# Satisfiability of Non-Linear (Ir)rational Arithmetic*

Harald Zankl and Aart Middeldorp

Institute of Computer Science, University of Innsbruck, Austria

**Abstract.** We present a novel way for reasoning about (possibly ir)rational quantifier-free non-linear arithmetic by a reduction to SAT/SMT. The approach is incomplete and dedicated to satisfiable instances only but is able to produce models for satisfiable problems quickly. These characteristics suffice for applications such as termination analysis of rewrite systems. Our prototype implementation, called MiniSmt, is made freely available. Extensive experiments show that it outperforms current SMT solvers especially on rational and irrational domains.

**Key words:** non-linear arithmetic, SMT solving, term rewriting, termination, matrix interpretations.

## 1 Introduction

Establishing termination of programs (automatically) is essential for many aspects of software verification. Contemporary termination analyzers for term rewrite systems (TRSs) rely on solving (non-)linear arithmetic, mostly—but not exclusively—over the natural numbers. Hence designated solvers for non-linear arithmetic are very handy when implementing termination criteria for term rewriting (e.g. recently in [7, 13, 14, 20, 31]) but also in different research domains dealing with verification (e.g. recently in [18]).

In this paper we explain the theory underlying our SMT solver MiniSmt, which is freely available under terms of the GNU lesser general public license version 3 from `http://cl-informatik.uibk.ac.at/software/minismt`. This tool is designed to find models for satisfiable instances of non-linear arithmetic quickly. Integral domains are handled by bit-blasting to SAT and, alternatively, by an appropriate transformation to bit-vector arithmetic before solvers for these logics are employed. Non-integral domains are also supported by a suitable reduction to the integral setting. To solve constraints over *rational* domains efficiently we propose a heuristic which is easy to implement. Experiments on various benchmarks show gains in power and efficiency compared to contemporary existing approaches. The support for irrational domains (by approximating comparisons involving $\sqrt{2}$) distinguishes our tool.

We expect two major effects of our contribution: MiniSmt eases the job to develop a new termination tool (fast reasoning about arithmetic is also relevant

* This research is supported by FWF (Austrian Science Fund) project P18763.

for implementing other termination criteria) and our test benches will spark further research in the SMT community on non-linear arithmetic.

The remainder of the paper is organized as follows. How we reduce non-linear non-integral arithmetic over (possibly ir)rational domains to integral arithmetic is outlined in Section 2. Experiments showing the benefit of our approach are presented in Section 3 before Section 4 compares our approach with related work. For the convenience of the reader an encoding of (bounded) integral non-linear arithmetic in SAT is given in Appendix A. These encodings are similar to known ones but take overflows into account. To obtain benchmarks for non-integral domains we generalize a popular termination criterion for rewrite systems—matrix interpretations [13, 20]—to non-negative *real* coefficients in Appendix B. To automate the method, models for non-linear arithmetic constraints must be found quickly for satisfiable instances.

## 2 Encoding Non-Linear Non-Integral Arithmetic

In this section we introduce a grammar for non-linear arithmetic constraints (which appear when automating matrix interpretations, among other termination criteria for term rewriting) and show how to reduce constraints over non-integral arithmetic to the integral case.

**Definition 1.** *An* arithmetic constraint $\varphi$ *is described by the BNFs*

$$\varphi ::= \bot \mid \top \mid p \mid (\neg\varphi) \mid (\varphi \circ \varphi) \mid (\alpha \star \alpha) \quad \textit{and} \quad \alpha ::= \mathrm{a} \mid r \mid (\alpha \diamond \alpha) \mid (\varphi \,?\, \alpha : \alpha)$$

*where* $\circ \in \{\vee, \wedge, \to, \leftrightarrow\}$, $\star \in \{>, =\}$, *and* $\diamond \in \{+, -, \times\}$.

Here $\bot$ ($\top$) denotes contradiction (tautology), $p$ (a) ranges over Boolean (arithmetical) variables, $\neg$ ($\vee$, $\wedge$, $\to$, $\leftrightarrow$) is logical not (or, and, implication, bi-implication), $>$ ($=$) greater (equal), $r$ ranges over the real numbers, and $+$ ($-$, $\times$) denotes addition (subtraction, multiplication). If-then-else is written as $(\cdot\,?\,\cdot : \cdot)$. The following example shows some (non-)well-formed constraints.

*Example 2.* The expressions 5, $p_{100}$, $(p_{10}\,?\,(2.1 \times \mathrm{a}_{12}) : 0)$, and $((((\mathrm{a}_{12} + (\sqrt{2} \times \mathrm{a}_{30})) + 7.2) > (0 - \mathrm{a}_5)) \wedge p_2)$ are well-formed whereas $-\mathrm{a}_{10}$ (unary minus) and $\mathrm{a} + 3$ (parenthesis missing) are not.

The binding precedence $\times \succ +, - \succ >, = \succ \neg \succ \vee, \wedge \succ \to, \leftrightarrow, (\cdot\,?\,\cdot : \cdot)$ allows to save parentheses. Furthermore the operators $+$, $\times$, $\vee$, $\wedge$, and $\leftrightarrow$ are left-associative while $-$ and $\to$ associate to the right. Taking these conventions into account the most complex constraint from the previous example simplifies to $\mathrm{a}_{12} + \sqrt{2} \times \mathrm{a}_{30} + 7.2 > 0 - \mathrm{a}_5 \ \wedge \ p_2$. To obtain smaller constraints already at the time of encoding trivial simplifications like $\varphi \wedge \top \to \varphi$, $\varphi \wedge \bot \to \bot$, $\cdots$ are performed.

In Appendix A we show how to mimic arithmetic over $\mathbb{N}$ and $\mathbb{Z}$ in SAT. Similar encodings have been presented (either for fixed bit width or for non-negative numbers only) in [13,14,23,31]. To our knowledge the two's complement

encoding taking overflows into account is new. In the remainder of this section we show how arithmetic over $\mathbb{Q}$ and (a fragment of) $\mathbb{R}$ can be reduced to the integral case. By $\mathbf{N}$, $\mathbf{Z}$, $\mathbf{Q}$, and $\mathbf{R}$ we denote the encodings of numbers from $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$, respectively. To clarify the domain we index operations by these sets whenever confusion can arise.

### 2.1 Rational Arithmetic

Rational numbers are represented as a pair consisting of the numerator and denominator similar as in [15]. The numerator is a bit-vector representing an integer (compared to a natural number in [15]) whereas the denominator is a positive integer (negative denominators would demand a case analysis for $>_{\mathbf{Q}}$). We also experimented with a fixed point representation, yielding slightly worse performance and less flexibility. All operations with the exception of $\times_{\mathbf{Q}}$ require identical denominators. This can easily be established by expanding the fractions beforehand (as demonstrated in Example 4).

Comparisons are performed just on the numerators if the denominators coincide. The operations $+_{\mathbf{Q}}$, $-_{\mathbf{Q}}$, and $\times_{\mathbf{Q}}$ are inspired from arithmetic over fractions.

**Definition 3.** *For* $(\mathbf{a}, d)$*,* $(\mathbf{b}, d)$*, and* $(\mathbf{b}, d')$ *representing rationals we define:*

$$\begin{aligned}
(\mathbf{a}, d) >_{\mathbf{Q}} (\mathbf{b}, d) &:= \mathbf{a} >_{\mathbf{Z}} \mathbf{b} \\
(\mathbf{a}, d) =_{\mathbf{Q}} (\mathbf{b}, d) &:= \mathbf{a} =_{\mathbf{Z}} \mathbf{b} \\
(\mathbf{a}, d) +_{\mathbf{Q}} (\mathbf{b}, d) &:= (\mathbf{a} +_{\mathbf{Z}} \mathbf{b}, d) \\
(\mathbf{a}, d) -_{\mathbf{Q}} (\mathbf{b}, d) &:= (\mathbf{a} -_{\mathbf{Z}} \mathbf{b}, d) \\
(\mathbf{a}, d) \times_{\mathbf{Q}} (\mathbf{b}, d') &:= (\mathbf{a} \times_{\mathbf{Z}} \mathbf{b}, d \times d')
\end{aligned}$$

Next we demonstrate addition.

*Example 4.* Consider $\frac{3}{2} +_{\mathbb{Q}} \frac{-1}{4} = \frac{5}{4}$. In the sequence below first both denominators are made equal. Then addition of the numerators is performed using $+_{\mathbf{Z}}$ (see Appendix A for an explanation of the notation):

$$\begin{aligned}
(\langle \bot, \top, \top \rangle, 2) +_{\mathbf{Q}} (\langle \top, \top \rangle, 4) &= (\langle \bot, \top, \top, \bot \rangle, 4) +_{\mathbf{Q}} (\langle \top, \top \rangle, 4) \\
&= (\langle \bot, \top, \top, \bot \rangle +_{\mathbf{Z}} \langle \top, \top \rangle, 4) = (\langle \bot, \bot, \top, \bot, \top \rangle, 4)
\end{aligned}$$

We conclude this subsection by introducing a concept that drastically improves performance of rational arithmetic. Consider the following computation where (intermediate) results are not canceled:

$$\left(\frac{1}{2} \times \frac{4}{2}\right) \times \frac{3}{2} + \frac{1}{2} \;=\; \frac{4}{4} \times \frac{3}{2} + \frac{1}{2} \;=\; \frac{12}{8} + \frac{1}{2} \;=\; \frac{16}{8} \tag{1}$$

Exactly this happens in the implementation since there the numerator is a bit-vector consisting of Boolean formulas. Hence its concrete value is unknown and

no cancellation is possible. We propose the following elegant escape which is very easy to implement and has positive effects on run-times (as shown in Section 3). We force that a fraction is canceled if the denominator exceeds some given limit. Computation (2) shows the positive aspects of this heuristic by allowing a denominator of at most 2:

$$\left(\frac{1}{2} \times \frac{4}{2}\right) \times \frac{3}{2} + \frac{1}{2} \;=\; \frac{2}{2} \times \frac{3}{2} + \frac{1}{2} \;=\; \frac{3}{2} + \frac{1}{2} \;=\; \frac{4}{2} \tag{2}$$

After every addition or multiplication the fraction is canceled whenever the denominator exceeds 2. The negative aspects become apparent if the denominator is chosen too small. Then some computations can no longer be performed, e.g., when allowing a denominator of 1, computation (3) gets stuck in the second step since $\frac{3}{2}$ cannot be canceled:

$$\left(\frac{1}{2} \times \frac{4}{2}\right) \times \frac{3}{2} + \frac{1}{2} \;=\; \frac{1}{1} \times \frac{3}{2} + \frac{1}{2} \;=\; \frac{?}{1} + \frac{1}{2} \tag{3}$$

In the implementation, canceling by two is achieved by dividing the denominator by two and dropping the least significant bit of the numerator while demanding that this bit evaluates to false. The latter is achieved by adding a suitable constraint. Hence in contrast to the example above, computations do not get stuck but may produce unsatisfiable formulas. In Section 3 we will see that this does not happen very frequently. Furthermore, there also the effectiveness of this very simple but efficient heuristic is demonstrated.

