Conflict Resolution

Konstantin Korovin, Nestan Tsiskaridze, and Andrei Voronkov

The University of Manchester
`{korovin|tsiskarn|voronkov}@cs.man.ac.uk`

Abstract. We introduce a new method for solving systems of linear inequalities over the rationals—the *conflict resolution method*. The method successively refines an initial assignment with the help of newly derived constraints until either the assignment becomes a solution of the system or a trivially unsatisfiable constraint is derived. We show that this method is correct and terminating. Our experimental results show that conflict resolution outperforms the Fourier-Motzkin method and the Chernikov algorithm, in some cases by orders of magnitude.

1 Introduction

In this paper we introduce a new algorithm for checking solvability of systems of linear inequalities, called *conflict resolution*. The method works with such a system S and an assignment to variables σ (initially arbitrary) and refines the assignment trying to make it into a solution of S. If such a refinement is impossible, it is due to a pair of inequalities in S of the forms $x + p \geq 0$ and $-x + q \geq 0$ with some properties. In this case we *resolve* the conflict by adding a new equation $p + q \geq 0$. The use of this rule makes the method similar to the Fourier-Motzkin variable elimination.

The first (rather naive) implementation of the method shows that in practice it normally behaves better, and sometimes orders of magnitude better than the Fourier-Motzkin method and the Chernikov algorithm. These experiments are confirmed by some properties of our method proved in this paper, namely, that it never derives redundant (in some natural sense) inequalities. The conflict resolution algorithm is well-suited both for proving inconsistency of systems of linear constraints and for finding satisfying assignments.

It is a bit too early to extensively compare this method to the existing implementations of the simplex or interior point methods since not much is known about the best strategies, optimisations and modifications of the method but we hope it can eventually become competitive also with the best previously known methods. In this paper we only present some initial experimental results.

This paper is structured as follows. Section 2 defines main notions. Section 3 overviews the Fourier-Motzkin variable elimination method. In Section 4 we introduce the conflict resolution and the assignment refinement rules used in our method and prove some of their properties. Section 5 presents the conflict resolution algorithm CRA. We prove that the algorithm is correct and terminating.

In Section 6 we compare our method with the Fourier-Motzkin variable elimination: we show that our method does not derive redundant inequalities and give an example of a sequence of systems on which the Fourier-Motzkin method is exponential while our method is linear independently of the strategy used and the order of rule applications. In Section 7 we briefly discuss how the method can be modified to handle strict inequalities and linear programming. Section 8 is dedicated to the implementation and Section 9 to experiments with our method. Finally, in Section 10 we mention related work on avoiding redundancy in the Fourier-Motzkin method.

2 Preliminaries

Let $\mathbb{Q}$ denote the set of rationals. Throughout the paper we denote by n a positive integer and by X a finite set of variables $\{x_1, \ldots, x_n\}$. We call a *rational linear constraint over* X either a formula $a_n x_n + \ldots + a_1 x_1 + b \diamond 0$, where $\diamond \in \{\geq, >, =, \neq\}$ and $a_i \in \mathbb{Q}$ for $1 \leq i \leq n$, or one of the formulas $\bot, \top$. The formula $\bot$ is always false and $\top$ is always true. The constraints $\bot$ and $\top$ are called *trivial*. For brevity, in the sequel we will call such rational linear constraints over X simply *linear constraints*. We call a *system of linear constraints* any finite set of linear constraints.

In this paper we will describe several algorithms for solving finite sets of rational linear constraints. Let $\succ$ be a total order on X. Without loss of generality we assume $x_n \succ x_{n-1} \succ \ldots \succ x_1$. A constraint is called *normalised* if it is of the form $\bot$, $\top$, $x_k + q \diamond 0$ or $-x_k + q \diamond 0$, where $\diamond$ is as defined above, x_k is the maximal variable in the respective constraint, and q does not contain x_k. Evidently, every constraint can be effectively transformed into an equivalent normalised constraint. In the sequel, we assume that all constraints are normalised.

We define an *assignment* σ over the set of variables X as a mapping from X to $\mathbb{Q}$, i.e. $\sigma : X \to \mathbb{Q}$. Given an assignment σ, a variable $x \in X$ and a value $v \in \mathbb{Q}$, we call the *update of* σ *at* x *by* v, denoted by σ_x^v, the assignment obtained from σ by changing the value of x by v and leaving the values of the other variables unchanged.

For a linear polynomial q over X, denote by $q\sigma$ the value of q after replacing all variables $x \in X$ by the corresponding values $\sigma(x)$. An assignment σ is called a *solution of a linear constraint* $q \diamond 0$ if $q\sigma \diamond 0$ is true; it is a *solution of a system* of linear constraints if it is a solution of every constraint in the system. If σ is a solution of a linear constraint c (or a system S of such constraints), we also say that σ *satisfies* c (respectively, S), denoted by $\sigma \models c$ (respectively, $\sigma \models S$), otherwise we say that σ *violates* c (respectively, S). A system of linear constraints is said to be *satisfiable* if it has a solution.

For simplicity, we consider only algorithms for solving systems of linear constraints of the form $q \geq 0$, $\bot$ and $\top$ and discuss the general case later.

