# An Online Proof-Producing Decision Procedure for Mixed-Integer Linear Arithmetic[⋆]

Sergey Berezin, Vijay Ganesh, and David L. Dill

Stanford University
{berezin,vganesh,dill}@stanford.edu

**Abstract.** Efficient decision procedures for arithmetic play a very important role in formal verification. In practical examples, however, arithmetic constraints are often mixed with constraints from other theories like the theory of arrays, Boolean satisfiability (SAT), bit-vectors, etc. Therefore, decision procedures for arithmetic are especially useful in combination with other decision procedures. The framework for such a combination is implemented at Stanford in the tool called Cooperating Validity Checker (CVC) [SBD02].

This work augments CVC with a decision procedure for the theory of mixed integer linear arithmetic based on the Omega-test [Pug91] extended to be *online* and *proof producing*. These extensions are the most important and challenging part of the work, and are necessary to make the combination efficient in practice.

## 1 Introduction

Formal verification methods benefit greatly from efficient automatic decision procedures. There has been ample research in developing efficient decision procedures for various satisfiability problems like *Boolean satisfiability* (SAT [MMZ$^+$01,MSS99]), *bit-vectors* [Möl98], *linear integer and real arithmetic*, etc.

In practical examples, constraints from different theories are often mixed together. For example, it is not uncommon to see a constraint like the following:

$$2 * y + x > z \wedge f(x) \neq z \rightarrow A[2 * y + x] > 0,$$

which belongs to the combined theory of arithmetic, arrays, and uninterpreted functions. Consequently, there is a need for a decision procedure for the combination of theories.

Several combination methods like Nelson-Oppen [NO79], Shostak [Sho84], and their variants [RS01,BDS02a] have been developed. All these methods impose certain requirements on the individual decision procedures in order to achieve a sound, complete, and efficient combination. Satisfying these requirements greatly improves the usability of the individual decision procedures.

[⋆] This research was supported by GSRC contract SA3276JB, NSF grant CCR-9806889-002, and NSF ITR grant CCR-0121403. The content of this paper does not necessarily reflect the position or the policy of any of the funding agencies or the Government, and no official endorsement should be inferred.

One of the tools that combine decision procedures is the *Cooperating Validity Checker* (CVC [SBD02]) developed at Stanford. It is based on the framework of cooperating decision procedures developed by Barrett [BDS02a] which, in turn, is based on Nelson-Oppen [NO79] and Shostak [Sho84] frameworks. Each decision procedure in the framework is responsible for solving the satisfiability problem for only one particular theory and does not interact directly with the other theories.

The current implementation involves decision procedures for the theory of uninterpreted functions, the theory of arrays [SBDL01], the theory of datatypes, and the theory of linear arithmetic over integers and reals, which is the subject of this paper. Additionally, Boolean combinations of constraints are handled on the top level by the SAT solver [BDS02b] based on Chaff [MMZ$^+$01]. Thus, CVC as a whole serves as a decision procedure for the quantifier-free first-order theory of equality with arrays, recursive datatypes, and linear arithmetic [BDS00].

As with all combination methods, CVC imposes certain requirements on the individual decision procedures. In particular, each decision procedure must be *online* and *proof producing*. Online means that a new constraint can be added to the set of existing constraints at any time, and the algorithm must be able to take it into account with only incremental amount of work. When the set of constraints is determined to be unsatisfiable, the algorithm must also produce a proof of this fact.

Additionally, the theory of linear arithmetic is extended with the predicate $\mathsf{int}(t)$, which evaluates to $\mathsf{true}$ on a real-valued arithmetic term $t$, when $t$ evaluates to an integer. The decision procedure must be able to handle constraints of the form $\mathsf{int}(t)$ and $\neg\mathsf{int}(t)$ in addition to the usual linear arithmetic constraints.

The reasons for the above requirements can be better understood from the architecture of CVC. At a very high level, CVC can be viewed as a SAT solver [DP60] which solves the satisfiability problem of the Boolean skeleton of the first-order formulas. Each time the SAT solver makes a decision, a new constraint is submitted to the appropriate decision procedure. Since decisions are dynamic, decision procedures must be able to receive a new constraint and process it efficiently (the *online* requirement).

Modern SAT solvers [SS96,MSS99,MMZ$^+$01] use *conflict clauses* and *intelligent backtracking* to enhance performance. Backtracking implies that the solver may retract some constraints dynamically, hence, the decision procedure must support this operation. From an implementation point of view, this means that all the internal data structures in the decision procedure must be backtrackable.

To construct a conflict clause, one needs to identify those decisions made by the SAT solver which lead to a contradiction. CVC identifies such decisions by extracting them from the proof that the decision procedure constructs when it derives a contradiction. This explains the need for proof production in the decision procedures in CVC. As a bonus, the proofs can be checked by an external proof checker [Stu02] to increase the confidence in the results produced by CVC.

In addition to CVC, a few other tools combine online and proof producing decision procedures. Perhaps, the most similar to CVC is the Touchstone tool [NL00] developed at Berkeley. In particular, it has a simplex-based arithmetic desicion procedure, but only for real arithmetic.

This paper describes a decision procedure for mixed-integer linear arithmetic designed to meet the above requirements, and thus, fit the CVC framework. The decision procedure is based on an existing algorithm called *Omega-test* [Pug91], which is extended to be online, proof producing, and handle the $\mathsf{int}()$ predicate. Additionally, this implementation supports arbitrary precision arithmetic based on the GMP library [GMP]. The arbitrary precision arithmetic is crucial for solving sizable systems of mixed-integer constraints using Fourier-Motzkin approach, since variable elimination may produce large coefficients even if the coefficients in the original input are relatively small.