Some remarks are in order. Although our representation of rationals allows fractions like $\frac{\mathbf{a}}{3}$, choosing the denominators as multiples of two is beneficial. This allows to efficiently extend fractions to equal denominators by bit-shifting the numerators. Furthermore, the heuristic (canceling by two) is most effective for even denominators. Obviously the technique extends to different denominators in principle but division by two can again be performed by bit-shifting while division by, e.g., three cannot and hence is more costly. However, for termination analysis the main benefit of rationals is that some number between zero and one can be represented while the exact value of this number is usually not so important.

### 2.2 Extending Rational Arithmetic by Roots of Numbers

Arithmetic over irrational numbers is the most challenging. To allow a finite representation we only consider a subset of $\mathbb{R}$ using a pair $(\mathbf{c}, \mathbf{d})$ where $\mathbf{c}$ and $\mathbf{d}$ are numbers from $\mathbf{Q}$. Such a pair $(\mathbf{c}, \mathbf{d})$ has the intended semantics of $\mathbf{c} + \mathbf{d}\sqrt{2}$. But problems arise when comparing two abstract numbers. Therefore the definition of $>_{\mathbf{R}}$ given below is just an approximation of $>_{\mathbb{R}}$. The idea is to under-approximate $\mathbf{d}\sqrt{2}$ on the left-hand side while over-approximating it on the right-hand side. We under-approximate $\mathbf{d}\sqrt{2}$ by $(\mathbf{5}, 4) \times_{\mathbf{Q}} \mathbf{d}$ if $\mathbf{d}$ is not negative and by $(\mathbf{3}, 2) \times_{\mathbf{Q}} \mathbf{d}$ if $\mathbf{d}$ is negative.[1] The approach is justified since $\frac{5}{4} = 1.25 <_{\mathbb{R}} 1.41 \approx \sqrt{2}$ and

[1] We abbreviate numbers by denoting them in boldface, i.e., $\mathbf{5}$ represents $\langle \bot, \top, \bot, \top \rangle$ from $\mathbf{Z}$ and $(\langle \bot, \top, \bot, \top \rangle, 1)$ from $\mathbf{Q}$. The context clarifies which one is meant.

similarly $-\frac{3}{2} = -1.5 <_{\mathbb{R}} -1.41 \approx -\sqrt{2}$. Analogous reasoning yields the over-approximation. This trick allows to implement $>_{\mathbf{R}}$ (an approximation of $>_{\mathbb{R}}$) based on $>_{\mathbf{Q}}$ which can be expressed exactly (cf. Definition 3).

Next we formally define the under- and over-approximation of $\mathbf{a}\sqrt{2}$ based on $\mathbf{a}$ from $\mathbf{Q}$ depending on the sign (denoted $\mathrm{sign}(\mathbf{a})$ with obvious definition) using the if-then-else operator. Recall that sign $\top$ indicates negative numbers (cf. Appendix A).

**Definition 5.** *For a number* $\mathbf{a}$ *from* $\mathbf{Q}$ *we define:*

$$\begin{aligned} \mathrm{under}(\mathbf{a}) &:= (\mathrm{sign}(\mathbf{a})\,?\,(\mathbf{3},2) : (\mathbf{5},4)) \times_{\mathbf{Q}} \mathbf{a} \\ \mathrm{over}(\mathbf{a}) &:= (\mathrm{sign}(\mathbf{a})\,?\,(\mathbf{5},4) : (\mathbf{3},2)) \times_{\mathbf{Q}} \mathbf{a} \end{aligned}$$

Using the under- and over-approximations we define $>_{\mathbf{R}}$ and $=_{\mathbf{R}}$. Note that since $>_{\mathbf{R}}$ is just an approximation, it may not appear at negative positions in Boolean formulas (which is not the case for the benchmarks we consider) and designing suitable approximations for $>_{\mathbf{R}}$ at negative positions is easy.

**Definition 6.** *For pairs* $(\mathbf{c},\mathbf{d})$ *and* $(\mathbf{e},\mathbf{f})$ *from* $\mathbf{R}$ *we define:*

$$\begin{aligned} (\mathbf{c},\mathbf{d}) >_{\mathbf{R}} (\mathbf{e},\mathbf{f}) &:= \mathbf{c} +_{\mathbf{Q}} \mathrm{under}(\mathbf{d}) >_{\mathbf{Q}} \mathbf{e} +_{\mathbf{Q}} \mathrm{over}(\mathbf{f}) \\ (\mathbf{c},\mathbf{d}) =_{\mathbf{R}} (\mathbf{e},\mathbf{f}) &:= \mathbf{c} =_{\mathbf{Q}} \mathbf{e} \wedge \mathbf{d} =_{\mathbf{Q}} \mathbf{f} \end{aligned}$$

For readability we unravel the pair notation in the sequel whenever useful, i.e., $(\mathbf{3},\mathbf{1})$ is identified with $3 +_{\mathbb{R}} \sqrt{2}$.

*Example 7.* The expression $(\mathbf{1},\mathbf{1}) >_{\mathbf{R}} (\mathbf{2},\mathbf{0})$ approximates $1+\sqrt{2} >_{\mathbb{R}} 2$. The $\sqrt{2}$ on the left-hand side is under-approximated by $\frac{5}{4}$ which allows to replace $>_{\mathbb{R}}$ by $>_{\mathbb{Q}}$. The resulting $1+\frac{5}{4} >_{\mathbb{Q}} 2$, i.e., $\frac{9}{4} >_{\mathbb{Q}} \frac{8}{4}$ shows that the above comparison is valid. Note that $(\mathbf{0},\mathbf{6}) >_{\mathbf{R}} (\mathbf{0},\mathbf{5})$ does not hold since obviously $6 \times \frac{5}{4} >_{\mathbb{Q}} 5 \times \frac{3}{2}$ evaluates to false.

The definitions for $+_{\mathbf{R}}$, $-_{\mathbf{R}}$, and $\times_{\mathbf{R}}$ are directly inspired from the intended semantics of pairs.

**Definition 8.** *For pairs* $(\mathbf{c},\mathbf{d})$ *and* $(\mathbf{e},\mathbf{f})$ *from* $\mathbf{R}$ *we define:*

$$\begin{aligned} (\mathbf{c},\mathbf{d}) +_{\mathbf{R}} (\mathbf{e},\mathbf{f}) &:= (\mathbf{c} +_{\mathbf{Q}} \mathbf{e}, \mathbf{d} +_{\mathbf{Q}} \mathbf{f}) \\ (\mathbf{c},\mathbf{d}) -_{\mathbf{R}} (\mathbf{e},\mathbf{f}) &:= (\mathbf{c} -_{\mathbf{Q}} \mathbf{e}, \mathbf{d} -_{\mathbf{Q}} \mathbf{f}) \\ (\mathbf{c},\mathbf{d}) \times_{\mathbf{R}} (\mathbf{e},\mathbf{f}) &:= (\mathbf{c} \times_{\mathbf{Q}} \mathbf{e} +_{\mathbf{Q}} \mathbf{2} \times_{\mathbf{Q}} \mathbf{d} \times_{\mathbf{Q}} \mathbf{f}, \mathbf{c} \times_{\mathbf{Q}} \mathbf{f} +_{\mathbf{Q}} \mathbf{d} \times_{\mathbf{Q}} \mathbf{e}) \end{aligned}$$

The next example demonstrates addition and multiplication for reals.

*Example 9.* The equality $(\mathbf{1},\mathbf{2}) +_{\mathbf{R}} (\mathbf{5},\mathbf{3}) = (\mathbf{6},\mathbf{5})$ is justified since the left-hand side represents the calculation $1+2\sqrt{2}+5+3\sqrt{2}$ which simplifies to $6+5\sqrt{2}$ corresponding to the right-hand side. The product $(\mathbf{1},\mathbf{2}) \times_{\mathbf{R}} (\mathbf{5},\mathbf{3}) = (\mathbf{17},\mathbf{13})$ is justified by $(1+2\sqrt{2}) \times (5+3\sqrt{2}) = 5+10\sqrt{2}+3\sqrt{2}+6\sqrt{2}\sqrt{2} = 17+13\sqrt{2}$.

