An Advanced Data Type with Irrational Numbers to Implement Time in DEVS Simulators

Damián Vicino
Université Nice Sophia Antipolis
I3S UMR CNRS 7271
Carleton University
Dept. of Systems and Computer Engineering
vicino@i3s.unice.fr

Olivier Dalle
Université Nice Sophia Antipolis
I3S UMR CNRS 7271
INRIA - Sophia Antipolis
olivier.dalle@unice.fr

Gabriel Wainer
Carleton University
Dept. of Systems and Computer Engineering
gwainer@sce.carleton.ca

ABSTRACT

In the Discrete-Event System Specification (DEVS) time variables are Real numbers. This is common to other simulation formalisms for Discrete-Event Simulation (DES). Current simulators for these formalisms approximate time variables using floating-point or rational representations. Neither of them is capable to adequately represent irrational numbers. The representation of these numbers is important, especially for studying systems with geometrical properties. The use of approximations, as floating-point, may silently introduce errors to the causality chain. These errors may produce incorrect simulation trajectories without informing about them. Here, we propose a new data type combining rational data types with computable calculus concepts. This new data type extension provides representation, and operation, with subsets of irrational numbers. The proposed data type only provides four operations (+, -, $<$, =), those are sufficient for implementing simulators for DEVS and other DES formalisms. Usage of this data type has no significant complexity penalty for simulation not using irrational numbers.

Author Keywords

Time; Data types; Irrational

ACM Classification Keywords

I.6.8 SIMULATION AND MODELING : Discrete event

1. INTRODUCTION

The Discrete Event Simulation (DES) family of methods consider that models generate and react to events. Those events are usually implemented as messages passed between models at discrete points in the simulation continuous timeline. The usual domain of the variables that hold the discrete points in the timeline in DES is $\mathbb{R}^+$. Unfortunately, it is impossible to find a computer representation for all the values in $\mathbb{R}^+$.

The problem of including irrational numbers in Time data types in simulation is not new, as it is common to use irrational numbers when defining models in certain fields (i.e, physics, mechanics, chemistry, etc.). In addition, it is also common to use irrationals for any model relying on geometrical concepts. For example, in a model of a pendulum clock, we could model the pendulum to produce an event every time it completes a period. From physics, we can define the period as $(2\pi \cdot \sqrt{\frac{L}{g}})$ where L, g, and π are constants.

We are interested in the use of irrational numbers to model Discrete Event systems in which time is continuous. We can categorize previously proposed solutions to represent continuous time in three groups: intervals arithmetic, fixed-length approximation, and symbolic solving. These solutions solve different subsets of representation issues, but they carry, in some cases, new concerns or limitations.

Most approaches introduce approximations. The major risk of approximating in DES is producing breaks of the causality chains. In DES, the state of the simulation can be thought as a function of its history, producing a causal relation between them. When an event occurrence is displaced, the evolution of the simulation may diverge. When this happens, we say that the causality chain is broken. Current simulator implementations are generally silent about these errors, because it is usually impossible to detect them properly.

From Computable Calculus [1] point of view, some Real numbers have been proven to be non computable (Chaitin constants for example [3]). Likewise, for computable Reals, to know if a number is rational or not (in the general case) is not decidable. And, the comparison of two computable reals (in the general case) is not decidable either.

Our goal is thus to define a new data type providing proper representation of time for simulation without generating timeline errors. This data type should include representation subsets of necessary computable irrational numbers. To do so, we introduce a new data type that augments the set of values that can be represented using rational data types for time variables. This data type allows defining and operating with controlled subsets of computable irrationals, and it is based on concepts from Computable Calculus [1]. It also prevents breaking causality chains due to imprecision in the timeline.

SpringSim-TMS/DEVS, 2016 April 3-6, Pasadena, CA, USA

Our new data type separates rational and irrational components. The irrational components are selected so we can provide a unique construction of the numbers, allowing comparison and tests for irrationality to be decidable.

We present irrational components including three subsets of computable numbers, multiples of π, multiples of e^{π} and irrational numbers obtained as root of integer numbers. We consider these are a large addition to current state of the art. Also, in Section 8., we explain how this set can be enlarged to cover other numbers needed in some modeling scenarios.

In particular, we will focus on the operations needed for implementing DEVS simulators. Thus, our data type only provides four operations ($+$, $-$, $=$, $<$), as these are the only operations needed by the abstract simulator algorithms defined in [17].

