PROVABLY SPURIOUS QUALITATIVE SIMULATION PREDICTIONS THAT JUST WON'T GO AWAY

Nuri Taşdemir and A. C. Cem Say
Boğaziçi University
Department of Computer Engineering
Bebek, 34342, İstanbul, Turkey
nuri.tasdemir@boun.edu.tr, say@boun.edu.tr

Abstract

It is known that sound and complete qualitative simulators do not exist; that is, there exist inputs which lead to ineradicable spurious behaviors, proving whose inconsistency is an undecidable task, and thus any sound qualitative simulator has to include them in its output. In this paper, we ask whether the next best thing, that is, a single sound qualitative simulator which detects and eliminates all *provably* inconsistent predictions, is possible, and obtain a negative answer. We prove that, for any sound qualitative simulator Q, which possesses two other reasonable properties that we define, there exists an input model which causes Q to predict a spurious prediction that can in fact be eliminated easily by many other qualitative simulators. Our result is a qualitative simulation version of Gödel's celebrated Incompleteness Theorem. We also show that, even when one restricts attention to models without self-reference, there exist infinitely many provably inconsistent inputs, which require so much time for a consistency check that such a simulator has to start printing out the spurious behaviors beginning with their initial states if it has a practical upper bound on its runtime.

1. Introduction

It is known [4] that sound and complete qualitative simulators do not exist; that is, there exist inputs which lead to *ineradicable* spurious behaviors, proving whose inconsistency is an undecidable task, and thus any sound qualitative simulator has to include them in its output. In this paper, we ask whether the next best thing, that is, a single sound qualitative simulator which detects and eliminates all *provably* inconsistent predictions, is possible, and obtain a negative answer. We prove that, for any sound qualitative simulator Q, which possesses two other reasonable properties that we define, there exists an input model which causes Q to predict a spurious prediction that can in fact be eliminated easily by many other qualitative simulators. Our result is a qualitative simulation version of Gödel's celebrated Incompleteness Theorem. We also show that, even when one restricts attention to models without self-reference, there exist infinitely many provably inconsistent inputs, which require so much time for a consistency check that such a simulator has to start printing out the spurious behaviors beginning with their initial states if it has a practical upper bound on its runtime.

2. Background

In the following, we make use of the terminology of QSIM [2], which is a state-of-the-art qualitative simulation methodology, although it should be noted that the results that we will be proving are valid for all reasoners whose input-output vocabularies are rich enough to support the representational techniques that will be used in our proofs.

This section starts by clarifying some of the additional terminology to be used in the rest of the paper. We then list a number of previously proven facts that will be utilized in our arguments.

2.1 Terminology

Qualitative simulator input: Qualitative simulators take a system model and a description of the initial system state as input. The model consists of one or more operating region descriptions and definitions of possible transitions between operating regions. Each operating region description contains variable-related definitions such as quantity spaces and legal ranges, and constraints that hold between the variables in that region. In this paper, the initial state description is always assumed to contain a complete assignment of qualitative values to all the variables of the initial operating region. When some control switches and parameters of the simulation need to be set to values other than their defaults (e.g. when the user wants QSIM to create no new landmarks for some variables during simulation,) the description of these settings is also part of the input. In the following discussion, the term *qualitative simulator input* denotes a single string encoding all the information mentioned above.

Soundness: A qualitative simulator is *sound* if it is guaranteed that, for any ODE and initial state that matches the simulator's input, there will be a behavior in its output which matches the ODE's solution. QSIM, for instance, is known to have the soundness property [2].

Completeness: A *complete* qualitative simulator would come with a guarantee that every behavior in its output corresponds to the solution of at least one ODE matching its input.

The output of a sound and complete qualitative simulator, if such a thing could exist, would thus contain a tree of qualitative states rooted at the initial state, such that all paths starting from the root and ending at a leaf (for finite branches) or containing an infinite sequence of states correspond to a solution of an ODE matching the input, and all such solutions would match such a path.

Consistent input: An input is *consistent* if and only if it could cause the prediction of at least one behavior on a hypothetical sound and complete qualitative simulator.