A natural question is if the approach from this section can be extended to a larger fragment of the reals. Before the limitations of the approach are discussed we mention possible generalizations. Triples $(\mathbf{c}, \mathbf{d}, \mathbf{n})$ with $\mathbf{c}, \mathbf{d} \in \mathbf{Q}$ and $\mathbf{n} \in \mathbf{N}$ (i.e., $\mathbf{n}$ is a variable taking non-negative integral values) allow to represent numbers of the shape $\mathbf{c} + \mathbf{d}\sqrt{\mathbf{n}}$. Adapting the constant factor for multiplication in Definition 8 and providing suitable under- and over-approximations for $\sqrt{\mathbf{n}}$ allows to replace $\sqrt{2}$ by a square root of some arbitrary natural number. But intrinsic to the approach is that the same (square) root must be used within all constraints to keep the triple representation of numbers. The reason is that e.g., $(\mathbf{a}, \mathbf{b}, \mathbf{n}) \times_{\mathbf{R}} (\mathbf{c}, \mathbf{d}, \mathbf{n}) = (\mathbf{ac} +_{\mathbf{R}} \mathbf{nbd}, \mathbf{ad} +_{\mathbf{R}} \mathbf{bc}, \mathbf{n})$ but in general there is no triple corresponding to $(\mathbf{a}, \mathbf{b}, \mathbf{n}) \times_{\mathbf{R}} (\mathbf{c}, \mathbf{d}, \mathbf{m})$ if $\mathbf{n}$ and $\mathbf{m}$ represent different numbers. Similar problems occur if non-square roots should be considered. Although the shape of real numbers allowed appears restricted at first sight, it suffices to prove the key system of [25] (see Example 10).

## 3 Experimental Evaluation

In the experiments[2] we considered the 470 problems from the quantifier-free non-linear integer arithmetic benchmarks (QF_NIA) of SMT-LIB 2009[3] and the 1391 TRSs in the termination problems database (TPDB) version 5.0 (available via `http://termination-portal.org/wiki/TPDB`). All tests have been performed on a server equipped with 8 dual-core AMD Opteron® processors 885 running at a clock rate of 2.6 GHz and 64 GB of main memory. Unless stated otherwise only a single core of the server was used.

We implemented the approach presented in Appendix A and Section 2 and used MiniSat [11] as back-end (after a satisfiability preserving transformation to CNF [28]). The result, called MiniSmt, accepts the SMT-LIB syntax for quantifier-free non-linear arithmetic. For a comparison of MiniSmt with other SMT solvers see Section 3.1. We also integrated matrix interpretations as presented in Appendix B in the termination prover T$_{\text{T}}$T$_2$ [22] based on the constraint language of Definition 1. The constraints within T$_{\text{T}}$T$_2$ are solved with an interfaced version of MiniSmt. Experiments are discussed in Section 3.2.

### 3.1 Comparison with SMT Solvers

First we compare MiniSmt with other recent SMT solvers. Since 2009 the QF_NIA category is part of SMT-COMP in which Barcelogic [27] and CVC3 [5] participated. The results when comparing these tools on the QF_NIA benchmarks of SMT-LIB are given in Table 1. The column labeled yes (no) counts how many systems could be proved (un)satisfiable while time indicates the total time needed by the tool in seconds. A "–" indicates that the solver does not support the corresponding setting. If no answer was produced within 60 seconds

[2] See `http://cl-informatik.uibk.ac.at/ttt2/arithmetic` for full details.
[3] See `http://www.smtcomp.org/2009` for information on SMT-COMP and SMT-LIB.

**Table 1.** SMT solvers on 470 problems from SMT-LIB

| | yes | no | time | t/o |
|---|---|---|---|---|
| Barcelogic | 266 | 189 | 1188 | 15 |
| CVC3 | 113 | 139 | 13169 | 218 |
| MiniSmt(sat) | 267 | – | 6427 | 54 |
| MiniSmt(bv) | 268 | – | 3190 | 42 |
| $\sum$ | 269 | 194 | | |

**Table 2.** Statistics on various benchmarks

| | # | #var | | #add | | #mul | |
|---|---|---|---|---|---|---|---|
| | | avg | max | avg | max | avg | max |
| SMT-LIB (calypto) | 303 | 10 | 50 | 6 | 36 | 6 | 33 |
| SMT-LIB (leipzig) | 167 | 301 | 2606 | 113 | 1136 | 164 | 1420 |
| SMT-LIB (calypto + leipzig) | 470 | 113 | 2606 | 44 | 1136 | 62 | 1420 |
| matrices (dimension 1) | 1391 | 25 | 811 | 145 | 5824 | 226 | 10688 |
| matrices (dimension 2) | 1391 | 78 | 2726 | 1420 | 164276 | 1863 | 164452 |

the execution is killed (column t/o). The row labeled $\sum$ shows the accumulative yes (no) score for the corresponding column. In Table 1 MiniSmt makes use of the multi-core architecture of the server and searches for satisfying assignments based on two different settings. Instances where small domains suffice are handled by the configuration which uses 3 bits for arithmetic variables and 4 bits for intermediate results. The second setting employs 33 and 50 bits, respectively. As an alternative to the SAT back-end (denoted MiniSmt(sat)) in MiniSmt we also developed a transformation that allows to use SMT solvers for bit-vector logic to solve arithmetic constraints (called MiniSmt(bv)). Although bit-vector arithmetic cannot be used blindly for our setting—it does not take overflows into account and hence can produce unsound results—it can be adapted for solving non-linear arithmetic by sign-extension operations. Given the details in Appendix A this transformation is straightforward to implement. As a back-end for MiniSmt(bv) we use Yices [10] as designated SMT solver for bit-vector logic. As can be inferred from Table 1 even when dealing with large numbers MiniSmt performs competitively, i.e., it solves the most satisfiable instances. MiniSmt(bv) finds models for the problems calypto/problem-006547.cvc.1, leipzig/term-gZE9f0, and leipzig/term-lFYv5w while Barcelogic finds a model for leipzig/term-BKc7xf which MiniSmt(bv) misses. MiniSmt(sat) cannot handle leipzig/term-XbWQfu in contrast to its bit-vector pendant.

Since the SMT-LIB benchmarks consider only the integers as domain we also generated (with T$_{\text{T}}$T$_2$) typical constraints from termination analysis. More precisely we generated for every TRS a constraint that is satisfiable if and only if a direct proof with matrix interpretations over a non-negative carrier of a fixed dimension removes at least one rewrite rule.[4] This constraint is then solved

[4] Our benchmarks are available from the URL in Footnote 2.

**Table 3.** SMT solvers on 1391 matrix constraints (dimension 1)

| | $\mathbb{N}$ | | | | $\mathbb{Z}$ | | | | $\mathbb{Q}$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | yes | no | time | t/o | yes | no | time | t/o | yes | no | time | t/o |
| Barcelogic | 335 | 3 | 23k | 172 | 414 | 3 | 16k | 95 | – | – | – | – |
| CVC3 | 168 | 415 | 45k | 748 | 197 | 380 | 45k | 754 | 120 | 409 | 48k | 802 |
| MiniSmt(sat) | 337 | – | 1701 | 5 | 539 | – | 7279 | 40 | 337 | – | 1592 | 3 |
| MiniSmt(bv) | 337 | – | 902 | 5 | 553 | – | 1387 | 8 | 337 | – | 1258 | 6 |
| nlsol | 332 | – | 3203 | 14 | 479 | – | 8158 | 83 | 333 | – | 3924 | 20 |
| $\sum$ | 338 | 415 | | | 553 | 380 | | | 338 | 409 | | |

**Table 4.** SMT solvers on 1391 matrix constraints (dimension 2)

| | $\mathbb{N}$ | | | | $\mathbb{Z}$ | | | | $\mathbb{Q}$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | yes | no | time | t/o | yes | no | time | t/o | yes | no | time | t/o |
| Barcelogic | 408 | 3 | 41k | 578 | 832 | 3 | 17k | 204 | – | – | – | – |
| CVC3 | 117 | 130 | 66k | 1084 | 112 | 84 | 68k | 1135 | 58 | 125 | 69k | 1147 |
| MiniSmt(sat) | 402 | – | 7193 | 63 | 995 | – | 23k | 214 | 407 | – | 6505 | 60 |
| MiniSmt(bv) | 405 | – | 5248 | 57 | 1068 | – | 13k | 140 | 400 | – | 6418 | 69 |
| nlsol | 301 | – | 18k | 190 | 454 | – | 51k | 771 | 289 | – | 20k | 240 |
| $\sum$ | 412 | 130 | | | 1142 | 84 | | | 409 | 125 | | |

over various domains, to allow a comprehensive comparison with nlsol [6], a recent solver for polynomial arithmetic, which follows a similar approach as Barcelogic but in addition handles non-integral domains. Our benchmarks are in the QF_NIA syntax but are of a different structure than the SMT-LIB instances. Specifically, our problems admit more arithmetic operations while typically having less variables. In Table 2 some statistics and comparisons regarding the different test benches are given. There the column # indicates the number of systems in the respective benchmark family and the other columns give accumulated information on the size (i.e., number of variables, additions, and multiplications) of the problems. Since nlsol requires a slightly different input format our benchmarks are preprocessed for this tool.

Table 3 presents the results for the matrix benchmarks of dimension one. Times postfixed with "k" should be multiplied by a factor of 1000. For the solvers nlsol and MiniSmt variables range over the domain $\{0, \ldots, 15\}$ ($\mathbb{N}$), $\{-16, \ldots, 15\}$ ($\mathbb{Z}$), and $\{\frac{0}{2}, \frac{1}{2}, \frac{2}{2}, \ldots, \frac{15}{2}\}$ ($\mathbb{Q}$). Table 4 considers the benchmarks with matrices of dimension two. Since these constraints are much larger, MiniSmt and nlsol use one bit less for representing numbers. We also considered different domains which produced similar results. In Tables 3 and 4 only the benchmarks considering the domains $\mathbb{N}$ and $\mathbb{Q}$ correspond to valid (parts of) termination proofs. The reason for including the $\mathbb{Z}$ benchmarks in the tables is that they allow the bit-vector back-end of MiniSmt to show its performance best.

