

BLIND SPOTS OF QUALITATIVE SIMULATORS

by

Nuri Taşdemir

BS, in Electrical and Electronics Engineering, Bilkent 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

2007

BLIND SPOTS OF QUALITATIVE SIMULATORS

APPROVED BY:

Prof. A.C. Cem Say

(Thesis Supervisor)

Prof. H. Levent Akin

Assist. Prof. Kıvanç Mihçak

DATE OF APPROVAL: 31.08.2007

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my thesis supervisor Prof. A. C. Cem Say for many insightful conversations during the development of the ideas in this thesis, and for the helpful comments in the text. Without his support and patience, it would be impossible for me to finish this thesis.

I would like to thank Prof. H. Levent Akın and Assist. Prof. Kıvanç Mihçak for being a part of the committee and for their support during investigation of the ideas in this thesis.

I would like to thank my mother Gülten Taşdemir and my sisters Berna and Asiye Berrin, for their love, patience and understanding. They gave me their love and support throughout all stages of my education.

I am grateful to all my friends from Boğaziçi University, especially Gül Çalıklı, Yunus Dönmez, Barış Gökçe, Abuzer Yakaryılmaz and Birkan Yılmaz. Last but not least I am grateful to Reyhan Aydoğan for being by my side whenever I am in need.

Finally, I would like to thank all my teachers, who have shared their knowledge and ideas with me over the years, and enlightened my path. I especially thank my my mathematics teacher Perihan Özdingiş from İstanbul Atatürk High School of Science.

ABSTRACT

BLIND SPOTS OF QUALITATIVE SIMULATORS

Qualitative simulators are important tools for analyzing the possible behaviors of a dynamical system. The technique of qualitative simulation has some theoretical limitations. For some input system models, a qualitative simulator either predicts spurious (impossible) behaviors or does not predict some possible behaviors. It has been shown that a “perfect” qualitative simulator cannot be built, because there are some spurious behaviors which cannot be proven to be spurious.

This thesis questions the possibility of building a qualitative simulator which can detect and eliminate all spurious behaviors which can be proven to be so. We find the answer to be negative. For each qualitative simulator which possesses some reasonable properties, there is an efficiently constructible, provably inconsistent input which cannot be rejected by the simulator in question.

In addition, it is shown that there exist spurious behaviors which require exponential time to detect.

ÖZET

NİTEL BENZETİMCİLERİN KÖR NOKTALARI

Nitel benzetimciler, dinamik sistemlerin olası davranışlarının analizi konusunda önemli araçlardır. Nitel benzetim yönteminin bazı kuramsal sınırları mevcuttur. Bir nitel benzetimci, kimi girdiler için ya yanlış tahminler vermekte ya da bazı olası davranışları göstermemektedir. Gösterilmiştir ki, mükemmel bir nitel benzetimci yapılamaz. Çünkü bazı yanlış tahminlerin yanlış olduğu ispatlanamamaktadır.

Bu tez, yanlışlığı ispatlanabilen bütün yanlış tahminleri fark edip eleyebilen bir nitel benzetimcinin yapılabiliğini sorgulamaktadır. Olumsuz bir sonuca ulaşılmıştır. Kimi makul özellikleri taşıyan herhangi bir nitel benzetimci için bu nitel benzetimci tarafından elenemeyen yanlışlığı ispatlanabilir bir yanlış tahmine neden olan ve sistematik bir şekilde elde edilebilen bir girdi mevcuttur.

Bunun yanı sıra, gösterilmiştir ki yanlışlığının ispatlanması üstel zaman gerektiren yanlış tahminler de bulunmaktadır.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
ÖZET	v
LIST OF FIGURES	viii
LIST OF TABLES	ix
LIST OF SYMBOLS/ABBREVIATIONS	x
1. INTRODUCTION	1
2. BACKGROUND	4
2.1. Recursion Theorem	4
2.2. Gödel’s Incompleteness Theorem	5
2.3. Qualitative Simulation	7
2.4. Exact Representation of Integers in Qualitative Models	12
2.5. Qualitative Simulators as General-Purpose Computers	13
2.5.1. Unlimited Register Machine	13
2.5.2. Qualitative Simulators can Simulate URMs	14
2.6. Artificial Spurious Behaviors	15
2.7. The Halting Problem is Reducible to Hilbert’s Tenth Problem	16
2.8. Hilbert’s Tenth Problem is Reducible to Qualitative Simulator Input Consistency Checking	16
3. BLIND SPOTS	18
4. STATES CHECKABLE WITH HIGH COST	22
5. CONCLUSION	25
APPENDIX A: BLIND SPOT INPUTS	26
A.1. Construction of Blind Spot Inputs	26
A.1.1. Construction of Multiple Region Blind Spot Input	26
A.1.2. Construction of Single Region Blind Spot Input	27
A.2. Blind Spots are Inconsistent	28
A.2.1. Blind Spot with Multiple Region	28
A.2.2. Blind Spot with Single Region	29

A.3. Inconsistent Blind Spot Inputs	30
A.4. Blind Spots are Inconsistent	31
REFERENCES	33

LIST OF FIGURES

Figure 2.1.	Constraint set in order to obtain “nine”	13
Figure 2.2.	URM program computing $f(x, y) = x + y$	14

LIST OF TABLES

Table 2.1.	Qualitative constraint types	8
Table 2.2.	URM instructions	14

LIST OF SYMBOLS/ABBREVIATIONS

$?$	Qualitative notation for unknown values
\in	Is an element of
\notin	Is not an element of
\exists	There exists a/an
∞	Infinity
d/dt	Qualitative constraint derivative
f	A function
M	A qualitative simulator input
$ M $	Length of M
M^+	Qualitative constraint monotonic increasing function
M^-	Qualitative constraint monotonic decreasing function
M_Q	Blind spot input of Q
$\langle M_Q \rangle$	Encoding of M_Q
\mathcal{N}	Set of natural numbers
$O(f)$	Set of functions asymptotically equal or smaller than f
p	A proposition
Q	A qualitative simulator
T	A Turing machine
U	An unlimited register machine
Σ	Alphabet
$\Omega(f)$	Set of functions asymptotically equal or bigger than f
EXPTIME	Set of languages which can be recognized in exponential time
ODE	Ordinary differential equation
QDE	Qualitative differential equation
qdir	qualitative direction
qmag	qualitative magnitude
QSIM	Qualitative simulator

TM	Turing machine
URM	Unlimited register machine

1. INTRODUCTION

Ordinary differential equation (ODE) models are heavily used in many areas of science and engineering for the analysis of dynamical systems. The full specification of an ODE requires quantitative information, like the exact shapes of functional dependencies, or precise numerical values of constants, which is generally not available. In some cases, such a precise specification is not even desirable, as when a designer only has a rough idea about the general structure of the system under consideration, and wants to analyze the possible behaviors of the constructs that would result for all possible settings of the above-mentioned parameters “in one shot”. *Qualitative reasoning* is the branch of artificial intelligence, which is based on a “weak” representation for dynamical systems, that aims to realize reasoning tools which would be useful in contexts such as the one described above.