Note that good qualitative simulators are supposed to produce an empty tree in response to an inconsistent input.

We now define two more desirable properties for qualitative simulators, indeed, for almost any program.

Steadfastness: A *steadfast* qualitative simulator is one which does not retract any part of its output that it has already printed. In particular, once a steadfast qualitative simulator has printed the root of the behavior tree, corresponding to the initial system state, its output is guaranteed to contain at least one behavior prediction starting from that state.

The motivation behind our explicit definition of this very reasonable and easily realizable property is the interesting fact that implementations of QSIM which start printing out the behavior tree before the simulation is over, (this is inevitable for inputs that cause trees which are either infinite, or finite but so big that running the simulation to completion is not an option,) are not steadfast; since inconsistency can propagate backward from the leaves to the root, QSIM may decide to prune a branch of the behavior tree after adding arbitrarily many states to it [2]. This is a result of the rule which states that all states, except the quiescent states and the transition states (which satisfy the operating region transition or termination conditions), should have at least one consistent successor in order to be consistent. A state which has no consistent successor state is also inconsistent even if it passes all other filters. So a state which has been added to the behavior tree may be labeled much later as inconsistent, if all of its successor states have been labeled inconsistent. Therefore, there are inputs which QSIM may announce as inconsistent only after building and then destroying a large tree rooted at them. If the simulator does not keep such a tree in memory, but instead starts to print it out before the end of the simulation, the later announcement that the input was, after all, inconsistent constitutes a violation of steadfastness as defined above.

Responsiveness: A *responsive* qualitative simulator starts printing a nonempty output within a finite amount of time after it starts running.

Note that a responsive qualitative simulator should produce an output even if the input is inconsistent. In such a case, the simulator should print a statement to the effect that the simulation result is an empty tree.

A responsive and steadfast qualitative simulator announces its final verdict about the input (i.e. either reports an inconsistency or prints the initial state as the root of the behavior tree, meaning that it has deemed the input consistent) in finite time. In the discussion below, we refer to this announcement as the *response* of the qualitative simulator to its input.

2.2 Facts

2.2.1 Exact Representation of Integers in Qualitative Simulator Inputs

For any integer *z*, there exists a set of QSIM variable quantity spaces and constraints, from which *z*'s equality to a particular variable in that set can be unambiguously deduced [6]. This can be achieved easily by encoding the required value with addition and multiplication constraints. For example, if we want to express that a variable has value 5, then we can use following structure where all the variables are defined to be constant and ONE is initialized to a positive finite value:

ONE = ONE × ONE
TWO = ONE + ONE
THREE = TWO + ONE
FOUR = THREE + ONE
FIVE = FOUR + ONE

Here, it is obvious that ONE equals 1, and so FIVE is 5.

2.2.2 Computationally Universal Qualitative Simulators Exist

The unlimited register machine (URM), which is equivalent in power to the Turing machine (TM) model, is one of the many mathematical idealizations of computers [1]. A URM has finitely many registers which can store nonnegative integers. There is no upper limit for the value contained in a register. Every URM has a program which contains an ordered list of instructions (Table 1) to be performed on the registers. When an instruction (other than a *jump*) is executed, the next instruction to be executed is the one right after the current one. Table 2 contains the description of a simple URM, which gets two integers as input in registers 1 and 2, and gives the sum of these numbers as its output in register 1.

URM Instructions	
$inc(r_j)$	increments the value in register j
$zero(r_j)$	resets the value in register j to zero
$jump(r_j, r_k, i_m)$	If j is equal to k, jumps to instruction m, otherwise, the next instruction is executed
end	terminates the computation

Table 1: URM Instructions

i_1: zero(r_3)
i_2: jump(r_2,r_3,i_6)
i_3: inc(r_1)
i_4: inc(r_3)
i_5: jump(r_2,r_2,i_2)
i_6: end

