Bachelor of Science Thesis

SMT Solving for Linear Integer Arithmetic

Dustin Hütter

Supervisors:
Prof. Dr. Erika Ábrahám
Prof. Dr. Jürgen Giesl

Advisor:
Dipl.-Inf. Florian Corzilius

28.08.2014

Erklärung

Hiermit versichere ich, dass ich die vorgelegte Arbeit selbstständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe. Alle benutzten Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate wurden als solche gekennzeichnet.

Dustin Hütter
Aachen, den 28. August 2014

Abstract

Deciding the satisfiability of systems of linear inequalities over integers is a crucial problem in several domains e.g. formal verification and scheduling problems. Multiple existing approaches in linear integer arithmetic each provide different advantages and disadvantages. In this thesis we address three simplex-based algorithms namely *branch and bound*, *Gomory cuts* and *cuts from proofs* just as possible optimizations leading to an efficient implementation of those into the SMT solver *SMT-RAT*. Furthermore, they are combined such that the resulting global strategy solves a wide range of benchmarks in an appropriate amount of time and memory usage. The experimental results show that the chosen global strategy can keep up with state-of-the-art SMT solvers.

Contents

Chapter 1

Introduction

1.1 Historical Context

George Dantzig [CJW07] published in 1947, employed at the *U.S. Air Force* after declining an offer of the *University of California, Berkeley*, an article introducing the *simplex algorithm*. After the second world war raised the necessity to allocate the available resources more efficiently, his employer used his findings to solve those optimization problems including the production of war commodities and the coordination of operations. In the following several other disciplines made use of Dantzig's simplex method in order to optimize e.g. vehicle routes and commodity flows. The solutions obtained by applying the simplex algorithm are real-valued but more and more problems occurred making it necessary to ensure that at least some of the involved variables are integer-valued e.g. in scheduling and transporting problems, compiler construction as well as in formal verification. Therefore Ralph Gomory, a pioneer in the field of *linear integer arithmetic*, introduced a *cutting-planes* method in the 1950s whose functionality we will see later. Linear integer arithmetic is still an open field of research as for example the third algorithm that we consider, called *cuts from proofs*, has been recently developed.

1.2 Problem Definition

The satisfiability problem in linear integer arithmetic (*LIA*) is defined as follows:

Input:

$A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$ and an n-dimensional vector $\vec{x}$ of variables with domain D.

Output:

$$\begin{cases} SAT, & \text{if } \exists \vec{x} \in D : A\vec{x} \leq b \\ UNSAT, & otherwise \end{cases} .$$

The native case in LIA is that $D = \mathbb{Z}^n$, meaning that all components of $\vec{x}$ have an integer domain. Often in real-world problems the more general case occurs involving that some of the solution components have a real and some have an integer domain, formally: $D = \mathbb{R}^s \times \mathbb{Z}^t$ with $s + t = n$. Instances with such a setting belong to the field of *mixed LIA*. The formalisms and algorithms in this thesis consider the mixed LIA case. From now on we will therefore not distinguish between pure LIA problems and their generalization. The satisfiability problem that we address is NP-complete and therefore probably not solvable by a polynomial time algorithm. The proof for the NP-completeness can be found in [Sch98].

As already mentioned, problems, which make it necessary to ensure that at least some of the involved variables are integer-valued, occur in several domains. While one can apply the simplex method in *linear real arithmetic* (*LRA*), $\vec{x} \in \mathbb{R}^n$, which performs very well in practice, developing and extending LIA algorithms is still part of current research. This thesis focuses on branch and bound (BB), Gomory cuts and cuts from proofs (CFP). I have implemented these methods in the SMT solver *SMT-RAT* as an extension of an implementation of the simplex method.

1.3 Related Work

Besides the presented simplex-based techniques there are two main approaches trying to tackle LIA problems from another perspective.

1.3.1 Omega Test

One of those is the *Omega test* presented in [Pug91]. It is an extension of the *Fourier-Motzkin elimination* [DE73] to solve linear real arithmetic problems. Fourier-Motzkin elimination basically eliminates one real-valued variable per iteration from the considered system of linear inequalities such that the newly obtained system with one dimension less is solvable if and only if the previous system is solvable. This elimination is repeated until all variables are eliminated. We eliminate one variable, say x, by combining all its lower bounds $l_1 \leq x, ..., l_r \leq x$ with all its upper bounds $u_1 \geq x, ..., u_s \geq x$ and obtain the formula $\bigwedge_{\substack{1 \leq i \leq r \\ 1 \leq j \leq s}} l_i \leq u_j$. In the context of the Omega test, the Fourier-Motzkin elimination is applied to the real relaxation of a mixed LIA problem in the *real shadow*. If the real shadow of the given system has no real solutions we can deduce that this also true for the original system over integers and return $UNSAT$. The converse argument is not true. Therefore the Omega test introduces a so-called *dark shadow* for which a sufficient condition guarantees an integer solution. If an integer point is detected in the dark shadow we can return *SAT*, elsewise the algorithm completes its proceeding with searching for a solution in the set theoretic difference $R \setminus D$ of the real shadow R and the dark shadow D, called *grey shadow* G in the terminology of this algorithm. If G contains an integer point the algorithm returns SAT otherwise $UNSAT$. The author of [Pug91] points out that the Omega test is beneficial especially in data dependency analysis. However, although it is complete, it is not used in current SMT solvers as it scales worse than the other approaches with a growing number of variables.

1.3.2 Automata-theoretic Approach

Another approach presented in [Min] makes use of automata theory. Given a formula $\phi(x_1,...,x_n)$ being a conjunction of linear inequalities over $(x_1,...,x_n)^T \in \mathbb{Z}^n$, the aim is to construct an automaton A_ϕ accepting $(x_1,...,x_n)^T \in \mathbb{Z}^n$ if and only if $(x_1,...,x_n)^T \models \phi$, i.e., if $(x_1,...,x_n)^T$ is a solution of ϕ. Hence the problem of determining satisfiability for a system of linear inequalities is reduced on checking whether A_ϕ has an accepting path. Although this approach is complete, as stated in [DDA09] it can not compete against any of the other techniques.

Chapter 2

Preliminaries

This section briefly introduces some important notations and formalisms that are used throughout this thesis.

2.1 Terminology

$\alpha(x)$: current assignment of variable x

l_x, u_x ; lower/upper bound of variable x

$dom(x)$: domain of variable x

Φ, Ψ : set of formulas

ϕ, ψ : single formulas

SAT, *UNSAT*, *UNKNOWN* : return values of decision procedures for "satisfiable", "unsatisfiable" and "unknown"

2.2 QFLIA Formulas

In order to have a formal fundament for the LIA expressions that we consider, we introduce quantifier-free mixed linear integer arithmetic (QFLIA) formulas. In the sense of the following abstract grammar, QFLIA formulas ϕ are Boolean combinations of constraints comparing linear expressions (multivariate linear polynomials) to 0 with respect to $\leq$ and $\geq$.

Definition 2.2.1 (QFLIA formulas).

A QFLIA formula ϕ is of the following form:

$$\begin{aligned} \phi &::= \ c \ \mid \ (\phi \wedge \phi) \ \mid \ \neg\phi \\ c &::= \ p \leq 0 \ \mid \ p \geq 0 \\ p &::= \ z \ \mid \ z \cdot x \quad \mid \ p + p \end{aligned}$$

, where $z \in \mathbb{Z}$ is an integer constant and x is a real- or integer-valued variable. In an expression $z \cdot x$ we call z a coefficient *and x a* monomial*. Terms p are called* polynomials *and c are called* constraints*. The semantics of QFLIA formulas is defined in the common way.*

Note that we can represent $\phi \vee \phi$ by $\neg(\neg\phi \wedge \neg\phi)$. Furthermore, $p \circ 0$, with p being a multivariate linear polynomial, for $\circ \in \{< \ , \ > \ , \ \neq \ , \ =\}$, is constructible with the given grammar by $p < 0 \equiv \neg(p \geq 0)$, $p > 0 \equiv \neg(p \leq 0)$, $p \neq 0 \equiv p < 0 \vee p < 0)$ and $p = 0 \equiv \neg(p \neq 0)$. We also omit the multiplication sign when its clear from the context and we sometimes use for clarification more brackets than are required by the grammar. In addition to that, we also write $p_1 \sim p_2$ for $p_1 + p_2' \sim 0$ with $\sim \ \in \{< \ , \ \leq \ , \ = \ , \ \neq \ , \ > \ , \ \geq\}$ and p_2' results from p_2 by multiplying each coefficient with -1.

Example 2.2.1.

The following formula ϕ is a QFLIA formula with $dom(x_1) = \mathbb{R}$, $dom(x_2) = dom(x_3) = \mathbb{Z}$:

$$\phi := (x_1 - 3x_2 + 4x_3 + 1 \leq 0 \wedge x_1 \geq 0)$$

We denote the left-hand-side of a constraint c by p_c and implicitly assume that the domain requirements are inherent to the formula.

2.3 Polytopes and Relaxation

Definition 2.3.1 (Convex Polytope). *A* convex polytope *is a set $P \subseteq \mathbb{R}^n$ such that there are $A \in \mathbb{Z}^{m \times n}$ and $b \in \mathbb{Z}^m$ such that P is the solution set for the real-valued variables $\vec{x} = (x_1, ..., x_n)^T$ in $A\vec{x} \leq b$.*

Definition 2.3.2 (Relaxation). *Given a system of linear inequalities $A\vec{x} \leq b$ with constants $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$ and variables $\vec{x} \in \mathbb{R}^s \times \mathbb{Z}^t$ with $s + t = n$, the system's* relaxation *is the extension of the integer domains to $dom(x_i) = \mathbb{R}$ for $i \in \{s+1,...,n\}$.*

In the following we will denote the relaxation of a system given by a QFLIA formula ϕ by $Rel(\phi)$. Furthermore we will often use this concept by considering the relaxation of a given LIA problem and then adding appropriate constraints leading to a solution not violating any domain requirements.

Figure 2.1 illustrates the introduced terms in a graphical representation:

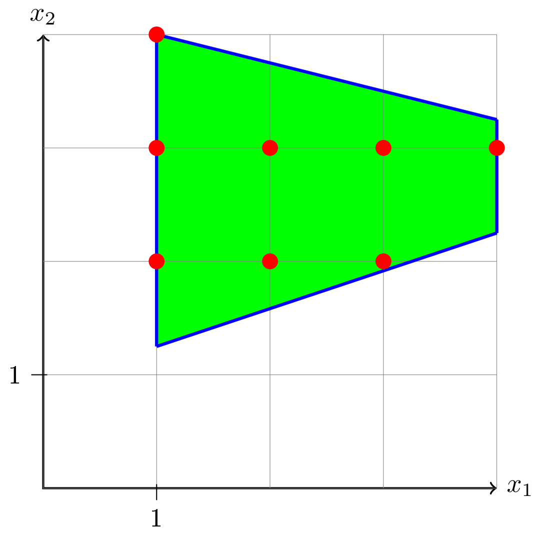


Figure 2.1: Convex polytope with integer solutions

In Figure 2.1, we see a two-dimensional convex polytope defined by four inequalities. The green area represents the set of all real points satisfying all of these inequalities including the borders and is therefore the convex polytope of this system. If we restrict this system such that all variables have to be integer-valued the set of solutions is represented by the red integer points inside the polytope. Conversely the real polytope is the relaxation of the pure integer system.

