k)$, and so $C_A^P \cup \overline{C}_B^P \models (0 \leq x_i^+ - x_i^- + 2k)$. Therefore, $C_A \cup \overline{C}_B^P \models I'$, and thus $A \cup \overline{P}_B \models_{\mathcal{UTVPI}} I$, from which $A \models_{\mathcal{UTVPI}} (\overline{P}_B \rightarrow I)$.

(ii) Since I' is a $\mathcal{DL}$-interpolant for $(C_A \setminus C_A^P \cup \{(0 \leq x_i^+ - x_i^- + 2k)\}, C_B \setminus C_B^P)$, $I' \wedge (C_B \setminus C_B^P)$ is $\mathcal{DL}$-inconsistent, and thus $I \wedge B$ is $\mathcal{UTVPI}$-inconsistent. Since by construction $B \models_{\mathcal{UTVPI}} \overline{P}_B$, $(\overline{P}_B \rightarrow I) \wedge B$ is $\mathcal{UTVPI}$-inconsistent.

(iii) From $I' \preceq C_B \setminus C_B^P$ we have that $I \preceq B$, and from $I' \preceq C_A \setminus C_A^P \cup \{(0 \leq x_i^+ - x_i^- + 2k)\}$ that $I \preceq A$. Moreover, all the variables occurring in the constraints in $\overline{C}_B^P$ are end-point variables, so that $\overline{C}_B^P \preceq C_A$ and $\overline{C}_B^P \preceq C_B$, and thus $\overline{P}_B \preceq A$ and $\overline{P}_B \preceq B$. Therefore, $(\overline{P}_B \rightarrow I) \preceq A$ and $(\overline{P}_B \rightarrow I) \preceq B$.

Example 5. We partition the set S of constraints of Example 2 into A and B as follows:

$$A \begin{cases} (0 \leq x_1 - x_2 + 4) \\ (0 \leq -x_2 - x_3 - 5) \\ (0 \leq x_5 + x_2 + 3) \\ (0 \leq x_2 + x_6 - 4) \end{cases} \qquad B \begin{cases} (0 \leq x_3 - x_1 + 2) \\ (0 \leq -x_6 - x_4) \\ (0 \leq x_4 - x_5) \end{cases}$$

Consider the zero-weight cycle C of $G(A' \wedge B')$ shown in Figure 6. In this case, neither the path $x_2^+ \rightsquigarrow x_2^-$ nor the path $x_2^- \rightsquigarrow x_2^+$ consists only of constraints of A', and thus we cannot use any of the two tightening edges $x_2^+ \xrightarrow{1-1} x_2^-$ and $x_2^- \xrightarrow{-1-1} x_2^+$ *directly* for computing an interpolant. However, we can compute the summary $x_2^- \xrightarrow{-2} x_2^+$ for $x_2^- \rightsquigarrow x_2^+$ *conditioned to* $x_5^+ \xrightarrow{0} x_6^-$, which is the summary constraint of the B-path $x_5^+ \rightsquigarrow x_6^-$, and whose corresponding $\mathcal{UTVPI}$

constraint is $(0 \le -x_6 - x_5)$. By replacing the path $x_2^- \leadsto x_2^+$ with such summary, we obtain a negative-weight cycle $\overline{C}$, from which we generate the $\mathcal{DL}$-interpolant $(0 \le x_1^+ - x_3^+ - 3)$, corresponding to the $\mathcal{UTVPI}$ formula $(0 \le x_1 - x_3 - 3)$. Therefore, the generated $\mathcal{UTVPI}$-interpolant is $(0 \le -x_6 - x_5) \rightarrow (0 \le x_1 - x_3 - 3)$.

As in Example 4, notice that we cannot generate an interpolant from the conjunction of summary constraints of maximal C_A paths, since the formula we obtain (i.e. $(0 \le x_1 + x_6) \wedge (0 \le x_5 - x_3 - 2)$) is not inconsistent with B.

