Making Rigorous Linear Programming Practical for Program Analysis

Tengbin Wang
College of Computer, National University of Defense Technology, Changsha, China

Liqian Chen[1]
College of Computer, National University of Defense Technology, Changsha, China

Taoqing Chen
State Key Laboratory of High Performance Computing, College of Computer, National University of Defense Technology, Changsha, China

Guangsheng Fan
State Key Laboratory of High Performance Computing, College of Computer, National University of Defense Technology, Changsha, China

Ji Wang[1]
State Key Laboratory of High Performance Computing, College of Computer, National University of Defense Technology, Changsha, China

Abstract

Linear programming is a key technique for analysis and verification of numerical properties in programs, neural networks, etc. In particular, in program analysis based on abstract interpretation, many numerical abstract domains (such as Template Constraint Matrix, constraint-only polyhedra, etc.) are designed on top of linear programming. However, most state-of-the-art linear programming solvers use floating-point arithmetic in their implementations, leading to an approximate result that may be unsound. On the other hand, the solvers implemented using exact arithmetic are too costly. To this end, this paper focuses on advancing rigorous linear programming techniques based on floating-point arithmetic for building sound and efficient program analysis. Particularly, as a supplement to existing techniques, we present a novel rigorous linear programming technique based on Fourier-Mozkin elimination. On this basis, we implement a tool, namely, RlpSolver, combining our technique with existing techniques to lift effectiveness of rigorous linear programming in the scene of analysis and verification. Experimental results show that our technique is complementary to existing techniques, and their combination (RlpSolver) can achieve a better trade-off between cost and precision via heuristic rules.

2012 ACM Subject Classification Theory of computation → Linear programming; Software and its engineering → Formal methods

Keywords and phrases Linear programming, rigorous linear programming, abstract interpretation, program analysis, Fourier-Mozkin elimination

Digital Object Identifier 10.4230/LIPIcs.CP.2021.22

Acknowledgements This work is supported by the National Key R&D Program of China (No. 2017YFB1001802), and the National Natural Science Foundation of China (Nos. 61872445, 62032024).

1 Introduction

Linear programming [23] is increasingly widespread in the field of analysis and verification, such as program analysis, neural network verification, etc. Up to now, there have emerged a

[1] Corresponding authors

© Tengbin Wang, Liqian Chen, Taoqing Chen, Guangsheng Fan and Ji Wang;
licensed under Creative Commons License CC-BY 4.0

wide variety of high-efficiency and scalable linear programming solvers. One common trait of these state-of-the-art solvers is that they are based on floating-point arithmetic and can only produce an approximate solution which may be far away from the optimal solution or even outside the feasible domain. On the other hand, the solvers based on exact arithmetic can get exact optimal solutions, but are time-consuming and have poor scalability in practice.

In this paper, we particularly consider the scene of applying linear programming in program analysis based on numerical abstract interpretation [6], in the context of which it is important to guarantee the soundness of the analysis. Linear programming is one of the basic operations of many numerical abstract domains, such as Template Constraint Matrix [21], constraint-only polyhedra [7], etc. To achieve high efficiency while guaranteeing soundness, this paper considers the scene of using low-cost rigorous linear programming techniques based on floating-point arithmetic to implement numerical abstract domains.

Rigorous linear programming has received much attention in the last two decades in the field of mathematics. In 2003, Neumaier and Shcherbina [19] propose a method that computes safe bounds of the objective function of the primal problem by solving the dual problem using floating-point linear programming. Jansson [11] proposes another technique which focuses on linear programming problem with uncertain input data and infinite bounds of decision variables. Although the above techniques produce general solution for rigorous linear programming problems, they may provide too conservative results for some linear programming problems in practice, especially when variables involve infinite bounds or problems are ill-conditioned. Hence, they may cause much precision loss when being used to implement program analysis.

Our approach. To make rigorous linear programming more practical for program analysis, we propose a novel rigorous linear programming technique based on Fourier-Mozkin elimination and interval arithmetic. This new technique is complementary to existing rigorous linear programming techniques and their combination via some heuristic rules can achieve a better trade-off between cost and precision than the identical ones.

The main contributions of this paper are as follows:

- We introduce a new rigorous linear programming technique based on Fourier-Mozkin elimination and interval arithmetic.
- We conduct an experimental evaluation of the usefulness and effectiveness of our technique for solving linear programming problems. The results show the availability of our technique.
- We develop a tool called RlpSolver that wraps existing rigorous linear programming techniques together, and provide heuristic rules to help choosing proper solvers in different cases.

For the sake of brevity, we use the following abbreviations throughout the paper:

- LP: linear programming.
- RLP: rigorous linear programming.
- FME: Fourier-Mozkin elimination.

The rest of the paper is organized as follows. In Section 2, we review some preliminaries of FME and RLP. Section 3 presents an overview of our work via a motivating example. In Section 4, we illustrate our RLP approach based on FME in detail. Section 5 presents the framework and heuristic rules of our tool named RlpSolver. Section 6 presents experimental results. Section 7 discusses some related work. Finally, conclusions and future work are given in Section 8.

2 Preliminaries

In this section, we first review some basic concepts and notations of linear system with interval coefficients. After that, we illustrate two existing representative RLP techniques, i.e., SafeBound [19] and ErrorBound [11].

2.1 Interval Linear System

Let $\underline{A}, \overline{A} \in \mathbb{R}^{m\times n}$ be real matrices with $\underline{A} \leq \overline{A}$. We define the following set of real matrices:

$$\mathbf{A} = \left[\underline{A}, \overline{A}\right] = \left\{A \in \mathbb{R}^{m\times n} : \underline{A} \leq A \leq \overline{A}\right\}$$

We call $\mathbf{A}$ an interval matrix, whose lower and upper bounds are $\underline{A}$ and $\overline{A}$ respectively. An interval vector is a special interval matrix with one column, i.e., $\mathbf{d} = \left\{d \in \mathbb{R}^m : \underline{d} \leq d \leq \overline{d}\right\}$. Assume that $\mathbf{A} \in \mathbb{IR}^{m\times n}$ (where $\mathbb{IR}$ denotes the set of intervals whose bounds are real numbers) is a interval matrix whose dimension is $m \times n$ and $b \in \mathbb{R}^m$ is a m-dimension real vector. We define

$$\mathbf{A}x \leq b \tag{1}$$

as an interval linear inequality system, representing a set of linear inequality systems constituted by all $Ax \leq b$ where $A \in \mathbf{A}$.

