Comparing Proof Systems for Linear Real Arithmetic with LFSC[*]

Andrew Reynolds
Cesare Tinelli
Aaron Stump
The University of Iowa

Liana Hadarean
Yeting Ge
Clark Barrett
New York University

Abstract

LFSC is a high-level declarative language for defining proof systems and proof objects for virtually any logic. One of its distinguishing features is its support for computational side conditions on proof rules. Side conditions facilitate the design of proof systems that reflect closely the sort of high-performance inferences made by SMT solvers. This paper investigates the issue of balancing declarative and computational inference in LFSC focusing on (quantifier-free) Linear Real Arithmetic. We discuss a few alternative proof systems for LRA and report on our comparative experimental results on generating and checking proofs in them.

1 Introduction

A current challenge for the SMT community is to devise a common proof format for proof-producing SMT solvers. The diversity of theories and solving algorithms in SMT makes this difficult, as it seems practically infeasible to design a single set of universally suitable inference rules. To address this difficulty, previous work introduced LFSC ("Logical Framework with Side Conditions"), a meta-level language for specifying proof systems in SMT [9], and showed how to apply it for encoding proofs in the QF_IDL logic of SMT-LIB [8, 1]. LFSC is based on the Edinburgh Logical Framework (LF), a high-level declarative language in which logics (understood as inference systems over a certain language of formulas) can be specified [5]. LFSC increases LF's flexibility by including support for computational side conditions on inference rules. These conditions, expressed in a small functional programming language, enable some parts of a proof to be established by computation. The flexibility of LFSC facilitates the design of proof systems that reflect closely the sort of high-performance inferences made by SMT solvers.

As with LF, a big advantage of LFSC is that, as a meta-level format, it allows the use of a *single* type checker to check proofs in any LFSC-specified logic. But in LFSC, the presence of side conditions opens up a continuum of possible LFSC encodings of a given inference system, from completely declarative, using rules

[*]This work was partially supported by NSF grants 0914877 and 0914956.

with no side conditions, at one end; to completely computational, using a single rule with a huge side condition, at the other.

We argue that supporting this continuum is a major strength of LFSC. Solver implementors have the freedom to choose where to draw the dividing line between declarative and computational inference when devising a proof system. This freedom cannot be abused since any decision is explicitly recorded in the LFCS specification and becomes part of the proof system's trusted computing base. Note, however, that the possibility to create with a relatively small effort different LFSC proof systems for the same logic provides an additional level of trust even for proof systems with a substantial computational component—since at least during the developing phase one could produce proofs in a more declarative, if less efficient, proof system as well.

Instead of developing a dedicated LFSC checker one could imagine embeddind LFSC in existing declarative languages such as Maude or Haskell. While the advantages of prototyping symbolic tools in these languages are well known, in our experience their performance lags too far behind carefully engineered imperative code in C/C++ for high-performance proof checking of very large proofs. Our approach seeks to strike a pragmatic compromise between trustworthiness and efficiency. It would certainly be possible to reduce the size and complexity of the trusted computing base by using a less optimized implementation of LFSC, at the cost of reduced performance. A longer-term solution might be to compile a more declarative description of the checker into a comparably efficient implementation.

Contributions. In this research, we provide further evidence of the viability of LFSC for representing and checking unsatisfiability proofs in SMT. We devise a LFSC proof system, or *calculus*, for the quantifier-free fragment of Linear Real Arithmetic (QF_LRA) [1], and instrument the Cvc3 SMT solver [2] to produce proofs in it. To investigate the balance between declarative and computational inference in proof systems for SMT, we develop alternative translations from Cvc3 proofs to LFSC that exercise different rules of our calculus. The first one is a direct encoding in LFSC syntax of the LRA fragment of the proof system used by Cvc3. Since Cvc3's proof system predates LFSC, it uses, by and large, declarative rules with no side conditions. The second translation uses new arithmetic rules developed with the explicit goal of taking advantage of LFSC's side condition facility. It produces more compact subproofs of the arithmetic lemmas used in the original Cvc3 proof. The third translation is an aggressive version of the second that tries to compact also portions of the Cvc3 proof that rely on general equality reasoning.

Paper outline. We begin with a brief introduction to LFSC. Next, we describe the LFSC LRA calculus, as well as Cvc3's, abstractly in terms of textbook logic rules. Then we explain informally how proofs in Cvc3's native format are converted in proofs in the LFSC calculus.[1] We discuss the main features of these calculi and our various translations, and then provide comparative experimental results on generating and checking LFSC proofs.

[1]A complete and formal description of the two calculi, both at the abstract and the concrete syntax level, and of the conversion process will be provided in a forthcoming technical report.

2 LF with Side Conditions

As mentioned above, LFSC ("Logical Framework with Side Conditions") extends the Edinburgh LF with support for computational side conditions. LF has been used extensively as a metalanguage for encoding deductive systems including logics, semantics of programming languages, and many others. Proof systems are encoded in *signatures*, which are lists of typing declarations. Each proof rule is encoded as a constant symbol, whose type represents the inference allowed by the rule. For example, the following is a standard natural-deduction introduction rule for conjunction, and its encoding in LF (using the prefix syntax of LFSC):

$$\frac{F_1 \quad F_2}{F_1 \ \wedge \ F_2}\ \textsf{and\_intro}$$

```
and_intro : (! F1 formula (! F2 formula (! u1 (pf F1) (! u2 (pf F2)
               (pf (and F1 F2))))))
```

The encoded rule can be read as saying: "for any formulas `F1` and `F2`, and any proofs `u1` and `u2` of `F1` and `F2` respectively, `and_intro` constructs a proof of `(and F1 F2)`."

Unfortunately, pure LF is not suitable for encoding large proofs from SMT solvers. Because LF is purely declarative, computational side conditions like those considered in this paper would need to be encoded via inference rules. This would lead to unacceptable bloating of proofs and proof-checking time, as every inference with a side condition would require an additional non-trivial proof. In contrast, LFSC allows side conditions to be expressed as computational checks, which are performed by the LFSC checker when the inference is proof-checked.

3 Proof Generation and Checking

Proofs in our LFSC calculus for LRA are generated from proofs produced by Cvc3 in its own calculus. We will refer to the former calculus as $\mathcal{L}$ and the latter as $\mathcal{C}$. The reason for translating from Cvc3's native proofs is that its proof-generation facility is deeply embedded in the system's code, and so it is arduous to modify the system to produce LFSC proofs directly. Instead, a translation module was added to Cvc3 that traverses the internal data structure storing the proof, and produces an LFSC proof from it.

