# An Efficient Decision Procedure for UTVPI Constraints

Shuvendu K. Lahiri      Madanlal Musuvathi

June 15, 2005

Technical Report
MSR-TR-2005-67

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

This page intentionally left blank.

# An Efficient Decision Procedure for UTVPI Constraints

Shuvendu K. Lahiri and Madanlal Musuvathi

Microsoft Research
{shuvendu,madanm}@microsoft.com

**Abstract.** A unit two variable per inequality (UTVPI) constraint is of the form $a.x+b.y \leq d$ where $x$ and $y$ are integer variables, the coefficients $a, b \in \{-1, 0, 1\}$ and the bound $d$ is an integer constant. This paper presents an efficient decision procedure for UTVPI constraints. Given $m$ such constraints over $n$ variables, the procedure checks the satisfiability of the constraints in $O(n.m)$ time and $O(n+m)$ space. This improves upon the previously known $O(n^2.m)$ time and $O(n^2)$ space algorithm based on transitive closure. Our decision procedure is also equality generating, proof generating, and model generating.

## 1 Introduction

A unit two variable per inequality (UTVPI) constraint is of the form $a.x+b.y \leq d$ where $x$ and $y$ are integer variables, the coefficients $a, b \in \{-1, 0, 1\}$ and the bound $d$ is an integer constant. This is a useful fragment of integer linear arithmetic as many hardware and software verification queries are naturally expressed in this fragment.

For example, Ball et al. [1] note that most queries that arise during the predicate abstraction refinement process in SLAM [2] fit into this fragment. Others, including Pratt [18] and Seshia et al. [19] have observed that significant portion of linear arithmetic queries are restricted to difference logic (a fragment of UTVPI constraints of the form $x \leq y + c$.).

The fragment UTVPI is also important because it is the most expressive fragment of linear arithmetic that enjoys a polynomial decision procedure [11]. Extending this fragment to contain three variables (with just unit coefficients) per inequality or adding non-unit coefficients for two variable inequalities can make the decision problem NP-Complete [12]. Having an integer solver is often useful when dealing with variables for which rational solutions are unacceptable. Such examples often arise when modeling indices of an array or queues in hardware or software [7, 13].

In this paper, we present an efficient decision procedure for UTVPI constraints. Given $m$ such constraints over $n$ variables, the procedure checks the satisfiability of the constraints in $O(n.m)$ time and $O(n + m)$ space. This improves upon the previously known $O(n^2.m)$ time and $O(n^2)$ space algorithm provided by Jaffar et al. [11] based on transitive closure. The space improvement of our algorithm is particularly evident when $m$ is $O(n)$, which occurs very frequently in practice, as the number of constraints that arise in typical verification queries have a sparse structure. In fact, the actual complexity of our algorithm is $O(\mathcal{NCD})$, which is the complexity of an algorithm that can determine if there is a negative weight cycle in a directed graph. [1] Accordingly, the time bound of our algorithm can be further improved by using a more efficient negative cycle detection algorithm [4].

In addition to checking satisfiability of a set of UTVPI constraints, the decision procedure is also *equality* generating, *proof* producing and generates models for satisfiable formulas. The decision procedure generates equalities between variables implied by a set of UTVPI constraints in $O(n.m)$ time. The algorithm can generate a proof of unsatisfiability and equalities implied in $O(n.m)$ time. Both these algorithms use a linear $O(n + m)$ space. The model generation algorithm can run in $O(n.m + n^2.logn)$ time and $O(n^2)$ space.

Finally, we conclude the paper by showing that the problem of finding *diverse* models for UTVPI constraints is NP-complete. A diverse model $\rho$ for a set of UTVPI constraints $\phi$ is an assignment from the set of variables of $\phi$ to integers, such that $\rho(x) = \rho(y)$ if and only if $x = y$ is implied by $\phi$. We also relate the problem of generating disjunctions of equalities from the theory to the problem of diverse model generation.

## 2 Background

### 2.1 Requirements from a decision procedure

For a given theory $T$, a decision procedure for $T$ checks if a formula $\phi$ in the theory is *satisfiable*, i.e. it is possible to assign values to the symbols in $\phi$ that are consistent with $T$, such that $\phi$ evaluates to `true`.

Decision procedures, nowadays, do not operate in isolation, but form a part of a more complex system that can decide formulas involving symbols shared across multiple theories. In such a setting, a decision procedure has to support the following operations efficiently:

1. *Satisfiability Checking*: Checking if a formula $\phi$ is satisfiable in the theory.
2. *Model Generation*: If a formula in the theory is satisfiable, find values for the symbols that appear in the theory that makes it satisfiable. This is crucial for applications that use theorem provers for test-case generation.
3. *Equality Generation*: The Nelson-Oppen framework for combining decision procedures [17] requires that each theory (at least) produces the set of equalities over variables that are implied by the constraints.
4. *Proof Generation*: Proof generation can be used to certify the output of a theorem prover [16]. Proofs are also used to construct conflict clauses efficiently in a lazy SAT-based theorem proving architecture [6].

[1] The traditional Bellman-Ford algorithm for negative cycle detection runs in $O(n.m)$ time.

### 2.2 Graph Notations

Let $G(V, E)$ be a directed graph with vertices $V$ and edges $E$. For each edge $e \in E$, we denote $s(e)$, $d(e)$ and $w(e)$ to be the source, destination and the weight of the edge. A *path* $P$ in $G$ is a sequence of edges $[e_1, \ldots, e_n]$ such that $d(e_i) = s(e_{i+1})$, for all $1 \leq i \leq n-1$. For a path $P \doteq [e_1, \ldots, e_n]$, $s(P)$ denotes $s(e_1)$, $d(P)$ denotes $d(e_n)$ and $w(P)$ denotes the sum of the weights on the edges in the path, i.e. $\sum_{1 \leq i \leq n} w(e_i)$. A cycle $C$ is a sequence of edges $[e_1, \ldots, e_n]$ where $s(e_1) = d(e_n)$. We use $u \rightsquigarrow v$ in $E$ to denote that there is a path from $u$ to $v$ through edges in $E$.

### 2.3 Negative Cycle Detection

Given a graph $G(V, E)$, the problem of determining if $G$ has a cycle $C$, such that $w(C) < 0$ is called the *negative cycle detection* problem. Various algorithms can be used to determine the existence of negative cycles in a graph [4]. Negative cycle detection (NCD) algorithms have two properties:

1. The algorithm determines if there is a negative cycle in the graph. In this case, the algorithm produces a particular negative cycle as a witness.
2. If there are no negative cycles, then the algorithm generates a *feasible* solution $\delta : V \rightarrow \mathcal{Z}$, such that for every $(u, v) \in E$, $\delta(v) \leq \delta(u) + w(u, v)$.