We note that nlsol allows more flexibility in choosing the variable domain since MiniSmt bounds variables by powers of two. However our approach admits more freedom in bounding intermediate results which reduces the search space

**Table 5.** Matrices with dependency pairs for 1391 TRSs

| | $1 \times 1$ | | | $2 \times 2$ | | | $3 \times 3$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | yes | time | t/o | yes | time | t/o | yes | time | t/o | $\sum$ |
| $\mathbb{N}$ | 545 | 8885 | 83 | 618 | 23820 | 326 | 627 | 25055 | 349 | 659 |
| $\mathbb{Q}$ | 599 | 8574 | 67 | 597 | 20238 | 261 | 496 | 19490 | 252 | 638 |
| $\mathbb{Q}_1$ | 606 | 5906 | 46 | 655 | 15279 | 173 | 643 | 14062 | 164 | 685 |
| $\mathbb{Q}_2$ | 627 | 10109 | 93 | 651 | 23102 | 308 | 619 | 23806 | 330 | 687 |
| $\mathbb{R}$ | 535 | 17029 | 198 | 630 | 16517 | 200 | 599 | 29346 | 415 | 648 |
| $\sum$ | 639 | | | 674 | | | 664 | | | 703 |

(e.g. for columns $\mathbb{Q}$ in Tables 3 and 4 we require that intermediate results are integers) which results in efficiency gains. For the sake of a fair comparison we configured our tool such that arithmetic variables are represented in as many bits as intermediate results. However, usually it is a good idea to allow more bits for intermediate results.

We summarize the tables with the following observations: (a) MiniSmt is surprisingly powerful on the SMT-LIB benchmarks (containing few multiplications but large numbers, i.e., requiring more than 30 bits), (b) our tool performs best (note its speed) on the matrix benchmarks (containing many multiplications and usually small domains suffice), (c) MiniSmt is by far the most powerful tool on rational domains, (d) while in general the SMT back-end of MiniSmt is favorable, for column $\mathbb{Q}$ in Table 4 the SAT back-end shows more problems satisfiable than the SMT counterpart, and (e) CVC3 could be used to cancel termination proof attempts early due to its power concerning unsatisfiability.

Our tool fills two gaps that current SMT solvers admit. It is fastest on small and rational domains and to our knowledge the only solver that efficiently supports irrational domains, e.g., only our tool can solve the constraint $2 = x \times x$ for a real-valued variable $x$. Due to a lack of interesting benchmarks and competitor tools, MiniSmt cannot show its full strength here.

### 3.2 Evaluation within a Termination Prover

Next we compare matrix interpretations over $\mathbb{N}$, $\mathbb{Q}$, and $\mathbb{R}$ (cf. Appendix B) and show that MiniSmt admits a fast automation of the method. The coefficients of a matrix over dimension $d$ are represented in $\max\{2, 5-d\}$ bits (for reals we allow $\max\{1, 3-d\}$ bits due to the more expensive pair representation). Every rational coefficient is represented as a fraction with denominator two. Hence a matrix of dimension two admits natural coefficients $\{0, 1, \ldots, 7\}$, rational coefficients $\{0, \frac{1}{2}, 1, 1\frac{1}{2}, 2, 2\frac{1}{2}, 3, 3\frac{1}{2}\}$, and real coefficients $\{0, 1, \sqrt{2}, 1+\sqrt{2}\}$. The number of bits for representing intermediate computations was chosen to be one more than the number of bits allowed for the coefficients. Restricting the bit-width is essential for performance, especially for larger dimensions. It is well-known that for interpretation based termination criteria usually small coefficients suffice.

In Table 5 matrices of dimensions one to three are considered. The rows labeled $\mathbb{N}$ indicate that only natural numbers are allowed as coefficients whereas

$\mathbb{Q}$ refers to the naive representation of rationals without canceling the fractions and $\mathbb{R}$ to the subset of real coefficients mentioned above. The rows $\mathbb{Q}_n$ indicate that a fraction is canceled if its denominator exceeds $n$. The column labeled yes shows the number of successful termination proofs while time indicates the total time needed by the tool in seconds. If no answer was produced within 60 seconds the execution is killed (column t/o). The row (column) labeled $\sum$ shows the accumulative yes score for the corresponding column (row).

Matrix interpretations over $\mathbb{N}$ are used by most contemporary termination tools and serve as a reference. The performance of $\mathbb{Q}$ is satisfactory for matrices with dimension one (which correspond to linear polynomial interpretations and confirms the results in [15]) but poor for larger dimensions. In contrast, the overall performance of $\mathbb{Q}_1$ is excellent, i.e., it is much faster than $\mathbb{N}$ and more powerful. The combination of all 15 methods from Table 5 together can prove 703 systems terminating, yielding a gain of almost 50 systems compared to the standard setting allowing natural coefficients only. This number is remarkable since Jambox [12]—a powerful termination prover based on various termination criteria—proved 750 systems terminating in the 2008 competition and took 3rd place. Since the competition execution software allows to run 16 processes in parallel the (single!) method we propose is a good starting point for new termination analyzers. Looking beyond TPDB, our implementation also shows its strength for real coefficients. It masters the TRS $\mathcal{R}_{\mathbb{R}}$ from Example 10 below. This system stems from [25] where it was proved that no direct termination proof based on polynomial interpretations over the natural or rational numbers can exist which orients all rules strictly. However a proof over the reals is possible and our implementation finds such a proof fully automatically. Due to the statement "... only the techniques [...] which concern *non-negative rational numbers* have been included in Mu-term ..." in [26], we believe that T$_{\text{T}}$T$_2$ is the only automatic termination analyzer for TRSs that supports reasoning about irrational domains.

*Example 10.* For the TRS $\mathcal{R}_{\mathbb{R}}$ from [25] consisting of the seven rules

$$\begin{aligned}
\mathsf{k}(x,x,\mathsf{b}_1) &\to \mathsf{k}(\mathsf{g}(x),\mathsf{b}_2,\mathsf{b}_2) & \mathsf{g}(\mathsf{c}(x)) &\to \mathsf{f}(\mathsf{c}(\mathsf{f}(x)))\\
\mathsf{k}(x,\mathsf{a}_2,\mathsf{b}_1) &\to \mathsf{k}(\mathsf{a}_1,x,\mathsf{b}_1) & \mathsf{f}(\mathsf{f}(x)) &\to \mathsf{g}(x)\\
\mathsf{k}(\mathsf{a}_4,x,\mathsf{b}_1) &\to \mathsf{k}(x,\mathsf{a}_3,\mathsf{b}_1) & \mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{f}(x)))) &\to \mathsf{k}(x,x,x)\\
\mathsf{k}(\mathsf{g}(x),\mathsf{b}_3,\mathsf{b}_3) &\to \mathsf{k}(x,x,\mathsf{b}_4)
\end{aligned}$$

T$_{\text{T}}$T$_2$ finds the following interpretation that orients all rules strictly

$$\begin{aligned}
\mathsf{a}_{1\mathbb{R}} &= 0 & \mathsf{b}_{1\mathbb{R}} &= 2+\sqrt{2} & \mathsf{f}_{\mathbb{R}}(x) &= \sqrt{2}x+\sqrt{2}\\
\mathsf{a}_{2\mathbb{R}} &= 1+2\sqrt{2} & \mathsf{b}_{2\mathbb{R}} &= 0 & \mathsf{g}_{\mathbb{R}}(x) &= 2x+1+\sqrt{2}\\
\mathsf{a}_{3\mathbb{R}} &= 0 & \mathsf{b}_{3\mathbb{R}} &= 1+\sqrt{2} & \mathsf{c}_{\mathbb{R}}(x) &= x+1+2\sqrt{2}\\
\mathsf{a}_{4\mathbb{R}} &= 1+\sqrt{2} & \mathsf{b}_{4\mathbb{R}} &= \sqrt{2} & \mathsf{k}_{\mathbb{R}}(x,y,z) &= x+y+\sqrt{2}z+3\sqrt{2}
\end{aligned}$$

within a fraction of a second. While a direct proof with polynomials over $\mathbb{N}$ is not possible, natural coefficients suffice in the dependency pair setting (after

computing the SCCs of the dependency graph). Hence all modern termination tools can prove this system terminating.

## 4 Related Work and Concluding Remarks

First we discuss related approaches for solving non-linear arithmetic. Barcelogic follows [6] where constraints are linearized by assigning a finite domain to variables occurring in non-linear constraints. The resulting linear arithmetic formula is solved by a variant of the simplex algorithm. In contrast to the work in [6] which mentions heuristics (decide which variables should be used for the linearization) our approach does not require such considerations. However both approaches require some fixed domain for the variables. CVC3 implements a Fourier-Motzkin procedure for linear arithmetic while treating non-linear terms as if they were linear (Dejan Jovanović, personal conversation, 2009). The last tool we considered for comparison, nlsol, uses a similar approach as Barcelogic but also supports non-integral domains. It transforms non-linear constraints to linear arithmetic before it calls Yices as back-end. The difference to MiniSmt(bv) is that nlsol employs Yices for solving linear arithmetic whereas our tool uses it for bit-vector arithmetic. We are aware of the fact that the first order theory of real arithmetic is decidable [29] but because of the underlying computational complexity of the method the result is mainly of theoretical interest. Improvements of the original procedure are still of double exponential time complexity [8]. Nevertheless it might be interesting to investigate how this worst case complexity affects the performance for applications. Note that SAT solving techniques and the simplex method admit exponential time worst case complexity but are surprisingly efficient in practice.