2.2 Linearization Technique

Now, we describe a method (called Orthant Reduction in this paper) to transform interval linear inequalities into linear inequalities [4]. Considering any interval linear inequality $\sum_i [a_i, b_i]\, x_i \leq c\, (0 \leq i \leq n)$, where the variable $x_k\, (0 \leq k \leq n)$ is always non-negative or always non-positive. When $x_k \geq 0$, it always holds that $a_k x_k \leq b_k x_k$. Hence, in this case, $\sum_i [a_i, b_i]\, x_i \leq c$ is equivalent to $\sum_{i\neq k} [a_i, b_i]\, x_i + a_k x_k \leq c$. Similarly, when $x_k \leq 0$, it always holds that $a_k x_k \geq b_k x_k$. Hence, in this case, $\sum_i [a_i, b_i]\, x_i \leq c$ is equivalent to $\sum_{i\neq k} [a_i, b_i]\, x_i + b_k x_k \leq c$.

2.3 SafeBound

Neumaier and Shcherbina [19] propose a method (called SafeBound in this paper) that derives the safe bounds of the objective function by post-processing on the approximate result produced by a general floating-point LP solver.

Consider a LP model represented in the following form:

$$\begin{aligned} &\min\ c^T x \\ &s.t.\ Ax \leq b \end{aligned} \tag{2}$$

the dual of which is

$$\begin{aligned} &\max\ b^T y \\ &s.t. \begin{cases} A^T y = c \\ y \leq 0 \end{cases} \end{aligned} \tag{3}$$

Suppose y is an approximate result of (3). By introducing interval arithmetic, we have $r := A^T y - c \in \mathbf{r} = [\underline{r}, \overline{r}]$. Remind that $Ax \leq b$ and $y \leq 0$, and thus we have $c^T x =$

$\left(A^T y - r\right)^T x = y^T A x - r^T x \geq y^T b - r^T x \in y^T b - \mathbf{r}^T \mathbf{x}$. Hence, $\mu := inf\left(y^T b - \mathbf{r}^T \mathbf{x}\right)$ is the safe lower bound for $c^T x$, which can be calculated via floating-point arithmetic as follows:

```
**rounddown:**
$\underline{r} = A^T y - c$;
$t = y^T b$;
**roundup:**
$\overline{r} = A^T y - c$;
$\mu = \max\{\ \underline{r}^T \underline{x},\ \underline{r}^T \overline{x},\ \overline{r}^T \underline{x},\ \overline{r}^T \overline{x}\ \} - t$;
$\mu = -\mu$;
```

where **rounddown** (**roundup**) denotes that we set rounding mode to $-\infty$ ($+\infty$).

2.4 ErrorBound

Jansson [11] proposes a method (called ErrorBound in this paper) to derive rigorous error bounds for the optimal value from boxes that are verified to contain feasible points. And Keil implements this method in Lurupa [14]. The method is described as follows (we refer the proofs to [11]).

Consider the following LP model:

$$f := \min c^T x \qquad s.t. \begin{cases} Ax \leq a \\ \underline{x} \leq x \leq \overline{x} \end{cases} \tag{4}$$

where the simple bounds of x, i.e., $\underline{x}$ and $\overline{x}$, may be infinite (i.e., $\underline{x} = -\infty$ or $\overline{x} = +\infty$), which can lead to uncertainties.

LP model (4) can be formally represented by the parameter tuple $P := (A, a, c)$. To cope with uncertainties of the input data, we introduce interval arithmetic by rewriting P with corresponding interval parameter tuple $\mathbf{P} := (\mathbf{A}, \mathbf{a}, \mathbf{c})$. Then we focus on this resulting interval LP problem $\mathbf{P}$.

The dual of interval LP problem $\mathbf{P}$ is depicted as follows:

$$f := \max \mathbf{a}^T \mathbf{y} + \underline{\mathbf{x}}^T \mathbf{u} + \overline{\mathbf{x}}^T \mathbf{v} \qquad s.t. \begin{cases} \mathbf{A}^T \mathbf{y} + \mathbf{u} + \mathbf{v} = \mathbf{c} \\ \mathbf{y} \leq 0, \mathbf{u} \geq 0, \mathbf{v} \leq 0 \end{cases} \tag{5}$$

▶ **Theorem 1** (Lower Bound). *Consider an interval linear program $\boldsymbol{P}$ with simple bounds $\underline{x} \leq \overline{x}$. Suppose interval vectors $\boldsymbol{y} \leq 0$ satisfy*

1. *for all free x_j and all $A \in \boldsymbol{A}$, there exists $y \in \boldsymbol{y}$ such that*

   $$c_j - (A_{:j})^T y = 0$$

   holds, and
2. *for all variables x_j bounded on one side only the defects*

   $$\boldsymbol{d}_j := c_j - (A_{:j})^T \boldsymbol{y}$$

   are non-negative if the variable is bounded from below and non-positive if it is bounded from above.

Then a rigorous lower bound for the optimal value can be computed as

$$\underset{P\in\boldsymbol{P}}{inf}\ f(P) \geq \underline{f} := inf\left(\boldsymbol{a}^T\boldsymbol{y} + \sum_{\underline{x}_j\neq-\infty} \underline{x}_j\boldsymbol{d}_j^+ + \sum_{\overline{x}_j\neq+\infty} \overline{x}_j\boldsymbol{d}_j^-\right) \tag{6}$$

▶ **Theorem 2** (Upper Bound)**.** *Consider an interval linear program* $\boldsymbol{P}$ *with simple bounds* $\underline{x} \leq \overline{x}$*. Suppose interval vector* $\boldsymbol{x}$ *satisfies*

$$\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{a}, \underline{x} \leq \boldsymbol{x} \leq \overline{x} \tag{7}$$

Then a rigorous upper bound for the optimal value can be computed as

$$\underset{P\in\boldsymbol{P}}{sup}\ f(P) \leq \ \overline{f} := max\{\boldsymbol{c}^T\boldsymbol{x}\} \tag{8}$$

3 Overview

In this section, we provide a simple but typical example to illustrate our motivation. Consider the following LP problem:

$$\begin{aligned} &\min\ z = -9x_0 + 6x_1 - 4x_2 \\ &s.t. \begin{cases} 3x_0 - 8x_1 + 5x_2 & \leq -14 \\ -5x_0 - 2x_1 + 6x_2 & \leq 17 \\ 5x_0 + 2x_1 - 6x_2 & \leq -17 \\ -2x_0 + 4x_1 & \leq 19 \\ x_0, x_1, x_2 & \geq 0 \end{cases} \end{aligned} \tag{9}$$