The choice of Omega-test over other algorithms for solving mixed-integer linear arithmetic problems (simplex, interior point method [BT97], earlier versions of Fourier-Motzkin elimination [Wil76], etc.) is driven by its simplicity and practical efficiency for a large class of verification problems. In particular, proof production is relatively easy to implement for the Omega-test.

The rest of the paper is organized as follows. Sections 2 and 3 review the original Fourier-Motzkin elimination method for real variables and its extension to integers (Omega-test), respectively. Both versions are then redesigned to make the algorithm online as described in sections 4 and 5. Section 4 also gives a brief overview of CVC. Proof production is described in section 6, and we conclude in section 7.

## 2 Fourier-Motzkin Elimination for Inequalities over Real Variables

**The problem.** Given a system of linear inequality constraints over real-valued variables of the form

$$\sum_{i=1}^{n} a_i x_i + c < 0,$$

where $a_i$'s and $c$ are rational constants, determine if this system is satisfiable. We only consider strict inequalities, since $\alpha \leq 0$ can be handled similarly to $\alpha < 0$. For simplicity, we assume that $\alpha \leq 0$ can be expanded into $\alpha < 0 \vee \alpha = 0$ and solved for each case in the disjunction. Here and in the rest of the paper, linear arithmetic terms are often denoted as $\alpha$, $\beta$, $\gamma$, or $t$, possibly with subscripts. In this section, we do not consider equalities, since any equality can be solved for some variable and instantiated into the other constraints, thus obtaining an equivalent system without the equality.

**Terminology.** For the sake of terminological clarity, we say that a variable is ***eliminated*** if an equality constraint is solved for this variable and the result is substituted in the remaining constraints. When the Fourier-Motzkin reasoning on inequalities is applied to a variable, we say that such a variable is ***projected***.

Throughout the paper we assume that all the constants and coefficients are *rational*. Although we often refer to variables as real-valued, it is well-known that, under the above conditions, the system of linear constraints is satisfiable in reals if and only if it is satisfiable in rationals.

Intuitively, Fourier-Motzkin elimination procedure [DE73] iteratively projects one variable $x$ by rewriting the system of inequalities into a new system without $x$ which has a solution if and only if the original system has a solution (i.e. the two systems are *equisatisfiable*). This process repeats until no variables are left, at which point all of the constraints become inequalities over numerical constants and can be directly checked for satisfiability.

More formally, the projection procedure is the following. First, pick a variable present in at least one inequality, say $x_n$. All the inequalities containing this variable are then rewritten in the form $\beta < x_n$ or $x_n < \alpha$, depending on the sign of the coefficient $a_n$, where $x_n$ does not occur in $\alpha$ or $\beta$. This creates three types of constraints: those with $x$ on the right, with $x$ on the left, and those without $x$:

$$\left\{\begin{array}{c}\beta_1 < x_n\\ \vdots\\ \beta_{k_1} < x_n\end{array}\right. \qquad \left\{\begin{array}{c}x_n < \alpha_1\\ \vdots\\ x_n < \alpha_{k_2}\end{array}\right. \qquad \left\{\begin{array}{c}\gamma_1 < 0\\ \vdots\\ \gamma_{k_3} < 0.\end{array}\right. \tag{1}$$

If this system of constraints has a solution, then $x_n$ must satisfy

$$\max(\beta_1, \ldots, \beta_{k_1}) < x_n < \min(\alpha_1, \ldots, \alpha_{k_2}).$$

Since real numbers are dense, such $x_n$ exists if and only if the following constraint holds:

$$\max(\beta_1, \ldots, \beta_{k_1}) < \min(\alpha_1, \ldots, \alpha_{k_2}).$$

This constraint can be equivalently rewritten as

$$\beta_i < \alpha_j \quad \text{for all } i = 1 \ldots k_1,\ j = 1 \ldots k_2, \tag{2}$$

which is again a system of linear inequalities. We call them the *shadow constraints*, because they define an $n-1$-dimensional shadow of the $n$-dimensional shape defined by the original constraints (1). The shadow constraints (2) combined together with $\gamma_l < 0$ comprise a new system of constraints which is equisatisfiable with (1), but does not contain the variable $x_n$. This process can now be repeated for $x_{n-1}$, and so on, until all the variables are projected.

Observe that for a system of $m$ constraints each elimination step may produce a new system with up to $(m/2)^2$ constraints. Therefore, eliminating $n$ variables may, in the worst case, create a system of $4 \cdot (m/4)^{2^n}$ constraints. Thus, the decision procedure for linear inequalities based on Fourier-Motzkin even in the case of real variables has a doubly exponential worst case complexity in the number of variables.

## 3 Extension of Fourier-Motzkin Elimination to Integer Variables (Omega-Test)

Our version of the extension is largely based on the Omega approach [Pug91] with a few differences. First, we consider the system of *mixed integer linear constraints* which, in addition to linear equalities and (strict) inequalities may also contain $\mathsf{int}(t)$ or

$\neg\mathsf{int}(t)$ for any linear term $t$, meaning that the linear term $t$ is restricted to only integer (respectively, fractional) values.

If the term $t$ is not a variable, the constraint $\mathsf{int}(t)$ is satisfiable iff $\mathsf{int}(z) \wedge z = t$ is satisfiable, where $z$ is a new variable. Furthermore, $\neg\mathsf{int}(t)$ is satisfiable for any term $t$ iff $\mathsf{int}(y) \wedge y < t < y + 1$ is satisfiable for a new variable $y$. Hence, any system of mixed integer linear constraints may be converted to an equisatisfiable system of constraints with only equalities, inequalities, and predicates of the form $\mathsf{int}(x)$, where $x$ is a variable.

### 3.1 Elimination of Equalities

As in the case of reals, all the equalities are eliminated first. If an equality contains a variable $x$ that is not an integer, then we solve the equality for this variable and eliminate $x$ from the system. Since this is the most efficient way of reducing the dimensionality of the problem, all such equalities are eliminated first.

