# Real Arithmetic in TLAPM [⋆]

Ovini V.W. Gunasekera, Andrew Sogokon,
Antonios Gouglidis, and Neeraj Suri

School of Computing and Communications, Lancaster University
{o.gunasekera|a.sogokon|a.gouglidis|neeraj.suri}@lancaster.ac.uk

**Abstract.** TLA$^+$ is a formal specification language for modelling systems and programs. While TLA$^+$ allows writing specifications involving real numbers, its existing tool support does not currently extend to automating real arithmetic proofs. This functionality is crucial for proving properties of hybrid systems, which may exhibit both continuous and discrete behaviours. In this paper, we address this limitation by enabling support for deciding first-order real arithmetic formulas (involving only polynomials). Specifically, we update the TLA$^+$ Proof System (TLAPS) to support reals and basic real arithmetic operations and implement them in the TLA$^+$ Proof Manager. The latter generates assertions in SMT-LIB and directs them to a selected backend (currently the Z3 SMT solver, which supports the theory of nonlinear real arithmetic). We motivate this functionality with problems arising in safety verification.

**Keywords:** formal verification · real arithmetic · hybrid systems · TLA$^+$ .

## 1 Introduction

TLA$^+$ is a general-purpose formal language based on the Zermelo-Fraenkel set theory for specifying digital systems and is supported by industrial strength tools. Over the years, TLA$^+$ gained considerable attention from both the academic community and industry [8], where it was used by major companies such as Amazon, Intel and Microsoft in applications ranging from concurrent to distributed systems. Indeed, TLA$^+$ is so expressive that it can even be applied to model *hybrid systems* [6] (and therefore *cyber-physical systems* in which the state may evolve in discrete time steps or *continuously*). Modelling and reasoning about continuous state evolution in these systems fundamentally requires real numbers. TLA$^+$ allows one to work with variables ranging over the set of real numbers (see [7, §18.4]) and was designed anticipating the use of decision procedures that could work with the structures defined in its standard arithmetic modules [7, Ch. 18][1]. Work by Merz et al. [9] created the necessary infrastructure to handle arithmetic problems involving integers and created an interface to

[⋆] Research supported by the UKRI Trustworthy Autonomous Systems (TAS) Node in Security. EPSRC Grant EP/V026763/1.

[1] *"If you want to prove something about a specification, you can reason about numbers however you want. Tools like model checkers and theorem provers that care about these operators will have their own ways of handling them." – L. Lamport [7, §18.4]*

SMT solvers. However, support for real arithmetic has, up to now, been notably absent, which represents a fundamental limitation that must be addressed before hybrid system verification in TLA$^+$ can become a practical endeavour.

*Contributions.* We extend TLAPM (the TLA$^+$ Proof Manager) to support real arithmetic, enabling automatic proofs using the Z3 SMT solver. In this work, we are concerned solely with decidable first-order real arithmetic conjectures.

*Related Work.* Our work is very close in spirit to that of Denman and Muñoz [2], who enabled automatic handling of real arithmetic conjectures in PVS [12] using an external oracle (MetiTarski [13]). Our work builds on an earlier effort by Merz et al. [11], who developed the TLA$^+$ Proof System (TLAPS), i.e., a proof manager for TLA$^+$ (TLAPM) and the infrastructure necessary for interfacing with SMT solvers (and who indeed commented that support for real numbers should be facilitated, which we carry through in this work). There exist a number of purpose-built theorem proving systems for reasoning about cyber-physical systems, such as KeYmaera X [3] and the HHL Prover [14], which likewise treat real arithmetic backends (e.g., Mathematica) as trusted external oracles.

