COMPUTABILITY-THEORETIC LIMITATIONS OF QUALITATIVE SIMULATION

by

Özgür Yılmaz

B.S. in Computer Engineering, Boğaziçi University, 2003

B.S. in Mathematics, Boğaziçi University, 2003

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2005

ACKNOWLEDGEMENTS

I wish to express my appreciation to my thesis supervisor, Prof. A. C. Cem Say for his enlightening and insightful advice throughout this thesis. The most important ideas of this work came out in the joyful brainstorming sessions we held together.

I am indebted to all the teachers passed through my life, who have equipped me with all the necessary knowledge to complete this study, especially to those of the Computer Engineering and Mathematics departments of Boğaziçi University and to those of the Austrian College.

Also many thanks to all the assistants of Computer Engineering Department and all my friends, who have provided an amusing and motivating environment during my research. They have never hesitated to help.

Finally, I would like to thank my family for the love and support they have given to me in all phases of my life.

This thesis is dedicated to those who like to teach and to learn.

ABSTRACT

COMPUTABILITY-THEORETIC LIMITATIONS OF QUALITATIVE SIMULATION

Qualitative reasoning and simulation are useful mathematical tools, especially for the analysis, design and diagnosis of dynamic systems. Simulators which are seen to be incomplete, that is, which produce spurious predictions for a particular input, are usually augmented with additional filters eliminating that behaviour. Kuipers introduced a simulator (QSIM) which has the soundness property, that is, no trajectory which is the solution of a concrete equation matching the input can be missing from the output. It has been proven that there does not exist a sound and complete simulator whose input and output vocabularies are identical to those of the "pure" QSIM.

This thesis contains a series of further results about computability-theoretic limitations of qualitative simulation. Firstly, it demonstrates a method for modeling and simulating an arbitrary Unlimited Register Machine (URM) using QSIM, and thereby establishes that qualitative simulation has universal computational power. By making use of reductions from the famous undecidable Halting Problem for computation tools possessing universal computational power, it proves that it is impossible to build a sound and complete qualitative simulator using the QSIM representation for input and output for several "weakened" versions of the representation. It finishes with an ultimate result that achieving a sound and complete simulator, which operates only in a single operating region in which continuity rules are fully obeyed, is impossible,.

The results in this thesis are demonstrated using the QSIM representation for input and output, and are valid for all qualitative simulators whose input and output vocabularies are identical to that of QSIM. They are important in the sense that they provide deeper insight to the causes of spurious predictions, and they are also interesting for researchers aiming to construct provably sound and complete simulators using weaker representations.

ÖZET

NİTEL BENZETİMİN HESAPLANABİLİRLİK KURAMINA DAYALI SINIRLARI

Nitel uslamlama ve benzetim, özellikle dinamik sistemlerin analizi, tasarımı ve tanısı için faydalı matematiksel araçlardır. Eksik, yani belirli bir girdi için yanlış tahmin üreten benzeticilere, genellikle o davranışı elimine eden ilave süzgeçler eklenir. Kuipers tutarlılık özelliğini içeren, bir başka deyişle, çıktısında, girdiye uyan nicel bir denklemin sonucu olan hiçbir davranışı kaçırmayan bir benzetici (QSIM) sunmuştur. Girdi ve çıktı alfabesi "saf" QSIM ile aynı olan tutarlı ve tam bir benzeticinin varolmadığı ispatlanmıştır.

Bu tez, nitel benzetimin hesaplanabilirlik kuramına dayalı sınırları üzerine bir seri ilave sonuçlar içermektedir. İlk olarak, herhangi bir sınırsız yazmaçlı makinenin modellemesi ve benzetimi için bir method ortaya koymakta ve bununla nitel benzetimin evrensel hesaplama gücüne sahip olduğunu tesis etmektedir. Evrensel hesaplama gücüne sahip olan hesaplama araçlarının meşhur kararlaştırılamaz Durma Problemi'nden indirgeme yardımıyla, girdi ve çıktı da QSIM alfabesi kullanmak suretiyle, girdi ve çıktı alfabesinin pek çok "daraltılmış" versiyonu için, tutarlı ve tam bir benzetici yapılamayacağını ispat etmektedir. Çalışma, içinde süreklilik kurallarına tamamı ile uyulduğu tek bir çalışma alanı içinde işleyen tutarlı ve tam bir benzetici elde etmenin imkansız olduğunu kanıtlayan nihai bir ispatla sona ermektedir.

Bu tezdeki sonuçlar QSIM girdi ve çıktı alfabesi kullanılarak ortaya konmuştur ve girdi çıktı alfabesi QSIM ile aynı olan tüm benzeticiler için geçerlidir. Sonuçlar, yanlış tahminlerin sebepleri ile ilgili daha derin ipuçları vermeleri açısından önemlidir ve daha daraltılmış temsili alfabeler kullanarak tutarlı ve tam nitel benzeticiler yaratmaya çalışan araştırmacıları yakınen ilgilendirmektedir.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii

ABSTRACT iv

ÖZET v

TABLE OF CONTENTS vi

LIST OF TABLES vii

LIST OF SYMBOLS/ABBREVIATIONS ix

1. INTRODUCTION 1

2. THE PREVIOUS PROOF AND GENERALIZATION OF ITS IDEA 7

2.1. Hilbert's Tenth Problem 7

2.2. The Proof 7

2.3. Ineradicable Spurious Behaviors 11

3. DESIDERATA FOR SOUND AND COMPLETE QUALITATIVE SIMULATORS ... 13

4. UNLIMITED REGISTER MACHINES 14

5. NEW INCOMPLETENESS RESULTS FOR QUALITATIVE SIMULATORS 17

6. CONCLUSIONS 44

APPENDIX A: REVISED LEMMAS FOR THEOREM 5.8 45

REFERENCES 50

LIST OF TABLES

Table 1.1. Qualitative constraint types 3

Table 2.1. QSIM model demonstrating representability of integers 8

Table 4.1. URM program computing $f(x, y) = x + y$ 15

Table 5.1. Model of the operating region $OpReg_0$, corresponding to the initialization of the URM 18

Table 5.2. Variables which should inherit magnitudes according to type of the target operating region 19

Table 5.3. Model template for operating regions corresponding to *zero*(*n*) instructions of the URM 20

Table 5.4. Model template for operating regions corresponding to *succ*(*n*) instructions of the URM 21

Table 5.5. Model template for operating regions corresponding to *transfer*(*m, n*) instructions 21

Table 5.6. Model template for operating regions corresponding to *jump*(*m, n, q*) instructions 22

Table 5.7. Model of the operating region $OpReg_{|P|+1}$, corresponding to the end of the URM program 23

Table 5.8. Alternative model template for operating regions corresponding to *jump*(*m, n, q*) instructions which avoids negative numbers 26

Table 5.9. QSIM constraint set built for expressing the equality "$p_6 = 6 * unit$" 27

Table 5.10. Model template to obtain the desired behavior for the variable *U* as a clock variable oscillating between qualitative values (0, *std*) and (*unit*, *std*) 30

Table 5.11. The quantity spaces and initial qualitative values of all variables used in the model 32

Table 5.12. Model template for the first operating region corresponding to the initialization stage 33

Table 5.13. Model template for the second operating region corresponding to the initialization stage 34

Table 5.14. Model template for the first operating region corresponding to *zero*(*n*) instructions of the URM 34

Table 5.15. Model template for the second operating region corresponding to *zero*(*n*) instructions of the URM 35

Table 5.16. Model template for the first operating region corresponding to *succ*(*n*) instructions of the URM 35

Table 5.17. Model template for the second operating region corresponding to *succ*(*n*) instructions of the URM 36

Table 5.18. Model template for the first operating region corresponding to *jump*(*m*, *n*, *q*) instructions of the URM 36

Table 5.19. Model template for the second operating region corresponding to *jump*(*m*, *n*, *q*) instructions of the URM 37

Table 5.20. Model of the operating region $Opreg_{\text{end}}$, corresponding to the end of the URM program 37