In this thesis, we will examine some theoretical limitations of an important qualitative reasoning technique, namely, qualitative simulation. The task of a qualitative simulator is to compute all the qualitatively distinct possible behaviors of a system whose model and initial state description are given using a weak language where, for instance, an initial value may be described as merely “positive”, rather than by a real constant, and a functional relationship may be just stated to be “monotonically increasing”, without a quantitative formula.

An explanation of the qualitative simulation framework that we will examine will be given in detail in Section 2.3, but we can state now that a qualitative simulator receives as input a description of the values of the system variables that are real-valued functions of time at the initial moment of simulation, and the model of the system in the form of a qualitative differential equation (QDE), which is basically a collection of time-independent constraints specified among the values of the variables, and produces a tree, each of whose paths starting from the root, which corresponds to the initial state, corresponds to a predicted behavior of the system, as output. The behaviors are also expressed in the qualitative language, describing changes that the

system variables undergo only in terms of the signs of their first derivatives such as “Variable X increases in this interval”; they do not contain quantitative information. A QDE usually corresponds to infinitely many ODE’s, because of the relationship of the qualitative representation with that of the real numbers. Similarly, a single behavior seen in the qualitative simulator output may represent the solutions of infinitely many related ODE’s.

There are two important and desirable theoretical properties that have been studied for many years in the context of qualitative simulation; namely, soundness and completeness. A qualitative simulator is defined to be *sound*, if for all possible inputs, the behavior tree it outputs contains the qualitative counterparts of the solutions of all ODE’s which correspond to the QDE in the input. And a qualitative simulator is said to be *complete* if, for all possible inputs, all the qualitative behaviors in the output match a solution of an ODE which corresponds to the input QDE. In [1], it has been shown that no qualitative simulator can possess both these properties.

An input is consistent if and only if it could cause the prediction of at least one behavior on a hypothetical sound and complete qualitative simulator. Note that good qualitative simulators are supposed to produce an empty tree in response to an inconsistent input.

We define two new properties for qualitative simulators:

- *Steadfastness*: A qualitative simulator is *steadfast* if it does not prune the behavior tree which it has outputted at any point in its execution.
- *Responsiveness*: A qualitative simulator is *responsive*, if it starts printing a non-empty output within a finite amount of time after it starts running.

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

In this thesis, we show that, every sound, responsive and steadfast qualitative simulator has an effectively constructible and provably inconsistent input, which the given simulator cannot label as inconsistent. We call this constructed input the “*blind spot*” of the simulator. As a result of this, it can be shown that no sound, responsive and steadfast qualitative simulator can detect all provably spurious behaviors. Also it is shown that, there are input models, which cause spurious behaviors that can only be detected with an intractably high time complexity.

2. BACKGROUND

2.1. Recursion Theorem

Theorem 1. *Recursion Theorem:* *Let P be a program that computes a function $t : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$. There is a program R that computes a function $r : \Sigma^* \rightarrow \Sigma^*$, where for every w , $r(w) = t(\langle R \rangle, w)$.*

This theorem [2], which is a well-known fact of computability theory, enables us to construct programs that have the ability of reproducing their own code and processing the generated code when it is needed. This generated code can be held in a variable of the program. By applying this technique, there is no more need to supply the code of the program as an input to the program when necessary, since it can reproduce its own code in runtime. Assume that we have a program called *Main*, which requires its own code as an input in order to perform its task. By using the recursion theorem, we construct another program called *Main2* consisting of three parts; *A*, *B* and *Main*, that are executed in this order. This program has the ability of reproducing its own code, $\langle A, B, Main \rangle$. The string $\langle B, Main \rangle$ is hardcoded in *A*. When the program starts, part *A* runs first, and stores the string $\langle B, Main \rangle$ in a variable called *v*. Part *B*, which follows *A*, inherits *v* and its value. Since *A* keeps a hardcoded copy of the *B* and *Main* parts, we can see intuitively that these two parts cannot contain the hardcoded copy of *A*. As a result, *B* should find out the string $\langle A \rangle$ by itself. *B* manages this by computing the code of the partial program which would store the string *B* currently sees in variable *v* to the variable *v*. Care is taken in the original construction of *A* to make sure that its structure matches the string that *B* would compute when working with the variable *v* containing $\langle B, Main \rangle$. After generating the partial program (*A*) in this manner, *B* appends $\langle A \rangle$ to the current content of the variable *v*, $\langle B, Main \rangle$. At this time, we have $\langle A, B, Main \rangle$ in variable *v*, which represents the code of the program *Main2* itself. After this process, the variable *v* is passed to part *Main*. Now, part *Main* can use the code stored in variable *v* when it is necessary. It can be seen easily that *A* and *B* are used just for implementing the recursion technique described

above and the *Main* part can be used for performing any actual task that is expected to be done by a program which uses its own code without taking it as an input.

2.2. Gödel’s Incompleteness Theorem

Theorem 2. *Gödel’s Incompleteness Theorem:* *For any sound (consistent) formal, computably enumerable theory that proves basic arithmetical truths, an arithmetical statement that is true, but not provable in the theory, can be constructed.*

Gödel’s famous incompleteness theorem is about *formal systems*. A formal system consists of a set of *axioms* and *inference rules* in order to obtain theorems from these axioms. Axioms are unconditional truths of this system, and theorems are the provably true *sentences*. Sentences are the formulas which have a truth value and can be true or false but naturally not both of them. The *proof* of a theorem is a sequence of statements ending with the theorem itself, such that each statement in the sequence is either an axiom, or can be obtained by applying inference rules on the previous statements.

A formal system must be associated with a *model* to give “meaning” to the sentences. A model contains a set and relations defined on the elements of this set. For instance, $(\mathcal{N}, +, \times)$ is a model, where \mathcal{N} is the set of the natural numbers, and $+$ and \times are the usual ternary addition and multiplication relations on this set. Gödel’s incompleteness theorem states that no formal system, which is strong enough, can be both sound and complete. Note that the terms “soundness” and “completeness” have somewhat different meanings here, and care should be taken not to confuse them with their counterparts in the qualitative simulation terminology. A formal system is sound if and only if every provable sentence is true. In other words a false sentence should be unprovable. And a formal system is complete if and only if all the true sentences can be proven.

Before the basics of Gödel’s proof, it is enlightening to review the “Richard Paradox” [3]. The Richard paradox is about the arithmetic properties of natural numbers.