Now suppose that an equality contains only integer variables:

$$\sum_{i=1}^{n} a_i x_i + c = 0. \tag{3}$$

Here we use the same variable elimination algorithm as in the Omega-test [Pug91]. If $x$ is the only variable in (3), then there is only one value of $x$ which can satisfy this equality constraint, namely $x = -(c/a)$. If this value is integral, we substitute it for $x$, and otherwise the system is unsatisfiable.

If there is more than one variable in (3), the equality is normalized such that all the coefficients $a_i$ and the free constant $c$ are relatively prime integers. It can always be done when the coefficients are rational numbers. If, after the normalization, there is a variable $x_k$ whose coefficient is $|a_k| = 1$, then we simply solve for $x_k$ and eliminate it from the rest of the system. Otherwise pick a variable $x_k$ whose coefficient $a_k$ is the smallest by the absolute value and define $m = |a_k| + 1$. Define also a modulus operation with the range $\left[-\frac{m}{2}, \frac{m}{2}\right)$ as follows:

$$a \,\mathbf{mod}\, m = a - m \left\lfloor \frac{a}{m} + \frac{1}{2} \right\rfloor .$$

The important properties of $\mathbf{mod}$ are that $a_k \,\mathbf{mod}\, m = -\mathrm{sign}(a_k)$, and that it distributes over addition and multiplication, where $\mathrm{sign}(x)$ is $-1$, $0$, or $1$ when $x < 0$, $x = 0$ and $x > 0$ respectively.

The next step is to choose a new variable $\sigma$ and introduce two new constraints into the system:

$$\mathsf{int}(\sigma) \quad \text{and} \quad \sum_{i=1}^{n} (a_i \,\mathbf{mod}\, m) x_i + (c \,\mathbf{mod}\, m) = m\sigma. \tag{4}$$

The second constraint is derivable from (3) by applying $\cdot \,\mathbf{mod}\, m$ on both sides of (3) and propagating $\mathbf{mod}$ over addition and multiplication to the coefficients. Hence, the system remains equisatisfiable with the original.

Since $a_k \,\mathbf{mod}\, m = -\mathrm{sign}(a_k)$, the equation (4) can be solved for $x_k$ and $x_k$ is eliminated from the system:

$$x_k = -\mathrm{sign}(a_k)m\sigma + \sum_{i\in[1..n]-\{k\}} \mathrm{sign}(a_k)(a_i \,\mathbf{mod}\, m)x_i + \mathrm{sign}(a_k)(c \,\mathbf{mod}\, m). \quad (5)$$

Substituting the result into the original equation (3) and using the facts $|a_k| = m - 1$ and $a - (a \,\mathbf{mod}\, m) = m \left\lfloor \frac{a}{m} + \frac{1}{2} \right\rfloor$ (from the definition of $m$ and $\mathbf{mod}$) we obtain:

$$-|a_k|\sigma + \sum_{i\in[1..n]-\{k\}} a'_i x_i + c' = 0, \quad (6)$$

where $a'_i = \left\lfloor \frac{a_i}{m} + \frac{1}{2} \right\rfloor + (a_i \,\mathbf{mod}\, m)$ and $c' = \left\lfloor \frac{c}{m} + \frac{1}{2} \right\rfloor + (c \,\mathbf{mod}\, m)$. The new system (which is the original system with $x_k$ eliminated using (5), and (3) rewritten as (6) ) contains the same number of variables as the original one. Moreover, the new coefficients $a'$ in (6) are guaranteed to decrease by absolute value compared to (3), namely $|a'_i| \le \frac{2}{3}|a_i|$, except for the coefficient of $\sigma$ which remains as large as that of $x_k$. This ensures that repeating the process above will eventually result in equality (6) having some variable with a coefficient 1 or $-1$. That variable can then be eliminated, reducing the overall dimensionality.

### 3.2 Projecting Variables from Inequalities

After eliminating all of the equalities, we are left with the system of (strict) inequalities over real and integer variables. Similar to the equality case, all the remaining real variables are projected first with the standard Fourier-Motzkin elimination procedure, resulting in a system of inequalities with only integer variables.

At this point, all the inequalities are normalized to make the coefficients be relatively prime integers, and a variable $x_n$ is chosen for projection. Since $x_n$ has an additional integral constraint, we cannot simply divide the inequality by the coefficient $a_n$ unless it is 1 or $-1$, and in general, the system of inequalities is rewritten in the equivalent form, very much like in (1), only the coefficients of $x_n$ are preserved:

$$\left\{\begin{array}{c} \beta_1 < b_n^1 x_n \\ \vdots \\ \beta_{k_1} < b_n^{k_1} x_n \end{array}\right. \qquad \left\{\begin{array}{c} a_n^1 x_n < \alpha_1 \\ \vdots \\ a_n^{k_2} x_n < \alpha_{k_2} \end{array}\right. \qquad \left\{\begin{array}{c} \gamma_1 < 0 \\ \vdots \\ \gamma_{k_3} < 0, \end{array}\right. \quad (7)$$

where the coefficients $a_i^j$ and $b_i^j$ are positive integers. Similar to the original Fourier-Motzkin construction, for each pair of inequalities $\beta < bx_n$ and $ax_n < \alpha$, which is equivalent to

$$a\beta < abx_n < b\alpha, \quad (8)$$

the *real shadow constraint* is constructed:

$$a\beta < b\alpha. \quad (9)$$

However, the real shadow is a necessary but not a sufficient condition for the satisfiability of (8), since there might not be an integer value $abx_n$ between $a\beta$ and $b\alpha$, even if there is a real one. In addition to the real shadow, at least one point $ab \cdot i$ must exist between $a\beta$ and $b\alpha$ for some integer $i$. A sufficient (but not necessary) condition is to demand that the gap between $a\beta$ and $b\alpha$ be at least $ab + 1$ wide:

$$\mathbf{D} \equiv b\alpha - a\beta > ab. \tag{10}$$

This constraint is called the *dark shadow constraint* (the object is "thick enough" to contain an integer point, and therefore casts a darker shadow; the term *dark shadow* is from [Pug91]). The dark shadow constraint is sufficient, but not necessary for an integer solution of $x_n$ to exist. Therefore, if equation (10) makes the system unsatisfiable, we have to look for an integer solution outside of $\mathbf{D}$, i.e. in the *gray shadow*: $b\alpha \leq a\beta + ab$. Following the construction in the Omega-test [Pug91], $b\alpha$ on the right-hand side of (8) is replaced by the larger $a\beta + ab$, and dividing the result by $a$ yields the following:

$$\beta < bx_n < \beta + b.$$

This means that if there is an integer solution to $x_n$, it must satisfy $bx_n = \beta + i$ for some $0 < i < b$, since $\beta$ contains only integer variables with integer coefficients. We then try each such $i$ in succession until a solution is found. In other words, the *gray shadow constraint* is:

$$\mathbf{G} \equiv \bigvee_{i=1}^{b-1} bx_n = \beta + i.$$

This is, obviously, the most expensive step of the algorithm, since it involves a lot of backtracking, but according to [Pug91], the dark shadow constraint almost always suffices in practice, and the gray shadow is often empty. Therefore, as a practical heuristic, the dark shadow constraint $\mathbf{D}$ is always tried first, and only if it fails, then a solution is searched for in the gray shadow $\mathbf{G}$.

## 4 Online Version of Fourier-Motzkin for Reals

In CVC, decision procedures are most effective when they are *online*, that is, the constraints are not given all at once but are fed to the decision procedure one at a time, and for each constraint the algorithm performs some relatively small amount of work to take that constraint into account and derive new constraints that follow from it.

In order to understand the reasons for being online and to clarify the important interface features that the decision procedure relies on, we give a brief introduction to the CVC framework. The goal of the following subsection is to provide just enough information about the interface and underlying structure of the CVC framework to understand the requirements for the online version of the decision procedure for mixed integer linear arithmetic. Therefore, some of the features are greatly simplified or omitted. For more details on CVC framework the reader is referred to [BDS00,BDS02b,BDS02a].

### 4.1 Brief Introduction to CVC Framework

At a very high level, CVC can be viewed as a SAT solver for the Boolean skeleton of the quantifier-free first-order formulas (figure 1). The SAT solver treats the atomic constraints from different theories as Boolean variables. It solves the satisfiability problem by *splitting cases* on each variable; that is, picking a variable, assigning it values true and false (making a *decision*), and solving the rest of the formula for each case recursively. If it finds a *satisfying assignment* to the variables, then the original formula is satisfiable. When a particular set of decisions results in a contradiction, the SAT solver backtracks and tries a different decision. If in all branches it derives a contradiction, then the formula is unsatisfiable.

Since the Boolean variables represent constraints from various theories, each time the SAT solver makes a decision, a new constraint is produced, which is simplified and dispatched to the appropriate decision procedure. When a decision procedure receives a constraint, it derives new constraints from the current and previously seen constraints, and asserts them back to the SAT solver. If a contradiction is detected, then the decision procedure asserts false as a new constraint. Note, that the new constraints may contain arbitrary Boolean combinations of atomic formulas, but the decision procedure always receives atomic constraints (equalities and theory-specific predicates over terms). In other words, the decision procedure can assume it always solves the satisfiability problem for a *conjunction of atomic constraints*. However, it is allowed to infer Boolean combinations of new constraints from the input set of constraints.

These decisions are dynamic, which requires decision procedures to be able to receive a new constraint and process it efficiently, deriving a contradiction as soon as possible to cut off the search early. This explains the *online* requirement. In some cases, however, a simplified constraint may be returned directly to the SAT solver without going through a decision procedure.

When the SAT solver backtracks, some of the constraints are effectively removed. Therefore, if a decision procedure stores some information about previously received constraints, it must be able to roll back to the appropriate state when the SAT solver backtracks. In other words, all the data structures which persist across calls to the decision procedure must be *backtrackable*. Below, in the description of the algorithm, we always assume that such backtracking mechanism is properly implemented and is completely transparent to the decision procedure.

To boost the efficiency of the SAT solver, the *intelligent backtracking* technique is utilized along with *conflict clauses* [MSS99]. To construct a conflict clause, one needs to identify a (preferably small) set of decisions made by the SAT solver that lead to the contradiction. One way of identifying such decisions is to extract them from the *proof* that the decision procedure constructs when it derives false. This explains the need for proof production in CVC decision procedures. As a bonus, the proofs can be checked by an external proof checker [Stu02] to increase the confidence in the results produced by CVC.

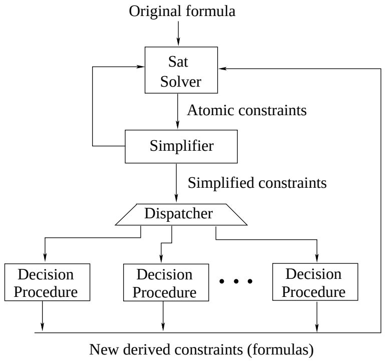

**Fig. 1.** Flow of constraints in CVC

Note, that intelligent backtracking requires only the set of assumptions (from the SAT solver decisions) used in the proof, and does not depend on the specific proof rules. This set can be computed by simply traversing the proof tree to the leaves and collecting the assumptions.

