Formalization of a Proof Calculus for Incremental Linearization for Satisfiability Modulo Nonlinear Arithmetic and Transcendental Functions

Tomaz Mascarenhas*
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
tomgm1502@gmail.com

Harun Khan*
Stanford University
Stanford, USA
harun19@stanford.edu

Abdalrhman Mohamed
Stanford University
Stanford, USA
abdal@stanford.edu

Andrew Reynolds
The University of Iowa
Iowa City, USA
andrew.j.reynolds@gmail.com

Haniel Barbosa
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
hbarbosa@dcc.ufmg.br

Clark Barrett
Stanford University
Stanford, USA
barrettc@stanford.edu

Cesare Tinelli
The University of Iowa
Iowa City, USA
cesare-tinelli@uiowa.edu

Abstract

Determining the satisfiability of formulas involving nonlinear real arithmetic and transcendental functions is necessary in many applications, such as formally verifying dynamic systems. Doing this automatically generally requires costly and intricate methods, which limits their applicability. In the context of SMT solving, *Incremental Linearization* was introduced recently to facilitate reasoning on this domain, via an incomplete but easy to implement and highly effective approach. The approach, based on abstraction-refinement via an incremental axiomatization of the nonlinear and transcendental operators, is currently implemented in the SMT solvers MathSAT and cvc5. The cvc5 implementation is also proof-producing. This paper presents two contributions: a formalization in the Lean proof assistant of the proof calculus employed by cvc5, and an extension of the lean-smt [12] plugin to reconstruct the proofs produced by cvc5 using this proof calculus. These contributions ensure the soundness of the proof calculus, making the underlying algorithm more trustworthy. Moreover, they allow users to check cvc5 results obtained via incremental linearization, as well as improve Lean's automation for problems in nonlinear arithmetic. We discuss how we modeled the rules in the proof assistant and the challenges encountered while formalizing them, as well as the issues in reconstructing proofs involving these rules in Lean, and how we solved them.

Keywords: SMT solvers, Lean, Transcendental functions, Proof generation

ACM Reference Format:
Tomaz Mascarenhas, Harun Khan, Abdalrhman Mohamed, Andrew Reynolds, Haniel Barbosa, Clark Barrett, and Cesare Tinelli. 2026. Formalization of a Proof Calculus for Incremental Linearization for Satisfiability Modulo Nonlinear Arithmetic and Transcendental Functions. In *Proceedings of the 15th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '26), January 12–13, 2026, Rennes, France.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3779031.3779111

1 Introduction

SMT solvers are crucial reasoning engines for reasoning about the correctness and security of safety-critical applications. However, their complexity can lead to implementation errors, raising concerns about their trustworthiness. To mitigate this problem, proof-producing SMT solvers, such as cvc5 [1], generate machine-checkable certificates [2], allowing for the independent verification of results. While SMT solvers routinely reason over a wide range of theories, it is a significant challenge to extend them to reason in nonlinear real arithmetic (NRA) since state-of-the-art complete solvers for the satisfiability of NRA formulas exhibit a doubly exponential worst-case time complexity [6]. Adding transcendental functions such as the exponential and sine functions increases the complexity of the satisfiability problem to the point of undecidability [14]. Despite these challenges, given the prevalence of constraints from the theory of non-linear arithmetic with transcendental functions (NTA) in various

*Tomaz Mascarenhas and Harun Khan contributed equally to this work.

CPP '26, January 12–13, 2026, Rennes, France

ACM ISBN 979-8-4007-2341-4/2026/01
https://doi.org/10.1145/3779031.3779111

engineering disciplines, including dynamic systems, digital signal processing, and motion planning, being able to reason about them is highly desirable.

The SMT solver cvc5 handles NTA via an *incremental linearization* method introduced by Cimatti et al. [3], which approximates transcendental functions using linear polynomials. The technique can also be used for NRA, providing an alternative to complete techniques [9]. Although incremental linearization does not guarantee termination, it offers a scalable alternative when exact solutions are not strictly necessary. The approach uses an abstraction refinement loop over abstractions of the input formula obtained by treating nonlinear operations and transcendental functions as uninterpreted. The abstractions are refined with the incremental addition, through a lemma-on-demand mechanism, of linear constraints that express increasingly tighter upper and lower bounds on those functions. In practice, this works well since proving properties in nonlinear systems can typically be handled by constructing a piecewise-linear invariant. Also, many systems are only nonlinear in a small part of their domain; hence, reasoning over them can be effectively handled by linear approximations. Moreover, the incremental linearization is performed on demand, tightening the envelope around a transcendental function only as necessary. This makes the approach less costly, if more brittle, than exact NRA and NTA reasoning.

The original proof calculus for incremental linearization [3], initially implemented in the MathSAT solver [4], provides a theoretical foundation for reasoning about nonlinear and transcendental arithmetic. The cvc5 implementation introduces variations to the original proof rules that enable a better integration in cvc5 and accommodate internal optimizations. In this work, we formalize the cvc5 variant of the proof calculus. This involves formalizing and proving within the Lean Theorem Prover [7] all NRA and NTA proof rules used by cvc5 in terms of Lean's native datatypes, thus verifying the soundness of the proof calculus. Since there are fewer NRA rules, this paper primarily focuses on NTA rules. All of our work on this formalization is publicly available in a git repository.[1]

The formalization effort was motivated by the observation that, while conceptually straightforward and easy to implement in principle, incremental linearization is actually tricky to get right in practice. This point was already recognized and discussed by the original authors in a later paper [8] that provides more details on how to generate sound linearization lemmas for transcendental functions. But it was also confirmed during our formalization in Lean, as we identified implementation errors and mistakes and missing hypotheses in the documentation of several proof rules in the cvc5 calculus, highlighting the usefulness of a mechanized formalization.[2] The formalization relies heavily on `mathlib`, the mathematical library for Lean, which provides a rich set of mathematical structures and theorems [11]. Some of cvc5's proof rules can be formulated as applications of theorems already in `mathlib`. Other rules must instead be formalized through Lean tactics due to their intricate nature. A third set of rules all involve bounding transcendental functions using Taylor polynomials. Proving those sound turned out to be the most challenging task of our formalization work as it required developing a new theory in Lean to bridge the gap between `mathlib` and those rules.

The second contribution we present here is an extension of LEAN-SMT [12], a hammer-like tool for Lean, with the rules in this calculus. The tactic works by *proof reconstruction*, where proofs from cvc5 are reconstructed step-by-step in Lean. This requires every proof rule and rewrite rule used by cvc5 to be formally verified in Lean to ensure the correct Lean theorem is applied at each step. LEAN-SMT already had the capability of reconstructing proofs in several theories, including uninterpreted functions and linear arithmetic. By leveraging our formalization, now it can also reconstruct proofs for problems in NRA and NTA solved via incremental linearization. This has two major benefits: the first is that we can confirm that we have proved the correct theorems in our formalization. If this were not the case, then we would see a mismatch between the proposition expected for a proof step and the statement of our theorem, making the reconstruction fail. We have seen this error happening and corrected our theorems, which are now succesfully being applied in the reconstruction. The second one is enhancing automation in Lean itself for reasoning over the theories we are supporting. As far as we know, the only tactics that support nonlinear arithmetic in Lean are `nlinarith` and `grind`, and both of them only do linear solving enhanced with elementary lemmas about multiplication and there are no tactics directly supporting transcendental functions. Thus, our extension of LEAN-SMT can potentially improve Lean's usability.

Related Work. To the best of our knowledge, this is the first work that provides a method for certifying results obtained via incremental linearization. Other approaches for nonlinear arithmetic have been formally verified: Mahboubi et al. [10] implemented Tarski's original quantifier elimination procedure [16] and proved the soundness of their Rocq [5] implementation. This constituted the first fully formalized complete method for nonlinear real arithmetic. While complete, Tarski's method is notoriously inefficient and therefore unsuitable for use in practical SMT solving. More recently, Scharager et al. [15] developed a formally verified implementation of the quadratic virtual substitution method [17] in Isabelle/HOL [13], providing an efficient and

[1] https://github.com/ufmg-smite/lean-smt/tree/3c6b049/Smt/Reconstruct/Real/TransFns.

[2] Going forward, we plan to work with the cvc5 team to address the issues discovered in the cvc5 proof rules.

complete decision procedure for formulas whose polynomial degrees are at most 2. These developments take a different approach from ours: they verify the implementation of the method directly within the proof assistant and establish its correctness once and for all. This avoids the need to re-check certificates each time the procedure runs, but it also freezes the implementation, as any change or optimization in the algorithm requires reproving its correctness. In contrast, our work certifies the output of an external solver through proof reconstruction, enabling trusted reasoning while allowing the SMT solver to evolve independently.

Outline. In Section 2, we describe the mathematical background that underlies the proof rules. In Section 3 and Section 4 we explain how we formalized and proved their soundness in Lean. During this process, we identified and proved additional theorems, which we contributed to `mathlib`,[3] particularly for rules bounding the exponential and sine functions using Taylor polynomials. In Section 5 we give more details on how proof reconstruction works in LEAN-SMT and present some of the challenges we had to solve in order to extend it with our work. Finally, Section 6 summarizes our work and discusses future directions.

2 Background

In this section we introduce the main mathematical concepts involved in our formalization. We also give a brief introduction to how they are modeled in `mathlib`. We focus on two transcendental functions: the sine (sin) function and the natural exponential function e^x (exp). All of the other transcendental functions supported by cvc5 are handled via well-known reductions to one of these two functions; thus, we will not discuss them further here.