For instance, list all the English predicates which indicate an arithmetic property of a number such as “is a primitive number” or “is a multiple of 7”. Every such predicate has a specific rank in a lexicographic order. Therefore there is a *property* predicate for each positive integer. Now, we will make a definition. Number x is *Richardian* if it does not have the property in rank x of the property list. So “*is a Richardian*” also has a rank, say r . In this point, the key question is whether r is a Richardian number or not. If r is Richardian, by definition it should not have the property in that rank, but then it is not Richardian. On the other hand, if r is not Richardian, due to the definition of Richardian, it should have the property in that rank, and then it is Richardian. It looks like r is neither Richardian nor non-Richardian. This looks like a paradox, but it is not. The point is that being Richardian or not is not an arithmetic property. It is an “out of the system” property; technically, it is a *meta* property. Gödel used the same trick but he subtly pulled this key meta-property inside the system.

The key sentence in the proof of Gödel is “This sentence cannot be proven (in this system)”. Gödel showed that an arithmetical sentence can be written which has this meaning. In order to achieve this, he first used a methodology which is now known as Gödel numbering. This methodology is basically a one-to-one computable function which maps a formula to an integer. In other words every formula has a unique Gödel number. Then he showed that there exists an arithmetic formula d with two free variables, say x and y , which becomes a true sentence if and only if x is the Gödel number of a proof for the sentence whose Gödel number is y . From this point, Gödel showed that there exists an arithmetic sentence p with the meaning “This sentence cannot be proven (in this system)”. All details can be found in [3].

The important part of the proof which is related with the work in this thesis is concluding that there is a true sentence which cannot be proven in this system. We now analyze the truth value of p . As a sentence p can be true or false, not both of them. If we assume that p , which is “This sentence cannot be proven (in this system)”, is false, then p can be proven, which is the opposite of what p says, since p is false. But, p is a false sentence and in a sound system, false sentences cannot be proven. So the case where p is false is not possible due to the contradiction. Then the only remaining

option is that p is true. So p cannot be proven, but at the same time p is true. There is no contradiction in this case, so we conclude that there exists a true sentence which cannot be proven in the system. This logical deduction will play an important role in our proofs later on.

2.3. Qualitative Simulation

We base our description of the representational language of qualitative simulation on Kuipers' QSIM algorithm [4], which is the most popular and mathematically sophisticated qualitative simulator in the literature. It should be noted however, that in this thesis we use the term "qualitative simulator" for all possible qualitative simulation algorithms which have the same input/output vocabulary as Kuipers' QSIM. We will use the term "Kuipers' QSIM" when we specifically mean the latest version of that algorithm, and not a generic one.

In QSIM, the system model is given as input to the simulator in the form of a set of qualitative differential equations (QDE's). These equations are built of system variables and constraints which show the time-independent relations that are forced on these variables by the "laws" of the system. The *variables* are continuously differentiable functions of time, and are restricted to have only finitely many critical points in any bounded interval. Every variable has a *quantity space*, which is a totally ordered set of landmark values. A *landmark* represents an extended real number, but only the default landmarks $-\infty$, 0 and ∞ are explicitly associated with specific values by the modeling language. The user may include as many new landmarks as necessary for each variable. For instance; if a variable has $(0, max, \infty)$ as its quantity space, this means that this variable can only attain nonnegative values, and *max* is a positive real number which is important to the user's understanding of the system, so that the simulator is supposed to distinguish cases where this variable's value is at, below or above *max* in any description of the system's behavior.

In each segment of the behavior descriptions, each variable value is represented by the pair $\langle qmag, qdir \rangle$, where *qmag* denotes *qualitative magnitude* and *qdir* denotes

qualitative direction. This pair is called the *qualitative value* of the variable. The qualitative magnitude shows the location of the variable in its quantity space. A variable can be exactly on a landmark or between two landmarks. The qualitative direction indicates the “movement” of the variable in its quantity space. It is defined to be the sign of the time derivative of the variable. It can be *increasing* (*inc* or $+$), *steady* (*std* or 0) or *decreasing* (*dec* or $-$). For instance, for the variable x with a qualitative space $(0, max, \infty)$, the value $\langle(0, max), inc\rangle$ means that x is between 0 and max and increasing. A variable may have a specified initial qualitative value for the starting point (initial state) of the simulation. This initial value may be incompletely specified, so that of its qualitative magnitude and qualitative direction, only one may be known. For instance, it may be known that x is between 0 and max , but its direction may not be available.

The QDE’s can be built using seven types of *constraints*. These are *constant*, *add*, *multiplication*, *derivative*, *minus*, M^+ and M^- . These constraints are explained in Table 2.1. Each constraint invalidates some combinations of the qualitative values of the variables. For instance, in the presence of the constraint $A(t) = -B(t)$; when A has a *qdir* of *inc*, B cannot have *inc* as its *qdir*.

Table 2.1. Qualitative constraint types

Name	Representation	Explanation
<i>add</i>	$X(t) + Y(t) = Z(t)$	
<i>constant</i>	$X(t) = \text{a landmark}$	$\frac{d}{dt}X(t) = 0$
<i>derivative</i>	$\frac{d}{dt}X(t) = Y(t)$	
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
<i>minus</i>	$X(t) = -Y(t)$	
<i>mult</i>	$X(t) \times Y(t) = Z(t)$	

In any time point or interval, the set of qualitative values of all the variables is called a *state* of the system. Every state should satisfy the constraints given in the QDE, otherwise this state should be eliminated and therefore not included in the output behavior descriptions by a good simulator like Kuipers' QSIM due to inconsistency. Even if a state satisfies all the constraints, it may still be eliminated by Kuipers' QSIM if it is proven to be inconsistent by one of the *global filters*, which are subroutines which scan the behavior descriptions under construction for various different families of mathematically provable inconsistencies that can exist between them and the QDE.

QDE's may represent infinitely many ODE's, since each M^+ and M^- may correspond to infinitely many different "real" functions, and each nonzero landmark may match infinitely many reals in general. For instance, in the case of the variable x with the qualitative space $(0, max, \infty)$, max may be any positive real number if there are no constraints which invalidate some possibilities, as is the case in the advanced representational techniques that will be discussed later.

At the beginning of its execution, QSIM determines the initial values of all the variables. As mentioned earlier, the user may specify the initial values for any variable but she can keep some parts open for the simulator to fill. For instance, if the variable x is given an initial qualitative magnitude but its qualitative direction is unknown, then QSIM considers the three possible cases, in which the direction is *inc*, *std* or *dec*. Due to similar incomplete inputs about other variables, the number of possible complete initial states may even be exponential in terms of the number of variables. All complete initial states are checked whether they satisfy the constraints or not. The ones, which satisfy all the constraints form the start points of the simulation. The simulation algorithm is run independently from each start point. If there is no completion of the given incomplete input, which satisfies all the constraints, then the input is deemed to be *inconsistent*.