3 Fourier-Motzkin elimination

In this section we briefly describe the Fourier-Motzkin elimination method. Consider a system S of linear constraints. The method either determines that S has no solution, or finds at least one. The method is based on an iterative algorithm changing S by eliminating a variable at each step. We assume that the variables are eliminated according to the order $\succ$, that is, x_n is eliminated first. At each step, if the maximal variable in the current system of linear constraints is x_k, we denote the current system by S_k, thus $S_n = S$. When the algorithm terminates, we obtain a system containing only trivial constraints, we denote this system by S_0.

Let $k > 0$. The system S_{k-1} is obtained from S_k by (i) adding new linear constraints as follows: for every pair of linear constraints $x_k+p \geq 0$ and $-x_k+q \geq 0$ in S_k we add to S_{k-1} a new constraint $p + q \geq 0$ and (ii) removing all linear constraints containing x_k.

One can show that the original system S is unsatisfiable if and only if S_0 contains $\bot$. If S_0 does not contain $\bot$, we can build a solution σ of S using the following observation. An assignment σ satisfies S_k if and only if σ satisfies S_{k-1} and

$$x_k\sigma \in [\max\{-p\sigma \mid (x_k + p \geq 0) \in S_k\}, \min\{q\sigma \mid (-x_k + q \geq 0) \in S_k\}] \,. \quad (1)$$

As usual, we assume that the minimum of the empty set is $+\infty$ and the maximum of it is $-\infty$. Condition (1) essentially says that the value of x_k lies in a certain interval determined by the values of variables $x_1, \ldots, x_{k-1}$. One can prove that this interval is non-empty whenever σ satisfies S_{k-1}. Thus, we can change any solution σ of S_{k-1} into a solution of S_k by updating σ at x_k by an arbitrary value in this interval. In this way we can build a solution to $S = S_n$ as follows. We start with an arbitrary assignment σ (which obviously satisfies S_0) and update it at $x_1, \ldots, x_n$ as described above. In fact, all solutions of the initial system can be derived this way.

Note that the Fourier-Motzkin algorithm applied to a set of linear constraints always terminates and generates only a finite number of linear constraints. However, the algorithm is in general exponential.[1] In general, the number of linear constraints in S_{k-1} is in the worst case quadratic in the number of constraints in S_k.

Unlike the Fourier-Motzkin method, our conflict resolution method does not eliminate variables. It uses the rule deriving $p + q \geq 0$ from $x_k + p \geq 0$ and $-x_k + q \geq 0$ but derives new constraints in a more restrictive way.

[1] Some papers claim it is double-exponential but we could not find any paper proving this. Schrijver [10] defines a sequence of systems of the size $O(n^3)$ on which the method generates $O(2^n)$ constraints. Some papers refer to [3] as giving an example of double-exponential behaviour but [3] only repeats the example from [10] verbatim.

4 Conflict Resolution

In this section we introduce our *conflict resolution method* for solving systems of linear rational constraints.

Let c be a linear constraint. If the maximal variable in c is x_k, then we say that k is the *level* of c. If c contains no variables, then we define the level of c to be 0. Note that, since all constraints are assumed to be normalised, a constraint written in the form $x_k + p \geq 0$ or $-x_k + q \geq 0$ is of the level k. The notion of level induces a partial order on linear constraints, which we will denote also by $\succ$, as follows. For two linear constraints c_1 and c_2, we have $c_1 \succ c_2$ if and only if the level of c_1 is strictly greater than the level of c_2.

We call a *state* a pair (S, σ), where S is a system of linear constraints and σ an assignment. Let $\mathbb{S} = (S, \sigma)$ be a state and k a positive integer. We say that $\mathbb{S}$ *contains a k-conflict* $(x_k + p \geq 0, -x_k + q \geq 0)$ if (i) both $x_k + p \geq 0$ and $-x_k + q \geq 0$ are linear constraints in S and (ii) $p\sigma + q\sigma < 0$. Instead of "k-conflict" we will sometimes simple say "conflict". Note that if σ is a solution of S, then $\mathbb{S}$ contains no conflicts.

We will now formulate our method. Given a system S of linear constraints, it starts with an initial state (S, σ), where σ is an arbitrary assignment and repeatedly transforms the current state either by either adding a new linear constraint to S or updating the assignment. We will formulate these rules below as transformation rules on states $\mathbb{S} \Rightarrow \mathbb{S}'$, meaning that $\mathbb{S}$ can be transformed into $\mathbb{S}'$. Let k be an integer such that $1 \leq k \leq n$.

The conflict resolution rule (CR) (at the level k) is the following rule:

$$(S, \sigma) \Rightarrow (S \cup \{p + q \geq 0\}, \sigma),$$

where (S, σ) contains a k-conflict $(x_k + p \geq 0, -x_k + q \geq 0)$.

The assignment refinement rule (AR) (at the level k) is the following rule:

$$(S, \sigma) \Rightarrow (S, \sigma^v_{x_k}),$$

where

1. σ satisfies all constraints in S of the levels $0, \dots, k-1$.
2. σ violates at least one constraint in S of the level k.
3. $\sigma^v_{x_k}$ satisfies all constraints in S of the level k.

We will call any instance of an inference rule an *inference*. Thus, our algorithm will perform CR-inferences and AR-inferences.

