Causes of Ineradicable Spurious Predictions in Qualitative Simulation

Özgür Yılmaz and A. C. Cem Say
Boğaziçi University
Department of Computer Engineering
Bebek, 34342, İstanbul, Turkey
yilmozgu@boun.edu.tr, say@boun.edu.tr

Abstract

It has recently been proven that it is impossible to build a sound and complete qualitative simulator using the QSIM representation for input and output. We provide an alternative proof which employs a smaller subset of the QSIM vocabulary, and show that the problem persists for several "weakened" versions of the representation. For this purpose, we demonstrate a method for modeling and simulating an arbitrary Unlimited Register Machine using QSIM, and thereby establish that QSIM has universal computational power. Our findings may be helpful for researchers interested in constructing provably sound and complete qualitative simulators using weaker representations.

1 Introduction

State-of-the-art qualitative simulators [Weld and de Kleer, 1990; Forbus, 1990; Kuipers, 1994] are known to be sound[1]; no trajectory which is the solution of a concrete equation matching the input can be missing from the output. However, it has recently been proven [Say and Akın, 2003] that it is impossible to provide the additional guarantee of completeness that such a simulator will never produce a spurious prediction for any input: For any sound qualitative simulator using the input-output representation and task specification of the QSIM [Kuipers, 1994] methodology, there exist input models and initial states whose simulation output will contain "behaviors" that do not correspond to any possible solution of the input equations.

The proof in [Say and Akın, 2003] shows that a "sound and complete" qualitative simulator employing the vocabulary mentioned above, if it existed, could be used to solve any given instance of Hilbert's Tenth Problem, which is famously undecidable [Matiyasevich, 1993]. The procedure involves building a QSIM model representing the given problem, simulating it several times starting from carefully constructed initial states representing candidate solutions, and examining the output to read out the solution.

It is important to note that this proof does not necessarily mean that all hope of constructing a provably sound and complete qualitative simulator is completely lost. One may try to "weaken" the input-output representation so that it no longer possesses the problematic power which enables one to unambiguously encode instances of Hilbert's Tenth Problem into a QSIM model. (Of course, this weakening must be kept at the minimum possible level for the resulting program to be a useful reasoner; for instance, removing the program's ability to distinguish between negative and non-negative numbers would possibly yield a sound and complete simulator, but the output of that program would just state that "everything is possible" and this is not what we want from these methods.) This is why one should examine the incompleteness proof in [Say and Akın, 2003] to see exactly which features of the QSIM representation are used in the construction of the reduction; any future qualitative simulator supporting the same vocabulary subset would be incorporating the same problem from the start.

Here is a listing of the QSIM representational items used in that proof: Of the several qualitative constraint types available in the vocabulary [Kuipers, 1994], only the monotonic increasing function (M^+), derivative (d/dt), multiplication (*mult*) and *constant* constraints are utilized. (Note the absence of the *add* constraint, which can be "implemented" using the others, in this list.) Qualitative interval magnitudes like $(0, \infty)$, with what one might call "infinite uncertainty" about the actual value of the represented number, are used for initializing several variables and form an essential part of the argument. QSIM's ability to explicitly represent infinite limits is utilized for equating a landmark to the number π, by stating that it is twice the limit of the function *arctan x* as x nears infinity. Finally, the operating region transition feature is used heavily, since it is thanks to this characteristic that the sine function can be represented in the qualitative vocabulary.

In this paper, we consider several alternative subsets of the representation, and show that the ineradicable spurious prediction problem persists even when only the *add* and *constant* constraints are allowed, and infinite landmarks are banished. If one allows the *mult* constraint as well, then the resulting qualitative simulator is inherently incomplete even when the representation of negative numbers is forbidden and every variable is forced to be specified with zero uncer-

[1] The terms *sound* and *complete* are used in the same sense as in [Kuipers, 1994] throughout the paper.

tainty (i.e. as a single unambiguous real number) in the initial state.

The rest of the paper is structured as follows: In Section 2, we clarify what one means when one talks about a "sound and complete" qualitative simulator. Section 3 describes the Turing-equivalent abstract automata called Unlimited Register Machines (URM) used in our proof of incompleteness. Section 4 contains the main results of this paper, whereas Section 5 is a conclusion.