During simulation, for each state, Kuipers' QSIM generates a set of candidate successor states, which provably contains any state corresponding to the solution of any ODE matching the QDE, and then eliminates the ones, which do not satisfy the

constraints and the global filters. This process produces a state tree. The initial state is the root of the tree. In the case of multiple initial states, there are multiple trees. Every path starting from the root of the state tree is to be interpreted as the description of a possible behavior of the modeled family of systems when started from a point in the set corresponding to the initial state. The construction of this tree continues until the end of the simulation. The simulation is completed when all branches finish. A branch finishes in two cases: First, a behavior may end with a state which can be proven to be possible only at $t = \infty$, thus ruling out the existence of a successor. Second, a termination condition can be met. A termination condition basically corresponds to one or more variables attempting to reach values which are outside their legal ranges (subsets of quantity spaces), indicating that the input model would no longer be adequate for computing the dynamics of the system after that point. For instance, 0, corresponding to the ground level, would be a legal range boundary for the quantity space of a variable representing the height of an object in a model of free fall, since the constraints linking the relevant variables like height, velocity, etc. are no longer valid after we have seen $height = \langle 0, dec \rangle$, that is, the object has hit the ground. Some simulations may not halt or take too much time to halt. That is the reason for simulators starting to output the behavior tree when it is still incomplete most of the time.

An input model may contain several QDE's. Each QDE corresponds to a different operating region, in which some aspects of the dynamics of the system are different than the others. Consider extending the model mentioned in the previous paragraph by adding another QDE describing the physics of what happens when the object hits the ground. One of the operating regions is defined as the initial operating region in the simulator input. Transition conditions are associated with each operating region. When one of the transition conditions is satisfied, to continue with our running example, when $height = \langle 0, dec \rangle$, a transition to the appropriate new operating region is triggered. This means that the simulation continues with the new variables and constraints after this point. During the transition, some variables may inherit their old values, whereas some others may receive new values; making operating region transitions the only points in a qualitative simulator output where discontinuities in variable values are

allowed.

As mentioned, a QDE may represent infinitely many ODE's. By assigning real values to the landmarks and replacing M^+ or M^- with functions obeying the defining restrictions in Table 2.1, an ODE may be obtained from a QDE. When given a QDE, an ideal qualitative simulator should produce any behavior which corresponds to the solution of any ODE which is represented by the given QDE. This property is called *soundness*. Also, an ideal qualitative simulator should not produce any *spurious* behaviors, that is, behaviors that do not correspond to the solution of any ODE which can be represented by the given QDE. This property is called *completeness*. In [1], it is proven that a qualitative simulator which has the input/output vocabulary of Kuipers' QSIM cannot be both sound and complete.

We define two more properties for qualitative simulators: steadfastness and responsiveness. These are reasonable and desirable properties for qualitative simulators.

A *steadfast* qualitative simulator is one which does not retract any part of its output that it has already printed. In particular, once a steadfast qualitative simulator has printed the root of the behavior tree, corresponding to the initial system state, its output is guaranteed to contain at least one behavior prediction starting from that state. The motivation behind our explicit definition of this very reasonable and easily realizable property is the interesting fact that implementations of Kuipers' QSIM, which start printing out the behavior tree before the simulation is over, are not steadfast. This is inevitable for inputs that cause trees which are either infinite, or finite but so big that running the simulation to completion is not an option. Since inconsistency can propagate backward from the leaves to the root, Kuipers' QSIM may decide to prune a branch of the behavior tree after adding arbitrarily many states to it [4]. This is a result of the rule which states that all states, except the quiescent states and the transition states, which satisfy the operating region transition or termination conditions, should have at least one consistent successor in order to be consistent. A state which has no consistent successor state is also inconsistent even if it passes all other filters. So a state which has been added to the behavior tree may be labeled much later

as inconsistent, if all of its successor states have been labeled inconsistent. Therefore, there are inputs which QSIM may announce as inconsistent only after building and then destroying a large tree rooted at them. If the simulator does not keep such a tree in memory, but instead starts to print it out before the end of the simulation, the later announcement that the input was, after all, inconsistent constitutes a violation of steadfastness as defined above.

A *responsive* qualitative simulator starts printing a non-empty output within a finite amount of time after it starts running. Note that a responsive qualitative simulator should produce an output even if the input is inconsistent. In such a case, the simulator should print a statement to the effect that the simulation result is an empty tree.

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

2.4. Exact Representation of Integers in Qualitative Models

For any integer z , there exists a set of QSIM variable quantity spaces and constraints, from which z 's equality to a particular variable in that set can be unambiguously deduced [5]. This can be achieved easily by encoding the required value with addition and multiplication constraints. For example, if we want to express that a variable has the value nine, then we can use following structure in Figure 2.1 where all the variables are defined to be constant and *ONE* is initialized to a positive finite value.

Here, it is obvious that *ONE* equals one, and so *NINE* has to have the value nine.

ONE	$=$	$ONE \times ONE$
TWO	$=$	$ONE + ONE$
$FOUR$	$=$	$TWO \times TWO$
$EIGHT$	$=$	$FOUR \times TWO$
$NINE$	$=$	$EIGHT + ONE$

Figure 2.1. Constraint set in order to obtain “nine”

It is clear that the total length of the submodel for encoding integer n in this manner is bounded by a polynomial function of the length of the description in bits of n . In fact it is $O(n \log n)$, since $O(n)$ constraints are needed and every constraint is at $O(\log n)$ length.

2.5. Qualitative Simulators as General-Purpose Computers

As explained earlier, a qualitative simulator input includes, in addition to the QDE’s, quantity space descriptions and initial values of variables, termination conditions, and operating region transition conditions. After this point, when we use the term “qualitative simulator input”, we will mean a single string into which all of this information has been packaged. In fact, at some points in our discussion, we will think of qualitative simulators as computers, and their input as programs. In Section 2.5.2, it will be shown that qualitative simulators can be made strong enough to be used as a general purpose computation model. But first we will mention *unlimited register machines*, a computation model which is equivalent in power to the standard computation model of *Turing machines* [6].

2.5.1. Unlimited Register Machine

The unlimited register machine (URM), which is equivalent in power to the Turing machine (TM) model, is one of the many mathematical idealizations of computers [6]. A URM has finitely many registers which can store nonnegative integers. There is no upper limit for the value contained in a register. Every URM has a program which

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

Figure 2.2. URM program computing $f(x, y) = x + y$

contains an ordered list of instructions shown in Table 2.2 to be performed on the registers. When an instruction other than a *jump* is executed, the next instruction to be executed is the one right after the current one. Figure 2.2 contains the description of a simple URM, which gets two integers as input in registers r_1 and r_2 , and gives the sum of these numbers as its output in register r_1 .