We developed three different translations from Cvc3 proofs, differing in how close they are to the original proof. We refer to these as the *literal*, the *liberal* and the *aggressively liberal* translation, and name them LRA1, LRA2, and LRA2a, respectively. To describe their main differences, it is helpful to know that, roughly speaking, Cvc3's proofs have a two-tiered structure, typical of solvers based on the DPLL(T) architecture [7], with a propositional skeleton filled with several theory-specific subproofs [4]. The conclusion is reached by means of propositional or purely equational inferences applied to a set of input formulas and a set of *theory lemmas*. The latter are disjunctions of arithmetic atoms deduced from no assumptions, mostly using proof rules specific to the theory in question—the theory of real arithmetic in this case.

In the literal translation, LRA1, an LFSC proof is produced directly from Cvc3's proof, using whenever possible $\mathcal{L}$ rules that mirror the corresponding $\mathcal{C}$ rules, and resorting to additional $\mathcal{L}$-specific rules only for those few $\mathcal{C}$ rules that cannot be checked by simple pattern matching (but require, for instance, to verify that a certain expression in the $\mathcal{C}$ rule is a normalized version of another).

In the two liberal translations, the Cvc3 proof is used as a guide to produce a compact proof that relies on rules with side conditions specific to $\mathcal{L}$—that is, not encoding a rule of $\mathcal{C}$. The use of side conditions enables a level of compaction that is otherwise infeasible due to the declarative nature of rules in the $\mathcal{C}$ calculus. In LRA2, the subproofs of all theory lemmas are systematically converted to more compact proofs that use $\mathcal{L}$-specific rules; the rest of the Cvc3 proof is translated as in the literal translation. The LRA2a translation is identical to LRA2 except that it tries to compact also parts of the proof that rely on generic equality reasoning (for instance, applications of congruence rules), again by using $\mathcal{L}$-specific rules. This translation, which is still somewhat experimental, uses an adaptive strategy to switch from $\mathcal{L}$-specific equality rules to $\mathcal{C}$-like equality rules and back, making heuristic decisions on when it is worthwhile to do so.

On average, the translation times for the three translations are the same. However, the size of a proof produced with the liberal translations is often considerably smaller than the size of the corresponding proof generated with the literal translation. The compression is achieved to a great extent thanks to $\mathcal{L}$ proof rules that work with *normalized polynomial atoms*, atoms of the form

$$c_1 \cdot x_1 + \ldots + c_n \cdot x_n + c_{n+1} \sim 0$$

where each c_i is a rational constant, each x_i is a real variable, and $\sim$ is one of the relational operators $=, >, \geq$. These rules take normalized atoms in their premise and introduce only normalized atoms in their conclusion. The computation of the atoms in the conclusion is delegated to the rule's side condition, which consists of a call to a simple polynomial normalization function. Overall, the LSFC side condition code that defines these functions has about 60 lines of LISP-like code for a total of less than 2 kilobytes, and is similar in structural complexity to the implementation of a merge sort of lists of key/value pairs.

Proof Checking. LFSC proofs are checked using a high-performance LFSC type checker, developed (in around 5kloc of C++) by Reynolds and Stump. In previous work [8], they showed that the LFSC checker is highly competitive with alternative approaches to SMT proof checking such as the one used with the fx7 system [6]. As explained in the cited work, the LFSC checker implements compilation to C++ of side-condition code, for higher performance. Another important optimization, called *incremental checking*, interleaves parsing and type checking. This makes it possible to parse and type check large proofs, without needing to build first an abstract syntax tree in memory for the whole proof. These two optimizations each lead to significant reductions in running time (on the order of 5x), and also memory usage.

For this work, a few new features were added to the LFSC checker:

- support for arbitrary-precision rational arithmetic;
- local definitions via a `let` construct; and
- a `compare` primitive function, to allow side-condition code to impose an ordering on LFSC variables.[2]

[2]This is used in sorting lists of monomials, for example.

$$\frac{\Gamma_1 \vdash t_1 \sim t_2 \Leftrightarrow t_3 \sim t_4 \quad \Gamma_2 \vdash t_3 \sim t_4 \Leftrightarrow t_5 \sim t_6}{\Gamma_1, \Gamma_2 \vdash t_1 \sim t_2 \Leftrightarrow t_5 \sim t_6}\ \mathsf{iff\_trans}$$

$$\frac{\Gamma_1 \vdash t_1 \sim t_2 \quad \Gamma_2 \vdash t_1 \sim t_2 \Leftrightarrow t_3 \sim t_4}{\Gamma_1, \Gamma_2 \vdash t_3 \sim t_4}\ \mathsf{iff\_mp}$$

$$\frac{\Gamma_1 \vdash t_1 = t_2 \quad \Gamma_2 \vdash t_3 = t_4}{\Gamma_1, \Gamma_2 \vdash t_1 \sim t_3 \Leftrightarrow t_2 \sim t_4}\ \mathsf{congr\_1} \qquad \frac{\Gamma_1 \vdash t_1 = t_2 \quad \Gamma_2 \vdash t_2 = t_3}{\Gamma_1, \Gamma_2 \vdash t_1 \ = t_3}\ \mathsf{eq\_trans}$$

$$\frac{\Gamma_1 \vdash t_1 = t_2 \quad \Gamma_2 \vdash t_3 = t_4}{\Gamma_1, \Gamma_2 \vdash t_1 \bowtie t_3 = t_2 \bowtie t_4}\ \mathsf{congr\_2} \qquad \frac{\Gamma_1 \vdash t_1 = t_2}{\Gamma_1, \Gamma_2 \vdash t_2 = t_1}\ \mathsf{eq\_symm}$$

Figure 1: Some of Cvc3's propositional and equality proof rules for QF_LRA.

$$\frac{\Gamma_1 \vdash t_1 > t_2 \quad \Gamma_2 \vdash t_2 > t_3}{\Gamma_1, \Gamma_2 \vdash t_1 > t_3}\ \mathsf{gt\_trans} \qquad \frac{\Gamma_1 \vdash t_1 > t_2 \quad \Gamma_2 \vdash t_2 > t_1}{\Gamma_1, \Gamma_2 \vdash \bot}\ \mathsf{gt\_acyc}$$

$$\frac{[0 \not\sim c]}{\vdash (0 \sim c) \Leftrightarrow \bot}\ \mathsf{const\_pred\_1} \qquad \frac{}{\vdash t_1 \sim t_2 \Leftrightarrow 0 \sim t_2 - t_1}\ \mathsf{right\_minus\_left}$$

$$\frac{[t' \text{ canonical form of } t]}{\vdash t = t'}\ \mathsf{canon} \qquad \frac{[c \text{ non-negative}]}{\vdash t_1 \sim t_2 \Leftrightarrow c \cdot t_1 \sim c \cdot t_2}\ \mathsf{mult\_pred}$$

$$\frac{}{\vdash t_1 > t_2 \Leftrightarrow t_2 < t_1}\ \mathsf{flip\_ineq} \qquad \frac{}{\vdash t_1 \sim t_2 \Leftrightarrow t_1 + t_3 \sim t_2 + t_3}\ \mathsf{plus\_pred}$$

Figure 2: Some of Cvc3's arithmetic proof rules for QF_LRA.