Note that the conflict resolution rule derives a linear constraint violated by σ:

Lemma 1. *Let (S, σ) contain a k-conflict $(x_k + p \geq 0, -x_k + q \geq 0)$. Then $\sigma \not\models p + q \geq 0$.* ❑

Let us introduce a new notation. For any system S of linear constraints, a non-negative integer k and an assignment σ denote

$$\begin{aligned} L(S,\sigma,k) &\overset{\text{def}}{=} \max\{-p\sigma \mid (x_k + p \geq 0) \in S\}; \\ U(S,\sigma,k) &\overset{\text{def}}{=} \min\{q\sigma \mid (-x_k + q \geq 0) \in S\}; \\ I(S,\sigma,k) &\overset{\text{def}}{=} [L(S,\sigma,k), U(S,\sigma,k)]. \end{aligned}$$

For every set S of linear constraints and a positive integer k, denote by $S_{=k}$ (respectively, $S_{<k}$) the subset of S consisting of all constraints of the level k (respectively, of all levels strictly less than k).

Lemma 2. *(i) Condition (2) of the assignment refinement rule implies $x_k\sigma \notin I(S,\sigma,k)$. (ii) Condition (3) of the assignment refinement rule is equivalent to $v \in I(S,\sigma,k)$. (iii) The interval $I(S,\sigma,k)$ is non-empty if and only if $\mathbb{S}$ contains no k-conflicts.*

Proof. (i) We assume that $x_k\sigma \in I(S,\sigma,k)$ and prove that σ satisfies $S_{=k}$. Take any constraint in $S_{=k}$. Without loss of generality assume that it has the form $x_k + p \geq 0$. Since $x_k\sigma \in I(S,\sigma,k)$, we have $x_k\sigma \geq L(S,\sigma,k)$, that is, $x_k\sigma \geq \max\{-p\sigma \mid (x_k + p \geq 0) \in S\}$. This implies $x_k\sigma \geq -p\sigma$, hence σ is a solution of $x_k + p \geq 0$.

(ii) In one direction, assume $v \in I(S,\sigma,k)$. Note that $x_k\sigma^v_{x_k} = v$, so $x_k\sigma^v_{x_k} \in I(S,\sigma,k)$ Using the same arguments as in (i) but with σ replaced by $\sigma^v_{x_k}$ we can prove $\sigma^v_{x_k} \models S_{=k}$. In the other direction, assume $\sigma^v_{x_k} \models S_{=k}$. We have to prove $v \in I(S,\sigma,k)$, that is, $v \geq L(S,\sigma,k)$ and $v \leq U(S,\sigma,k)$. We will only prove the former condition, the latter one is similar. The former condition means $v \geq \max\{-p\sigma \mid (x_k + p \geq 0) \in S\}$. To prove it, we have to show that for all constraints of the form $x_k + p \geq 0$ in S (and hence in $S_{=k}$) we have $v \geq -p\sigma$. Since p may only contain variables in $\{x_1,\ldots,x_{k-1}\}$ and σ agrees with $\sigma^v_{x_k}$ on all such variables, we have $-p\sigma = -p\sigma^v_{x_k}$, so $v \geq -p\sigma^v_{x_k}$. Using $x_k\sigma^v_{x_k} = v$, we obtain $x_k\sigma^v_{x_k} \geq -p\sigma^v_{x_k}$, hence $\sigma^v_{x_k}$ is a solution of $x_k + p \geq 0$, and we are done.

(iii) We will prove that $I(S,\sigma,k)$ is empty if and only if $\mathbb{S}$ contains a k-conflict. In one direction, assume $I(S,\sigma,k)$ is empty. Then $L(S,\sigma,k) > U(S,\sigma,k)$. Note that this implies that both $L(S,\sigma,k)$ and $U(S,\sigma,k)$ are finite. Since they are finite, $S_{=k}$ contains two constraints of the form $x_k + p \geq 0$ and $-x_k + q \geq 0$ such that $-p\sigma = L(S,\sigma,k)$ and $q\sigma = U(S,\sigma,k)$. This and $L(S,\sigma,k) > U(S,\sigma,k)$ implies $-p\sigma > q\sigma$, and so $0 > p\sigma + q\sigma$. Therefore, $(x_k + p \geq 0, -x_k + q \geq 0)$ is a k-conflict. The proof in other direction is similar. ❏

The following is a key lemma for our method.

Lemma 3. *Let (S,σ) be a state and $1 \leq k \leq n$. Let σ satisfy all constraints in S of the levels $0,\ldots,k-1$ and violate at least one constraint of the level k. If $I(S,\sigma,k)$ is empty, then the conflict resolution rule at the level k is applicable and the assignment refinement rule at this level is not applicable. If $I(S,\sigma,k)$ is non-empty, then the assignment refinement rule at the level k is applicable and the conflict resolution rule at this level is not applicable.*