Table 5.21. Model template to obtain a variable $Y = \sin t$ 39

Table 5.22. Model template to obtain a variable $Y_i = \sin X_i * t$ 40

Table 5.23. Model template guaranteeing that X_i's are integers and solutions of the Hilbert's polynomial 42

LIST OF SYMBOLS/ABBREVIATIONS

t_0	time-point label of the initial state in qualitative simulation
?	qualitative notation for unknown initial values
∞	infinity
π	the number pi
$\in$	is an element of
$\exists$	there exists a\an
add	qualitative constraint addition
constant	qualitative constraint, denoting qualitative direction is steady
d/dt	qualitative constraint derivative
mult	qualitative constraint multiplication
M^+	qualitative constraint monotonic increasing function
M^-	qualitative constraint monotonic decreasing function
dec	qualitative direction decreasing
inc	qualitative direction increasing
std	qualitative direction steady
qdir	qualitative direction
qmag	qualitative magnitude
jump	jump instruction of URM
succ	successor instruction of URM
transfer	transfer instruction of URM
zero	zero instruction of URM
ODE	Ordinary Differential Equation
QDE	Qualitative Differential Equation
QSIM	Qualitative Simulation
TM	Turing Machine
URM	Unlimited Register Machine

1. INTRODUCTION

Mathematical analysis, specifically, ordinary differential equations, (ODE's) have been used for centuries to describe the behavior of dynamical systems. However, quantitative mathematical techniques need precisely specified quantities and models of the system under consideration, which is not always possible, especially in modeling the real world. Qualitative Reasoning (QR) is an alternative approach to proceed to reason about systems with incomplete or imprecise knowledge in the sense that it focuses on distinctions which make an important, qualitative difference, ignoring the unimportant ones. A qualitative simulator consists of two basic parts: an input-output vocabulary and an algorithm. In this thesis, we are going to use the notation and terminology of QSIM [1], one of the most popular qualitative reasoning systems, although it should be emphasized that the results presented here are valid for all simulators whose input-output vocabularies are "strong" enough to represent the facilities used in a presented proof.

The input to QSIM is given in the form of *qualitative differential equations* (QDE's), which are structural abstractions of ODE's describing the system. QDE's are built with variables occurring in the model and constraints which govern the functional relationship among those variables. The *variables* are continuously differentiable functions of time, which have only finitely many critical points in any bounded interval. Each variable possesses a *quantity space*, a finite, totally ordered set of landmark values, where a landmark is a symbolic name representing a particular value in R. The landmarks $-\infty$, 0 and ∞ are usually part of any quantity space, although QSIM allows these to be omitted in the case that their presence is not necessary. (For instance, a variable representing kinetic energy of a mass is certainly nonnegative and will be bounded with zero on the left.) QSIM has also the optional ability of enlarging a variable's quantity space during the simulation by adding new landmarks.

The representation of *qualitative values* of variables are also part of the input-output vocabulary. A variable's qualitative value is a pair of the form <*qmag*, *qdir*>, where *qmag* denotes the *qualitative magnitude* and *qdir* denotes the *qualitative direction* of the variable. The *qualitative magnitude* possible for a variable are all the points and intervals in its

quantity space, whereas the *qualitative direction* of a variable is simply defined to be the sign of its time derivative. Hence, it can take three values: *inc* (+), standing for increasing, *dec* (-), standing for decreasing, and *std* (0), denoting steady. The *state* of the system in a certain time point or time interval is defined to be the whole set of qualitative values of its variables.

The rules determining the flow of the simulation are specified by the so-called *constraints*, which describe the functional relationships among the variables. That is the point where the algorithm part of the simulator plays its part. The algorithm consists of the transition rules according which the possible next qualitative values of the variables are determined. Different filters, that have the aim to eliminate the states, which do not satisfy the constraints and to let the states survive, which cover the actual next states of the system, are also part of the algorithm.

QSIM has seven basic types of constraints. (See Table 1.1.) Each constraint acts like a predicate imposing a structural relationship among its parameters. For example if we have the constraint $A(t) = \frac{d}{dt} B(t)$, then the qualitative direction of the variable *B* depends on the sign of the variable *A*, i.e. if *A* is positive, then the qualitative values for *A*, in which the qualitative direction is *std* or *dec*, are filtered out. An additional knowledge source of the constraints are the *corresponding values*, which are tuples (pairs or triples usually) of landmark values that the variables in some given constraint can take on at the same time. In the case of an *add* constraint, a corresponding value tuple (*a*, *b*, *c*) where *a*, *b* and *c* are landmarks in appropriate quantity spaces provides a constraint $a + b = c$ on the real numbers *a*, *b* and *c* stand for, which is going to be valid throughout the simulation. Only derivative type constraints can not have such correspondence equations. The arithmetic constraints are manipulated through a "sign algebra" [1]. Note that a QDE can correspond to infinitely many ODE's since an M^+ (M^-) relationship can correspond to an infinite number of possible "quantitative" monotonic functions. Similarly, a qualitative state may correspond to infinitely many quantitative states, since an interval on landmarks contains an infinite number of real values.

Table 1.1. Qualitative constraint types

Constraint Name	Representation	Explanation
add	$X(t) + Y(t) = Z(t)$	
constant	$X(t)$ = a landmark	$\frac{d}{dt} X(t) = 0$
derivative	$\frac{d}{dt} X(t) = Y(t)$	
M^+	$X(t) = f(Y(t)),\ f \in M^+$	$\exists f$ such that $X(t) = f(Y(t))$, where $f' > 0$ over f's domain
M^-	$X(t) = f(Y(t)),\ f \in M^-$	$\exists f$ such that $X(t) = f(Y(t))$, where $f' < 0$ over f's domain
minus	$X(t) = -Y(t)$	
mult	$X(t) * Y(t) = Z(t)$	

The QSIM input vocabulary also provides the possibility of defining several different constraint sets in the case of more complex systems. Each constraint set is valid within some domain or *operating region*. The boundaries of operating regions are specified by landmark values of certain variables, or more generally by boolean conditions of qualitative values of particular variables. When the conditions are satisfied, then the simulation continues with (or "triggers" to) the specified new operating region.

An operating region transition allows several types of information to be specified:

- The qualitative values or boolean combinations of qualitative values of variables, which are going to trigger the transition,
- The name of the new operating region,
- The constraint set (or QDE) for the new operating region,
- The names of the variables which should inherit their qualitative values from the final state of the old operating region to the initial state of the new operating region,
- The names of the variables which should inherit their qualitative directions from the final state of the old operating region to the initial state of the new operating region,
- The variables for which qualitative values are explicitly "asserted", such that they become those specified values in the initial state of the new operating region.

An operating region transition can be interpreted as a discontinuous change in the actual state of the system. For example, if we model a bouncing ball as simply reflecting off the floor, we change its velocity instantaneously from negative to positive at the moment of the bounce. So operating regions correspond to different "instantaneous" events such as strings breaking or switches turning.

Another input given to the simulator are the initial values of the variables in the system. From the most specific to the most general case, a variable's initial value can be specified as a landmark, as a finite interval between two landmarks or as an infinite interval like $(0, \infty)$. QSIM also allows to specify a value of "*?*" for the initial value of a variable, if the initial knowledge about the variable in question does not even include the information of its sign. Note that, in such a situation of incomplete knowledge, it is the simulator's job to find out the possible initial states of the system which are consistent with the constraints of the given model. These n "legal" completions of the initial states built up the n roots of n trees, which are going to be generated by QSIM. The roots correspond to initial states of the system and are labeled with time-point label t_0, and each of the nodes are system states which represent segments of the solutions of the QDE that is given in terms of constraints as input. Each path from the root to the leaf of a tree is a predicted *behavior* of the system. The behavior consists of alternating time-point and time-interval states in the same operating region, whereas an operating region transition is marked with two time point states following each other.