4 The Cvc3 and LFSC Calculi for LRA

The $\mathcal{C}$ calculus. The fragment of Cvc3's proof system for QF_LRA[3] can be described mathematically as a sequent calculus $\mathcal{C}$ with judgments of the form $\Gamma \vdash \varphi$, where Γ is a set of quantifier-free LRA formulas and φ is a single LRA formula, also quantifier-free. Each rule is an instance of this general schema:

$$\frac{\Gamma_1 \vdash \varphi_1 \ \cdots \ \Gamma_n \vdash \varphi_n}{\Gamma \vdash \varphi}$$

for some $n \geq 0$. The *axioms* of the calculus, i.e., the proof rules with with $n = 0$, have, however, conclusions of a more restricted form: either $\varphi \vdash \varphi$ or $\vdash \psi$. The first kind of axiom is used to introduce assumptions in sequents; the second to introduce some basic LRA theorems. A small sample of Cvc3's rules is provided in Figures 1 and 2.[4] The rules are fairly standard and self-explanatory, with the possible exception of canon. This rule asserts an equality between a term t and its equivalent *canonical form* produced by Cvc3's canonizer module, which applies some standard equivalence-preserving simplifications.

Although the $\mathcal{C}$ calculus itself is quite general, all Cvc3 proofs in it are

[3]The complete proof system is a lot larger because it supports a much larger logic than QF_LRA.

[4]A more extensive set of rules is provided in the appendix. To ease formatting, some rule names may have a different name from the one used by Cvc3.

$$\frac{}{\vdash c_{\mathrm{t}} = c_{\mathrm{p}}}\ \textsf{poly\_norm\_const} \qquad \frac{}{\vdash v_{\mathrm{t}} = v_{\mathrm{p}}}\ \textsf{poly\_norm\_var}$$

$$\frac{\Gamma_1 \vdash t_1 = p_1 \quad \Gamma_2 \vdash t_2 = p_2}{\Gamma_1, \Gamma_2 \vdash t_1 + t_2 = (p_1 + p_2){\downarrow}}\ \textsf{poly\_norm}_{+} \qquad \frac{\Gamma \vdash t = p}{\Gamma \vdash c_{\mathrm{t}} \cdot t = (c_{\mathrm{p}} \cdot p){\downarrow}}\ \textsf{poly\_norm}_{c\cdot}$$

$$\frac{\Gamma \vdash t_1 - t_2 = p}{\Gamma \vdash t_1 = t_2 \Leftrightarrow p = 0}\ \textsf{poly\_norm}_{=} \qquad \frac{\Gamma \vdash t_1 - t_2 = p}{\Gamma \vdash t_1 > t_2 \Leftrightarrow p > 0}\ \textsf{poly\_norm}_{>}$$

$$\frac{\Gamma_1 \vdash t_1 \sim t_2 \quad \Gamma_2 \vdash t_1 \sim t_2 \Leftrightarrow p \sim 0}{\Gamma_1, \Gamma_2 \vdash p \sim 0}\ \textsf{poly\_form}$$

Figure 3: Some conversion rules of $\mathcal{L}$. The expressions c_{t} and c_{p} denote the same rational constant, in one case seen as a term and in the other as a polynomial (similarly for the variables v_{t} and v_{p}).

$$\frac{\Gamma_1 \vdash p \geq 0 \quad \Gamma_2 \vdash p' \geq 0 \quad [p + p' = 0]}{\Gamma_1, \Gamma_2 \vdash p = 0}\ \textsf{lra}{\geq}{\geq}\textsf{to}{=}$$

$$\frac{[c \geq 0]}{\vdash c \geq 0}\ \textsf{lra}{\geq}\textsf{axiom} \qquad \frac{\Gamma \vdash p = 0 \quad [p \neq 0]}{\Gamma \vdash \bot}\ \textsf{lra\_contra}_{=}$$

$$\frac{\Gamma \vdash p > 0 \quad [c > 0]}{\Gamma \vdash (c \cdot p){\downarrow} > 0}\ \textsf{lra\_mult}_{c>} \qquad \frac{\Gamma_1 \vdash p_1 > 0 \quad \Gamma_2 \vdash p_2 > 0}{\Gamma_1, \Gamma_2 \vdash (p_1 + p_2){\downarrow} > 0}\ \textsf{lra\_add}_{>>}$$

Figure 4: Some of the polynomial rules of $\mathcal{L}$.

refutations, that is, have a conclusion of the form $\Gamma \vdash \bot$ where Γ is a subset of the formulas whose joint satisfiability Cvc3 was asked to check.

The $\mathcal{L}$ calculus. At the abstract level, the LFSC calculus $\mathcal{L}$ can be described as a proper superset of $\mathcal{C}$. In reality, $\mathcal{L}$ includes a set of rules that, while essentially the same as those in $\mathcal{C}$, have a different concrete syntax. The remaining rules of $\mathcal{L}$ are those used by the two liberal translations. Some of these rules convert arithmetic terms used in the $\mathcal{C}$ rules to polynomials, and back. This is because, in the concrete LFSC syntax, arithmetic terms—denoted by the letter t in the rules—and polynomials—denoted by p—belong to different types, with the polynomial type optimized to support fast normalization operations. The conversion rules operate on mixed-type atoms that have a term argument and a polynomial argument. A sample of such rules is provided in Figure 3. There, $e{\downarrow}$ denotes the results of converting the polynomial expression e into a normalized polynomial. The normalization is done by the rule's side condition, which is however left implicit to simplify the notation.