Algorithm 1 The Conflict Resolution Algorithm CRA

Input: A set S of linear constraints.
Output: A solution of S or "unsatisfiable".

```
 1: if $\bot \in S$ then return "unsatisfiable"
 2: $\sigma :=$ arbitrary assignment;
 3: $k := 1$
 4: while $k \leq n$ do
 5:     if $\sigma \not\models S_{=k}$ then
 6:         while $(S, \sigma)$ contains a $k$-conflict $(x_k + p \geq 0, -x_k + q \geq 0)$ do
 7:             $S := S \cup \{p + q \geq 0\}$;                    ▷ application of CR
 8:             $k :=$ the level of $(p + q \geq 0)$;
 9:             if $k = 0$ then return "unsatisfiable"
10:         end while
11:         $\sigma := \sigma^v_{x_k}$, where $v$ is an arbitrary value in $I(S, \sigma, k)$   ▷ application of AR
12:     end if
13:     $k := k + 1$
14: end while
15: return $\sigma$
```

Proof. Suppose $I(S, \sigma, k)$ is empty. By Lemma 2 (iii) $\mathbb{S}$ contains a k-conflict, so the conflict resolution rule is applicable at the level k. Since $I(S, \sigma, k)$ is empty, by Lemma 2 (ii) condition (3) of the assignment refinement rule is violated, so the assignment refinement rule at this level is not applicable.

Suppose that $I(S, \sigma, k)$ is non-empty. Then by Lemma 2 (iii) $\mathbb{S}$ contains no conflict, so the conflict resolution rule at the level k is not applicable. Take an arbitrary value $v \in I(S, \sigma, k)$. By Lemma 2 (ii) condition (3) of the assignment refinement holds. Conditions (1) and (2) of this rule hold by the assumptions of this lemma, so the assignment refinement rule is applicable. ❏

5 The Conflict Resolution Algorithm

The *Conflict Resolution Algorithm CRA* is given as Algorithm 1.

Let us note that the algorithm is well-defined, that is, the interval $I(S, \sigma, k)$ at line 11 is non-empty. Indeed, the algorithm reaches this line if (S, σ) contains no conflict at the level k (by line 6). Then $I(S, \sigma, k)$ is non-empty by Lemma 2 (iii).

Example 1. This example illustrates the algorithm. Let S_0 be the following set of constraints.

$$\begin{array}{rcl}
x_4 - 2x_3 \phantom{{}- 3x_2} + x_1 + 5 \geq 0 & & (1) \\
-x_4 - x_3 - 3x_2 - 3x_1 + 1 \geq 0 & & (2) \\
-x_4 + 2x_3 + 2x_2 + x_1 + 6 \geq 0 & & (3) \\
-x_3 + x_2 - 2x_1 + 5 \geq 0 & & (4) \\
x_3 \phantom{{}+ x_2} + 3x_1 - 1 \geq 0 & & (5)
\end{array}$$

Assume that the initial assignment σ maps all variables to 0. The algorithm starts at the level 0. The sets $S_{=0}, S_{=1}, S_{=2}$ are empty, so the assignment σ trivially satisfies them. However, it violates constraint (5) and so violates $S_{=3}$. The interval $I(S, \sigma, 3)$ is $[1, 5]$. It is non-empty, so by Lemma 3 we can apply the assignment refinement rule at level 3 by updating σ at x_3 by any value in $[1, 5]$. Let us choose, for example, the value 4. Let σ_1 denote the newly obtained assignment $\{x_4 \mapsto 0, x_3 \mapsto 4, x_2 \mapsto 0, x_1 \mapsto 0\}$. Now we move to the next level 4. There is a 4-conflict between constraints (1) and (2) (line 6). We make a CR-inference between these two clauses deriving a new constraint

$$-x_3 - x_2 - \tfrac{2}{3}x_1 + 2 \geq 0 \qquad (6)$$

added to the set S at line 7. According to line 8 of the algorithm we set the level k to the level of the new constraint, that is, to 3. Now there are no more conflicts on level 3 and we have $I(S, \sigma, 3) = [1, 2]$. We should update the assignment at x_3 by an arbitrary value in this interval. Suppose, for example, that we have chosen 1 as the value for x_3 obtaining $\{x_4 \mapsto 0, x_3 \mapsto 1, x_2 \mapsto 0, x_1 \mapsto 0\}$ and increase k by 1 proceeding to level 4. At this moment all constraints at level 4 are satisfied and the algorithm terminates returning σ. ❏

Our aim is to prove that the algorithm is correct and terminating.

Theorem 1. *The conflict resolution algorithm CRA always terminates. Given an input set of constraints S_0, if CRA outputs "unsatisfiable", then S_0 is unsatisfiable. If CRA outputs an assignment σ, then σ is a solution of S_0.*

This theorem will be proved after a series of lemmas establishing properties of the algorithm. In these lemmas we always denote the input set of constraints by S_0.

Lemma 4. *At any step of the algorithm the set S is equivalent to S_0, that is, S and S_0 have the same set of solutions.*

Proof. Observe that line 7 is the only line that changes S. It is easy to see that the application of this line does not change the set of solutions of S since the constraint $p + q \geq 0$ added to S is implied by S. ❏

The following lemma is obvious.