## 2 Enabling Real Arithmetic in the TLA$^+$ Proof Manager

The TLA$^+$ Proof System (TLAPS) includes a proof manager which interfaces with backend verifiers. This enables the proof system to perform deductive verification of safety properties of TLA$^+$ specifications. The proof manager interprets the proofs, generates a set of proof obligations and directs them to a solver (trusted external oracle) such as Z3, Yices, etc. *Fig. 1* provides a partial representation of the TLAPS architecture in [1, 9, §1], with an emphasis on the components we updated in this work. The basic operations including pre-processing, boolification, post-processing, normalisation and optimisation performed by the proof manager are described in detail in [9–11]. The components

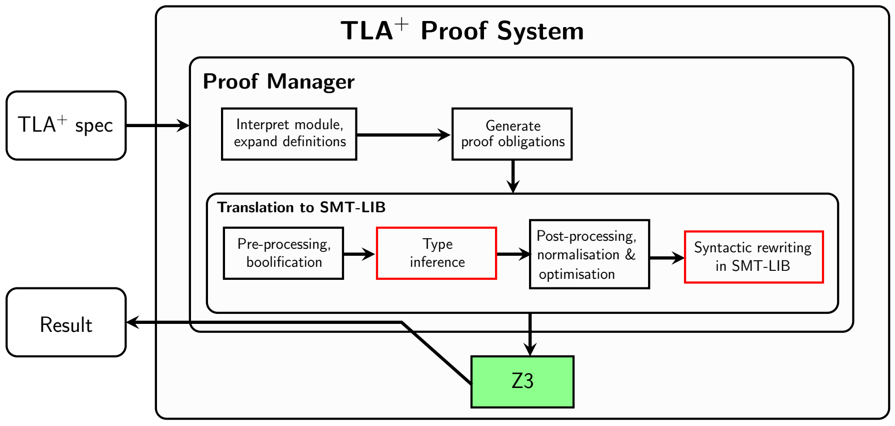

TLA+ Proof System
Proof Manager
TLA+ spec
Interpret module, expand definitions
Generate proof obligations
Translation to SMT-LIB
Pre-processing, boolification
Type inference
Post-processing, normalisation & optimisation
Syntactic rewriting in SMT-LIB
Result
Z3

Fig. 1: TLAPS architecture with the Z3 backend

with a red outline in *Fig. 1* depict the changes to the *SMT translation* process that we made in order to support the proving of real arithmetic conjectures using Z3. TLAPS features a *type inference* algorithm that assigns types to the untyped TLA$^+$ variables and values; this is updated to interpret reals (see *Sect. 2.1*). During the *syntactic rewriting* stage, TLA$^+$ expressions are encoded in the SMT-LIB language, at which point they are ready to be passed to a backend verifier. In the latter stage, an alternative *untyped encoding* process can be employed to instead delegate type inference to the SMT-solvers. This process requires new lifting axioms to assert that TLA$^+$ arithmetic coincides with the SMT arithmetic over reals (described in *Sect. 2.2*). The implementation of the extended TLAPM supporting real arithmetic can be found in [5].

## 2.1 Typed encodings

TLA$^+$ is an untyped language. Hence, type encoding is required to assign types to the untyped variables and values. Assigning types enforces restrictions on admissible formulas that can be directed to the backend verifiers. This operation is performed by a type inference algorithm consisting of a constraint generation and solving phase. The result of the constraint generation phase is a set of constraints based on the type environment, a TLA$^+$ expression and a type variable. The constraint generation rules are derived from the corresponding typing rules, and the constraint solving phase solves the equality and subtyping constraints and proves some residual subtype checking constraints [10].

To enable the interpretation of TLA$^+$ expressions containing reals, we extend the type system and type inference algorithm developed by Merz et al. in [10] by introducing a new type *Real*, i.e., $\tau ::= Real$ to the existing grammar that describes the supported types and introducing a set of additional typing rules. In TLA$^+$ , the *Reals* module extends the *Integers* module and defines the set *Real* of real numbers along with the standard arithmetic operations, including the ordinary division operator (/) [7, §18.4]. In our extension we employ a decimal representation of real constants, which is consonant with the representation in TLA$^+$ ($c_1 \dots c_m.d_1 \dots d_n \triangleq c_1 \dots c_m d_1 \dots d_n / 10^n$ [7, §16.1.11]). We further define rules for real variables and expressions involving the usual real arithmetic operations (some shown in Table 1 following notational conventions in [10, §3]), which enables the typing of real arithmetic formulas which can then be passed on to backend verifiers as real arithmetic conjectures.