A further set of rules operate only on polynomial atoms and are used by the liberal translations to generate proofs of LRA lemmas. A selection of these rules is provided in Figure 4. To ease formatting, explicit side conditions are written together with the premises, but enclosed in brackets. Although side conditions use the same syntax used in the sequents, they should be read as a mathematical notation. For example, $p = 0$ in a premise denotes an atomic formula whose left-hand side is an arbitrary polynomial and whose right-hand side is the 0

$$\dfrac{\dfrac{\dfrac{}{\Gamma_1 \vdash 2x > 2y} \quad \dfrac{\dfrac{}{\vdash x > y \Leftrightarrow 2x > 2y}}{\vdash 2x > 2y \Leftrightarrow x > y}\,\mathsf{e}}{\Gamma_1 \vdash x > y}\,\mathsf{m} \quad \dfrac{}{\Gamma_2 \vdash y > x + 5}}{\Gamma \vdash x > x + 5}\,\mathsf{t} \quad \dfrac{\vdots}{\vdash x > x + 5 \Leftrightarrow \bot}\,\mathsf{e}}{\Gamma \vdash \bot}\,\mathsf{m}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Gamma_1 \vdash 2x - 2y > 0}}{\Gamma_1 \vdash x - y > 0} \quad \dfrac{\vdots}{\vdash 0 = 0}}{\Gamma_1 \vdash x - y > 0} \quad \dfrac{\vdots}{\Gamma_2 \vdash -x + y - 5 > 0}}{\Gamma \vdash -5 > 0} \quad \dfrac{\vdots}{\vdash 0 = 0}}{\dfrac{\Gamma \vdash -5 > 0}{\Gamma \vdash \bot}}$$

Figure 5: Example of a proof fragment in the $\mathcal{C}$ calculus (top) and the corresponding fragment in the $\mathcal{L}$ calculus using polynomial rules (bottom), with $\Gamma_1 = \{2x > 2y\}$, $\Gamma_2 = \{y > x + 5\}$ and $\Gamma = \Gamma_1 \cup \Gamma_2$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Gamma_1 \vdash 2x - 2y > 0}}{\Gamma_1 \vdash x - y > 0} \quad \dfrac{\vdots}{\Gamma_2 \vdash -x + y - 5 > 0}}{\Gamma \vdash -5 > 0}}{\Gamma \vdash \bot}$$

Figure 6: Proof fragment eventually produced by the liberal translations from the $\mathcal{C}$ proof fragment in the previous figure.

polynomial; in contrast, the side condition $[p + p' = 0]$, say, denotes the result of checking whether the expression $p + p'$ evaluates to 0 in the polynomial ring $\mathbb{Q}[X]$, where $\mathbb{Q}$ is the field of rational numbers and X the set of all variables (or "free constants" in SMT-LIB parlance).

5 From $\mathcal{C}$ proofs to $\mathcal{L}$ proofs

In this section, we describe briefly the main approach used by the liberal translations to produce more compact proofs. Due to space restrictions, se cannot give a comprehensive description of the translation algorithm here. Instead, we give the main intuition, focusing on the translation of theory lemma (sub)proofs, and provide an example.

Theory lemmas in Cvc3 proofs are derived by first proving $\varphi_1, \ldots, \varphi_n \vdash \bot$ from assertions (axioms) of the form $\psi \vdash \psi$, where all the ψ's and φ_i's are arithmetic atoms. These proofs rely on a variety of rules, including equivalence axioms of the form $\vdash \psi_1 \Leftrightarrow \psi_2$, used a rewrite rules, and standard rules for natural deduction. In contrast, theory lemma proofs in the $\mathcal{L}$ calculus amount to determining a list of rational coefficients that when multiplied by the asserted atomic formulas, allow one to produce an inconsistent polynomial atom $c_p \sim 0$ for some constant polynomial c_p (such as $3 = 0$ or $-1 > 0$), using simple polynomial ring operations like addition and subtraction. An important point is that these coefficients are computed directly, deterministically and efficiently, from the Cvc3 proof.

The translation process exploits the fact that in many cases, a proof of a polynomial atom can be derived in $\mathcal{L}$ that is at least as strong as the formula that is proved by the Cvc3 proof. That is to say, if a subproof P of a theory lemma proves $\Gamma \vdash \psi$ in $\mathcal{C}$, there often exists a proof of $\Gamma \vdash p \sim 0$ in $\mathcal{L}$ such that Γ entails $(p \sim 0) \Rightarrow \psi$.[5] In fact, it is often the case that Γ entails $(p \sim 0) \Leftrightarrow \psi$.

The translation to $\mathcal{L}$ proofs is performed incrementally and bottom-up over the structure of the Cvc3 proof, where applications of rules in $\mathcal{C}$ are first translated to applications of corresponding rules for polynomials in $\mathcal{L}$. The proof obtained this way is then compacted to eliminate superfluous subproofs. For instance, consider a Cvc3 subproof whose conclusion is obtained with an application of the rule gt_trans from Figure 2. The corresponding proof in the $\mathcal{L}$ calculus will then have the form

$$\frac{\begin{array}{cc} \vdots & \vdots \\ \Gamma_1 \vdash p_1 > 0 & \Gamma_2 \vdash p_2 > 0 \end{array}}{\Gamma \vdash p_1 + p_2 > 0}$$

where $p_1 > 0$ implies (or is equivalent to) the formula $t_1 > t_2$ in the premise of gt_trans; likewise, $p_2 > 0$ implies (or is equivalent to) $t_2 > t_3$. The conclusion $p_1 + p_2 > 0$ will then be equivalent to formula $t_1 + t_2 > t_2 + t_3$, or simply $t_1 > t_3$, as derived by gt_trans.

For a concrete example of the translation mechanism, Figure 5 gives a proof fragment of a $\mathcal{C}$ refutation of the set $\{2x > 2y,\ y > x+5\}$. The lines marked with t correspond to an application of gt_trans, those marked with m to an application of iff_mp, and those marked with e to an application of an equivalence rule. Below the $\mathcal{C}$ proof, the figure shows a corresponding $\mathcal{L}$ proof (fragment) that closely matches the $\mathcal{C}$ proof but uses polynomial rules instead. For each node in the $\mathcal{C}$ proof, a polynomial inference (possibly preceded by a constant multiplication inference) suffices to prove an equivalent or stronger conclusion. Equivalence inferences in the $\mathcal{C}$ proof are replaced by inferences of the trivial identity $0 = 0$ in the $\mathcal{L}$ proof, modus ponens inferences by polynomial subtraction inferences, and transitivity inferences for $>$ by polynomial addition inferences. Similar conversions apply to all $\mathcal{C}$ rules used in theory lemma proofs. Note how the $\mathcal{L}$ proof ends up containing subproofs that clearly do not contribute to the value of the inconsistent polynomial atom $-5 > 0$, namely those with conclusion $\vdash 0 = 0$. The translation will prune all such subproofs and compact the rest of the proof accordingly. The final result of applying the liberal translations to the $\mathcal{C}$ proof of Figure 5 is shown in Figure 6.

6 Experimental Results

To evaluate the various translations experimentally, we looked at all the QF_LRA benchmarks from SMT-LIB. Our results do not contain comparisons with third party proof checkers because of a lack of viable alternatives. A potential candidate was a former system developed by Ge and Barrett that used the HOL Light prover as a proof checker for Cvc3 [3]. Unfortunately, that system, which was

[5]It is our experience that all subproofs P of theory lemmas in Cvc3 exhibit this property.