5 Experimental evaluation

We have implemented the algorithm described in the previous sections within our SMT solver MathSAT [4]. The assessment of the procedure can not be carried out by means of a direct comparison, since there exists no other system able to interpolate over $\mathcal{UTVPI}(\mathbb{Z})$. In order to assess both the efficiency and the usefulness of the procedure, we have performed two kinds of experiments, one on interpolation over $\mathcal{UTVPI}(\mathbb{Q})$, and one on the application of interpolation for $\mathcal{UTVPI}(\mathbb{Z})$ to software model checking.

The programs and benchmark instances used are available at `http://disi.unitn.it/~griggio/papers/cade09_itp_utvpi.tar.gz`. All the tests have been performed on 2.66 GHz Intel Xeon machines, with 16 GB of RAM and 6 MB of cache, running Linux. For each instance, we used a time limit of 20 minutes and a memory limit of 2 GB.

Comparison with $\mathcal{LA}(\mathbb{Q})$ interpolation. In the first part of our experiments, we compare our novel $\mathcal{UTVPI}(\mathbb{Q})$ interpolation algorithm with our implementation of a state-of-the-art $\mathcal{LA}(\mathbb{Q})$ interpolation algorithm [5], in order to evaluate its efficiency. Both algorithms are implemented within MathSAT, and thus they share the same environment (same DPLL engine, same search strategy, same optimizations, etc.). This ensures that the comparison is fair.

We have randomly generated several $\mathcal{UTVPI}(\mathbb{Q})$ interpolation problems of varying size and difficulty, and run both algorithms. The results are collected in Figure 7. The scatter plots show that the $\mathcal{UTVPI}(\mathbb{Q})$ solver clearly outperforms the $\mathcal{LA}(\mathbb{Q})$ solver (sometimes by more than an order of magnitude), thus justifying the interest for the subclass. Furthermore, it can be seen that the computed interpolants, in addition to being within $\mathcal{UTVPI}(\mathbb{Q})$, are generally smaller, both in terms of nodes in the formula DAG and in number of atoms.

$\mathcal{UTVPI}$ interpolation in software model checking. In the second part of our experiments, we evaluate the usefulness of the $\mathcal{UTVPI}$ interpolation procedure in the context of software model checking based on the counterexample-guided abstraction refinement (CEGAR) paradigm. This is one of the most successful applications of interpolation in formal verification [8]: in this setting, interpolants are used to automatically refine abstractions when *spurious* error traces are generated. A spurious error trace is an execution of the abstract program that leads to an error, but does not correspond to any execution of the concrete program. The interpolation procedure receives as input formulas corresponding to such spurious error traces, which are typically conjunctions of literals, and the interpolants generated are used to prevent the same spurious error trace from being generated again in the future. This technique is used for example by the software model checker BLAST, whose description in [2] we refer to for the details.

Due to the limitations of current interpolation procedures, [5] when computing interpolants program variables are interpreted over the rationals even if in the program they have an integral type. This makes it impossible to compute interpolants for spurious error traces which are inconsistent because of the violation of the integrality constraints on the variables. When this happens, automatic abstraction refinement cannot be performed, and the verification of the program fails.

We have written a collection of small C programs which cannot be verified by BLAST because the interpolation procedures that it uses ([3, 18, 22]) do not handle integrality constraints. When using MATHSAT as interpolation procedure instead, BLAST could successfully verify all the programs.

Example 6. Consider the simple C program on the right. [6] In the process of proving that the `ERROR` label is not reachable, BLAST generates the following spurious counterexample:

$$\texttt{y} = *;\ \texttt{x} = \texttt{y};\ \texttt{z} = 1 - \texttt{y};\ (\texttt{x} == \texttt{z});\ \texttt{ERROR},$$

which corresponds to the following formula ξ:

$$\xi \overset{\text{def}}{=} (x = y) \wedge (z = 1 - y) \wedge (x = z)$$

```
main() {
  int x, y, z;
  while (*) {
    y = *;
    x = y;
    z = 1 - y;
    if (x == z) {
      ERROR: ;
    }
  }
}
```