Table 1 shows the results and execution time of solving the above LP problem via different LP and RLP techniques. The first row of Table 1 shows the exact result given by *glp_exact* which is based on exact arithmetic from GLPK [16], a linear programming kit maintained by GNU. The existing RLP technique proposed by Neumaier and Shcherbina [19] (called SafeBound in Sect. 2.4) only produces minus infinity which is sound but too conservative. The reason lies in that the bounds of most variables in problem (9) involve infinity. Lurupa [14] which implements Jansson's method [11] (we call ErrorBound in Sect. 2.4), produces a finite lower bound. Compared with Lurupa and SafeBound, our FME-based RLP produces a more precise result which is also a rigorous finite lower bound. Besides, from the third column of Table 1, we can see that among these techniques, our FME-based RLP has the best performance.

Table 1 Results of motivating example.

approaches	results	time(s)
glp_exact	6.04166666666666785090	0.000292
Lurupa [14]	6.04166582676214503067	0.000063
SafeBound [19]	$-\infty$	0.000160
FME-based RLP	6.04166666666666607454	0.000014

4 RLP based on Fourier-Mozkin Elimination

In this section, we will present our RLP approach based on Fourier-Mozkin elimination (FME). First, we review how to solve LP problem using FME. Then, we introduce how to derive a sound floating-point FME to construct a RLP approach. After that, we introduce techniques to improve efficiency of FME, so as to improve efficiency of RLP.

4.1 LP via Fourier-Mozkin Elimination

Fourier-Mozkin elimination is a general technique to perform variable elimination from a system of linear inequalities. Solving LP problems mathematically via FME has been discussed by Williams [26]. In this subsection, we will introduce the principle of applying FME to solve the following LP problem:

$$\max\ c^T x \qquad s.t. \begin{cases} Ax \leq b \\ x \geq 0 \end{cases} \tag{10}$$

To solve LP problem (10) by FME, we need to reconstruct the objective function by generating a new linear inequality, namely, $x_{n+1} - c^T x \leq 0$ and then we get a new linear inequality system:

$$\begin{cases} Ax \leq b \\ x_{n+1} \leq c^T x \\ x \geq 0 \end{cases} \tag{11}$$

For (11), we can get the upper bound of newly added variable, i.e., x_{n+1}, by applying FME to eliminate all the other variables in (11). Obviously, the upper bound of x_{n+1} is the maximum value of $c^T x$. Computing the minimum value of $c^T x$ can be reformulated as the following problem:

$$\min\ c^T x = -\left(\max\ -c^T x\right) \tag{12}$$

Following the above process, if applying FME based on exact arithmetic, we can get the exact maximum (minimum) value of the LP problem. However, if we conduct FME using floating-point arithmetic, the result may be unsound. In other words, we may get a smaller (larger) value than the exact maximum (minimum) value.

To this end, in this paper, we propose a RLP approach based on FME using floating-point arithmetic. The key idea is to use interval arithmetic and linearization techniques to derive a sound floating-point FME. In the following subsections, we first describe how to get a sound floating-point FME and then describe techniques to improve the precision and efficiency.

4.2 Sound Floating-Point FME

The key idea of constructing sound floating-point FME is to use interval arithmetic with outward rounding, that is, rounding up for computing upper bound and rounding down for computing lower bound. With interval arithmetic, we will get a new form of linear inequality in which all coefficients of variables are intervals.

For the sake of presentation, we introduce the following notations to denote floating-point operations:

- $\oplus_r$: floating-point addition.
- $\ominus_r$: floating-point minus.
- $\otimes_r$: floating-point multiplication.
- $\oslash_r$: floating-point division.

where $r \in \{+\infty, -\infty\}$ represents rounding mode in which $+\infty$ means upward and $-\infty$ means downward.

Assume we want to eliminate variable x_i $(i \leq n)$ from the following two inequalities:

$$\begin{cases} a_i^+ x_i + \sum\limits_{k \neq i, k \leq n} a_k^+ x_k + [\underline{a_{n+1}^+}, \overline{a_{n+1}^+}]x_{n+1} \leq c^+ & \text{if } a_i^+ > 0 \quad (13) \\ a_i^- x_i + \sum\limits_{k \neq i, k \leq n} a_k^- x_k + [\underline{a_{n+1}^-}, \overline{a_{n+1}^-}]x_{n+1} \leq c^- & \text{if } a_i^- < 0 \quad (14) \end{cases}$$

where only the coefficient for variable x_{n+1} (that is introduced in (11) to denote the objective value of the original LP problem) is an interval and all coefficients for other variables x_k $(k \leq n)$ are scalars. After dividing (13) and (14) respectively by the absolute value of the coefficient of x_i using interval arithmetic with outward rounding, we have

$$x_i + \sum_{k \neq i, k \leq n} [a_k^+ \oslash_{-\infty} a_i^+, a_k^+ \oslash_{+\infty} a_i^+]x_k$$
$$+ [\underline{a_{n+1}^+} \oslash_{-\infty} a_i^+, \overline{a_{n+1}^+} \oslash_{+\infty} a_i^+]x_{n+1} \leq c^+ \oslash_{+\infty} a_i^+ \quad (15)$$

where $a_i^+ > 0$ and

$$-x_i + \sum_{k \neq i, k \leq n} [a_k^- \oslash_{-\infty} (\ominus a_i^-), a_k^- \oslash_{+\infty} (\ominus a_i^-)]x_k$$
$$+ [\underline{a_{n+1}^-} \oslash_{-\infty} (\ominus a_i^-), \overline{a_{n+1}^-} \oslash_{+\infty} (\ominus a_i^-)]x_{n+1} \leq c^- \oslash_{+\infty} (\ominus a_i^-) \quad (16)$$

where $a_i^- < 0$. By adding (15) and (16), we have

$$\sum_{k \neq i, k \leq n} [(a_k^+ \oslash_{-\infty} a_i^+) \oplus_{-\infty} (a_k^- \oslash_{-\infty} (\ominus a_i^-)), (a_k^+ \oslash_{+\infty} a_i^+) \oplus_{+\infty} (a_k^- \oslash_{+\infty} (\ominus a_i^-))]x_k +$$
$$[(\underline{a_{n+1}^+} \oslash_{-\infty} a_i^+) \oplus_{-\infty} (\underline{a_{n+1}^-} \oslash_{-\infty} (\ominus a_i^-)), (\overline{a_{n+1}^+} \oslash_{+\infty} a_i^+) \oplus_{+\infty} (\overline{a_{n+1}^-} \oslash_{+\infty} (\ominus a_i^-))]x_{n+1}$$
$$\leq (c^+ \oslash_{+\infty} a_i^+) \oplus_{+\infty} (c^- \oslash_{+\infty} (\ominus a_i^-)) \quad (17)$$