The most common concern about exact representation and operation is complexity. We will show that the impact of our approach is constant and negligible for timelines that do not require the support for using irrational numbers. In the case that the timeline requires their use, we only have higher complexity in the "compare by greater" than operation. Another common concern is the halting of operations between irrational numbers. We will explain, how given the irrational subsets being included cover the properties we require, the completion of the operations can be guaranteed.

2. BACKGROUND

2.1. Discrete-Event System Specification

We have particular interested in the Discrete-Event System Specification (DEVS) formalism [17]. This is a DES formalism proposed by Zeigler in the 70s, whose simulator has been implemented in multiple languages and platforms. In DEVS, models are defined in a modular and hierarchical way. The modeling hierarchy has two kinds of components: atomic models and coupled models.

An atomic model is always in a specific state. The model defines internal and external transition functions to process endogenous and exogenous events. Each time an event is processed the state is modified. Using this state, a time-advance function is used to compute a life-span delay. Endogenous events are produced when this lifespan delay is consumed without the arrival of any other event. At the time of processing an endogenous event, the output-function is used to output an event.

Coupled models define hierarchy and interconnection between atomic models. In this hierarchy leaves are atomic models, and oriented links represent the routing of events between them.

The common approach used for passing events between models has four steps. First, the time of next event (the minimum of the times remaining in each model until the next internal transition) is computed. Second, the resulting output events are computed. Third, the events produced as output of submodels are routed in the network to their destination inputs. The time remaining until the next round (subtracting the time elapsed from the scheduled time advance) is recomputed.

DEVS was extended in several ways to adapt the formalism for various goals. For example, in Symbolic DEVS [16, 17], the formalisms was extended to define time as linear polynomials in place of real numbers. Another extension related to the time definition is called Rational Time Advance DEVS (RTA-DEVS) [12]. In RTA-DEVS time is defined as intervals with rational borders.

2.2. Computable Calculus

Computable real numbers were introduced by Alan Turing in his seminal paper "On Computable Numbers, with an Application to the Entscheidungsproblem" [13]. He defines: "The computable numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means". Different authors provided several equivalent definitions, for example an alternative definition is "a number is computable if it exists an algorithm to produce each digit of its decimal expansion, and for any digit requested it finishes the execution successfully". There are subsets of real numbers that have been proven not to be computable. Examples of such a subset include the family of Chaitin's constants [3].

A new area of applied mathematics was developed based on the theory of computable real numbers, called Computable Calculus [1]. The main goal of this area is finding the adjustments that need to be done to theorems and properties on real calculus that do not apply to computable real numbers. For example, since there is an algorithm for describing each computable number, the set of computable numbers is enumerable, while real numbers are not [13].

Some interesting results in Computable Calculus are the following non-decidable problems [1]:

- In the general case, it is impossible to decide if a computable real number is irrational or not.
- In the general case, it is impossible to decide if a computable real number is greater than zero or equal to zero.
- Other non-decidable problems can be derived from the previous two examples, which are also non-solvable; for instance the equality of two numbers and the order of two numbers.

Detailed proofs of these non-decidable problems can be found in "Computable Calculus" by Oliver Aberth [1].

To operate with computable numbers, we can use interval arithmetic considering k-digits expansion and the resulting intervals [1]. In interval arithmetic, the addition of two intervals is defined as the interval where the lower border is defined as the addition of the lower borders of the operands.

Likewise, the higher border is defined by the addition of the higher borders of the operands [10]. Similar definitions exist for other operations as subtraction, multiplication and division [10]. The definition of comparison operations of intervals is more complicated because the two intervals being compared may intersect, which requires multiple comparison operations [1, 10].

3. RELATED WORK

Foundational work about Time representations in computers was developed in the 70s and 80s. Many of these focus on studying problems related to the synchronization of distributed systems and real time applications [5].

Lamport formalized time [6] for distributed systems, defining it as a sequence of partially ordered events using the relation "happened before". Later, formal approaches were proposed based on temporal logic [8, 9]. Temporal logic allows the specification of constraints on events and continuous intervals of time. Temporal logic can be used for writing formal proofs on properties, data types and algorithms.

In 2009, Clock Constraint Specification Language (CCSL) [2] was proposed as a standard to extend the Unified Modeling Language (UML). This new addition is used to describe relations between time instants in dense and discrete time representations.