2.1 Real Numbers

As transcendental functions, both exp and sin can produce irrational numbers for rational inputs. Also, more importantly, the incremental linearization algorithm is fundamentally based on approximating these functions by their Taylor series, which requires a setting where limits are well-defined. Since rational numbers alone do not form a complete space, we need a rigorous representation of real numbers to properly state the proof rules.

There are multiple ways to construct the real numbers formally. In `mathlib`,[4] they are defined as equivalence classes of Cauchy sequences of rational numbers. This particular choice has the upside of facilitating formal manipulations. More precisely, a sequence $a : \mathbb{N} \to \mathbb{Q}$ is called a *Cauchy* sequence if its terms become arbitrarily closer to each other as the index increases. That is, a is Cauchy if:

$$\forall \epsilon > 0.\, \exists i \in \mathbb{N}.\, \forall j \geq i.\, |a(i) - a(j)| < \epsilon$$

[3] https://github.com/leanprover-community/mathlib4/pull/22790.
[4] https://github.com/leanprover-community/mathlib4/blob/bump/v4.17.0/Mathlib/Data/Real/Basic.lean.

It is known that the set of limits of all Cauchy sequences corresponds to the set of real numbers. However, multiple Cauchy sequences may converge to the same limit. For example, the sequences $a(i) = \frac{1}{i}$ and $b(i) = 0$ both converge to 0. This gives rise to a natural equivalence relations between Cauchy sequences where two sequences a and b are equivalent ($a \simeq b$) if their difference converges to zero. Formally:

$$a \simeq b \iff \forall \epsilon > 0.\, \exists i \in \mathbb{N}.\, \forall j \geq i.\, |a(j) - b(j)| < \epsilon$$

The set $\mathbb{R}$ of all real numbers can be then defined as the quotient of all Cauchy sequences with respect to this equivalence relation.

2.2 The number π

The transcendental number π, which has a variety of important properties, is closely related to our objectives since it defines the points where the sin function vanishes and where its concavity changes.

Recall that $\cos(\frac{\pi}{2}) = 0$. In `mathlib`, the *intermediate value theorem* is used to prove that there is a real number in the real interval $[1, 2]$ whose cosine is equal to 0. It is also proved that there is a unique number with this property. π is then defined as the double of that number. Notice that this is not a constructive definition, meaning that it does not directly provide a way to compute, even approximately, the real value of π. This is an obstacle as the incremental linearization algorithm relies on increasingly more accurate approximations of π to compute bounds for sin. Luckily, `mathlib` comprises results bounding π with up to 20 decimal digits of precision, obtained by indirect methods. This means that we can correctly reconstruct any result that requires bounding π with up to this level of precision. There are methods for getting arbitrary precision but they are exceedingly slow beyond 20 decimal digits. In any case, this is not a limitation for our framework because cvc5 currently supports only up to 10 decimal digits of π.

2.3 Exponential and Sine

The two transcendental functions handled by incremental linearization are sin and exp. To formally define them, we first define the exponential function $\exp_c$ over the complex numbers via its power series:

$$\exp_c(z) := \sum_{k=0}^{\infty} \frac{z^k}{k!}$$

We can define the complex version of sin (which can also be derived from the trigonometric definition of sin using *Euler's formula*) in terms of $\exp_c$:

$$\sin_c(z) := (\exp_c(-z * i) - \exp_c(z * i)) * \tfrac{i}{2}$$

When the argument is real, both $\exp_c$ and $\sin_c$ are themselves real numbers. Therefore, the real functions exp and sin can be defined simply by restricting the complex functions to real inputs:

$$\exp(x) := \exp_c(x)$$

$$\sin(x) := \sin_c(x)$$

where x is a real number and Re is the function that maps a complex number to its real component.

2.4 Derivatives

Incremental linearization relies on expansions into Taylor polynomials, which in turn rely deeply on the notion of *derivative*. We recall that the (first) derivative of a real-valued function $f : \mathbb{R} \to \mathbb{R}$ is a partial function $f^{(1)} : \mathbb{R} \to \mathbb{R}$ that, when defined at point p, measures f's instantaneous rate of change at that point, that is, the slope of the tangent line to f's graph at p. For all $x \in \mathbb{R}$, $f^{(1)}(x)$ is defined informally as the following limit

$$f^{(1)}(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} .$$

If that limit exists at a point x we say that x is *differentiable* at x. The second derivative $f^{(2)}$ of f is the first derivative of $f^{(1)}$. More generally, the $(n+1)^{\text{th}}$ derivative of f is the derivative of $f^{(n)}$.

The exponential function exp is differentiable everywhere and is equal to its derivative: $\exp^{(1)}(x) = \exp(x)$ for all $x \in \mathbb{R}$. The sin function is also differentiable everywhere and its derivatives satisfy the follow properties for all $x \in \mathbb{R}$:

$$\sin^1(x) = \cos(x)$$
$$\sin^2(x) = -\sin(x)$$
$$\sin^3(x) = -\cos(x)$$
$$\sin^4(x) = \sin(x)$$

In `mathlib`, the notion of derivative is provided as a special case of a much broader notion, that of a Frechet's derivative, which applies to functions from and to infinite-dimensional spaces. However, the library contains all theorems about derivatives that we need for our formalization purposes.

2.5 Taylor Polynomials

As mentioned, the derivative of a one-variable function f differentiable at a point a is the slope of the tangent of f at a. It is known that this tangent line is the best possible linear approximation for f around a, in the sense that it is the unique linear function L such that the following equation holds:

$$\lim_{x \to a} \frac{f(x) - L(x)}{x - a} = 0$$

This concept is generalized by the *Taylor polynomial* of a function around a point.

Given a point $a \in \mathbb{R}$, a natural number d and a function $f : \mathbb{R} \to \mathbb{R}$ d times differentiable at a, its d-th Taylor polynomial $P_{f,d}$ at a is given by the following formula:

$$P_{f,d}(x) = \sum_{k=0}^{d} \frac{f^{(k)}(a)}{k!}(x-a)^k$$

If a is equal to 0, this is known as the *Maclaurin polynomial* of f.

The Taylor polynomial of degree d gives the best possible approximation for f around the point a among all polynomials of degree d or less (the tangent line is a special case). This result is made even more powerful by *Taylor's theorem*, which describes the error between the approximation and the actual function. In particular, *Taylor's theorem with the Lagrange form of the remainder* gives the following formula for the error $f(x) - P_{f,d}(x)$ when $x < a$, assuming f is of class C^d on the interval $[x, a]$ and $f^{(d)}$ is differentiable on the interval (x, a):

$$\exists x' \in [x, a].\, f(x) - P_{f,d}(x) = f^{(d+1)}(x') * (x-a)^{d+1} * \tfrac{1}{(d+1)!}$$

For point $x > a$, the same result applies but x' will be in the interval $[a, x]$. The limitation of this formula is that it depends on x', and we don't know its exact value, only an interval bounding it. Assuming that $f^{(d+1)}$ is continuous, we can take the value of x' in $[x, a]$ that maximizes the formula and derive an upper bound (and a lower bound) for f in this interval. This is a very powerful tool for approximating complicated but differentiable functions, and it lies at the core of the incremental linearization algorithm.

`Mathlib` contains all these results exactly as stated above but only for the case where $a < x$. However, both cases are necessary for proving cvc5's rules. In Section 4 we show the steps we took to extend the library to include both cases.

2.6 Convexity

A function $f : \mathbb{R} \to \mathbb{R}$ is *convex* (resp. *concave*) in an interval $[l, r]$ if, for any two points x_1 and x_2 in $[l, r]$, the graph of f is below (resp. above) the secant line between the points $(x_1, f(x_1))$ and $(x_2, f(x_2))$ in the Cartesian plane. One way to formally state this is as follows. To start, we can describe the line segment between $(x_1, f(x_1))$ and $(x_2, f(x_2))$ as $(tx_1 + (1-t)x_2, tf(x_1) + (1-t)f(x_2))$, where t ranges from 0 to 1. For a given $t \in [0, 1]$, the projection of this segment onto the x-axis is $tx_1 + (1-t)x_2$. Thus, the condition for convexity can be expressed as the next inequality. Concavity is expressed similarly but with $\geq$.

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$$

If f is twice differentiable in $[l, r]$, then the condition for f being convex (resp., concave) in $[l, r]$ is equivalent to its second derivative being nonnegative (resp., nonpositive) in $[l, r]$.

Once again, the definition of convexity in `mathlib` is broader as it refers to any function whose domain and codomain satisfy certain algebraic properties — which are satisfied by $\mathbb{R}$. Nevertheless, the general definition does reduce to the one above for functions from $\mathbb{R}$ to $\mathbb{R}$.

3 Basic Rules

In this section and in the next one, we describe the main rules in cvc5's proof calculus[5] for the incremental linearization algorithm and present our formalization of them in Lean. The calculus is used by cvc5 to produce proof certificates for unsatisfiable sets of NRA and NTA formulas.

The calculus contains 22 rules that range from basic properties of the sin and exp functions to bounds given by their Taylor polynomials. Table 1 gives a comprehensive list of these rules. The calculus also contains a number of rules for reducing formulas containing transcendental functions to equisatisfiable formulas whose transcendental functions are exclusively sin and exp. We will not describe these reduction rules here since they are pretty standard (e.g., replacing occurrences of $\cos(t)$ with $\sin(t+\frac{\pi}{2})$, occurrences of $\log(t)$ with a fresh variable l subject to the global constraint $\exp(l) = t$, and so on).