In order to refine the abstraction, BLAST asks the interpolation procedure to compute an interpolant for the following partition of ξ into (A, B):

$$\underbrace{(x = y) \wedge (z = 1 - y)}_{A} \wedge \underbrace{(x = z)}_{B}.$$

ξ is unsatisfiable in $\mathbb{Z}$, but satisfiable in $\mathbb{Q}$. Therefore, interpolation procedures that work on the rationals are not able to compute the interpolant, causing BLAST to fail. Using the $\mathcal{UTVPI}$ interpolation algorithm, instead, MATHSAT computes the following interpolant: [7]

$$I \overset{\text{def}}{=} (0 \leq x + z - 1) \wedge (0 \leq -x - z + 1),$$

[5] With the exception of [10], which however is limited to equations and modular equations, and cannot handle inequalities.

[6] A '`*`' indicates a nondeterministic value.

[7] After rewriting equalities $(x_1 = x_2 + c)$ into $(0 \leq x_1 - x_2 + c) \wedge (0 \leq x_2 - x_1 - c)$.

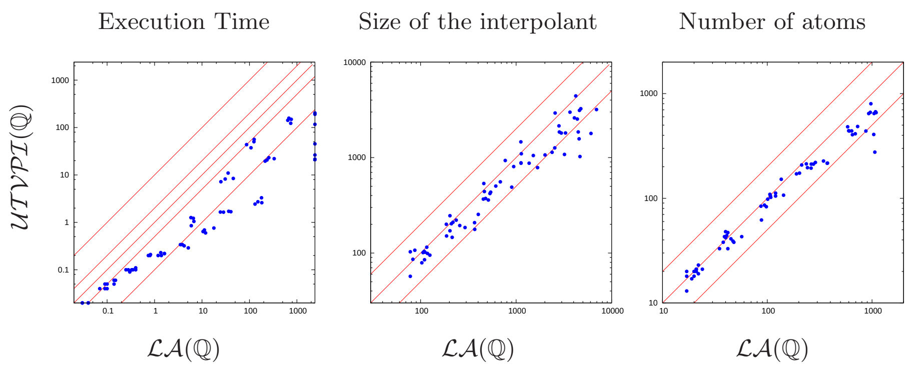

Fig. 7. Comparison between $\mathcal{UTVPI}(\mathbb{Q})$ and $\mathcal{LA}(\mathbb{Q})$ interpolation within MathSAT.

with which Blast can refine the abstraction and successfully prove that the `ERROR` label is not reachable.

6 Conclusions

In this paper we have tackled the problem of generating interpolants in SMT for the theory of Unit-Two-Variable-Per-Inequality ($\mathcal{UTVPI}$), an important fragment of linear arithmetic. Our approach results in interpolants that are within the same theory, and it can be easily implemented on top of efficient graph-based procedures used in many state-of-the-art SMT solvers. Our work covers both the case of rationals, where we experimentally demonstrate the efficiency over a general purpose procedure for $\mathcal{LA}(\mathbb{Q})$, as well as the case of integers, up to now an open problem. In the future, we plan to tackle the full-blown case of interpolation for $\mathcal{LA}(\mathbb{Z})$, where the $\mathcal{UTVPI}$ procedure can be used as a component in a layered approach [23], and to apply SMT-based interpolation procedures in various verification settings, including counterexample guided abstraction refinement.

References

1. T. Ball, B. Cook, S. K. Lahiri, and L. Zhang. Zapato: Automatic Theorem Proving for Predicate Abstraction Refinement. In *Proc. CAV'04*, volume 3114 of *LNCS*. Springer, 2004.
2. D. Beyer, T. A. Henzinger, R. Jhala, and R. Majumdar. The software model checker Blast: Applications to software engineering. *STTT*, 9(5-6):505–525, 2007.
3. D. Beyer, D. Zufferey, and R. Majumdar. CSIsat: Interpolation for LA+EUF. In *CAV*, volume 5123 of *LNCS*. Springer, 2008.
4. R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, and R. Sebastiani. The MathSAT 4 SMT Solver. In *Proc. of CAV*, volume 5123 of *LNCS*, pages 299–303. Springer, 2008.