2.4 Satisfiability Modulo Theories

SMT solving is the process of checking the satisfiability of Boolean combinations of constraints belonging to some theory e.g. linear real arithmetic by combining a SAT solver with a solver for the theory underlying the input formula. After the *Boolean skeleton* of the input formula has been generated, meaning that all constraints are substituted by fresh Boolean variables, the SAT solver either determines the unsatisfiability of this Boolean skeleton and hence for the input formula or finds a satisfying assignment for it. In the latter case, the respective theory solver checks whether this assignment is consistent with the underlying theory. If so the theory solver returns SAT for the input formula, otherwise $UNSAT$ for the current assignment and communicates this result to the SAT solver. If the theory solver determines the unsatisfiability of its input constraints, it detects at least one infeasible subset of these constraints. The SAT solver negates the combination of the corresponding literals and adds a clause containing them negated to its considered Boolean formula.

Figure 2.2 exemplifies this process:

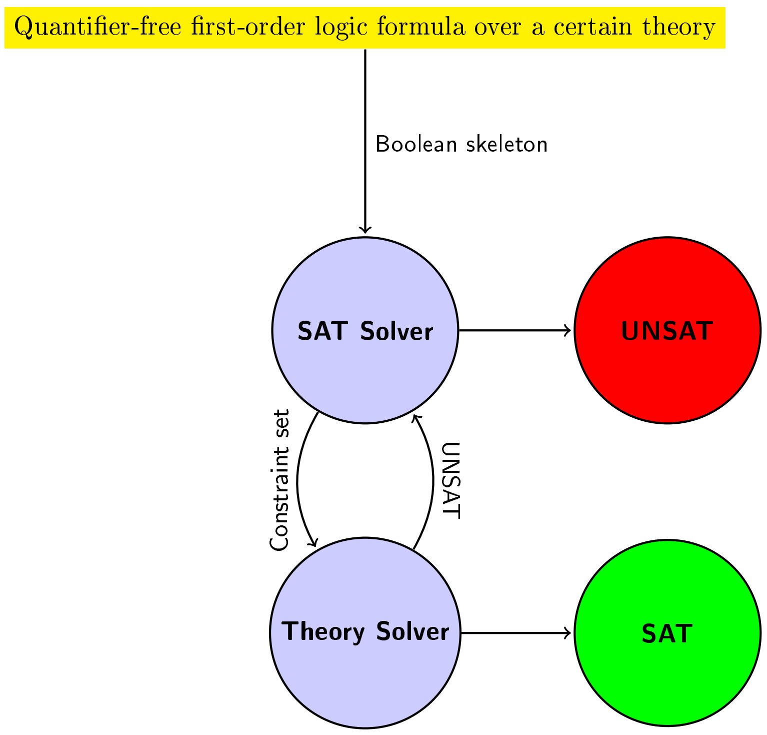


Figure 2.2: SMT solving

2.5 SMT-RAT

SMT-RAT is the C++ framework in which I have implemented the later presented algorithms. It generalizes the classical SMT procedure. The architecture of *SMT-RAT* consists of *modules*, a *strategy* and a *manager*.

Modules offer functionality to check the satisfiability of an SMT formula ϕ. A module has a *set of input formulas* $\Phi = \{\phi_1, ..., \phi_n\}$, in our case QFLIA formulas, which stands for the formula $\bigwedge_{i=1}^{n} \phi_i$. In order to execute the satisfiability check, which the module M provides, we call $check_M(\Phi)$. Since the approaches offered by the modules are not always complete, SAT and $UNSAT$ are not the only possible return values but also $UNKNOWN$. Modules can add and remove SMT formulas to/from the set of input formulas via $add_M(\phi)$ respectively $remove_M(\phi)$). If a module determines unsatisfiability for its set of input formulas, it returns besides $UNSAT$ at least one infeasible subset. This can be used to accelerate the solving process. In addition to that a module can create lemmas, which are, in our case, QFLIA tautologies. The latter exhibit information which is deduced from a modules' internal state and are communicated to the other modules.

SMT-RAT exhibits the opportunity of modules interacting with each other by calling the method $runBackends_M(\Psi)$ submitted by a module M asking certain other modules to check the satisfiability of the set of QFLIA formulas. The *manager* administrates such calls according to a *strategy* which is basically a tree-structure coding under which circumstances the method $runBackends_M(\Psi)$ leads to a call $check_{M'}(\Psi)$ of a module M' that can handle Ψ.

In our setup, the *CNFerModule* receives an arbitrary QFLIA formula, say ϕ and translates it into an equisatisfiable formula, say ϕ_{CNF}, in conjunctive normal form (CNF). After the SAT module has obtained ϕ_{CNF} and determined a satisfying assignment for the Boolean skeleton of it, the LRA module checks whether this assignment is consistent. The LRA module provides an implementation of the simplex method which is presented later. Since the simplex method generates rational solutions, lemmas are added here in order to preserve the domain requirements. After lemmas were *learned* in the LRA module, it returns UNKNOWN such that these are involved in the solving process by the SAT solver by adding them as clauses to ϕ_{CNF}.

Note that the generalization of the *SMT-RAT* practice compared to classical SMT solving is constituted by allowing the classical SMT interaction between any modules and, therefore, also theory modules can collaborate.

Figure 2.3 is a graphical representation of a SMT-RAT setup taken from [CLJÁ12]:

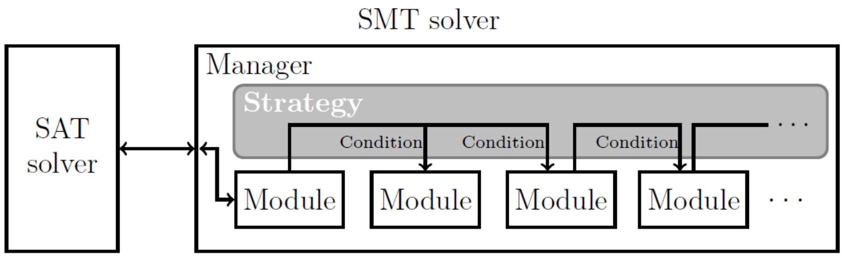


Figure 2.3: A possible *SMT-RAT* setup

For further information about SMT-RAT the reader may consult [JLCÁ13] or [CLJÁ12].

2.6 Simplex

The simplex algorithm is originally used in linear optimization over $\mathbb{R}$. More precisely, the simplex algorithm maximizes $c^T \cdot \vec{x}$ for a given vector of benefit $c \in \mathbb{R}^n$ and a possible solution vector $\vec{x} \in \mathbb{R}^n$ such that the inequalities given by $A\vec{x} \leq \vec{b}$ with $A \in \mathbb{Q}^{m\times n}$ and $b \in \mathbb{Q}^m$ are satisfied. Since this thesis addresses the satisfiability problem, this method is not directly suitable for our purposes. We will therefore introduce a slightly modified version of the latter called *general simplex* which efficiently decides the satisfiability problem over $\mathbb{R}$. We also want to guarantee that (some of) the components of $\vec{x}$ are integer-valued. As we will see later, the general simplex method is still applied in LIA by combining it with sophisticated techniques establishing the integer requirements. The following variant of the general simplex algorithm bases on the one presented in [KS08].

2.6.1 Preprocessing

Given m linear inequalities of the form

$$\sum_{i=1}^{n} a_i x_i \circ b_j \text{ with } \circ \in \{= , \leq , \geq\},$$

the first step of the general simplex method transforms these into the so called *general form.* This is done by replacing

$$\sum_{i=1}^{n} a_i x_i \circ b_j$$

with

$$\sum_{i=1}^{n} a_i x_i - s_j = 0 \tag{2.1}$$

$$s_j \circ b_j \tag{2.2}$$

for all $j \in \{1, ..., m\}$. The s_j are called *slack variables.* The transformed original system in general form is equisatisfiable to the original system. The m equations that we receive can be rewritten as a matrix A' of dimension $m \times (n + m)$ of the form:

$$A' = \begin{array}{c} \begin{array}{cccccc} x_1 & \dots & x_n & s_1 & \dots & s_m \end{array} \\ \begin{pmatrix} a_{11} & \dots & a_{1n} & -1 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} & 0 & \dots & -1 \end{pmatrix} \end{array}$$

For clarity every column is captioned with the variable that corresponds to the coefficients in this column.

General simplex works with a so called *tableau* which we receive from A' by reformulating every row $i \in \{1, ..., m\}$ as an equation depending on the slack variable s_i. We obtain

$$\begin{array}{c|ccc} & x_1 & \dots & x_n \\ s_1 & a_{11} & \dots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ s_m & a_{m1} & \dots & a_{mn} \end{array} .$$

We call the variables representing columns *non-basic variables*, denoted by N, and the variables representing rows *basic variables*, denoted by B. At first the slack variables represent the basic variables and the original variables of the given system represent the non-basic variables and all occurring variables x_1, ..., x_n, s_1, ..., s_m are assigned to zero, which satisfies the equation system (2.1) being one invariant of the simplex algorithm. The second condition (2.2) requires that all non-basic variables have to satisfy their bounds.

2.6.2 Algorithm

After we have transformed the input system into tableau form and assigned the variables their initial value 0, the actual algorithm can be executed. If all variables, basic and non-basic variables, satisfy their bounds then we return *SAT*. Otherwise there exists a basic variable χ, say in row i, not satisfying its bounds which either means that $\alpha(\chi) < l_\chi$ or $\alpha(\chi) > u_\chi$. Without loss of generality we consider the first case. The second case is completely analogous. Let χ be represented by $\sum_{k=1}^{n} a_{ik}x_k$.

We need to find an $x_j \in N$ such that:

$$(a_{ij} > 0 \wedge \alpha(x_j) < u_{x_j}) \vee (a_{ij} < 0 \wedge \alpha(x_j) > l_{x_j})$$

Informally spoken this means that we can fix the invalid assignment of χ either by increasing the assignment of a non-basic variable occurring in the row of χ with a positive coefficient such that the current assignment of this variable is smaller than its upper bound or by decreasing the assignment of a non-basic variable occurring in the row of χ with a negative coefficient such that the current assignment of this variable is greater than its lower bound.

If such a non-basic variable does not exist then we can not fix the assignment of the variable χ without violating the bounds. In this case we return *UNSAT*. Otherwise we call x_j ***suitable*** and execute the following ***pivoting step***:

- Rewrite

  $$\chi = \sum_{k=1}^{n} a_{ik}x_k$$

  as

  $$x_j = \frac{\chi}{a_{ij}} - \sum_{k \in \{1,\dots,n\} \setminus \{j\}} \frac{a_{ik}}{a_{ij}} x_k \ .$$

- Swap χ and x_j and adapt the coefficients in the row that is from now on represented by x_j. Also update the coefficients in the remaining rows.

- Update the assignment of x_j in the following way:

  $$\alpha(x_j) = \alpha(x_j) + \frac{l_\chi - \alpha(\chi)}{a_{ij}}$$

  and set $\alpha(\chi)$ to l_χ.

- Update the assignments of the basic variables $x \in B \setminus \{x_j\}$ such that the equations they represent hold.

This pivotization is repeated until either all basic variables satisfy their bounds and *SAT* is returned or we do not find a suitable non-basic variable and return *UNSAT*. The completeness of the algorithm bases on a fixed variable order.

Example 2.6.1.