Instead, we start with a description of the rules[6] `MULT_TANGENT`, `MULT_SIGN`, `TRANS_SINE_SHIFT` and `TRANS_PI` in this section, and rules `TRANS_EXP_APPROX_BELOW`, `TRANS_EXP_APPROX_ABOVE_POS`, `TRANS_SINE_APPROX_BELOW_NEG` and `TRANS_SINE_APPROX_BELOW_POS` in the next.

We present the statements of the rules in the same format used in cvc5's documentation, that is, for a rule with conclusion ψ, premises $\psi_1, \cdots, \psi_n$, parameters $t_1, \cdots, t_n$ and side condition C, we write:

$$\frac{\psi_1, \cdots, \psi_n \mid t_1, \cdots, t_m}{\psi} \text{ if } C$$

Premises and conclusions are formulas, as usual in proof systems. A parameter can be a variable or more complex term/formula that occurs in the premises or the conclusion. A side condition is an (effectively verifiable) condition over the parameters.

The incremental linearization method works by abstracting nonlinear multiplication and transcendental functions as uninterpreted functions in the original set of formulas, solving the resulting problem with a solver for the theory of linear real arithmetic with uninterpreted functions (UFLRA) and then, if a *model* (i.e., a solution) is found, checking whether it satisfies the original set. If it does not, that is, if it is a *spurious* model, new formulas, or *refinement lemmas* are generated and added to the abstracted set that hold in NTA but are falsified by the model, effectively ruling it out. This process repeats until a genuine model is found or the abstracted set with the added lemmas is unsatisfiable in UFLRA.

The subsections are dedicated to discussing the challenges we encountered in formalizing in Lean the proof rules used in the refinement process and how we addressed them.

3.1 Non-linear monomials

If a model is spurious due to an inconsistency between the abstracted multiplication function and the real one, the method relies on a series of properties about multiplication, such as associativity, commutativity, monotonicity and so on (see [3, Fig. 4] for a full list), to produce a refinement lemma falsified by the model. We will not go into details here on how this is done, but depending on how exactly the model is inconsistent with the properties of multiplication, a particular quantifier-free instantiation of one or more of these lemmas is added to the formula.[7] The versions of these lemmas used by cvc5 have corresponding rules in its proof calculus. We have proven the soundness of these rules in NTA, which guarantees that the addition of refinement lemmas about multiplication will only rule out spurious models. We discuss a number of these proof rules next.

3.2 Bounding Multiplication By The Tangent Plane

First, we consider the rules that bound the multiplication of two variables by its tangent plane (i.e., the rules in the "Tangent plane" category in [3, Fig. 4]). At each point (a, b, ab) in the Cartesian space, the graph of the function $f(x, y) = xy$ has a tangent plane described by the expression $z = bx + ay - ab$. The graph of f is above that plane when $x - a$ has the same sign as $y - b$, and it is below otherwise. This is captured by the following pair of rules, with no premises, from cvc5's proof calculus:[8]

$$\frac{- \mid x, y, a, b}{xy \leq bx + ay - ab \leftrightarrow ((x \leq a \wedge y \geq b) \vee (x \geq a \wedge y \leq b))}$$

$$\frac{- \mid x, y, a, b}{xy \geq bx + ay - ab \leftrightarrow ((x \leq a \wedge y \leq b) \vee (x \geq a \wedge y \geq b))}$$

The following snippet shows how we encoded their soundness in Lean:

```
theorem arithMulTangentLower {α : Type}
    [LinearOrderedField α] (x y a b : α) :
    x * y ≤ b * x + a * y - a * b ↔ ((x ≤ a ∧ y ≥
    b) ∨ (x ≥ a ∧ y ≤ b))

theorem arithMulTangentUpper {α : Type}
    [LinearOrderedField α] (x y a b : α) :
    x * y ≥ b * x + a * y - a * b ↔ ((x ≤ a ∧ y ≤
    b) ∨ (x ≥ a ∧ y ≥ b))
```

[5] See https://cvc5.github.io/docs/cvc5-1.3.1/proofs/output_cpc.html. A complete list of the proof rules in the calculus can be found at https://cvc5.github.io/docs/cvc5-1.3.1/api/cpp/enums/proofrule.html. The semantics of the rules is also defined in the Eunoia logical framework, described in the user manual of the Ethos proof checker: https://github.com/cvc5/ethos/blob/main/user_manual.md.

[6] For readability we omit the prefix `ARITH_` from the original rule names.

[7] Due to internal optimizations, the version of the linearization method implemented in cvc5 uses slightly different versions of these lemmas, but the idea is the same.

[8] Reference: `MULT_TANGENT` in cvc5's documentation.

Instead of directly using reals, we proved a polymorphic version of the lemma that can be applied to any structure that satisfies the axioms of a *Linear Ordered Field*.

Both proofs are very similar so we focus on the first. For the right-to-left direction of the double implication we do case analysis on the disjunction; in both cases the theorem follows from nonlinear arithmetic reasoning, which can be handled by the `nlinarith` tactic in Lean. This tactic is a very light-weight extension over the original `linarith` tactic and handles only a few problems in nonlinear arithmetic. Luckily, it is powerful enough to prove our goal. For the left-to-right direction, we consider the relationship between $x - a$ and 0 and between $y - b$ and 0. Since each of these differences can be positive, negative, or zero, this results in 9 possible cases. Two of these are invalid: both $x - a < 0 \wedge y - b < 0$ and $x - a > 0 \wedge y - b > 0$, are contradictory with our hypothesis. We used again `nlinarith` to derive this contradiction. In the remaining seven cases, the conclusion follows directly from the assumptions established through case analysis, making the proof straightforward.

3.3 Sign of monomials

Next we consider the proof rules under the "Zero" category in [3, Fig. 4]. It is not hard to prove them in Lean as stated. However, cvc5 uses a more general version of the rules that determines the sign of xy based on the sign of x and y. Instead of considering only binary applications of multiplication, it considers the signs of monomials with an arbitrary number of variables. The precise statement of the equivalent rule in cvc5[9] is the following:

$$\frac{- \mid f_1, \cdots, f_k, m}{(f_1 \wedge \cdots \wedge f_k) \longrightarrow m \bowtie 0} \text{ if } C$$

where the side condition C requires that each f_i have the form $x_i \bowtie 0$ ($\bowtie$ being one of $<$, $>$ or $\neq$) and m be of the form $x_1^{p_1} x_2^{p_2} \cdots x_k^{p_k}$, for some natural numbers p_i. Also required is that any variable x_i whose corresponding parameter f_i has the form $x_i \neq 0$ have an even exponent. The operator on the conclusion is either $<$ or $>$ based on the parity of the exponents and the hypothesis f_i. This side condition would be quite complicated to state as a Lean theorem. Instead, we implemented a tactic, `arithMulSign`, for handling the entire rule. The tactic takes as input a list of variables that form the target monomial, the polarity of the hypothesis (whether each x_i is greater than, lesser than or not equal to 0) and the exponents of the variable. Here is an example of how the tactic can be used:

```
example (a b c : Real) : a > 0 → b < 0 → c > 0 →
    a ^ 2 * b ^ 3 * c ^ 3 < 0 :=
  by arithMulSign [a,b,c], [1,-1,1], [2,3,3]
```

To implement the tactic, we first proved theorems relating the parity of the exponent and the sign of the input with the sign of the result of the exponentiation:

```
variable {k : Nat} {r : Real}
theorem powNegOdd      : r < 0 → Odd k  → r ^ k < 0
theorem powNegEven     : r < 0 → Even k → r ^ k > 0
theorem nonZeroEvenPow : r ≠ 0 → Even k → r ^ k > 0
theorem powPos         : r > 0          → r ^ k > 0
```

Then we proved the following set of results, relating the sign of two variables with the sign of their product:

```
variable {a b : Real}
theorem combineSigns₁ : a > 0 → b > 0 → b * a > 0
theorem combineSigns₂ : a > 0 → b < 0 → b * a < 0
theorem combineSigns₃ : a < 0 → b > 0 → b * a < 0
theorem combineSigns₄ : a < 0 → b < 0 → b * a > 0
```

The implementation of `arithMulSign` essentially combines these two sets of results, selecting the correct sequence of theorems to apply depending on the context in which it was invoked.

The tactic itself cannot be used as an artifact to formalize the calculus. We claim, however, that the theorems used in the backend of the tactic form a theory that ensures the soundness of the proof rule. Moreover, the tactic can be used for proof reconstruction, as discussed in Section 5.

3.4 Shifting sine

The incremental linearization algorithm relies heavily on the convexity of the transcendental functions in order to compute bounds for them. Determining the convexity of exp at a given point is trivial, since it is always convex (i.e., its second derivative is positive). For sin, the question is more complicated since the sign of its second derivative depends on the exact value of π. One may be able to do it with an approximation of π, but if we are too far away from the origin of the real line we would have to multiply our approximation by some constant, increasing the error.

The linearization method deals with this issue by replacing the argument in each occurrence of sin by a fresh variable constrained to be in the range $[-\pi, \pi]$. A constraint is added to ensure that applying sin to the fresh variable has the same result as applying sin to the original argument. This step is formalized in cvc5's calculus using the following proof rule:[10]

$$\frac{- \mid x}{\exists y \in \mathbb{R}.\, \exists s \in \mathbb{Z}.\, -\pi \leq y \leq \pi \wedge \sin(x) = \sin(y) \wedge x = y + 2\pi s}$$

We model the rule with the following theorem:

```
theorem arithTransSineShift : ∀ (x : Real) , ∃ (y :
    Real) (s : Int),
```

[9] Reference: `MULT_SIGN` in cvc5's documentation.