### 4.2 Online Fourier-Motzkin Elimination for Reals

In this and the subsequent sections the term *decision procedure* will refer to the decision procedure component from figure 1. In particular, we can always assume that the constraints dispatched to the decision procedure are *atomic* (no Boolean connectives) and *simplified*. The simplification step consists of theory-specific rewrites such as normalization of arithmetic constraints and elimination of equalities, so that only normalized inequality constraints reach the decision procedure.

Hence, the description of the algorithm consists of two parts: the simplifier, which is a set of equivalent transformations, and the decision procedure itself. The latter is presented as a function that takes a simplified atomic constraint and returns (possibly a Boolean combination of) new constraints back to the framework.

In the algorithm below, we assume a total ordering $\lessdot$ on all variables which defines the order in which the variables are projected from inequalities. In particular, $x$ is said to be the *maximal* variable from a set of variables when it is the highest in this set w.r.t. $\lessdot$. In this section, we only consider equalities and inequalities over real-valued variables. Handling the constraints with $\mathsf{int}(t)$ predicate and integer variables will be described later in section 5.

*Simplification step.* Each equality constraint $t_1 = t_2$ is first rewritten as $t_1 - t_2 = 0$ and simplified by grouping like terms. If the resulting equality contains no variables (meaning $t_1 - t_2$ simplifies to a numerical constant), then it is checked for satisfiability, and the result is reported directly to the top-level SAT solver. Otherwise, it is rewritten in the form $x = \alpha$ for some variable $x$, and then $x$ is replaced by $\alpha$ everywhere in the system, completely eliminating the variable $x$.

Similarly, an inequality $t_1 < t_2$ is rewritten as $t_1 - t_2 < 0$ and simplified. If the left-hand side simplifies to a constant, the inequality is evaluated to $\mathsf{true}$ or $\mathsf{false}$ and submitted back to the solver. Otherwise, it is rewritten as $\beta < x$ or $x < \alpha$ for the maximum variable $x$ in $t_1 - t_2$ w.r.t. $\lessdot$, and forwarded to the decision procedure.

*Decision procedure.* Due to the simplification step above, the decision procedure receives only inequalities of the form $\beta < x$ or $x < \alpha$, where $x$ is the maximal variable in $\alpha$ and $\beta$ w.r.t. $\lessdot$. We say that the variable $x$ in such inequalities is *isolated*.

The decision procedure maintains a backtrackable database $\mathrm{DB}_<$ of inequalities indexed by the isolated variable. Whenever a new inequality $x < \alpha$ arrives, this database is searched for the opposite inequalities $\beta < x$ and for each such inequality the new shadow constraint $\beta < \alpha$ is constructed and asserted back to the framework. The received constraint $x < \alpha$ is then added to the database. The inequalities of the form $\beta < x$ are handled similarly.

The newly generated constraint $\beta < \alpha$ is eventually simplified and submitted back to the decision procedure with a smaller variable (w.r.t. $\lessdot$) isolated, and this process repeats until no variables remain in the constraint.

The ordering $\lessdot$ on the variables guarantees that all the intermediate constraints that would be constructed by the offline version of the procedure are eventually constructed and processed by this online version, provided both algorithms project variables according to the same ordering. Assuming that the original offline version of Fourier-Motzkin elimination is complete and terminating implies that the online procedure is also complete and terminating. We formulate the completeness part of this statement more precisely as a lemma.

**Lemma 1.** ***(Local Completeness.)*** *Let $\mathbf{C} = \{C_1, \ldots, C_k\}$ be a set of linear arithmetic inequalities over real-valued variables $x_1, \ldots, x_n$, and $\lessdot$ be a total ordering on the variables. Let $\mathcal{C}$ be the set of constraints constructed by the offline algorithm while solving the original set of constraints $\mathbf{C}$. Then any constraint $C \in \mathcal{C}$ is also constructed by the online algorithm, regardless of the order in which the original constraints from $\mathbf{C}$ are submitted to the online algorithm.*

For the proof the reader is referred to the technical report version of this paper [BGD02].

## 5 Online Fourier-Motzkin Elimination for Mixed Integer Constraints

The online version of the decision procedure for integers cannot have such a direct correspondence to the offline version, since the order of the projection of variables depends on the integrality constraints, $\mathsf{int}(x)$ and $\neg\mathsf{int}(x)$, and the variable may become known to be integer only after it has already been projected or eliminated. A naive solution would be to backtrack and redo the projection and elimination steps. This could be a very costly operation.

Fortunately, there is a simple and elegant solution to this problem. Whenever a constraint $\mathsf{int}(x)$ arrives, a new constraint $x - \sigma = 0$ is added to the system, where $\sigma$ is a new integer variable, and the fact $\mathsf{int}(\sigma)$ is recorded into a local database $\mathrm{DB}_{\mathrm{int}}$ indexed by $\sigma$. The resulting system is equisatisfiable with the original one (which includes $\mathsf{int}(x)$), but the variable $x$ remains real-valued in the new system. Therefore, the projections and eliminations of $x$ do not have to be redone. At the same time, the integrality of $x$ is enforced by the integrality of $\sigma$.

In addition, for any integer constraint $\mathsf{int}(t)$ in $\mathrm{DB}_{\mathrm{int}}$, whenever the term $t$ is rewritten to $t'$ (because of some variable elimination), and $t'$ simplifies to a constant term $c$, one must check that $c$ is indeed an integer and assert unsatisfiability if it is not.

Just like the algorithm for only real variables, the online algorithm for deciding mixed-integer linear constraints consists of two parts: simplification and decision procedure.