Given the inequation system

$$\begin{aligned} 2x_1 + x_2 &\geq 3 \\ -3x_1 + x_2 &\geq 0 \\ x_2 &\geq -1 \ , \end{aligned}$$

its transformation to general form results in

$$\begin{aligned} 2x_1 + x_2 - s_1 &= 0 \\ -3x_1 + x_2 - s_2 &= 0 \\ x_2 - s_3 &= 0 \\ s_1 &\geq 3 \\ s_2 &\geq 0 \\ s_3 &\geq -1 \ . \end{aligned}$$

Deduce the initial tableau from the general form:

$$\begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \left[\begin{array}{cc} x_1 & x_2 \\ \hline 2 & 1 \\ -3 & 1 \\ 0 & 1 \end{array}\right]$$

with the bound requirements $s_1 \geq 3$, $s_2 \geq 0$, $s_3 \geq -1$, variable order $x_1 < x_2 < s_1 < s_2 < s_3$ and initial assignments: $\alpha(s_1) = \alpha(s_2) = \alpha(s_3) = \alpha(x_1) = \alpha(x_2) = 0$. Hence s_1 violates its lower bound and we pivot with the suitable non-basic variable x_1 (suitable because $u_{x_1} = \infty$):

$$s_1 = 2x_1 + x_2 \Leftrightarrow x_1 = \tfrac{1}{2}s_1 - \tfrac{1}{2}x_2$$

Substitute $\frac{1}{2}s_1 - \frac{1}{2}x_2$ for x_1 in the remaining rows:

$$s_2 = -3(\tfrac{1}{2}s_1 - \tfrac{1}{2}x_2) + x_2 \Leftrightarrow s_2 = -\tfrac{3}{2}s_1 + \tfrac{5}{2}x_2$$

$$s_3 = 0(\tfrac{1}{2}s_1 - \tfrac{1}{2}x_2) + x_2 \Leftrightarrow s_3 = x_2$$

From that we obtain the new tableau:

$$\begin{array}{c} \\ x_1 \\ s_2 \\ s_3 \end{array} \left[\begin{array}{cc} s_1 & x_2 \\ \hline \frac{1}{2} & -\frac{1}{2} \\ -\frac{3}{2} & \frac{5}{2} \\ 0 & 1 \end{array}\right]$$

Adapt the assignments:

$$\alpha(x_1) = 0 + \tfrac{3-0}{2} = \tfrac{3}{2},\ \alpha(x_2) = 0\ \alpha(s_1) = 3,\ \alpha(s_2) = -\tfrac{9}{2},\ \alpha(s_3) = 0$$

After the first pivoting step s_2 violates its lower bound. Hence we pivot again with the suitable non-basic variable x_2:

$$s_2 = -\tfrac{3}{2}s_1 + \tfrac{5}{2}x_2 \Leftrightarrow x_2 = \tfrac{3}{5}s_1 + \tfrac{2}{5}s_2$$

Substitute $\frac{3}{5}s_1 + \frac{2}{5}s_2$ for x_2 in the remaining rows:

$$x_1 = \tfrac{1}{2}s_1 - \tfrac{1}{2}(\tfrac{3}{5}s_1 + \tfrac{2}{5}s_2) \Leftrightarrow x_1 = \tfrac{1}{5}s_1 - \tfrac{1}{5}s_2$$

$$s_3 = \tfrac{3}{5}s_1 + \tfrac{2}{5}s_2$$

From that we obtain the new tableau:

$$\begin{array}{c} \\ x_1 \\ x_2 \\ s_3 \end{array} \left[\begin{array}{cc} s_1 & s_2 \\ \hline \frac{1}{5} & -\frac{1}{5} \\ \frac{3}{5} & \frac{2}{5} \\ \frac{3}{5} & \frac{2}{5} \end{array}\right]$$

Adapt the assignments:

$$\alpha(x_2) = 0 + \frac{0-(-\frac{9}{2})}{\frac{5}{2}} = \tfrac{9}{5}\ \alpha(s_2) = 0\ \alpha(x_1) = \tfrac{3}{5}\ \alpha(s_3) = \tfrac{9}{5}$$

Since all constraints are satisfied we can deduce the satisfiability of the initial system.

Figure 2.4 shows the obtained feasible solution of the given system.

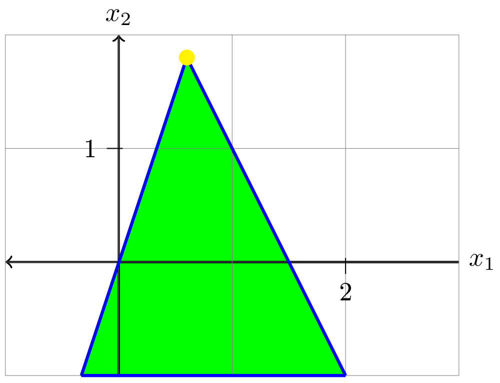


Figure 2.4: Feasible solution of the polytope from Example 2.6.1.

The simplex method is correct and complete which is shown e.g. in [DDM06]. The completeness bases on

2.7 Tableau

In order to understand the adjustments that were necessary to embed the algorithms into *SMT-RAT* we will briefly have a look at the most important data structure of the LRA module, namely the so called *tableau*. It stores the coefficients of the given system of linear inequalities in a matrix-like structure including rows and columns. Each column j exhibits the coefficients of a multivariate linear polynomial p_j which is given by $p_j = \sum\limits_{\forall k : a_{kj} \neq 0} a_{kj} p'_k$. Each row contains all coefficients such that the resulting polynomial p'_i, also multivariate and linear, is analogously given by $p'_i = \sum\limits_{\forall k : a_{ik} \neq 0} a_{ik} p_k$. Furthermore, each row and column contains the assignment and the bounds of the variable representing the polynomial of the corresponding row respectively column. Since the problems occurring in LIA are typically sparsed, it would not be efficient to store the zero values. Therefore every coefficient $a_{ij} \neq 0$ has links to its neighbors in all cardinal directions skipping the zero coefficients. Note that, for performance issues, the tableau which is used for the computations contains only integer coefficients. Therefore the rows of input instances containing coefficients $a_{ij} \in \mathbb{Q} \setminus \mathbb{Z}$ are normalized such that the corresponding rows contain only integers.

Figure 2.5 illustrates the tableau:

$$
p'_i \left[\begin{array}{ccccc|c}
 & & p_j & & & \\ \hline
 & & a_{i'j} & & & \\
 & \ldots & \uparrow & \ldots & & \\
a_{ij'} & \leftarrow & a_{ij} & \rightarrow & a_{ij''} & \alpha(y_i);[l_i,u_i] \\
 & \ldots & \downarrow & \ldots & & \\
 & & a_{i''j} & & & \\ \hline
 & & \alpha(x_j) & & & \\
 & & [l_j,u_j] & & &
\end{array}\right]
$$

Figure 2.5: *SMT-RAT* tableau

Chapter 3

Algorithms

3.1 Branch and Bound

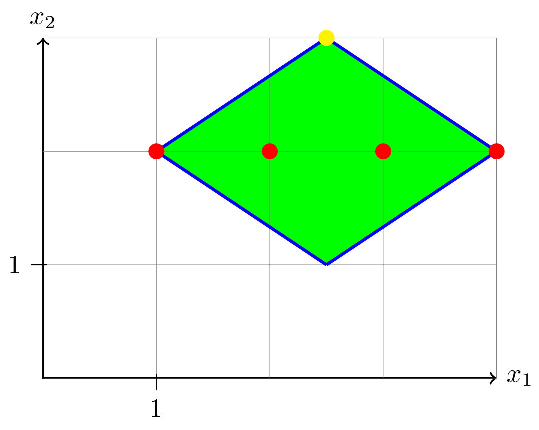
The first basic LIA technique that we consider is *branch and bound*, abbreviated by BB, which is widely used in today's LIA solvers.

3.1.1 Algorithm

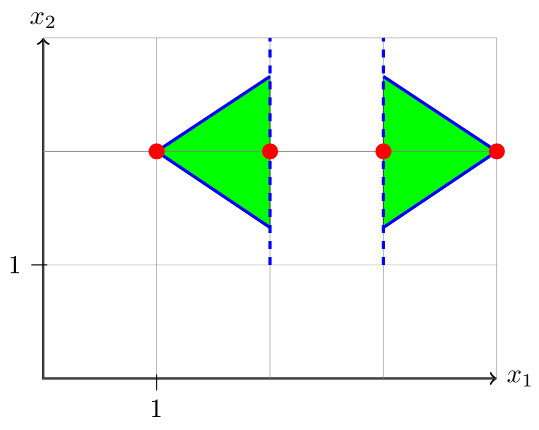
Branch and bound starts with checking via the general simplex method whether a solution for the relaxation $Rel(\phi)$ of the given system of linear inequalities, represented by ϕ, exists. If that is not the case, this naturally implies that ϕ also does not have a valid solution, for the original domains of the variables. Otherwise it can be checked whether all components of the current solution satisfy the domain requirements. Since the implementation of the general simplex method provides rational assignments, this basically means that one has to check whether the integer variables are assigned to an integer. If that is the case, ϕ has a valid solution. Otherwise there exists a variable, say x_i, such that $dom(x_i) = \mathbb{Z} \wedge \alpha(x_i) \notin \mathbb{Z}$. In order to fix this, one can recursively search for valid solutions on the one hand in the branch $\phi \wedge (x_i \leq \lfloor \alpha(x_i) \rfloor)$ and on the other hand in the branch $\phi \wedge (x_i \geq \lceil \alpha(x_i) \rceil)$, respectively $\phi \wedge (x_i - \lfloor \alpha(x_i) \rfloor \leq 0)$ and $\phi \wedge (x_i - \lceil \alpha(x_i) \rceil \geq 0)$ if we stick to our definition of QFLIA formulas.

A graphical representation of a BB run can be found below where x_1 and x_2 are integer-valued variables.

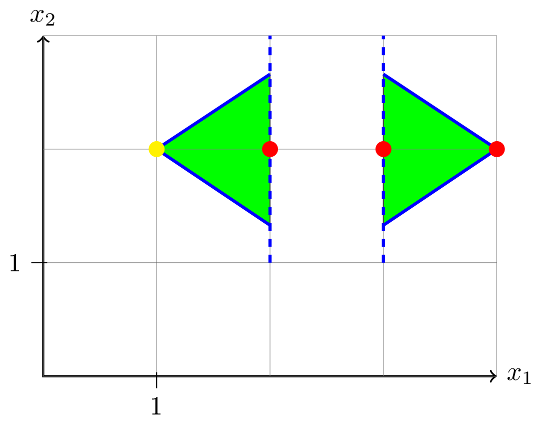
The yellow x_1-x_2 point (2.5,3) represents the solution that the general simplex method has found for the relaxation of the given problem. Since the x_1 component of the solution is not an integer a branch can be conducted.



The previous solution is excluded from the polytope without limiting the set of integer solutions. According to BB, the two branches are analyzed recursively for integer solutions.



The left branch contains an integer solution which is found by the simplex algorithm. Hence we return SAT.



3.1.2 Implementation

My embedding of the BB method into *SMT-RAT* is straightforward. If the implementation of the general simplex algorithm, called $generalSimplex()$, determines unsatisfiability, we return $UNSAT$. For the case that the simplex method finds a feasible solution, we can access it. Otherwise we check whether there exists at least one variable whose assignment violates its domain such that we would learn the corresponding BB constraint as a lemma and return UNKNOWN. Hence the SAT solver involves this constraint in the solving process and moves on with the computation. Otherwise there is no domain violation meaning that we can return SAT.

Algorithm 1 $check^{BB}_{LRAModule}(\phi)$

```
Input: A QFLIA formula $\phi$
Output: SAT, UNSAT or UNKNOWN
  if $\neg(generalSimplex())$ then
    return UNSAT
  end if
  if $\exists\ 1 \leq i \leq n : dom(x_i) = \mathbb{Z} \wedge \alpha(x_i) \notin \mathbb{Z}$ then
    Learn lemma: $(x_i - \lfloor \alpha(x_i) \rfloor \leq 0) \vee (x_i - \lceil \alpha(x_i) \rceil \geq 0)$
    return UNKNOWN
  end if
  return SAT
```

3.1.3 Example

Example 3.1.1. *The following example shows the application of BB in SMT-RAT on a benchmark instance belonging to a set of benchmarks later being used to measure the performance of the implemented algorithms. All original variables x_1,...,x_{10} in this instance have an integer domain. Assume we are given a tableau of the following form:*

$$\begin{array}{c}
\\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ \\ \\
\end{array}
\left[\begin{array}{ccccccc|c}
x_4 & x_2 & x_0 & x_8 & p_6 & p_7 & x_1 & \\
\hline
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0; [-\infty,0] \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0; [0,\infty] \\
-2 & 0 & 0 & 0 & 1 & 2 & 0 & \frac{2}{3}; [-\infty,\infty] \\
1 & 0 & 0 & 0 & 1 & -1 & 0 & -\frac{1}{3}; [-\infty,\infty] \\
1 & 0 & 3 & 3 & 1 & -1 & 3 & -\frac{1}{3}; [-\infty,0] \\
\hline
0 & 0 & 0 & 0 & 0 & 1 & 0 & \\
[-\infty,\infty] & [-\infty,\infty] & [-\infty,\infty] & [-\infty,\infty] & [-\infty,0] & [1,\infty] & [-\infty,\infty] &
\end{array}\right]$$

Let $p_1 = x_0 + x_2 + x_4$, $p_2 = x_2 + x_8$, $p_3 = 3x_3$, $p_4 = 3x_7$, $p_5 = 3x_0 + 3x_1 + 3x_7 + 3x_8$, $p_6 = x_3 + 2x_7$ and $p_7 = x_3 + x_4 - x_7$. The factor (3) e.g. of the original variable x_3 is due to the normalization that was explained in the tableau section.