For example, the Bellman-Ford [3, 8] algorithm for singe-source shortest path in a graph can be used to detect negative cycles in a graph. The Bellman-Ford algorithm can determine in $O(n.m)$ time and $O(n+m)$ space, if there is a negative cycle in $G$, and a feasible solution otherwise.

In this paper, we assume that we use one such NCD algorithm. We will define the complexity $O(\mathcal{NCD})$ as the complexity of the NCD algorithm under consideration. This allows us to leverage all the advances in NCD algorithms in recent years [4], that have complexity better than the Bellman-Ford algorithm.

## 3 UTVPI Constraints

The unit two variables per inequality (UTVPI) constraints are a fragment of linear arithmetic constraints of the form $a.x + b.y \leq d$ where $x$ and $y$ are integer variables, the coefficients $a, b \in \{-1, 0, 1\}$ and the bound $d$ is an integer constant. The fragment also includes single variable per inequality (SVPI) constraints $a.x \leq d$.

Figure 1 describes the set of inference rules that is sound and complete for this fragment. Jaffar et al. [11] showed that a set of UTVPI constraints $C$ is unsatisfiable if and only if the closure of $C$ with respect to the transitive and the tightening rule in Figure 1, contains a constraint $0 \leq d$, where $d < 0$.

$$\frac{a.x + b.y \leq c \qquad -a.x + b'.z \leq d}{b.y + b'.z \leq c + d} \quad (\textsc{Transitive})$$

$$\frac{a.x + b.y \leq c \qquad a.x - b.y \leq d \qquad a \in \{-1, 1\}}{a.x \leq \lfloor (c + d)/2 \rfloor} \quad (\textsc{Tightening})$$

$$\frac{a.x + b.y \leq c \qquad -a.x - b.y \leq d \qquad c + d < 0}{\perp} \quad (\textsc{Contradiction})$$

**Fig. 1.** Inference rules for UTVPI. The constants $a, b, b'$ range over $\{-1, 0, 1\}$, and $c, d$ range over $\mathcal{Z}$.

## 3.1 Existing Decision Procedures for UTVPI

The only algorithms known for solving a set of UTVPI constraints are based on transitive (and tightening) closure.

Jaffar et al. [11] provided the first decision procedure for UTVPI. The algorithm was based on incrementally processing a set of constraints $C$ and maintaining a transitive and tight closure $C^*$ of the set of constraints seen so far. After the addition of a new constraint $a.x + b.y \leq d$, the algorithm computes the set of new UTVPI constraints as follows:

1. For every $-a.x + b'.z \leq d' \in C^*$, and for every $-b.y + b''.w \leq d'' \in C^*$, compute the closure of $\{-a.x + b'.z \leq d', a.x + b.y \leq d, -b.y + b''.w \leq d''\}$ using the transitive rule in Figure 1.
2. For any constraint $2a'.w \leq d'$ produced in step 1, we add the tightening constraint $a'.w \leq \lfloor d'/2 \rfloor$ to the closure.
3. For each *new* tightening constraint $a'.w \leq d'$ produced in step 2, and for every constraint $b'.z - a'.w \leq d''$, we add the transitive constraint $b'.z \leq d + d''$ to the closure, and compute the transitive closure.

The runtime of the algorithm is $O(m.n^2)$ and the space requirement is $O(n^2)$.

Harvey et al. [9] improved on this algorithm by showing that the transitive and tightening steps can be combined together in a single step (i.e. step 1) without the need for the subsequent steps. The asymptotic complexity of the algorithm (both time and space), however, remains unchanged.

In this paper, we provide an $O(\mathcal{NCD})$ time algorithm based on negative cycle detection that strictly improves upon the previous decision procedures for UTVPI constraints. Also, our algorithm has an $O(n + m)$ space complexity that performs better when $m$ is $O(n)$. On the downside, our algorithm is not incremental and requires all the constraints to be provided at once. Currently, we are using this decision procedure in a lazy SAT-based theorem proving framework [6], where nonincremental decision procedures suffice. However, we hope to make this algorithm incremental in our future work.

| UTVPI | Assoc. Difference Constraints | Graph Edges |
|---|---|---|
| $x - y \leq k$ | $x^+ - y^+ \leq k$ , $y^- - x^- \leq k$ | $y^+ \xrightarrow{k} x^+$ , $x^- \xrightarrow{k} y^-$ |
| $x + y \leq k$ | $x^+ - y^- \leq k$ , $y^+ - x^- \leq k$ | $y^- \xrightarrow{k} x^+$ , $x^- \xrightarrow{k} y^+$ |
| $-x - y \leq k$ | $x^- - y^+ \leq k$ , $y^- - x^+ \leq k$ | $y^+ \xrightarrow{k} x^-$ , $x^+ \xrightarrow{k} y^-$ |
| $x \leq k$ | $x^+ - x^- \leq 2.k$ | $x^- \xrightarrow{2.k} x^+$ |
| $-x \leq k$ | $x^- - x^+ \leq 2.k$ | $x^+ \xrightarrow{2.k} x^-$ |

**Table 1.** Edges in constraint graphs

## 3.2 Constraint Graph

Given a set of UTVPI constraints, our algorithm reduces the problem of checking the satisfiability of the constraints to finding negative cycles in an appropriate graph. This transformation is a simple extension of a similar transformation for difference constraints [5] and has been previously used by Miné [15], for solving UTVPI constraints over rationals.

Let $\phi$ be a set of UTVPI constraints $\phi$. One can construct the *constraint graph* $G_\phi(V, E)$ as follows. For each variable $x$ in $\phi$, the vertex set $V$ contains two vertices $x^+$ and $x^-$ that respectively represent the positive and negative occurrences of $x$. For any vertex $v \in V$, $-v$ represents its negative counterpart. That is, $-x^+$ represents $x^-$, and $-x^-$ represents $x^+$. To avoid confusion, we will use $x, y, z, \ldots$ to represent variables in $\phi$ and $u, v, w, \ldots$ to represent vertices in $V$.

Each UTVPI constraint in $\phi$ can be transformed to a set of difference constraints over vertices as shown in Table 1. For each such difference constraint $u - v \leq k$, the graph $G_\phi(V, E)$ contains an edge $(v, u)$ with weight $k$. It is obvious that if $\phi$ contains $m$ UTVPI constraints in $n$ variables then $G_\phi(V, E)$ will contain at most $2.n$ vertices and $2.m$ edges. The following propositions are obvious.