2 Desiderata for a sound and complete qualitative simulator

It is important at this point to clarify exactly what one would expect from a hypothetical "sound and complete" qualitative simulator. If the input model yields a finite behavior tree of genuine solutions, it is obvious that the program is supposed to print the descriptions of the behaviors forming the branches of this tree, and nothing else, in finite time. If the input model and initial state are inconsistent, i.e., the "correct" output is the empty tree, the program should report this inconsistency in finite time.

Finally, if the input yields a behavior tree with infinitely many branches, (QSIM's ability of introducing new landmarks during the simulation makes this possible) the program is supposed to run forever, adding a new state to its output every once in a while. More formally, for every positive *i*, there has to be an integer *s* such that the program will have printed out the "first" *i* states of the behavior tree (according to some ordering where the root, i.e. the initial state, is state number 1, and no descendants of any particular state are printed before that state itself) at the end of the s^{th} step in its execution. Note that these requirements mean that a sound and complete simulator would have to be able to decide whether the initial system state description given to it is consistent with the input model or not within a finite time. This necessity is used in the proof of incompleteness in Section 4.

3 Unlimited register machines

The easiest way of thinking about a URM is to see it as a computer with infinite memory which supports a particularly simple programming language. A URM [Cutland, 1980] program P consists of a finite sequence of instructions I_1, I_2, ..., $I_{|P|}$. The instructions may refer to the machine's registers R_i, each of which can store an arbitrarily big natural number. We use R_1, R_2, ... to refer to URM registers, and r_1, r_2, r_3 ... for the register contents.

There are four types of URM instructions:

succ(*n*): Increment the content of register *n* by one.

$$R_n \leftarrow r_n + 1$$

zero(*n*): Set the content of register *n* to zero.

$$R_n \leftarrow 0$$

jump(*m, n, q*): Compare registers *m* and *n*. If they are equal, continue with instruction *q*.

$$If\ r_m = r_n\ then\ jump\ to\ I_q$$

transfer(*m, n*): Transfer the contents of R_m to R_n. Only R_n is modified.

$$R_n \leftarrow r_m$$

A URM program starts execution with the first instruction. If the current instruction is not a *jump* whose equality condition is satisfied, it is followed by the next instruction in the list. The program ends if it attempts to continue beyond the last instruction, or if a *jump* to a nonexistent address is attempted.

If $P = I_1, ..., I_{|P|}$ is a URM program, it computes a function $P^{(k)} : N^k \rightarrow N$. $P^{(k)}(a_1, ..., a_k)$ is computed as follows:

- Initialization: Store $a_1, ..., a_k$ in registers $R_1, ..., R_k$, respectively, and set all other registers referenced in the program to 0.
- Iteration: Starting with I_1, execute the instructions in the order described above.
- Output: If the program ends, then the computed value of the function is the number r_1 contained in register R_1. If the program never stops, then $P^{(k)}(a_1, ..., a_k)$ is undefined.

Table 1 contains an example of a URM program which computes the function $f(x, y) = x + y$. Note that the function is from N^2 to N, where the input values x and y are stored in registers R_1 and R_2, and the output of the function is expected to be stored in R_1 at the end of the program.

I_1: *zero*(3)
I_2: *jump*(2, 3, 6)
I_3: *succ*(1)
I_4: *succ*(3)
I_5: *jump*(1, 1, 2)

TABLE 1. URM program computing $f(x, y) = x + y$

The program first sets R_3 to zero. It checks to see if $R_3 = R_2$ (in the case that $y = 0$). Otherwise, it increments both R_1 and R_3. This continues until x has been incremented y times, and the value in R_1 is returned.

The URM model of computation is equivalent to the numerous alternative models such as the Turing machine model, the Gödel-Kleene partial recursive functions model and Church's lambda calculus [Cutland, 1980; Shepherdson and Sturgis, 1963] in the sense that the set of functions computable by URM's is identical to the set of the functions that can be computed by any other model. This means that a model which can simulate any given URM is as powerful as a Turing machine, since it can simulate any given Turing machine. In our new proof of QSIM incompleteness in the next section, we will make use of the fact that the halting problem for URM's is undecidable. [Cutland, 1980]

4 New incompleteness results for qualitative simulators

All the incompleteness results about new subsets of the QSIM vocabulary that are presented in this paper are based on the following theorem, which shows that QSIM can simulate any URM, and thereby has Turing-equivalent computational power.