Table 2: URM Program Computing $f(x, y) = x + y$

Yılmaz and Say proved [10] that any given URM/input pair can be simulated in a qualitative simulator which supports one of several quite restricted subsets of the input/output vocabulary of QSIM. To simulate a URM program with p instructions, one constructs a qualitative simulator input with $p+2$ operating regions: one for each instruction, one for the initialization, and one more for the finalization of the computation. In the qualitative simulation of the URM's computation process, each state of the behavior tree (except the root, which corresponds to the initialization,) corresponds to the execution of an instruction. This simulation can be performed in a behavior tree with a single branch. (Note that this requires some additional filters which "decode" the input to obtain and then keep track of the exact numerical values of the simulation variables to be incorporated to presently available qualitative simulators, and nobody has seriously tried to implement the construction in [10] to our knowledge. However, an

implementation is entirely possible, and in fact quite straightforward when compared with some of the mathematically much more sophisticated filters that have been developed for QSIM, e.g. [5, 7].)

In fact, such qualitative simulators can be thought of as an alternative computational model like the URM, and appropriately prepared qualitative simulator inputs play the roles of the programs to run on this computational model.

Note that qualitative simulators can be (and are) simulated by our computers; therefore they can be simulated in a TM, which is capable of doing everything which can be done by our computers [8]. It follows from the computational universality of URM's that any qualitative simulator can be simulated by a URM. As a result, a qualitative simulator which supports one of the subsets of the QSIM input vocabulary listed in [10] can simulate any other qualitative simulator implementation.

2.2.3 The Recursion Theorem

This theorem, [8] which is a well-known fact of computability theory, provides the following technique, which can be used when one needs to construct programs which can store their own code in a variable, and then process it as necessary: We construct a program which consists of three parts; *A*, *B*, and *Main*, which run in this order. When executed, part *A* stores the code of the other two parts, namely, a string of the form <*B*,*Main*>, into a variable *v*. Part *B* then starts running, and uses the string in *v* to construct the description of a partial program which stores the value that *B* sees in *v* into the variable *v*. Note that the partial program *B* prepares in this manner is *A* itself. *B* then appends <*A*>, which it has just constructed, with the current contents of *v*, stores the resulting longer string, which is none other than the code of our program itself, namely <*A*,*B*,*Main*>, in *v*, and passes control to part *Main*, which can use the program's code stored in *v* when needed. *Main* contains the rest of the code which makes the program accomplish whatever its designated task is; *A* and *B* are used just for implementing the recursion technique described above.

2.2.4 The Halting Problem Is Reducible to Hilbert's Tenth Problem

As the name suggests, Hilbert's Tenth Problem is the tenth of 23 problems which were announced in 1900 by the famous mathematician David Hilbert as a challenge to the mathematicians of the 20th century. It asks for an algorithm for deciding whether a given multivariate polynomial with integer coefficients has integer solutions. In 1970, Yuri V. Matiyasevich showed that no such algorithm exists, by demonstrating a method which can be used to construct, for any given Turing machine *T*, a polynomial *P* with integer coefficients, such that *P* has a solution in the natural numbers if and only if *T* halts on the empty input. As mentioned above, the original statement of the problem talks about the domain of integers, rather than natural numbers. However, this can be shown to be equivalent in difficulty to the version with the domain restricted to the natural numbers; see, for instance, [3].

2.2.5 Hilbert's Tenth Problem Is Reducible to Qualitative Simulator Input Consistency Checking

Yılmaz and Say have proven [10] that, even if the qualitative representation is narrowed so that only the *derivative*, *add, mult* and *constant* constraints can be used in

QDE's, and the simulation proceeds only in a single operating region, it is still impossible to build a sound and complete qualitative simulator based on this input-output vocabulary. This proof uses a reduction from Hilbert's Tenth Problem, namely, a technique that can be used to build, for any given multivariate polynomial *P* with integer coefficients, a qualitative simulator input *QI*, such that *QI* is consistent if and only if *P* has a solution in the integers. This means that a sound and complete qualitative simulator, if it existed, could be used to solve Hilbert's Tenth Problem. Although this proves that there can be no qualitative simulator which is both sound and complete, the transformation used for this purpose in [10] can also be used fruitfully to obtaining interesting results about sound, steadfast, responsive and naturally incomplete simulators, as will be seen in Section 3.2.