**Proposition 1.** *For every edge $(u, v) \in E$, the constraint graph $G_\phi(V, E)$ contains an edge $(-v, -u) with equal weight$.*

**Proposition 2.** *If there is a path $P$ from $u$ to $v$ in the constraint graph, then there is a path $P'$ from $-v$ to $-u$ such that $w(P) = w(P')$.*

For vertices $u, v \in V$, let $SP(u, v)$ represent any of the shortest path from $u$ to $v$ in $G_\phi$. Let $wSP(u, v)$ represent $w(SP(u, v))$.

**Proposition 3.** *Let $u$ and $v$ two vertices in $G_\phi$ such that $u \rightsquigarrow v$. Also, let the vertices respectively represent variables $a.x$ and $b.y$. Then $\phi$ implies the tightest bound $b.y - a.x \leq k$ exactly when $k = wSP(u, v)$*

The proof follows by transitivity of constraints in $\phi$.

**Lemma 1.** *A set of UTVPI constraints $\phi$ is unsatisfiable in $\mathcal{Q}$ if and only if the constraint graph $G_\phi(V, E)$ contains a negative weight cycle [15].*

$$y + x \le -5 \tag{1}$$
$$w - x \le 4 \tag{2}$$
$$-w - x \le 3 \tag{3}$$
$$z - y \le 2 \tag{4}$$
$$-z - y \le 1 \tag{5}$$

**Fig. 2.** UTVPI constraints that are unsatisfiable in $\mathcal{Z}$, but satisfiable in $\mathcal{Q}$

The proof of the lemma simply follows from a similar proof for the satisfiability of difference constraints [5].

Lemma 1 essentially solves the satisfiability problem for rationals. However, this lemma is not sufficient for integers. For instance, Figure 3.2 shows a set of constraints which is unsatisfiable in integers but the corresponding constraint graph has no negative cycles. In this example, Equations (2) and (3) imply $-2.x \le 7$ which for integers can be tightened to $-x \le 3$. Similarly, Equations (4) and (5) imply $-y \le 1$. These bounds on $-x$ and $-y$ contradict Equation (1).

## 4 Efficient Decision Procedure for UTVPI Constraints

As described in the previous section, the constraint graph $G_\phi$ of a set of constraints $\phi$ contains a negative cycle only if the $\phi$ are unsatisfiable in $\mathcal{Q}$. To extend this result for $\mathcal{Z}$, this section describes a method to extend the constraint graph by adding *tightening* edges. The resulting graph contains a negative cycle exactly when the constraints are unsatisfiable in $\mathcal{Z}$.

### 4.1 Tightening Edges

Given a constraint graph $G_\phi(V, E)$, our goal is to capture the tightening rule in Figure 1. For a constraint graph $G_\phi$, define the set of tightening edges $T$ as follows:

$$T = \{(u, -u) \mid wSP(u, -u) \text{ is odd}\}$$

For each edge in $T$, the weight of the edge is defined as follows

$$w_T(u, -u) = wSP(u, -u) - 1, \text{ for all edges } (u, -u) \in T$$

Now, whenever $\phi$ implies the tightest bound $2.x \le k$ where $k$ is odd, then by Proposition 3, $wSP(x^-, x^+) = k$ in $G_\phi$. By the Tightening Rule in Figure 1, $\phi$ implies $2.x \le k - 1$. This "tightened" constraint is captured by the tightening edge $(x^-, x^+)$ in $T$. Note, that the weight of an edge in $T$ is even.

Given a constraint graph $G_\phi$, define the graph $G_{\phi \cup T}$ as the one obtained by adding all edges in $T$ to $G_\phi$. The following lemma describes a way to check if the input constraints $\phi$ is satisfiable in $\mathcal{Z}$.

**Lemma 2.** *A set of UTVPI constraints $\phi$ is satisfiable in $\mathcal{Z}$ if and only if the graph $G_{\phi \cup T}$ has a negative cycle.*

Lemma 2 leads to the following naive algorithm

**Proposition 4.** *Naive Algorithm: The set of UTVPI constraints can decided in $O(nm + n^2 logn)$ time and $O(n + m)$ space.*

This algorithm uses a minor modification of Johnson's *All Pair Shortest Paths* algorithm to identify the edges in $T$ in $O(nm + n^2 logn)$ time and $O(n + m)$ space. Then, negative cycles in $G_{\phi \cup T}$ can be found in $O(\mathcal{NCD})$. Note, that this is an improvement over the Harvey and Stuckey's algorithm.

## 4.2 Efficient Decision Procedure

Our goal is to improve upon the naive algorithm to decide a set of UTVPI constraints in $O(\mathcal{NCD})$ time. The crucial insight is to notice that the naive algorithm is computing *all* edges in $T$ while only some of them might potentially lead to negative cycles. Our algorithm precisely identifies those edges in $T$ by looking for *zero-weighted* cycles in $G_\phi$.

We present the algorithm below:

**Algorithm 1** NCD-UTVPI *Algorithm:*

```
1. Given a set of UTVPI constraints $\phi$, construct the constraint graph $G_\phi(V, E)$.
2. Run a negative cycle detection algorithm.
   (a) If $G_\phi$ contains a negative cycle, return UNSAT.
   (b) Otherwise, the negative cycle detection algorithm returns a feasible solution $\delta$, such that $\delta(v) - \delta(u) \leq w(u, v)$ for all edges $(u, v) \in E$.
3. Let $E'$ be set of edges in $G$ such that an edge $(u, v) \in E'$ if and only if $\delta(v) - \delta(u) = w(u, v)$
4. Create the induced subgraph $G'_\phi(V, E')$ from $G_\phi(V, E)$.
5. Group the vertices in $G'_\phi$ into *strongly connected components* (SCCs). Vertices $u$ and $v$ are in the same SCC if and only if $u \rightsquigarrow v$ and $v \rightsquigarrow u$ in $E'$. This can be done in linear time [20]. Moreover, $u$ and $v$ are in the same SCC exactly when there is a zero-weight cycle in $G_\phi$ containing $u$ and $v$.
6. For each vertex $u \in V$,
   (a) if $-u$ is in the same SCC as $u$ and if $\delta(u) - \delta(-u)$ is odd, then return UNSAT.
7. return SAT
```

In the algorithm above, all steps except step 2 can be done in linear time. Thus the algorithm has $O(\mathcal{NCD})$ time complexity and $O(n+m)$ space complexity. To prove the soundness and completeness of this algorithm, we need the following definitions and lemmas.