We see that e.g. we have $\alpha(x_3) \notin \mathbb{Z}$ for the original variable x_3. After adding the branching constraint $(x_3 \leq 0) \vee (x_3 - 1 \geq 0)$ to the original formula the SAT solver re-assigns the Boolean skeleton of the latter and calls the LRA module which pivots on x_3 such that:

$$
\begin{array}{c}
 \\ p_1 \\ p_2 \\ p_6 \\ \frac{p_4}{3} \\ \frac{p_5}{3} \\ \\ \\
\end{array}
\left[
\begin{array}{ccccccc|c}
x_4 & x_2 & x_0 & x_8 & x_3 & p_2 & x_1 & \\
\hline
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0; [-\infty,0] \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0; [0,\infty] \\
2 & 0 & 0 & 0 & 3 & -2 & 0 & -2; [-\infty,0] \\
1 & 0 & 0 & 0 & 1 & -1 & 0 & -1; [-\infty,\infty] \\
1 & 0 & 1 & 1 & 1 & -1 & 1 & -1; [-\infty,0] \\
\hline
0 & 0 & 0 & 0 & 0 & 1 & 0 & \\
[-\infty,\infty] & [-\infty,\infty] & [-\infty,\infty] & [-\infty,\infty] & [-\infty,0] & [1,\infty] & [-\infty,\infty] &
\end{array}
\right]
$$

Since a valid solution was found in the "left" branch we return SAT.

3.1.4 Termination behavior

Termination of BB can be guaranteed when all variables x_i with $i \in \{1, ..., n\}$ and $dom(x_i) = \mathbb{Z}$ satisfy $l_{x_i} \neq -\infty$ and $u_{x_i} \neq \infty$. Otherwise there exists a variable with unbounded real solutions such that the assignments which are found by the general simplex algorithm increase respectively decrease without ever satisfying the domain requirements. Consider e.g. the QFLIA formula $\phi := (5x_1 - 5x_2 - 3 \geq 0) \wedge (5x_1 - 5x_2 - 4 \leq 0)$ where $dom(x_1) = dom(x_2) = \mathbb{Z}$.

Figure 3.1 shows an extract of the relaxed polytope of ϕ after the application of one BB step showing that the assignments of the simplex method can arbitrarily escape the branch constraints.

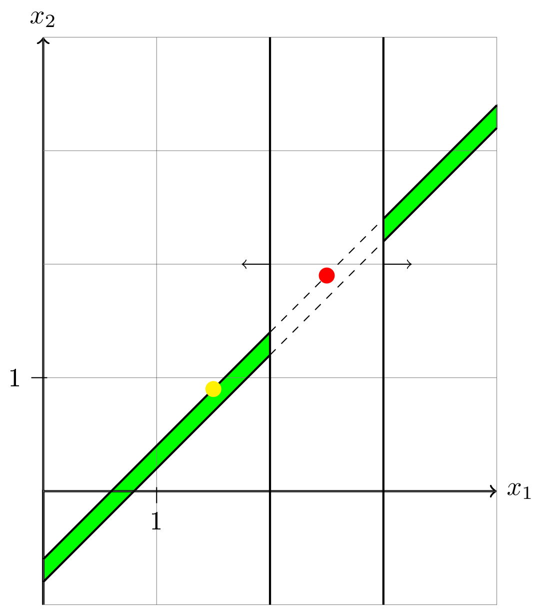


Figure 3.1: First assignment: (2.5,1.9), second assignment: (1.5,0.9)

3.1.5 Premises

A possible idea to reduce the vast number of branches that have to be considered is to add some information to the lemmas we learn and are later considered by the SAT solver. Assume we call $check_{SATModule}(\phi \wedge ((x_1 \leq 0) \vee (x_1 - 1 \geq 0)))$. In this case $c := ((x_1 \leq 0) \vee (x_1 - 1 \geq 0))$ is always considered but especially in the case of applying BB it is desirable to consider only those branching constraints that are relevant for the current state of calculation and therefore one can call $check_{SATModule}(\phi \wedge (p \rightarrow c))$ for an appropriate premise p such that c is only considered when the premise p is true. My idea for such a p is to take the conjunction of those constraints whose slack variable's assignment is at one of the variable's bounds, formally $p = \bigwedge_{\psi \in \Psi : p_\psi(\vec{\alpha})=0} \psi$ where Ψ is the set of received constraints and $\vec{\alpha}$ the vector containing the variable's assignments. This idea is depicted in Figure 3.2 .

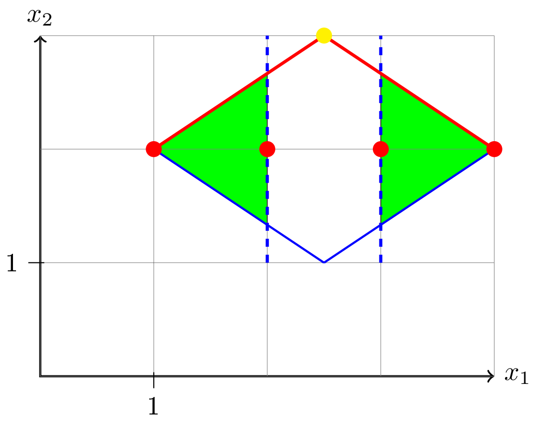


Figure 3.2: Premise construction

The yellow point, (2.5,3), denotes the current assignment which is excluded by BB and the red lines represent the constraints whose conjunction is taken as premise.

3.1.6 Branching Strategies

As a matter of fact the order in which the variables are branched on can have a significant impact on the algorithms' runtime and therefore most modern LIA solvers make use of heuristics that determine a preferably beneficial branching order. My implementation exhibits the following heuristics choosing the next variable to branch on. In the following let X_{Fix} denote the set of variables with an integer domain and a non-integer assignment:

$$X_{Fix} := \{x_i : 1 \leq i \leq n \wedge dom(x_i) = \mathbb{Z} \wedge \alpha(x_i) \notin \mathbb{Z}\}$$

In addition to that we denote for each $x_i \in X_{Fix}$ the distance between $\alpha(x_i)$ and the closest integer by

$$n_{x_i} := min(\lceil \alpha(x_i) \rceil - \alpha(x_i), \alpha(x_i) - \lfloor \alpha(x_i) \rfloor).$$

Most Infeasible Branching

Most infeasible branching chooses the variable $x_i \in X_{fix}$ to branch with n_{x_i} maximal and therefore "most infeasible". In other words we branch on the variable $x_i \in X_{fix}$ if

$$n_{x_i} = max(\{n_y \mid y \in X_{Fix}\}).$$

Most Feasible Branching

In contrary, most feasible branching chooses the variable $x_i \in X_{fix}$ to branch with n_{x_i} minimal and therefore "most feasible". In other words we branch on the variable $x_i \in X_{fix}$ if

$$n_{x_i} = min(\{n_y \mid y \in X_{Fix}\}).$$

Minimize Pivoting Effort

This heuristic aims to minimize the effort for the pivoting step that is necessary in order to satisfy the constraint which is added by BB. It therefore branches on a variable with a low number of entries respectively coefficients in its row.

We will analyze the advantages and disadvantages of these heuristics in the experimental results section.

3.2 Gomory Cuts

In the last chapter we have seen how to determine the satisfiability of a QFLIA formula ϕ by adding appropriate constraints excluding solutions from the polytope of the problem that do not satisfy the domain requirements for some variables but do not limit the set of valid solutions. Below we will consider another technique of this type working well on a lot of instances in practice. The technique is called *Gomory cuts*.

3.2.1 Derivation

The subsequent variant of Gomory cuts and its construction bases on [DDM06]. In order to deduce a Gomory cut from a given tableau the following needs to be satisfied:

$\exists x_i \in B : \alpha(x_i) \notin \mathbb{Z} \wedge dom(x_i) = \mathbb{Z} \wedge$
$\forall x_j \in N : a_{ij} \neq 0 \rightarrow (\alpha(x_j) = l_{x_j} \vee \alpha(x_j) = u_{x_j})$

According to this we need to find a basic variable x_i that is assigned to a value not included in its domain such that all non-basic variables occurring in row i are either assigned to their lower or their upper bound. Define $f_0 := \alpha(x_i) - \lfloor \alpha(x_i) \rfloor$. Because $\alpha(x_i) \notin \mathbb{Z}$ we have $0 < f_0 < 1$. We split the column indices of the non-basic variables occurring in row i into those that are assigned to their lower and those that are assigned to their upper bound:

$$J = \{j \mid \alpha(x_j) = l_{x_j} \wedge a_{ij} \neq 0\}$$
$$K = \{j \mid \alpha(x_j) = u_{x_j} \wedge a_{ij} \neq 0\}$$

Since the simplex algorithm guarantees the consistence of all occurring equations,

$$\alpha(x_i) = \sum_{x_j \in N} a_{ij}\alpha(x_j) \text{ (i)}$$

is valid for

$$x_i = \sum_{x_j \in N} a_{ij}x_j \text{ (ii)}.$$

Subtracting (i) from (ii) results in:

$$x_i - \alpha(x_i) = \sum_{x_j \in N} a_{ij}x_j - \sum_{x_j \in N} a_{ij}\alpha(x_j)$$

which is equivalent to

$$x_i - \alpha(x_i) = \sum_{j \in J} a_{ij}(x_j - l_{x_j}) - \sum_{j \in K} a_{ij}(u_{x_j} - x_j)$$

which is equivalent to

$$x_i - \lfloor\alpha(x_i)\rfloor = f_0 + \sum_{j \in J} a_{ij}(x_j - l_{x_j}) - \sum_{j \in K} a_{ij}(u_{x_j} - x_j).$$

This equation does not restrict the valid solutions. We keep in mind that $f_0 + \sum_{j \in J} a_{ij}(x_j - l_{x_j}) - \sum_{j \in K} a_{ij}(u_{x_j} - x_j)$ is desired to be an integer and move on with a case-by-case analysis. Therefore we split K and J in the following way:

$$\begin{aligned} J^+ &= \{j \in J \mid a_{ij} > 0\} \\ J^- &= \{j \in J \mid a_{ij} < 0\} \\ K^+ &= \{j \in K \mid a_{ij} > 0\} \\ K^- &= \{j \in K \mid a_{ij} < 0\} \end{aligned}$$

First Case: $\sum_{j \in J} a_{ij}(x_j - l_{x_j}) - \sum_{j \in K} a_{ij}(u_{x_j} - x_j) \geq 0$