$$\frac{\Gamma \vdash e_i : a_i \quad \Gamma \vdash a_i \prec: \text{Real} \quad i \in \{1,2\}}{\Gamma \vdash e_1 + e_2 : \{x : \text{Real} \mid x = e_1 + e_2\}}\text{[T-PLUS-REAL]} \qquad \frac{\Gamma \vdash e_i : a_i \quad \Gamma \vdash a_i \prec: \text{Real} \quad i \in \{1,2\}}{\Gamma \vdash e_1 < e_2 : \text{Bool}}\text{[T-LESS-REAL]}$$

Table 1: Examples of added typing rules for Reals

### 2.2 Untyped encodings

Untyped encodings for $\mathsf{TLA}^+$ formulas are an alternative encoding method implemented in $\mathsf{TLAPS}$, where type inference is delegated to the SMT solver. The reader may find a helpful discussion of some of the advantages afforded by using this kind of encoding in [9,10], along with some of the disadvantages which may carry a performance penalty.

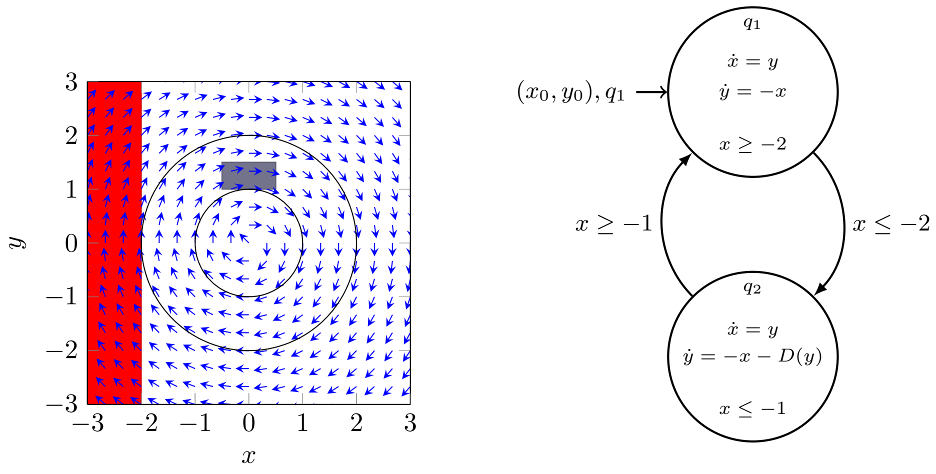
With untyped encoding, a single SMT sort $\mathsf{U}$ is used to represent $\mathsf{TLA}^+$ values and its operators are represented as uninterpreted functions having sort $\mathsf{U}$ as their arguments [9]. In order to encode real arithmetic formulas, we declare uninterpreted functions that embed SMT reals into the sort $\mathsf{U}$ representing $\mathsf{TLA}^+$ values, i.e., $real2u : \mathsf{Real} \rightarrow \mathsf{U}$ and $u2real : \mathsf{U} \rightarrow \mathsf{Real}$. $\mathsf{Real}$ represents SMT reals, $real2u$ embeds SMT reals into a sort $\mathsf{U}$ representing $\mathsf{TLA}^+$ values and $u2real$ performs the reverse. This is supported by the axiom $\forall m \in \mathsf{Real} : u2real(real2u(m)) = m$ to ensure the consistency and soundness of translating SMT reals into the sort $\mathsf{U}$. Real arithmetic operations over $\mathsf{TLA}^+$ values are homomorphically defined over the image of $real2u$ using axioms. In the axiom $\forall m, n \in \mathsf{Real} : plus(real2u(m), real2u(n)) = real2u(m + n)$, the $+$ operation on the right-hand side denotes the built-in addition operation over SMT reals, while $plus$ has function type $\mathsf{U} \times \mathsf{U} \rightarrow \mathsf{U}$. Similar axioms are defined for other real arithmetic operations and utilise the existing uninterpreted functions for operators that are introduced by Merz et al. in [9,10].