Given a feasible solution $\delta$ for the constraint graph $G_\phi$, define the *slack* of an edge as: $sl_\delta(u, v) = \delta(u) - \delta(v) + w(u, v)$. When the feasible solution $\delta$ is obvious from the context, $sl(u, v)$ refers to $sl_\delta(u, v)$. From the definition of feasibility of

$\delta$, we have the fact that $sl(u,v) \geq 0$ for all edges $(u,v)$ in $G_\phi$. Note, the step 3 of the algorithm above identifies $E'$ to be the set of edges with slack zero. Also, the slack of a path $P$ is defined as $sl(P) = \Sigma_{e \in P} sl(e)$.

**Proposition 5.** *Let $P$ be a path from $u$ to $v$ in $G_\phi$. Then, $w(P) = sl(P) + \delta(v) - \delta(u)$.*

The proof follows from a simple induction on the length of the path $P$.

**Proposition 6.** *For any cycle $C$ in $G_\phi$, $w(C) = sl(C)$.*

The proof follows from Proposition 5 when $P$ forms a cycle.

**Proposition 7.** *If $P$ is a path from $u$ to $v$ and all edges in $P$ have a slack zero, then $wSP(u,v) = \delta(v) - \delta(u)$.*

*Proof.* We have $sl(P) = 0$. Thus $w(P) = \delta(v) - \delta(u)$ from Proposition 5. However, as slacks of all edges are nonnegative, $sl(SP(u,v)) \geq 0$. From Proposition 5, we have $wSP(u,v) \geq w(P)$, which can be possible only when the inequality is tight. Thus, the proposition is true.

**Theorem 1.** *The* NCD-UTVPI *algorithm is sound.*

*Proof.* The algorithm returns UNSAT at two places. In step 2a, the graph $G_\phi$ contains a negative cycle and thus by Lemma 1, $\phi$ is unsatisfiable in $\mathcal{Q}$ and thus in $\mathcal{Z}$. If the algorithm returns UNSAT in step 6a, then we show that $\phi$ is unsatisfiable in $\mathcal{Z}$ (but satisfiable in $\mathcal{Q}$). We have two vertices $u$ and $-u$ that are in the same SCC in $G'_\phi$ such that $\delta(-u)-\delta(u)$ is odd. Since the vertices are in the same SCC, we have a path in from $u$ to $-u$ in $G'_\phi$. By Proposition 7, $wSP(u,-u)$ is odd, and thus $T$ contains an edge $(u,-u)$ of weight $\delta(-u)-\delta(u)-1$. Similarly, $T$ contains an edge $(-u,u)$ of weight $\delta(u)-\delta(-u)-1$. These two edges form a negative cycle (of weight $-2$) in $G_{\phi \cup T}$. Thus, $\phi$ is unsatisfiable by Lemma 2.

To prove the completeness, we have to show that given a set of UTVPI constraints $\phi$ that is unsatisfiable, the *NCD-UTVPI* algorithm returns UNSAT. The proof of this theorem is more involved, and requires the following lemmas.

**Lemma 3.** *If $C$ is a cycle in $G_{\phi \cup T}$, then there is a cycle $C'$ in $G_{\phi \cup T}$ with at most two edges from $T$ such that either $w(C') < 0$ or $w(C') \leq w(C)$.*

*Proof.* This lemma is crucial for restricting negative cycle detection to those cycles with at most two tightening edges. The proof of this lemma is as follows. The lemma is trivially true if $G_{\phi \cup T}$ only contains cycles with at most two edges from $T$. Otherwise, let $C$ be a cycle in $G_{\phi \cup T}$ such that $C$ contains $n$ tightening edges with $n > 2$. For $0 \leq i < n$, let $t_i = (v_i, -v_i)$ be the tightening edges in the order they appear in $C$. Also, the fragment $P_i$ of $C$ denotes a path from $-v_i$ to $v_{i+1}$ in $G_\phi$, where the addition is performed modulo $n$. From Proposition 2, there is a path $P'_i$ from $-v_{i+1}$ to $v_i$ in $G_\phi$, such that $w(P'_i) = w(P_i)$. Define $C_i$ as the cycle consisting of $t_i, P_i, t_{i+1}, P'_i$. Obviously, $C_i$ contains only two tightening edges, for $0 \leq i < n$.

We can show that at least one of the $C_i$ satisfies the conditions of the lemma. Suppose this is not the case, then $w(C_i) \geq 0$ and $w(C_i) > w(C)$, for all $0 \leq i < n$. We have,

$$\begin{aligned}
\textstyle\sum_{i=0}^{n-1} w(C_i) &= \Sigma_{i=0}^{n-1} w(t_i) + w(P_i) + w(t_{i+1}) + w(P_i') \\
&= \Sigma_{i=0}^{n-1} w(t_i) + 2 * w(P_i) + w(t_{i+1}) && \text{as } w(P) = w(P') \\
&= \Sigma_{i=0}^{n-1} 2 * w(t_i) + 2 * w(P_i) && \text{by reordering the sum} \\
&= 2 * w(C)
\end{aligned}$$

By assumption, the left hand side $\Sigma_{i=0}^{n-1} w(C_i) \geq 0$, which implies that $w(C) \geq 0$. Also by assumption, $2 * w(C) = \Sigma_{i=0}^{n-1} w(C_i) > n * w(C)$. However, this contradicts with the fact that $n > 2$. Thus, at least there is a $C_i$ that satisfies the requirements of the lemma.

**Corollary 1.** *$G_{\phi \cup T}$ contains a negative cycle precisely when it contains a negative cycle with at most most two edges from $T$.*

**Lemma 4.** *Suppose $G_\phi$ contains no negative cycles but $G_{\phi \cup T}$ contains a negative cycle. Then there is a zero weight cycle in $G_\phi$ containing vertices $u$ and $-u$ such that $SP(u, -u)$ is odd.*