Theorem 1: For any URM program P with |P| instructions, there exists a QSIM model QP with |P|+2 operating regions, which simulates it.

Proof: The proof will be by construction. Suppose we are given a URM program P with instructions I_1, ..., $I_{|P|}$. Let R_1, ..., R_N be the registers mentioned in the instructions of P. Now define your QSIM variables as follows:

For any R_i in P, define a QSIM variable NR_i which will represent it. Define *U*, *V*, *Z*, and *X*, which will serve as auxiliary variables. *U*'s legal range is the interval (0, *one*), where *one* is a landmark equal to 1. (Exact representation of any integer is possible in QSIM using a collection of *add*, *constant* and *mult* constraints. [Say and Akın, 2003]) *V* is the derivative of *U* and is a finite positive constant in every operating region. *Z* is constant at zero in every operating region. So QP has a total of N+4 variables.

Our QSIM model will have |P|+2 operating regions: Each instruction I_i of P will have a corresponding operating region named $OpReg_i$. The two remaining regions are $OpReg_0$, corresponding to the "initialization" stage of P, and $OpReg_{|P|+1}$, corresponding to its end.

The specification of each operating region must contain the constraints that are valid in that region, the boolean conditions (composed of primitives of the form *Variable=<qualitative magnitude, qualitative direction>*) which would trigger transitions to other operating regions when they are obtained, and lists that detail which variables inherit their previous magnitudes after such a transition, and which of them are initialized to new values during that switch. Tables 2-8 describe how to prepare these items for the operating regions in our target model, based on the program P. There are six different operating region templates (or "types") used in the construction; one for each URM instruction type, one for $OpReg_0$, and one for $OpReg_{|P|+1}$.

The model of $OpReg_0$ is depicted in Table 2. This is where our simulation of P will start. All the NR_i variables are supposed to be set to landmarks equated to their proper initial values specified by the "user" of P in the initial state. *U* is supposed to be initialized to (0, *inc*) in the initial state. Since *V* is always positive, QSIM will compute a single qualitative behavior segment, which ends with a transition to $OpReg_1$ when *U* reaches (*one*, *inc*) at time-point t_1 for this region.

As seen in Tables 2-8, exactly which variables keep their values during a transition depends on the type of the target operating region. Regions corresponding to instructions of the type *zero*(*n*) and *transfer*(*m, n*) should not inherit the value of R_n from their predecessors, since they involve the replacement of that value by another one anyway. All other types of regions, including the *succ*(*n*) type, inherit all the register contents from their predecessors. (Although the value of R_n *does* change in a *succ* instruction, the new value depends on the old one, unlike the cases of *zero*(*n*) and *transfer*(*m, n*). The corresponding QSIM variable NR_n increases continuously during the simulation of a region of type *succ*(*n*), and a new region transition occurs exactly at the moment when it has increased by one unit).

Operating Region:	$OpReg_0$
{Type: *Initialization***}**	
Constraint Set:	*constant*(*V*)
	constant(*Z*)
	constant(*X*)
	constant(NR_i) (*for all* $i \in \{1, ..., N\}$)
	d/dt(*U, V*)
Possible Transition:	
Trigger: ($U = (one, inc)$)	
New Operating Region: $OpReg_1$	
Variables inheriting qualitative magnitudes:	
See Table 3, indexed by the type of $OpReg_1$	
Variables with new asserted values:	$U \leftarrow (0, inc)$

TABLE 2. Model of the operating region $OpReg_0$, corresponding to the initialization of the URM

Type of target operating region: *succ*(*n*) *OR jump*(*m, n, q*)
Variables inheriting qualitative magnitudes:
NR_i *for all* $i \in \{1,..., N\}$, *V, Z*

Type of target operating region: *End*
Variables inheriting qualitative magnitudes:
NR_i *for all* $i \in \{1,..., N\}$, *V, Z, X*

Type of target operating region: *zero*(*n*) *OR transfer*(*m, n*)
Variables inheriting qualitative magnitudes:
NR_i *for all* $i \in \{1,..., N\}$-{*n*}, *V, Z, X*

TABLE 3. Variables which should inherit magnitudes according to type of the target operating region