3. Every Sound, Steadfast and Responsive Qualitative Simulator Has a "Blind Spot"

We will now prove that every qualitative simulator which possesses the soundness, steadfastness, and responsiveness properties necessarily predicts a provably spurious behavior *B*, and that this same *B* can be recognized as spurious and filtered out easily by many other feasibly constructible qualitative simulators. Section 3.1 demonstrates this fact for qualitative simulators which support the operating region transition feature. In Section 3.2, we show that this feature is not required for the phenomenon we describe here to occur.

3.1.The Blind Spot Theorem: Multi-Region Version

We start by observing that qualitative simulator inputs can be designed to use a simple adaptation of the recursion technique of Section 2.2.3 to obtain and store their own code in a simulation variable. Such an input will consist of three submodels: *A*, *B*, *Main*. *A*, which consists of a single operating region, will contain a variable *V*, which it initializes to an integer encoding the string <*B*, *Main*>. Another variable in *A* is constrained to reach a landmark which will trigger a transition to the starting operating region of the multiple-region submodel *B*. Variable *V* inherits its value during all operating region transitions. *B* models a URM which uses its knowledge of the value in *V* to prepare the description of a qualitative input submodel, which models a URM that initializes variable *V* to the value *B* now sees in *V*, and then triggers a transition to the starting operating region of *B*. Note that this submodel description prepared by *B* is none other than <*A*>. *B* then combines <*A*> and <*B*, *Main*> to obtain <*A*, *B*, *Main*>, stores this value in *V*, and triggers a transition to *Main*, where the description of the entire input <*A*, *B*, *Main*> can be used as needed.

We now note that, given any qualitative simulator *C*, one can build a qualitative simulator input M_C as follows:

M_C contains the representation of a URM program. Upon starting execution, M_C first acquires its own code $<M_C>$ using the recursion technique described above, and then starts to simulate *C*, whose code has been embedded in that of the program of M_C, with $<M_C>$ as input. The simulation of *C* is performed until *C* gives its response about the

input, i.e. until C either declares inconsistency or prints the initial state as the root of the behavior tree. If C rejects the initial state of $<M_C>$, the program of M_C ends by arriving at an operating region where a variable increases until it reaches a bound of its legal range, constituting a successful termination of the corresponding branch of the behavior tree, meaning that $<M_C>$ was a consistent input. On the other hand, if C prints the initial state of its input, the program of M_C jumps to an instruction represented by the operating region OR_C, which causes a contradiction. This can be achieved by a variable, say, S, which is defined in all operating regions, and whose value is inherited in all operating region transitions. S is constrained to be constant in all operating regions and it is initialized to a positive finite value. In the operating region OR_C, S is constrained to be constant at zero. Therefore a transition into this region causes an inconsistent behavior.

Now let Q be any sound, steadfast and responsive qualitative simulator. We claim that the input M_Q is inconsistent, and yet Q does not reject this input; it starts printing a provably spurious prediction that begins with the initial state of M_Q. We justify this claim with the following analysis of the execution of Q on input M_Q:

To prevent confusion, let Q_0 denote the "outer" Q, and let Q_1 denote the "inner" Q, which will be simulated as described above by the program M_Q. Since Q_0 and Q_1 are implementations of the same qualitative simulator which are working on the same input (M_Q), their actions will be exactly the same.

There are two possibilities for the response of Q to the input M_Q: Q either rejects M_Q, or prints out the initial state of M_Q.

Let us first analyze the case where Q_0 rejects M_Q. Then Q_1 will also reject M_Q. But now consider what the program described by M_Q does: It simulates Q_1 for a finite number of steps to see how Q_1 responds to $<M_Q>$, and when it sees a rejection, it terminates successfully, without reaching a contradictory state. This is a perfectly valid behavior of the described system, and should of course be printed out by any sound qualitative simulator. Since Q is sound, we conclude from this argument that it cannot reject M_Q.