A major property of the qualitative simulation methodology is its *soundness* property, that is no trajectory which is the solution of a concrete equation matching the input can be missing from the output, i.e. every real solution to an ODE consistent with the given QDE and initial state is included among the predictions of QSIM. However, one has noted that some qualitative simulation outputs contained more predictions than the real ones, namely *spurious* behaviors. A simulator is said to be *complete*, if it provides the guarantee not to produce any spurious prediction in its output. However, it has recently been proven [2] that it is impossible to provide the additional guarantee of completeness that a qualitative 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 [1] 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 [2] shows that a "sound and complete" qualitative simulator employing the QSIM vocabulary mentioned above, if it existed, could be used to solve any given instance of Hilbert's Tenth Problem, which is famously undecidable [3]. 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 nonnegative 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 [2] 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.

In this thesis, 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 uncertainty. (i.e. as a single unambiguous real number) in the initial state. Moreover, we prove that qualitative simulation can not be sound and complete, even if only a single operating region and the constraint types *add*, *mult*, *d/dt* and *constant* are allowed.

The rest of this thesis is structured as follows: Chapter 2 examines the proof in [2] and generalizes the proof idea. In Chapter 3, we clarify what one means when one talks about a “sound and complete” qualitative simulator. Chapter 4 describes the Turing-equivalent abstract automata called Unlimited Register Machines (URM) used in our proof of incompleteness. Chapter 5 contains the main results of this paper, whereas Chapter 6 is a conclusion.

2. THE PREVIOUS PROOF AND GENERALIZATION OF ITS IDEA

2.1. 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. It asks for an algorithm for deciding whether a given multivariate polynomial with integer coefficients has integer solutions. It has been proven that no such algorithm exists [3]. The incompleteness proof in [2] makes use of a reduction from this famous problem in the following way: It firstly emphasizes that a given polynomial with integer coefficients $P(x_1, x_2, \ldots, x_n) = 0$ has integer solutions if and only if

$$\sum_{i=1}^{n} \sin^2 \pi.x_i + P^2(x_1, x_2, \ldots, x_n) = 0 \tag{2.1}$$

has real solutions [4]. The representation of the above stated equation in QSIM turns out to include the subproblems of representing a given integer, the number π and the relationship $y = \sin x$ in qualitative simulation in an unambiguous way. The following subsection gives a detailed outline of the proof in [2].

2.2. The Proof

In [2] a real number *r* is said to be *QSIM-representable* if there exists a set of QSIM variable quantity spaces and constraints, from which *r*'s equality to a particular landmark symbol *p* in that set can be unambiguously deduced. Table 2.1 includes the demonstration of QSIM-representability of integers in [2]. These represented integers are used to construct the integer coefficients of the polynomial *P*, where *P* is constructed as a

combination of intermediate terms, which again consist of products of the unknowns in the polynomial. The construction utilizes only the *add*, *mult and constant* constraints.

Table 2.1. QSIM model demonstrating representability of integers

CONSTRAINTS	CORRESPONDENCES	CONCLUSIONS
$A(t) = a_1$		
$A(t) * B(t) = A(t)$	$a_1 * b_1 = a_1$	$b_1 = 1$
$B(t) + C(t) = D(t)$	$b_1 + c_1 = 0$	$c_1 = -1$
$B(t) + B(t) = E(t)$	$b_1 + b_1 = e_1$	$e_1 = 2$
$E(t) + G(t) = H(t)$	$e_1 + g_1 = 0$	$g_1 = -2$
$E(t) + B(t) = J(t)$	$e_1 + b_1 = j_1$	$j_1 = 3$

In order to represent the number π, the proof utilizes QSIM's ability to explicitly represent infinite limits, by stating that it is twice the limit of the function *arctan x* as *x* nears infinity. The representation of the *arctan* function is reached through the following equation systems:

$$K = D^2 + 1 \tag{2.2}$$

$$\frac{dG/dt}{dD/dt} = \frac{1}{D^2 + 1} \quad \Rightarrow \quad \frac{dG}{dD} = \frac{1}{D^2 + 1} \tag{2.3}$$

$$\Rightarrow \quad G = \arctan D + c \tag{2.4}$$

where *K*, *D* and *G* are QSIM variables. The first equation (2.2) is pretty easy to obtain using the *mult* constraint and the representability of the number one in QSIM. The second equation (2.3) is obtainable by just multiplying the derivative of *G* with the variable *K* (= $D^2 + 1$) and equating the product to the derivative of *D*. By specifying *G* as a differentiable function of *D* with the M^+ constraint the chain rule leads us to the third step, which by integration ends with an *arctan* relationship. The remaining part involves substituting the correspondence values $f(0) = 0$ and $f(\infty) = g_1$ to equation 2.4 and concluding that the landmark g_1 has the value $\pi/2$.

For the representation of the sine function a similar way is followed, namely its inverse function is represented. Integrating Equation 2.5 and substituting the initial value $f(0)=0$ we obtain an *arcsin* relationship between the variables G and H. (Equation 2.6)

$$\frac{dH}{dG} = \frac{1}{\sqrt{1-G^2}} \tag{2.5}$$

$$\Rightarrow \quad H = \arcsin G \tag{2.6}$$

However, the domain of the inverse sine function is [-1, 1] and its range is $[-\pi/2, \pi/2]$. Therefore the relationship $G = sin\ H$ only holds in the domain $[-\pi/2, \pi/2]$. In order to represent the relationship over the entire real line, [2] offers two additional relations (Equations 2.7. and 2.8.), where each of these relations are represented using a different operating region (named as OPREG-MINUS and OPREG-PLUS, respectively). The relation given in Equation 2.6 is represented in an operating region called SIN-ORIG. Equations 2.7 and 2.8 are given as following:

$$H = f(G) = -\arcsin G + r_1 \tag{2.7}$$

$$H = f(G) = \arcsin G + r_2 \tag{2.8}$$

where r_1 and r_2 are arbitrary constants.

The representation of the sine function is then obtained via an appropriate trigger mechanism between SIN-ORIG, SIN-PLUS and SIN-MINUS operating regions. Note that a sine relationship over the entire real line can be obtained by consecutive application of these relations in an appropriate order. (i.e. by concatenating them at their endpoints of definition with triggers). Since each sine function is represented with three operating regions, the method requires 3^n different operating regions for n different sine functions.

The remaining part of the proof contains expressing Equation 2.1 using representational techniques explained so far in this section. Each of the variables x_i of P

will be represented by a corresponding QSIM variable X_i. The integer coefficients c_j's of the terms of the polynomial P can be represented as depicted in Table 2.1. Each term $term_j$ of P is a product of integer powers of some of the variables with the coefficient c_j and can be represented therefore by the necessary number of *mult* constraints and intermediate QSIM variables. The terms $term_j$'s can be added with *add* constraints and intermediate QSIM variables to a polynomial variable P_Q, which can then easily be squared with a single *mult* constraint.

The QSIM-representability of π is already demonstrated, hence one can trivially obtain with n new *mult* constraints variables, whose values are equal to $\pi.x_i$. The next thing to accomplish is creating 3^n different operating regions for n different sine functions to equate a QSIM variable S_i to $sin\ \pi.x_i$. Trivially, one can square each S_i with a single *mult* constraint and sum them up with *add* constraints to obtain the sum of the squares of sines represented as a QSIM variable say *SS*. Finally one can add the variables *SS* and P_Q to end up in a variable E, which stands for the whole left side of the Equation 2.1.

The only task to face with is asking the simulator the question whether there exists a tuple of x_i's that make $E = 0$, in such a way that the answer, if it could be computed, could be read off a QSIM output. Firstly, all the X_i variables are to be specified to have the qualitative value <*?, std*>, meaning that we do not even know their signs, let alone their numerical values. For each of the n x_i's there are two possible initial operating regions (*PLUS* or *MINUS*). For all possible combinations of initial states, QSIM is run 2^n times. Since its input is specified incompletely, QSIM is supposed to create all consistent completions and start simulation from there. If QSIM were able to delete all inconsistent completions (i.e. were sound and complete) then the input in all cases would be rejected if and only if P has no solutions in integers. On the other hand, the case, where some initial states survive, would mean that P has integer roots. So this implies that a sound and complete simulator does not exist [2], since such a simulator, if it existed, would decide Hilbert's Tenth Problem whose undecidability is already proven [3]. The proof in [2] demonstrates also, by slightly modifying the above procedure, that a sound and complete simulator can not exist even if the initial input state is required to be consistent.