Then according to the LP model (10), we have $x_k \geq 0$ when $k \leq n$. Hence, via Orthant Reduction technique described in Sect. 2.2, (17) can be linearized into the following form (where all coefficients for x_k $(k \leq n)$ are scalars):

$$\sum_{k \neq i, k \leq n} d_k x_k + [\underline{d_{n+1}}, \overline{d_{n+1}}]x_{n+1} \leq c' \quad (18)$$

where

$$d_k = ((a_k^+ \oslash_{-\infty} a_i^+) \oplus_{-\infty} (a_k^- \oslash_{-\infty} (\ominus a_i^-)) \qquad \textit{when} \quad k \leq n$$

$$\underline{d_{n+1}} = (a_{n+1}^+ \oslash_{-\infty} a_i^+) \oplus_{-\infty} (a_{n+1}^- \oslash_{-\infty} (\ominus a_i^-))$$

$$\overline{d_{n+1}} = (a_{n+1}^+ \oslash_{+\infty} a_i^+) \oplus_{+\infty} (a_{n+1}^- \oslash_{+\infty} (\ominus a_i^-))$$

and

$$c' = (c^+ \oslash_{+\infty} a_i^+) \oplus_{+\infty} (c^- \oslash_{+\infty} (\ominus a_i^-))$$

We use the above process to eliminate a variable x_i $(i \leq n)$ from a system of inequalities where only the coefficient for variable x_{n+1} is an interval and the coefficients for other variables x_k $(k \leq n)$ are scalars, which results in a system of the same form. Obviously, from the implementation point of view, we can skip the calculation of upper bound of interval coefficient of x_k (when $k \leq n$) in (15) (17), which can reduce some unnecessary calculations.

The division operation during converting (13) ~(14) into (15) ~(16) may introduce many interval coefficients for other variables, and converting interval coefficients into scalar ones using linearization may introduce precision loss. Considering that integer values of normal (not too large) magnitude can be represented exactly in floating-point representation, we also consider implementing FME using multiplication described as follows.

Assume $a_i^+ \otimes_{-\infty} a_i^- = a_i^+ \otimes_{+\infty} a_i^-$ can be represented exactly by floating-point representation (e.g., an integer). Then, after multiplying (13) by the minus of coefficient of x_i in (14) and multiplying (14) by the coefficient of x_i in (13) using interval arithmetic with outward rounding, we have

$$\begin{aligned} &- a_i^+ a_i^- x_i + \sum_{k \neq i, k \leq n} [a_k^+ \otimes_{-\infty} (\ominus a_i^-), a_k^+ \otimes_{+\infty} (\ominus a_i^-)] x_k \\ &\qquad + [\underline{a_{n+1}^+} \otimes_{-\infty} (\ominus a_i^-), \overline{a_{n+1}^+} \otimes_{+\infty} (\ominus a_i^-)] x_{n+1} \leq c^+ \otimes_{+\infty} (\ominus a_i^-) \end{aligned} \tag{19}$$

$$\begin{aligned} &a_i^+ a_i^- x_i + \sum_{k \neq i, k \leq n} [a_k^- \otimes_{-\infty} a_i^+, a_k^- \otimes_{+\infty} a_i^+] x_k \\ &\qquad + [\underline{a_{n+1}^-} \otimes_{-\infty} a_i^+, \overline{a_{n+1}^-} \otimes_{+\infty} a_i^+] x_{n+1} \leq c^- \otimes_{+\infty} a_i^+ \end{aligned} \tag{20}$$

where $a_i+ > 0$ and $a_i^- < 0$. By adding (19) and (20), we have

$$\begin{aligned} &\sum_{k \neq i, k \leq n} [(a_k^+ \otimes_{-\infty} (\ominus a_i^-)) \oplus_{-\infty} (a_k^- \otimes_{-\infty} a_i^+), (a_k^+ \otimes_{+\infty} (\ominus a_i^-)) \oplus_{+\infty} (a_k^- \otimes_{+\infty} a_i^+)] x_k + \\ &[(\underline{a_{n+1}^+} \otimes_{-\infty} (\ominus a_i^-)) \oplus_{-\infty} (\underline{a_{n+1}^-} \otimes_{-\infty} a_i^+), (\overline{a_{n+1}^+} \otimes_{+\infty} (\ominus a_i^-)) \oplus_{+\infty} (\overline{a_{n+1}^-} \otimes_{+\infty} a_i^+)] x_{n+1} \\ &\qquad \leq (c^+ \otimes_{+\infty} (\ominus a_i^-)) \oplus_{+\infty} (c^- \otimes_{+\infty} a_i^+) \end{aligned} \tag{21}$$

Then we can linearize (21) into scalar form similarly as linearizing (17). Similarly, we can skip the calculation of upper bound of interval coefficient of x_k in (21).

Finally, after we eliminate all variables x (consisting of x_k's $(k \leq n)$ from the system (11), we will result in an one-variable interval linear system over x_{n+1}:

$$\begin{cases} [\underline{a_1}, \overline{a_1}] x_{n+1} \leq b_1 \\ \cdots \\ [\underline{a_m}, \overline{a_m}] x_{n+1} \leq b_m \end{cases}$$

which is equivalent to the disjunction of the following two linear systems:

$$\begin{cases} -x_{n+1} \leq 0 \\ \underline{a_1} x_{n+1} \leq b_1 \\ \cdots \\ \underline{a_m} x_{n+1} \leq b_m \end{cases} \quad \vee \quad \begin{cases} x_{n+1} \leq 0 \\ \overline{a_1} x_{n+1} \leq b_1 \\ \cdots \\ \overline{a_m} x_{n+1} \leq b_m \end{cases}$$

from which we can easily drive the upper bound of variable x_{n+1}.