The remaining possibility is that both Q_0 and Q_1 will print out the initial state of M_Q. Since Q is steadfast, printing the initial state is an irreversible action, and means that Q announces the input M_Q to be consistent. Let us consider what the program of M_Q does in this case: It simulates Q_1 for a finite number of steps, and when it sees Q_1 print out the initial state of M_Q, it jumps to a contradictory operating region, making the branch of the behavior tree describing its entire execution a spurious one. Since the model is so constrained that no other nonspurious branches are possible, as explained in section 2.2.2, we conclude that M_Q is, after all, inconsistent. By the argument of the previous paragraph, Q must announce this inconsistent input to be consistent.

There exist other qualitative simulators which can correctly detect the inconsistency of this input and reject it: Consider, for example, a computationally universal version of QSIM, to which the "numerical" filters mentioned in section 2.2.2, that are required for the simulation of a URM to produce a single-branch behavior tree, have been incorporated. Such a simulator will start "running" the program of the input M_Q, which in turn will simulate Q on the input M_Q, see Q accept M_Q as proven above, and jump to the contradictory operating region, at which point the "outer" simulator will propagate the inconsistency all the way back to the initial state and reject the input M_Q. Interestingly, almost every sufficiently sophisticated qualitative simulator other than Q is capable of

rejecting M_Q in this manner. It is this fact which leads us to use the term "blind spot" in the title of this section.

As a somewhat frustrating thought exercise, one can show that some qualitative simulators can sometimes "understand" that their present input will cause a blind spot spurious prediction, but they just cannot announce it loud, so to speak: Assume that our sound, responsive and steadfast simulator *Q* has been written by someone who knows about the trick that we have been discussing above. The programmer has coded *Q* so that it obtains its own code using the recursion technique, and then uses this to construct the string $<M_Q>$, which will cause it so much trouble. *Q* can now compare its present input with $<M_Q>$, but even this capability does not save it: Even when *Q* "knows" that the input is $<M_Q>$, it cannot announce it to be inconsistent as proven above, and the only option available is to start print the spurious prediction.

The argument we use to prove the existence of blind spots in sound, steadfast and responsive qualitative simulators has been inspired by the proof of Gödel's incompleteness theorem, which states that a sound formal system of axioms and rules of inference cannot be complete if it satisfies some simple conditions. (A *sound* formal system is one in which one cannot prove a statement to be both true and false at the same time. In a *complete* formal system, every true statement is provable.) The key point of Gödel's proof is the sentence *T*="This sentence cannot be proven," which can be defined mathematically in any system *M* which satisfies the conditions. If this sentence *T* can be proven, then it is obvious that a false statement is provable; since *T* states that *T* itself cannot be proven. Since system *M* is sound, this is not a valid choice. The other possibility is the non-existence of a proof in system *M* for statement *T*. This means that *T* is true, and therefore *M* is incomplete. For more information on Gödel's proof, see [9]. The resemblance between the argument in this section and Gödel's proof is a result of self reference. In our proof, the input has to be consistent if the qualitative simulator rejects it, and in Gödel's proof, the statement has to be wrong if the system can be used to prove it.

3.2.The Blind Spot Theorem: Single-Region Version

In the proof of Section 3.1, the operating region transition feature of the QSIM vocabulary played a critical role, since it is due to that representational item that URM's can be modeled. In this section, we show that the problem demonstrated in that section persists even when the operating region transition feature is excluded from the qualitative simulation vocabulary.

Let *Q* be any sound, steadfast and responsive qualitative simulator which works with the restricted vocabulary described above. We will now demonstrate that there exists an input which is inconsistent but which is announced to be consistent by *Q*. For this purpose, we first construct a Turing machine named *T*.

T starts to simulate *Q*'s simulation of an input *M*, whose preparation will be described shortly. On the first response of *Q*, *T* stops the simulation. If *Q* rejects its input, *T* halts; otherwise, *T* loops forever.

T prepares the input *M*, which it feeds to *U*, as follows: *T* first obtains its own code $<T>$ by recursion, and then it sets up a polynomial D, which has a solution if and only if *T* halts with the empty string as input, (The details of this computation are explained in the next two paragraphs.) *T* then encodes D as a qualitative simulator input *M* using the

technique described in [10], (Section 2.2.5) setting the initial magnitudes of all the simulation variables representing the polynomial variables to (0, ∞).