Lemma 5. *Every constraint occurring in S at any step of the CRA algorithm belongs to the set of constraints derived by the Fourier-Motzkin algorithm applied to S_0.* ❏

Lemma 6. *The assignment σ at lines 4 and 6 satisfies $S_{<k}$.*

Proof. By induction on the number of iterations of the outermost while-loop. Before the first iteration the property is obvious since $k = 1$ and $\perp \notin S$. So we assume that the property holds before an iteration of the loop and show it holds after this iteration. If $\sigma \models S_{=k}$ at line 5, then by $\sigma \models S_{<k}$ we have $\sigma \models S_{<k+1}$. It

remains to consider the case when $\sigma \not\models S_{=k}$ at line 5. In this case the algorithm may enter the internal while-loop starting at line 6. It is easy to see that at the exit of this loop the property is satisfied as well, since k only decreases in the loop and the new constraint $p + q \geq 0$ is at the level k. So it remains to show that after line 11 we have $\sigma \models S_{=k}$. But this is guaranteed by Lemma 2 (ii), so we are done. ❑

Lemma 7. *Let* (S, σ) *contain a conflict* $(x_k + p \geq 0, -x_k + q \geq 0)$ *at line 6. Then we have* $(p + q \geq 0) \notin S$.

Proof. By Lemma 6 at line 6 we have $\sigma \models S_{<k}$. But we have $\sigma \not\models (p + q \geq 0)$, hence $(p + q \geq 0) \notin S_{<k}$. Since the level of $(p + q \geq 0)$ is strictly less than k this implies $(p + q \geq 0) \notin S$. ❑

This lemma means that the same constraint will never be added again to S. In fact, the algorithm has a much stronger property formulated below in Lemma 8.

Let us now give the proof of Theorem 1.

Proof. We start with proving termination. By Lemma 7 the algorithm never adds the same constraint twice. By Lemma 5 we can add only a finite number of different constraints. Therefore, condition on line 6 can hold only a finite number of times. From the moment this condition becomes permanently false, k will always increase by 1, so the outermost while-loop will terminate.

Suppose now that the algorithm returns “unsatisfiable”. If this happens at line 1, then $\perp \in S_0$, so S_0 is unsatisfiable. Otherwise, this happens at line 9. Then $\sigma \not\models p + q \geq 0$ by Lemma 1. Since $k = 0$, then the constraint $p + q \geq 0$ contains no variables, so this constraint is trivial and unsatisfiable. By Lemma 4, this constraint is implied by S_0, hence S_0 is unsatisfiable too.

It remains to consider the case when the algorithm return an assignment σ. This only can happen at the last line of the algorithm. At this line, $k = n + 1$. By Lemma 6, σ satisfies $S_{<n+1}$. Note that $S_{<n+1} = S$, so σ also satisfies S. By Lemma 4, S is equivalent to S_0, hence σ also satisfies S_0. ❑

6 Conflict Resolution and the Fourier-Motzkin Method

We say, that a CR-inference at a level k is *redundant* w.r.t. a state (S, σ) if the conclusion of this inference is a consequence of constraints in $S_{<k}$. Let us prove a key property that distinguishes our algorithm from the Fourier-Motzkin method.

Lemma 8. *Every CR-inference performed by the CRA algorithm is non-redundant.*

Proof. Suppose that the algorithm performs a redundant inference adding $p+q \geq 0$ at line 7. Then by the definition of redundancy $p+q \geq 0$ is implied by $S_{<k}$. By Lemma 6 we have $\sigma \models S_{<k}$, then σ must also satisfy $p + q \geq 0$. This contradicts to the definition of a conflict. ❑

To illustrate this lemma, let us come back to Example 1. Note that in this example we have not applied the conflict resolution inference between constraints (1) and (3). It is easy to see that the conclusion of this inference is implied by constraints (4) and (5) at smaller levels, therefore *this inference would not be applied independently of the choices of assignments made by the algorithm.*

Let us now show that the Fourier-Motzkin algorithm cannot polynomially simulate our algorithm in a very strong sense. This example is taken from [10]. It contains all inequalities of the form $\pm x_k \pm x_l \pm x_m \geq 0$, where $n \geq k > l > m \geq 1$. Evidently, the size of the system is $O(n^3)$ and there exists only a single solution assigning 0 to all variables. It is shown in [10] that the Fourier-Motzkin method generates exponentially many (in n) inequalities for this example. Let σ be an arbitrary assignment. Our method will start generating conflicts from level 3 containing 8 inequalities until it updates σ so that $x_1\sigma = x_2\sigma = x_3\sigma = 0$. After that it will proceed to level 4, where the interval $I(S, \sigma, 4)$ will consist of a single point 0. The assignment refinement will set $x_4\sigma$ to 0 and no conflicts will be generated. The same will happen with all levels greater than 4, so the algorithm will terminate in a linear number of steps. Essentially, apart from the initial work on level 3, the conflict resolution algorithm will only evaluate every inequality once and so work in time linear in the size of the system, that is $O(n^3)$. Note that this running time does not depend on either the choice of the initial assignment or the choice of values in the assignment refinement inferences.

7 Extensions

In this section we briefly mention two extensions of the method: one is for working with strict inequalities and another one for linear programming.

The modification of the algorithm for working with *strict inequalities* $p > 0$ is straightforward. First, when we consider the interval

$$I(S, \sigma, k) = [L(S, \sigma, k), U(S, \sigma, k)]$$

if any endpoint of this interval corresponds to a strict inequality, we use a semi-open or an open interval instead. For example, if there is a strict inequality $(x_k + p > 0) \in S$ such that $-p\sigma = L(S, \sigma, k)$ but no strict inequality $(-x_k + q > 0) \in S$ such that $q\sigma = U(S, \sigma, k)$, then we use the semi-open interval

$$I(S, \sigma, k) = (L(S, \sigma, k), U(S, \sigma, k)].$$

Second, the result of the conflict resolution rule is a strict inequality if at least one of the premises is strict. It is not hard to generalise our method to deal with disequalities $p \neq 0$ as well.