That implies $f_0 + \sum_{j \in J} a_{ij}(x_j - l_{x_j}) - \sum_{j \in K} a_{ij}(u_{x_j} - x_j) \geq 1$ because $dom(x_i) = \mathbb{Z}$ and therefore $x_i - \lfloor\alpha(x_i)\rfloor$ must be an integer. Taking the splitting of K and J into account we can therefore conclude

$$f_0 + \underbrace{\sum_{j \in J^+} \underbrace{a_{ij}}_{\geq 0}(x_j - l_{x_j})}_{\geq \sum_{j \in J} a_{ij}(x_j - l_{x_j})} - \underbrace{\sum_{j \in K^-} \underbrace{a_{ij}}_{< 0}(u_{x_j} - x_j)}_{\leq \sum_{j \in K} a_{ij}(u_{x_j} - x_j)} \geq 1$$

which is equivalent to

$$\sum_{j \in J^+} \tfrac{a_{ij}}{1-f_0}(x_j - l_{x_j}) - \sum_{j \in K^-} \tfrac{a_{ij}}{1-f_0}(u_{x_j} - x_j) \geq 1. \text{ (iii)}$$

Second Case: $\sum_{j \in J} a_{ij}(x_j - l_{x_j}) - \sum_{j \in K} a_{ij}(u_{x_j} - x_j) < 0$

In this case the inequality

$$f_0 + \sum_{j \in J} a_{ij}(x_j - l_{x_j}) - \sum_{j \in K} a_{ij}(u_{x_j} - x_j) \leq 0$$

has to hold because $dom(x_i) = \mathbb{Z}$. After estimating the sums and transforming the inequality equivalently we receive

$$\underbrace{\sum_{j \in K^+} \underbrace{a_{ij}}_{\geq 0} (u_{x_j} - x_j)}_{\geq \sum_{j \in K} a_{ij}(u_{x_j} - x_j)} - \underbrace{\sum_{j \in J^-} \underbrace{a_{ij}}_{< 0} (x_j - l_{x_j})}_{\leq \sum_{j \in J} a_{ij}(x_j - l_{x_j})} \geq f_0$$

which is equivalent to

$$\sum_{j \in K^+} \frac{a_{ij}}{f_0}(u_{x_j} - x_j) + \sum_{j \in J^-} \frac{-a_{ij}}{f_0}(x_j - l_{x_j}) \geq 1. \quad \text{(iv)}$$

The inequalities (iii) and (iv) imply the final cut:

$$\sum_{j \in J^+} \frac{a_{ij}}{1-f_0}(x_j - l_{x_j}) + \sum_{j \in J^-} \frac{-a_{ij}}{f_0}(x_j - l_{x_j}) + \sum_{j \in K^+} \frac{a_{ij}}{f_0}(u_{x_j} - x_j) + \sum_{j \in K^-} \frac{-a_{ij}}{1-f_0}(u_{x_j} - x_j) \geq 1$$

which excludes the current solution not satisfying the integer requirements but does not limit the set of valid solutions. An example Gomory cut is shown in Figure 3.3 .

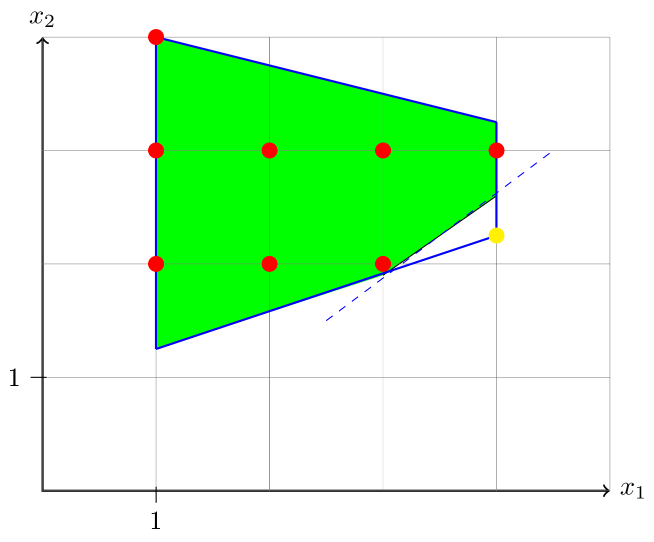
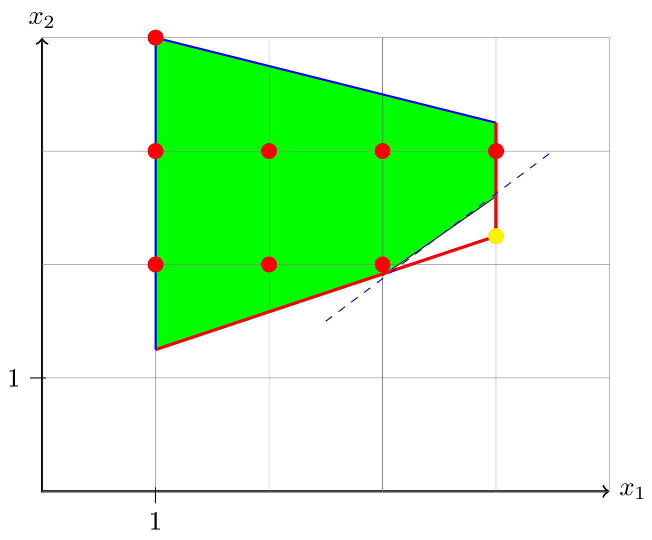


Figure 3.3: A Gomory cut is added to a set of constraints excluding the domain violating assignment

3.2.2 Premises

For the generation of premises for Gomory cuts we use the same mechanism that we have already used for the generation of premises for BB.



When calling $check_{SATModule}(\phi \wedge c)$ we want to guarantee that $\phi \equiv \phi \wedge c$ and therefore we consider constraints that do not limit the set of solutions of ϕ or in other words are tautologies. So we have to reason that $(\bigwedge_{\psi \in \Psi : p_\psi(\vec{\alpha})=0} \psi) \rightarrow g$, where g is a Gomory cut, is a tautology. If the antecedence is false the implication is true. If the antecedence is true we distinguish two cases regarding possible valid assignments for the premise. We split the constraints that we consider into the ones that are involved in the premise generation and the remaining ones, namely $\Phi = \{\psi \in \Psi : p_\psi(\vec{\alpha}) = 0\}$ and $\Phi' = \Psi \setminus \Phi$. For the first case, we assume that the assignment satisfying the antecedence also satisfies all $\psi \in \Phi'$ from which we can instantly deduce that g is also satisfied because the construction of Gomory cuts guarantees that no integer points of the considered polytope are excluded. For the second case, we assume that the assignment satisfying the antecedence does not satisfy all $\psi \in \Phi'$. Therefore we have a $\psi \in \Phi'$ that is not satisfied. The formula ψ geometrically does not cross the excluded assignment and hence assignments satisfying the antecedence and $\neg\psi$ are not in the half, regarding the subspace induced by g, of the excluded assignment containing the assignments that do not satisfy the Gomory cut. So g is also satisfied by such an assignment.

3.2.3 Algorithm

The idea of applying pure Gomory cuts is to iteratively construct and add them to the input formula until either satisfiability or unsatisfiability can be determined or we can not derive a Gomory cut anymore such that we have to return $UNKNOWN$. The following algorithm in pseudocode illustrates that:

Algorithm 2 $check^{GC}_{LRAModule}(\phi)$

```
Input: A QFLIA formula $\phi$
Output: SAT, UNSAT or UNKNOWN
  if $\neg(generalSimplex())$ then
    return UNSAT
  end if
  varViolatesDomain := false
  for $x_i \in B : dom(x_i) = \mathbb{Z} \wedge \alpha(x_i) \notin \mathbb{Z}$ do
    varViolatesDomain := true
    if $\forall x_j \in N : a_{ij} \neq 0 \rightarrow (\alpha(x_j) = l_{x_j} \vee \alpha(x_j) = u_{x_j})$ then
      - Construct the cut constraint $g$ and determine the premise $p$
      - Learn lemma: $p \rightarrow g$
    end if
  end for
  if varViolatesDomain then
    return UNKNOWN
  else
    return SAT
  end if
```

Note that the generation of the premise is necessary in this case because because we want to ensure that the added lemma is actually a tautology.

The application of pure Gomory cuts is actually not very reasonable since these can not always be constructed. But as we will see later we can benefit from complementing BB with Gomory cuts.

3.2.4 Example

The following example shows the application of pure Gomory cuts in *SMT-RAT*.

Example 3.2.1. *Let* $p_1 = x_0 - x_1 - x_2 - 2x_3 - x_5 + x_6 + x_8$, $p_2 = x_4 - x_7 + x_8 + x_9$, $p_3 = x_2 - x_3$, $p_4 = x_2 + x_4$, $p_5 = x_0 + x_3 + x_5$, $p_6 = x_1 - x_2 - x_7$, $p_7 = x_1 + x_6 - x_7 - x_8$, $p_8 = x_3 - x_6 - x_7 - x_9$, $p_9 = x_2 + x_4 + x_9$ *and* $p_{10} = x_3 + x_4 + x_8$.

$$
\begin{array}{c|cccccccccc|c|}
 & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 & p_8 & p_9 & p_{10} & \\
\hline
3x_6 & 0 & 1 & 0 & 0 & 0 & -2 & 2 & -1 & -2 & 1 & \frac{1}{3};\, [-\infty,\infty] \\
-3x_4 & 0 & -1 & 3 & -6 & 0 & -1 & 1 & 1 & 2 & 2 & -\frac{8}{3};\, [-\infty,\infty] \\
3x_0 - 3x_1 - 3x_5 - 6x_9 & 3 & -4 & 0 & 0 & 0 & -1 & 1 & 4 & 2 & 2 & \frac{2}{3};\, [0,\infty] \\
3x_3 & 0 & -1 & 0 & -3 & 0 & -1 & 1 & 1 & 2 & 2 & \frac{5}{3};\, [-\infty,\infty] \\
x_9 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 1;\, [-1,\infty] \\
6x_5 & -3 & 8 & -3 & 12 & 3 & -1 & -2 & -5 & -13 & -7 & -\frac{8}{3};\, [-\infty,\infty] \\
x_1 & 0 & -1 & 1 & -1 & 0 & 1 & 0 & 0 & 1 & 1 & 1;\, [-\infty,\infty] \\
6x_0 - 3x_3 + 3x_4 & 3 & -4 & 0 & 3 & 3 & 5 & -2 & 1 & 5 & -1 & -\frac{7}{3};\, [-\infty,0] \\
3x_7 & 0 & -2 & 0 & 0 & 0 & 1 & -1 & -1 & 1 & 1 & -\frac{2}{3};\, [-\infty,\infty] \\
4x_2 + 2x_4 + 2x_5 - 2x_9 & -1 & 2 & 1 & 6 & 1 & -1 & 0 & -1 & -5 & -1 & -3;\, [-\infty,0] \\
2x_0 & 1 & -2 & 1 & -2 & 1 & 1 & 0 & 1 & 3 & 1 & 1;\, [-\infty,\infty] \\
3x_1 - 3x_6 - 3x_7 & 0 & -2 & 3 & -3 & 0 & 4 & -1 & 2 & 4 & 1 & \frac{4}{3};\, [-1,\infty] \\
3x_2 & 0 & -1 & 3 & -3 & 0 & -1 & 1 & 1 & 2 & 2 & \frac{5}{3};\, [-\infty,\infty] \\
x_8 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 2;\, [-\infty,\infty] \\
\hline
 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 1 & \\
 & [-\infty,0] & [-\infty,1] & [0,\infty] & [-\infty,-1] & [-\infty,0] & [-\infty,0] & [0,\infty] & [1,\infty] & [0,\infty] & [1,\infty] &
\end{array}
$$