Next we discuss related work on matrix interpretations. An extension to rational domains was already proposed in 2007 [16] (for termination proofs of string rewrite systems) where evolutionary algorithms [3] were suggested to find suitable rational coefficients. However, no benchmarks are given there that show a gain in power. In [15] polynomial interpretations are extended to rational coefficients. This work is related since linear polynomial interpretations coincide with matrix interpretations of dimension one. Our experiments confirm the gains in power when using matrices of dimension one but the method from [15] results in a poor performance for larger dimensions without further ado. Independently to our research, [1] extends the theory of matrix interpretations to coefficients over the reals. However their (preliminary) implementation can only deal with rationals. Furthermore no benchmarks are given in [1] showing any gains in power by allowing rationals. Hence our contribution for the first time gives evidence that matrix interpretations over the non-negative reals do really extend the power of termination criteria in practice. Recently non-linear matrix interpretations have been introduced [9] by considering matrix domains instead of vector domains. We would like to investigate if this more general setting can also benefit from rational domains.

Before we conclude this section with ideas for future work we mention one specialty of MiniSmt. Due to the bottom-up representation of domains (naturals, integers, rationals, reals) our solver can be used for instances that require arithmetic variables of different types. This distinguishes MiniSmt from the other solvers that do currently not support such problems appropriately. In the future we would like to add support for reasoning about unsatisfiable instances. As an immediate consequence this would improve the cumulative execution time of MiniSmt and as a side effect this would also be beneficial for termination analysis of rewriting; a termination proof can be aborted immediately if the corresponding constraints are unsatisfiable and a different termination criterion can be considered. Another extension aims at improving the handling of real domains. Instead of restricting to approximations of $\sqrt{2}$ one could consider $\sqrt{\mathbf{n}}$ where $\mathbf{n}$ is some abstract expression representing a non-negative integer (as discussed at the end of Section 2.2). Moreover, allowing $\mathbf{n}$ to be negative admits reasoning about complex domains. However, we are not aware of any termination criteria that require such a domain.

**Acknowledgments.** We thank Nikolaj Bjørner for encouraging us to investigate the bit-vector back-end, René Thiemann for pointing out a bug, and the anonymous referees for numerous suggestions that helped to improve the presentation.

## References

1. Alarcón, B., Lucas, S., Navarro-Marset, R.: Proving termination with matrix interpretations over the reals. In: WST 2009. pp. 12–15 (2009)
2. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. TCS 236(1-2), 133–178 (2000)
3. Ashlock, D.: Evolutionary Computation for Modeling and Optimization. Springer (2006)
4. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)
5. Barrett, C., Tinelli, C.: CVC3. In: CAV 2007. LNCS, vol. 4590, pp. 298–302 (2007)
6. Borralleras, C., Lucas, S., Navarro-Marset, R., Rodriguez-Carbonell, E., Rubio, A.: Solving non-linear polynomial arithmetic via SAT modulo linear arithmetic. In: CADE 2009. LNCS (LNAI), vol. 5663, pp. 294–305 (2009)
7. Codish, M., Lagoon, V., Stuckey, P.: Solving partial order constraints for LPO termination. JSAT 5, 193–215 (2008)
8. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Proc. 2nd International Conference on Automata Theory and Formal Languages 1975. LNCS, vol. 33, pp. 134–183 (1975)
9. Courtieu, P., Gbedo, G., Pons, O.: Improved matrix interpretation. In: SOFSEM 2010. LNCS, vol. 5901, pp. 283–295 (2010)
10. Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for DPLL(T). In: CAV 2006. LNCS, vol. 4144, pp. 81–94 (2006)
11. Eén, N., Sörensson, N.: An extensible SAT-solver. In: SAT 2004. LNCS, vol. 2919, pp. 502–518 (2004)

12. Endrullis, J.: (Jambox). Available from `http://joerg.endrullis.de`.
13. Endrullis, J., Waldmann, J., Zantema, H.: Matrix interpretations for proving termination of term rewriting. JAR 40(2-3), 195–220 (2008)
14. Fuhs, C., Giesl, J., Middeldorp, A., Schneider-Kamp, P., Thiemann, R., Zankl, H.: SAT solving for termination analysis with polynomial interpretations. In: SAT 2007. LNCS, vol. 4501, pp. 340–354 (2007)
15. Fuhs, C., Navarro-Marset, R., Otto, C., Giesl, J., Lucas, S., Schneider-Kamp, P.: Search techniques for rational polynomial orders. In: AISC 2008. LNCS (LNAI), vol. 5144, pp. 109–124 (2008)
16. Gebhardt, A., Hofbauer, D., Waldmann, J.: Matrix evolutions. In: WST 2007. pp. 4–8 (2007)
17. Giesl, J., Thiemann, R., Schneider-Kamp, P., Falke, S.: Mechanizing and improving dependency pairs. JAR 37(3), 155–203 (2006)
18. Gulwani, S., Tiwari, A.: Constraint-based approach for analysis of hybrid systems. In: CAV 2008. LNCS, vol. 5123, pp. 190–203 (2008)
19. Hirokawa, N., Middeldorp, A.: Automating the dependency pair method. I&C 199(1-2), 172–199 (2005)
20. Hofbauer, D., Waldmann, J.: Termination of string rewriting with matrix interpretations. In: RTA 2006. LNCS, vol. 4098, pp. 328–342 (2006)
21. Hofbauer, D.: Termination proofs by context-dependent interpretations. In: RTA 2001. LNCS, vol. 2051, pp. 108–121 (2001)
22. Korp, M., Sternagel, C., Zankl, H., Middeldorp, A.: Tyrolean Termination Tool 2. In: RTA 2009. LNCS, vol. 5595, pp. 295–304 (2009)
23. Kroening, D., Strichman, O.: Decision Procedures. Springer (2008)
24. Lucas, S.: Polynomials over the reals in proofs of termination: From theory to practice. TIA 39(3), 547–586 (2005)
25. Lucas, S.: On the relative power of polynomials with real, rational, and integer coefficients in proofs of termination of rewriting. AAECC 17(1), 49–73 (2006)
26. Lucas, S.: Practical use of polynomials over the reals in proofs of termination. In: PPDP 2007. pp. 39–50 (2007)
27. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: from an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). J. ACM 53(6), 937–977 (2006)
28. Plaisted, D., Greenbaum, S.: A structure-preserving clause form translation. JSC 2(3), 293–304 (1986)
29. Tarski, A.: A Decision Method for Elementary Algebra and Geometry. 2nd edn. University of California Press, Berkeley (1957)
30. Zankl, H.: Lazy Termination Analysis. PhD thesis, University of Innsbruck (2009)
31. Zankl, H., Hirokawa, N., Middeldorp, A.: KBO orientability. JAR 43(2), 173–201 (2009)
32. Zantema, H.: Termination. In: TeReSe (ed.) Term Rewriting Systems. Cambridge University Press, Cambridge 181–259 (2003)

## A Encoding Non-Linear Integral Arithmetic in SAT

In this section arithmetic constraints (cf. the grammar in Definition 1) are reduced to SAT. To obtain formulas of finite size, every arithmetic variable is represented by a given number of bits. Then operations such as $+_{\mathbf{N}}$ and $\times_{\mathbf{N}}$ are unfolded according to their definitions using circuits. Such definitions for

$+_{\mathbf{N}}$ and $\times_{\mathbf{N}}$ have already been presented in [13] for bit-vectors of a fixed width. In contrast, we take overflows into account. The encodings of $>_{\mathbf{N}}$ and $=_{\mathbf{N}}$ for bit-vectors given below are similar to the ones in [7].

### A.1 Arithmetic over $\mathbb{N}$

We fix the number $k$ of bits that is available for representing natural numbers in binary. Let $a < 2^k$. We denote by $\mathbf{a}_k = \langle a_k, \ldots, a_1 \rangle$ the binary representation of $a$ where $a_k$ is the most significant bit. Hence e.g. $\langle \top, \top, \bot \rangle = 6$. Whenever $k$ is not essential we abbreviate $\mathbf{a}_k$ to $\mathbf{a}$. Furthermore the operation $(\cdot)_k$ on bit-vectors is used to drop bits, i.e., $\langle a_4, a_3, a_2, a_1 \rangle_2 = \langle a_2, a_1 \rangle$.

**Definition 11.** *For natural numbers in binary representation we define:*

$$\mathbf{a}_k >_{\mathbf{N}} \mathbf{b}_k \;:=\; \begin{cases} \bot & \text{if } k = 0 \\ (a_k \wedge \neg b_k) \vee \big((b_k \rightarrow a_k) \wedge \mathbf{a}_{k-1} >_{\mathbf{N}} \mathbf{b}_{k-1}\big) & \text{if } k > 0 \end{cases}$$

$$\mathbf{a}_k =_{\mathbf{N}} \mathbf{b}_k \;:=\; \bigwedge_{i=1}^{k} (a_i \leftrightarrow b_i)$$

Since two $k$-bit bit-vectors sum up to a $(k+1)$-bit number an additional bit is needed for the result. Hence the case arises when two summands are not of equal bit-width. Thus, before adding $\mathbf{a}_k$ and $\mathbf{b}_{k'}$ the shorter one is padded with $|k - k'|$ $\bot$'s. To keep the presentation simple we assume that $\bot$-padding is implicitly performed before the operations $+_{\mathbf{N}}$, $>_{\mathbf{N}}$, and $=_{\mathbf{N}}$.

**Definition 12.** *We define* $\mathbf{a}_k +_{\mathbf{N}} \mathbf{b}_k$ *as* $\langle c_k, s_k, \ldots, s_1 \rangle$ *for* $1 \leqslant i \leqslant k$ *with*

$$c_0 = \bot \qquad s_i = a_i \otimes b_i \otimes c_{i-1} \qquad c_i = (a_i \wedge b_i) \vee (a_i \wedge c_{i-1}) \vee (b_i \wedge c_{i-1})$$

*were* $\otimes$ *denotes exclusive or, i.e.,* $x \otimes y \;:=\; \neg(x \leftrightarrow y)$.