Note that the proof utilizes the monotonic increasing function (M^+), derivative (*d/dt*), multiplication (*mult*) and *constant* constraints of the several qualitative constraint types available in QSIM vocabulary. (Note the absence of the *add* constraint, for which the paper demonstrates an implementation using the others, in this list.) The initial value <*?, std*> used in the proof includes qualitative interval magnitudes like (0, ∞) or (-∞, 0), with what one might call "infinite uncertainty" about the actual value of the represented number. As demonstrated earlier, for equating a landmark to the number π, QSIM's ability to explicitly represent infinite limits is utilized. 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.

This work basically examines several alternative subsets of QSIM vocabulary which are sufficiently rich, that the problem of building a sound and complete simulator persists. It aims to give more insight about the problem of *spurious* behaviors, which is going to be explained in more detail in the coming subsection.

2.3. Ineradicable Spurious Behaviors

The problem of spurious behaviors in qualitative simulation occurred before the proof we outlined in Section 2.2 came out. See [1, 5, 6, 7, 8, 9, 10] for discussions on specific types of spurious behaviors. But proving a particular behavior of a simulator to be spurious actually shows us actually the way to change our algorithm and arrive at a new simulator, for which the above mentioned behavior is not spurious anymore. (Just add your proof as a filter to your algorithm). The rest of this section is devoted to emphasize that the discovered spurious behaviors via reductions from undecidable problems are different in a remarkable way (namely, what is discovered is their existence, and not a particular instance of them).

Undecidable problems have the remarkable property that no algorithm exists for their solutions. Hence, a reduction from an undecidable problem in our case means that it is valid for all simulators supporting the same input-output vocabulary, whatever change you

perform in the algorithm of your simulator or whichever filters you add to your algorithm. So the proof idea can be generalized as: Formulate an undecidable problem using your vocabulary and ask your simulator for the answer of it, such that you will be sure your simulator can not provide an answer in a sound and complete manner. As an example consider you have an imaginary Turing machine simulator, which we expect to accomplish the following task: If it is given a Turing machine (TM), which will halt, then its job is to reject it due to some inconsistency. If the input is a TM, which is going not to halt then it will simulate it. Remembering that the halting problem is undecidable, what should we expect as its behavior when a TM is given as input, is it going to start simulation or report inconsistency? With the additional claim that it is sound, it certainly will start simulation in order not to miss a possible solution. But there is no computational tool which is able to comment on spuriousness of that created behavior. Such behaviors are *ineradicable* spurious behaviors, whose spuriousness can not be decided by any algorithm.

3. DESIDERATA FOR SOUND AND COMPLETE QUALITATIVE SIMULATORS

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 Chapter 5.

4. 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 [11] 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*.

$$\textit{If } r_m = r_n \textit{ then jump to } I_q$$

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 4.1. 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.

Table 4.1. URM program computing $f(x, y) = x + y$

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

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 [11, 12] 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 [11].

5. 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 5.1: 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. (See Section 2.2.)) *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 5.1–5.7 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 5.1. 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.

Table 5.1. Model of the operating region $OpReg_0$, corresponding to the initialization of the URM

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*)

As seen in Tables 5.1-5.7, 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 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.)

Table 5.2. Variables which should inherit magnitudes according to type of the target operating region

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*

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 $(0, l_i, \infty)$, 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 5.1. 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 qualita 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.

Table 5.3. Model template for operating regions corresponding to *zero*(*n*) instructions of the URM

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 5.4. Model template for operating regions corresponding to *succ*(*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 5.5. Model template for operating regions corresponding to *transfer*(*m, n*) instructions

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 5.6. Model template for operating regions corresponding to *jump*(*m, n, q*) 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: $U \leftarrow (0, inc)$

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.

Table 5.7. Model of the operating region $OpReg_{|P|+1}$, corresponding to the end of the URM program

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

Theorem 5.2: 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 5.1 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 5.3: 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 5.2. We observe that in the construction of Theorem 5.1, *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 5.2, 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.

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

Theorem 5.4: 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 impossible 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 5.6 with Table 5.8 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 5.8, *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.

Table 5.8. Alternative model template for operating regions corresponding to *jump*(*m, n, q*) instructions which avoids negative numbers

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: $U \leftarrow (0, inc)$

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 5.5: 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 5.1-5.3 only for equating landmark values to unambiguous integers. Assume that we delete the *mult* constraints from our model of Theorem 5.3. 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 5.9 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.

Table 5.9. QSIM constraint set built for expressing the equality "$p_6 = 6 * unit$"

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

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 5.3.

As a side remark, note that if one has access to a semi-quantitative simulator [1] where it is possible for the user to specify bounding numerical intervals for the landmark values, one can use our construction of Theorem 5.1, 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.)

We observe that some of the variables change their qualitative magnitudes and directions discontinously during operating region transitions in the proofs of previous theorems. The next theorem proves that maintaining soundness and completeness simultaneously is impossible even if we do not allow any qualitative variable to perform such a change, and force each variable's magnitude and direction to be inherited to the next operating region.

Theorem 5.6: Even if the qualitative representation is narrowed so that only the *d/dt*, *add* and *constant* constraints can be used in QDE's, and no variable's magnitude and direction are allowed to perform a discontinous change during operating region transitions, it is still impossible to build a sound and complete qualitative simulator based on this input-output vocabulary.

Before proceeding to the proof of Theorem 5.6, we state the following lemma, of which we will make use in the proof.

Lemma 5.1: The Unlimited Register Machine (URM) is Turing equivalent even if we remove the *transfer* instruction from its instruction set.

Proof of Lemma 5.1: We are going to show that the *transfer* instruction can be simulated by means of *succ*, *jump* and *add* instructions. Assume that a URM has the instruction *transfer*(*m*, *n*) in the *k*'th position of its program. We replace that instruction with the following program segment, and renumber the remaining instructions and labels in the program accordingly:

I_k: *zero*(*n*)
I_{k+1}: *jump*(*m, n, k*+4)
I_{k+2}: *succ*(*n*)
I_{k+3} : *jump*(1, 1, *k* +1)

Note that the program segment above first sets the *n*'th register to 0 and goes into a loop where the value of *n*'th register is incremented until the condition in I_{k+1} is satisfied implying that the values of *n*'th and *m*'th registers are equal. That is the expected result of a *transfer*(*m*, *n*) instruction.

Proof of Theorem 5.6: The proof basically makes some changes in the QSIM models used for simulating the given URM in the previous theorems. Note that relying on Lemma 5.1 there is no need to give a model template for *transfer* instructions. As in the proof of Theorem 5.1, we define a QSIM variable NR_i for each of the N registers R_i appearing in the URM program. In addition to that, we define the auxiliary variables TR_i for all $i \in \{1, ...,N\}$ and also the variables D_{ij} for all $i \in \{1, ...,N\}$ and for all $j \in \{1, ...,N\}$ such that $i \neq j$. Each variable D_{ij} satisfies the following equation throughout the simulation: $D_{ij} = NR_i - NR_j$, i.e. we keep track of the differences of register values. To achieve this we put the constraint $add(D_{ij,}\ NR_{j,}\ NR_i)$ for all $i, j \in \{1,...,N\}$ such that $i \neq j$ to each operating region we are going to construct. These difference variables will enable us to compare two register values (to see whether they are equal, meaning their difference is 0) in *jump* type operating regions.

Moreover, for each of the k registers TR_i which have nonzero initial values, we define a variable named L_i, $i \in \{1, ...,k\}$ which are equated to the corresponding initial values of the registers, i.e. to $l_i = a_i$ * unit, and remain constant at these landmarks

throughout the simulation. (Note that the procedure to obtain these conditions is explained in the proof of Theorem 5.5)

For each instruction type in the given URM program, we define *two* operating regions. In this case our clock variable U will increase in the first of these operating regions from (0, *std*) to the value (*unit*, *std*), and decrease in the next one from the qualitative value (*unit*, *std*) to (0, *std*), performing no discontinous jump in the program. In order to achieve a variable with such behavior we make use of a simple harmonic oscillator model given in Table 11, where the variable X (note that the variable X here is different than the one used in the previous proofs, and denotes the "position" of the oscillating "object") oscillates between its amplitudes *unit*/2 and −*unit*/2, and the variable U is equal to the value X + *unit*/2, alternating between 0 and *unit*. The model template for producing this behavior is given in Table 5.10 and it is added to each constructed operating region.