To sum up, overall, we can derive FME-based RLP by substituting the process of FME described in Sect. 4.1 with sound floating-point FME described in this subsection. As floating-point FME is sound, so the result of FME-based RLP is also rigorous.

4.3 Redundancy Removal

It is known that FME for eliminating multiple variables may introduce a large number of redundant constraints during the process, which may lead to extra space and time cost. Hence, redundancy removal is a significant point to make FME practical.

In this paper, we utilize bit vector to implement two techniques, that is, Chernikov's rule [5] and Kohler's rule [15], to remove redundant constraints. The main ideas of these two techniques are described in the following subsections.

To simplify the descriptions of redundancy removal techniques, we rewrite linear inequality system (11) as

$$A'x' \leq b' \tag{22}$$

where A', x' and b' respectively are

$$A' = \begin{pmatrix} A & 0 \\ -I & 0 \\ -c^T & 1 \end{pmatrix}, \qquad x' = \begin{pmatrix} x \\ x_{n+1} \end{pmatrix}, \qquad b' = \begin{pmatrix} b \\ 0 \\ 0 \end{pmatrix} \tag{23}$$

Note that sometimes we also represent linear system (22) via the parameter tuple (A', b').

4.3.1 Bit Vector

We associate each inequality respectively with an index set which consists of integers representing the row index of the inequality in the original system (22). For efficiency, we encode the index set via a bit vector. The main idea is that if one inequality generated during FME is a combination result of some original inequalities, then in the bit vector of the generated inequality, the bits corresponding to the combined original inequalities will be set to 1, while the remaining bits are set to 0. E.g., if an inequality φ is a combination result of the first and third inequalities of the original inequality system, then the bit vector of φ is $0 \cdots 00101$ (the lowest bit from right corresponds to the first inequality in the system).

4.3.2 Chernikov's Rule

In [5], Chernikov proposes a heuristic rule to avoid generating some redundant constraints during FME by restricting the length of the index set associated with each inequality. To simplify description, we write q_i to denote the index set of inequality i. Before the starting of elimination, the index sets of each constraint q_i in the original inequality system (22) are initialized as singleton sets which consist of the row index i of the corresponding constraint q_i in the constraint matrix A' (i.e., q_i is one of {1, 2, ..., m}, where m is the number of rows in A'). Assume that we want to make a combination between inequalities i and j. Before doing combination, we can calculate the index set of this combination, that is, $q_{ij} = q_i \cup q_j$. If the size of q_{ij} is strictly greater than $s + 1$, where s means the combination is to be conducted during the process of eliminating the s-th variable (after eliminating $s - 1$ variables), then we can skip this combination since the resulting constraint is definitely redundant [5].

Algorithm 1 depicts the procedure of FME integrated with Chernikov's rule.

Algorithm 1 Procedure of FME with Chernikov's rule.

```
/* $(A', b')$ denotes the linear inequality system (22)*/
/* $bv'$ denotes the set of bit vectors corresponding to the inequalities in $(A', b')$ */
/* $j$ means the index of the variable to be eliminated, also corresponding to the $j$-th column of $A'$*/
procedure $FME\_Iteration(A', b', bv', j)$
    // $i$ means the i-th row of $A'$ $(i = 1, 2, \ldots, m)$
    $I_j^+ \leftarrow \{i : a_{ij} > 0\}$;   $I_j^- \leftarrow \{i : a_{ij} < 0\}$;   $I_j^0 \leftarrow \{i : a_{ij} = 0\}$
    // Extracting inequalities and their corresponding bit vectors of $A'$ with indices in $I_j^0$
    $(A'', b'', bv'') \leftarrow (A', b', bv') |_{I_j^0}$
    for $k^+ \in I_j^+$ do
        for $k^- \in I_j^-$ do
            // $bv\_1bits()$ derives the number of one's in a bit vector
            $ij\_1bits \leftarrow bv\_1bits(bv_{k^+} \mid bv_{k^-})$   //Here $|$ denotes bitwise OR operation
            // Check Chernikov's rule
            if $ij\_1bits > j + 1$ then
                continue
            end if
            // $FME\_combine()$ combines of two inequalities, as described in Sect. 4.2
            // $FME\_add()$ adds one inequality together with its corresponding
            //     bit vector into the resulting inequality system
            $(a_c, b_c) \leftarrow FME\_combine((A'_{k^+}, b'_{k^+}), (A'_{k^-}, b'_{k^-}))$
            $(A'', b'', bv'') \leftarrow FME\_add((A'', b'', bv''), (a_c, b_c, bv_{k^+} \mid bv_{k^-}))$
        end for
    end for
    return $(A'', b'', bv'')$
end procedure
```

Algorithm 2 Procedure of redundancy removal with Kohler's Rule.

```
procedure $RMR\_ByKohler(A', b', bv')$
    $(A'', b'') \leftarrow \emptyset$;   $bv'' \leftarrow \emptyset$
    // $nb\_rows()$ derives the number of rows for the parameter matrix
    for $i \leftarrow 0$ to $nb\_rows(A') - 1$ do
        $flag = false$
        for $j \leftarrow i + 1$ to $nb\_rows(A')$ do
            if $(bv_i \mid bv_j) == bv_i$ then
                $flag = true$
                $break$
            end if
        end for
        if $flag == false$ then
            $(A'', b'', bv'') \leftarrow FME\_add((A'', b'', bv''), (A'_i, b'_i, bv'_i))$
        end if
    end for
    return $(A'', b'', bv'')$
end procedure
```

4.3.3 Kohler's Rule

Kohler's rule is another well-known technique for removing redundant constraints during FME. In our design, after doing FME in one iteration step (i.e., eliminating one variable), we can derive a new inequality system, (A', b'), together with its bit vectors, bv'. Then we check each inequality in (A', b') with others to find out whether the subset relation exists between them, to determine the redundant inequality, more clearly, superset being redundant and subset not. Algorithm 2 depicts the procedure of Kohler's rule to remove redundant constraints.

Finally, we integrate Chernikov's rule and Kohler's rule into the process of FME. Algorithm 3 depicts the procedure of our FME-based RLP, with initial inequality system as input and maximum value of objective function as output.