The transformation used by *T* to encode its own halting status in a multivariate polynomial is realized in two stages. T first employs the techniques of [3] (Section 2.2.4) to produce a polynomial D_1 defined on the domain $\mathcal{N}$ (including zero). Since we will specify to the simulator that we are looking for a solution where all the polynomial variables are positive, as mentioned in the previous paragraph, what we really want here is a polynomial which has positive roots if *T* halts. So *T* transforms D_1 to another polynomial D in the following manner:

Given a polynomial $D_1(x_1, x_2 \cdots x_n)$, which is defined on $\mathcal{N}$, we will build a new polynomial D, which has a solution in the positive integers, if and only if $D_1(x_1, x_2 \cdots x_n)$ has a solution in the set of natural numbers. D is the product of all the variations of D_1. By a *variation* of D_1, we mean a polynomial which can be obtained by setting some of the variables of D_1 directly to zero. Since there are two possibilities (zero or not) for each variable, there are 2^n variations of D_1.

$$D = \prod_{i \in \{0 \cup 1\}^n} D_i, \text{ where } D_i = D_1(i_1 \times x_1, i_2 \times x_2 \ldots i_n \times x_n)$$

where i_j is the j^{th} character of the string i, i.e. 0 or 1.

Having described *T*, we immediately proceed to our proof. *Q* can either find the input *M* inconsistent, or can start to print out the initial state. Let us analyze these cases.

Assume that *Q* says that *M* is inconsistent, and that therefore *T* halts. But if *T* halts, then D has a solution, and *M* is consistent. So *Q* has incorrectly rejected a consistent behavior. Since *Q* is sound, this is impossible, so *Q* cannot reject *M*.

So *Q* accepts *M*. But then *T* is a TM that does not halt, meaning that D has no solution, and that *M* is inconsistent. So *Q* accepts an inconsistent model.

4. States Checkable with High Cost

In [10], it is shown that a qualitative simulator can simulate a URM. Now, we will use this fact to construct a qualitative simulator input whose consistency requires $\Theta(2^{n^k})$ time to be detected, where n is the size of the input and $k > 0$.

Consider any EXPTIME-complete language *A*. The fastest algorithm which decides *A* has superpolynomial time complexity, since all other languages in the class EXPTIME can be reduced to *A* in polynomial time, and it is known that $P \subset EXPTIME$ [8]. Since *A* is decidable, a URM which decides it exists, call this URM *U*.

We will use a modified version of the technique in [10], to encode *U* and its input as an input for a qualitative simulator *Q*. The only difference from [10] will be the number of finalization operating regions. We need two separate finalization operating regions. Since *Q* will simulate a decider, one of the regions will stand for *yes*, while the other will stand for *no*. The *no* operating region contains an inconsistency with regard to the other operating regions, so if the simulation reaches the *no* operating region, this will result in a spurious behavior, and since there will be at most one simulation branch, the input will be inconsistent in this case. The *yes* operating region does not contain an inconsistency, and

therefore in the case of reaching there, the simulation will output a single nonempty behavior successfully.

Now, let us construct a Turing machine *T* for deciding *A*. *T* reads its input string *x*, and uses the technique described in the previous paragraph to construct a qualitative simulator input *M*, which encodes the URM *U* working on input *x*, and then simulates *Q* on input *M* until *Q* gives its response to the initial state. If *Q* prints the initial state, this means that $x \in A$, and *T* prints *yes*; if *Q* rejects the input *M* due to inconsistency, this means $x \notin A$, and *T* prints *no*. (Note that this construction is guaranteed to be valid only if *Q* is steadfast.)