Table 5.10. Model template to obtain the desired behavior for the variable U as a clock variable oscillating between qualitative values (0, *std*) and (*unit*, *std*) (this template is to be inserted to all constructed operating regions)

CONSTRAINTS	**CORRESPONDENCES**	**MEANING**
$Z = 0$		
$B = unit$		
$C = c_1$		
add(*C, C, B*)	$c_1 + c_1 = unit$	$c_1 = unit / 2$
add(*C, E, V*)	$c_1 + 0 = v_1$	$v_1 = unit / 2$
add(*C, D, X*)	$c_1 + 0 = x_1$	$x_1 = unit / 2$
d/dt(*X, V*)		$\frac{dX}{dt} = V$
d/dt(*V, A*)		$\frac{d^2X}{dt} = A$
add(*X, A, Z*)		$\frac{d^2X}{dt} + X = 0$
add(*X, C, U*)	$x_1 + c_1 = u_1$	$u_1 = unit$

Looking at Table 5.10, we obtain the equation:

$$\frac{d^2}{dt} X(t) + X(t) = 0 \tag{5.1}$$

which has a solution of the form:

$$X(t) = c_1 \times \sin t + c_2 \times \cos t \tag{5.2}$$

and hence its time derivative V has the form:

$$V(t) = c_1 \times \cos t - c_2 \times \sin t \tag{5.3}$$

Together with the initial values depicted in Table 5.11,

$$X(t_0 = 0) = 0$$
$$V(t_0 = 0) = unit / 2 \tag{5.4}$$

we obtain that $c_1 = unit/2$ and $c_2 = 0$ and X obeys the equation:

$$X(t) = unit / 2 \times \sin t \tag{5.5}$$

proving that the variable X oscillates between the values $unit/2$ and $-unit/2$, as expected. (Note that without the assumption of t_0 being 0, we still obtain the expected sine behavior just shifted by t_0.)

The model templates for initialization operating regions $OpReg_{0,1}$ and $OpReg_{0,2}$ are given in Tables 5.12 and 5.13, respectively. The constraints $add(NR_i, Z, L_i)$ and $add(TR_i, Z, L_i)$ where Z is the zero variable, ensure that initially all registers with nonzero initial values start the simulation at appropriate values.

All variables' values are inherited in all possible transitions, such that no variable's qualitative value ever performs a discontinous change. We are going to denote the two

operating regions corresponding to the *i*'th instruction of the URM program with $Opreg_{i,1}$ and $Opreg_{i,2}$.

The quantity spaces for all variables and the initial values given to the simulator are given in Table 5.11. Note that the initial values of variables not given in Table 5.11 are set by the simulator in a unique way due to the constraints.

Table 5.11. The quantity spaces and initial qualitative values of all variables used in the model

VARIABLE	**QUANTITY SPACE**	**INITIAL VALUE**
NR_i (a_i=0)		(0, *std*)
NR_i ($a_i \neq 0$)	(0, ∞)	((0, ∞), *std*)
TR_i (a_i=0)		(0, *std*)
TR_i ($a_i \neq 0$)	(0, ∞)	((0, ∞), *std*)
D_{ij}	(-∞, 0, ∞)	
L_i	(0, l_i, ∞)	(l_i, *std*)
U	(0, u_1)	(0, *inc*)
B	(0, *unit*)	
C	(0, c_1)	(c_1, *std*)
D	(-∞, 0, ∞)	
E	(-∞, 0, ∞)	
X	(-∞, 0, x_1, ∞)	(0, *inc*)
V	(-∞, 0, v_1, ∞)	(v_1, *std*)
A	(-∞, 0, ∞)	((-∞, 0), *dec*)
Z	(0, ∞)	(0, *std*)

Looking carefully at tables 5.12-5.20, which correspond to instructions of the URM, we deduce that the simulation flows in a unique way, i.e. in the first operating regions of an instruction, the variable *U* increases from (0, *std*) to the value (*unit*, *std*), where the simulation triggers to the second operating region of that instruction. The variable *U* decreases from (*unit*, *std*) to (0, *std*) in that second region and the simulation continues with the first operating region of the next instruction. The simulation ends when $OpReg_{|P|+1,1}$ ($=Opreg_{end}$), which is a single operating region corresponding to the end of the URM program, is reached. Only in *zero* type operating regions there is the possibility that the simulation branches into more than one behavior, where the behaviors which do not correspond to the expected trajectory of the actual URM are directed to $OpReg_{|P|+1,1}$ ($=Opreg_{end}$). The rest of the proof is the same as in Theorem 5.2. Our contradiction variable *S* ensures that only the behavior of a non-halting URM leads to a consistent initial state, hence determining the consistency of the initial state is equivalent to deciding the halting problem, leading to a contradiction.

Table 5.12. Model template for the first operating region corresponding to the initialization stage

Operating Region: $OpReg_{0,1}$
{Type: *Initialization***}**
Constraint Set: $add(NR_i, Z, L_i)$ (*for all* $i \in \{1,...,k\}$)
$add(TR_i, Z, L_i)$ (*for all* $i \in \{1,...,k\}$)
Possible Transition:
Trigger: ($U = (unit, std)$)
New Operating Region: $OpReg_{0,2}$
Variables inheriting qualitative values and directions:
All variables

Table 5.13. Model template for the second operating region corresponding to the initialization stage

Operating Region: $OpReg_{0,2}$

{Type: *Initialization***}**

Constraint Set: $add(NR_i, Z, L_i)$ $(for\ all\ i \in \{1,...,k\})$

$add(TR_i, Z, L_i)$ $(for\ all\ i \in \{1,...,k\})$

Possible Transition:

Trigger: $(U = (0, std))$

New Operating Region: $OpReg_{1,1}$

Variables inheriting qualitative values and directions:

All variables

Table 5.14. Model template for the first operating region corresponding to *zero*(*n*) instructions of the URM

Operating Region: $OpReg_{i,1}$

{Type: *zero*(*n*)**}**

Constraint Set: $constant(NR_i)$ $(for\ all\ i \in \{1,...,N\}\text{-}\{n\})$

$constant(TR_i)$ $(for\ all\ i \in \{1,...,N\})$

Possible Transition:

Trigger: $(U = (unit, std))\ AND\ (NR_i = (0, std))$

New Operating Region: $OpReg_{i,2}$

Variables inheriting qualitative values and directions:

All variables

Possible Transition:

Trigger: $(U = (unit, std))\ AND\ (NR_i \neq (0, std))$

$OR\ (NR_i = (\infty, inc))\ OR\ (NR_i = ((0,\infty), std))$

New Operating Region: $Opreg_{end}$

Variables inheriting qualitative values and directions:

All variables

Table 5.15. Model template for the second operating region corresponding to *zero*(*n*) instructions of the URM

Operating Region: $OpReg_{i,2}$
{Type: *zero*(*n*)**}**
Constraint Set: $constant(NR_i)$ (*for all* $i \in \{1,...,N\}$)
$constant(TR_i)$ (*for all* $i \in \{1,...,N\}-\{n\}$)
Possible Transition:
Trigger: ($U = (0, std)$) *AND* ($TR_i \neq (0, std)$)
New Operating Region: $OpReg_{i+1,1}$
Variables inheriting qualitative values and directions:
All variables

Possible Transition:
Trigger: ($U = (unit, std)$) *AND* ($TR_i \neq (0, std)$)
OR ($TR_i = (\infty, inc)$) *OR* ($TR_i = ((0,\infty), std$)
New Operating Region: $Opreg_{end}$
Variables inheriting qualitative values and directions:
All variables

Table 5.16. Model template for the first operating region corresponding to *succ*(*n*) instructions of the URM