Table 2.2. URM instructions

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

2.5.2. Qualitative Simulators can Simulate URMs

Yilmaz and Say [7] proved that any given URM/input pair can be simulated in a qualitative simulator which supports one of several quite restricted subsets of the input/output vocabulary of QSIM. To simulate a URM program with p instructions, one constructs a qualitative simulator input with $p + 2$ operating regions: one for each instruction, one for the initialization, and one more for the finalization of the computation. There are three corresponding states in the simulation for each line of code execution. The first state corresponds to the initialization of the new operating

region. Then in the second state, corresponding to a finite time interval, variables change continuously to reach the values they are supposed to obtain after the execution of the instruction. After this point, in the third state, variables have their final values and a transition to the operating region corresponding to the next instruction in the program flow is triggered. In all transitions, the variables which represent the URM's registers keep their values to ensure a correct simulation.

This simulation can be performed in a behavior tree with a single branch. Note that this requires some additional filters, which “decode” the input to obtain and then keep track of the exact numerical values of the simulation variables to be incorporated to presently available qualitative simulators, and nobody has seriously tried to implement the construction in [7] to our knowledge. However, an implementation is entirely possible, and in fact quite straightforward when compared with some of the mathematically much more sophisticated global filters that have been developed for QSIM as in [8, 9].

A qualitative simulator with such appropriate filter mechanisms can be thought of as an alternative computational model like the URM, and qualitative simulator inputs play the roles of the programs to run on this computational model. It follows from the computational universality of URM's, that a qualitative simulator which supports one of the subsets of the QSIM input vocabulary listed in [7] can simulate any other qualitative simulator implementation.

2.6. Artificial Spurious Behaviors

Normally, users of qualitative simulators try to eliminate spurious predictions as much as possible. But when qualitative simulator is viewed as a computer, they can be pretty useful. They provide a functionality similar to that of *exceptions* in a programming language.

The way to produce a controlled spurious prediction is, as explained in [7], to keep a shared constant variable, say X , which has a positive value, in all operating regions.

In every operating region transition, the value of X is inherited. To produce a spurious behavior when a specific condition is met, the system is led using the region transition mechanism to an operating region which contains another constant variable, say Z , whose value is zero. This operating region also has the constraint $Z + Z = X$. This means that zero plus zero is a positive number, which is false. So when the condition in question is satisfied, an artificial spurious behavior is created.

2.7. The Halting Problem is Reducible to Hilbert's Tenth Problem

As the name suggests, Hilbert's Tenth Problem is the tenth of 23 problems which were announced in 1900 by the famous mathematician David Hilbert as a challenge to the mathematicians of the 20th century. It asks for an algorithm for deciding whether a given multivariate polynomial with integer coefficients has integer solutions. In 1970, Yuri V. Matiyasevich showed that no such algorithm exists, by demonstrating a method which can be used to construct, for any given Turing machine T , a polynomial P with integer coefficients, such that P has a solution in the natural numbers if and only if T halts on the empty input. As mentioned above, the original statement of the problem talks about the domain of integers, rather than natural numbers. However, this can be shown to be equivalent in difficulty to the version with the domain restricted to the natural numbers; as presented in [10].

2.8. Hilbert's Tenth Problem is Reducible to Qualitative Simulator Input Consistency Checking

Yilmaz and Say [7] have proven that, even if the qualitative representation is narrowed so that only the *derivative*, *add*, *mult* and *constant* constraints can be used in QDE's, and the simulation proceeds only in a single operating region, it is still impossible to build a sound and complete qualitative simulator based on this input/output vocabulary. This proof uses a reduction from Hilbert's Tenth Problem, namely, a technique that can be used to build, for any given multivariate polynomial P with integer coefficients, a qualitative simulator input M , such that M is consistent if and only if P has a solution in the integers. This means that a sound and complete qualitative

simulator, if it existed, could be used to solve Hilbert's Tenth Problem: One would first reduce the polynomial to the corresponding qualitative simulator input, and then simply run the simulator on that input. If the input is consistent, the QDE would have at least one solution, and the first state of that behavior would be printed sooner or later. On the other hand, if the input is inconsistent, the simulator would report this sooner or later. Note the implicit assumption of responsiveness in this argument. All the user would have to do is thus to observe the simulator's output to determine the answer of the original problem. Although this proves that there can be no single-region qualitative simulator which is both sound, responsive and complete, the transformation used for this purpose in [7] can also be used fruitfully to obtaining interesting results about sound, steadfast, responsive and naturally incomplete simulators, as will be seen in the proof of Theorem 4.

3. BLIND SPOTS

In this chapter, we will prove that there cannot be a single sound, steadfast and responsive qualitative simulator which can eliminate all provably spurious behaviors. A class of inputs which lie at the heart of this problem will be demonstrated. The consistency of the input of this class includes an encoding of program Q and depends on the response given to it by Q . The input is built such that it is consistent if and only if Q finds it to be inconsistent in the first response, and it is inconsistent for all other cases.

- *Blind spot input*: A qualitative simulator input M_Q is blind spot input of a given program Q , if and only if M_Q is constructed from Q as described in Appendix A.1.
- *Blind spot*: An inconsistent blind spot input of a sound, steadfast and responsive qualitative simulator is its blind spot.

Blind spot input M_Q is going to be shown to be inconsistent and yet ineradicable for a sound, steadfast and responsive simulator Q . There exists such a blind spot for each sound, steadfast and responsive simulator, and that the same input can easily be detected as inconsistent by many other simulators, lead us to give the name “blind spots” to such inputs.

The argument we use to prove the existence of blind spots in sound, steadfast and responsive qualitative simulators has been inspired by the proof of Gödel’s incompleteness theorem. The resemblance between the argument in this chapter and Gödel’s proof is a result of self reference. In our proof, the input has to be consistent if the qualitative simulator rejects it, and in Gödel’s proof, the statement has to be false if the system can be used to prove it.

Blind spot inputs can be constructed because the qualitative simulation vocabulary is powerful enough to perform some representational feats. If the modeling lan-

guage is weakened further, building a sound, steadfast and responsive qualitative simulator, which can detect every provable spurious behavior, may perhaps become possible. Even a sound and complete qualitative simulator may be possible for a sufficiently weak representation language. For this reason, we have to be careful in stating exactly which subset of the qualitative vocabulary is essential for our arguments to be valid. Theorem 3 is needed for our main result in the case of qualitative simulators which support the operating region transition feature, even if no constraints other than those of *add*, *mult* and *constant* types are allowed. In Theorem 4, we show that even the operating region transition feature is not required for the phenomenon we describe here to occur; blind spots still exist when the set of available constraint types is reduced to contain only *derivative*, *add*, *mult* and *constant*, and the model contains a single operating region [7].