To use our algorithm for linear programming, we can use the following trick. Suppose, for example, that we want to find a maximum of a linear function p. To this end we assume that the constraint do not contain the variable x_1 and add the equality $p - x_1 = 0$. After that we use our algorithm with the only modification that we always select the maximal possible value for x_1 in the assignment refinement rule. A special care should be taken when we have no a priory upper bound on x_1. However, using the method for linear programming is beyond the scope of this paper.

8 Implementation

In this section we briefly describe details of our implementation of the conflict resolution algorithm. Our implementation works with linear constraints of the form $q \diamond 0$, for $\diamond \in \{\geq, >, =\}$. In order to compare conflict resolution with other methods for solving systems of linear constraints we also implemented the standard Fourier-Motzkin algorithm and the Chernikov algorithm [4] using the same data structures as used in the implementation of CRA.

Informally, the Chernikov algorithm extends the Fourier-Motzkin algorithm, (see Section 3) with the following restriction on added linear constraints. Let $S = S_n$ be the set of linear constraints. With each linear constraint we associate a set of initial constraints used in the derivation of this constraint, called the *index set*. Define the index set of an initial constraint $c \in S_n$ to be $\{c\}$. Let $k > 0$. The system S_{k-1} is obtained from S_k by removing all linear constraints containing x_k and adding new linear constraints as follows. For every pair of linear constraints $x_k + p \geq 0$ and $-x_k + q \geq 0$ in S_k, with index sets I, J respectively, we add to S_{k-1} a new constraint $p + q \geq 0$ with the index set $I \cup J$, if the following conditions (i-ii) hold. Let l be the level of $p + q \geq 0$ (see Section 4), then (i) the cardinality of $I \cup J$ is less than or equal to $n - l + 1$, and (ii) there is no constraint c in S_{k-1} of the level l with the index set $U \subseteq (I \cup J)$. It is shown in [4] that the original system S is unsatisfiable if and only if S_0 contains $\perp$.

Our implementation of conflict resolution follows Algorithm 1. There are a number of key parameters that can be used to fine-tune the CRA algorithm, namely

1. strategies for selecting conflicts,
2. strategies for selecting values in the assignment refinement rule,
3. order on variables.

Let us briefly describe possible choices for these parameters in our current implementation.

The strategy for selecting conflicts in the current implementation is based on *maximal overlaps*, as described below. At the line 7 of Algorithm 1 we select a k-conflict $x_k + p \geq 0$ and $-x_k + q \geq 0$ in S (i.e. $p\sigma + q\sigma < 0$), such that $-p\sigma = L(S, \sigma, k)$ and $q\sigma = U(S, \sigma, k)$. To explain the rationale behind this strategy let us extend our notion of redundancy to constraints. We call a constraint c at a level k *redundant* if this constraint is implied by $S_{<k+1} - \{c\}$. One can modify our algorithm and show that any redundant constraint can be removed[2].

[2] We cannot remove redundant constraints simultaneously since removal of a redundant constraint can make another previously redundant constraint non-redundant.

It is not hard to prove that constraints $x_k + p \geq 0$ such that $-p\sigma = L(S, \sigma, k)$ are "almost" non-redundant in the following sense.

Lemma 9. *Consider the set S^+ of all constraints at a level k having the form $x_k + p \geq 0$. Consider its subset S' consisting of all constraints $x_k + p \geq 0$ such that $-p\sigma = L(S, \sigma, k)$. Then S' is not implied by $S_{<k} \cup (S^+ - S')$.* ❑

One can formulate a symmetric property for constraints $-x_k + q \geq 0$ such that $q\sigma = U(S, \sigma, k)$.

Although our algorithm does not perform redundant inferences, the system may contain redundant constraints at a level k for two reasons: (i) it may contain redundant constraints initially; and (ii) addition of new constraints to a level k may make other constraints at this and higher levels redundant. Choosing a k-conflict $x_k + p \geq 0$ and $-x_k + q \geq 0$ in S (i.e. $p\sigma + q\sigma < 0$), such that $-p\sigma = L(S, \sigma, k)$ and $q\sigma = U(S, \sigma, k)$ does not, in general, guarantee, that the constraints forming the conflict are non-redundant but it guarantees that they are "almost" non-redundant in the sense of Lemma 9.

We tried several strategies for selecting values in the assignment refinement rule. One of the strategies is just selection of the middle point in the interval $I(S, \sigma, k)$ (line 11). Our experiments show that using this strategy frequently results in a rapid growth of the sizes of numerators and denominators of rational values in the assignment. In order to avoid this problem we used the following strategy for selecting the update values from the interval $I(S, \sigma, k)$. First, if the endpoints of $I(S, \sigma, k)$ coincide, then we select one of them. Otherwise, we select a rational number n/m in $I(S, \sigma, k)$ such that (i) m is the least power of 2 among denominators of all rationals in $I(S, \sigma, k)$, and (ii) n is such that, n/m is the closest rational to the middle point of the interval, among all rationals satisfying (i). It is possible to show that a rational satisfying both (i) and (ii) always exists. In particular, if $I(S, \sigma, k)$ contains integer points, then our strategy will select an integer in $I(S, \sigma, k)$ closest to the middle point. As our experiments show, such choice of values considerably simplifies the assignment values and constraint evaluation.