Operating Region: $OpReg_{i,1}$
{Type: *succ*(*n*)**}**
Constraint Set: $add(TR_n, U, NR_n)$
$constant(NR_i)$ (*for all* $i \in \{1,...,N\}-\{n\}$)
$constant(TR_i)$ (*for all* $i \in \{1,...,N\}$)
Possible Transition:
Trigger: ($U = (unit, std)$)
New Operating Region: $OpReg_{i,2}$
Variables inheriting qualitative values and directions:
All variables

Table 5.17. Model template for the second operating region corresponding to *succ*(*n*) instructions of the URM

Operating Region: $OpReg_{i,2}$

{Type: *succ*(*n*)**}**

Constraint Set: $add(TR_n, U, NR_n)$

$constant(NR_i)$ (*for all* $i \in \{1,...,N\}$)

$constant(TR_i)$ (*for all* $i \in \{1,...,N\}-\{n\}$)

Possible Transition:

Trigger: ($U = (0, std)$)

New Operating Region: $OpReg_{i+1,1}$

Variables inheriting qualitative values and directions:

All variables

Table 5.18. Model template for the first operating region corresponding to *jump*(*m*, *n*, *q*) instructions of the URM

Operating Region: $OpReg_{i,1}$

{Type: *jump*(*m*, *n*, *q*)**}**

Constraint Set: $constant(NR_i)$ (*for all* $i \in \{1,...,N\}$)

$constant(TR_i)$ (*for all* $i \in \{1,...,N\}$)

Possible Transition:

Trigger: ($U = (unit, std)$)

New Operating Region: $OpReg_{i,2}$

Variables inheriting qualitative values and directions:

All variables

Table 5.19. Model template for the second operating region corresponding to *jump*(*m*, *n*, *q*) instructions of the URM

Operating Region: $OpReg_{i,2}$

{Type: *jump*(*m*, *n*, *q*)**}**

Constraint Set: *constant*(NR_i) (*for all* $i \in \{1,...,N\}$)

constant(TR_i) (*for all* $i \in \{1,...,N\}$)

Possible Transition:

Trigger: (D_{mn}= (0, *std*)) *AND* (*U*= (0, *std*))

New Operating Region: $OpReg_{q,1}$

Variables inheriting qualitative values and directions:

All variables

Possible Transition:

Trigger: ($D_{mn} \neq$ (0, *std*)) *AND* (*U* = (0, *std*))

New Operating Region: $OpReg_{i+1,1}$

Variables inheriting qualitative values and directions:

All variables

Table 5.20. Model of the operating region $Opreg_{end}$, corresponding to the end of the URM program

Operating Region: $OpReg_{|P|+1,\ 1}$ (= $Opreg_{end}$)

{Type: *End***}**

Constraint Set: *constant*(NR_i) (*for all* $i \in \{1,...,N\}$)

constant(TR_i) (*for all* $i \in \{1,...,N\}$)

constant(*V*)

constant(*A*)

constant(*X*)

Theorem 5.7: 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 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.

We are going to start our proof with some preliminary lemmas:

Lemma 5.2: The function $Y = \sin t$ is QSIM-representable, if it is given that t_0=0.

Proof: We will firstly examine the solutions of the following equation:

$$\frac{d^2}{dt} Y(t) + Y(t) = 0 \tag{5.6}$$

which has a solution of the form:

$$Y(t) = c_1 \times \sin t + c_2 \times \cos t \tag{5.7}$$

and hence its time derivative V has the form:

$$V(t) = c_1 \times \cos t - c_2 \times \sin t \tag{5.8}$$

substituting the initial values :

$$Y(t_0 = 0) = 0$$
$$V(t_0 = 0) = 1 \tag{5.9}$$

to Equations (5.7) and (5.8), we obtain that $c_1 = 1$ and $c_2 = 0$, hence Y obeys the equation $Y = \sin t$.

Table 5.21 shows that the Equation (5.6) with the initial values (5.9) is QSIM-representable.

Table 5.21. Model template to obtain a variable $Y = \sin t$

CONSTRAINTS	CORRESPONDENCES	MEANING
$Z = 0$		
$B = b_1$		
mult(*B, C, B*)	$b_1 * c_1 = c_1$	$c_1 = 1$
add(*C, D, V*)	$c_1 + 0 = v_1$	$v_1 = 1$
d/dt(*Y, V*)		$\frac{dY}{dt} = V$
d/dt(*V, A*)		$\frac{d^2Y}{dt} = A$
add(*Y, A, Z*)		$\frac{d^2Y}{dt} + Y = 0$

INITIAL VALUES
$Y(t_o) = 0 \;\; V(t_o) = v_1$

Lemma 5.3: The function $Y_i = \sin X_i * t$, where X_i is a constant, is QSIM-representable, if it is given that t_0=0.

Proof: We will firstly examine the solutions of the following equation set:

$$\frac{d^2}{dt} Y_i(t) + M_i(t) * Y_i(t) = 0 \tag{5.10}$$

$$\begin{aligned} M_i(t) = X_i * X_i \;\Rightarrow\; & \sqrt{M_i(t)} = X_i,\; X_i \geq 0 \\ & \sqrt{M_i(t)} = -X_i,\; X_i < 0 \end{aligned} \tag{5.11}$$

where the initial values are:

$$\begin{aligned} Y_i(t_0 = 0) &= 0 \\ V_i(t_0 = 0) &= X_i \end{aligned} \tag{5.12}$$

The solution for Equation (5.10) is given as:

$$Y_i(t) = c_1 \times \sin(\sqrt{M_i(t)} * t) + c_2 \times \cos(\sqrt{M_i(t)} * t) \tag{5.13}$$

hence substituting M_i from Equation (5.11) to Equation (5.13) we get:

$$\begin{aligned}
&\text{For } X_i \geq 0 \\
&Y_i(t) = c_1 \times \sin(X_i * t) + c_2 \times \cos(X_i * t) \\
&V_i(t) = c_1 \times X_i \times \cos(X_i * t) - c_2 \times X_i \times \sin(X_i * t) \\
&\text{For } X_i < 0 \\
&Y_i(t) = c_1 \times \sin((-X_i) * t) + c_2 \times \cos((-X_i) * t) \\
&V_i(t) = c_1 \times (-X_i) \times \cos((-X_i) * t) - c_2 \times (-X_i) \times \sin((-X_i) * t)
\end{aligned} \tag{5.14}$$

where V_i(t) is the time derivative of Y_i(t). Substituting the initial values in (7) above, we obtain for $X_i \geq 0$ $c_1 = 1$ and $c_2 = 0$ and for $X_i < 0$ $c_1 = -1$ and $c_2 = 0$ implying, $Y_i(t) = \sin(X_i * t)$ for $X_i \geq 0$ or $X_i < 0$.

Table 5.22. Model template to obtain a variable $Y_i = \sin X_i * t$

CONSTRAINTS	**INITIAL VALUES**	**MEANING**
$Z = 0$		
add(X_i, C_i, V_i)	$C_i(t_o) = 0$	$V_i(t_o) = X_i$,
d/dt(Y_i, V_i)		$\frac{dY_i}{dt} = V_i$
d/dt(V_i, A_i)		$\frac{d^2Y_i}{dt} = A_i$
mult(X_i, X_i, M_i)		$M_i = X_i^2$
mult(M_i, Y_i, L_i)		$L_i = M_i * Y_i$
add(A_i, L_i, Z)		$\frac{d^2Y_i}{dt} + M_i * Y_i = 0$
	$Y_i(t_o) = 0$	

Table 5.22 shows that Equations (5.10) and (5.11) and the initial values given in (5.12) are QSIM-representable.

Lemma 5.4: Starting at t_0=0, the function Y = sin t reaches the value 0 for the first time at time point t_i=π. Moreover the function Y_i = sin X_i*t reaches 0 at the same time point t_i if and only if X_i is an integer.