*Proof.* Let $C$ be a negative cycle in $G_{\phi \cup T}$. By Corollary 1, $C$ has at most two tightening edges without loss of generality. Since $G_\phi$ contains no negative cycles, $C$ contains at least one tightening edge. Thus, there are the following two cases:
*Case 1:* $C$ contains exactly one tightening edge $(u, -u)$. Define $P$ as the fragment of $C$ from $-u$ to $u$. Consider the cycle $C'$ formed by $P$ along with $SP(u, -u)$. By definition of the tightening edge, $w(C) = wSP(u, -u) - 1 + w(P) = w(C') - 1 < 0$. Also, as $C'$ is a cycle in $G_\phi$, we have $w(C') \geq 0$. These constraints imply that $w(C') = 0$ and is the cycle required by the lemma.
*Case 2:* $C$ contains two tightening edges. Let $(u, -u)$ and $(v, -v)$ be the tightening edges in order they appear in $C$. Define $P_u$ as the fragment of $C$ from $-u$ to $v$ and define $P_v$ as the fragment of $C$ from $-v$ to $u$. Without loss of generality, $w(P_u) \leq w(P_v)$. Also, by Proposition 2, there is a path $P_u'$ from $-v$ to $u$, such that $w(P_u) = w(P_u')$. Consider the cycle $C'$ consisting of $SP(u, -u), P_u, SP(v, -v), P_u'$. By definition of tightening edges both $wSP(u, -u)$ and $wSP(v, -v)$ are odd. Thus, $w(C')$ is even. Also,

$$\begin{aligned}
w(C) &= wSP(u, -u) - 1 + w(P_u) + wSP(v, -v) - 1 + w(P_v) \\
&\geq wSP(u, -u) - 1 + w(P_u) + wSP(v, -v) - 1 + w(P_u) && \text{by assumption} \\
&= w(C') - 2 && \text{as } w(P_u) = w(P_u')
\end{aligned}$$

Thus, $w(C') \leq w(C) + 2$. Since $C$ is a negative cycle, we have $w(C') \leq 1$. However, we can tighten this constraint as $w(C')$ is even. Thus, we have $w(C') \leq 0$. Since, $C'$ is a cycle in $G_\phi$ $w(C') \geq 0$. Thus, $C'$ is the cycle required by the lemma.

The proof of completeness of the *NCD-UTVPI* algorithm follows from the above lemma.

**Theorem 2.** *The* NCD-UTVPI *algorithm is complete.*

*Proof.* Let $\phi$ be a set of UTVPI constraints. If $\phi$ is unsatisfiable in $\mathcal{Q}$, then the constraint graph $G_\phi$ has a negative cycle by Lemma 1. The NCD-UTVPI algorithm returns UNSAT in step 2a. Suppose $\phi$ is satisfiable in $\mathcal{Q}$ but unsatisfiable in $\mathcal{Z}$. Then, by Lemma 2, $G_{\phi\cup T}$ contains a negative cycle, while $G_\phi$ does not. Thus, by Lemma 4, $G_\phi$ contains a zero weight cycle with a vertex $u$ such that $SP(u,-u)$ is odd. By Proposition 6, all edges in $C$ have a slack equal to zero. Thus $u$ and $-u$ are in the same SCC in the graph $G'$ defined in step 5 of the NCD-UTVPI algorithm. Finally, $SP(u,-u) = \delta(-u,u)$ by Proposition 7. Thus, the NCD-UTVPI algorithm will detect the vertex $u$ in step 6a.

## 5 Equality Generation

This section describes how to generate variable equalities implied by a set of UTVPI constraints. This is very essential when the UTVPI decision procedure is combined with other theories in a Nelson-Oppen framework. Given a set of $m$ UTVPI constraints $\phi$ over $n$ variables, we show how to infer variable equalities from $G_\phi$ in $O(n.m)$ time and $O(n+m)$ space.

### 5.1 Naive Algorithm

Akin to the decision procedure described in Section 4, we first provide a naive algorithm for generating equalities in $O(nm + n^2 logn)$ time to capture the main intuition, and then improve to a $O(nm)$ algorithm. Though $m$ can be $n^2$ in the worst case, this improvement is motivated by the fact that in practice $m$ is $O(n)$.

The naive algorithm proceeds as in Proposition 4 by explicitly constructing $G_{\phi\cup T}$ by identifying all tightening edges in $O(nm + n^2 logn)$ time. Given $G_{\phi\cup T}$ with no negative cycles (§2), the following lemma provides a way to generate variable equalities in $O(\mathcal{NCD})$ time

**Lemma 5.** *Let $\delta_{\phi\cup T}$ be a feasible solution produced by a negative cycle detection algorithm for $G_{\phi\cup T}$. The set of constraints $\phi$ implies $x = y$ exactly when the following is true:*

1. *$\delta_{\phi\cup T}(x^+) = \delta_{\phi\cup T}(y^+)$, and*
2. *there is a zero weight cycle in $G_{\phi\cup T}$ that contains both $x^+$ and $y^+$.*

*Proof.* This lemma is directly derived from the equality generation algorithm for difference constraints [14]. First, we will show that the conditions in the lemma imply a variable equality. By the second condition, there is a path $P_{xy}$ from $x^+$ to $y^+$ and a path $P_{yx}$ from $y^+$ to $x^+$ such that $sl(P_{xy}) = sl(P_{yx}) = 0$. By Proposition 7, we have $wSP(x^+,y^+) = \delta_{\phi\cup T}(y^+) - \delta_{\phi\cup T}(x^+) = 0$. Moreover, by Proposition 3, $\phi$ implies $x - y \leq 0$. Similarly, $\phi$ implies $y - x \leq 0$, which together imply that $x = y$. The proof of the other direction is similar.

The conditions in this lemma can be checked by performing a SCC computation on the subgraph of $G_{\phi \cup T}$ induced by edges with slack zero, similar to the *NCD-UTVPI* algorithm. Now, $\phi$ implies $x = y$ whenever $x$ and $y$ are in the same SCC with the same $\delta_{\phi \cup T}$ values. This can be done in average linear time using a hashtable, or in $O(nlogn)$ time by sorting all vertices in the same SCC according to their $\delta_{\phi \cup T}$ values.

We state the naive algorithm below.

**Algorithm 2** *EqGen-Naive Algorithm:*

1. Starting with $G_{\phi \cup T}$, run a negative cycle detection algorithm to produce a feasible solution $\delta_{\phi \cup T}$.
2. Let $E_0$ be the set of edges such that $e \in E_0$ if and only if $sl(e) = 0$.
3. Create the subgraph $G_0$ induced by $E_0$.
4. Group the vertices of $G_0$ into strongly connected components.
5. If vertices $x^+$ and $y^+$ are in the same SCC and $\delta_{\phi \cup T}(x^+) = \delta_{\phi \cup T}(y^+)$, then report the variable equality $x = y$.

## 5.2 Efficient Equality Generation

In this section, we improve the naive algorithm by precisely inferring those tightening edges that can result in a zero-weighted cycle in $G_{\phi \cup T}$. We do this by using the following lemma, similar to Lemma 4.