*Remark 1.* At present, Z3 is the default real arithmetic backend; however, other tools exist which likewise accept SMT-LIB input and could serve as alternatives. It must be noted that some of these tools (e.g., MetiTarski [13]) can only work with SMT-LIB inputs of a particular form and would not be able to handle problems in the untyped SMT encoding as described above.

## 3 Safety Verification of Cyber-Physical Systems

As an illustrative example of safety verification that involves real-valued variables, we will use a model of an oscillator. The motion of a simple harmonic oscillator, such as a mass $m$ suspended from a spring (with spring constant $k$) can be described by a second-order differential equation $\ddot{x} + \omega^2 x = 0$, where $\omega = \sqrt{\frac{k}{m}}$ is the frequency of oscillation, the state variable $x$ measures the displacement of the mass from the point of equilibrium and $\ddot{x}$ represents the second derivative of $x$ with respect to time, i.e., the acceleration $\frac{d^2x}{dt^2}$. The dynamics of this system can be written down as a system of linear differential equations $\dot{x} = y,\ \dot{y} = -\omega^2 x$. For simplicity, let $m = 1$ and $k = 1$, so that the system becomes $\dot{x} = y,\ \dot{y} = -x$. The system can be geometrically represented as a *vector field* $\langle y, -x \rangle$ defined on the real plane (as shown in Fig. 2a). To reduce the amplitude of oscillations, a *damping term* $D(y)$ can be introduced into the system to yield a damped oscillator $\dot{x} = y,\ \dot{y} = -x - D(y)$ in which oscillations die down over time.

Let us consider a hybrid automaton (illustrated in Fig. 2b) with two modes evolving according the undamped (mode $q_1$) and damped oscillator dynamics (mode $q_2$). In this model, the system switches on damping when the displacement $x$ falls below $-2$ (this could be done e.g., to prevent damage to the spring). Let us suppose that the initial displacement $x_0$ of the oscillator is only known



(a) Vector field (mode $q_1$) (b) Hybrid automaton

Fig. 2: Hybrid system model of an oscillator

to be within the bounds $-\frac{1}{2} \leq x_0 \leq \frac{1}{2}$ and the initial velocity $y_0$ is in the range $1 \leq y_0 \leq \frac{3}{2}$. From this set of initial conditions we wish to prove that damping need never be applied (i.e., the hybrid automaton in Fig. 2b never transitions into mode $q_2$). There are a number of ways in which one can prove the safety specification described above. A common approach (which does *not* involve computing solutions to the differential equation) is to exhibit an appropriate *inductive invariant*, i.e., a set of states $I \subseteq \mathbb{R}^2$ such that all trajectories starting inside the invariant remain within the invariant. A standard proof of safety using an inductive invariant involves showing three things: (1.) that the proposed invariant is indeed inductive (i.e., that the system cannot transition outside the invariant), (2.) that all possible initial states of the system lie within the invariant, and (3.) that the invariant contains no *unsafe* states that are deemed undesirable. In order to prove our property of interest, we can employ an inductive invariant given by formula $x^2+y^2 < 4 \wedge x^2+y^2 \geq 1$, which corresponds an annular region illustrated in Fig. 2a, where it is seen to include all the initial states (represented by the grey box) and none of the unsafe states (shown in red) from which the system may transition into mode $q_2$ where damping is applied.