		Solve + Pf Gen + Pf Conv (s)						Proof Size (MB)					Pf Check Time (s)				
Bench Class	#	cvc	cvcpf	lra1	lra2	lra2a	lrant	cvcpf	lra1	lra2	lra2a	lrant	lra1	lra2	lra2a	lrant	T%
check	1	0.1	0.2	0.3	0.2	0.2	0.2	0.3	0.5	0.2	0.1	0.1	0.1	0.1	0.1	0.04	72%
clock_synchro	18	9.0	18.1	23.2	22.7	21.6	22.3	9.1	14.1	12.4	7.2	12.0	3.8	3.5	2.2	3.2	17%
gasburner	19	2.3	6.3	9.5	8.2	8.3	7.3	8.5	13.3	7.5	6.7	6.3	3.6	2.3	2.0	1.8	46%
pursuit	8	13.9	22.7	26.6	26.2	26.3	25.9	3.9	4.9	3.5	3.5	3.2	1.3	1.0	0.9	0.8	36%
spider	35	6.9	13.8	18.4	18.1	18.3	17.7	10.0	12.0	10.5	10.4	10.2	3.4	3.0	3.2	3.1	15%
tgc	16	3.6	8.6	11.1	10.7	11.2	10.5	7.9	8.1	6.7	7.2	6.5	2.2	2.3	2.0	1.8	11%
TM	1	16.3	26.0	29.0	28.9	28.9	29.0	1.3	2.8	2.8	2.8	2.8	0.6	0.6	0.6	0.6	0%
tta_startup	24	24.0	55.5	66.0	65.7	67.9	65.5	36.3	39.2	39.1	45.7	38.8	8.2	8.4	10.0	8.2	2%
windowreal	23	16.8	34.5	35.6	35.5	36.6	35.4	18.5	19.9	19.7	21.4	19.6	5.3	5.3	5.9	5.3	3%
Total	145	92.8	185.7	219.6	216.2	219.1	213.9	95.8	114.8	102.4	104.9	99.5	28.6	26.4	26.8	24.9	11%

Table 1: Cumulative results, grouped by benchmark class. Column 2 gives the numbers of benchmarks in each class. Columns 3 through 8 give Cvc3's (aggregate) runtime for each of the five configurations. Columns 9 through 13 show the proof sizes for each of the 5 proof-producing configurations. Columns 14 through 17 show LFSC proof checking times. The last column gives the percentage of proof nodes found beneath theory lemmas in Cvc3's native proofs.

never tested on QF_LRA benchmarks and was not kept in sync with the latest developments of Cvc3, currently breaks on most of them. While we expect that it could be fixed, the required amount of effort is beyond the scope of this work.

We ran our experiments on a Linux machine with a 2.67GHz 4-core Xeon processor and 8GB of RAM. From all the 317 unsatisfiable QF_LRA benchmarks, we selected the 189 benchmarks that Cvc3 could solve in 60s. We discuss only 145 of these 189 benchmarks here because for the rest Cvc3 either could not produce a proof in 300s or produced one containing a few known proof rules that we have not been able to implement in LSFC yet because of time constraints.

We collected runtimes for the following five main configurations of Cvc3.

cvc Default configuration, solving benchmarks but with no proof generation.

cvcpf Solving with proof generation in Cvc3's native format.

lra1 Solving with proof generation and literal translation to LFSC.

lra2 Solving with proof generation and liberal translation to LFSC.

lra2a Like **lra2** but with aggressively liberal translation to LFSC.

We also ran a sixth configuration, **lrant**, for the purpose of isolating the non-theory component of proof sizes and checking times. This configuration ignores (trusts) all theory lemmas, but otherwise is identical to both **lra1** and **lra2**. Comparisons with this configuration are useful because the liberal translations work mostly by compacting the theory-specific portion of a proof. Hence, their effectiveness is expected to be correlated with the amount of *theory content* of a proof. We measure that as the percentage of nodes in a Cvc3 proof that belong to the (sub)proof of a theory lemma. For this data set, the average theory content is very low, about 11%.

Table 1 shows a summary of our results for various classes of benchmarks.[6] As can be seen from the table, Cvc3's solving times with native proof generation are on average 2 times slower than without. The translation to LFSC proofs

[6]Detailed results, together with LFSC definitions of all proof rules, are available at `http://clc.cs.uiowa.edu/smt10` .

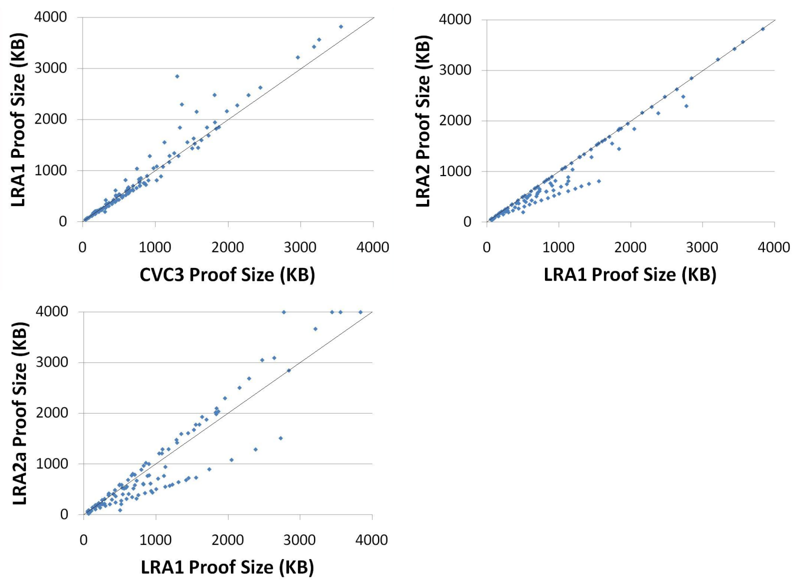

Figure 7: Comparing proof sizes.

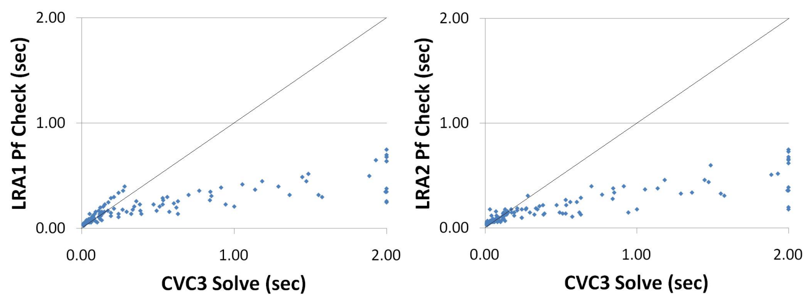

Figure 8: Solving times vs. proof checking times.

adds additional overhead, which is however less than 18% on average for all translations.