**Lemma 6.** *Assuming $G_{\phi \cup T}$ has no negative cycles and if $C$ is a zero weight cycle in $G_{\phi \cup T}$ containing a tightening edge $(u, -u)$, then there is a cycle $C'$ in $G_\phi$ containing $u$ and $-u$, and such that $w(C') \leq 2$.*

*Proof.* By Lemma 3, we can assume that $C$ has at most two tightening edges. Let $P$ be the fragment of $C$ from $-u$ to $u$. Consider the cycle $C_1$ formed by $SP(u, -u), P$. By definition of the tightening edge, this cycle has weight $w(C_1) = wSP(u, -u) + w(P) = w(C) + 1 = 1$. If $C$ contains no other tightening edge, then $C_1$ is a cycle in $G_\phi$ proving the lemma. Otherwise, $C$ contains at most one other tightening edge, which can be removed like before to obtain $C_2$ such that $w(C_2) = 2$. Now, $C_2$ is the required cycle in $G_\phi$.

By Lemma 6, we only need to add tightening edges whose endpoints are in cycles of weight less than or equal to two. Moreover, under any feasible solution $\delta$, any edge $e$ in such a cycle will have $sl(e) \leq 2$. By identifying such edges, the following algorithm generates all variable equalities implied by $\phi$ in $O(n.m)$ time.

**Algorithm 3** *EqGen Algorithm:*

1. Given a set of UTVPI constraints $\phi$, construct the constraint graph $G_\phi(V, E)$.
2. Assume *NCD-UTVPI* algorithm returns SAT. Also, assume that a feasible solution $\delta$ exists for $G_\phi$.

3. Let $E_2 = \{(u,v) | sl(u,v) \leq \mathit{2}\}$, and let $G_2(V, E_2)$ be the subgraph of $G_\phi$ induced by $E_2$.
4. Let $T_2$ be the set of tightening edges, initially set to the empty set.
5. For each vertex $v$
   (a) Find $P_v$ the path in $G_2$ from $v$ to $-v$ with the smallest slack, if any. This can be done in $O(m)$ by using a modified breadth-first-search.
   (b) If $P_v$ exists,
      i. Let $wSP(v, -v) = \delta(-v) - \delta(v) + sl(P_v)$.
      ii. If $wSP(v, -v)$ is odd, then add the edge $(v, -v)$ to $T_2$ and assign a weight $w_{T_2}(v, -v) = wSP(v, -v) - 1$.
6. Consider the graph $G_{\phi \cup T_2}$ obtained by adding all the edges in $T$.
7. Now proceed as in the EqGen-Naive algorithm with $G_{\phi \cup T_2}$ instead of $G_{\phi \cup T}$.

**Proposition 8.** *When the EqGen algorithm computes $wSP(v, -v)$ in step 5(b)i, the value computed correctly represents the weight of the shortest path between $v$ and $-v$.*

**Lemma 7.** *$G_{\phi \cup T}$ contains a zero weight cycle exactly when $G_{\phi \cup T_2}$ contains a zero weight cycle.*

*Proof.* Since $G_{\phi \cup T_2}$ is a subgraph of *tightT*, one way of the proof is trivial. Suppose *tightG* contains a zero weight cycle $C$. If $C$ has no tightening edges, then $C$ is a cycle in $G_{\phi \cup T_2}$. Otherwise, let $(u, -u)$ be a tightening edge in $C$. By Lemma 6, there is a cycle $C'$ in $G_\phi$ such that $w(C') \leq 2$ and $C'$ contains $u$ and $-u$. Since $sl(C') \leq \mathit{2}$, all edges in $C'$ have a slack less than or equal to 2, and thus are in $E_2$. Thus, the EqGen algorithm will add the tightening edge in $T_2$.

## 6 Proof Generation

Using the UTVPI decision procedure in a lazy-proof-explication framework requires the procedure to produce proofs whenever it reports the input constraints as unsatisfiable, or whenever it propagates an implied variable equality. This section describes how to generate the proofs for the decision procedure described in this paper.

Both the *NCD-UTVPI* algorithm and the EqGen algorithm rely on two vertices $u$ and $v$ (say) being in the same SCC in an appropriate graph. As a witness for this fact, we need a path from $u$ to $v$ and a path from $v$ to $u$. We assume that the standard SCC algorithm can be modified to provide this witness.[2].

### 6.1 Generating Unsatisfiability Proofs

The *NCD-UTVPI* algorithm returns UNSAT in two cases. In the first case, the constraint graph $G_\phi$ has a negative cycle. By assumption, the negative cycle

[2] For details refer [14]

detection algorithm produces a witness which is the proof of unsatisfiability of the input constraints. In the second case, the algorithm produces a vertex $u$ and $-u$ in the same SCC of $G'_\phi$. Applying the transitivity rule along the path from $u$ to $-u$, we have $-2.a.x \leq k$ where $a.x$ is the variable corresponding to $u$ and $k = \delta(-v) - \delta(v)$ is an odd number. By applying the tightening rule, we get $-2.a.x \leq k - 1$. Using the path from $-u$ to $u$ we have $2.a.x \leq -k$ which by tightening we get $2.a.x \leq -k - 1$. This is the proof of the contradiction.

### 6.2 Generating Proofs for Implied Variable Equalities

The EqGen algorithm detects zero-weight cycles in the graph $G_{\phi \cup T_2}$. First, every tightening edge in $G_{\phi \cup T_2}$ has a proof involving transitivity along a particular path of odd length, followed by a tightening rule. Also, whenever the EqGen algorithm reports an equality $x = y$, $x^+$ and $y^+$ are in the same SCC. The proof of this equality can be inferred along the proof of Lemma 5.

## 7 Model Generation

In this section, we describe how to generate models for a set of constraints $C$, when $C$ is satisfiable.

Let $\rho$ be a function that maps each variable to an integer. Let $\preceq$ be a linear order of the variables that appear in $C$. The assignment $\rho$ is constructed using the following algorithm that assigns values to the variables in the order $\preceq$:

1. Construct the set of UTVPI constraints $C^*$ that is the closure of $C$ under transitivity and tightening.
2. $V_\rho \leftarrow \{\}$.
3. For each variable $x \in V$ in the order $\preceq$:
   (a) Obtain the set of bounds for $x$ ($B_x$) as follows:
$$\begin{aligned} B_x \doteq\ & \{x \leq c - b.\rho(y) \mid x + b.y \leq c \in C_x, \text{ and } y \preceq x\} \\ & \cup \{x \geq -c + b.\rho(y) \mid -x + b.y \leq c \in C_x, \text{ and } y \preceq x\} \\ & \cup \{x \leq c \mid x \leq c \in C^*\} \\ & \cup \{x \geq -c \mid -x \leq c \in C^*\}. \end{aligned}$$
   (b) Assign $\rho(x)$ to be a value that satisfies all the bounds in $B_x$.
   (c) $V_\rho \leftarrow V_\rho \cup \{x\}$.
4. Return $\rho$.