Algorithm 3 Procedure of FME-based RLP.

```
Input: Initial linear inequality system $(A', b')$
Output: The maximum value of $x_{n+1}$
1: // $bv\_initial()$ initializes bit vectors for initial system, as described in Sect. 4.3.1
2: $bv' \leftarrow bv\_initial(A', b')$
3: for $j \leftarrow 1$ to $nb\_columns(A')$ do
4:     $(A', b', bv') \leftarrow FME\_Iteration(A', b', bv', j)$
5:     $(A', b', bv') \leftarrow RMR\_ByKohler(A', b', bv')$
6: end for
7: // $max()$ derives the maximum value for objective function (noting
8: //     that at this location, $(A', b')$ only involves one variable, i.e., $x_{n+1}$)
9: return $max(A', b')$
```

4.4 Optimization Considering Sparsity

In the field of program analysis, the objective function and constraint system are usually sparse [24, 25], i.e., they contain mostly zeros. Hence, when we solve RLP problems encountered during program analysis, we may make use of the sparsity in the LP problem to accelerate the solving process.

Consider the LP problem encoded in (22), we say two variables x_i and x_j are *relevant*, if there exists a constraint ϕ in $A'x' \leq b'$ of (22) such that the coefficients of x_i and x_j in ϕ are not zero. The defined *relevant* relation is an equivalence relation on the set of variables in x', and the collection of its equivalence classes forms a partition of set of variables in x'. Let S denote the equivalence class that variable x_{n+1} belongs in, and let $\bar{S}$ denote the set of variables not in S. Then the LP problem encoded in (22) can be reformulated as

$$A'x' \leq b'$$

where

$$A' = \begin{pmatrix} A_S & 0 \\ 0 & A_{\bar{S}} \end{pmatrix}, \qquad x' = \begin{pmatrix} x_S \\ x_{\bar{S}} \end{pmatrix}, \qquad b' = \begin{pmatrix} b_S \\ b_{\bar{S}} \end{pmatrix} \tag{24}$$

Thus, the LP problem encoded via

$$A'x' \leq b'$$

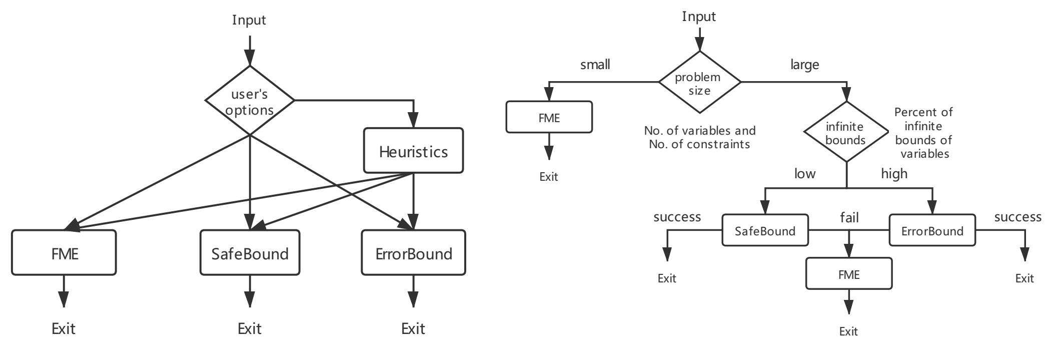
can be reduced to the following equivalent LP problem:

$$A_S x_S \leq b_S \tag{25}$$

Note that to derive the variable bound for x_{n+1}, (25) is equivalent to (22). It is worth mentioning that solving (25) will be more efficient than solving (22), since solving (25) involves less variables to be eliminated from a linear system with less constraints.

5 Integrating RLP Techniques

Our proposed FME-based RLP, and two existing RLP techniques (i.e., SafeBound and ErrorBound) have their own advantages and disadvantages. E.g., FME-based RLP has high precision and specializes in small-scale LP problems but degrades a lot in efficiency when the scale of LP problems increases. SafeBound and ErrorBound can handle large-scale LP problems but have poor accuracy. To make RLP more practical and effective, we implement a tool called RlpSolver, combining our FME-based RLP together with two existing RLP techniques, that is, SafeBound [19], ErrorBound [11] (implemented in Lurupa [14]). Fig. 1 provides an overview of RlpSolver.



(a) Framework of RlpSolver **(b)** Workflow of Heuristics in RlpSolver

Figure 1 Overview of RlpSolver

As shown in Fig. 1a, in RlpSolver, we can choose any of the three techniques to solve one LP problem. Moreover, RlpSolver provides an option to automatically choose a proper technique to solve a LP problem via heuristic rules. The workflow of the heuristics we design is depicted in Fig. 1b, which automatically chooses a proper RLP technique for a given LP problem. The details of heuristic rules are as follows:

- FME-based RLP is often more precise than SafeBound and ErrorBound in many cases, but may cost more time when the number of constraints is greater than some threshold. Hence, by default, RlpSolver will choose FME-based RLP when the number of constraints is less than some threshold. In other cases, we will try SafeBound or ErrorBound.
- When the number of constraints is greater than some threshold, we will choose between ErrorBound and SafeBound. Specifically, when many variables' bounds are infinite (i.e., when the percent of infinite bounds exceeds some threshold), we will use ErrorBound since SafeBound may often give infinity as results and ErrorBound behaves better than SafeBound for infinite bounds.
- Since ErrorBound and SafeBound depend upon floating-point simplex or other LP algorithms, they may encounter numerical instability when solving LP problems and may not fit for dealing with ill-conditioned (dual) problems, while FME-based RLP is more robust than ErrorBound and SafeBound, thus we will try FME-based RLP if ErrorBound or SafeBound fails.

6 Implementation and Experiments

To evaluate the precision and efficiency of our proposed FME-based RLP and the integration tool RlpSolver, we conduct experiments over randomly generated LP problems. We apply glp_exact, SafeBound, Lurupa (which implements ErrorBound), and FME-based RLP to our benchmark respectively. We use the exact LP API from GLPK (a linear programming kit maintained by GNU) [16], i.e., glp_exact (which is implemented via exact arithmetic), as our baseline.

By setting thresholds for the number of variables and constraints (10 and 15 respectively), we split our benchmark into four categories, that is, small number of variables with small number of constraints (SV_SC), small number of variables with large number of constraints (SV_LC), large number of variables with small number of constraints (LV_SC), and large number of variables with large number of constraints (LV_LC). In our benchmark, the ranges of sizes of constraints (and variables) in SV_SC, SV_LC, LV_SC and LV_LC are 6~14 (4~9), 15~25 (4~9), 12~14 (10~12), 15~25 (10~18) respectively. Fig. 2 shows the log of execution time. From Fig. 2, we can see that when the number of constraints is small (as shown in Fig. 2 (a) and (c)), the performance of FME-based RLP is almost at the same level as Lurupa and SafeBound, even better in many cases. When the number of constraints is large (as shown in Fig. 2 (b) and (d)), FME-based RLP costs more time than SafeBound and Lurupa, but its performance is mostly better than that of glp_exact.