[10] Reference: `TRANS_SINE_SHIFT` in cvc5's documentation.

```
-Real.pi ≤ y ∧ y ≤ Real.pi ∧ Real.sin x =
  Real.sin y ∧ y = x + 2 * Real.pi * s
```

The proof of the theorem goes as follows: Let $s = \lceil \frac{x-\pi}{2\pi} \rceil$ and $y = x - 2\pi s$. Then:

- The second conjunct ($\sin(x) = \sin(y)$) is a consequence of an existing result in `mathlib`, which is named `sin_sub_int_mul_two_pi` and ensures that adding an integer multiple of 2π to the argument of sin does not change its output.
- The third conjunct essentially states that $x = x - 2\pi s + 2\pi s$, which can be proven with `mathlib`'s theorem `sub_add_cancel`.
- The first conjunct consists of two statements: $-\pi \leq x - 2\pi\lceil \frac{x-\pi}{2\pi} \rceil$ and $x - 2\pi\lceil \frac{x-\pi}{2\pi} \rceil \leq \pi$. The second one can be proven by first reorganizing the inequality to $\frac{x-\pi}{2\pi} \leq \lceil \frac{x-\pi}{2\pi} \rceil$ and then applying an existing lemma in `mathlib` that ensures $a \leq \lceil a \rceil$ for any real a. The first statement can be reorganized to $\lceil \frac{x-\pi}{2\pi} \rceil \leq \frac{x+\pi}{2\pi}$. Note that $\lceil a \rceil \leq a + 1$, for any real a. We use this fact to transform the inequality to $\frac{x-\pi}{2\pi} + 1 \leq \frac{x+\pi}{2\pi}$. Now, the left side can be proven to be equal to the right one, and we can close the goal by invoking the reflexivity of $\leq$.

3.5 Bounding π

In the process of obtaining bounds for the sin function, the incremental linearization method computes increasingly more accurate lower and upper bounds for π. Its correctness depends on these bounds being valid. Generating these bounds corresponds to applications of the following rule:

$$\frac{- \mid l, u}{l \leq \pi \wedge \pi \leq u} \text{ if } C$$

where C requires l and u to be concrete constants that are valid lower and upper bounds for π. Applying this rule then requires inspecting that the parameters l and u are instantiated by concrete values which are respectively below and above π. We model the rule in Lean as a tactic that inspects the values received for l and u and succeeds if and only if they are valid bounds for π.

The implementation of the tactic relies on `mathlib` theorems that bound π above and below with an error smaller than 10^{-20}. Specifically, the tactic tries to apply `norm_num`, a powerful tactic for normalizing numerical expressions, to prove that its first input is smaller than `mathlib`'s lower bound on π and its second input is greater than the upper bound. If this fails, the tactic throws an error. Otherwise, it applies transitivity to yield a proof that its two inputs correctly bound π.

Currently, if the arguments passed to our tactic are within the range $[\pi - 10^{-20}, \pi + 10^{-20}]$, the tactic will, incorrectly, fail, making our method incomplete with respect to the rules. We leave as future work the extension of the techniques implemented in `mathlib` to compute bounds for π with arbitrary precision.

4 Bounding Transcendental Functions With Taylor Polynomials

In this section, we describe our formalization of seven proof rules in cvc5's calculus that bound transcendental functions with Taylor polynomials. Although `mathlib` includes a collection of theorems on Taylor polynomials, we needed to generalize and extend that collection to bridge the gap between the theorems and the proof rules. First, we present the foundation for the formalization of these rules.

4.1 Preliminaries

The main result that we use and build upon is Taylor's theorem with the Lagrange form of the remainder. The theorem applies to every (proper) real interval $[x_0, x]$ and function $f : \mathbb{R} \to \mathbb{R}$ that is in C^n on $[x_0, x]$ and whose n^{th} derivative is differentiable on (x_0, x). It states that:

$$f(x) - \left(\sum_{j=0}^{n} \frac{f^{(j)}(x_0)}{j!} (x - x_0)^j \right) = \frac{f^{(n+1)}(x')}{(n+1)!} (x - x_0)^{n+1} \quad (1)$$

for some $x' \in (x_0, x)$. With properly defined auxiliary functions, the theorem can be expressed in Lean as follows.

```
theorem taylor_mean_remainder_lagrange {f : ℝ → ℝ}
    {x x₀ : ℝ} {n : ℕ}
  (hx : x₀ < x) (hf : ContDiffOn ℝ n f (Icc x₀ x))
  (hf' : DifferentiableOn ℝ (iteratedDerivWithin n f
    (Icc x₀ x)) (Ioo x₀ x)) :
  ∃ (x' : ℝ) (_ : x' ∈ Ioo x₀ x), f x -
    taylorWithinEval f n (Icc x₀ x) x₀ x =
    iteratedDerivWithin (n + 1) f (Icc x₀ x) x' *
                        (x - x₀) ^ (n + 1) / (n + 1)!
```

The assumption $x_0 < x$ presents a barrier for the formalization of some of the cvc5 proof rules. For example, one of the rules bounds the exponential function from above on $[l, u] \cap \mathbb{R}^-$ by a Taylor polynomial centered at zero with even degree (i.e. $x_0 = 0$ and $x < 0$ holds). Hence, the first step in our formalization was to extend this theorem to the case where $x_0 > x$.

There are two approaches to this task. We can adjust the proof of the theorem for $x_0 > x$ or we can apply the current theorem to $f(-x)$ and prove theorems to handle the negative sign. The first approach is straightforward but results in code duplication. The second approach is more complex since it requires formalizing several theorems to pushing negations into derivatives and power series. Nevertheless, we chose the latter as it builds a more comprehensive and useful theory. A summary of our approach follows below.

We want to prove that, given a function f and points x and x_0 with $x < x_0$ and where f satisfies the derivability conditions in $[x, x_0]$, there is an $x' \in (x, x_0)$ such that Equation 1 holds. First, applying `taylor_mean_remainder_lagrange` to the function $g(x) := f(-x)$ using the points $-x_0$ and $-x$

gives us

$$g(-x) - \left(\sum_{j=0}^{n} \frac{g^{(j)}(-x_0)}{j!}(x_0 - x)^j\right) = \frac{g^{(n+1)}(x')}{(n+1)!}(x_0 - x)^{n+1} \tag{2}$$

for some $x' \in (-x_0, -x)$. Simplifying the equation using $g^{(i)}(-x) = (-1)^i f^{(i)}(-x)$ yields

$$f(x) - \left(\sum_{j=0}^{n} \frac{f^{(j)}(x_0)}{j!}(x - x_0)^j\right) = \frac{f^{(n+1)}(-x')}{(n+1)!}(x - x_0)^{n+1}. \tag{3}$$

which is the identity we wanted to prove for $x < x_0$. The argument involves many algebraic transformations mainly pushing negations into derivatives and sums. We highlight some important auxiliary lemmas used in the following section:

1. `taylorWithinEval_neg`: We show that the n-th Taylor polynomial of f centered at x_0 evaluated at x is equal to the n-th Taylor polynomial of g centered at $-x_0$ evaluated at $-x$. We first rewrite g as $f(-1 * x)$ and then apply a `mathlib` theorem that rewrites $f^{(i)}(-1*x)$ as $(-1)^i f^{(i)}(-1 * x)$ for the term of degree i in the sum. Since the Taylor polynomial is centered at $-x_0$ and evaluated at $-x$ we get another factor of $(-1)^i$ from the term $((-x) - (-x_0))^i$. After several algebraic manipulations, the result follows. One important point to keep in mind is that Taylor polynomials in `mathlib` are always associated with a specific subset S of the reals where the function is n-times differentiable. To facilitate the proof, we state this theorem with S being the entire real line for both polynomials.
2. `taylorWithinEval_eq`: Notice that in the statement of the theorem `taylor_mean_remainder_lagrange` the Taylor polynomial has $S = [x_0, x]$. The result we wish to prove also associates g with the same interval. However, to use the previous result, we need to extend the associated set S of g to the whole real line. For that, we prove `taylorWithinEval_eq` stating `(taylorWithinEval f n S x₀) = (taylorWithinEval f n Set.univ x₀)`. In other words, if f is infinitely differentiable everywhere (which is satisfied by our functions of interest, sin and exp), then the n-th Taylor polynomial of f centered at x_0 evaluated at x with $S = [x_0, x]$ is equal to the n-th Taylor polynomial of f centered at x_0 evaluated at x with $S = \mathbb{R}$.

The second generalization we made involves secants and convexity. Notably, the four proof rules bounding the exponential and sine functions by their Taylor polynomials from above all follow the same structure. For example, one rule[11] bounds exp from above with the secant of the Taylor polynomial centered at zero (which we denote as p), where the secant $g(x)$ is defined as the line passing through the points $(l, p(l))$ and $(u, p(u))$, and where l, u are given real values with $l \leq u$:

$$g(x) := \frac{p(l) - p(u)}{l - u} \cdot (x - l) + p(l)\ .$$

Taking $x = l$, the secant is equal to $p(l)$, and hence, the rule simply bounds $\exp(l)$ by $p(l)$. We need to at least recover this as a special case when proving the rule. In fact, it is sufficient to show only the special case because the extension to the general case is simply a consequence of convexity. Hence, as part of our preliminary theory-building we formalize this in the following theorems.