*Simplification step.* This version of the simplifier performs the same transformations as the one for real variables (section 4.2). An equality constraint is first rewritten as $\gamma = 0$ and $\gamma$ is checked for being a constant. If it is, the constraint is immediately checked for satisfiability and the result is returned directly to the SAT solver. Otherwise, if $\gamma$ contains real-valued variables, then one such variable $x$ is isolated and eliminated. If only integer variables remain in $\gamma$, then the iterative equality elimination algorithm is performed, as described in section 3.1. At the end of this process the equality $\gamma = 0$ is rewritten into an equisatisfiable system of equations

$$\begin{cases} x_1 = \beta_1 \\ \vdots \quad \vdots \\ x_k = \beta_k, \end{cases}$$

where each equation $x_i = \beta_i$ corresponds to the equation (5) in section 3.1 from each iteration of the algorithm. All the variables $x_i$ are then eliminated by replacing them with their right-hand sides. Thus, equations are handled in the simplification step and never submitted to the actual decision procedure.

Inequalities are also transformed and simplified into $\gamma < 0$, then evaluated if $\gamma$ is a constant. If $\gamma$ contains variables, the inequality is rewritten to the form $\beta < ax$ or $ax < \alpha$ for some positive integer coefficient $a$, where the variable $x$ is the maximal w.r.t. $\lessdot$. The new inequality is then forwarded to the decision procedure. Similar to the offline version of the algorithm, it is important to project real variables first. Therefore, we define $\lessdot$ such that real-valued variables are always higher in the ordering than the integer ones.

In the constraints of the form $\mathsf{int}(t)$ and $\neg\mathsf{int}(t)$ only the term $t$ is simplified by combining like terms, and otherwise these constraints are passed to the decision procedure unmodified.

### 5.1 The Decision Procedure

First, observe that, due to the simplification step, only inequalities of the form $\beta < bx$ and $ax < \alpha$ and integer constraints $\mathsf{int}(t)$ and $\neg\mathsf{int}(t)$ are submitted to the decision procedure. Notice that inequality constraints always have the maximal variable isolated w.r.t. $\lessdot$. These inequalities are stored in the local database $\mathrm{DB}_<$. Additionally, whenever a term $t$ in any constraint $\mathsf{int}(t) \in \mathrm{DB}_{\mathrm{int}}$ is rewritten to $t'$ by the simplifier, $\mathsf{int}(t)$ is automatically replaced by $\mathsf{int}(t')$ in $\mathrm{DB}_{\mathrm{int}}$. Both local databases are also backtrackable.

The decision procedure receives a constraint $C$ from the simplifier and returns, or *asserts*, new constraints back to the framework. We describe it as a case-by-case analysis of the constraint $C$.

1. $C \equiv \mathsf{int}(t)$:
   (a) If $t$ is a constant, then evaluate $\mathsf{int}(t)$, assert the result to the framework, and return. If t is not a constant, go to step 1b.
   (b) If $t \notin \mathrm{DB}_{\mathrm{int}}$, then create a new integer variable $z$, add $t$ and $z$ into $\mathrm{DB}_{\mathrm{int}}$, assert the new facts:
   $$\mathsf{int}(z) \quad \text{and} \quad t - z = 0.$$
   Otherwise, if $\mathrm{t} \in \mathrm{DB}_{\mathrm{int}}$, then ignore $C$ and return.
2. $C \equiv \neg\mathsf{int}(t)$:
   Introduce a new integer variable $z$, add $z$ to $\mathrm{DB}_{\mathrm{int}}$ and assert the new constraints:
   $$\mathsf{int}(z) \quad \text{and} \quad z < t < z + 1.$$

| #experiments<br>in each suite | #formulas / #vars<br>in each experiment | CVC<br>completed* | Omega<br>completed* | avg. slow-down<br>factor |
|---|---|---|---|---|
| 5996 | 1-4/1-5 | 5990 (99.9%) | 5568 (93.0%) | 13.4 |
| 395 | 1-10/1-20 | 393 (99.5%) | 322 (81.5%) | 192 |
| 65 | 10-50/10-50 | 63 (96.9%) | 8 (12.3%) | 7.8 |

**Table 1.** Experimental comparisons of CVC vs. Omega on suites of randomly generated examples. CVC is generally slower than Omega approximately by a factor of 10.
*An experiment is completed if the tool terminates with the correct answer within 10 minutes.

3. $C \equiv a \cdot x < \alpha$ (or $C \equiv \beta < b \cdot x$):

   (a) Find all inequalities of the form $\beta < b \cdot x$ (respectively, $a \cdot x < \alpha$) in the database $\mathrm{DB}_<$, and for each such inequality perform the following steps:

      i. Generate and assert the real shadow constraint $R \equiv a \cdot \beta < b \cdot \alpha$.

      ii. If $x \in \mathrm{DB}_{\mathrm{int}}$ (in which case all the variables in $\alpha$ and $\beta$ must also be in $\mathrm{DB}_{\mathrm{int}}$), then generate the integrality constraint $\mathbf{D} \vee \mathbf{G}$ (dark and gray shadows), where $\mathbf{D}$ and $\mathbf{G}$ are defined as in section 3.2:

$$\mathbf{D} \equiv b \cdot \alpha - a \cdot \beta > ab \quad \text{and} \quad \mathbf{G} = \bigvee_{i=1}^{b-1} b \cdot x = \beta + i.$$

      Following the heuristic of Pugh [Pug91], the top-level SAT solver should first search for a solution in $\mathbf{D}$ before trying $\mathbf{G}$.

   (b) Add the received constraint $C$ to $\mathrm{DB}_<$ and return.

It is not hard to see that each step of the algorithm above corresponds very closely to the similar steps in the offline version of the algorithm. The soundness and completeness of the procedure follow from the fact that the set of constraints asserted by the decision procedure at each step is always equisatisfiable with the given constraint $C$. The details of the formal proof (including termination) are in the technical report [BGD02].