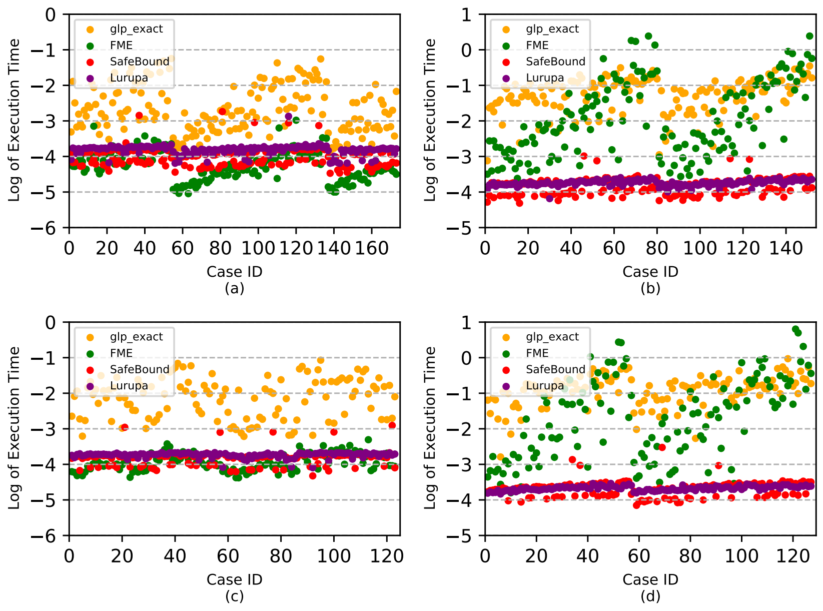


Figure 2 Execution time over benchmark SV_SC (a), SV_LC (b), LV_SC (c), and LV_LC (d).

For the precision, as shown in Table 2, we provide the statistics of the number of instances that FME-based RLP, SafeBound and Lurupa respectively provide the highest precision in benchmark. For example, in SV_LC category, there are 149 out of 152 cases where FME-based RLP obtains the highest precision. Experimental results show that FME-based RLP is mostly more precise than SafeBound and Lurupa. Thus, when the number of constraints is small, FME-based RLP is the best choice and also a good choice in the cases where the number of constraints is large but requiring high precision. Moreover, during experiments, we find that there are many cases where SafeBound outputs too conservative results (i.e., infinity as the objective value) while FME-based RLP can output bounded (finite) results which are close to that of glp_exact. In other words, FME-based RLP can be used in these cases where SafeBound or Lurupa provide too conservative results, to improve the precision of program analysis. The last column of Table 2 shows that there are 272 out of 575 cases where integration tool RlpSolver can derive the highest precision.

Table 2 The number of cases where each technique provides the highest precision.

Categories	Total	FME-based RLP	SafeBound	Lurupa	Equal[1)]	RlpSolver
SV_SC	173	157[2)]	8	24	7	156
SV_LC	152	149	2	1	0	1
LV_SC	123	115[3)]	1	10	1	115
LV_LC	127	127	0	0	0	0

1) "Equal" means the results of FME-based RLP, SafeBound and Lurupa are equal.
2) There is 1 case where FME-based RLP is equal to only SafeBound and 1 case where FME-based RLP is equal to only Lurupa in this category.
3) There is 1 case where FME-based RLP is equal to only Lurupa in this category.

For the execution time, Fig. 3 shows the cumulative time of different techniques for solving LP problems in benchmark. The dashed line at the top left corner of Fig. 3 means the time-out period. The curves of FME-based RLP and RlpSolver show that FME-based RLP and RlpSolver take less execution time over the left half part than Lurupa and SafeBound. Over the right half part (which consists of problems with large number of constraints), the performance of FME-based RLP degrades a lot, while the execution times of RlpSolver, SafeBound and Lurupa are still in the same order of magnitude. On the whole, via heuristic rules, RlpSolver can achieve a good balance in terms of time cost and precision by choosing a proper technique for different cases.

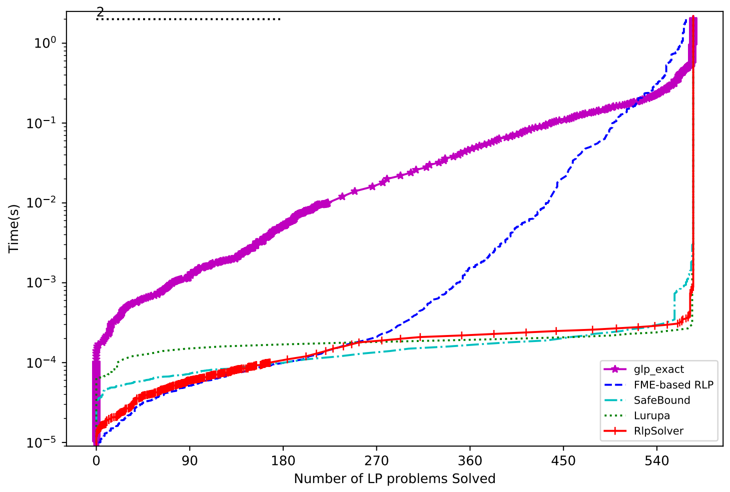


Figure 3 Execution time of several techniques.

7 Related Work

FME and FME-based LP. Fourier-Mozkin elimination is a general technique to perform variable elimination from a system of linear inequalities. Kohler [15] proposes a heuristic rule (called Kohler's rule now) for removing redundant constraints. Chernikov [5] proposes additional rules to avoid generating some redundant combinations during elimination. Bastrakov et al. [1] propose a new way of checking Chernikov rules using bit pattern trees as an accelerating data structure to avoid extensive enumeration. Maréchal et al [17] present a raytracing algorithm that replaces most LP problem resolutions by distance computations to efficiently eliminate redundancies in polyhedra. Solving LP problems via FME has been continuously studied. Williams [26] first adapts FME to solve LP problems mathematically. Kanniappan et al. [12] propose a modified FME method of solving LP problems which can reduce the number of additional constraints to a considerable extent.