1. `le_of_ConvexOn`: If f is convex on some set S, then for all $\lambda \in [0, 1]$ and $x, z \in S$ with $x \leq z$, $f(\lambda x + (1 - \lambda)z) \leq \lambda f(x) + (1 - \lambda)f(z)$. Similar theorems exist in `mathlib`. However, we need this particular form. In fact, the proof simply uses the following `mathlib` theorem followed by algebraic manipulations using the `linarith` tactic.

```
theorem ConvexOn.secant_mono_aux2 (hf :
    ConvexOn _ s f) (hx : x ∈ S)
  (hz : z ∈ S) (hxy : x < y) (hyz : y < z) :
  (f y - f x) / (y - x) ≤ (f z - f x) / (z - x)
```

2. `le_convex_of_le`: If f is convex on some set S, and $l, t, u \in S$ with $l \leq t \leq u$, and f is bounded above by some function p at l and u, then

$$f(t) \leq \frac{p(l) - p(u)}{l - u} \cdot (t - l) + p(l)\ .$$

This is the main result where we go from an upper bound on f at l and u to an upper bound on f over the whole interval $[l, u]$. The proof proceeds by rewriting the right-hand side of the inequality in the form of `le_of_ConvexOn`. First, we consider the trivial case when $l = u$, which reduces to our assumption $f(l) \leq p(l)$. Next, we define $C := \frac{t-l}{u-l}$. Then through algebraic manipulations we get exactly `le_of_ConvexOn`. What is left to show is that $0 \leq C \leq 1$, which holds because t is in between l and u.

3. `ge_concave_of_ge`: If f is concave on some set S, and $l, t, u \in S$ with $l \leq t \leq u$, and f is bounded below by some function p at l and u, then

$$f(t) \geq \frac{p(l) - p(u)}{l - u} \cdot (t - l) + p(l).$$

We need this version to prove a rule bounding sin from below on the positive real numbers [12] since sin is concave in the region $[0, \pi]$, and we use a shifting argument to restrict to this region. The proof applies our lemma above `le_of_ConvexOn` to $-f$ since $-f$ is convex and proceeds analogously.

[11] Rule `TRANS_EXP_APPROX_ABOVE_NEG` in cvc5's documentation.

[12] Rule `TRANS_SINE_APPROX_BELOW_POS` in cvc5's documentation.

4.2 Formalizing cvc5's proof rules for bounding exp

We used our theory to formalize seven proof rules from cvc5, three for exp and four for sin, that bound those functions from above and below using Taylor polynomials. For brevity, we discuss only four representative cases demonstrating the main ideas behind the formalization.[13]

4.2.1 TRANS_EXP_APPROX_BELOW.

This rule bounds $\exp(t)$ from below on the entire number line using its Taylor polynomial centered at zero (i.e., the Maclaurin polynomial) with an odd degree d. Formally, it states the following:

$$\frac{- \mid d, c, t}{t \geq c \rightarrow \exp(t) \geq \texttt{maclaurin}(\exp, d, c)}$$

We formalize this rule by proving two theorems, one for the positive reals and the other for the negative reals.

```
theorem arithTransExpApproxBelowPos (d n : ℕ) (t : ℝ
    ) (_ : d = 2*n + 1) (hx : 0 < t) :
  Real.exp t ≥ taylorWithinEval Real.exp d Set.univ
    0 t

theorem arithTransExpApproxBelowNeg (d n : ℕ) (t : ℝ
    ) (h : d = 2*n + 1) (hx : t < 0) :
  Real.exp t ≥ taylorWithinEval Real.exp d Set.univ
    0 t
```

We handle two separate cases. For positive t, we rewrite the difference $\exp(t) - p(t)$ as the remainder term using `taylor_mean_remainder_lagrange`. Then we have to show that $\exp^{(d+1)}(t_1) * t^{d+1}/(d+1)!$ is positive for some $t_1 \in (0, t)$. This holds because the derivative of exp is itself, and exp is always positive. For negative t, we use our generalization `taylor_mean_remainder_lagrange'` (i.e. the version with $x < x_0$) and we are left with showing that $\exp^{(d+1)}(t_1) * t^{d+1}/(d+1)!$ is positive for some $t_1 \in (t, 0)$. This is true because exp is its own derivative and is always positive, and $d+1$ is even, implying t_1^{d+1} is also positive. We also give a straightforward proof for the case where $t = 0$, and combine all three theorems in a final theorem without any assumption on t's sign. With the appropriate theory in place, the proof is a straightforward application of several theorems, demonstrating the power of abstraction in theorem proving.

4.2.2 TRANS_EXP_APPROX_ABOVE_POS.

Let p be the Taylor polynomial of exp centered at zero with degree d. Let $p^*(t) := p(t)/(1 - t^{d+1}/(d+1)!)$. Then $\exp(t)$ is bounded above by the secant between $(l, p^*(l))$ and $(u, p^*(u))$ for $l \leq t \leq u$ and t positive under the assumption that $t^{d+1} < (d+1)!$. This assumption holds for d sufficiently large since $(d+1)! > \sqrt{\frac{d+1}{2}}^{d+1} > t^{d+1}$. Formally we encapsulate

$$\frac{- \mid d, t, l, u}{(t \geq l \wedge t \leq u) \rightarrow exp(t) \leq \texttt{secant-pos}(\exp, l, u, t)}$$

in the following theorem:

```
theorem arithTransExpApproxAbovePos (l u t : ℝ) (ht
    : l ≤ t ∧ t ≤ u)
  (hl : 0 < l) (hd : u^(d+1) < Nat.factorial (d+1)) :
  let r : ℕ → ℝ → ℝ := fun d => (fun t =>
    (1-t^(d+1)/(d+1)!))
  let p : ℕ → ℝ → ℝ := fun d =>
    ((taylorWithinEval Real.exp d Set.univ 0) / (r
    d))
  Real.exp t ≤ ((p d l - p d u) / (l - u)) * (t - l)
    + p d l
```

The main challenge is that the secant is divided by $(1 - t^{d+1}/(d+1)!)$. Using `le_convex_of_le` it suffices to show $\exp(t) \leq p^*(t)$ for $t > 0$ and $t^{d+1} < (d+1)!$. However, Taylor's theorem gives us a bound on the difference $\exp(t) - p(t)$ instead of $\exp(t) - p^*(t)$. We rewrite

$$\begin{aligned} \exp(t) - p^*(t) &= \exp(t) - \frac{p(t)}{1 - t^{d+1}/(d+1)!} \\ &= \frac{\exp(t) - p(t) - \exp(t)\frac{t^{d+1}}{(d+1)!}}{1 - \frac{t^{d+1}}{(d+1)!}} \end{aligned}$$

and then apply the bound to the first part of the numerator. We then obtain

$$\frac{\exp^{(d+1)}(x')\frac{t^{d+1}}{(d+1)!} - \exp(t)\frac{t^{d+1}}{(d+1)!}}{1 - t^{d+1}/(d+1)!} = \frac{(\exp(x') - \exp(t))\frac{t^{d+1}}{(d+1)!}}{1 - \frac{t^{d+1}}{(d+1)!}}$$

which is smaller than or equal to 0, since $x' \in (0, t)$ and $1 - \frac{t^{d+1}}{(d+1)!} > 0$. The proof uses some algebraic manipulations coupled with the fact that the derivative of exp is itself, which we proved earlier as `iteratedDeriv_exp`.

When proving this rule we encountered a few issues with cvc5. First, it was using a wrong formula for $p^*(t)$, namely $p(t)(1 + t^{d+1}/(d+1)!)$. Besides that, it was also producing instantiations of the rule with negative l, which is not supposed to happen. There were also mistakes in the documentation, which was missing the side condition $t^{d+1} < (d+1)!$, crucial in our proof since otherwise we would divide by a nonpositive number. Lastly, the documentation restricts d to be even, which is unnecessary. We notified the cvc5 developers of these inaccuracies, which were acknowledged and promptly fixed.

4.2.3 TRANS_SINE_APPROX_BELOW_NEG.

This rule bounds sin from below on the non-positive reals using its Taylor polynomial centered at zero minus an error term:

$$\frac{- \mid d, t, c, lb, ub}{(t \geq lb \wedge t \leq ub) \rightarrow \sin(t) \geq \texttt{lower}(\sin, c)}$$

The documentation of the rule counts Taylor polynomials differently (since even terms are zero in the Maclaurin series of sine) whereas our Lean definitions define d as the degree of the Taylor polynomial. The rule's documentation and code were unclear, particularly the definition of $\text{lower}(\sin, c)$. After refining multiple candidate statements, we arrived at the following definition: $\text{lower}(\sin, c)$ is the

[13] The formalizations of the remaining rules are similar and can be found in the appendix.