We can deduce Gomory cuts for the basic variables x_4, x_6, x_5, x_7, x_3, x_2 since they are not assigned to an integer and furthermore all occurring non-basic variables are either assigned to their upper or their lower bound:

$$\begin{aligned} g_1 &: x_4 \leq -3 \\ g_2 &: -2x_2 - x_3 - x_4 + x_8 \leq -1 \\ g_3 &: 3x_0 - 2x_1 - 4x_2 - 6x_3 + x_5 + 3x_6 - x_7 + 3x_8 \leq -12 \\ g_4 &: -x_2 - x_3 + x_8 + x_9 \leq -1 \\ g_5 &: -x_3 \leq -2 \\ g_6 &: -x_2 \leq -2 \end{aligned}$$

After adding these to the tableau we obtain:

$$\begin{array}{c} \\ x_6 \\ -2x_4 \\ 2x_0-2x_1-2x_5-4x_9 \\ 2x_3 \\ x_9 \\ -2x_5 \\ x_1 \\ -2x_0+x_3-x_4 \\ -2x_7 \\ 4x_2+2x_4+2x_5-2x_9 \\ 2x_0 \\ -2x_1+2x_6+2x_7 \\ 2x_2 \\ x_8 \\ -p_{g_2} \\ -p_{g_3} \\ 2p_7 \\ \\ \\ \end{array} \left[\begin{array}{cccccccccc|c} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & -p_{g_4} & p_8 & p_9 & p_{10} & \\ \hline 0 & 1 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 & 1;\,[-\infty,\infty] \\ 0 & 0 & 2 & -4 & 0 & 0 & 1 & 0 & 1 & 1 & -3;\,[-\infty,-3] \\ 2 & -2 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 1 & 1;\,[0,\infty] \\ 0 & 0 & 0 & -2 & 0 & 0 & 1 & 0 & 1 & 1 & 2;\,[2,\infty] \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 1;\,[-1,\infty] \\ 1 & -2 & 1 & -4 & -1 & 1 & 1 & 1 & 4 & 2 & -3;\,[-\infty,\infty] \\ 0 & -1 & 1 & -1 & 0 & 1 & 0 & 0 & 1 & 1 & 1;\,[-\infty,\infty] \\ -1 & 2 & 0 & -1 & -1 & -1 & 1 & -1 & -2 & 0 & -3;\,[-\infty,0] \\ 0 & 2 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & -1 & -1;\,[-\infty,\infty] \\ -1 & 2 & 1 & 6 & 1 & -1 & 0 & -1 & -5 & -1 & -3;\,[-\infty,0] \\ 1 & -2 & 1 & -2 & 1 & 1 & 0 & 1 & 3 & 1 & 1;\,[-\infty,\infty] \\ 0 & 2 & -2 & 2 & 0 & -2 & 1 & -2 & -3 & -1 & 1;\,[-1,\infty] \\ 0 & 0 & 2 & -2 & 0 & 0 & 1 & 0 & 1 & 1 & 2;\,[2,\infty] \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 2;\,[-\infty,\infty] \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1;\,[1,\infty] \\ -1 & -4 & 2 & -8 & -2 & 1 & 2 & 2 & 8 & 4 & -12;\,[-\infty,-12] \\ 0 & 2 & 0 & 0 & 0 & 2 & 3 & -2 & -1 & -1 & 1;\,[0,\infty] \\ \hline 0 & 1 & 0 & -1 & 0 & 0 & 1 & 1 & 0 & 1 & \\ {[-\infty,0]} & {[-\infty,1]} & {[0,\infty]} & {[-\infty,-1]} & {[-\infty,0]} & {[-\infty,0]} & {[1,\infty]} & {[1,\infty]} & {[0,\infty]} & {[1,\infty]} & \end{array}\right]$$

3.2.5 Termination behavior

Applying pure Gomory cuts as explained above is incomplete. On the one hand Gomory cuts are not always constructible and on the other hand the approaching of assignments towards an integer evoked by the previous cuts can be infinitesimal small such that the assignment never reaches this integer. We will still see later how the global strategy combining BB and Gomory cuts leads to an improvement.

3.3 Cuts from Proofs

The two techniques that we considered before, BB and Gomory cuts, both add in the case of a domain violation constraints to the set of existing constraints such that the current assignment is excluded but the set of valid solutions stays untouched. The *cuts from proofs* method presented in [DDA09] aims to exclude whole subspaces of the polytope not containing integer points.

3.3.1 Theoretical Preliminaries

In order to make us familiar with this method we will first have a look at some necessary theory.

Definition 3.3.1 (Linear Diophantine Equations [DDA09]). *A linear equation of the form* $\sum_{i=1}^{n} a_i x_i = c$ *is* diophantine *if all coefficients* a_i *are integers and* c *is an integer.*

As already mentioned, integer arithmetic provides better performance than fractional arithmetic. Therefore the *SMT-RAT* tableau works on integers by converting LIA instances exhibiting fractions. The rows of the tableau represent linear diophantine equations.

We proceed with the following lemma [NW88] giving us a criterion to judge the solvability of a linear diophantine equation:

Lemma 3.3.1. *A linear diophantine equation of the form* $\sum_{i=1}^{n} a_i x_i = c$ *has an (integer) solution iff* c *is an integral multiple of the greatest common divisor* $gcd(a_1,...,a_n)$*.*

Lemma 3.3.1 implies the following corollary [DDA09]:

Corollary 3.3.2. *Let* E *be a plane defined by* $\sum_{i=1}^{n} a_i x_i = c$ *with no integer solutions and let* $g = gcd(a_1,...,a_n)$*. Then the two closest planes parallel to and on either side of* E *containing integer points are* $\lfloor E \rfloor$ *and* $\lceil E \rceil$*, given by* $\sum_{i=1}^{n} \frac{a_i}{g} x_i = \lfloor \frac{c}{g} \rfloor$ *and* $\sum_{i=1}^{n} \frac{a_i}{g} x_i = \lceil \frac{c}{g} \rceil$*, respectively.*

Corollary 3.3.2 is implied by the previous lemma because $\lfloor \frac{c}{g} \rfloor$ and $\lceil \frac{c}{g} \rceil$ are trivially multiples of $gcd(\frac{a_1}{g},...,\frac{a_n}{g}) = 1$ and therefore $\lfloor E \rfloor$ and $\lceil E \rceil$ have according to Lemma 3.3.1 integer solutions. They are the closest planes with that characteristic because every other right side of such a linear diophantine equation would not be as tight as $\lfloor \frac{c}{g} \rfloor$ respectively $\lceil \frac{c}{g} \rceil$. The cuts from proofs algorithm uses such planes to exclude hyperplanes E containing no integer solutions just as $\lfloor E \rfloor$ and $\lceil E \rceil$ to ensure that the set of valid solutions stays untouched.

Another concept that we need is the one of *Hermite matrices.* From the slightly distinguishing definitions of the latter we use:

Definition 3.3.2 (Hermite Normal Form [DDA09]). *An* $m \times m$ *integer matrix* H *is said to be in* Hermite normal form (HNF) *if (i)* H *is lower triangular, (ii)* $h_{ii} > 0$ *for* $0 \leq i < m$*, and (iii)* $h_{ij} \leq 0$ *and* $\mid h_{ij} \mid < h_{ii}$ *for* $i > j$*.*

3.3.2 Hermite Normal Form

As we will see, the cuts from proofs method uses the Hermite normal form. I use the following algorithm for the calculation of the latter because it makes use of operations that can be executed cheaply in the tableau of *SMT-RAT*. Given a $m \times n$ matrix A of full row rank, meaning that the maximal number of linearly independent rows of A is equal to the number of rows of A, the unique Hermite normal form A_{HNF} of A is established by the following operations:

- swapping columns
- multiplying the entries of a column by -1
- adding an integer multiple of one column to another one

After step k we receive a matrix of the form:

$$A_{HNF_k} = \begin{pmatrix} h_{1,1} & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \vdots \\ h_{k-1,1} & \dots & \ddots & 0 & \vdots & \ddots & \vdots \\ h_{k,1} & \dots & \dots & h_{k,k} & 0 & \dots & 0 \\ & & & & & & \\ & A_k & & & & B_k & \\ & & & & & & \end{pmatrix}$$

where the matrix

$$\begin{pmatrix} h_{1,1} & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ h_{k-1,1} & \dots & \ddots & 0 \\ h_{k,1} & \dots & \dots & h_{k,k} \end{pmatrix}$$

is a matrix in Hermite normal form with k rows and columns. We obtain $A_{HNF_{k+1}}$ by eliminating all but one entry in the first row of B_k. First multiply the columns of all negative entries of the first row of B_k by -1 such that all those entries are positive. Then apply the Euclidean algorithm to the entries of the first row of B_k by adding the columns correspondingly such that afterwards only one positive entry, say b, is left in this row which, by swapping columns, is located on the main diagonal of B_k. We might have to normalize row $(k+1)$ of A_{HNF_k} by subtracting suitable multiples of b such that the other entries in row $(k+1)$ are not positive and their absolute value is smaller than b according to the definition of the Hermite normal form. This procedure is then repeated until $k = m$.

Example 3.3.1. *Assume that:*

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 5 & 2 & -10 \end{pmatrix}$$

In the first iteration we just have to swap the first two columns since the first row only contains one element:

$$A_{HNF_1} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 5 & -10 \end{pmatrix}$$

After inverting (multiplying with -1) the third column, the Euclidean algorithm is applied to 5 and 10 by adding the corresponding columns appropriately. After normalization we obtain:

$$A_{HNF_2} = \begin{pmatrix} 1 & 0 & 0 \\ -3 & 5 & 0 \end{pmatrix}$$

Excluding the zero column we obtain:

$$A_{HNF} = \begin{pmatrix} 1 & 0 \\ -3 & 5 \end{pmatrix}$$

The following lemma from [NW88] creates a connection between the *Hermite normal form H* of a matrix A and the calculation of integer solutions for a given system of linear equalities $A\vec{x} = b$.

Lemma 3.3.3. *Proof of Unsatisfiability*
The system $A\vec{x} = b$ has an integer solution if and only if $A_{HNF}^{-1}b \in \mathbb{Z}^m$. If $A\vec{x} = b$ has no integer solutions, there exists a row, say i, of the matrix $T = A_{HNF}^{-1}A$ such that the corresponding entry $\frac{n_i}{d_i}$ of $A_{HNF}^{-1}b$ is not an integer. We call the linear diophantine equation $\sum_{j=1}^{n}(d_i t_{i,j} x_j) - n_i = 0$, where the $t_{i,1}$, ..., $t_{i,n}$ are the entries of the i-th row of T, with no integer solutions a proof of unsatisfiability *of $A\vec{x} = b$.*

In order to exploit this lemma we need further considerations since it argues about a system $A\vec{x} = b$ of linear equalities but we consider systems of linear inequalities. The following definition helps to close this gap.

Definition 3.3.3 (Defining Constraint)**.** *A constraint of the form $\sum_{j=1}^{n}(a_j x_j) - z_0 \leq 0$ occurring in a system given by the QFLIA formula ϕ is a* defining constraint *of ϕ if $\sum_{j=1}^{n}(a_j \alpha(x_j)) - z_0 = 0$.*

3.3.3 Algorithm

Taking the aforesaid into account, we use the following algorithm: Determine in case of a domain violation the defining constraints of the given system of linear inequalities for the current assignment. Then check, using Lemma 3.3.3., whether for those a proof of unsatisfiability c can be constructed by using the Hermite normal form of the defining constraints. If so, the defining constraints have no integer solutions such that we can exclude this subspace of the polytope not containing integer solutions by branching on the two closest hyperplanes of c according to Corollary 3.3.2 containing integer points. Otherwise use BB. Furthermore choose a β with $\beta \geq n \cdot |a_{max}|$ where a_{max} is the coefficient of the given system with the highest absolute value. The value β is needed to ensure termination. Hence, like BB and Gomory cuts, the cuts from proofs algorithm excludes invalid assignments, even subspaces, but does not restrict the set of valid solutions.