RLP. Rigorous linear programming has received much attention in the recent two decades. In the 2003 seminal paper, Neumaier and Shcherbina [19] propose a technique that computes safe bounds of objective value of primal problem by solving the dual problem with floating-point linear programming. At almost the same time, Jansson [11] proposes another method for LP with uncertain input data and infinite bounds. Keil implements Jansson's method [11] in a tool, named Lurupa [14]. Guilbeau et al. [10] review the technique proposed in[19] and point out typographical errors in original publication and give some advice for other implementers. Rump [20] gives some details on how to obtain mathematically rigorous results for global optimization implemented in floating-point arithmetic.

LP and RLP in Analysis and Verification. LP is widely-used in analysis and verification of programs, neural networks, etc. Sankaranarayanan et al. [22] use standard LP in their Template Constraint Matrix (TCM) domain to compute the right-hand constants for templates. Chen et al. [3] use RLP in their floating-point polyhedra domain [3] and interval polyhedra domain [4] to support domain operations. David Monniaux [18] proposes a simple but sound and complete preprocessing phase which can be adapted to existing SMT solvers via floating-point computations to help an exact linear arithmetic decision procedure. Besson [2] explains how to design a sound procedure for linear arithmetic built on an inexact floating-point LP solver. Dillig et al. [8] propose a sound and complete simplex-based algorithm for solving linear inequalities over integers which can be viewed as a semantic generalization of the branch-and-bound technique. It is also worth mentioning that LP and mixed integer LP (MILP) are used in recent neural network verification [9, 13].

8 Conclusion

Rigorous linear programming is an important technique to make implementations of program analysis and verification techniques sound and efficient in practice. Existing RLP techniques sometimes produce too conservative (even unbounded) results, especially when many bounds of variables in the problem are infinite or when the LP problem is ill-conditioned. To address this problem, we propose a new technique, FME-based RLP, which can be treated as a supplement to existing RLP techniques. On this basis, we implement a tool, RlpSolver, wrapping FME-based RLP and existing RLP techniques together, and propose heuristics to select a proper technique for different LP problems. Experimental results show that our FME-based RLP is complementary to existing RLP techniques and provides more precise results than existing RLP techniques for many cases. Experimental results also show that our RLP integration tool, i.e., RlpSolver, achieves a good performance in terms of precision and efficiency by choosing a proper technique for different cases.

For future work, we plan to make use of more techniques to expedite removing redundancy during the process of FME. We also plan to conduct experiments in the context of using RLP in program analysis and verification.

References

1 SI Bastrakov, AV Churkin, and N Yu Zolotykh. Accelerating fourier–motzkin elimination using bit pattern trees. *Optimization Methods and Software*, pages 1–14, 2020.

2 Frédéric Besson. On using an inexact floating-point lp solver for deciding linear arithmetic in an smt solver. In *8th International Workshop on Satisfiability Modulo Theories*, 2010.

3 Liqian Chen, Antoine Miné, and Patrick Cousot. A sound floating-point polyhedra abstract domain. In *Asian Symposium on Programming Languages and Systems*, pages 3–18. Springer, 2008.

4 Liqian Chen, Antoine Miné, Ji Wang, and Patrick Cousot. Interval polyhedra: An abstract domain to infer interval linear relationships. In *International Static Analysis Symposium*, pages 309–325. Springer, 2009.

5 Sergei Nikolaevich Chernikov. The convolution of finite systems of linear inequalities. *USSR Computational Mathematics and Mathematical Physics*, 5(1):1–24, 1965.

6 Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252, 1977.

7 Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 84–96, 1978.

8 Isil Dillig, Thomas Dillig, and Alex Aiken. Cuts from proofs: A complete and practical technique for solving linear inequalities over integers. In *International Conference on Computer Aided Verification*, pages 233–247. Springer, 2009.

9 Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 269–286. Springer, 2017.

10 Jared T Guilbeau, Md Istiaq Hossain, Sam D Karhbet, Ralph Baker Kearfott, Temitope S Sanusi, and Lihong Zhao. A review of computation of mathematically rigorous bounds on optima of linear programs. *Journal of Global Optimization*, 68(3):677–683, 2017.

11 Christian Jansson. Rigorous error bounds for the optimal value of linear programming problems. In *International Workshop on Global Optimization and Constraint Satisfaction*, pages 59–70. Springer, 2002.

12 P Kanniappan and K Thangavel. Modified fourier's method of solving linear programming problems. *Opsearch*, 35(1):45–56, 1998.

13 G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, 2017.

14 Christian Keil. Lurupa-rigorous error bounds in linear programming. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2006.

15 David A Kohler. Projections of convex polyhedral sets. Technical report, CALIFORNIA UNIV BERKELEY OPERATIONS RESEARCH CENTER, 1967.

16 Andrew Makhorin. Gnu linear programming kit. *Moscow Aviation Institute, Moscow, Russia*, 38, 2001.

17 Alexandre Maréchal and Michaël Périn. Efficient elimination of redundancies in polyhedra by raytracing. In *International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 367–385. Springer, 2017.

18 David Monniaux. On using floating-point computations to help an exact linear arithmetic decision procedure. In *International Conference on Computer Aided Verification*, pages 570–583. Springer, 2009.

19 Arnold Neumaier and Oleg Shcherbina. Safe bounds in linear and mixed-integer linear programming. *Mathematical Programming*, 99(2):283–296, 2004.

20 Siegfried M Rump. Mathematically rigorous global optimization in floating-point arithmetic. *Optimization Methods and Software*, 33(4-6):771–798, 2018.

21 Sriram Sankaranarayanan, Thao Dang, and Franjo Ivančić. Symbolic model checking of hybrid systems using template polyhedra. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 188–202. Springer, 2008.

22 Sriram Sankaranarayanan, Henny B Sipma, and Zohar Manna. Scalable analysis of linear systems using mathematical programming. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 25–41. Springer, 2005.

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

24 Gagandeep Singh, Markus Püschel, and Martin T. Vechev. Making numerical program analysis fast. In David Grove and Stephen M. Blackburn, editors, *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*, pages 303–313. ACM, 2015.

25 Gagandeep Singh, Markus Püschel, and Martin T. Vechev. A practical construction for decomposing numerical abstract domains. *Proc. ACM Program. Lang.*, 2(POPL):55:1–55:28, 2018.

26 H Paul Williams. Fourier's method of linear programming and its dual. *The American mathematical monthly*, 93(9):681–695, 1986.