The scatter plots in Figure 7 are helpful in comparing the sizes of Cvc3 native proofs versus their literal translation LRA1, and the size of the latter proofs versus the size of LFSC proofs produced with the liberal translations LRA2 and LRA2a. The first plot clearly shows that, save a couple of outliers, LRA1 suffers only a small constant overhead, which we believe is due to structural differences between the Cvc3 and the LFSC proof languages. The second plot shows that the liberal translation introduces constant compression factors over the literal one. A number of benchmarks in our test set do not benefit from the LRA2 translation. We have found that such benchmarks are not heavily dependent on theory reasoning, having a theory content of less than 5%. In contrast, for benchmarks with a high theory content, LRA2 is effective at proof compression. When focusing on theory lemma subproofs (by subtracting proofs sizes in **lrant** from both **lra1** and **lra2**), LRA2 presents an average compression factor of 5.39. Over the set of all benchmarks with *enough* theory content, quantified as 10% or more, LRA2 compresses proof sizes an average of 34%. The differences in proof sizes between benchmarks with enough theory content and the rest are magnified in the LRA2a translation. With the former set, LRA2a compacts the proof size by 62% on average. However, LRA2a suffers on the other benchmarks, showing a 11% increase in size on average. This can be attributed to cost incurred by context switching between compact and literal translation modes. LRA2 is the more effective of the two liberal translations, showing an average compressions of 12%.

We compared the proof checking times of LRA1 vs. LRA2 and LRA2a, using the LFSC checker. Perhaps unsurprisingly, their scatter plots (not shown here) are very similar to the corresponding ones in Figure 7. Over benchmarks with enough theory content, checking LRA2 proofs is on average 25% faster than checking the corresponding LRA1 proofs. Looking just at theory lemmas, this time by subtracting the checking times of **lrant**, reveals that proof checking times are 2.3 times faster for LRA2 than for LRA1.

It is generally expected that proof checking should be substantially faster than proof generation or even just solving. This is generally the case for both LRA1 and LRA2. Compared against Cvc3's solving times alone, LFSC proof checking are 3.25 times faster with LRA1 proofs, and 3.5 times faster with LRA2 proofs. A more detailed comparison can be seen in Figure 8. We have noticed that LRA1 proof checking times are actually slower than Cvc3 solving times for a certain set of theory heavy benchmarks, as shown by the steep linear line above the diagonal of the first scatter plot of Figure 8. It is interesting to observe that the proof checking times for these benchmarks are significantly lower for LRA2, with the net result that, modulo measurement errors, all proof checking times are equal or smaller than their corresponding solving times, with a significant portion being considerably smaller.

7 Conclusion and Further Work

We have investigated alternative proof systems for quantifier-free Linear Real Arithmetic in the LFSC framework. Proofs in these systems were produced by translating in LFSC proofs generated by the Cvc3 SMT solver. We discussed three translations, differing in their degree of faithfulness to native Cvc3 proofs and compaction of arithmetic or equational reasoning steps. These differences were made possible by the flexibility of LFSC which allows one to modulate the amount of declarative and computational proof checking that goes into a proof system.

We have shown that a translation mixing declarative propositional and equational reasoning proof rules with more computational arithmetic proof rules is effective in producing smaller proof sizes and proof checking times, while also maintaining a high level of trust because of the simplicity of the side conditions. Because LFSC is a meta-logical framework, such experimentation was done without modification to the core of the proof checker, thus allowing an effective methodology for comparing multiple approaches.

Our most immediate next effort will go towards collecting more substantial experimental evidence that our approach scales well. While we have every reason to believe so, our current data set consists of relatively small proofs ($<$ 1MB on average), which can be checked very quickly ($<$ 0.3s on average). For this goal, at least in the short run, we may need to consider other SMT-LIB logics beside QF_LRA.

Acknowlegments We thank the anonymous reviewers for their helpful suggestions for improving the presentation of the paper.

References

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

[2] Clark Barrett and Cesare Tinelli. CVC3. In W. Damm and H. Hermanns, editors, *Proceedings of the 19th International Conference on Computer Aided Verification (CAV'07), Berlin, Germany*, volume 4590 of *Lecture Notes in Computer Science*, pages 298–302. Springer, 2007.

[3] Yeting Ge and Clark Barrett. Proof translation and SMT-LIB benchmark certification: A preliminary report. In *Proceedings of SMT'08*, 2008.

[4] Amit Goel, Sava Krstić, and Cesare Tinelli. Ground interpolation for combined theories. In R. Schmidt, editor, *Proceedings of the 22nd International Conference on Automated Deduction (Montreal, Canada)*, volume 5663 of *Lecture Notes in Artificial Intelligence*, pages 183–198. Springer, 2009.

[5] R. Harper, F. Honsell, and G. Plotkin. A Framework for Defining Logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[6] M. Moskal. Rocket-Fast Proof Checking for SMT Solvers. In C. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, 2008.

[7] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, November 2006.

[8] Duckki Oe, Andrew Reynolds, and Aaron Stump. Fast and flexible proof checking for SMT. In *Proceedings of SMT'09*, 2009.

[9] A. Stump. Proof Checking Technology for Satisfiability Modulo Theories. In A. Abel and C. Urban, editors, *Logical Frameworks and Meta-Languages: Theory and Practice*, 2008.

A $\mathcal{C}$ Proof Rules

The following is a representative list of rules in the $\mathcal{C}$ calculus. The letters c and t, possibly with subscripts, denote rational constants and arithmetic terms, respectively.