The last parameter of the CRA algorithm we consider here is the order on variables. In the current implementation the order on variables is selected randomly before the run of the CRA algorithm. We believe that a careful selection of the order on variables based on the properties of the input problem can considerably improve the performance of our implementation and we will make experiments with the order selection in the nearest future.

The CRA algorithm is implemented in C++ using the GMP library for arbitrary precision arithmetic [3]. Thus, all computations with rational numbers are done with arbitrary precision. We implemented two different representations of constraints, one using arrays of the size n to store vectors and one storing only non-zero coefficients. Not surprisingly, on randomly generated problems the first implementation is slightly better in both time and space while the second one

[3] http://gmplib.org/

4000 problems vars 3-12 (unsat/sat)				
	CRA	CVC3	FM	Ch
timeout (20s)	0/0	11/9	790/329	149/10
av. time (s)	0/0	0/0	0.4/0.1	0.6/0.1

400 problems vars 13-22 (unsat/sat)				
	CRA	CVC3	FM	Ch
timeout (20s)	5/2	21/33	183/144	155/65
av. time (s)	0.2/0.3	0/0	0.1/0.5	1.9/0.6

Table 1. Randomly Generated Problems

can be much faster (and consume much less space) on non-random problems, where vectors are normally sparse.

Finally, let us note that in the context of satisfiability modulo theories (SMT), it is desirable for solvers to be incremental and be able to generate explanations for the unsatisfiability. The CRA algorithm and our implementation can easily be made incremental: after adding/removing constraints we can always continue with the current assignment, moreover the CRA never performs redundant inferences and in particular, never performs the same inference twice (unless the conclusion was removed). Explanations can be generated from the proofs of unsatisfiability which are easily extractable from runs of the CRA algorithm.

9 Experimental Results

In this section we experimentally evaluate our implementation of the conflict resolution algorithm, the Fourier-Motzkin algorithm and the Chernikov algorithm. We implemented the algorithm in C++ using the GNU Multiple Precision Arithmetic Library (GMP) for handling arbitrary-precision rationals.

We compare our implementation with CVC3 [1] and Barcelogic [?], which are well-developed solvers for satisfiability modulo theories (SMT). CVC3 incorporates a variant of the Fourier-Motzkin algorithm and Barcelogic incorporates the simplex algorithm for reasoning with linear arithmetic. Let us note that our implementation is at a very early stage, no preprocessing was used and crucial for efficiency heuristics such as selection of suitable variable order are yet to be implemented. Already for this implementation, our experimental results are very encouraging, showing that the conflict resolution algorithm is considerably more efficient in solving linear constraints than the standard Fourier-Motzkin algorithm. For example, an order of magnitude difference occurs already on small problems.

We evaluated the solvers on two sets of benchmarks[4]. The first set of benchmarks consists of randomly generated systems of linear constraints. The second set of benchmarks consists of systems of linear constraints extracted from real-life SMT problems [2], using our tool called Hard Reality (HRT) [9]. All experiments were run on a Linux laptop with CPU 2.8GHz and memory 4Gb.

Results for randomly generated problems are shown in Table 1. The conflict resolution algorithm can solve all 4000 randomly generated problems with the

[4] http://www.cs.man.ac.uk/˜korovink/cra_bench

304 problems (unsat)				
	CRA	CVC3	FM	Ch
timeout (60s)	1	4	44	42
av. time	0.2	0.13	0.1	0.12

Table 2. Hard Reality Problems

$$\begin{array}{rrrrrr}
2x_5 - 3x_4 + & x_3 - 3x_2 - & 2x_1 + 3 \geq 0 \\
2x_5 + x_4 - & 2x_3 - & 2x_1 + 2 \geq 0 \\
-x_5 & + 3x_2 + & x_1 + 2 \geq 0 \\
-3x_5 + & 2x_3 - & 3x_1 - 2 \geq 0 \\
x_5 - 2x_4 & - 2x_2 + & 3x_1 - 2 \geq 0 \\
-2x_5 + 2x_4 - & 3x_3 - x_2 + & 2x_1 + 3 > 0 \\
3x_5 - 2x_4 + & 2x_3 + 3x_2 + & 2x_1 + 1 > 0 \\
x_5 & + & 2x_1 + 2 > 0 \\
2x_4 - & x_3 - 3x_2 - & x_1 + 3 = 0
\end{array}$$

Fig. 1. A randomly generated problem

number of variables ranging from 3 to 12 (within the total time of 7 seconds) and on the problems with the number of variables ranging from 13 to 22 fails only on 7. The CVC3 implementation of the Fourier-Motzkin algorithm fails to solve 20 problems and 54 problems respectively. Our implementation of the Fourier-Motzkin algorithm solves considerably fewer problems than CRA. The Chernikov algorithm improves over the Fourier-Motzkin but solves considerably fewer problems than CRA.