Taylor polynomial of sin evaluated at c minus $c^{d+1}/(d+1)!$ where $c = \text{argmin}_{x \in [lb,ub]} \sin x$. We proved the following theorem first before reasoning about c. We have validated that the theorem correctly captures the rule by successfully reconstructing proofs generated by cvc5, as discussed in Section 5.

```
theorem arithTransSineApproxBelowNeg_self
  (d k : Nat) (hd : d = 2*k + 1) (hx : x ≤ 0) :
  let p : ℝ → ℝ := taylorWithinEval Real.sin d
    Set.univ 0
  p x - x ^ (d + 1)/(d + 1).factorial ≤ sin x
```

We proceed analogously to TRANS_EXP_APPROX_BELOW's proof from Subsection 4.2.1. However, in this case, we use the theorem `taylor_mean_remainder_lagrange` to rewrite the difference $\sin(x) - p(x)$ as the remainder term. Hence we have to show that $\sin^{(d+1)}(x') * x^{d+1}/(d+1)! + x^{d+1}/(d+1)!$ is non-negative for $x' \in (x, 0)$. It suffices to show the derivatives of sin are at least -1 (since $d+1$ is even) which is true because sin and cos are bounded by -1 and 1 (a theorem in `mathlib`). To handle derivatives of sin we proved a general theorem encapsulating the periodicity modulo 4 which we present below.

```
theorem iteratedDeriv_sin_cos (n : Nat) :
  (iteratedDeriv n sin =
    if n % 4 = 0 then sin else
    if n % 4 = 1 then cos else
    if n % 4 = 2 then -sin else -cos) ∧
  (iteratedDeriv n cos =
    if n % 4 = 0 then cos else
    if n % 4 = 1 then -sin else
    if n % 4 = 2 then -cos else sin)
  := by induction' n with n ih
  · simp [iteratedDeriv]
  · simp [ih.1, ih.2, iteratedDeriv_succ']
    have :=  Nat.mod_lt n (show 4 > 0 by decide)
    interval_cases hn : n % 4
    <;> simp [hn, Nat.add_mod]
    <;> ext
    <;> simp [iteratedDeriv_neg, ih]
```

Notice that the theorem statement is long and, in fact, even lists the derivatives of cos. This is a deliberate inductive strategy, in which we state a sufficiently general statement for a powerful induction hypothesis that allows us to prove the inductive step with ease. In fact, the proof is quite short, as displayed above, and uses automation to handle all cases simultaneously.

Finally, we prove the main theorem that encapsulates the rule.

```
theorem arithTransSineApproxBelowNeg
  (d k : ℕ) (hd : d = 2*k + 1)
  (hl : -π ≤ lb) (hu : ub ≤ 0)
  (hx1 : lb ≤ x) (hx2 : x ≤ ub)
  (hc : c = if -π/2 < lb then lb
            else if - π/2 < ub then - π/2
            else ub) :
taylorWithinEval Real.sin d Set.univ 0 c - c ^ (d +
    1) / (d + 1).factorial ≤ sin x
```

We have $-\pi \leq lb$ and $ub \leq 0$ since the sine function is periodic as we showed in Subsection 3.4 and the solver uses the periodicity to restrict us to either $[-\pi, 0]$ or $[0, \pi]$. The proof follows by applying the previous theorem to the left-hand side leaving to prove $\sin c \leq \sin x$. We case-split on c, and use a theorem from `mathlib` that sin is monotonically increasing in $[-\pi/2, \pi/2]$, and $\sin \pi/2 = 1$ and the `linarith` tactic.

4.2.4 TRANS_SINE_APPROX_BELOW_POS.

This rule bounds sin from below on the positive reals.

$$\frac{- \mid d, t, lb, ub, l, u}{(t \geq lb \wedge t \leq ub) \rightarrow \sin(t) \geq \text{secant}^*(\sin, l, u, t)}$$

where lb, ub are symbolic lower and upper bounds on t and l, u are evaluated lower and upper bounds on t and secant^* is defined below. We formalize this rule as the following theorem where we stipulate $0 < l \leq t \leq u \leq \pi$ since the linearization procedure stays within the same convexity region of sin.

```
theorem arithTransSineApproxBelowPos (d : Nat) (l u
    t : ℝ)
  (ht : l ≤ t ∧ t ≤ u) (hl : 0 < l) (hu : u ≤ π):
  let p : ℝ → ℝ :=
    fun t => taylorWithinEval Real.sin d Set.univ 0
    t - (t ^ (d + 1) / (d + 1).factorial)
  ((p l - p u) / (l - u)) * (t - l) + p l ≤ sin t
```

Applying `ge_concave_of_ge` it suffices to show $\sin t \geq p(t)$ for $0 < t$. Let $p^*(t)$ be the Taylor Polynomial of degree d. We rewrite the difference $\sin t - p^*(t)$ as the remainder term using `taylor_mean_remainder_lagrange'` leaving the error term as is. Then we have to show that $\sin^{(d+1)}(t_1) * t^{d+1}/(d+1)! + t^{d+1}/(d+1)!$ is nonnegative for $t_1 \in (0, t)$. This is true because the derivatives of sin are always between 1 and -1. So we split $(d+1)\%4$ into four cases using `iteratedDeriv_sin_cos` and handle each case simultaneously using `linarith`.

Our theorem is more general than the cvc5 rule since we do not require d to be odd. We are able to prove the theorem for all d using an automated framework using `linarith` that demonstrates the power of theorem proving. Moreover, using this framework we can easily check whether the other theorems can be generalized. If we delete the hypothesis on the parity of d we give `simp` the fact that the derivatives of sin are always between 1 and -1, powers of negative numbers are negative if and only if $d+1$ is odd, and we use `linarith`. If all cases succeed then we did not need the hypothesis in the first place.

5 Proof Reconstruction

In this section we describe how proof reconstruction works in LEAN-SMT and the challenges in extending the tool with our specific set of rules, as well as our solutions and their current limitations. Finally we present a set of test cases we used to validate proof reconstruction with our formalized proof calculus.

Given either a problem in SMT-LIB or a Lean goal, LEAN-SMT is capable of invoking cvc5 on the former or in a translation of the latter. If the solver succeeds in proving the problem unsatisfiable, the resulting proof is translated into Lean's language and checked with its kernel. This is done via a process known as *proof replay*: each proof rule in cvc5's proof is sequentially translated into either the application of a previously proved theorem or the invocation of a tactic, which is then checked by Lean's kernel and added as a lemma to the context of the proof, if succesful. If a rule has a side condition, the corresponding theorem will have an extra hypothesis matching it. LEAN-SMT will then try to find automatically a proof for this hypothesis (via other predefined tactics) in order to apply the theorem. The replay of the subsequent steps will rely on this lemma, which means that it is very likely that the whole process will fail if the statement of a theorem is not matching cvc5's expectations. After all the steps are processed, the tool checks whether the conclusion derived in the end matches the original goal.

Challenges in the reconstruction. As we pointed out in Sections 3 and 4, during the process of extending LEAN-SMT to use our formalization we realized that we had misinterpreted the documentation of several rules, as the output of cvc5 did not match what was expected by our theorems. In order to refine the statements, we had to investigate cvc5's code base and run multiple tests observing its output. After solving these issues, the next main difficulty was the noncomputable nature of several elements used in the rules. First, the definition of the Maclaurin polynomial from `mathlib` does not allow computing the value of the polynomial at a specific point. This is a deliberate design decision, as it facilitates manipulating the polynomial in proofs. Therefore when one of the proof rules has e.g. a conclusion of the form $f(t) \geq \texttt{maclaurin}(f, d, c)$, the rest of the proof will require that the right-hand side is actually the corresponding rational number and not the unevaluated function application. We solved this issue by defining a computable version of these polynomials and proving that both definitions evaluate to the same number at every point. For instance, for the exponential the definition and the theorem are the following:

```
def expTaylor (d : Nat) (x : Real) : Real :=
  match d with
  | 0 => 1
  | d + 1 =>
    expTaylor d x + (x ^ (d + 1)) / (d + 1).factorial

theorem expEmbedding (d : Nat) (x : Real) :
    taylorWithinEval Real.exp d Set.univ 0 x =
     expTaylor d x
```

This was not hard to prove, given all the facts we had already proven to formalize the calculus.

A second noncomputability issue is related to the real numbers. Multiple rules have side conditions of the form $l < r$, where l and r are real numbers. It is known that it is not possible to decide this kind of comparison in general [14]. Nevertheless, for most of the rules it is guaranteed that both l and r will be rational numbers, so we can automate the proof for their side conditions with tactics like `norm_num`. Some rules require proving that a parameter is greater than $-\pi$ or smaller than π. We could automate the proof of such goals by leveraging the tactic that does linear reasoning in Lean extended with lemmas bounding π up to 20 decimal places. This, like the issue described in Section 3.5, will not work for parameters arbitrarily close to π, and we leave as future work to extend this technique to find proofs for arbitrary parameters.

5.1 Validating the Formalization

For these tests, we used the version of cvc5 updated with our corrections, which is the one in the main fork with hash `d509acd` and we used as a basis the commit `db6a7ca` of the main fork of LEAN-SMT. We tested the reconstruction in benchmarks coming from the original incremental linearization paper [3]. Our goal was to find at least one goal that exercised each one of the proof rules in the calculus. For all the problems we considered, all the steps corresponding to rules in this calculus were successfully reconstructed. We could not use SMT-LIB problems, since NTA is not an official SMT-LIB theory, and thus it does not contain any problems compatible with it. Since some rules were not used by cvc5 in any of the problems in the benchmark, we also crafted a few simple problems that exercise them. Table 1 presents the full set of proof rules and indicates a problem in the supplementary material using the rule (as well as other rules from CPC, a proof format designed to faithfully represent cvc5's internal reasoning) that is reconstructed via LEAN-SMT.[14]

Figure 2 presents an example of a problem in NTA and Figure 3 shows an excerpt of the proof produced by cvc5 in the CPC format that certifies its unsatisfiability. Note the application of the rule `TRANS_EXP_ZERO` in step `@p12`, which is used to conclude the refutation in step `@p14`. This is the key step in the proof, as this rule states that the only number whose exponential is equal to 1 is 0, whereas the rest of

[14] We could not find a problem for which cvc5 used `TRANS_SINE_SYMMETRY` since there is a rewriting rule for `sin` that seems to subsume it.