4. COMMON APPROACHES IN DES

Discrete-Event Simulators usually represent time with one of the following data types: floating-point, fixed-point, integers, or rational. Most of these data types are native to programming language and processors. Those not natively available can be easily implemented combining native ones. Nevertheless, these data types have well-known limitations.

- Floating-point [4] cannot work properly with periodic numbers (e.g. 1/3). They also lack closure under addition, which forces us to work with approximate results when adding two numbers that are represented accurately. And, they have no tracking mechanisms to detect when an operation produced an approximate result.
- Integer and fixed-point (an integer with a fixed position decimal point) have the issue of not being able to represent periodicity properly. In addition, they also have low ability to deal with multiple scales, forcing the adoption of a lowest common scale factor among all models. Using the lowest common scale factor introduces space-time constraints. For example, the size of an integer required for representing every value covered by a floating-point is exponential on the size of the floating-point exponent.
- The limitations of using rational numbers are not related to periodicity, which is properly supported. The main limitations are the complexity, which is higher than previous data types. The space complexity is affected by the fact that rational numbers have multiple representations per value. Thus, representing the same set of values requires more space.

Also, neither floating-point, fixed-point, integer, or rational support irrational number representation. They are approximated by programmers to the closest representable number when programming models.

Any uncontrolled approximation can lead to three common timeline errors: time-shifting [14], event-reorder [14], and Zeno [7, 8, 14].

The most common errors in DES timelines are the time-shifting errors. We are in presence of time-shifting if events occur slightly earlier or later than formally defined and their order is not modified. Time-shifting errors usually have a minor impact or no impact at all on the simulation execution.

Other errors produced by approximating values are the event-reordering errors. Here, events on the timeline are permuted and the partial order of events is erroneous.

Finally, the distance between two events in the timeline can be defined as any real number, particularly those very close to zero. Approximating time values close to zero may produce zero. Thus, the advance of the simulation could stale because of an infinite sequence of zero time-advances. Systems producing this behavior were studied in concurrent systems. This is called the Zeno problem [7, 8]. In some simulation formalisms, as DEVS, legal models [17] never reproduce Zeno behavior. However, legal models can still reproduce the error if time is approximated when computing the simulation.

Model correctness' proofs are useless for detecting these errors. Formalisms are agnostic about computation models, and the errors are introduced by them.

Furthermore, event-reordering, and Zeno errors are practically impossible to predict. They may occur following irregular frequency patterns, which, in the worst case, will pass undetected through the validation tests.

The problem of including irrational numbers on Time data types for DESs is not new. Previous proposed solutions could be categorized in two groups: interval arithmetic, and symbolic solving. These solutions solve different subsets of representation issues. However, they carry some new concerns.

In approaches based on interval arithmetic, each number is represented by an interval rather than an exact value. The main problem with interval arithmetic is the comparison operations, they are not as decisive as with real numbers. It is not possible to decide order of the values when the intervals overlap. Additionally, since intervals are distances, summing intervals is always incrementing in size.

The simulation current-time in DES is computed as a large sum. This sum increments the length of the intervals until overlapping cannot be avoided. Sometimes, such overlapping

results in non-decidable comparisons crashing the simulation. In that case, the simulation can restart with smaller size intervals. However, this does not guarantee to solve the problem. Intervals representing the same number always overlap, whatever their size. In addition, DEVS defines especial functions for handling simultaneous events. Being comparison by equality non-decidable, the models never use these functions. An example of this approach is the proposed in Rational Time Advance DEVS (RTA-DEVS) [12].

Finally, in SymbolicDEVS [16, 17], it was proposed the use of symbolic algebra to represent time. In symbolic algebra, the numbers are represented by expression defining them, the operations are defined as composition rules of these expressions, and solvers are implemented to evaluate comparisons when needed. This approach looks promising in the sense of accuracy, but as far as we know, SymbolicDEVS [16, 17] was the only formalism using it. Here, the expressions were limited to represent roots of polynomials. The methods for solving polynomials have high complexity, and there is no representation for transcendental numbers using them.

Symbolic Algebra is also implemented in several Mathematical applications and libraries, but restrictions apply to them. In these tools, when symbolical manipulation is unknown for an expression, they operate using a fixed length expansion of digits, which has the approximation problems mentioned in Section 4..