Table 2 compares the solvers on the problems extracted from SMT benchmarks using the Hard Reality Tool. These problems have different structure than the randomly generated problems, in particular the number of variables and constraints are considerably higher, most of problems contain several hundred of different variables and constraints.

The CRA also solves more problems in the Hard Reality benchmarks than any of CVC3, Fourier-Motzkin, and Chernikov algorithms. The average time of the CRA is a bit higher than of CVC3 due to extra solved problems. Indeed, in a pairwise comparison on all solved problems in these benchmarks the CRA is faster than CVC3.

One of the most striking examples showing the difference in performance is shown in Figure 1. The problem on this figure which was randomly generated and contains 5 variables and 10 linear constraints.

The standard Fourier-Motzkin algorithm run on this problem generated over 280 million linear constraints, while the conflict resolution algorithm generated only 21 constraints. However, this example is not outstanding as compared to our other experiments described above.

400 problems vars 13-22 (unsat/sat)				
	faster	same	av. time	timeout (20s)
Barcelogic	28/29	146/167	0.04/0	0/0
CRA	23/7	146/167	0.2/0.3	5/2
400 problems vars 23-32 (unsat/sat)				
Barcelogic	110/67	31/88	0.25/1.0	0/0
CRA	63/41	31/88	0.7/1.6	60/37

Table 3. CRA vs Barcelogic

Compared to the simplex algorithm, the conflict resolution shows promising potential. In Table **??** the CRA is compared to Barcelogic. Already our non-optimized implementation is faster than Barcelogic on a number of problems, although Barcelogic can solve more problems than CRA within 20 seconds.

To summarise, our experiments show that a naive implementation of the conflict resolution algorithm outperforms the Fourier-Motzkin and Chernikov algorithms in solving systems of linear constraints and has promising potential compared to the simplex algorithm. For the future work we are planning to extend the CRA algorithm with various heuristics for choosing conflicts, order on variables, values in the assignment update rule and methods for avoiding unnecessary re-evaluation of constraints.

10 Related Work

In this section we compare various modifications of the Fourier-Motzkin method with the conflict resolution method.

Most of modifications of the Fourier-Motzkin method aim at identifying potentially redundant constraints by providing some easy-to-check sufficient conditions for redundancy. One of the most prominent methods for restricting generation of redundant constraints was suggested by Chernikov [4]. His idea is to associate with each constraint some bookkeeping information on how this constraint was derived. Under certain conditions a newly derived constraint can be shown to be redundant based on this information (see Section 8). There are a number of extensions and modifications of this and other ideas developed over the past decades (e.g., [5, 8, 6, 7]). Our notion of redundancy seems to be orthogonal to that of Chernikov and the others, in particular it is based on the ordering on constraints and semantic entailment from the smaller constraints. One of important properties of the conflict resolution algorithm is that it never performs redundant inferences as defined in this paper. As a future work, we will investigate whether it is possible to combine our notion of redundancy with restrictions used by other methods.

11 Conclusions

We presented a new algorithm for solving systems of linear constraints, called conflict resolution. The method successively refines an initial assignment with the help of newly derived constraints until either the assignment becomes a solution of the system or the inconsistency of the initial system is proved. We have shown that this method is correct and terminating. The conflict resolution method has a number of attractive properties such as blocking of redundant inferences. We implemented our method and experimental results show that on the majority of problems we tried conflict resolution considerably outperforms well-developed methods such Fourier-Motzkin and Chernikov algorithms. We are currently working on improving our implementation and integration of crucial for efficiency heuristics such as various strategies for conflict selection, assignment refinement and variable order.

References

1. C. Barrett and C. Cesare Tinelli. CVC3. In W. Damm and H. Hermanns, editors, *CAV '07*, volume 4590 of *LNCS*, pages 298–302. Springer-Verlag, 2007. Berlin, Germany.
2. C. Barrett, S. Ranise, A. Stump, and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2008.
3. V. Chandru. Variable elimination in linear constraints. *Comput. J*, 36(5):463–472, 1993.
4. S.N. Chernikov. *Linejnye Neravenstva.* Nauka, Moscow, 1968. (In Russian).
5. R.J. Duffin. On Fourier's analyse of linear inequality systems. *Mathematical Programming Study*, 1:71–95, 1974.
6. JL. Imbert and P. Van Hentenryck. A note on redundant linear constraints. Technical Report CS-92-11, CS Department, Brown University, 1992.
7. Joxan Jaffar, Michael J. Maher, Peter Stuckey Roland, and Roland H. C. Yap. Projecting CLP($\mathcal{R}$) constraints. *New Generation Computing*, 11, 1993.
8. D.A. Kohler. *Projection of Convex Polyhedral Sets.* PhD thesis, University of California, Barkaley, 1967.
9. K. Korovin and A. Voronkov. Hard Reality Tool. Submitted, available at http://www.cs.man.ac.uk/~korovink/hr, 2009.
10. R. Nieuwenhuis and A. Oliveras. Decision Procedures for SAT, SAT Modulo Theories and Beyond. The BarcelogicTools. (Invited Paper). In G. Sutcliffe and A. Voronkov, editors, *12h International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR'05*, volume 3835 of *Lecture Notes in Computer Science*, pages 23–46. Springer, 2005.
11. A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, 1998.