Several methods exist to check whether a proposed invariant is inductive (i.e., to solve (1.)); while these fall outside the scope of the present paper, these methods essentially reduce the problem to one of real arithmetic. In this example, the fact that $x^2+y^2$ is a conserved quantity can be established by checking that its time derivative $2x\dot{x}+2y\dot{y}$ is everywhere zero (i.e., $2xy-2yx=0$). For solving (2.) and (3.), the first-order theory of real arithmetic provides us with a formal language that is expressive enough to state properties such as inclusion or non-intersection of sets, provided that these are described using formulas that only involve *polynomials*. Establishing the inclusion of the initial states within the invariant and its non-intersection with the unsafe states in this example reduces to proving the following sentences:

$$\forall\ x,y \in \mathbb{R}.\ \underbrace{-0.5 \leq x \wedge x \leq 0.5 \wedge 1 \leq y \wedge y \leq 1.5}_{\text{Initial states}} \rightarrow \underbrace{x^2+y^2 < 4 \wedge x^2+y^2 \geq 1}_{\text{Invariant}},$$

$$\forall\ x,y \in \mathbb{R}.\ \neg \left( \underbrace{x \leq -2}_{\text{Unsafe}} \wedge \underbrace{x^2+y^2 < 4 \wedge x^2+y^2 \geq 1}_{\text{Invariant}} \right).$$

Figure 3 shows how these conjectures are represented in the TLA$^+$ syntax and solved using our implementation with Z3 as a real arithmetic backend.

```
EXTENDS TLAPS, Reals

(*
 * All possible initial states of the systems lie within the
 * proposed invariant Init => Inv. The conjecture is proven true.
 *)
THEOREM \A x,y \in Real: ((-0.5 <= x /\ x <= 0.5 /\ 1.0 <= y /\ y <= 1.5)
                          => ((x*x)+(y*y) < 4.0 /\ (x*x)+(y*y) >= 1.0))
                             BY Z3

(*
 * The invariant does not contain unsafe states.
 * ¬( Unsafe /\ Inv) The conjecture is proven true.
 *)
THEOREM \A x,y \in Real: ~((x <= - 2.0) /\ ((x*x) + (y*y) < 4.0  /\
                                           (x*x) + (y*y) >= 1.0)) BY Z3
```

Fig. 3: Hybrid system safety verification: real arithmetic conjectures (2.) and (3.)

Unlike hybrid automata, *discrete-time* dynamical systems can be modelled in TLA$^+$ in a straightforward way. Let us consider the following discrete-time system in which the state variables $x$ and $y$ take values in the real numbers:

$$x(t+1) = \frac{2x(t)}{3} + \frac{y(t)}{2},$$
$$y(t+1) = \frac{x(t)}{2} - \frac{y(t)}{3}.$$

Let us suppose that the system may be initialised from any state satisfying the formula $x^2 + y^2 \leq 1$ (i.e., $x(0)^2 + y(0)^2 \leq 1$) and that we wish to show that the absolute value of $x$ can never exceed 1 as the system evolves. To prove this, let us take $x^2 + y^2 \leq 1$ as our candidate inductive invariant. The inclusion of all initial states within the invariant is trivial in this case, so it remains to show that (1.) no unsafe state satisfies $x^2+y^2 \leq 1$, and (2.) $x^2+y^2 \leq 1$ is an inductive invariant. To show (1.) , one needs to prove that the invariant implies the safety of the system, i.e., that for all $x, y \in \mathbb{R}$ one has $x^2 + y^2 \leq 1 \rightarrow x \leq 1 \wedge x \geq -1$. To prove (2.) one needs to show that the set of states satisfying $x^2 + y^2 \leq 1$ is closed under the dynamics of the system. In TLA$^+$ one may model the dynamics of such a system as follows:

```
x' = (2.0/3.0)*x + 0.5*y
y' = 0.5*x - (1.0/3.0)*y
```

where the primed symbols `x'` and `y'` stand for the value of the variables `x` and `y` in the next state. Showing that the invariant is inductive ultimately reduces to proving that the following implication holds for all $x, y, x', y' \in \mathbb{R}$:

$$\left(x^2 + y^2 \leq 1 \wedge x' = \frac{2}{3}x + \frac{1}{2}y \wedge y' = \frac{1}{2}x - \frac{1}{3}y\right) \rightarrow (x')^2 + (y')^2 \leq 1\,.$$

Proving safety specifications for discrete-time systems such as the one above in TLA$^+$ can now be done conveniently with the help of the extended proof manager.