The experiments summarized in table 1 indicate that most of the time the overhead of the CVC framework and arbitrary precision arithmetic slow down the algorithm only by a constant factor. Since this implementation is not yet tuned for efficiency, there are a few exceptional cases when CVC performs much worse than Omega, which explains the large slow-down factor in the second line of table 1. Specifically, in a few exceptionally slow examples in the second suite (the slowest takes less than 6 seconds) the slowdown is due to the large CVC framework overhead (over 90%), and lack of optimizations. In any case, this is a very reasonable price to pay for having the arithmetic decision procedure combined with other theories of CVC, since in many practical applications the arithmetic problems are very small and are not the bottleneck of the verification. Additionally, the resulting implementation proves to be much more stable than the original Omega-test, and produces proofs.

# 6 Proof Production

When the algorithm in section 5 reports that the system of constraints is unsatisfiable, it produces a proof of this fact which can be verified independently by an external proof checker. This increases our confidence in the soundness of the implementation.

Additionally, the proof production mechanism allows CVC framework to extract logical dependencies that drive the backtracking mechanism of the built-in SAT solver (see section 4.1). The details are out of the scope of this paper, but intuitively, if the proof of unsatisfiability depends only on a small subset of decisions made by the SAT solver, then the SAT solver memorizes this combination of decisions and avoids it in the future. This can dramatically reduce the size of the decision tree.

Due to the page limit, only a few proof rules are presented in this version of the paper. For the complete description of the proof system the reader is referred to the technical report [BGD02].

## 6.1 Natural Deduction

The proofs are represented as derivations in natural deduction extended with arithmetic and with specialized derived or admissible rules. The algorithm maintains the invariant that every constraint appearing in the algorithm has an associated proof with it. Since the online decision procedure is presented as a set of relatively simple transformations, it is natural to provide a specialized proof rule for each such transformation. These specialized rules can then be proven sound externally, either by manual inspection or with the help of automated theorem provers.

**Definitions.** An *inference rule*, or a *proof rule*, in general, is of the form

$$\frac{P_1 \;\cdots\; P_n \quad S_1 \;\cdots\; S_m}{C}\,\mathrm{rule\,name}$$

where formulas $P_i$ are the *premisses* (they are assumed to have proofs), $S_i$ are side conditions (the rule is applicable only if all $S_i$ are true), and the formula $C$ is the *conclusion* of the rule. The semantics of a proof rule is that if all $P_i$ are valid, then $C$ is also valid, provided that all the side conditions $S_j$ are true. In CVC, however, the dual semantics is used, i.e. whenever $C$ is unsatisfiable, the system of constraints $\{P_1, \ldots, P_n\}$ is also unsatisfiable, provided that all side conditions $S_j$ hold.

Recall, that our algorithm consists of two parts: the simplifier, which performs equivalent transformations, and the decision procedure, which derives new constraints out of existing ones. The rules for the equivalent transformations in the simplifier have a special form:

$$\frac{S_1 \;\cdots\; S_m}{t_1 \equiv t_2}$$

where $t_1$ and $t_2$ are arithmetic terms or constraints and $S_i$ are side conditions (there are no premisses). The rules for the decision procedure steps normally have premisses, which are the constraints submitted to the decision procedure.

### 6.2 Proof Rules for Equivalent Transformations

**Normalization.** The simplifier step normalizes the constraints by making all coefficients relatively prime integers, which is done by multiplying a constraint by a constant. The corresponding proof rules state that multiplying an (in)equality by a (positive) number preserves the constraint. Below, only the rule for inequality is shown.

$$\frac{b \in \mathcal{R},\ b > 0}{\sum_{i=1}^{n} a_i \cdot x + c < 0 \quad \equiv \quad b \cdot (\sum_{i=1}^{n} a_i \cdot x + c) < 0}\mathrm{norm}_{<}$$

**Variable elimination for equalities.** Given an (in)equality, pick a variable to isolate and transform the constraint in such a way that the variable with some positive coefficient is solely on one side, and the rest of the term is on the other. For equalities, the isolated variable must always be on the left-hand side. For inequalities, it depends on the sign of the coefficient: if it is positive, the variable stays on the left-hand side, and all the other terms are moved to the right-hand side; otherwise the variable is isolated on the right-hand side. We only show one proof rule for equality and positive coefficient below. The other 3 rules (one for equality and negative coefficient, and the two cases for inequalities) are similar.

$$\frac{a_i > 0}{c + \sum_{j=1}^{n} a_j \cdot x_j = 0 \equiv a_i \cdot x_i = -\left(c + \sum_{j \in [1 \ldots n] - \{i\}} a_j \cdot x_j\right)}\mathrm{VI}^{+}_{=}$$

The next rule is for solving a constraint for the isolated variable $x$:

$$\frac{a \neq 0}{a \cdot x = \alpha \quad \equiv \quad x = \alpha / a}\mathrm{Eq\ Elim.}$$

If the variable $x$ is real-valued, then it can be eliminated from the system and replaced by $\alpha/a$. The rules for integer variable elimination are more complex, but similar in spirit, and are omitted from this version of the paper.

### 6.3 Proof Rules for Inequalities

The proof rules in this section derive new constraints from already existing ones. These types of rules correspond to the actual Fourier-Motzkin projection of variables from inequalities.

**Real shadow.** Deriving the real shadow from two opposing constraints makes a simple and obvious proof rule:

$$\frac{\beta < b \cdot x \quad a \cdot x < \alpha}{a \cdot \beta < b \cdot \alpha}\mathrm{Real\ Shadow.}$$

The rules for introducing dark and gray shadows follow the style of the $\mathrm{Real\ Shadow}$ rule. In its simplified form, the rule can be given as follows:

$$\frac{\beta < b \cdot x \quad a \cdot x < \alpha \quad \mathsf{int}(x)}{\mathbf{D} \vee \mathbf{G}}\mathrm{Int\ Shadows,}$$

where $\alpha$ and $\beta$ contain only integer variables, and $\mathbf{D}$ and $\mathbf{G}$ formulas are defined as in the algorithm in section 5.1.