A.1 Arithmetic Axioms

$$\frac{[0 \not\sim c]}{\vdash (0 \sim c) \Leftrightarrow \bot}\ \textsf{const\_pred\_1} \qquad \frac{[0 \sim c]}{\vdash (0 \sim c) \Leftrightarrow \top}\ \textsf{const\_pred\_2}$$

$$\frac{[c \text{ non-negative}]}{\vdash t_1 \sim t_2 \Leftrightarrow c \cdot t_1 \sim c \cdot t_2}\ \textsf{mult\_ineqn} \qquad \frac{[c \text{ non-zero}]}{\vdash t_1 = t_2 \Leftrightarrow c \cdot t_1 = c \cdot t_2}\ \textsf{mult\_eqn}$$

$$\frac{}{\vdash t_1 \sim t_2 \Leftrightarrow 0 \sim t_2 - t_1}\ \textsf{right\_minus\_left} \qquad \frac{}{\vdash t_1 > t_2 \Leftrightarrow t_2 < t_1}\ \textsf{flip\_ineq}$$

$$\frac{}{\vdash t_1 \sim t_2 \Leftrightarrow t_1 + t_3 \sim t_2 + t_3}\ \textsf{plus\_pred} \qquad \frac{}{\vdash \neg(t_1 > t_2) \Leftrightarrow t_1 \leq t_2}\ \textsf{negated\_ineq}$$

$$\frac{[c_1 > c_2]}{\vdash 0 > c_1 + t \Rightarrow 0 > c_2 + t}\ \textsf{weaker\_ineq}$$

$$\frac{}{\vdash (t_1 - t_2) = t_1 + (-1 \cdot t_2)}\ \textsf{minus\_to\_plus} \qquad \frac{[t' \text{ canonical form of } t]}{\vdash t = t'}\ \textsf{canon}$$

A.2 Natural Deduction Rules

$$\frac{\Gamma_1 \vdash t_1 \sim t_2 \Leftrightarrow t_3 \sim t_4 \quad \Gamma_2 \vdash t_3 \sim t_4 \Leftrightarrow t_5 \sim t_6}{\Gamma_1, \Gamma_2 \vdash t_1 \sim t_2 \Leftrightarrow t_5 \sim t_6}\ \textsf{iff\_trans}$$

$$\frac{\Gamma_1 \vdash t_1 \sim t_2 \quad \Gamma_2 \vdash t_1 \sim t_2 \Leftrightarrow t_3 \sim t_4}{\Gamma_1, \Gamma_2 \vdash t_3 \sim t_4}\ \textsf{iff\_mp}$$

$$\frac{\Gamma_1 \vdash t_1 \sim t_2 \Leftrightarrow t_3 \sim t_4}{\Gamma_1 \vdash t_3 \sim t_4 \Leftrightarrow t_1 \sim t_2}\ \textsf{iff\_symm}$$

$$\frac{\Gamma_1 \vdash t_1 \sim t_2 \Rightarrow t_3 \sim t_4 \quad \Gamma_2 \vdash t_3 \sim t_4 \Rightarrow t_5 \sim t_6}{\Gamma_1, \Gamma_2 \vdash t_1 \sim t_2 \Rightarrow t_5 \sim t_6}\ \textsf{impl\_trans}$$

$$\frac{\Gamma_1 \vdash t_1 \sim t_2 \quad \Gamma_2 \vdash t_1 \sim t_2 \Rightarrow t_3 \sim t_4}{\Gamma_1, \Gamma_2 \vdash t_3 \sim t_4}\ \textsf{impl\_mp}$$

A.3 Equality Rules

$$\frac{}{\vdash t_1 = t_1}\ \textsf{refl}$$

$$\frac{\Gamma_1 \vdash t_1 = t_2 \quad \Gamma_2 \vdash t_3 = t_4}{\Gamma_1, \Gamma_2 \vdash t_1 \sim t_3 \Leftrightarrow t_2 \sim t_4}\ \textsf{congr\_1} \qquad \frac{\Gamma_1 \vdash t_1 = t_2 \quad \Gamma_2 \vdash t_2 = t_3}{\Gamma_1, \Gamma_2 \vdash t_1 = t_3}\ \textsf{eq\_trans}$$

$$\frac{\Gamma_1 \vdash t_1 = t_2 \quad \Gamma_2 \vdash t_3 = t_4}{\Gamma_1, \Gamma_2 \vdash t_1 \bowtie t_3 = t_2 \bowtie t_4}\ \textsf{congr\_2} \qquad \frac{\Gamma_1 \vdash t_1 = t_2}{\Gamma_1, \Gamma_2 \vdash t_2 = t_1}\ \textsf{eq\_symm}$$

A.4 Arithmetic Rules

$$\frac{\Gamma_1 \vdash t_1 > t_2 \quad \Gamma_2 \vdash t_2 > t_3}{\Gamma_1, \Gamma_2 \vdash t_1 > t_3}\ \textsf{gt\_trans} \qquad \frac{\Gamma_1 \vdash t_1 > t_2 \quad \Gamma_2 \vdash t_2 > t_1}{\Gamma_1, \Gamma_2 \vdash \bot}\ \textsf{gt\_acyc}$$

$$\frac{\Gamma_1 \vdash t_1 \geq t_2 \quad \Gamma_2 \vdash t_1 \leq t_2}{\Gamma_1, \Gamma_2 \vdash t_1 = t_2}\ \textsf{gt\_antisym}$$

B $\mathcal{L}$ Proof Rules

In the proof rules below, the expression $e{\downarrow}$ denotes the result of normalizing the polynomial expression e to a polynomial. The normalization is done by the rules side condition, which is however left implicit here to keep the notation uncluttered.

B.1 Arithmetic Axioms

$$\frac{}{\vdash 0 = 0}\ \textsf{lra=axiom}$$

$$\frac{[c > 0]}{\vdash c > 0}\ \textsf{lra>axiom}$$

$$\frac{[c \geq 0]}{\vdash c \geq 0}\ \textsf{lra}{\geq}\textsf{axiom}$$

B.2 Equality Deduction Rule

$$\frac{\Gamma_1 \vdash p \geq 0 \quad \Gamma_2 \vdash p' \geq 0 \quad [p + p' = 0]}{\Gamma_1, \Gamma_2 \vdash p = 0}\ \textsf{lra}{\geq}{\geq}\textsf{to=}$$

B.3 Contradiction Rules

$$\frac{\Gamma \vdash p = 0 \quad [p \neq 0]}{\Gamma \vdash \bot}\ \textsf{lra\_contra}_{=}$$

$$\frac{\Gamma \vdash p > 0 \quad [p \not> 0]}{\Gamma \vdash \bot}\ \textsf{lra\_contra}_{>}$$

$$\frac{\Gamma \vdash p \geq 0 \quad [p < 0]}{\Gamma \vdash \bot}\ \textsf{lra\_contra}_{\geq}$$

$$\frac{\Gamma \vdash p \neq 0 \quad [p = 0]}{\Gamma \vdash \bot}\ \textsf{lra\_contra}_{\neq}$$

B.4 Multiplication Rules

$$\frac{\Gamma \vdash p = 0}{\Gamma \vdash (c \cdot p){\downarrow} = 0}\ \mathsf{lra\_mult}_{c=}$$

$$\frac{\Gamma \vdash p > 0 \quad [c > 0]}{\Gamma \vdash (c \cdot p){\downarrow} > 0}\ \mathsf{lra\_mult}_{c>}$$

$$\frac{\Gamma \vdash p \geq 0 \quad [c \geq 0]}{\Gamma \vdash (c \cdot p){\downarrow} \geq 0}\ \mathsf{lra\_mult}_{c\geq}$$

$$\frac{\Gamma \vdash p \neq 0 \quad [c \neq 0]}{\Gamma \vdash (c \cdot p){\downarrow} \neq 0}\ \mathsf{lra\_mult}_{c\neq}$$