## 4 Conclusion

We have developed support in the TLA$^+$ Proof Manager for handling first-order real arithmetic sentences. Real arithmetic problems arise naturally in the verification of hybrid and cyber-physical systems and our work represents a step towards facilitating their formal verification using TLA$^+$. Currently, our implementation employs only Z3 as a real arithmetic backend; however, we note the potential for supporting additional backends in the future. In particular, tools such as MetiTarski [13] and dReal [4] can – in addition to serving as alternative real arithmetic backends – enable reasoning about *special functions* (such as $\sin, \cos, \ln, e$, etc., the presence of which makes real arithmetic *undecidable*). The TLA$^+$ Proof Manager currently does not offer support for working with these kinds of functions and further extensions to the system could be pursued to enable this functionality.

## References

1. Chaudhuri, K., Cousineau, D., Doligez, D., Lamport, L., Libal, T., Merz, S., Tristan, J.B., Vanzetto, H.: GitHub: The TLA$^+$ Proof Manager. https://github.com/tlaplus/tlapm, [Online; accessed December 2023]

2. Denman, W., Muñoz, C.A.: Automated real proving in PVS via MetiTarski. In: FM 2014. LNCS, vol. 8442, pp. 194–199. Springer (2014). https://doi.org/10.1007/978-3-319-06410-9_14
3. Fulton, N., Mitsch, S., Quesel, J., Völp, M., Platzer, A.: KeYmaera X: an axiomatic tactical theorem prover for hybrid systems. In: CADE 2015. LNCS, vol. 9195, pp. 527–538. Springer (2015). https://doi.org/10.1007/978-3-319-21401-6_36
4. Gao, S., Kong, S., Clarke, E.M.: dreal: An SMT solver for nonlinear theories over the reals. In: CADE 2013. LNCS, vol. 7898, pp. 208–214. Springer (2013). https://doi.org/10.1007/978-3-642-38574-2_14
5. Gunasekera, O.V.W.: GitHub: TLA$^+$ proof system with real arithmetic support. https://github.com/Ovini99/TLAPS_Real, [Online; accessed December 2023]
6. Lamport, L.: Hybrid systems in TLA$^+$. In: Hybrid Systems. LNCS, vol. 736, pp. 77–102. Springer (1992). https://doi.org/10.1007/3-540-57318-6_25
7. Lamport, L.: Specifying Systems: The TLA$^+$ Language and Tools for Hardware and Software Engineers. Addison-Wesley (June 2002)
8. Lamport, L.: Industrial Use of TLA$^+$. https://lamport.azurewebsites.net/tla/industrial-use.html (2019), [Online; accessed March 2023]
9. Merz, S., Vanzetto, H.: Harnessing SMT solvers for TLA$^+$ proofs. Electron. Commun. EASST **53** (2012). https://doi.org/10.14279/TUJ.ECEASST.53.766
10. Merz, S., Vanzetto, H.: Refinement types for TLA$^+$. In: NFM 2014. LNCS, vol. 8430, pp. 143–157. Springer (2014). https://doi.org/10.1007/978-3-319-06200-6_11
11. Merz, S., Vanzetto, H.: Encoding TLA$^+$ into many-sorted first-order logic. In: ABZ 2016. LNCS, vol. 9675, pp. 54–69. Springer (2016). https://doi.org/10.1007/978-3-319-33600-8_3
12. Owre, S., Rushby, J.M., Shankar, N.: PVS: A prototype verification system. In: CADE 1992, Proceedings. LNCS, vol. 607, pp. 748–752. Springer (1992). https://doi.org/10.1007/3-540-55602-8_217
13. Paulson, L.C.: MetiTarski: Past and future. In: ITP 2012. LNCS, vol. 7406, pp. 1–10. Springer (2012). https://doi.org/10.1007/978-3-642-32347-8_1
14. Wang, S., Zhan, N., Zou, L.: An improved HHL prover: An interactive theorem prover for hybrid systems. In: ICFEM 2015. LNCS, vol. 9407, pp. 382–399. Springer (2015). https://doi.org/10.1007/978-3-319-25423-4_25