The simulation of the given URM program proceeds as follows: As described in the previous section, the URM starts with an initial configuration, where the registers R_1, ..., R_k store the nonnegative integers a_1, ..., a_k, which form the input of the program, respectively. The other N-k registers are set to 0. Correspondingly our QSIM program has for each of the first k NR_i variables the quantity spaces (0, l_i, ∞), if the corresponding input a_i is nonzero, where the landmarks l_i's are equated to the natural numbers a_i. (The additional auxiliary variables and constraints necessary for the unambiguous expression of these numbers are supposed to be included in $OpReg_0$, in addition to what is presented in Table 2. All these additional variables are inherited and held constant in all operating regions.) These NR_i variables with nonzero initial values start on the landmarks, with qualitative values (l_i, *std*), whereas those with zero initial values

(NR_i, $i \in \{1, ..., k\}$ s.t. $a_i=0$) and the remaining NR_i variables $i \in \{k+1, ..., N\}$ have quantity spaces $(0, \infty)$ and start on (0, *std*). The variable X has the quantity space $(-\infty, 0, \infty)$ and starts initially at (0, *std*). The quantity space of the variable U is (0, *one*) where the landmark *one* is equated to 1, as mentioned above. U starts initially at qualitative value (0, *inc*).The derivative of U, V, has as quantity space (0, *speed*, ∞), where *speed* is also equated to 1. It starts at qualitative value (*speed*, *std*) and is constant in the whole simulation.

Note that all variables start the simulation at landmarks, whose values are given with zero initial uncertainty. □

Operating Region:	$OpReg_i$
{Type: *zero*(n)**}**	
Constraint Set:	*add*(Z, Z, NR_n)
	constant(V)
	constant(Z)
	constant(X)
	constant(NR_i) (*for all* $i \in \{1,...,N\}$)
	d/dt(U, V)
Possible Transition:	
Trigger:	(U = (*one*, *inc*))
New Operating Region:	$OpReg_{i+1}$
Variables inheriting qualitative magnitudes:	
See Table 3, indexed by the type of $OpReg_{i+1}$	
Variables with new asserted values:	$U \leftarrow$ (0, *inc*)

TABLE 4. Model template for operating regions corresponding to *zero*(n) instructions of the URM

Operating Region:	$OpReg_i$
{Type: *succ*(n)**}**	
Constraint Set:	*add*(X, U, NR_n)
	constant(V)
	constant(Z)
	constant(X)
	constant(NR_i) (*for all* $i \in \{1,...,N\}$-{n})
	d/dt(U, V)
Possible Transition:	
Trigger:	(U = (*one*, *inc*))
New Operating Region:	$OpReg_{i+1}$
Variables inheriting qualitative magnitudes:	
See Table 3, indexed by the type of $OpReg_{i+1}$	
Variables with new asserted values:	$U \leftarrow$ (0, *inc*)

TABLE 5. Model template for operating regions corresponding to *succ*(n) instructions of the URM

Our model is so constrained that a sound and complete qualitative simulator is guaranteed to produce exactly one behavior prediction for any initial state corresponding to a valid URM input. To see this, it is sufficient to observe that, at any step of the simulation, there is sufficient information available to the simulator to compute the exact numerical value of every variable in the model. (This just corresponds to "tracing" the URM program and keeping note of the register contents up to that step.) If the modeled URM halts on the particular input given in the initial state, the QSIM behavior is supposed to be a finite one, ending when the variable U attempts to exceed *one* in $OpReg_{|P|+1}$. If the URM computation does not halt, then the QSIM behavior is supposed to be a single infinite sequence of states, which never visits $OpReg_{|P|+1}$. We are now ready to state the new version of the incompleteness theorem.

Operating Region:	$OpReg_i$
{Type: *transfer*(m, n)**}**	
Constraint Set:	*add*(NR_m, Z, NR_n)
	constant(V)
	constant(Z)
	constant(X)
	constant(NR_i) (*for all* $i \in \{1,...,N\}$)
	d/dt(U, V)
Possible Transition:	
Trigger:	(U = (*one*, *inc*))
New Operating Region:	$OpReg_{i+1}$
Variables inheriting qualitative magnitudes:	
See Table 3, indexed by the type of $OpReg_{i+1}$	
Variables with new asserted values:	$U \leftarrow$ (0, *inc*)

TABLE 6. Model template for operating regions corresponding to *transfer*(m, n) instructions