Let us calculate how fast the fastest possible qualitative simulator *Q* can respond to *M* in this scenario. Let the length of *x* be *n*. The length of the input *M* of the qualitative simulator is $\Theta(n)$, since the only operating region of *M* whose size depends on *n* is the starting region, where the value *x* is supposed to be encoded as the initial value of *U*'s first register, and this can be done using a set of constraints that can be expressed in $O(n)$ symbols. All the other operating regions have fixed lengths that do not depend on *x*. So $|M|$ is $\Theta(n)$. Now, we know that for some values of *x*, the fastest possible *T* will have to run for $\Theta(2^{n^k})$ steps. If one leaves the simulation of Q aside, it is clear that the remaining parts of T have a total runtime of $O(n)$. Since the total time is $\Theta(2^{n^k})$, this concludes that time required for Q should also be $\Theta(2^{n^k})$. Since $|M|$ is $\Theta(n)$, and n is $\Theta(|M|)$, Q is seen to require $\Theta(2^{\Theta(|M|)^k})$ steps, that is, an exponential amount of time in terms of the size of its own input, to decide about the consistency of its initial state.

If one thinks about QSIM (a version which has been augmented with the numerical filters to ensure a single branch while simulating the URM, and which has been guaranteed to act steadfastly, at least for the inputs it will encounter in this construction, by making sure that it starts printing the constructed state tree only when the simulation is over,) in this scenario, it is clear that the announcement of the verdict about the initial state will take $\Theta(2^{\Theta(|M|)^k})$ steps, since QSIM would construct the branch all the way to its end, and then, in case of a *no* answer, propagate the inconsistency all the way back to the initial state. The proof above shows that this runtime is the best that can be achieved by any qualitative simulator.

We conclude that, for any sound, responsive and steadfast qualitative simulator which has a practical (i.e. polynomial) upper bound on its runtime, there exist infinitely many provably inconsistent inputs, which require so much time for a consistency check that the simulator has to start printing out the spurious behaviors beginning with their initial states.

There exists an infinite hierarchy of languages which require worse and worse runtimes than those in EXPTIME [8]. All these can be used to demonstrate the existence of spurious behaviors which are eradicable in principle, but ineradicable in practice, by the same argument as above.

5. Conclusion

We proved that there is no single sound, responsive and steadfast qualitative simulator which can detect and eliminate all eradicable spurious predictions. Furthermore, when practical limits are imposed on the runtime, the set of spurious predictions that can be eliminated is a dramatically small subset of the set of all eradicable spurious predictions.

We acknowledge that the models involved in our arguments are not of the kind that would normally be submitted to a qualitative simulator by a sensible user. But getting rid of the occasionally predicted eradicable spurious behavior is a desirable thing for those normal users as well, and we hope that the findings reported here might be useful for researchers interested in constructing qualitative simulators with improved theoretical guarantees and additional filters of increasing mathematical sophistication.

References

[1] Nigel J. Cutland, 1980. Computability: An Introduction to Recursive Function Theory. Cambridge University Press.

[2] Benjamin J. Kuipers, 1994. Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge, MIT Press, Cambridge, MA.

[3] Yuri V. Matiyasevich, 1993. Hilbert's Tenth Problem. Cambridge, Mass.: The MIT Press.

[4] A. C. Cem Say and H. Levent Akın, 2003. Sound and complete qualitative simulation is impossible, Artificial Intelligence Vol. 149, pp. 251-266.

[5] A. C. Cem Say, 1998. L'Hôpital's filter for QSIM, IEEE Transactions on Pattern Analysis and Machine Intelligence Vol. 20, pp. 1-8.

[6] A. C. Cem Say, 1997. Numbers representable in pure QSIM, in: Proc. Eleventh International Workshop on Qualitative Reasoning, Cortona, Italy, pp. 337–344.

[7] A. C. Cem Say, Selahattin Kuru, 1993. Improved filtering for the QSIM algorithm, IEEE Transactions on Pattern Analysis and Machine Intelligence Vol. 15, pp. 967-971.

[8] Michael Sipser, 1997. Introduction to the Theory of Computation, PWS Publishing Company.

[9] Raymond M. Smullyan, 1992. Gödel's Incompleteness Theorems. Oxford University Press.

[10] Özgür Yılmaz and A. C. Cem Say, 2006. Causes of ineradicable spurious predictions in qualitative simulation, Journal of Artificial Intelligence Research Vol.27, pp. 551-575.