5. A. Cimatti, A. Griggio, and R. Sebastiani. Efficient Interpolant Generation in Satisfiability Modulo Theories. In *TACAS*, volume 4963 of *LNCS*. Springer, 2008.
6. S. Cotton and O. Maler. Fast and Flexible Difference Constraint Propagation for DPLL(T). In *Proc. SAT*, volume 4121 of *LNCS*. Springer, 2006.
7. C. Flanagan, K. R. M. Leino, M. Lillibridge, G. Nelson, J. B. Saxe, and R. Stata. Extended Static Checking for Java. In *PLDI*, 2002.
8. T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan. Abstractions from proofs. In *POPL*. ACM, 2004.
9. J. Jaffar, M. J. Maher, P. J. Stuckey, and R. H. C. Yap. Beyond Finite Domains. In *PPCP*, volume 874 of *LNCS*. Springer, 1994.
10. H. Jain, E. M. Clarke, and O. Grumberg. Efficient Craig Interpolation for Linear Diophantine (Dis)Equations and Linear Modular Equations. In *CAV*, volume 5123 of *LNCS*. Springer, 2008.
11. R. Jhala and K. L. McMillan. A Practical and Complete Approach to Predicate Refinement. In H. Hermanns and J. Palsberg, editors, *TACAS*, volume 3920 of *LNCS*. Springer, 2006.
12. D. Kapur, R. Majumdar, and C. G. Zarba. Interpolation for data structures. In M. Young and P. T. Devanbu, editors, *SIGSOFT FSE*. ACM, 2006.
13. D. Kroening and G. Weissenbacher. Lifting Propositional Interpolants to the Word-Level. In *FMCAD*, pages 85–89, Los Alamitos, CA, USA, 2007. IEEE Computer Society.
14. S. Krstić, A. Fuchs, A. Goel, J. Grundy, and C. Tinelli. Ground Interpolation for the Theory of Equality. In *Proc. of TACAS*, LNCS. Springer, 2009. To appear.
15. S. K. Lahiri and R. E. Bryant. Deductive Verification of Advanced Out-of-Order Microprocessors. In *CAV*, volume 2725 of *LNCS*. Springer, 2003.
16. S. K. Lahiri and M. Musuvathi. An Efficient Decision Procedure for UTVPI Constraints. In *Proc. of FroCos*, volume 3717 of *LNCS*. Springer, 2005.
17. K. McMillan. Interpolation and SAT-based model checking. In *Proc. CAV*, 2003.
18. K. L. McMillan. An interpolating theorem prover. *Theor. Comput. Sci.*, 345(1), 2005.
19. K. L. McMillan. Lazy Abstraction with Interpolants. In *Proc CAV*, volume 4144 of *LNCS*. Springer, 2006.
20. A. Miné. The Octagon Abstract Domain. In *Proc. of WCRE*, pages 310–319, Washington, DC, USA, 2001. IEEE Computer Society.
21. R. Nieuwenhuis and A. Oliveras. DPLL(T) with Exhaustive Theory Propagation and Its Application to Difference Logic. In *Proc. CAV*, volume 3576 of *LNCS*. Springer, 2005.
22. A. Rybalchenko and V. Sofronie-Stokkermans. Constraint Solving for Interpolation. In *VMCAI*, LNCS. Springer, 2007.
23. R. Sebastiani. Lazy Satisfiability Modulo Theories. *Journal on Satisfiability, Boolean Modeling and Computation, JSAT*, Volume 3, 2007.
24. S. A. Seshia and R. E. Bryant. Deciding Quantifier-Free Presburger Formulas Using Parameterized Solution Bounds. In *LICS*. IEEE Computer Society, 2004.
25. V. Sofronie-Stokkermans. Interpolation in Local Theory Extensions. In *Proc. of IJCAR*, volume 4130 of *LNCS*, pages 235–250. Springer, 2006.
26. G. Yorsh and M. Musuvathi. A combination method for generating interpolants. In *Proc. of CADE*, volume 3632 of *LNCS*, pages 353–368. Springer, 2005.