Operating Region:	$OpReg_i$
{Type: *jump*(m, n, q)**}**	
Constraint Set:	*add*(NR_m, X, NR_n)
	constant(V)
	constant(Z)
	constant(NR_i) (*for all* $i \in \{1, ...,N\}$)
	d/dt(U, V)
Possible Transition:	
Trigger:	(U = (*one*, *inc*)) *AND* ($X \neq$ (0, *std*))
New Operating Region:	$OpReg_{i+1}$
Variables inheriting qualitative magnitudes:	
See Table 3, indexed by the type of $OpReg_{i+1}$	
Variables with new asserted values:	$U \leftarrow$ (0, *inc*)
Possible Transition:	
Trigger:	(U = (*one*, *inc*)) *AND* (X = (0,*std*))
New Operating Region:	$OpReg_q$
Variables inheriting qualitative magnitudes:	
See Table 3, indexed by the type of $OpReg_q$	
Variables with new asserted values:	$U \leftarrow$ (0, *inc*)

TABLE 7. Model template for operating regions corresponding to *jump*(m, n, q) instructions

Operating Region:	$OpReg_{|P|+1}$
{Type: *End***}**	
Constraint Set:	*constant*(V)
	constant(Z)
	constant(X)
	constant(NR_i) (*for all* $i \in \{1,...,N\}$)
	d/dt(U, V)

TABLE 8. Model of the operating region $OpReg_{|P|+1}$, corresponding to the end of the URM program

Theorem 2: Even if the qualitative representation is narrowed so that only the *d/dt*, *add*, *mult*, and *constant* constraints can be used in QDE's, and each variable is forced to start at a finite landmark whose value is given with zero uncertainty in the initial state, it is still impossible to build a sound and complete qualitative simulator based on this input-output vocabulary.

Proof: Assume, for the sake of the contradiction, that such a sound and complete simulator exists. We know show how to solve the halting problem for URM's using that algorithm as a subroutine.

Construct the corresponding QSIM model as described in Theorem 1 for the URM program P whose halting status on a particular input is supposed to be decided. Now define a new variable S with quantity space (0, *one*, ∞), where the landmark *one* is equated to the number 1. S starts at the value (*one*, *std*) in the initial state. Add the constraint *constant*(*S*) to all the operating regions, and specify that the value of S is inherited in all possible transitions. Insert the new constraint *add*(*Z*, *Z*, *S*) in $OpReg_{|P|+1}$. Consider what the simulator is supposed to do when checking the initial state for consistency. Note that we would have an inconsistency if the simulation ever enters $OpReg_{|P|+1}$, since the *add* constraint that we inserted to that region implies that S is zero, which would contradict with the inherited value of *one*. So a simulator which is supposed not to make any spurious predictions is expected to reject the initial state at time t_0 as inconsistent, if the simulation is going to enter $OpReg_{|P|+1}$, in other words, if the URM program under consideration is going to halt. If this sound and complete simulator does not reject the initial state due to inconsistency, but goes on with the simulation, then we can conclude that the program P will not halt. This forms a decision procedure for the halting problem. Since the halting problem is undecidable, a sound and complete simulator using this representation can not exist. □

It is in fact possible to remove the derivative constraint (which is only used in our proof to ensure that the behavior tree has at most one branch) from the representation as well, and the incompleteness result shown above would still stand:

Theorem 3: Even if the qualitative representation is narrowed so that only the *add*, *mult*, and *constant* constraints can be used in QDE's, and each variable is forced to start at a finite landmark whose value is given with zero uncertainty in the initial state, it is still impossible to build a sound and complete qualitative simulator based on this input-output vocabulary.

Proof: We will make a minor modification to the proof of Theorem 2. We observe that in the construction of Theorem 1, *U* always starts every operating region at (0, *inc*) and the fact that its derivative is a positive constant forces it to reach the value (*one*, *inc*) in the next time point. Then the transition to next operating region occurs, and *U* again receives the value (0, *inc*). What happens if we remove the variable *V* and all *d/dt* constraints from the model? In this case, since *U*'s derivative is not fixed, there are three possible future states for *U*: (*one*, *inc*), (*one*, *std*), and ((0, *one*), *std*). We fix this problem by inserting another possible region transition specification to all of our regions, except $OpReg_{|P|+1}$. This transition will be triggered when *U* has one of the values (*one*, *std*), and ((0, *one*), *std*), and its target will be $OpReg_{|P|+1}$. The variable S from the proof of Theorem 2, as well as all other variables, are inherited completely during this transition. So all the "unwanted" behaviors which would be created due to the elimination of *U*'s derivative end up in $OpReg_{|P|+1}$, and should therefore be rejected as spurious in accordance with the argument of the previous proof. Hence, once again, the simulator is supposed to accept the initial state as consistent if and only if P does not halt, meaning that a sound and complete simulation is impossible with this representation as well. □