We said in Section 2.2., it was proven in Computable Calculus that comparisons of arbitrary expressions are non-decidable [1], and using fixed lengths approximations is not a valid solution. For example, in a one-digit expansion, π and 3 are considered equal, while we know they are different. However, this approach seems to be the most promising one, using expression manipulation techniques allows, in some cases, to detect relations between two expressions without the need for expanding any digit. The equality problem can then be reduced for some well know expressions providing a way to avoid falling into non-decidable operations.

An example to show how this method is effective is the implementation of the comparisons of square roots of rational numbers. In fixed-length or intervals it may not be possible to compare for equality the square root of 2 against the square root of 2.00000000001 because they may be considered equal after a fixed-length rounding or they may produce overlapping intervals when using interval arithmetic. On the contrary, it is easy to compare them using Symbolic Algebra manipulations, for example by comparing the two radicands.

5. PROPOSED REPRESENTATION

We mentioned, in Section 2.2., there are formal proofs about numbers that are not computable. In addition, it was proven for the general case that it is not possible to decide if a computable number is rational or not, or sorting two of them. However, even when it is proven that there is no solution in the general case, we can restrict our set of numbers to obtain satisfactory results.

In this section, we present a new data type. This data type extend rational data types introducing well-known subsets of computable irrational numbers. Our approach is inspired of ideas from Computable Calculus, and Symbolic Algebra. And, our main objective is to generate correct trajectories.

Our new data type represents a real with two components, a rational and an irrational. For the rational component we could use any rational data type with no approximation. For the irrational part we use a composition of known subsets of irrationals, and provide a method to construct them uniquely. The reason for the unique construction is being able to decide irrationality and equality by construction. Once we are able to decide these two operations, the other operations can be decided too.

For providing unique construction, we define the structure of the data type as a tuple $\langle r, I\rangle$ where r is a rational number, and I is a set of tuples $i_k = \langle c_k, A_k\rangle$ with c_k a rational coefficient and A_k an expansion algorithm describing a computable irrational number selected from a predefined set. This representation can be interpreted as shown in Formula 1.

$$r + \sum_{i \in I} i.c \cdot i.A$$

Formula 1: Irrational data types interpretation

Initially, we propose two subsets of irrational numbers, rational multiples of π, and rational multiples of e^{π}. We introduce later, the irrational numbers obtained as square root of integer and rational numbers (using integer exponents). We consider these are a large addition to current state of the art.

For example, we can represent π as $\langle 0, \{\langle 1, \pi\rangle\}\rangle$, $\frac{1}{3}$ as $\langle \frac{1}{3}, \emptyset\rangle$, and $\frac{e^{\pi}}{2}$ as $\langle 0, \{\langle \frac{1}{2}, e^{\pi}\rangle\}\rangle$. If we need to add these three numbers, the result can be represented as $\langle \frac{1}{3}, \{\langle 1, \pi\rangle, \langle \frac{1}{2}, e^{\pi}\rangle\}\rangle$.

$$A = \texttt{set of all irrational algorithms defined}$$
$$d = dim(A), \forall i \in \mathbb{N}, 0 < i \leq d, \forall r \in \mathbb{Q}^d$$
$$\sum_{j=0}^{d} r_j \cdot A_j = 0 \Leftrightarrow \forall j \in \mathbb{N}, 0 < j \leq d, r_j = 0$$

Formula 2: Linear Independence over $\mathbb{Q}$

The idea is to be certain that a number defined by an algorithm cannot be described by the linear composition of the others over $\mathbb{Q}$. In the case of A having π and e^{π}, there is no rational number that can be multiplied or added to it in order to make them equal or rational, the proof can be found in [11].

While we cover the requirements, we can keep introducing new numbers, for example, we can use $\langle \pi, e^{\pi} \rangle$, or $\langle \pi, \sqrt{2} \rangle$, or $\langle e, \sqrt{3} \rangle$, or other useful combinations, for defining the set A, and it can be defined in a simulation by simulation basis.

6. PROPOSED OPERATIONS

The first operation we are interested in is the addition. We define the addition of two numbers as the addition of each of their components. First, we add the rational components. Second, we add the coefficients of each element in the irrational component of the first operand with its matching part in the second operand. This has low impact on the performance of the addition operation. For instance, if we allow a single irrational component, as rational multiples of π, we require adding two rational values, the rational component, and the coefficients associated to π. The addition complexity for this is the complexity of 2 rational additions. In the general case, the complexity of an addition is the complexity of $dim(A)+1$ rational additions, where A is the set of irrational constants defined. Similarly, subtraction can be implemented in a component by component basis.

