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 typed and untyped encodings in 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 present this new functionality with a safety verification example featuring a hybrid system.

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

1 Introduction

TLA$^+$ is a general-purpose formal language based on the Zermelo-Fränkel 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 [4], 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* [2] (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 [3, §18.4]) and was designed anticipating the use of decision procedures that could work with the structures defined in its standard arithmetic modules [3, Ch. 18][1]. Work by Merz et al. [5] created the necessary infrastruc-

* 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 [3, §18.4]*

ture to handle arithmetic problems involving integers and created an interface to 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 real arithmetic conjectures that are sentences in the first-order theory of real closed fields.

Related Work. Our work is very close in spirit to that of Denman and Muñoz [1], who enabled automatic handling of real arithmetic conjectures in PVS using an external oracle (MetiTarski). Our work builds on an earlier effort by Merz et al. [7], 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 indeed commented that support for real numbers should be facilitated, which we carry through in this work).

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. The architecture of TLAPS is elaborated in [5, Sect. 1] and the basic operations performed by the proof manager are described in detail in [5–7]. In order to support proving real arithmetic conjectures using Z3, certain changes to the *SMT translation* process are required. TLAPS features a type inference algorithm which assigns types to the untyped TLA$^+$ expressions, which we adapt to interpret reals (see *Sect. 2.1*). During the syntactic rewriting stage, TLA$^+$ expressions are encoded in the SMT-LIB language. In the latter stage, an untyped encoding process can alternatively delegate type inference to the SMT-solvers. This process requires new lifting axioms to assert that TLA$^+$ arithmetic coincides with SMT arithmetic over reals (which we adapt to include interpretation for reals; see *Sect. 2.2*).

2.1 Typed encodings

TLA$^+$ is an untyped language. Hence, type encoding is required to assign types to the untyped expressions. 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 [6].

To enable the interpretation of TLA$^+$ expressions containing reals, we extend the type system and type inference algorithm developed by Merz et al. in [6] by introducing a new type *Real*, i.e., $\tau ::= Real$ to the existing grammar that describes the supported types and introducing a set of typing rules. The *Reals* module in TLA$^+$ extends *Integers* and defines the set *Real* of real numbers and the standard arithmetic operations, including the ordinary division operator ($/$) [3, §18.4]. As depicted in Table 1, we define an axiom for the set of reals [T-REAL]. The "." in [T-NUM-REAL] represents the decimal point where literal reals are represented as decimal numbers, which is consonant with the representation of decimals in TLA$^+$: $c_1 \ldots c_m.d_1 \ldots d_n \triangleq c_1 \ldots c_m d_1 \ldots d_n / 10^n$ [3, §16.1.11]. We further define rules for the usual real arithmetic operations. The environment of the arguments governs the arithmetic operation typing rule to be used during constraint generation. For example, in the presence of a + operator, if the arguments contain integers or reals in the form of either a number or a variable, the typing rule of [T-PLUS] [6] or [T-PLUS-REAL] is selected, respectively.

$$\text{[T-REAL]}\quad \frac{}{\Gamma \vdash Real : \text{Set Real}} \qquad \text{[T-NUM-REAL]}\quad \frac{m \in \{0,1,2,3,4,...\} \quad n \in \{0,1,2,3,4,...\}}{\Gamma \vdash m.n : \{x : \text{Real} \mid x = m.n\}}$$

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

Table 1: Typing rules for Reals

In Table 1, Γ is a typing context. If ϕ is a formula and τ is a type, $\Gamma \vdash \phi : \tau$ is a pre-judgement that becomes a valid judgement if it can be derived from the typing rules. $\prec:$ is a non-unifiable subtype relation, i.e., it checks for subtyping without unifying type variables. These type variables are unified during the constraint solving phase, which solves such subtyping constraints. $\tau_1 \prec: \tau_2$ is valid iff both types are ground types and $\tau_1 <: \tau_2$ where $<:$ is used for subtyping and unifies type variables; the unification of type variables occurs during the constraint solving phase.