At any point in the above algorithm, $\rho$ has assigned a subset of the variables ($V_\rho$) in $C^*$ values over integers. For all the variables for which $\rho$ is undefined, $\rho(x) = x$. If $\rho$ be such a partial assignment, let us define $C_\rho$ to be the set of constraints $C \cup \{y = \rho(y) \mid y \in V_\rho\}$. It is easy to see that at any point in the above algorithm, $C_\rho$ implies $x \neq c$ for any $x \in V$ if and only if $C_\rho$ implies either $x < c$ or $x > c$.

**Lemma 8.** *A set of UTVPI constraints $C$ implies $x \neq d$ for some $d \in \mathcal{Z}$ if and only if $C$ implies $x < d$ or $C$ implies $x > d$.*

*Proof.* We only prove the "if" direction of this lemma. The proof relies on the following two claims:

1. The set of constraints $C' \doteq C \cup \{x = d\}$ is unsatisfiable (or equivalently $C \Rightarrow x \neq d$) if and only if there is negative cycle in the constraint graph (described in Section 4.1) after adding all the difference constraints for each constraint in $C'$ and all the resulting tightening edges to the graph.
2. Adding $x = d$ to the set $C$ does not imply any new tightening constraints.

The property 1 follows from Lemma 2. Since any tightening constraint for a variable $y_1$ can only result from two constraints $y_1 - y_2 \leq c_1$ and $y_1 + y_2 \leq c_2$ and the constraint $x = d$ can't give rise to a constraint with two variables (under transitive and tightening steps), the property 2 holds.

Therefore, adding the edges $x^+ - x^- \leq 2.d$ and $x^- - x^+ \leq -2.d$ for the constraint $x = d$ can result in a negative cycle in the difference graph, if and only there is a path from $x^-$ to $x^+$ of length less than $-2.d$ (that implies $x > d$) or there is a path from $x^+$ to $x^-$ of length less than $2.d$ (that implies $x < d$).

**Lemma 9.** *During the step 3a of the above algorithm, the set of constraints $C_\rho$ implies $x \neq c$ if and only if $B_x$ contains either $x \leq d$ with $d < c$ or $x \geq d$ with $d > c$.*

*Proof.* We will only prove the "if" direction. The other direction is obvious.

Let us first define the *partial evaluation* of the constraints in $C^*$ with respect to $\rho$, $\langle C^* \rangle_\rho \doteq \{a.\rho(x) + b.\rho(y) \leq c \mid a.x + b.y \leq c \in C^*\}$. If $C^*$ is the transitive and tight closure of $C$, then the transitive and tight closure of $C \cup \{y = c_1\}$ can be obtained by simply replacing $y$ with $c_1$ in all the constraints in $C^*$ (see [9]). Hence, at any point in the above algorithm, $\langle C^* \rangle_\rho$ represents the transitive and tight closure of $C_\rho$.

Now, let us assume that $C_\rho$ implies $x \neq c$. Clearly, either $x \leq d$ with $d < c$ or $x \geq d$ with $d > c$ is present in $B_x$ (by Lemma 8).

**Theorem 3.** *The assignment $\rho$ generated by the algorithm above satisfies all the constraints in $C$.*

The proof follows from a simple induction on the size of $V_\rho$. At each point in the algorithm, the assignment $\rho$ and the set of constraints $C$ are consistent.

An easy implementation of the above algorithm can be obtained in $O(n^3)$ time and $O(n^2)$ space. The transitive and tight closure $C^*$ can be computed by first constructing the difference graph for $C$ as described in Section 3 and performing the transitive closure. After the transitive closure operation, if the edge between $x^+$ and $x^-$ is odd for any variable $x$ in the difference graph, then we add the corresponding tightening constraint for $x$. Finally, we traverse all the constraints in the transitive closure and generate bounds for other variables that are implied by the tightened edges. The rest of the algorithm can be performed in $O(n^2)$ time if we maintain a adjacency list representation of the edges, where the edge list for a variable $x$ contains only those constraints in $C^*$ involving variables that precede $x$ in $\preceq$. This would ensure a single pass over the edges in $C^*$. The time complexity can be improved to $O(n.m + n^2.lg(n))$, when using Johnson's [5] algorithm for performing the all-pair shortest path.

## 7.1 Model generation in Nelson-Oppen Framework

For a conjunction of UTVPI constraints $C$, the assignment $\rho$ computed above satisfies all the constraints in $C$. However, this is not sufficient to produce a model in the Nelson-Oppen combination framework. We assume that the user is familiar with the high-level description of the Nelson-Oppen combination method [17]. Consider the following example where a formula involves the logic of equality with uninterpreted functions (EUF) and UTVPI constraints.

Let $\psi = (f(x) \neq f(y) \wedge x \leq y)$ be a formula in the combined theory. Nelson-Oppen framework will add $\psi_1 \doteq f(x) \neq f(y)$ to the EUF theory $(T_1)$ and $\psi_2 \doteq x \leq y$ to the UTVPI theory $(T_2)$. Since there are no equalities implied by either theory, and each theory $T_i$ is consistent with $\psi_i$, the formula $\psi$ is satisfiable. Now, the UTVPI theory generates the model $\rho \doteq \langle x \mapsto 0, y \mapsto 0\rangle$ for $\psi_2$. However, this is not a model for $\psi$.

To generate an assignment for the variables that are shared across two theories, each theory $T_i$ needs to ensure that the variable assignment $\rho$ for $T_i$ assigns two shared variables $x$ and $y$ equal values if and only if the equality $x = y$ is implied by the constraints in theory $T_i$.

**Definition 1.** *For a set of UTVPI constraints $C$, an assignment $\rho$ for the variables in $C$ is called diverse, if for any two variables $x$ and $y$ in $vars(C)$, $\rho(x) = \rho(y)$ if and only if $x = y$ is implied by $C$.*

We will show that the problem of checking if a set of UTVPI constraints has a diverse model is NP-Complete. Clearly, the problem is in NP. We will reduce the following (NP-complete) pipeline scheduling problem to this problem to show NP-hardness.