The complete set of rules is described in the full version of this paper [BGD02]. These rules can be thought of as a practical axiomatization of linear arithmetic, where every step in the decision procedure has a corresponding proof rule justifying that step.

## 7 Conclusion

This paper presents the theory and some implementation detail of an *online* and *proof producing* decision procedure for a theory of mixed-integer linear arithmetic extended with the $\mathsf{int}()$ predicate. Additionally, the decision procedure supports arbitrary precision arithmetic.

A decision procedure is much more useful to the research community when it can be combined with other decision procedures. Therefore, designing a stand-alone decision procedure is only the very first step in the design process. The next and more difficult task is to enhance the algorithm with additional properties which enable it to communicate with other decision procedures. Namely, the decision procedure must be *online* and *proof producing*, and must support *backtracking*.

In our experience, conceptually the most difficult is the online property. Just adapting the original Omega-test to an online algorithm required significant efforts (before any implementation!). Proof production is the next difficult problem in the design process. It could have easily been the hardest one if CVC did not already have a thoroughly worked-out methodology for adding proof production to existing decision procedures. Nevertheless, the implementation and especially debugging of proof production still presents a challenge. Finally, backtracking is relatively easy to design and implement in the context of CVC, since the framework provides all the necessary data structures.

Since our algorithm is largely based on Omega-test, its performance is comparable with that of the original implementation of Omega-test. The overhead of the CVC framework and the additional requirements on the decision procedure slow it down by about a factor of 10. This is a very reasonable price to pay for having an arithmetic decision procedure be combined with other powerful decision procedures of CVC.

This reimplementation adds arbitrary precision arithmetic, and generally is much more stable than the Omega library code. The arbitrary precision arithmetic is crucial for solving sizable systems of mixed-integer constraints using Fourier-Motzkin approach, since repeatedly generating shadow constraints produces large coefficients even if the coefficients in the original input are relatively small.

**Acknowledgements.** The authors would like to thank the principal developers of CVC Aaron Stump and Clark Barrett for many insightful discussions about the theory and implementation of this decision procedure.

## References

[BDS00] C. Barrett, D. Dill, and A. Stump. A Framework for Cooperating Decision Procedures. In David McAllester, editor, *17th International Conference on Computer Aided Deduction*, volume 1831 of *LNAI*, pages 79–97. Springer-Verlag, 2000.

[BDS02a] C. Barrett, D. Dill, and A. Stump. A Generalization of Shostak's Method for Combining Decision Procedures. In *4th International Workshop on Frontiers of Combining Systems (FroCos)*, 2002.

[BDS02b] C. Barrett, D. Dill, and A. Stump. Checking Satisfiability of First-Order Formulas by Incremental Translation to SAT. In *14th International Conference on Computer-Aided Verification*, 2002.

[BGD02] Sergey Berezin, Vijay Ganesh, and David L. Dill. Online proof-producing decision procedure for mixed-integer linear arithmetic. Unpublished manuscript. URL: http://www.cs.cmu.edu/~berez/publications.html, 2002.

[BT97] Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, Belmont, Massachusetts, 1997.

[DE73] George B. Dantzig and B. Curtis Eaves. Fourier-Motzkin elimination and its dual. *Journal of Combinatorial Theory (A)*, 14:288–297, 1973.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, July 1960.

[GMP] GMP library for arbitrary precision arithmetic. URL: http://swox.com/gmp.

[MMZ$^{+}$01] M. Moskewicz, C. Madigan, Y. Zhaod, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *39th Design Automation Conference*, 2001.

[Möl98] M. Oliver Möller. Solving bit-vector equations - a decision procedure for hardware verification, 1998. Diploma Thesis, available at http://www.informatik.uni-ulm.de/ki/Bitvector/.

[MSS99] J. Marques-Silva and K. Sakallah. GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[NL00] George C. Necula and Peter Lee. Proof generation in the Touchstone theorem prover. In David McAllester, editor, *17th International Conference on Computer-Aided Deduction*, volume 1831 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, June 2000. Pittsburgh, Pennsylvania.

[NO79] G. Nelson and D. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–57, 1979.

[Pug91] William Pugh. The omega test: a fast and practical integer programming algorithm for dependence analysis. In *Supercomputing*, pages 4–13, 1991.

[RS01] H. Ruess and N. Shankar. Deconstructing Shostak. In *16th IEEE Symposium on Logic in Computer Science*, 2001.

[SBD02] A. Stump, C. Barrett, and D. Dill. CVC: a Cooperating Validity Checker. In *14th International Conference on Computer-Aided Verification*, 2002.

[SBDL01] A. Stump, C. Barrett, D. Dill, and J. Levitt. A Decision Procedure for an Extensional Theory of Arrays. In *16th IEEE Symposium on Logic in Computer Science*, pages 29–37. IEEE Computer Society, 2001.

[Sho84] R. Shostak. Deciding combinations of theories. *Journal of the Association for Computing Machinery*, 31(1):1–12, 1984.

[SS96] J. P. M. Silva and K. A. Sakallah. GRASP – A new search algorithm for satisfiability. In *Proceedings of the ACM/IEEE International Conference on Computer-Aided Design*, pages 220–227, 11 1996.

[Stu02] A. Stump. *Checking Validities and Proofs with CVC and flea*. PhD thesis, Stanford University, 2002. In preparation: check http://verify.stanford.edu/˜ stump/ for a draft.

[Wil76] H. P. Williams. Fourier-Motzkin elimination extension to integer programming problems. *Journal of Combinatorial Theory (A)*, 21:118–123, 1976.