Proof: Obviously, solving the equation sin t = 0 we get the solution for t as t = n*π, $n \in$ Z and since we are interested for the first time point it becomes 0 we get $t_i = \pi$. For the "only if" part of the second statement, assume that the function Y_i = sin X_i*t reaches 0 at $t_i = \pi$. Then:

$$\begin{aligned} &\sin(X_i \times \pi) = 0 \\ &X_i \times \pi = n \times \pi \qquad n \in Z \;\Rightarrow\; X_i \in Z \end{aligned} \tag{5.15}$$

For the if part, we have:

$$X_i \in Z \text{ and } Y_i = \sin(X_i \times \pi) \;\Rightarrow\; Y_i = 0 \tag{5.16}$$

Proof of Theorem 5.7: Our proof will rely again on a contradiction, namely that a sound and complete simulator, if it existed, would solve an undecidable problem. The undecidable problem we are going to use in our reduction this time is the famous Hilbert's Tenth Problem. Hilbert's Tenth Problem asks about integer solutions of a given multivariate polynomial of the form $P(x_1, x_2, x_3, ..., x_n)$ with integer coefficients (See Section 2.1). It has been proven by Yuri Matiyasevich [3], that the problem is undecidable.

Assume that we are given a polynomial $P(x_1, x_2, x_3, ..., x_n)$ with integer coefficients. In [2], it has been shown how to represent integers and how to construct such a polynomial in QSIM in a single operating region.(See Section 2.2) The construction makes use of the constraints *mult*, *add* and *constant*. So for each variable x_i appearing in the polynomial we define a constant QSIM variable X_i and construct a polynomial variable P. We add Table

5.21 of Lemma 5.2 to our model and obtain a variable Y, where $Y(t)= \sin t$. We specify its quantity space as $(0, \infty)$, such that the simulation is guaranteed to finish after π time units. For each X_i, we define a C_i, L_i, M_i, V_i, A_i and Y_i, adding the constraints given in Table 5.22 to our model obtaining Y_i's, where each of them obeys the equation $Y_i = \sin X_i * t$. Moreover we construct a variable YS where YS is the sum of the squares of Y_i's. ($YS = \sum_{i=1}^{n} Y_i^2$)

Note that this is easy to obtain in QSIM where we define auxiliary variables YS_i for each Y_i with the constraints $mult(Y_i, Y_i, YS_i)$ implying $YS_i = Y_i^2$ and then adding these two by two using additional variables until we obtain YS. Note that the fact that the variable YS has become 0 implies that all the variables Y_i have become 0.

Finally we need to construct the constraints which guarantee that only the behavior is consistent, in which X_i's are integers (or in our case relying on Lemma 5.4 the behavior in which the variables Y and YS become 0 at t_i) and the polynomial is satisfied (or in our case the variable P is 0). To serve our aim, we lastly add Table 5.23 to our model. The first three constraints in the table are there to define a variable POS, which is surely positive.

Table 5.23. Model template guaranteeing that X_i's are integers and solutions of the Hilbert's polynomial

CONSTRAINTS	**QUANTITY SPACE**	**MEANING**
$F = f_1$	F: $(0, f_1, \infty)$	$F = f_1 > 0$
$mult(G,G, T)$	G: $(-\infty, \infty)$	$T \geq 0$
$add(F, T, POS)$	POS: $(0, \infty)$	$POS > 0$
$P = 0$		
$mult(POS, YS, Y)$		$Y = 0 \Rightarrow YS = 0$

Let us analyze what a supposedly sound and complete simulator would do when we start to simulate our model by specifying the variables X_i's as (*?*, *std*). A sound and

complete simulator would fill up the initial values of the given X_i's with those values which do not cause any inconsistency with our model. But those X_i's are exactly the integer solutions of Hilbert's Tenth problem as follows:

We know that the variable Y and the variables Y_i's, hence the variable YS as the square of their sums are initially 0, which is consistent with fifth constraint of Table 5.23. As the fourth constraint of the table indicates that the polynomial variable P is constant at 0 throughout the simulation, our simulation can start only with solutions of the polynomial. When the variable Y gets to 0 for a second time after π time units, we conclude due to the fifth constraint, that the variable YS has to have become 0, implying that all Y_i's must have become 0. By Lemma 5.4 we conclude that X_i's with which the simulation started must be integers. So if our supposedly complete and sound simulator rejects the initial values of our model due to inconsistency we know that the polynomial has no integer solutions, on the other hand, if it starts simulation we conclude that it has, which is a decision procedure for Hilbert's Tenth Problem leading to contradiction.

We want to note that the assumption t_0 is 0 at simulation start is not a necessary one for our proof to be valid and is used only for an easier mathematical demonstration of the proof. Without that assumption all variables in our model turn out to be shifted by t_0 at the time axis, i.e. the constraints in Table 5.21 imply that the variable Y satisfies the equation $Y = \sin (t- t_0)$ and Table 5.22 implies that $Y_i = \sin X_i*(t- t_0)$. The fact that after π time units (at $t_i = t_0+\pi$ this time) the variable Y becomes 0 is still valid. Moreover it remains also valid that all Y_i's become 0 at $t_i = t_0+\pi$ if and only if X_i's are integers, which is sufficient for the validity of our proof. (Refer to Appendix A for revised versions for Lemma 5.2, 5.3 and 5.4 without the assumption of t_0 being 0)

We want to emphasize that the construction of the model is in a single operating region and no constraints other than *constant*, *d/dt*, *add* and *mult* are used in it.

6. CONCLUSIONS

In this thesis, 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 behaviors. 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 [2] and the alternative proofs of theorems 5.1-5.7 presented in Chapter 5 make heavy use of transitions between multiple operating regions in the QSIM model. However, Theorem 5.8 indicates that sound and complete qualitative simulation can not be achieved using a single operating region.

APPENDIX A: REVISED LEMMAS FOR THEOREM 5.8

Lemma A.1 (Revised version of Lemma 5.2): Without the assumption $t_0 = 0$, the function $Y = \sin(t\text{-}t_0)$ is QSIM-representable.

Proof: We will firstly examine the solutions of the following equation:

$$\frac{d^2}{dt} Y(t) + Y(t) = 0 \tag{A.1}$$

which has a solution of the form:

$$Y(t) = c_1 \times \sin t + c_2 \times \cos t \tag{A.3}$$

and hence its time derivative V has the form:

$$V(t) = c_1 \times \cos t - c_2 \times \sin t \tag{A.3}$$

substituting the initial values :

$$Y(t_0 = 0) = 0$$
$$V(t_0 = 0) = 1 \tag{A.4}$$

to Equations (A.2) and (A.3), we obtain the equation system:

$$0 = c_1 \times \sin t_0 + c_2 \times \cos t_0$$
$$1 = c_1 \times \cos t_0 - c_2 \times \sin t_0 \tag{A.5}$$

where, using Cramer's rule for linear equations [13] we get:

$$c_1 = \frac{\begin{vmatrix} 0 & \cos t_0 \\ 1 & -\sin t_0 \end{vmatrix}}{\begin{vmatrix} \sin t_0 & \cos t_0 \\ \cos t_0 & -\sin t_0 \end{vmatrix}} = \frac{-\cos t_0}{-(\sin^2 t_0 + \cos^2 t_0)} = \cos t_0$$

and

$$c_2 = \frac{\begin{vmatrix} \sin t_0 & 0 \\ \cos t_0 & 1 \end{vmatrix}}{\begin{vmatrix} \sin t_0 & \cos t_0 \\ \cos t_0 & -\sin t_0 \end{vmatrix}} = \frac{\sin t_0}{-(\sin^2 t_0 + \cos^2 t_0)} = -\sin t_0$$

Substituting c_1 and c_2 into Equation (A.2) we get:

$$Y(t) = \cos t_0 \times \sin t - \sin t_0 \times \cos t = \sin(t - t_0) \tag{A.6}$$

Lemma A.2 (Revised version of Lemma 5.3): Without the assumption $t_0 = 0$, the function $Y_i = \sin X_i*(t-t_0)$, where X_i is a constant number, is QSIM-representable.