Algorithm 3 $check^{CFP}_{LRAModule}(\phi)$

```
Input: A QFLIA formula $\phi$
Output: SAT, UNSAT or UNKNOWN
  if $\neg(generalSimplex())$ then
    return UNSAT
  end if
  if $\exists\, 1 \leq i \leq n : dom(x_i) = \mathbb{Z} \wedge \alpha(x_i) \notin \mathbb{Z}$ then
    - Determine the defining constraints $A_{DC}$
    - Calculate the Hermite normal form $A_{HNF}$ of $A_{DC}$
    - Invert $A_{HNF}$
    if $A^{-1}_{HNF} \cdot b \notin \mathbb{Z}^m$ then
      - Determine a proof of unsatisfiability $\sum_{i=1}^{l}(a_i x_i) - c_i = 0$ (Lemma 3.3.3)
      if $\neg\exists\, 1 \leq i \leq n :\ a_i > \beta \cdot gcd(a_1,...,a_n)$ then
        - Learn lemma: $(\sum_{i=1}^{l}(\frac{a_i}{g}x_i) - \lfloor\frac{c_i}{g}\rfloor \leq 0) \vee (\sum_{i=1}^{l}(\frac{a_i}{g}x_i) - \lceil\frac{c_i}{g}\rceil \geq 0)$
        return UNKNOWN
      end if
    end if
    - Learn lemma: $(x_i - \lfloor\alpha(x_i)\rfloor \leq 0) \vee (x_i - \lceil\alpha(x_i)\rceil \geq 0)$
    return UNKNOWN
  end if
  return SAT
```

This algorithm is complete which is ensured by $\exists\, 1 \leq i \leq n :\ a_i > \beta \cdot gcd(a_1,...,a_n)$. The proof for that is given in [DDA09].

3.3.4 Example

Example 3.3.2. *Assume we are given the following defining constraints that occurred in the solving process of an instance:*

$$\begin{aligned} x_2 &\geq 1 \\ 5x_1 + 2x_2 - 10x_3 &\geq 0 \end{aligned}$$

The assignments may be $\alpha(x_1) = -\frac{2}{5}$, $\alpha(x_2) = 1$, $\alpha(x_3) = 0$ and the matrix A of defining constraints is given by:

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 5 & 2 & -10 \end{pmatrix}$$

We have already seen how the HNF of this matrix is determined which is given by:

$$A_{HNF} = \begin{pmatrix} 1 & 0 \\ -3 & 5 \end{pmatrix}$$

Inverting A_{HNF} leads to:

$$A_{HNF}^{-1} = \begin{pmatrix} 1 & 0 \\ \frac{3}{5} & \frac{1}{5} \end{pmatrix}$$

We can now check whether a proof of unsatisfiability exists by testing whether $A_{HNF}^{-1}b \notin \mathbb{Z}^2$:

$$A_{HNF}^{-1} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{3}{5} \end{pmatrix} \notin \mathbb{Z}^2$$

According to the algorithm, we can deduce a proof of unsatisfiability in the following way:

$$A_{HNF}^{-1} \cdot A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & -2 \end{pmatrix}$$

Using the second row of $A_{HNF}^{-1} \cdot A$, since the second component of $A_{HNF}^{-1} \cdot b$ is fractional, we create the following proof of unsatisfiability containing no integer points:

$$5x_1 + 5x_2 - 10x_3 = 3$$

Using Corollary 3.3.2. we obtain the two closest planes $\lfloor E \rfloor$ and $\lceil E \rceil$ containing integer points which can be used to branch according to:

$$x_1 + x_2 - 2x_3 \leq 0 \vee x_1 + x_2 - 2x_3 \geq 1$$

3.3.5 Implementation Details

We will briefly discuss some of the technical details that were necessary in order to efficiently implement the cuts from proofs method.

Firstly, since the swapping of two columns during the calculation of the Hermite normal form would have involved to fix the neighbor links for many entries, I decided to only "swap" the columns virtually by introducing a permutation $\sigma(x)$ with $x \in \{1,...,n\}$ such that for every column index $i \in \{1,...,n\}$, $\sigma(i)$ is the actual column index of this column in A_{HNF}.

As we have seen a matrix A that is in Hermite normal form has lower triangular form which makes it quite easy to determine A^{-1} by substituting backwards beginning in the first row only containing one entry.

Chapter 4

Global Strategy

As we have seen neither the BB nor the Gomory cuts method terminate for every problem instance. It is therefore reasonable to think of a possible strategy minimizing the possibility of failing to terminate and simultaneously solving given instances efficiently. Most modern LIA solvers combine BB with some cutting method. The *SMT-RAT* implementation provides a loop detection checking whether a probable circle of maximal three, an appropriate value we have chosen, variables exists such that the assignments of the involved variables either rise or fall permanently but do not reach an integer. The term 'probable' is used in this context because it is of course possible that the variables occurring in the circle reach an integer after a big amount of iterations or the circle has a length which is bigger than the value one chooses for the control length, 3 in our case. The loop detection labels a recurring circle of variables as a loop if it occurs fifty times, which is of course also adjustable, representing a good trade-off between the solvers' performance and the accuracy of the loop detection. If the loop detection detects a (probable) loop, my implemented strategy tries to derive Gomory cuts such that some of the instances, (probably) looping with the pure BB approach, can be solved. As a matter of fact there are also instances where the loop detection misleadingly detects a loop and therefore after a big amount of iterations BB would have terminated. If in this case Gomory cuts do not terminate we create non-terminating runs that would have terminated by using the pure BB approach. But, as we will see in the experimental results section, the advantages of this strategy clearly outweigh the disadvantages. We will also see why the cuts from proofs technique is not part of the global strategy, even if it guarantees termination.

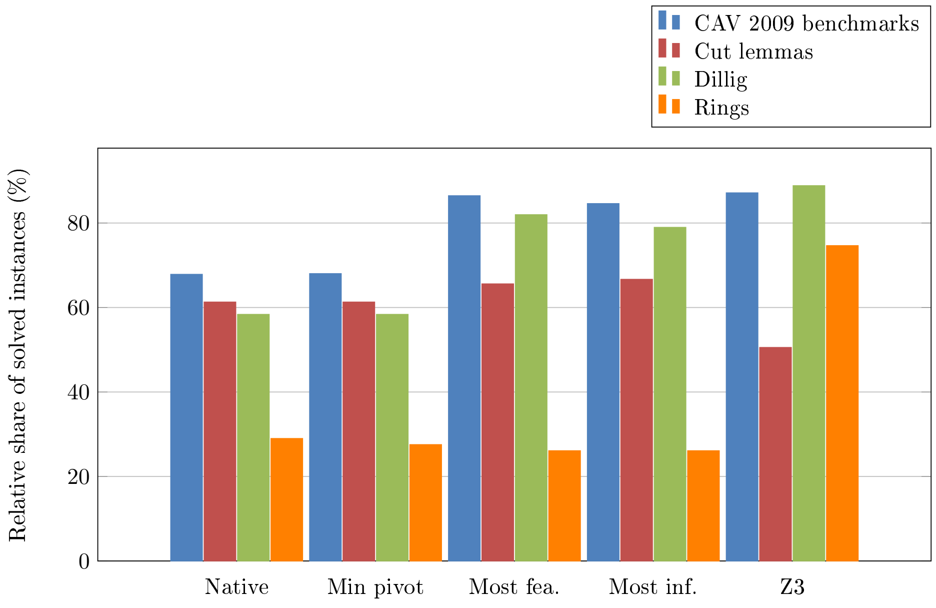
Chapter 5

Experimental Results

After the theoretical discussion of the different techniques, this chapter focuses on judging them by measuring their runtime on the QFLIA benchmark set of the SMT competition *SMT-COMP 2014*. Detailed information about the competition and the used benchmarks is available at [SMT] and [BST10]. We will also compare the results of the most promising strategies with those of Microsoft's well-known SMT solver *Z3*. The comparison of the strategies focuses on those benchmarks where the effect of my implemented techniques is high and the effect of other factors is low. On some of the other benchmarks, a significant part of the computation takes place for example in the SAT solver. Apart from that all instances from the vast amount of benchmark sets passed through error-free. All of the benchmarks were executed on blades running under Debian and each exhibiting 192 GB of memory and four 12-core *AMD Opteron 6172* processors each providing a frequency of 2.1 GHz. Every benchmark instance was given one core, a computation time of at most 200 seconds and a memory usage of at most 4 GB.

5.1 Branching Strategies

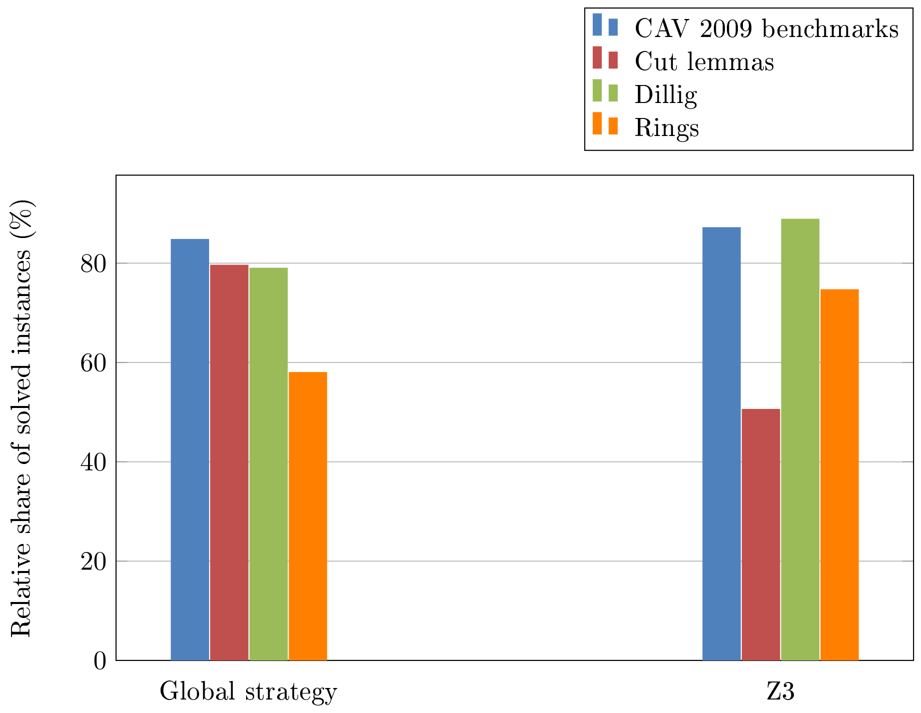
We have seen that during the execution of BB the order in which the variables are branched is nondeterministic. As a matter of fact the benchmark runs have shown that the use of the previously introduced branching heuristics has a significant impact on the performance of BB.



The results for the "CAV 2009 benchmarks" show the trend that the branch heuristics most infeasible and most feasible are most promising while min pivot generates a decent progress on these compared to branching natively. This observation is confirmed by the next two benchmarks. Compared to *Z3*, all heuristics perform rather poor on the "Rings" benchmark. Interestingly, the heuristic minimizing the pivoting effort performs relatively well on these instances. Having a closer look at the latter it gets obvious that they have relatively huge coefficients, making the elementary operations that are necessary during pivotization more expensive and suggesting that this heuristic which minimizes those elementary operations is a considerable option for instances of this kind. It is still desirable to minimize the gap to *Z3* on the "Rings" benchmark and comparable instances. We will see in the next section that the global strategy succeeds on that.