Operating Region:	$OpReg_i$
{Type:	*jump*(*m*, *n*, *q*)**}**
Constraint Set:	*add*(NR_m, *O*, *C*)
	add(NR_n, *O*, *Y*)
	mult(*X*, *C*, *Y*)
	constant(*O*)
	constant(*C*)
	constant(*Y*)
	constant(*V*)
	constant(*Z*)
	constant(NR_i) (*for all* $i \in \{1,...,N\}$)
	d/dt(*U*, *V*)
Possible Transition:	
Trigger:	($U = (one, inc)$) *AND* ($X \neq (one, std)$)
New Operating Region:	$OpReg_{i+1}$
Variables inheriting qualitative magnitudes:	
Depends on the type of $OpReg_{i+1}$	
Variables with new asserted values:	$U \leftarrow (0, inc)$
Possible Transition:	
Trigger:	($U = (one, inc)$) *AND* ($X = (one, std)$)
New Operating Region:	$OpReg_q$
Variables inheriting qualitative magnitudes:	
Depends on the type of $OpReg_q$	
Variables with new asserted values:	$U \leftarrow (0, inc)$

TABLE 9. Alternative model template for operating regions corresponding to *jump*(*m*, *n*, *q*) instructions which avoids negative numbers

Interestingly, one can even restrict the representation so that only nonnegative numbers are supported, and the incompleteness result we proved above still stands:

Theorem 4: Even if the qualitative representation is narrowed so that only the *add*, *mult*, and *constant* constraints can be used in QDE's, each variable is forced to start at a finite landmark whose value is given with zero uncertainty in the initial state, and no variable is allowed to have a negative value at any time during the simulation, it is still impos-

sible to build a sound and complete qualitative simulator based on this input-output vocabulary.

Proof: In our previous proof, only variable *X* ever has the possibility of receiving a negative value, and that occurs only in a *jump* region. We replace Table 7 with Table 9 and introduce the new variables *O*, *C*, and *Y* .To all operating regions we set the constraint that these variables are constant. The variable *O* has quantity space (0, *one*) and initially starts with qualitative value (*one*, *std*), where the landmark *one* is set to numerical value 1. *O* is inherited by all possible transitions. The remaining variables *C* and *Y* are also inherited by all transitions, except when the target region is of type *jump*. The reason for that is similar to other inheritance mechanisms, i.e. we want these variables to take new magnitudes appropriate for our simulation in a *jump* region. As can be seen in Table 9, *X* receives the value 1, if and only if the two compared register values are equal. If they are unequal, *X* has a positive value different than 1. Therefore *X*'s legal range can be perfectly defined as (0, *one*, ∞), where *one* is equated to 1 and no variable ever gets a negative value during the simulation. □

Alternatively, we can keep negative numbers and remove the *mult* constraint from the representation, if we drop the requirement that each variable starts simulation at a value with zero uncertainty.

Theorem 5: Even if the qualitative representation is narrowed so that only the *add* and *constant* constraints can be used in QDE's, and each variable is forced to start at a finite landmark in the initial state, it is still impossible to build a sound and complete qualitative simulator based on this input-output vocabulary.

Proof: We used the *mult* constraint in the proofs of Theorems 1-3 only for equating landmark values to unambiguous integers. Assume that we delete the *mult* constraints from our model of Theorem 3. We introduce a new variable called A, which is constant at a positive landmark named *unit*. For any given natural number *n*, it is possible to introduce a landmark p_n to any desired QSIM variable, such that p_n's equality to *n*unit* can be unambigously deduced. As an example, Table 10 illustrates how the landmark p_6 of variable NR_8 is equated to 6**unit*. Note that we only use *constant* and *add* constraints (and a lot of auxiliary variables) for this purpose.