The second operation we are interested in is the comparison by equality. The uniqueness of the representation can be used to compare two numbers for equality just by looking at their components. The complexity of this operation is also dependent on the rational data type used, an equality operation requires $dim(A)+1$ rational equality operations.

The third operation we are interested in is compare by greater-than. This is the most complex operation requiring multiple steps. The first step is checking for equality, if both numbers are equal, we return false. The second step is discarding the components that are equal, if we obtain a single component that is not equal, which is the common result when operating with rational numbers, we can compare the coefficient of the distinctive component for deciding. In case more than one of the components are different, this is the only moment we require algorithmic expansions of the distinctive components. The distinctive components are multiplied by the coefficient and compared following the algorithms described in [1]. The complexity varies depending on the expansion algorithms of each component; in case we operate between rational numbers, the complexity is only incremented in $dim(A)$ rational equality operations.

We show in Listing 1 a possible implementation of this data type with support for $\langle \pi, e^{\pi} \rangle$ in C++. The code is only for reference, more sophisticated versions can be written allowing selection by template parameters of the subsets being used, and detecting conflicts between them, this would be a proper approach for a production quality version.

```
class iTime{
    rational _q, _pi_coef, _e_pi_coef;
    const int get_digit_k(const int& k) const { ... }
public:
    iTime(): _q{0}, _pi_coef{0}, _e_pi_coef{0}{}
    iTime(const rational& q, const rational& pic, const
         rational& epic): _q{q}, _pi_coef{pic}, _e_pi_coef
         {epic} {}
  iTime(const iTime& rhs): _q (rhs._q), _pi_coef(rhs.
      _pi_coef), _e_pi_coef(rhs._e_pi_coef){}

    iTime operator+=(const iTime& rhs){
        _q += rhs._q;
        _pi_coef += rhs._pi_coef;
        _e_pi_coef += rhs._e_pi_coef;
        return *this;
    }

    iTime operator-=(const iTime& rhs){
        _q -= rhs._q;
        _pi_coef -= rhs._pi_coef;
        _e_pi_coef -= rhs._e_pi_coef;
        return *this;
    }

    bool operator==(const iTime& rhs) const{
        return (_q == rhs._q && _pi_coef == rhs._pi_coef
             && _e_pi_coef == rhs._e_pi_coef);
    }

    bool operator<(const iTime& rhs) const{
        //working with a single coeficients
        if (_pi_coef == rhs._pi_coef && _e_pi_coef == rhs.
             _e_pi_coef)
            return _q < rhs._q;
        if (_q == rhs._q && _pi_coef == rhs._pi_coef)
            return _e_pi_coef < rhs._e_pi_coef;
        if (_q == rhs._q && _e_pi_coef == rhs._e_pi_coef)
            return _pi_coef < rhs._pi_coef;
        //checking integer_part of the numbers
        bri l_rational = _q + _pi_coef * pi.rational() +
             _e_pi_coef * epi.rational();
        bri r_rational = rhs._q + rhs._pi_coef * pi.
             rational() + rhs._e_pi_coef * epi.rational();
        if (l_rational.numerator()/l_rational.denominator
             () < r_rational.numerator()/r_rational.
             denominator()) return true;
        if (l_rational.numerator()/l_rational.denominator
             () > r_rational.numerator()/r_rational.
             denominator()) return false;
        //checking decimal part of numbers
        for (int i=0; i < MAX_ALLOWED_DIGITS; i++){
            if (get_digit_k(i) < rhs.get_digit_k(i))
                 return true;
            if (get_digit_k(i) > rhs.get_digit_k(i))
                 return false;
        }
        throw std::exception();
    }
};
```

Listing 1: Irrational time

7. EXTENDING WITH PARAMETRIC SUBSETS

A third group of interesting irrational numbers to include in our data type are the square roots. Here, we are not referring to a single irrational number multiplied by a rational coefficient as in previous cases, but including a set of constants that can each be multiplied by a coefficient.

We chose to extend with square roots because of their intensive use in several areas, especially those including models using geometry. To introduce these numbers, we need to solve three problems: i) finding how they interact with the previously existing ones, ii) finding how to detect if the numbers

are rational, and iii) finding how to detect if the result of an addition is rational. We will start discussing these problems for roots of integers, and will later generalize it for rational roots.

First, we know these new numbers do not conflict with previously introduced irrationals because they are non-transcendental. All numbers included before are transcendental, and it is impossible to obtain a non-transcendental number as the product of a rational by a transcendental irrational number.