Proof rule	Problem
TRANS_EXP_APPROX_BELOW	E1.smt2
TRANS_EXP_APPROX_ABOVE_POS	E2.smt2
TRANS_EXP_APPROX_ABOVE_NEG	E3.smt2
TRANS_SINE_APPROX_BELOW_POS	E4.smt2
TRANS_SINE_APPROX_BELOW_NEG	E5.smt2
TRANS_SINE_APPROX_ABOVE_POS	E5.smt2
TRANS_SINE_APPROX_ABOVE_NEG	E6.smt2
TRANS_EXP_NEG	E7.smt2
TRANS_EXP_POSITIVITY	E8.smt2
TRANS_EXP_SUPERLIN	E9.smt2
TRANS_EXP_ZERO	E10.smt2
TRANS_PI	E11.smt2
TRANS_SINE_BOUNDS	E11.smt2
TRANS_SINE_SHIFT	E11.smt2
TRANS_SINE_SYMMETRY[1]	-
TRANS_SINE_TANGENT_PI	E12.smt2
TRANS_SINE_TANGENT_ZERO	E11.smt2
MULT_SIGN	E13.smt2
MULT_TANGENT	E14.smt2
MULT_POS	E13.smt2
MULT_NEG	E13.smt2
MULT_ABS_COMPARISON	E15.smt2

Figure 1. Problems validating each rule

```
(set-logic ALL)
(declare-const t Real)
(assert (not (= t 0)))
(assert (= (exp t) 1))
(check-sat)
```

Figure 2. Example of problem in NTA

```
(declare-const t Real)
(assume @p1 (not (= t (to_real 0))))
(assume @p2 (= (exp t) (to_real 1)))
(step @p3 (= (to_real 0) 0/1)
  :rule evaluate
  :args ((to_real 0)))
...
(step @p12 (= (= t 0/1) (= (exp t) 1/1))
  :rule arith_trans_exp_zero
  :args (t))
...
(step @p14 false
  :rule chain_resolution
  :premises (@p13 @p11 @p7)
  :args ((@list false true) (@list (= (exp t) 1/1)
    (= t 0/1))))
```

Figure 3. Example of proof using TRANS_EXP_ZERO

the steps [15] are merely applying basic logic rules to transform this fact into the negation of the problem. This proof is reconstructed by LEAN-SMT by replaying each step, and in particular step `@p12` is replayed by applying the theorem `arithTransExpZeroEq` in our formalization with type `∀ (t : ℝ), (t = 0) = (Real.exp t = 1)`.

There are currently two issues with the reconstruction that fall outside the scope of this work. The first one is that, in some proofs cvc5 can introduce steps with mixed integer-real arithmetic, i.e., expressions that mix integer and real values or variables. In this case, cvc5 should to add a casting step, turning all integer variables into real ones. Unfortunately, in some cases it currently fails to do so. When this happens, LEAN-SMT will receive a typing error while trying to reconstruct the expression, leaving a hole in the proof. Additionally, the incremental linearization solver in cvc5 currently performs a reasoning step that is not justified in the proof, which also leaves a hole in the proof assistant. Almost any non trivial proof produced by cvc5 contains an instance of the first issue, whereas the second issue happens in about half the problems. Therefore, both of them need to be addressed in the solver so that LEAN-SMT augmented with our formalization can be reliably used to reconstruct proofs found via incremental linearization. Nevertheless, this does not diminish the fact that we successfully validated the correspondence between the theorems stated in the formalization and the cvc5 proof calculus, as reconstruction of the corresponding steps succeeds for the examples in Figure 2. As future work, we plan to collaborate with the cvc5 team to resolve these issues and support larger problems.

6 Conclusion

We presented a formalization in the Lean theorem prover of the proof calculus used by cvc5 for incremental linearization in NRA and NTA. Our formalization ensures the soundness of the proof rules employed by cvc5 for reasoning about nonlinear arithmetic and transcendental functions. This effort not only increases the trustworthiness of the underlying method but also allows for the reconstruction of cvc5 proofs in Lean, which we have implemented as an extension of the LEAN-SMT plugin and evaluated on a series of benchmark. Overall we expect this work to enable higher reliability in safety-critical applications depending on nonlinear arithmetic and transcendental functions.

During the formalization process, we identified errors and missing hypotheses in the documentation of several cvc5 proof rules, which were shared with the cvc5 team. These findings highlight the importance of formal verification in identifying subtle issues that may otherwise go unnoticed. In the process, we also opened pull requests to `mathlib` to

[15] Most of them are not shown here. See Figure 4 in the appendix for the complete proof.

contribute generalizations and extensions of Taylor's theorem and convexity results, enriching its library for broader use.

Acknowledgments

The authors would like to thank the anonymous referees for their comments and helpful suggestions.

This work was partially supported by a gift from Amazon Web Services, the Stanford Center for Automated Reasoning, the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, and the Defense Advanced Research Projects Agency (DARPA) under contract FA8750-24-2-1001 (https://www.darpa.mil/research/programs/pipelined-reasoning-of-verifiers-enabling-robust-systems). Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of DARPA.

References

[1] Haniel Barbosa, Clark Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. 2022. cvc5: A Versatile and Industrial-Strength SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, Dana Fisman and Grigore Rosu (Eds.). Springer International Publishing, Cham, 415–442.

[2] Haniel Barbosa, Andrew Reynolds, Gereon Kremer, Hanna Lachnitt, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Arjun Viswanathan, Scott Viteri, Yoni Zohar, Cesare Tinelli, and Clark Barrett. 2022. Flexible Proof Production in an Industrial-Strength SMT Solver. In *Automated Reasoning*, Jasmin Blanchette, Laura Kovács, and Dirk Pattinson (Eds.). Springer International Publishing, Cham, 15–35.

[3] Alessandro Cimatti, Alberto Griggio, Ahmed Irfan, Marco Roveri, and Roberto Sebastiani. 2018. Incremental Linearization for Satisfiability and Verification Modulo Nonlinear Arithmetic and Transcendental Functions. *ACM Trans. Comput. Log.* 19, 3 (2018), 19:1–19:52. https://doi.org/10.1145/3230639

[4] Alessandro Cimatti, Alberto Griggio, Bastiaan Schaafsma, and Roberto Sebastiani. 2013. The MathSAT5 SMT Solver. In *Proceedings of TACAS (LNCS, Vol. 7795)*, Nir Piterman and Scott Smolka (Eds.). Springer.

[5] The Coq Development Team. 2021. *The Coq Proof Assistant Reference Manual - version 8.19.0.* Technical Report. INRIA. https://hal.science/hal-04523844

[6] James H. Davenport and Joos Heintz. 1988. Real quantifier elimination is doubly exponential. *Journal of Symbolic Computation* 5, 1 (1988), 29–35. https://doi.org/10.1016/S0747-7171(88)80004-X

[7] Leonardo de Moura and Sebastian Ullrich. 2021. The Lean 4 Theorem Prover and Programming Language *(Lecture Notes in Computer Science, Vol. 12699)*, André Platzer and Geoff Sutcliffe (Eds.). Springer, 625–635. https://doi.org/10.1007/978-3-030-79876-5_37

[8] Ahmed Irfan, Alessandro Cimatti, Alberto Griggio, Marco Roveri, and Roberto Sebastiani. 2019. Lemmas for Satisfiability Modulo Transcendental Functions via Incremental Linearization. In *Proceedings of the 4th SC-Square Workshop co-located with the SIAM Conference on Applied Algebraic Geometry, SC-square@SIAM AG 2019, Bern, Switzerland, 10th July 2019 (CEUR Workshop Proceedings, Vol. 2460)*, John Abbott and Alberto Griggio (Eds.). CEUR-WS.org. http://ceur-ws.org/Vol-2460/paper8.pdf

[9] Gereon Kremer, Andrew Reynolds, Clark W. Barrett, and Cesare Tinelli. 2022. Cooperating Techniques for Solving Nonlinear Real Arithmetic in the cvc5 SMT Solver (System Description). In *International Joint Conference on Automated Reasoning (IJCAR) (Lecture Notes in Computer Science, Vol. 13385)*, Jasmin Blanchette, Laura Kovács, and Dirk Pattinson (Eds.). Springer, 95–105. https://doi.org/10.1007/978-3-031-10769-6_7

[10] Assia Mahboubi and Cyril Cohen. 2012. Formal proofs in real algebraic geometry: from ordered fields to quantifier elimination. *Logical Methods in Computer Science* Volume 8, Issue 1 (Feb. 2012). https://doi.org/10.2168/lmcs-8(1:2)2012

[11] The mathlib Community. 2020. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, Jasmin Blanchette and Catalin Hritcu (Eds.). ACM, 367–381. https://doi.org/10.1145/3372885.3373824

[12] Abdalrhman Mohamed, Tomaz Mascarenhas, Harun Khan, Haniel Barbosa, Andrew Reynolds, Yicheng Qian, Cesare Tinelli, and Clark Barrett. 2025. LEAN-SMT: An SMT tactic for discharging proof goals in Lean. In *Computer Aided Verification - 37th International Conference, CAV 2025, Zagreb, Croatia, July 21-25, 2025, Proceedings (Lecture Notes in Computer Science)*, Ruzica Piskac and Zvonimir Rakamaric (Eds.). Springer.

[13] Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. 2002. *Isabelle/HOL: a proof assistant for higher-order logic.* Springer-Verlag, Berlin, Heidelberg.