Note that in practice it is essential to introduce new variables for the carry and the sum since in consecutive additions each bit $a_i$ and $b_i$ is duplicated (twice for the carry and once for the sum). Using fresh variables for the sum prevents an exponential blowup of the resulting formula. A further method to keep formulas small is to limit the bit-width when representing naturals. This can be accomplished after addition (or multiplication) by fixing a maximal number $m$ of bits. To restrict $\mathbf{a}_k$ to $m$ bits we demand that all $a_i$ for $m+1 \leqslant i \leqslant k$ are $\bot$ as a side constraint. Then it is sound (i.e., restricting bits can result in unsatisfiable formulas but never produce models for unsatisfiable input) to continue any computations with $\mathbf{a}_m$ instead of $\mathbf{a}_k$.

The next example demonstrates addition. To ease readability we only use $\bot$ and $\top$ in the following examples and immediately simplify formulas.

*Example 13.* We compute $3 +_{\mathbb{N}} 14 = 17$. In the sequence below the first step performs $\bot$-padding. Afterwards Definition 12 applies.

$$\langle \top, \top \rangle +_{\mathbf{N}} \langle \top, \top, \top, \bot \rangle = \langle \bot, \bot, \top, \top \rangle +_{\mathbf{N}} \langle \top, \top, \top, \bot \rangle = \langle \top, \bot, \bot, \bot, \top \rangle$$

Multiplication is implemented by addition and bit-shifting. Here $\mathbf{a} \ll n$ denotes a left-shift of $\mathbf{a}$ by $n$ bits, e.g., $\langle x, y\rangle \ll 3$ yields $\langle x, y, \bot, \bot, \bot\rangle$. The operation $(\cdot)$ takes a bit-vector and a Boolean variable and performs scalar multiplication, i.e., $\mathbf{a}_k \cdot x = \langle a_k \wedge x, \ldots, a_1 \wedge x\rangle$. In the sequel the operator $(\cdot)$ binds stronger than $\ll$, i.e., $\mathbf{a} \cdot x \ll 2$ abbreviates $(\mathbf{a} \cdot x) \ll 2$.
The product of two bit-vectors with $m$ and $n$ bits has $m + n$ bits.

**Definition 14.** *For bit-vectors* $\mathbf{a}_m$ *and* $\mathbf{b}_n$ *we define:*

$$\mathbf{a}_m \times_{\mathbf{N}} \mathbf{b}_n \;:=\; \big((\mathbf{a}_m \cdot b_1 \ll 0) +_{\mathbf{N}} \cdots +_{\mathbf{N}} (\mathbf{a}_m \cdot b_n \ll (n-1))\big)_{m+n}$$

In the following example we demonstrate multiplication.

*Example 15.* Let $\mathbf{a} = \langle \top, \bot, \top\rangle$ and $\mathbf{b} = \langle \top, \top, \bot\rangle$, i.e., we compute $5 \times_{\mathbb{N}} 6 = 30$. The first step below unfolds Definition 14. Then the scalar multiplications are evaluated before shifting is performed. After addition (using $+_{\mathbf{N}}$) the sum is restricted to six bits.

$$\begin{aligned}
\mathbf{a} \times_{\mathbf{N}} \mathbf{b} &= \big((\mathbf{a} \cdot \bot \ll 0) +_{\mathbf{N}} (\mathbf{a} \cdot \top \ll 1) +_{\mathbf{N}} (\mathbf{a} \cdot \top \ll 2)\big)_6 \\
&= \big((\langle \bot, \bot, \bot\rangle \ll 0) +_{\mathbf{N}} (\mathbf{a} \ll 1) +_{\mathbf{N}} (\mathbf{a} \ll 2)\big)_6 \\
&= \big(\langle \bot, \bot, \bot\rangle +_{\mathbf{N}} \langle \top, \bot, \top, \bot\rangle +_{\mathbf{N}} \langle \top, \bot, \top, \bot, \bot\rangle\big)_6 = \langle \bot, \top, \top, \top, \top, \bot\rangle
\end{aligned}$$

If-then-else is an abbreviation, i.e., $x \,?\, \mathbf{a} : \mathbf{b} \;:=\; (\mathbf{a} \cdot x) +_{\mathbf{N}} (\mathbf{b} \cdot \neg x)$. This expression evaluates to $\mathbf{a}$ if $x$ is true and to $\mathbf{b}$ otherwise. We omit its straightforward redefinitions when considering arithmetic over $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$. Note that this operator can encode the maximum of two numbers, i.e., $\max(\mathbf{a}, \mathbf{b}) \;:=\; \mathbf{a} >_{\mathbf{N}} \mathbf{b} \,?\, \mathbf{a} : \mathbf{b}$.

### A.2 Arithmetic over $\mathbb{Z}$

We represent integers using two's complement which allows a straightforward encoding of arithmetic operations. For a $k$-bit number the most significant bit denotes the sign, e.g., $\mathbf{a}_k = \langle a_k, \ldots, a_1\rangle$ with sign $a_k$ and bits $a_{k-1}, \ldots, a_1$. Sign $\top$ indicates negative values. Again some definitions expect operands to be of equal bit-width. This is accomplished by implicitly sign-extending the shorter operand. The operation $(\cdot)_k$ is abused for both sign-extending and discarding bits, e.g., $\langle \bot, \top\rangle_4 = \langle \bot, \bot, \bot, \top\rangle$, $\langle \top, \top\rangle_4 = \langle \top, \top, \top, \top\rangle$, and $\langle \top, \top, \top\rangle_2 = \langle \top, \top\rangle$. The integer represented by the bit-vector does not change when sign-extending. Similar to the case for $\mathbf{N}$, a bit-vector $\mathbf{a}_k$ can be restricted to $m$ bits. If the dropped bits take the same value as the sign, then $\mathbf{a}_k$ and $\mathbf{a}_m$ denote the same number. Adding a side constraint $a_k \leftrightarrow a_i$ for $m \leqslant i \leqslant k$ allows to proceed with $\mathbf{a}_m$ instead of $\mathbf{a}_k$.

Comparisons are defined based on the corresponding operations over $\mathbf{N}$. For $>_{\mathbf{Z}}$ a separate check on the sign is needed, i.e., $\mathbf{a}$ is greater than $\mathbf{b}$ if $\mathbf{b}$ is negative while $\mathbf{a}$ is not and otherwise the bits are compared using $>_{\mathbf{N}}$. For $+_{\mathbf{Z}}$ and $\times_{\mathbf{Z}}$ numbers are first sign-extended before the corresponding operation over $\mathbf{N}$ is employed. Superfluous bits are discarded afterwards.

**Definition 16.** *The operations* $>_{\mathbf{Z}}$, $=_{\mathbf{Z}}$, $+_{\mathbf{Z}}$, *and* $\times_{\mathbf{Z}}$ *are defined as follows:*

$$\begin{aligned}
\mathbf{a}_k >_{\mathbf{Z}} \mathbf{b}_k &:= \big(\neg a_k \wedge b_k\big) \vee \big((a_k \to b_k) \wedge \mathbf{a}_{k-1} >_{\mathbf{N}} \mathbf{b}_{k-1}\big) \\
\mathbf{a}_k =_{\mathbf{Z}} \mathbf{b}_k &:= \mathbf{a}_k =_{\mathbf{N}} \mathbf{b}_k \\
\mathbf{a}_k +_{\mathbf{Z}} \mathbf{b}_k &:= (\mathbf{a}_{k+1} +_{\mathbf{N}} \mathbf{b}_{k+1})_{k+1} \\
\mathbf{a}_m \times_{\mathbf{Z}} \mathbf{b}_n &:= (\mathbf{a}_{m+n} \times_{\mathbf{N}} \mathbf{b}_{m+n})_{m+n}
\end{aligned}$$

Subtraction is encoded using addition and two's complement, i.e., $\mathbf{a} -_{\mathbf{Z}} \mathbf{b} := \mathbf{a} +_{\mathbf{Z}} \mathrm{tc}_{\mathbf{Z}}(\mathbf{b})$ with $\mathrm{tc}_{\mathbf{Z}}(\cdot)$ as defined below.

**Definition 17.** *For a bit-vector* $\mathbf{a}_k$ *we define* ones' *and* two's *complement as:*

$$\mathrm{oc}(\mathbf{a}_k) := \langle \neg a_k, \ldots, \neg a_1 \rangle \qquad \mathrm{tc}_{\mathbf{Z}}(\mathbf{a}_k) := \big(\mathrm{oc}(\mathbf{a}_{k+1}) +_{\mathbf{N}} \langle \top \rangle\big)_{k+1}$$

Ones' complement flips all bits and two's complement computes ones' complement incremented by one. To avoid a case distinction on the sign for two's complement the operand first is sign-extended by one auxiliary bit. After computing ones' complement, one is added and then the overflow bit is discarded as shown in the next example.

*Example 18.* Since $-2^k$ can be represented in $k$ bits but $2^k$ cannot, $\mathrm{tc}_{\mathbf{Z}}(\mathbf{a}_k)$ must have $k+1$ bits (recall that we take overflows into account). For two's complement of 0 it is essential to first sign-extend the operand and then restrict the result to $k+1$ bits. We demonstrate this with 0 represented by two bits, using an additional bit for the sign:

$$\begin{aligned}
\mathrm{tc}_{\mathbf{Z}}(\langle \bot, \bot, \bot \rangle) &= \big(\mathrm{oc}(\langle \bot, \bot, \bot \rangle_4) +_{\mathbf{N}} \langle \top \rangle\big)_4 = \big(\mathrm{oc}(\langle \bot, \bot, \bot, \bot \rangle) +_{\mathbf{N}} \langle \top \rangle\big)_4 \\
&= \big(\langle \top, \top, \top, \top \rangle +_{\mathbf{N}} \langle \top \rangle\big)_4 = \big(\langle \top, \bot, \bot, \bot, \bot \rangle\big)_4 = \langle \bot, \bot, \bot, \bot \rangle
\end{aligned}$$

Next we calculate two's complement of $-4$ which evaluates to 4:

$$\begin{aligned}
\mathrm{tc}_{\mathbf{Z}}(\langle \top, \bot, \bot \rangle) &= \big(\mathrm{oc}(\langle \top, \bot, \bot \rangle_4) +_{\mathbf{N}} \langle \top \rangle\big)_4 = \big(\mathrm{oc}(\langle \top, \top, \bot, \bot \rangle) +_{\mathbf{N}} \langle \top \rangle\big)_4 \\
&= \big(\langle \bot, \bot, \top, \top \rangle +_{\mathbf{N}} \langle \top \rangle\big)_4 = \big(\langle \bot, \bot, \top, \bot, \bot \rangle\big)_4 = \langle \bot, \top, \bot, \bot \rangle
\end{aligned}$$

The next example illustrates addition/subtraction and multiplication.