B.5 Addition Rules

$$\frac{\Gamma_1 \vdash p_1 = 0 \quad \Gamma_2 \vdash p_2 = 0}{\Gamma_1, \Gamma_2 \vdash (p_1 + p_2){\downarrow} = 0}\ \mathsf{lra\_add}_{==}$$

$$\frac{\Gamma_1 \vdash p_1 > 0 \quad \Gamma_2 \vdash p_2 > 0}{\Gamma_1, \Gamma_2 \vdash (p_1 + p_2){\downarrow} > 0}\ \mathsf{lra\_add}_{>>}$$

$$\frac{\Gamma_1 \vdash p_1 \geq 0 \quad \Gamma_2 \vdash p_2 \geq 0}{\Gamma_1, \Gamma_2 \vdash (p_1 + p_2){\downarrow} \geq 0}\ \mathsf{lra\_add}_{\geq\geq}$$

$$\frac{\Gamma_1 \vdash p_1 = 0 \quad \Gamma_2 \vdash p_2 > 0}{\Gamma_1, \Gamma_2 \vdash (p_1 + p_2){\downarrow} > 0}\ \mathsf{lra\_add}_{=>}$$

$$\frac{\Gamma_1 \vdash p_1 = 0 \quad \Gamma_2 \vdash p_2 \geq 0}{\Gamma_1, \Gamma_2 \vdash (p_1 + p_2){\downarrow} \geq 0}\ \mathsf{lra\_add}_{=\geq}$$

$$\frac{\Gamma_1 \vdash p_1 > 0 \quad \Gamma_2 \vdash p_2 \geq 0}{\Gamma_1, \Gamma_2 \vdash (p_1 + p_2){\downarrow} > 0}\ \mathsf{lra\_add}_{>\geq}$$

$$\frac{\Gamma_1 \vdash p_1 = 0 \quad \Gamma_2 \vdash p_2 \neq 0}{\Gamma_1, \Gamma_2 \vdash (p_1 + p_2){\downarrow} \neq 0}\ \mathsf{lra\_add}_{=\neq}$$

B.6 Subtraction Rules

$$\frac{\Gamma_1 \vdash p_1 = 0 \quad \Gamma_2 \vdash p_2 = 0}{\Gamma_1, \Gamma_2 \vdash (p_1 - p_2){\downarrow} = 0}\ \mathsf{lra\_sub}_{==}$$

$$\frac{\Gamma_1 \vdash p_1 > 0 \quad \Gamma_2 \vdash p_2 = 0}{\Gamma_1, \Gamma_2 \vdash (p_1 - p_2){\downarrow} > 0}\ \mathsf{lra\_sub}_{>=}$$

$$\frac{\Gamma_1 \vdash p_1 \geq 0 \quad \Gamma_2 \vdash p_2 = 0}{\Gamma_1, \Gamma_2 \vdash (p_1 - p_2){\downarrow} \geq 0}\ \mathsf{lra\_sub}_{\geq=}$$

$$\frac{\Gamma_1 \vdash p_1 \neq 0 \quad \Gamma_2 \vdash p_2 = 0}{\Gamma_1, \Gamma_2 \vdash (p_1 - p_2){\downarrow} \neq 0}\ \mathsf{lra\_sub}_{\neq=}$$

B.7 Term Normalization Rules

In the rules below c_t and c_p denote the same rational constant, in one case considered of term type and in the other as of polynomial type (similarly for the variables v_t and v_p).

$$\frac{}{\vdash c_\mathrm{t} = c_\mathrm{p}}\ \textsf{poly\_norm\_const}$$

$$\frac{}{\vdash v_\mathrm{t} = v_\mathrm{p}}\ \textsf{poly\_norm\_var}$$

$$\frac{\Gamma_1 \vdash t_1 = p_1 \quad \Gamma_2 \vdash t_2 = p_2}{\Gamma_1, \Gamma_2 \vdash t_1 + t_2 = (p_1 + p_2){\downarrow}}\ \textsf{poly\_norm}_+$$

$$\frac{\Gamma_1 \vdash t_1 = p_1 \quad \Gamma_2 \vdash t_2 = p_2}{\Gamma_1, \Gamma_2 \vdash t_1 - t_2 = (p_1 - p_2){\downarrow}}\ \textsf{poly\_norm}_-$$

$$\frac{\Gamma \vdash t = p}{\Gamma \vdash c_\mathrm{t} \cdot t = (c_\mathrm{p} \cdot p){\downarrow}}\ \textsf{poly\_norm}_{c\cdot}$$

$$\frac{\Gamma \vdash t = p}{\Gamma \vdash t \cdot c_\mathrm{t} = (p_\mathrm{p} \cdot c){\downarrow}}\ \textsf{poly\_norm}_{\cdot c}$$

B.8 Equation Normalization Rules

$$\frac{\Gamma \vdash t_1 - t_2 = p}{\Gamma \vdash t_1 = t_2 \Leftrightarrow p = 0}\ \textsf{poly\_norm}_=$$

$$\frac{\Gamma \vdash t_1 - t_2 = p}{\Gamma \vdash t_1 > t_2 \Leftrightarrow p > 0}\ \textsf{poly\_norm}_>$$

$$\frac{\Gamma \vdash t_1 - t_2 = p}{\Gamma \vdash t_1 \geq t_2 \Leftrightarrow p \geq 0}\ \textsf{poly\_norm}_\geq$$

$$\frac{\Gamma \vdash t_2 - t_1 = p}{\Gamma \vdash t_1 < t_2 \Leftrightarrow p > 0}\ \textsf{poly\_norm}_<$$

$$\frac{\Gamma \vdash t_2 - t_1 = p}{\Gamma \vdash t_1 \leq t_2 \Leftrightarrow p \geq 0}\ \textsf{poly\_norm}_\leq$$

$$\frac{\Gamma \vdash t_2 - t_1 = p}{\Gamma \vdash t_1 \neq t_2 \Leftrightarrow p \neq 0}\ \textsf{poly\_norm}_\neq$$

$$\frac{\Gamma_1 \vdash t_1 \sim t_2 \quad \Gamma_2 \vdash t_1 \sim t_2 \Leftrightarrow p \sim 0}{\Gamma_1, \Gamma_2 \vdash p \sim 0}\ \textsf{poly\_form}$$