[14] Daniel Richardson. 1968. Some Undecidable Problems Involving Elementary Functions of a Real Variable. *J. Symb. Log.* 33, 4 (1968), 514–520. https://doi.org/10.2307/2271358

[15] Matias Scharager, Katherine Cordwell, Stefan Mitsch, and André Platzer. 2021. *Verified Quadratic Virtual Substitution for Real Arithmetic.* Springer International Publishing, 200–217. https://doi.org/10.1007/978-3-030-90870-6_11

[16] Alfred Tarski. 1998. A Decision Method for Elementary Algebra and Geometry. In *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Bob F. Caviness and Jeremy R. Johnson (Eds.). Springer Vienna, Vienna, 24–84.

[17] V. Weispfenning. 1997. Quantifier Elimination for Real Algebra — the Quadratic Case and Beyond. *Applicable Algebra in Engineering, Communication and Computing* 8, 2 (1997), 85–101. https://doi.org/10.1007/s002000050055

A Appendix

We present our formalization of the remaining three proof rules we omitted in Section 4.

A.1 TRANS_EXP_APPROX_ABOVE_NEG

Let p be the Taylor polynomial of exp centered at zero with even degree d. This rule bounds $\exp(t)$ from above by the secant between $(l, p(l))$ and $(u, p(u))$ for $l \le t \le u$ and t negative.

$$\frac{- \mid d, t, l, u}{(t \ge l \wedge t \le u) \rightarrow \exp(t) \le \mathsf{secant}(\exp, l, u, t)}$$

Our theorem states the following.

```
theorem arithTransExpApproxAboveNeg (d k : Nat) (hd
    : d = 2*k) (l u t : ℝ)
                                  (ht : l ≤ t ∧ t ≤
    u) (hu : u < 0) :
  let p: ℝ → ℝ := taylorWithinEval Real.exp d
    Set.univ 0
  Real.exp t ≤ ((p l - p u) / (l - u)) * (t - l) + p
    l
```

Recall by `le_convex_of_le`, it suffices to show $\exp(t) \leq p(t)$ for $t < 0$. We rewrite the difference $\exp(t) - p(t)$ as the remainder term using `taylor_mean_remainder_lagrange'`. Hence we have to show that $\exp^{(d+1)}(t_1) * t^{d+1}/(d+1)!$ is negative for some $t_1 \in (t, 0)$. This is true because the derivative of exp is itself, exp is always positive, and $d+1$ is odd so t^{d+1} is negative. Then we are done by simply applying `le_convex_of_le` as we now satisfy the hypothesis.

A.2 TRANS_SINE_APPROX_ABOVE_NEG

Let p be the Taylor polynomial of sin centered at zero with odd degree d. This rule bounds sin from above by the secant between $(l, p(l))$ and $(u, p(u))$ for $-\pi \leq l \leq t \leq u < 0$ analogous to TRANS_EXP_APPROX_ABOVE_NEG. It states

$$\frac{- \mid d, t, lb, ub, l, u}{(t \geq lb \wedge t \leq ub) \rightarrow \sin(t) \leq \mathsf{secant}(\sin, l, u, t)}$$

where lb, ub are symbolic lower and upper bounds on t and l, u are evaluated lower and upper bounds on t. We formalize this rule as the following theorem where we stipulate $-\pi \leq l \leq t \leq u < 0$ since the linearization procedure stays within the same convexity region of sin.

```
theorem arithTransSineApproxAboveNeg (d k : Nat)
  (hd : d = 2*k + 1)
  (ht : l ≤ t ∧ t ≤ u) (hu : u < 0) (hl : -π ≤
  l) :
  let p: ℝ → ℝ :=
    fun x => taylorWithinEval Real.sin d Set.univ
  0 t + (t ^ (d + 1)) / (d + 1).factorial
  Real.sin t ≤ ((p l - p u) / (l - u)) * (t - l) +
  p l
```

The proof is very similar to TRANS_EXP_APPROX_ABOVE_NEG. Applying `le_convex_of_le` it suffices to show $\sin t \leq p(t)$ for $-\pi \leq t < 0$. Let $p^*(t)$ be the Taylor Polynomial of degree d. We rewrite the difference $\sin t - p^*(t)$ as the remainder term using `taylor_mean_remainder_lagrange'` leaving the error term as is. Then we have to show that $\sin^{(d+1)}(t_1) * t^{d+1}/(d+1)! - t^{d+1}/(d+1)!$ is nonpositive for some $t_1 \in (t, 0)$. This is true because the derivatives of sin is always between -1 and 1 and t^{d+1} is positive.

A.3 TRANS_SINE_APPROX_ABOVE_POS

The strategy is similar to TRANS_SINE_APPROX_BELOW_NEG. The rule bounds sin from above on the non-negative reals using its Taylor polynomial centered at zero,

$$\frac{- \mid d, t, c, lb, ub}{(t \geq lb \wedge t \leq ub) \rightarrow \sin(t) \leq \mathsf{upper}(\sin, c)}$$

The rule's documentation and code were unclear, particularly the definition of $\mathsf{upper}(\sin, c)$ as discussed earlier. After refining multiple candidate statements, we arrived at the following definition: $\mathsf{upper}(\sin, c)$ is the Taylor polynomial of sin evaluated at c plus $c^{d+1}/(d+1)!$ where $c = \mathrm{argmin}_{x \in [lb, ub]} \sin x$. We proved the following theorem first before reasoning about c. All in all, the theorem is the correct encapsulation since we can successfully reconstruct some examples using it.

```
theorem arithTransSineApproxAbovePos_self
  (d k : Nat) (hd : d = 2*k + 1) (hx : 0 ≤ x) :
  sin x ≤ taylorWithinEval Real.sin d Set.univ 0 x +
    x ^ (d + 1)/(d + 1).factorial
```

Since we already proved an analogous theorem, we do not have to duplicate code. Instead, we use a slick strategy using the odd properties of sin, its Taylor polynomial, and `arithTransSineApproxBelowNeg_self`. We first prove the following theorem:

```
theorem taylorSin_neg (x : Real):
  let p: ℝ → ℝ := taylorWithinEval Real.sin d
    Set.univ 0
  p (-x) = -p x
```

The proof involves pushing the negation into each coefficient of the Taylor polynomial and using the fact that the even terms are zero, which we formalized separately. For the main proof, we simply replace x with $-x$ and use the odd properties of sin and its Taylor polynomial. Then we obtain `arithTransSineApproxBelowNeg_self` for $t := -x$ which satisfies the assumption $t \leq 0$. The proof is merely two lines using this strategy which is a powerful example of how our strategy of using odd properties generalized to this case yielding less code duplication and more maintainable code in the long run.

Finally, we prove the main theorem that encapsulates the rule.

```
theorem arithTransSineApproxBelowNeg
  (d k : ℕ) (hd : d = 2*k + 1)
  (hl : 0 ≤ lb) (hu : ub ≤ π)
  (hx1 : lb ≤ x) (hx2 : x ≤ ub)
  (hc : c = if ub < π/2 then ub
            else if lb < π/2 then π/2
            else lb) :
  sin x ≤ taylorWithinEval Real.sin d Set.univ 0 c +
    c ^ (d + 1) / (d + 1).factorial
```

We have $0 \leq lb$ and $ub \leq \pi$ since the sine function is periodic as we showed in Subsection 3.4 and the solver uses the periodicity to restrict us to either $[-\pi, 0]$ or $[0, \pi]$. The proof follows by applying the previous theorem to the right-hand side yielding $\sin x \leq \sin c$. We case-split on c, and use a theorem from `mathlib` that sin is monotonically increasing in $[-\pi/2, \pi/2]$, $\sin \pi/2 = 1$ and the `linarith` tactic.

B Full Version of Proof in Figure 3

Received 2025-09-13; accepted 2025-11-13

```
(declare-const t Real)
(assume @p1 (not (= t (to_real 0))))
(assume @p2 (= (exp t) (to_real 1)))
(step @p3 (= (to_real 0) 0/1)
  :rule evaluate
  :args ((to_real 0)))
(step @p4 (= t t)
  :rule refl
  :args (t))
(step @p5 (= (= t (to_real 0)) (= t 0/1))
  :rule cong
  :premises (@p4 @p3)
  :args ((= t (to_real 0))))
(step @p6 (= (not (= t (to_real 0))) (not (= t 0/1)))
  :rule cong
  :premises (@p5)
  :args ((not (= t (to_real 0)))))
(step @p7 (not (= t 0/1))
  :rule eq_resolve
  :premises (@p1 @p6))
(step @p8 (= (to_real 1) 1/1)
  :rule evaluate
  :args ((to_real 1)))
(step @p9 (= (exp t) (exp t))
  :rule refl
  :args ((exp t)))
(step @p10
  (= (= (exp t) (to_real 1)) (= (exp t) 1/1))
  :rule cong
  :premises (@p9 @p8)
  :args ((= (exp t) (to_real 1))))
(step @p11 (= (exp t) 1/1)
  :rule eq_resolve
  :premises (@p2 @p10))
(step @p12 (= (= t 0/1) (= (exp t) 1/1))
  :rule arith_trans_exp_zero
  :args (t))
(step @p13 (or (= t 0/1) (not (= (exp t) 1/1)))
  :rule equiv_elim2
  :premises (@p12))
(step @p14 false
  :rule chain_resolution
  :premises (@p13 @p11 @p7)
  :args ((@list false true) (@list (= (exp t) 1/1)
    (= t 0/1))))
```

Figure 4. Full Version of Proof in Figure 3