5.2 Global Strategy

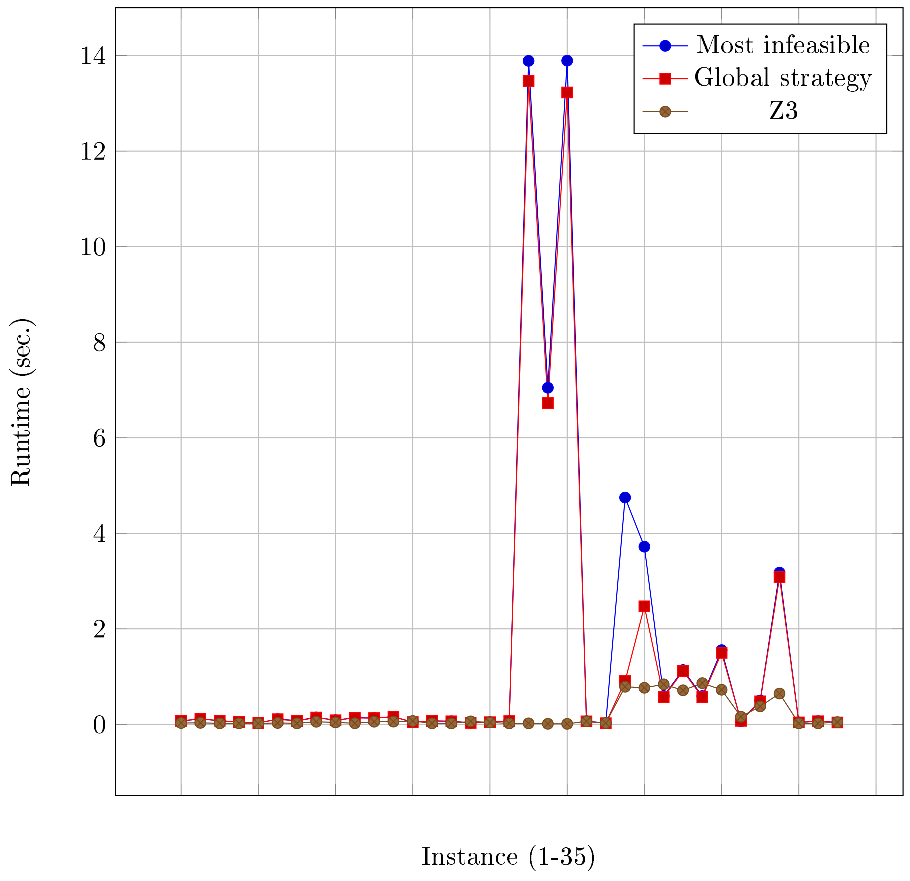
The last section suggests that the use of the branching heuristics most feasible and most infeasible provides good results but also that there are still instances for which they either do not terminate in the given time of 200 seconds respectively exceed the given memory of 4 GB or do not terminate at all. The use of the global strategy decreases the number of instances falling into one of the latter categories.



The global strategy increases the percentage of solved instances especially on the "Cut lemmas" where it even performs better than *Z3* and the "Rings" benchmark closing the gap that we detected previously. To complement BB with applying Gomory cuts on probably looping instances therefore significantly widens the range of instances that we can solve.

5.3 Runtime Analysis

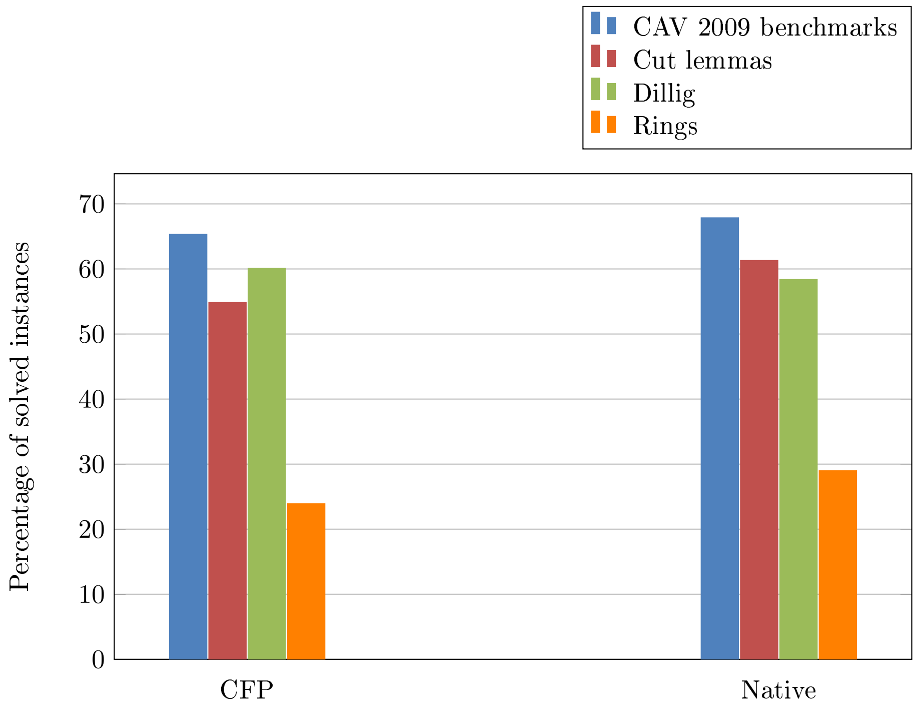
Although the relative share of solved instances in the given time and memory bound is a good indicator for the success of the corresponding strategy, it may not completely reveal differences in the efficiency of the implementations. Therefore we proceed with a runtime analysis for a benchmark set called *calypto* including 37 instances from which we consider those 35 for which all strategies/-solvers terminated. The results that we deduce from that are representative for the runtime behavior on the other considered benchmarks.



The diagram confirms the tendency that, except some outliers, the global strategy just as BB with most infeasible branching can keep up with *Z3* on the considered terminating instances. Furthermore we see that those outliers for *Z3* do not really occur which is probably due to the fact that it is highly optimized and of course follows an even more advanced strategy.
In addition to that it is interesting to see that the global strategy's runtime is significantly lower on two instances compared to BB with most infeasible branching. One would actually expect that these runtimes are either approximately the same or BB does not terminate while the global strategy does because the latter tries to derive Gomory cuts when the loop detection detects a loop in the branching process. This is due to the fact that the loop detection follows a trade-off regarding the length of a possible loop and a good but also not to exuberant detection. As these two instances show, a misleading detection of a loop is not necessarily harmful since Gomory cuts can even accelerate the solving process. It is of course also possible that no Gomory cuts could have been derived in this case and we therefore had to return *UNKNOWN* while BB would have terminated. The test runs show that this case is very rare and the chosen balance of the global strategy is appropriate.

5.4 Cuts from Proofs

Below we see a comparison of the cuts from proofs technique using the native heuristic for BB compared to pure BB with the native heuristic. Hence we get an impression of the difference that the cuts from proofs technique makes:



Taking a closer look at all results we can assert that there is a small set of instances where the CFP method terminates and native BB does not, regarding the time and memory requirements, but as we can see in the diagram this difference is not as big as the one we achieved by combining BB and Gomory cuts. Since there are also some instances where native BB terminates and the CFP method does not, especially on instances with huge coefficients and big dimensions, making the calculation of the Hermite normal form expensive, we rely on the global strategy as given above. Considering the solving process of cuts from proofs in *SMT-RAT*, I detected that the defining constraints frequently are those where the corresponding component of the right side b is zero such that $A_{HNF}^{-1}b \notin \mathbb{Z}^m$, the main condition for the derivation of a cut from proof, is hard to satisfy. The "Dillig" benchmark set is originally from the authors of [DDA09] and the "Rings" benchmark was not considered here because it does not reveal new insights.

Chapter 6

Conclusion

The aim of this thesis was to evaluate the three presented techniques and their synthesis to a global strategy theoretically just as to embed them efficiently into *SMT-RAT*.

6.1 Summary

After having established the theoretical foundations for this thesis, we firstly considered BB which is widely applied in today's LIA solvers. We have seen that it is a suitable technique for quite a lot of instances but is limited by incompleteness. Then we moved on with Gomory cuts which we used to supplement BB in order to derive a competitive strategy. Subsequently the quite recent cuts from proofs method closed the algorithms' chapter.
The experimental results evaluated the presented algorithms and suggested that the global strategy performs good against *Z3* but of course also that there is potential for further adjustments. The cuts from proofs technique does not perform as well as the global strategy.

6.2 Future Work

6.2.1 Avoiding unnecessary branches

One common weakness of BB is the fact that during the execution of the algorithm often a lot of branches are considered that either do not contribute to find a solution at all or not as fast as another order of branching would have done. One could try to tackle this by introducing a heuristic function evaluating the success of the branching process by saving the progress that branching on a certain variable generated in the recent iterations indicated, e.g., by the number of variables satisfying their domain.

6.2.2 Advancing the global strategy

The current global strategy applies Gomory cuts only when BB seems to loop. We can eventually benefit from embedding the latter even more into the global strategy. Such an adjustment could involve to already try to derive Gomory cuts when BB struggles to find a solution which could be indicated by similar ideas that were suggested in the last subsection.

6.2.3 Order of applying Gomory cuts

Until now Gomory cuts are added in a BFS fashion meaning that for a current tableau state all possible cuts are derived and added to the tableau. Experimental testing suggested that this is superior compared to adding cuts for one variable until it satisfies its domain since it can take a long time until the assignment converges against an integer. Still one could benefit of other heuristics in determining the variable for which a cut is derived. Using similar heuristics as were already used for BB is possibly promising.

6.3 Conclusion

Checking the satisfiability of linear inequalities over integers has its historical roots in the middle of the last century and is still an open field of research as more and more sophisticated algorithms enable an efficient satisfiability check in various application domains. A clever combination of BB with cutting techniques already provides good results and it will be interesting to track the progress of this and other approaches.

Bibliography

[BST10] Clark Barrett, Aaron Stump, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2010.

[CJW07] Richard Cottle, Ellis Johnson, and Roger Wets. George B. Dantzig (1914–2005). *Notices of the AMS*, 54(3):344–362, 2007.

[CLJÁ12] Florian Corzilius, Ulrich Loup, Sebastian Junges, and Erika Ábrahám. SMT-RAT: An SMT-compliant nonlinear real arithmetic toolbox. In *Theory and Applications of Satisfiability Testing(SAT)*, LNCS 7317, pages 442–448. Springer, 2012.

[DDA09] Isil Dillig, Thomas Dillig, and Alex Aiken. Cuts from proofs: A complete and practical technique for solving linear inequalities over integers. In *Proc. of Computer Aided Verification (CAV)*, LNCS 5643, pages 233–247. Springer, 2009.

[DDM06] Bruno Dutertre and Leonardo De Moura. Integrating simplex with DPLL (t). *Computer Science Laboratory, SRI International, Tech. Rep. SRI-CSL-06-01*, 2006.

[DE73] George B. Dantzig and Curtis Eaves. Fourier-Motzkin elimination and its dual. *Journal of Combinatorial Theory, Series A*, 14(3):288–297, 1973.

[JLCÁ13] Sebastian Junges, Ulrich Loup, Florian Corzilius, and Erika Ábrahám. On Gröbner bases in the context of satisfiability-modulo-theories solving over the real numbers. In Traian Muntean, Dimitrios Poulakis, and Robert Rolland, editors, *Algebraic Informatics*, volume 8080 of *Lecture Notes in Computer Science*, pages 186–198. Springer Berlin Heidelberg, 2013.

[KS08] Daniel Kroening and Ofer Strichman. *Decision procedures*, volume 5. Springer, 2008.

[Min] Mia Minnes. Mixed and integer linear programming using automata techniques. *www.math.cornell.edu/˜minnes/Automata/AutDec.pdf.*

[NW88] George L. Nemhauser and Laurence A. Wolsey. *Integer and combinatorial optimization*, volume 18. Wiley New York, 1988.

[Pug91] William Pugh. The Omega test: A fast and practical integer programming algorithm for dependence analysis. In *Proceedings of the*

1991 ACM/IEEE Conference on Supercomputing, pages 4–13. ACM, 1991.

[Sch98] Alexander Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1998.

[SMT] SMT-COMP 2014. `http://smtcomp.sourceforge.net/2014/`.