Second, not every square root of an integer is irrational. We need to check if numbers being represented are irrational. For that purpose, we obtain the prime factorization of the number to be represented and check if any factor has an odd exponent; if this is the case, we are in presence of an irrational number, else we should derivate the representation to the rational component.

Obtaining prime numbers factorization is an expensive operation for large numbers. In some programming languages, most of the complexity introduced can be palliated at compile time. For example, using template meta-programming, the prime factorization of the numbers used can be pre-computed.

To represent square roots uniquely, we simplify the expression factorizing the represented number as the product of a rational by a square root of product of primes with exponent 1. The factors extracted outside of the square root are used as coefficient of the irrational constant introduced. We show an example representing $\sqrt{72}$ in Formula 3.

$$\sqrt{72} = \sqrt{2^3 \cdot 3^2} = 3 \cdot 2 \cdot \sqrt{2} \Rightarrow n = \langle 0, \{\langle 6, \sqrt{2}\rangle\}\rangle$$

Formula 3: Representation for $\sqrt{72}$

In previous defined subsets $\langle \pi, e^{\pi} \rangle$ the digits-expansion function did not require any parametrization. Here, we are introducing a whole family of irrational constants that will be introduced on demand to the set. The parameter for defining these constants is the integer value in the radicand. We introduce a new pair coefficient and algorithm to the set of irrational constants (A) for each radicand used.

For comparisons we operate as before, we iterate the irrational constants and their coefficients, generating digits only when needed.

For adding, we have three scenarios to consider: First, when the addition produces a rational number; second when the addition produces a result in the same set of irrational square roots; and third, when the addition can not be mapped to any radicand in the set of irrational square roots.

First case is not possible. Addition of two square roots result is rational only using perfect square radicands, we show proof in Formula 4. Perfect square radicands are not irrational square roots. Then, they are never defined using irrational constants in our representation.

$$\begin{gathered}
a, b \in \mathbb{Z}, c \in \mathbb{Q} \\
\sqrt{a} + \sqrt{b} = c \Rightarrow a + b + 2\sqrt{ab} = c^2 \\
2\sqrt{ab} \in \mathbb{Q} \Leftrightarrow \exists d \in \mathbb{Z} : ab = d^2 \Rightarrow a = \frac{d^2}{b}, \frac{d}{\sqrt{b}} + \sqrt{b} = c \\
\Rightarrow d + b = c\sqrt{b} \\
c \in \mathbb{Q} \Rightarrow \exists e \in \mathbb{Z} : b^2 = e \\
a = \frac{d^2}{b} \Rightarrow \exists f \in \mathbb{Z} : a^2 = f \\
\texttt{A and B have to be perfect squares}
\end{gathered}$$

Formula 4: Proof that addition of square root is rational only if radicands are perfect squares

In second case, if we have addition of equal radicand square roots, we can operate with coefficients only. For example, $2 \cdot \sqrt{2} + 1 \cdot \sqrt{2} = 3 \cdot \sqrt{2}$.

In third case, it is possible to generate irrationals that are not square root of integers using addition. We show an example in Formula 5.

$$\begin{gathered}
\exists k \in \mathbb{Z} : \sqrt{2} + \sqrt{3} = \sqrt{k} \Rightarrow (\sqrt{2} + \sqrt{3})^2 = (\sqrt{k})^2 \\
\Rightarrow 2 + 3 + 2 \cdot \sqrt{2 \cdot 3} = k \Rightarrow 5 + 2 \cdot \sqrt{6} = k \\
\Rightarrow \frac{k-5}{2} = \sqrt{6} \\
\texttt{Absurd, because } k \in \mathbb{Z} \wedge \sqrt{6} \notin \mathbb{Z}
\end{gathered}$$

Formula 5: Proof that addition of square roots of integers can produce a new class of irrationals

If we do not include any other class of non-transcendental numbers representation, it is safe to operate in a component by component basis, adding matching radicand coefficients. In the case a new class is introduced, as grade 4 roots, special care has to be taken when operating to promote the addition result to the right family of values.

An extension to represent irrationals obtained as square roots of rationals is possible. For extending this way, we use now prime numbers with exponents 1 and -1. Any other exponent requires factorizing and compensating using the rational coefficients. In addition to previous concerns, we need to check what happens when these new irrationals are added.

In Formula 6, we show proof that adding inverses never produces a rational number.