*Example 19.* We compute $5 -_{\mathbb{Z}} 2 = 3$. The sequence below translates subtraction ($-_{\mathbf{Z}}$) into addition ($+_{\mathbf{Z}}$) in the first step. Then two's complement of 2 is calculated. Afterwards addition for integers is performed by first sign-extending both operands by one additional bit and then performing addition for naturals ($+_{\mathbf{N}}$). After this step the superfluous carry bit is disregarded, i.e.,

$$\begin{aligned}
\langle \bot, \top, \bot, \top \rangle -_{\mathbf{Z}} \langle \bot, \top, \bot \rangle &= \langle \bot, \top, \bot, \top \rangle +_{\mathbf{Z}} \mathrm{tc}(\langle \bot, \top, \bot \rangle) \\
&= \langle \bot, \top, \bot, \top \rangle +_{\mathbf{Z}} \langle \top, \top, \top, \bot \rangle = \big(\langle \bot, \top, \bot, \top \rangle_5 +_{\mathbf{N}} \langle \top, \top, \top, \bot \rangle_5\big)_5 \\
&= \big(\langle \bot, \bot, \top, \bot, \top \rangle +_{\mathbf{N}} \langle \top, \top, \top, \top, \bot \rangle\big)_5 = \big(\langle \top, \bot, \bot, \bot, \top, \top \rangle\big)_5 \\
&= \langle \bot, \bot, \bot, \top, \top \rangle.
\end{aligned}$$

Multiplication is similar, i.e., both operands $\mathbf{a}_m$ and $\mathbf{b}_n$ are first sign-extended to have $m+n$ bits. After multiplication ($\times_{\mathbf{N}}$) only the relevant $m+n$ bits are taken. We demonstrate multiplication by computing $-2 \times_{\mathbb{Z}} 5 = -10$:

$$\begin{aligned}\langle\top,\top,\bot\rangle \times_{\mathbf{Z}} \langle\bot,\top,\bot,\top\rangle &= \big(\langle\top,\top,\bot\rangle_7 \times_{\mathbf{N}} \langle\bot,\top,\bot,\top\rangle_7\big)_7\\ &= \big(\langle\top,\top,\top,\top,\top,\top,\bot\rangle \times_{\mathbf{N}} \langle\bot,\bot,\bot,\bot,\top,\bot,\top\rangle\big)_7\\ &= \big(\langle\bot,\bot,\bot,\bot,\top,\bot,\bot,\top,\top,\top,\bot,\top,\top,\bot\rangle\big)_7 = \langle\top,\top,\top,\bot,\top,\top,\bot\rangle\end{aligned}$$

# B Matrix Interpretations over the Reals

This section unifies two termination criteria for rewrite systems—matrix interpretations [13, 20] and polynomial interpretations over the non-negative reals [24, 25]—to obtain matrix interpretations over the reals.

## B.1 Preliminaries

We assume familiarity with the basics of rewriting [4] and termination [32].

A *signature* $\mathcal{F}$ is a set of function symbols with fixed arities. Let $\mathcal{V}$ denote an infinite set of variables disjoint from $\mathcal{F}$. Then $\mathcal{T}(\mathcal{F},\mathcal{V})$ forms the set of terms over the signature $\mathcal{F}$ using variables from $\mathcal{V}$. Next we shortly recapitulate the key features of the dependency pair framework [2,17,19]. Let $\mathcal{R}$ be a finite TRS over a signature $\mathcal{F}$. Function symbols that appear as a root of a left-hand side are called *defined*. The signature $\mathcal{F}$ is extended with *dependency pair symbols* $f^\sharp$ for every defined symbol $f$, where $f^\sharp$ has the same arity as $f$, resulting in the signature $\mathcal{F}^\sharp$. If $l \to r \in \mathcal{R}$ and $t$ is a subterm of $r$ with a defined root symbol that is not a proper subterm of $l$ then the rule $l^\sharp \to t^\sharp$ is a *dependency pair* of $\mathcal{R}$. Here $l^\sharp$ and $t^\sharp$ are the result of replacing the root symbols in $l$ and $t$ by the corresponding dependency pair symbols. The dependency pairs of $\mathcal{R}$ are denoted by $\mathsf{DP}(\mathcal{R})$.

A *DP problem* $(\mathcal{P},\mathcal{R})$ is a pair of TRSs $\mathcal{P}$ and $\mathcal{R}$ such that the root symbols of rules in $\mathcal{P}$ do neither occur in $\mathcal{R}$ nor in proper subterms of the left- and right-hand sides of rules in $\mathcal{P}$. The problem is said to be *finite* if there exists no infinite sequence $s_1 \to_{\mathcal{P}} t_1 \to^*_{\mathcal{R}} s_2 \to_{\mathcal{P}} t_2 \to^*_{\mathcal{R}} \cdots$ such that all terms $t_1, t_2, \ldots$ are terminating with respect to $\mathcal{R}$. The main result underlying the dependency pair approach states that termination of a TRS $\mathcal{R}$ is equivalent to finiteness of the DP problem $(\mathsf{DP}(\mathcal{R}),\mathcal{R})$.

To prove a DP problem finite, a number of *DP processors* have been developed. DP processors are functions that take a DP problem $(\mathcal{P},\mathcal{R})$ as input and return a set of DP problems as output. In order to be employed for proving termination DP processors must be *sound*, i.e., if all DP problems returned by a DP processor are finite then $(\mathcal{P},\mathcal{R})$ is finite.

Reduction pairs provide a standard approach for obtaining sound DP processors. Formally, a *reduction pair* $(\gtrsim, >)$ consists of a rewrite pre-order $\gtrsim$ (a pre-order on terms that is closed under contexts and substitutions) and a

well-founded order $>$ that is closed under substitutions such that the inclusion $> \cdot \gtrsim \subseteq >$ (compatibility) holds. Here $\cdot$ denotes composition of relations.

**Theorem 20 (cf. [2,17,19]).** *Let $(\gtrsim, >)$ be a reduction pair. The processor that maps a DP problem $(\mathcal{P}, \mathcal{R})$ to $\{(\mathcal{P} \setminus >, \mathcal{R})\}$ if $\mathcal{P} \subseteq \gtrsim \cup >$ and $\mathcal{R} \subseteq \gtrsim$ and to $\{(\mathcal{P}, \mathcal{R})\}$ otherwise is sound.* □

Next we address how to obtain reduction pairs. For a signature $\mathcal{F}$ an $\mathcal{F}$-algebra $\mathcal{A}$ consists of a *carrier* $A$ and an interpretation $f_A$ for every $f \in \mathcal{F}$. If $\mathcal{F}$ is irrelevant or clear from the context we call an $\mathcal{F}$-algebra simply algebra.

**Definition 21.** *An $\mathcal{F}$-algebra $\mathcal{A}$ over the non-empty carrier $A$ together with two relations $\geqslant$ and $>$ on $A$ is called* weakly monotone *if $f_A$ is monotone in all its coordinates with respect to $\geqslant$, $>$ is well-founded, and $> \cdot \geqslant \subseteq >$.*

Let $\mathcal{A}$ be an algebra over a non-empty carrier $A$. An assignment $\alpha$ for $\mathcal{A}$ is a mapping from the set of term variables $\mathcal{V}$ to $A$. Interpretations are lifted from function symbols to terms, using assignments, as usual. The induced mapping is denoted by $[\alpha]_{\mathcal{A}}(\cdot)$. For two terms $s$ and $t$ we define $s >_{\mathcal{A}} t$ if $[\alpha]_{\mathcal{A}}(s) > [\alpha]_{\mathcal{A}}(t)$ holds for all assignments $\alpha$. The comparison $\geqslant_{\mathcal{A}}$ is similarly defined. Whenever $\alpha$ is irrelevant we abbreviate $[\alpha]_{\mathcal{A}}(s)$ to $[s]_{\mathcal{A}}$.

Weakly monotone algebras give rise to reduction pairs.

**Theorem 22.** *If $(\mathcal{A}, \geqslant, >)$ is weakly monotone then $(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})$ is a reduction pair.*

*Proof.* Immediate from [13, Theorem 2, part 2] which is a stronger result. □

### B.2 Matrix Interpretations

Next we present a DP processor based on matrix interpretations over the reals. Formally, matrix interpretations are weakly monotone algebras $(\mathcal{M}, \geqslant, >)$ where $\mathcal{M}$ is an algebra over some carrier $M^d$ for a fixed $d \in \mathbb{N}^{>0}$. In the sequel we consider $M = \mathbb{R}^{\geqslant 0}$. To define the relations $\geqslant$ and $>$ that compare elements from $M^d$, i.e., vectors with non-negative real entries, we must fix how to compare elements from $M$ first. The obvious candidate $>_{\mathbb{R}}$ is not suitable because it is not well-founded. As already suggested in earlier works on polynomial interpretations [21, 24, 25], $>_{\mathbb{R}}$ can be approximated by $>^{\delta}_{\mathbb{R}}$ defined as $x >^{\delta}_{\mathbb{R}} y \;:=\; x - y \geqslant_{\mathbb{R}} \delta$ for $x, y \in \mathbb{R}$ and any $\delta \in \mathbb{R}^{>0}$. The next lemma shows that $>^{\delta}_{\mathbb{R}}$ has the desired property.