**Theorem 4 (Minimum precedence constrained sequencing with delays [10]).** *Given a set $T$ of tasks, a directed acyclic graph $G(T, E)$ defining precedence constraints for the tasks, a positive integer $D$, and for each task $t$ an integer delay $0 \leq d(t) \leq D$. The problem of determining a one-processor schedule for $T$ that obeys the precedence constraints and delays, within a deadline $k$, is NP-Complete. That is, checking if there exists an injective function for the start times $S : T \rightarrow \mathcal{Z}$ such that for every $(t_i, t_j) \in E$, $S(t_j) - S(t_i) > d(t_i)$, and $S(t_i) \leq k$ for all $t_i \in T$ is NP-complete.*

It is easy to construct a set of UTVPI constraints $C$ such that $C$ has a diverse model if and only if a one-processor schedule for tasks in $T$ exists. Observe that the problem can be mapped to the fragment of UTVPI that contains only $x - y < c$ constraints with positive $c$.

**Corollary 2.** *For a given set of constraints $C \doteq \{x_i - y_i < c_i \mid c_i \geq 0\}$, the complexity of finding a diverse model for $C$ is NP-complete.*

We now show an interesting consequence of Corollary 2 for combining the UTVPI theory with other theories in Nelson-Oppen setting.

### 7.2 Generating disjunction of equalities

To combine the decision procedures of a non-convex theory $T_1$ (e.g. UTVPI) with a convex theory $T_2$ (e.g. EUF) in Nelson-Oppen framework, $T_1$ needs to infer the *strongest* disjunction of equalities between variables that is implied by the set of $T_1$ constraints. That is, if $C_1$ is the set of UTVPI constraints, then we need to generate a disjunction of equalities $\{e_1, \ldots, e_k\}$ over variables such that $C_1 \Rightarrow e_1 \vee \ldots \vee e_k$, and the disjunction of no proper subset of $\{e_1, \ldots, e_k\}$ is implied by $C_1$.

**Theorem 5.** *For a set of UTVPI constraints $C$ that does not imply any equality between variables in $vars(C)$, $C$ has a diverse model $\rho$ if and only if $C$ does not imply any disjunction of equalities $\{e_1, \ldots, e_k\}$ (for $k > 1$) over pairs of variables in $vars(C)$.*

*Proof.* Throughout this proof we consider a set of constraints $C$ that does not imply any equality between a pair of variables.

First, let us assume that $C$ implies a disjunction over a minimal set of equalities $\{u_1 = v_1, \ldots, u_k = v_k\}$, for $k > 1$. This means that any model $\rho$ for $C$ is also a model for $C \wedge (u_1 = v_1 \vee \ldots u_k = v_k)$, and therefore must satisfy at least of the equalities. Hence $\rho$ can't be diverse.

For the other direction, assume that $C$ does not have any diverse models. This implies that for any model $\rho$ of $C$, there is at least a pair of variables $u$ and $v$ such that $\rho(u) = \rho(v)$. Thus the formula $C \wedge \bigwedge\{u \neq v \mid u \in vars(C), v \in vars(C), u \not\equiv v\}$ does not have any models or is unsatisfiable. Therefore $C$ implies $\bigvee\{u = v \mid u \in vars(C), v \in vars(C), u \not\equiv v\}$. Since none of the equalities in this set (in isolation) is implied by $C$, there has to be a minimal subset of equalities whose disjunction is implied by $C$.

The above theorem illustrates that the problem of checking if a set of UTVPI constraints imply a disjunction of equalities over the variables is NP-Complete.

## References

1. T. Ball, B. Cook, S. K. Lahiri, and L. Zhang. Zapato: Automatic Theorem Proving for Software Predicate Abstraction Refinement. In *Computer Aided Verification (CAV '04)*, LNCS 3114. Springer-Verlag, 2004.
2. T. Ball, R. Majumdar, T. Millstein, and S. K. Rajamani. Automatic predicate abstraction of C programs. In *Programming Language Design and Implementation (PLDI '01)*, Snowbird, Utah, June, 2001. *SIGPLAN Notices,* 36(5), May 2001.

3. R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.
4. B. V. Cherkassky and A. V. Goldberg. Negative-cycle detection algorithms. In *European Symposium on Algorithms*, pages 349–363, 1996.
5. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.
6. C. Flanagan, R. Joshi, X. Ou, and J. Saxe. Theorem Proving usign Lazy Proof Explication. In *Computer-Aided Verification (CAV 2003)*, LNCS 2725, pages 355–367. Springer-Verlag, 2003.
7. C. Flanagan, K. R. M. Leino, M. Lillibridge, G. Nelson, J. B. Saxe, and R. Stata. Extended static checking for java. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'02)*, pages 234–245, 2002.
8. L. R. Ford, Jr., and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.
9. W. Harvey and P. J. Stuckey. A unit two variable per inequality integer constraint solver for constraint logic programming. In *Proceedings of the 20th Australasian Computer Science Conference (ACSC '97)*, pages 102–111, 1997.
10. J. L. Hennessy and T. R. Gross. Postpass code optimization of pipeline constraints. *ACM Trans. Program. Lang. Syst.*, 5(3):422–448, 1983.
11. J. Jaffar, M. J. Maher, P. J. Stuckey, and H. C. Yap. Beyond Finite Domains. In *Proceedings of the Second International Workshop on Principles and Practice of Constraint Programming, PPCP'94.*
12. J. C. Lagarias. The computational complexity of simultaneous diophantine approximation problems. *SIAM Journal of Computing*, 14(1):196–209, 1985.
13. S. K. Lahiri and R. E. Bryant. Deductive verification of advanced out-of-order microprocessors. In *Computer-Aided Verification (CAV 2003)*, LNCS 2725, pages 341–354. Springer-Verlag, 2003.
14. S. K. Lahiri and M. Musuvathi. An efficient nelson-oppen decision procedure for difference constraints over rationals. Technical Report MSR-TR-2005-61, Microsoft Research, 2005.
15. A. Miné. The octagon abstract domain. In *AST 2001 in WCRE 2001*, IEEE, pages 310–319. IEEE CS Press, October 2001.
16. G. C. Necula and P. Lee. Proof generation in the touchstone theorem prover. In *Conference on Automated Deduction*, LNCS 1831, pages 25–44, 2000.
17. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 2(1):245–257, 1979.
18. V. Pratt. Two easy theories whose combination is hard. Technical report, Massachusetts Institute of Technology, Cambridge, Mass., September 1977.
19. S. A. Seshia and R. E. Bryant. Deciding quantifier-free presburger formulas using parameterized solution bounds. In *19th IEEE Symposium of Logic in Computer Science(LICS '04).* IEEE Computer Society, July 2004.
20. R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2):146–160, 1972.