So for those of the k registers whose corresponding initial values (a_i's) are nonzero, the corresponding initial landmarks, l_i's, can be set such that $l_i = a_i$ **unit*. The landmark *one* in *U*'s quantity space is renamed as *unit*. In this new model, execution of a *succ*(*n*) instruction increments R_n's value by one *unit*. The *jump* instruction compares landmarks whose values equal *u*unit* and *v*unit* instead of comparing two landmarks whose values equal the natural numbers *u* and *v*. The same reasoning applies for the *transfer* instruction, where, instead of the value *u*, *u*unit* is transferred to the target register. The *zero* instruction sets the target register to 0, as in the previous construction. So the modeled machine does just what the original URM does, since the multiplication of all values by the coefficient *unit* does not change the flow of the program, and, in particular, whether it halts on its input or not. The rest of the argument is identical to that of the proof of Theorem 3. □

CONSTRAINTS	CORRESPONDENCES	MEANING
A = unit		
add(*A*, *A*, *B*)	*unit* + *unit* = p_2	p_2 = 2 * *unit*
add(*B*, *A*, *C*)	p_2 + *unit* = p_3	p_3 = 3 * *unit*
add(*C*, *A*, *D*)	p_3 + *unit* = p_4	p_4 = 4 * *unit*
add(*D*, *A*, *E*)	p_4 + *unit* = p_5	p_5 = 5 * *unit*
add(*E*, *A*, NR_8)	p_5 + *unit* = p_6	p_6 = 6 * *unit*

TABLE 10. QSIM constraint set built for expressing the equality "p_6 = 6 * *unit*"

As a final remark, note that if one has access to a semi-quantitative simulator [Kuipers, 1994] where it is possible for the user to specify bounding numerical intervals for the landmark values, one can use our construction of Theorem 1, and the capability of semi-quantitative simulators to calculate such intervals for the landmarks that are introduced during the simulation, to employ this simulator for running an arbitrary URM and reading out its numerical output; which underlines the computational universality of the QSIM engine. (Note that the incompleteness results proven above apply automatically to semi-quantitative simulators, whose representations are an extension of that of pure QSIM.)

5 Conclusion

In this paper, we considered several alternative subsets of the qualitative representation, and showed that the ineradicable spurious prediction problem persists even when only the *add* and *constant* constraints are allowed, and infinite landmarks are banished. If one allows the *mult* constraint as well, then the resulting qualitative simulator is inherently incomplete even when the representation of negative numbers is forbidden and every variable is forced to be specified with zero uncertainty (i.e. as a single unambiguous real number) in the initial state. Our proof relies on showing that the QSIM engine is Turing equivalent, and this big computational power brings with it the undecidability problems famously associated with universal computational mechanisms (like whether a Turing machine will enter a certain state). Hence, from another point of view, this power can be interpreted as a cause of spurious behaviours. A computability tool, which has universal computation power, can hardly be expected to predict the future states in a complete manner, unless of course its representation power is so weakened to remove it. Note that both the construction of [Say

and Akın, 2003] and the alternative proof presented here make heavy use of transitions between multiple operating regions in the QSIM model, and it is an open problem to determine whether sound and complete qualitative simulation would be possible if the representation was weakened so that only a single operating region was allowed in the models.

References

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

[Forbus, 1990] K. D. Forbus. The Qualitative Process Engine, In D. S. Weld and J. de Kleer, eds. *Readings in Qualitative Reasoning About Physical Systems.* San Mateo, California: Morgan Kaufmann, 220-235, 1990.

[Kuipers, 1994] B. J. Kuipers. *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*. Cambridge, Mass.: The MIT Press, 1994.

[Matiyasevich, 1993] Y. Matiyasevich. *Hilbert's Tenth Problem*. Cambridge, Mass.: The MIT Press, 1993.

[Say and Akın, 2003] A. C. C. Say, H.L.Akın. Sound and complete qualitative simulation is impossible. *Artificial Intelligence* 149: 251-266, 2003.

[Shepherdson and Sturgis, 1963] J. C. Shepherdson and H. E. Sturgis. Computability of Recursive Functions. *Journal for the Association for Computing Machinery* 10: 217-255, 1963.

[Weld and de Kleer, 1990] D. S. Weld and J. de Kleer. *Readings in Qualitative Reasoning About Physical Systems.* San Mateo, California: Morgan Kaufmann, 1990.