Proof: We will firstly examine the solutions of the following equation set:

$$\frac{d^2}{dt} Y_i(t) + M_i(t) * Y_i(t) = 0 \tag{A.7}$$

$$\begin{aligned} M_i(t) = X_i * X_i \Rightarrow \sqrt{M_i(t)} &= X_i,\ X_i \geq 0 \\ \sqrt{M_i(t)} &= -X_i,\ X_i < 0 \end{aligned} \tag{A.8}$$

where the initial values are:

$$\begin{aligned} Y_i(t_0 = 0) &= 0 \\ V_i(t_0 = 0) &= X_i \end{aligned} \tag{A.9}$$

The solution for Equation (A.7) is given as:

$$Y_i(t) = c_1 \times \sin(\sqrt{M_i(t)} * t) + c_2 \times \cos(\sqrt{M_i(t)} * t) \tag{A.10}$$

hence substituting M_i from Equation (A.8) to Equation (A.10) we get:

$$\begin{aligned}
&\text{For } X_i \geq 0 \\
&Y_i(t) = c_1 \times \sin(X_i * t) + c_2 \times \cos(X_i * t) \\
&V_i(t) = c_1 \times X_i \times \cos(X_i * t) - c_2 \times X_i \times \sin(X_i * t) \\
&\text{For } X_i < 0 \\
&Y_i(t) = c_1 \times \sin((-X_i) * t) + c_2 \times \cos((-X_i) * t) \\
&V_i(t) = c_1 \times (-X_i) \times \cos((-X_i) * t) - c_2 \times (-X_i) \times \sin((-X_i) * t)
\end{aligned} \tag{A.11}$$

where $V_i(t)$ is the time derivative of $Y_i(t)$. Substituting the initial values of (A.9) in (A.11), we obtain the equation systems:

$$\begin{aligned}
&\text{For } X_i \geq 0 \\
&0 = c_1 \times \sin(X_i * t_0) + c_2 \times \cos(X_i * t_0) \\
&X_i = c_1 \times X_i \times \cos(X_i * t_0) - c_2 \times X_i \times \sin(X_i * t_0) \text{ or} \\
&1 = c_1 \times \cos(X_i * t_0) - c_2 \times \sin(X_i * t_0) \\
&\text{For } X_i < 0 \\
&0 = c_1 \times \sin((-X_i) * t_0) + c_2 \times \cos((-X_i) * t_0) \\
&X_i = c_1 \times (-X_i) \times \cos((-X_i) * t_0) - c_2 \times (-X_i) \times \sin((-X_i) * t_0) \text{ or} \\
&-1 = c_1 \times \cos((-X_i) * t_0) - c_2 \times \sin((-X_i) * t_0)
\end{aligned} \tag{A.12}$$

for $X_i \geq 0$, solving for c_1 and c_2:

$$c_1 = \frac{\begin{vmatrix} 0 & \cos(X_i * t_0) \\ 1 & -\sin(X_i * t_0) \end{vmatrix}}{\begin{vmatrix} \sin(X_i * t_0) & \cos(X_i * t_0) \\ \cos(X_i * t_0) & -\sin(X_i * t_0) \end{vmatrix}} = \frac{-\cos(X_i * t_0)}{-(\sin^2(X_i * t_0) + \cos^2(X_i * t_0))} = \cos(X_i * t_0)$$

$$c_2 = \frac{\begin{vmatrix} \sin(X_i * t_0) & 0 \\ \cos(X_i * t_0) & 1 \end{vmatrix}}{\begin{vmatrix} \sin(X_i * t_0) & \cos(X_i * t_0) \\ \cos(X_i * t_0) & -\sin(X_i * t_0) \end{vmatrix}} = \frac{\sin(X_i * t_0)}{-(\sin^2(X_i * t_0) + \cos^2(X_i * t_0))} = -\sin(X_i * t_0)$$

and substituting to $Y_i(t)$:

$$Y_i(t) = \cos(X_i * t_0) \times \sin(X_i * t) - \sin(X_i * t_0) \times \cos(X_i * t) = \sin(X_i * (t - t_0)) \ , \ X_i \geq 0 \qquad (A.13)$$

for $X_i < 0$, solving for c_1 and c_2:

$$c_1 = \frac{\begin{vmatrix} 0 & \cos((-X_i) * t_0) \\ -1 & -\sin((-X_i)_i * t_0) \end{vmatrix}}{\begin{vmatrix} \sin((-X_i) * t_0) & \cos((-X_i) * t_0) \\ \cos((-X_i)_i * t_0) & -\sin((-X_i)_i * t_0) \end{vmatrix}} = \frac{\cos(X_i * t_0)}{-(\sin^2(X_i * t_0) + \cos^2(X_i * t_0))} = -\cos(X_i * t_0)$$

$$c_2 = \frac{\begin{vmatrix} \sin((-X_i) * t_0) & 0 \\ \cos((-X_i) * t_0) & -1 \end{vmatrix}}{\begin{vmatrix} \sin((-X_i) * t_0) & \cos((-X_i)_i * t_0) \\ \cos((-X_i)_i * t_0) & -\sin((-X_i)_i * t_0) \end{vmatrix}} = \frac{\sin(X_i * t_0)}{-(\sin^2(X_i * t_0) + \cos^2(X_i * t_0))} = -\sin(X_i * t_0)$$

and substituting to $Y_i(t)$:

$$Y_i(t) = -\cos(X_i * t_0) \times \sin((-X_i) * t) - \sin(X_i * t_0) \times \cos((-X_i)_i * t) = \sin(X_i * (t - t_0)) \ , \ X_i < 0$$

Hence we have, $Y_i(t) = \sin(X_i * (t - t_0)) \ , \ X_i \geq 0$ or $X_i < 0$.

Lemma A.3 (Revised version of Lemma 5.3): Starting at t_0, the function $Y = \sin(t\text{-}t_0)$ reaches the value 0 for the first time at time point $t_i = t_0 + \pi$. Moreover the function $Y_i = \sin X_i * (t\text{-} t_0)$ reaches 0 at the same time point t_i if and only if X_i is an integer.

Proof: Obviously, solving the equation $\sin(t\text{-} t_0) = 0$ we get the solution for $t\text{-} t_0$ as $t\text{-} t_0 = n * \pi$, $n \in Z$ and since we are interested for the first time point it becomes 0 we get

$t_i = t_0 + \pi$. For the "only if" part of the second statement assume that the function $Y_i = \sin X_i * (t - t_0)$ reaches 0 at $t_i = t_0 + \pi$. Then:

$$\begin{aligned} &\sin(X_i \times (t_0 + \pi - t_0) = 0 \\ &X_i \times \pi = n \times \pi \qquad n \in Z \quad \Rightarrow \quad X_i \in Z \end{aligned} \tag{A.14}$$

For the if part we have:

$$X_i \in Z \text{ and } Y_i = \sin(X_i \times (t_0 + \pi - t_0)) \quad \Rightarrow \quad Y_i = 0 \tag{A.15}$$

REFERENCES

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

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

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

4. J. Moses, Algebraic Simplification: A Guide for the Perplexed. *Communications of the ACM* 14: pp. 527-537, 1971.

5. A. C. C. Say, L'Hôpital's Filter for QSIM, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20(1): pp. 1-8, 1998.

6. A. C. C. Say, Improved Reasoning About Infinity Using Qualitative Simulation. *Computing and Informatics* 20(5): pp. 487-507, 2001.

7. A. C. C. Say and S. Kuru, Improved Filtering for the QSIM Algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15(9): pp. 967-971, 1993.

8. P. Struss, Global Filters for Qualitative Behaviors, in *Proc. AAAI-88*, Saint Paul, Minn., pp. 275-279, 1988.

9. T. Könik and A. C. C. Say, Extracting and Using Relative Duration Information in Pure Qualitative Simulation, in *Qualitative Reasoning: The Twelfth International Workshop*, AAAI Technical Report WS-98-01, AAAI Press, Menlo Park, California., pp. 155-160, 1998.

10. P. Struss, Problems of Interval-Based Qualitative Reasoning, in D. S. Weld and J. de Kleer, eds. *Readings in Qualitative Reasoning About Physical Systems.* San Mateo, California: Morgan Kaufmann, pp. 288-305, 1990.

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

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

13. *http://en.wikipedia.org/wiki/Cramer%27s_rule.*