2.2 Untyped encodings

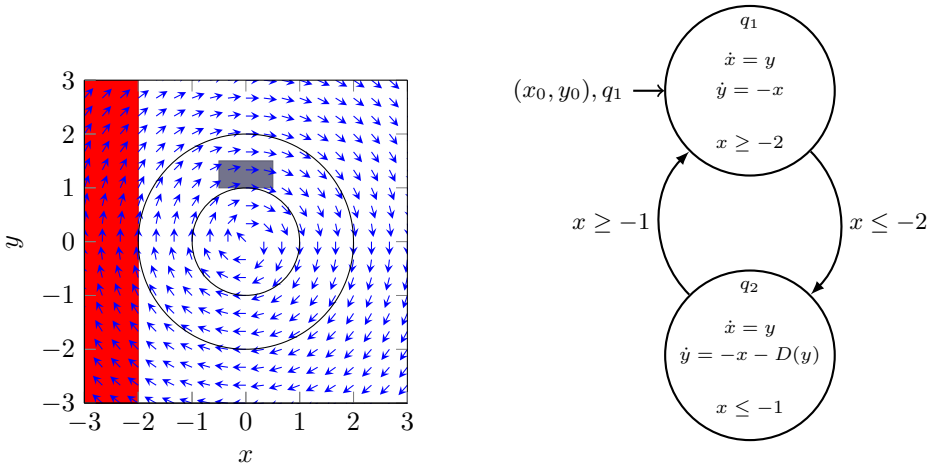
Untyped encodings for TLA$^+$ formulas is an alternative encoding implemented in TLAPS. When using this encoding, type inference is delegated to the SMT solver. Untyped encodings are used if type inference fails. In this case, a single SMT sort U is used to represent TLA$^+$ values and its operators are represented as uninterpreted functions having sort U as their arguments [5]. In order to encode real arithmetic formulas, we declare uninterpreted functions that embed SMT reals into the sort U representing TLA$^+$ values, i.e., $real2u : \text{Real} \rightarrow \text{U}$ and $u2real : \text{U} \rightarrow \text{Real}$. Real represents SMT reals, $real2u$ embeds SMT reals

into a sort U representing 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 U. Real arithmetic operations over 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 $U \times U \to U$. 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 [5,6].

3 Safety Verification of a Hybrid System

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. 1a). 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. 1b) 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 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. 1b 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



(a) Vector field (mode q_1)

(b) Hybrid automaton

Fig. 1: Hybrid system model of an oscillator

illustrated in Fig. 1a, 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 2 shows how both of these conjectures are represented in the TLA$^+$ syntax and successfully 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

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

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 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 and dReal can – in addition 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. 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
2. 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
3. Lamport, L.: Specifying Systems: The TLA$^+$ Language and Tools for Hardware and Software Engineers. Addison-Wesley (June 2002)
4. Lamport, L.: Industrial Use of TLA$^+$. https://lamport.azurewebsites.net/tla/industrial-use.html (2019), [Online; accessed March 2023]
5. Merz, S., Vanzetto, H.: Harnessing SMT solvers for TLA$^+$ proofs. Electron. Commun. EASST **53** (2012). https://doi.org/10.14279/TUJ.ECEASST.53.766
6. 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
7. 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

Appendix

Demo and Screencast

Instructions on how to build and install the tool along with a user manual are available in the *INSTALL* and *User_Manual.txt* files, respectively, at `https://github.com/Ovini99/TLAPS_Real`. A screencast of a demo of the extended TLAPM is available at `https://github.com/Ovini99/TLAPS_Real/tree/main/demo`.

Based on the encodings in *Sect. 2*, we generated two files to run through the extended proof manager (via Toolbox and command line). The first file contains a combination of arithmetic formulas where the proof manager successfully infers the types and are proven true; a sample of such formulas is depicted in *Fig. 4.3*. The complete list of examples is available in *RealIntsTest.tla*. A second file, i.e., *RealIntsTestFail.tla*, includes examples of formulas where the typed and untyped encodings fail or the encodings successfully infer the types, but the conjecture is proved false. A sample of such formulas is depicted in *Fig. 4.4*.

Further, we generated two files (*RealNumExample1.tla*, *RealNumExample2.tla*) to prove the theorems related to the hybrid automaton in *Section 3*.

```
EXTENDS TLAPS, Reals, Integers

\* Encoding will succeed.
\* All variables are of type Real.
\* The conjecture will prove true since there is a solution.
THEOREM \E x,y,a,b \in Real:((x*x)+(y*y))*((x-b)*(x-b))=a*a*x*x  BY Z3

\* Encoding will succeed.
\* All variables are of type Int.
\* All number are integers. Use of integer division.
\* The conjecture will prove true since there is a solution.
THEOREM \E x,y \in Int: y + (1 \div 1) = x  BY Z3

\* Encoding will succeed.
\* Variable x is a real between -5.0 and 5.0.
\* 0.0 and 5.0 are expressed as reals.
\* The conjecture will prove true since there is a solution.
THEOREM \E x \in (-5.0..5.0): x*x - 5.0 = 0.0  BY Z3
```

Fig. 4.3: Sample of conjectures from *RealIntsTest.tla*

```
EXTENDS TLAPS, Reals, Integers

\* Encoding will fail.
\* Mixing of reals and integers is not permitted.
\* Variable x is of type Real and number 0 is of type Int.
THEOREM \E x \in Real: x - 0 = 1.0  BY Z3

\* Encoding will fail.
\* Division is not supported between integers.
\* Integer division should have been used instead.
THEOREM \E x,y \in Int: y + (1/1) = x  BY Z3

\* Encoding will succeed.
\* Variable x and -1.0 are both Reals.
\* The conjecture will prove false since
\* there is no solution in Real.
THEOREM \E x \in Real: x*x = -1.0  BY Z3
```

Fig. 4.4: Sample of conjectures from *RealIntsTestFail.tla*