Theorem 3. *Blind Spot Theorem (Multi-Region Version):* *Even if the qualitative representation is narrowed so that only the derivative, add, mult and constant constraints can be used in QDE's, every multi-region qualitative simulator Q which possesses the soundness, steadfastness, and responsiveness properties has a blind spot, which is an inconsistent input which Q can not reject.*

Proof is presented in Appendix A.2.1.

There exist other qualitative simulators which can correctly detect the inconsistency of this input and reject it: Consider, for example, a computationally universal version of QSIM, to which the “numerical” filters mentioned in Section 2.5.2, that are required for the simulation of a URM to produce a single-branch behavior tree, have been incorporated. Such a simulator will start “running” the program of the input M_Q , which in turn will simulate Q on the input M_Q , see Q accept M_Q as proven at Appendix A.2.1, and jump to the contradictory operating region, at which point the “outer” simulator will propagate the inconsistency all the way back to the initial state and reject the input M_Q . Interestingly, almost every sufficiently sophisticated qualitative simulator other than Q is capable of rejecting M_Q in this manner. It is this fact which leads us to use the term “blind spot” in the title of this chapter.

In the proof of Theorem 3, the operating region transition feature of the QSIM vocabulary played a critical role, since it is due to that representational item that URM's can be modeled. Now, we show that the blind spots demonstrated in Theorem 3 persist even when the operating region transition feature is excluded from the qualitative simulation vocabulary, and only *derivative*, *add*, *mult* and *constant* constraints are allowed in the single QDE in the input.

Theorem 4. *Blind Spot Theorem (Single-Region Version):* *Even if the qualitative representation is narrowed so that only the derivative, add, mult and constant constraints can be used in QDE's, and the simulation proceeds only in a single operating region, every qualitative simulator Q which possesses the soundness, steadfastness, and responsiveness properties has a blind spot, which is an inconsistent input which Q cannot reject.*

Proof is presented in Appendix A.2.2.

Lemma 1. *A blind spot input M_Q is inconsistent if and only if program Q does not reject the M_Q as inconsistent in its first response.*

Proof is presented in Appendix A.3.

Theorem 5. *Every sound, steadfast and responsive qualitative simulator has a provably inconsistent blind spot input.*

Proof is presented in Appendix A.4.

Another confusing comment is about the case which occurs when someone tries to build a sound, steadfast, and responsive qualitative simulator which is also capable of obtaining its own code. By including a global filter which uses the blind spot theorem to check whether its input is of the problematic type to such a qualitative

simulator, inputs built using the blind spot trick can be caught. If that input is announced as inconsistent, this means that this qualitative simulator fails to be sound, steadfast and responsive, since, due to Theorem 3 and Theorem 4, a sound, steadfast and responsive simulator should accept its blind spot input in order to be sound. The other case is that the designer of the qualitative simulator makes it so that this input is accepted as consistent. In this case, if the designed qualitative simulator is successfully sound, steadfast and responsive as the designer tries to achieve, the blind spot input which is an inconsistent input is announced as consistent. Therefore a designer should accept such an input as consistent in order to be able to build a sound, steadfast and responsive simulator knowing all the while that if she succeeds this input will in fact be inconsistent.

4. STATES CHECKABLE WITH HIGH COST

In Section 2.5.2, it is shown that a qualitative simulator can simulate a URM. Now, we will use this fact to construct a qualitative simulator input whose consistency requires $2^{\Omega(n^k)}$ time to be detected, where n is the size of the input and $k > 0$. But this same proof idea can be used to show the existence of qualitative simulator inputs proving whose inconsistency requires arbitrarily more time than this.

Theorem 6. *There are spurious behaviors which requires exponential time to be detected.*

Proof. Consider any EXPTIME-complete language A . The fastest algorithm which decides A has superpolynomial time complexity [2]. Since A is decidable, a URM which decides it exists, call this URM U .

We will use a modified version of the technique in Section 2.5.2, to encode U and its input as an input for a qualitative simulator Q . The only difference from that section will be the number of finalization operating regions. We need two separate finalization operating regions. Since Q will simulate a decider, one of the regions will stand for *yes*, while the other will stand for *no*. Using the technique described in Section 2.6, the *no* operating region is set up to contain an inconsistency with regard to the other operating regions, so if the simulation reaches the *no* operating region, this will result in an easily detectable spurious behavior, and since there will be at most one possibly consistent simulation branch, the input will be inconsistent in this case. The *yes* operating region does not contain an inconsistency, and therefore in case of reaching there, the simulation will output a single nonempty behavior successfully.

Now, we construct a Turing machine T for deciding A . T reads its input string x , and uses the technique described in the previous paragraph to construct a qualitative simulator input M , which encodes the URM U working on input x , and then simulates

Q on input M until Q gives its response to the initial state. If Q prints the initial state, this means that $x \in A$, and T prints *yes*; if Q rejects the input M due to inconsistency, this means $x \notin A$, and T prints *no*. Note that this construction is guaranteed to be valid only if Q is steadfast.

We assume the length of x is n and calculate how fast the fastest possible qualitative simulator Q can respond to M in this scenario. The length of the input M of the qualitative simulator is $O(n \log n)$, since the only operating region of M whose size depends on n is the starting region, where the value x is supposed to be encoded as the initial value of U 's first register, and this can be done using a set of constraints that can be expressed in $O(n \log n)$ symbols, as described in Section 2.4. All the other operating regions have fixed lengths that do not depend on x . So $|M|$ is $O(n \log n)$. Now, we know that for some values of x , the fastest possible T will have to run for $2^{\Omega(n^k)}$ steps. If one leaves the simulation of Q aside, it is clear that the remaining parts of T have a total runtime of $O(n \log n)$, which only consist of the preparation of the input for Q and giving output according to the response of Q . Since the total time is $2^{\Omega(n^k)}$, this concludes that time required for Q should also be $2^{\Omega(n^k)}$. On the other hand, $|M|$ is $O(n \log n)$, which means $|M| = O(n^2)$ is also true. Then $n^2 = \Omega(|M|)$. Therefore Q is seen to require $2^{\Omega((n^2)^{k/2})} = 2^{\Omega((\Omega(|M|))^{k/2})} = 2^{\Omega(|M|^{k/2})} = 2^{\Omega(|M|^l)}$ steps, and since $k/2 = l > 0$, that is an exponential amount of time in terms of the size of its own input, to decide about the consistency of its initial state. \square

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

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

There exists an infinite hierarchy of languages which require worse and worse runtimes than those in EXPTIME [2]. All these can be used to demonstrate the existence of spurious behaviors which are eradicable in principle, but ineradicable in practice, by the same argument as above.