In Formula 7, we show proof that it is possible outcomes of adding irrational square roots of rationals is not representable as the square root of a rational number.

Similarly to the family of square roots of integers, if we do not include any other class of non-transcendental numbers

$$A \subset primes, B \subset primes, |A| < \infty, |B| < \infty,$$
$$A \cap B = \emptyset, A \cup B \neq \emptyset, p,q,k \in \mathbb{Q}$$
$$a = \prod A \qquad b = \prod B \qquad k = p\sqrt{\frac{a}{b}} + q\sqrt{\frac{b}{a}}$$
$$k^2 = p^2\frac{a}{b} + q^2\frac{b}{a} + 2pq\sqrt{\frac{ab}{ba}}$$
$$= p^2\frac{a}{b} + q^2\frac{b}{a} + 2pq\sqrt{1}$$
$$= \frac{p^2a^2 + q^2b^2 + 2pqab}{ab}$$
$$\Rightarrow k^2ab = (pa+qb)^2 \Rightarrow k\sqrt{ab} = pa + qb$$
$$\texttt{absurd}: k\sqrt{ab} \notin \mathbb{Q} \ \texttt{and} \ (pa+qb) \in \mathbb{Q}$$

Formula 6: Proof that adding square root of inverses is irrational

$$\sqrt{\frac{3}{2}} + \sqrt{\frac{5}{7}} = \sqrt{\frac{3}{2} + \frac{5}{7} + 2\sqrt{\frac{15}{14}}}$$
$$= \sqrt{\frac{31 + 2\sqrt{15.14}}{14}} = \sqrt{\frac{31 + 2\sqrt{210}}{14}}$$
$$\sqrt{210} \notin \mathbb{Q} \Rightarrow \frac{31 + 2\sqrt{210}}{14} \notin \mathbb{Q}$$

Formula 7: Proof that addition of square roots of rationals can produce a new class of irrationals

representation, it is safe to operate in a component by component basis. In the case a new class is introduced, as grade 4 roots, special care has to be taken when operating to promote the addition result to the right family of values.

8. EXTENDING THE DATA TYPE FURTHER

We have now basic support for irrational that may be extended or customized to support accurate simulation. The approach we discussed introduced a new subset of irrational numbers at each step. We cannot do this for every irrational. Not every subset is compatible with those shown before.

For example, it is not proven that π and e are linearly independent over $\mathbb{Q}$, then we are not certain that we can use both safely together. Two approaches can be taken, first we can remove π from the set of constants when using e; second we can set a limit for digits allowed to be expanded, and trigger an error when an operation could not be decided after reaching the expanding limit. We encourage the use of this limit even in case that it is theoretically safe.

A stronger property than the one we required named Algebraic Independence over $\mathbb{Q}$ is explained in [11]. Several irrational numbers have been proved to be Algebraic Independent over $\mathbb{Q}$; any of them can be used to extend our data type.

The same reasoning presented for square roots can be applied to any other subset of irrationals that need to be added, for instance introducing cubic roots. We can use similar reasoning for extending the arithmetic operations provided by the data type to something more than addition. We define them following the algorithms defined in [1], and check the results are always part of the original set. In case the results are not part of the set, a new set needs to be defined dynamically, as we do when we add two irrational square roots not sharing the radicand.

9. PERFORMANCE EVALUATION

We implemented a set of experiments using a partial implementation of the algorithms in C++11. The experiments were compiled using clang-600.0.57 for x86_64-apple-darwin14.1.0. We ran our experiments in a Intel core i7 2.8Ghz machine with 8GB of DDR3 ram.

Our experiments are focused in studying the use of the new data type when operating only with rational numbers. Here the operation of irrational coefficients is pure overhead, and we want to measure this overhead.

In our experiments, we draw from uniform distribution random sequences of thousand numerators and denominators between 1 and 10 and operate with them. We use the rationals generated and run repetitions of adding and comparing them using rational data type from Boost and iTime using rational from Boost as internal representation for coefficients.

9.1. Results

In Figure 1, we show the impact of using iTime against rational in additions. The overhead is constant and is lower than 50%. Also, we can see that 10 million addition operations could be produced in less than a minute.