**Lemma 23.** *The order $>^{\delta}_{\mathbb{R}}$ is well-founded on $\mathbb{R}^{\geqslant 0}$ for any $\delta \in \mathbb{R}^{>0}$.*

*Proof.* Obvious. □

With the help of $>^{\delta}_{\mathbb{R}}$ it is now possible to define a well-founded order on $M^d$ similar as in [13].

**Definition 24.** *For vectors* $\mathbf{u}$ *and* $\mathbf{v}$ *from* $M^d$ *we define:*

$$\mathbf{u} \geqslant \mathbf{v} \;:=\; u_i \geqslant_{\mathbb{R}} v_i \text{ for } 1 \leqslant i \leqslant d \qquad \mathbf{u} >^{\delta} \mathbf{v} \;:=\; u_1 >^{\delta}_{\mathbb{R}} v_1 \text{ and } \mathbf{u} \geqslant \mathbf{v}$$

Next the shape of the interpretations is fixed. For an $n$-ary function symbol $f \in \mathcal{F}^\sharp$ we consider linear interpretations $f_{M^d}(\mathbf{x}_1, \ldots, \mathbf{x}_n) = F_1\mathbf{x}_1 + \cdots + F_n\mathbf{x}_n + \mathbf{f}$ where $F_1, \ldots, F_n \in M^{d\times d}$ and $\mathbf{f} \in M^d$ if $f \in \mathcal{F}$ and $F_1, \ldots, F_n \in M^{1\times d}$ and $\mathbf{f} \in M$ if $f \in \mathcal{F}^\sharp \setminus \mathcal{F}$. (As discussed in [13], using matrices of a different shape for dependency pair symbols reduces the search space while preserving the power of the method.) Before addressing how to compare terms with respect to some interpretation we fix the comparison of matrices. Let $m, n \in \mathbb{N}$. For $B, C \in M^{m\times n}$ we define:

$$B \geqslant C \;:=\; B_{ij} \geqslant_{\mathbb{R}} C_{ij} \text{ for all } 1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant n$$

Because of the linear shape of the interpretations, for a rewrite rule $l \to r$ with variables $x_1, \ldots, x_k$, matrices $L_1, \ldots, L_k$, $R_1, \ldots, R_k$ and vectors $\mathbf{l}$ and $\mathbf{r}$ can be computed such that

$$[\alpha]_{\mathcal{M}}(l) = L_1\mathbf{x}_1 + \cdots + L_k\mathbf{x}_k + \mathbf{l} \tag{4}$$
$$[\alpha]_{\mathcal{M}}(r) = R_1\mathbf{x}_1 + \cdots + R_k\mathbf{x}_k + \mathbf{r} \tag{5}$$

where $\alpha(x) = \mathbf{x}$ for $x \in \mathcal{V}$. The next lemma states how to test $s >^{\delta}_{\mathcal{M}} t$ (i.e., $[\alpha]_{\mathcal{M}}(s) >^{\delta} [\alpha]_{\mathcal{M}}(t)$ for all assignments $\alpha$) and $s \geqslant_{\mathcal{M}} t$ effectively.

**Lemma 25.** *Let* $l \to r$ *be a rewrite rule with* $[\alpha]_{\mathcal{M}}(l)$ *and* $[\alpha]_{\mathcal{M}}(r)$ *as in* (4) *and* (5)*, respectively. Then for any* $\delta \in \mathbb{R}^{>0}$

- $l \geqslant_{\mathcal{M}} r$ *if and only if* $L_i \geqslant R_i$ $(1 \leqslant i \leqslant k)$ *and* $\mathbf{l} \geqslant \mathbf{r}$*,*
- $l >^{\delta}_{\mathcal{M}} r$ *if and only if* $L_i \geqslant R_i$ $(1 \leqslant i \leqslant k)$ *and* $\mathbf{l} >^{\delta} \mathbf{r}$*.*

*Proof.* Immediate from the proof of [13, Lemma 4]. □

Matrix interpretations over the reals yield weakly monotone algebras.

**Theorem 26.** *Let* $\mathcal{F}$ *be a signature,* $M = \mathbb{R}^{\geqslant 0}$*, and* $\mathcal{M}$ *an* $\mathcal{F}$*-algebra over the carrier* $M^d$ *for some* $d \in \mathbb{N}^{>0}$ *with* $f_{M^d}$ *of the shape described above for all* $f \in \mathcal{F}$*. Then for any* $\delta \in \mathbb{R}^{>0}$ *the algebra* $(\mathcal{M}, \geqslant, >^{\delta})$ *is weakly monotone.*

*Proof.* The interpretation functions are monotone with respect to $\geqslant$ because of the non-negative carrier. From Definition 24 it is obvious that $>^{\delta}$ is well-founded (on the carrier $M^d$) since $>^{\delta}_{\mathbb{R}}$ is well-founded on $\mathbb{R}^{\geqslant 0}$ for any $\delta \in \mathbb{R}^{>0}$. The latter holds by Lemma 23. The last condition for a weakly monotone algebra is compatibility, i.e., $>^{\delta} \cdot \geqslant \subseteq >^{\delta}$, which trivially holds. □

Matrix interpretations yield reduction pairs due to Theorems 26 and 22, making them suitable for termination proofs in the dependency pair setting.

**Corollary 27.** *If* $(\mathcal{M}, \geqslant, >^{\delta})$ *is a weakly monotone algebra then* $(\geqslant_{\mathcal{M}}, >^{\delta}_{\mathcal{M}})$ *is a reduction pair.* □

We demonstrate matrix interpretations on a simple example.

*Example 28.* The DP problem $(\{\mathsf{f}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{f}^\sharp(x,y)\},\varnothing)$ can be solved by the following interpretation of dimension 2 with $\delta = 1$:

$$\mathsf{f}^\sharp_{M^2}(\mathbf{x},\mathbf{y}) = \begin{pmatrix}1 & 0\end{pmatrix}\mathbf{x} \qquad\qquad \mathsf{s}_{M^2}(\mathbf{x}) = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}\mathbf{x} + \begin{pmatrix}\sqrt{2}\\ 0\end{pmatrix}$$

We have $[\mathsf{f}^\sharp(\mathsf{s}(x),\mathsf{s}(y))]_{\mathcal{M}} = \begin{pmatrix}1 & 0\end{pmatrix}\mathbf{x} + \sqrt{2} >^1 \begin{pmatrix}1 & 0\end{pmatrix}\mathbf{x} = [\mathsf{f}^\sharp(x,y)]_{\mathcal{M}}$. By Lemma 25 and Definition 24 we get $\begin{pmatrix}1 & 0\end{pmatrix} \geqslant \begin{pmatrix}1 & 0\end{pmatrix}$ and $\sqrt{2} >^1_{\mathbb{R}} 0$. The latter holds since $\sqrt{2} - 0 \geqslant_{\mathbb{R}} 1$.

Since $\delta$ influences if a rule can be oriented strictly or not, it cannot be chosen arbitrarily. E.g., the interpretation from Example 28 with $\delta = 2$ can no longer orient the rule strictly since $\sqrt{2} \not>^2_{\mathbb{R}} 0$. For DP problems containing only finitely many rules (this is the usual setting) a suitable $\delta$ can easily be computed. The reason is that for such DP problems only finitely many rules are involved in the strict comparison, i.e., to test for a rule $s \to t$ if $s >^\delta_{\mathcal{M}} t$ the comparison $\mathbf{s} >^\delta \mathbf{t}$ is needed (cf. Lemma 25) which boils down to $s_1 >^\delta_{\mathbb{R}} t_1$ (cf. Definition 24). Since $s_1 - t_1 \geqslant_{\mathbb{R}} \delta$ is tested for only finitely many rules $s \to t$, the minimum of all $s_1 - t_1$ is well-defined and provides a suitable $\delta$. The next lemma (generalizing [24, Section 5.1] to matrices) states that actually there is no need to compute $\delta$ explicitly.

**Lemma 29.** *Let $(\mathcal{P},\mathcal{R})$ be a DP problem. If $\mathcal{P}$ contains finitely many rules then $\delta$ need not be computed.*

*Proof.* The discussion preceding Lemma 29 allows to obtain a $\delta \in \mathbb{R}^{>0}$ such that for every $s \to t \in \mathcal{P}$ we have $s_1 >^\delta_{\mathbb{R}} t_1$ if and only if $s_1 >_{\mathbb{R}} t_1$. Hence for all strict comparisons that occur the relations $>^\delta_{\mathbb{R}}$ and $>_{\mathbb{R}}$ coincide. Consequently it is safe if an implementation uses $>_{\mathbb{R}}$ instead of $>^\delta_{\mathbb{R}}$ in Definition 24 and ignores the exact $\delta$. □

This section is concluded with some comments on automating matrix interpretation, i.e., the problem to find for a given DP problem a matrix interpretation that achieves some progress in the termination proof. Implementing matrix interpretations is a search problem. After fixing the dimension $d$, for every $n$-ary function symbol $f$ we obtain matrices $F_1,\dots,F_n$ and a vector $\mathbf{f}$ filled with arithmetic variables. Lifting addition, multiplication, and comparisons from coefficients to matrices as usual allows to interpret terms. Comparing the term interpretations using Lemma 25 yields an encoding of the DP processor from Theorem 20. For details see [13, 30]. From a model returned by the underlying solver the rules which are deleted by the DP processor and the corresponding (part of the) termination proof can be determined. We stress that for matrix interpretations (and many other termination criteria) a plain YES/NO answer from the underlying SMT solver is not sufficient whenever a (modular) proof should be constructed.