Kuipers [11] has provided a simple proof that there exists a qualitative simulation input M which would cause an output tree whose size is exponential in terms of the size of M , and therefore qualitative simulation has exponential worst-case runtime. The difference of our contribution is the fact that it is about the complexity of the consistency checking of a single state in a tree whose excessive size is not a result of branching (the “tree” is in fact a linear chain in this case) but of the length of the single branch.

5. CONCLUSION

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

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

APPENDIX A: BLIND SPOT INPUTS

A.1. Construction of Blind Spot Inputs

A.1.1. Construction of Multiple Region Blind Spot Input

We start by observing that qualitative simulator inputs can be designed to use a simple adaptation of the recursion technique of Section 2.1 to obtain and store their own code in a simulation variable. Such an input will consist of three submodels: $A, B, Main$. A , which consists of a single operating region, will contain a variable V , which it initializes to an integer encoding the string $\langle B, Main \rangle$. This is possible with the techniques explained in Section 2.4. Another variable in A is constrained to reach a landmark which will trigger a transition to the starting operating region of the multiple-region submodel B . Variable V inherits its value during all operating region transitions. B models a URM which uses its knowledge of the value in V to prepare the description of a qualitative input submodel, which models a URM that initializes variable V to the value B now sees in V , and then triggers a transition to the starting operating region of B . Note that this submodel description prepared by B is none other than $\langle A \rangle$. B then combines $\langle A \rangle$ and $\langle B, Main \rangle$ to obtain $\langle A, B, Main \rangle$, stores this value in V , and triggers a transition to $Main$, where the description of the entire input $\langle A, B, Main \rangle$ can be used as needed.

We now note that, given any program C , one can build a qualitative simulator input M_C as follows: M_C contains the representation of a URM program. Upon starting execution, M_C first acquires its own code $\langle M_C \rangle$ using the recursion technique described above, and then starts to simulate C , whose code has been embedded in that of the program of M_C , with $\langle M_C \rangle$ as input. The simulation of C is performed until C gives its response about the input, i.e. until C either declares inconsistency or prints something else which is possibly irrelevant such as an output of “Hello world!”. If C rejects the initial state of $\langle M_C \rangle$, the program of M_C ends by arriving at an operating region where a variable increases until it reaches a bound of its legal range, constituting a successful

termination of the corresponding branch of the behavior tree, meaning that $\langle M_C \rangle$ was a consistent input. On the other hand, if C prints something else, the program of M_C jumps to an instruction represented by the operating region OR_C , which causes a contradiction, using the method described in Section 2.6.

This completes the construction of the multi region blind spot input for a program.

A.1.2. Construction of Single Region Blind Spot Input

We first construct a Turing machine named T . T starts to simulate Q 's simulation of an input M_Q , whose preparation will be described shortly. On the first response of Q , T stops the simulation. If Q rejects its input, T halts; otherwise, T loops forever.

T prepares the input M_Q , which it feeds to Q , as follows: T first obtains its own code $\langle T \rangle$ by recursion, and then it sets up a polynomial D , which has a solution if and only if T halts with the empty string as input, T then encodes D as a qualitative simulator input M_Q using the technique described in [7] in Section 2.8 setting the initial magnitudes of all the simulation variables representing the polynomial variables, which are constant throughout the simulation in the construction of [7] to $(0, \infty)$.

The transformation used by T to encode its own halting status in a multivariate polynomial is realized in two stages. T first employs the techniques of [10] in Section 2.7 to produce a polynomial D_1 defined on the domain \mathcal{N} including zero. Since we will specify to the simulator that we are looking for a solution where all the polynomial variables are positive, as mentioned in the previous paragraph, what we really want here is a polynomial which has positive roots if T halts. So T transforms D_1 to another polynomial D in the following manner.

Given a polynomial $D_1(x_1, x_2, \dots, x_n)$, which is defined on \mathcal{N} , we will build a new polynomial D , which has a solution in the positive integers, if and only if $D_1(x_1, x_2, \dots, x_n)$ has a solution in the set of natural numbers. D is the product of

all the variations of D_1 . By a *variation* of D_1 , we mean a polynomial which can be obtained by setting some of the variables of D_1 directly to zero. Since there are two possibilities (zero or not) for each variable, there are 2^n variations of D_1 . $D = \prod_{i \in \{0,1\}^n} D_i$, where $D_i = D_1(i_1 \times x_1, i_2 \times x_2 \dots i_n \times x_n)$ and i_j is the j^{th} character of the string i , i.e. 0 or 1.

At the end, when T is executed M_Q , which is the single region blind spot input of Q , is obtained during runtime.

A.2. Blind Spots are Inconsistent

A.2.1. Blind Spot with Multiple Region

In this part, we give the proof of Theorem 3. This theorem states that even if the qualitative representation is narrowed so that only the derivative, add, mult and constant constraints can be used in QDE's, every multi-region qualitative simulator Q , which possesses the soundness, steadfastness, and responsiveness properties has a blind spot, which is an inconsistent input which Q can not reject.

We assume that Q is a sound, steadfast and responsive qualitative simulator. We claim that the input M_Q is inconsistent, and yet Q does not reject this input; it starts printing a provably spurious prediction that begins with the initial state of M_Q . We justify this claim with the following analysis of the execution of Q on input M_Q : To prevent confusion, we assume that Q_0 denote the “outer” Q , and Q_1 denote the “inner” Q , which will be simulated as described above by the program M_Q . Since Q_0 and Q_1 are implementations of the same qualitative simulator which are working on the same input (M_Q), their actions will be exactly the same. There are two possibilities for the response of Q to the input M_Q : Q either rejects M_Q , or prints out the initial state of M_Q . We first analyze the case where Q_0 rejects M_Q . Then Q_1 will also reject M_Q . But now consider what the program described by M_Q does: It simulates Q_1 for a finite number of steps to see how Q_1 responds to M_Q , and when it sees a rejection, it terminates successfully, without reaching a contradictory state. This is a perfectly

valid behavior of the described system, and should of course be printed out by any sound qualitative simulator. Since Q is sound, we conclude from this argument that it cannot reject M_Q .

The remaining possibility is that both Q_0 and Q_1 will print out the initial state of M_Q . Since Q is steadfast, printing the initial state is an irreversible action, and means that Q announces the input M_Q to be consistent. We consider what the program of M_Q does in this case: It simulates Q_1 for a finite number of steps, and when it sees Q_1 print out the initial state of M_Q , it jumps to a contradictory operating region, making the branch of the behavior tree describing its entire execution a spurious one. Since the model is so constrained that no other nonspurious branches are possible, as explained in Section 2.5.2, we conclude that M_Q is, after all, inconsistent. By the argument of the previous paragraph, Q must announce this inconsistent input to be consistent.