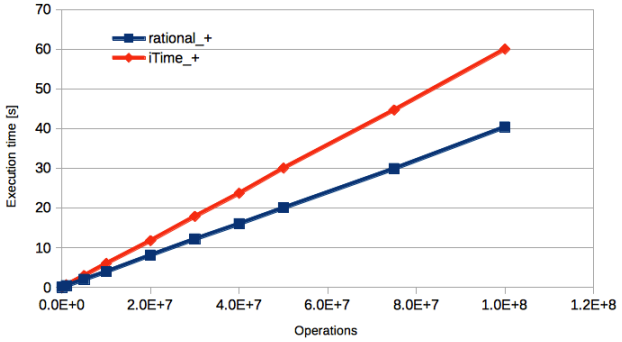

Figure 1: Comparison of rational vs iTime execution time running addition operations.

Similar experiments were conducted for evaluating the other operations.

For comparing by equality overhead resulted constant and lower than 15%. In this experiment we could evaluate up to 500 million operations in less than 40 seconds.

For comparing by lower than, the overhead is constant and lower than 25%. In this experiment we could evaluate up to 500 million operations in less than 40 seconds.

10. CONCLUSIONS

In this paper, we presented a method for representing time values in simulators. We described the limitations of previous approaches from the literature. In particular, we interested in the lack of representation for irrational numbers.

Our new data type is based in concepts from computable calculus. This new data type has the limitation of not allowing arbitrary irrationals to participate in every operation. However, the method introduces significant increments to current state of the art.

The method does not introduce significant overhead compared to rational data types. The complexity is significantly incremented only in scenarios where expanding digits for solving greater-than comparison is a must. The complexity in those cases is related to the complexity of expanding the digits of the required constants.

We presented the four operations required for implementing a DEVS simulator (+, -, $<$, =). In addition, we explained how to extend to other operations. We introduced the subsets of multiples of π, e^{π}, and the square roots of integer and rational numbers. We explained how to extend the data type to support further irrational values. And, we provided a simple performance test for the overhead introduced on simulations that can be run with simpler old data types.

In the future, we plan to introduce the new data type to our simulator [15] and experiment with new subsets of irrationals.

REFERENCES

[1] ABERTH, O. *Computable Calculus*. Academic Press, 2001.

[2] ANDRÉ, C. Syntax and semantics of the clock constraint specification language (CCSL).

[3] CHAITIN, G. J. A theory of program size formally identical to information theory. *Journal of the ACM (JACM) 22*, 3 (1975), 329–340.

[4] GOLDBERG, D. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys (CSUR) 23*, 1 (1991), 5–48.

[5] GUPTA, V., HENZINGER, T. A., AND JAGADEESAN, R. Robust timed automata. In *Hybrid and Real-Time Systems* (1997), Springer, pp. 331–345.

[6] LAMPORT, L. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM 21*, 7 (1978), 558–565.

[7] LEE, E. A. Constructive models of discrete and continuous physical phenomena. Tech. rep., Technical Report UCB/EECS-2014-15, EECS Department, University of California, Berkeley, 2014.

[8] MALER, O., MANNA, Z., AND PNUELI, A. From timed to hybrid systems. In *Real-time: theory in practice* (1992), Springer, pp. 447–484.

[9] MANNA, Z., AND PNUELI, A. *The temporal logic of reactive and concurrent systems: specifications*, vol. 1. Springer Science & Business Media, 1992.

[10] MOORE, R. E. *Interval analysis*, vol. 4. Prentice-Hall Englewood Cliffs, 1966.

[11] PHILIPPON, P. *Introduction to Algebraic Independence Theory*. No. 1752. Springer Science & Business Media, 2001.

[12] SAADAWI, H. *Verification Methodology for DEVS Models*. PhD thesis, Carleton University, 2012.

[13] TURING, A. M. On computable numbers, with an application to the entscheidungsproblem. *J. of Math 58*, 345-363 (1936), 5.

[14] VICINO, D., DALLE, O., AND WAINER, G. A data type for discretized time representation in devs. In *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques* (2014), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp. 11–20.

[15] VICINO, D., NAYUNKURU, D., WAINER, G., AND DALLE, O. Sequential pdevs architecture. In *Proceedings of the TMS/DEVS 2015* (2015), SCS (The society for modeling and simulation international).

[16] ZEIGLER, B. P., AND CHI, S. Symbolic discrete event system specification. *Systems, Man and Cybernetics, IEEE Transactions on 22*, 6 (1992), 1428–1443.

[17] ZEIGLER, B. P., PRAEHOFER, H., AND KIM, T. G. *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. Academic press, 2000.