The model in question can be constructed using QDE's that only use the *add* and *constant* types, if an alternative and less efficient technique of integer encoding and the "routing" of branches, which do not correspond to correct computation paths to the contradictory operating region OR_Q is used [7].

A.2.2. Blind Spot with Single Region

In this part, we give the proof of Theorem 4. This theorem states that even if the qualitative representation is narrowed so that only the derivative, add, mult and constant constraints can be used in QDE's, and the simulation proceeds only in a single operating region, every qualitative simulator Q , which possesses the soundness, steadfastness, and responsiveness properties has a blind spot, which is an inconsistent input which Q cannot reject.

Now, we assume that Q is a sound, steadfast and responsive qualitative simulator which works with the restricted vocabulary described above. We will demonstrate that there exists an input which is inconsistent but which is announced to be consistent by Q .

Having described T , we immediately proceed to our proof. Q can either find the input M_Q inconsistent, or it can start to print out the initial state. We are going to analyze these cases.

Assume that Q says that M_Q is inconsistent, and that therefore T halts. But if T halts, then D has a solution, and M_Q is consistent. Thus, Q has incorrectly rejected a consistent behavior. Since Q is sound, this is impossible, therefore Q cannot reject M_Q . Thus, Q accepts M_Q . However T is a TM that does not halt, meaning that D has no solution, and that M_Q is inconsistent. Thus, at the end Q accepts an inconsistent model.

A.3. Inconsistent Blind Spot Inputs

In this part, we give the proof of Lemma 1. This lemma states that a blind spot input M_Q is inconsistent if and only if program Q does not reject the M_Q as inconsistent in its first response.

There are two blind spot inputs for a program Q , where Q may be any program, i.e. Q is not restricted to be a qualitative simulator: Its multiple operating region blind spot input and its single region blind spot input. We analyze these cases separately.

First, consider the multi region blind spot input M_Q constructed as explained in Appendix A.1.1 for program Q , recall that now Q is any program. By the construction of M_Q , if Q prints the initial state of M_Q or something else, an artificial spurious behavior is created. The only possible consistent behavior of M_Q is reachable when Q 's first response is to reject M_Q . This proves the statement above for this case.

In the second case, M_Q is single region blind input. M_Q is constructed as explained in Appendix A.1.2 for Q . If Q prints the initial state of M_Q or something else as a first response, then M_Q is inconsistent, as follows: We analyze the construction of the input in Appendix A.1.2. M_Q is constructed such that it is inconsistent if and only if the polynomial D , which is used to build M_Q , has no solution. On the other

hand, D is obtained from a Turing machine T and D has no solution if and only if T never halts. And if we examine the halting condition of T , T never halts if and only if Q prints the initial state of M_Q . Therefore M_Q is inconsistent if and only if Q prints the initial state of M_Q . And this completes the proof of Lemma 1.

A.4. Blind Spots are Inconsistent

In this part, we give the proof of Theorem 5. This theorem states that every sound, steadfast and responsive qualitative simulator has a provably inconsistent blind spot input.

What distinguishes this theorem from Theorem 3 and Theorem 4 is the fact that it does not assume the simulator Q , which corresponds to the blind spot in question, to be given to the prover. The method we use in this proof can be used to show that there exist other qualitative simulators which could “catch” blind spot inputs that have been constructed as described in Appendix A.1.

We will show that for any sound, responsive and steadfast qualitative simulator Q , the inconsistency of the input M_Q can be proven.

Given an input M , first check whether M is single region input or multi region input. Then consider all programs in lexicographic order, and produce the blind spot input of each program with using the appropriate technique in Appendix A.1. Compare M to the blind spot inputs you generate.

If M is indeed an input in the form of such a blind spot input, this procedure is guaranteed to give Q , since we will have a match, but otherwise this process will not halt, since there is no terminating condition of the procedure in the case of mismatch. But our aim is only to show that there is a proof for the inconsistency of blind spots. So if the input is not in the form of a blind spot, there is nothing we have to do. Now, we have Q and M . And M is the blind spot input of Q . From this point on, we will use M_Q instead of M in the proof.

At this point, we have a program Q and its blind spot input M_Q in hand. We do not know about the Q or the properties that it possesses, but this is not important, as we will see soon. We now run Q with its blind spot input M_Q . During the execution, if Q prints the initial state of M_Q or something else, we can say that M_Q is inconsistent due to Lemma 1. And this is independent of the properties of Q . Recall that our aim is to prove the inconsistency when Q is sound, responsive and steadfast. And due to Theorem 3 and Theorem 4, we can say that Q will print the initial state of M_Q if it has the properties. Therefore we can conclude that M_Q is inconsistent. This procedure, which shows the inconsistency of the M_Q for sound, responsive and steadfast qualitative simulator Q , may loop forever for qualitative simulators which does not possess the properties.

REFERENCES

1. Say, A. C. C. and H. L. Akin, “Sound and Complete Qualitative Simulation is Impossible”, *Artificial Intelligence*, Vol. 149, No. 2, pp. 251–266, 2003.
2. Sipser, M., *Introduction to the Theory of Computation*. International Thomson Publishing, 1996.
3. Nagel, E. and J. R. Newman, *Gödel’s Proof*, Routledge & Kegan Paul, London, 1959.
4. Kuipers, B., *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*, MIT Press, Cambridge, MA, USA, 1994.
5. Say, A. C. C., “Numbers Representable in Pure QSIM”, In *Proceedings of Eleventh International Workshop on Qualitative Reasoning*, Cortona, 3–6 June 1997, pp. 337–344, 1997.
6. Cutland, N. J., *Computability: An Introduction to Recursive Function Theory*, Cambridge University Press, Cambridge, UK, 1980.
7. Yılmaz, Ö. and A. C. C. Say, “Causes of Ineradicable Spurious Predictions in Qualitative Simulation”, *Journal of Artificial Intelligence Research*, Vol. 27, pp. 551–575, 2006.
8. Say, A. C. C., “L’hopital’s Filter for QSIM”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 20, No. 1, pp. 1–8, 1998.
9. Say, A. C. C. and S. Kuru, “Improved Filtering for the QSIM Algorithm”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15, No. 9, pp. 967–971, 1993.
10. Matiyasevich, Y. V., *Hilbert’s Tenth Problem*, MIT Press, Cambridge, MA, USA,

1993.

11. Kuipers, B., “Qualitative Simulation”, *Artificial Intelligence*, Vol. 29, pp. 289–338, 1986.