

Understanding Unexpected Model Behaviors

SAIC UVA Scholars Research Stipend Proposal

Ross J. Gore
University of Virginia
Department of Computer Science

October 14, 2008

1 INTRODUCTION

Computational models are being used more and more to predict potential outcomes of systems involving human lives and costly resources. Generally when computational models are used to predict outcomes uncertainty exists about conditions affecting the system being modeled, and about the model itself. As a result, developers often experience unexpected program behaviors and must then explore whether the behaviors reflect an implementation error or an unexpected behavior of the system. This proposal addresses the development of a methodology to enable the explanation of behaviors that are unexpected at the time they are first observed. The explanation supports users in determining the validity of the unexpected model behaviors.

The ability to understand and validate unexpected model behaviors will have a significant impact on the development of models affecting millions of people and billions of dollars. Recently, the inability of researchers to explain the results of a developing computational model, Episims, has led to public policy debate. Episims models the nationwide spread of the smallpox virus under various vaccination strategies [Euba 2004]. The results of Episims show that in the event of a smallpox outbreak the number of infections and death under a targeted vaccination is similar to the number of infections and death under a mass vaccination. Previous established estimates of a nationwide spread of the smallpox virus had shown that a mass vaccination drastically reduced the number of infections and death compared to a targeted vaccination. The difference between these predictions has led to policy debate over "whether or not it's necessary to synthesize enough smallpox vaccine for the entire country" [Cha 2005]. The Institute of Medicine of the National Academies has published a collection of critical opinions of the predictions from Episims. The chief complaint is that the model developers cannot provide a clear explanation for the difference between their predictions under these vaccination strategies and previously established estimates [Baci 2005]. Methodology to facilitate the understanding and subsequent determination of validity of Episims' behavior is needed.

This need motivates the thesis of my proposed work: *Semi-automated search, uncertainty representation, program slicing, and causal inference procedures can be employed to provide users with more precise and richer analysis of unexpected program behavior than current tools, and this analysis will improve user understanding and the subsequent determination of validity of unexpected model behavior.*

I will apply my ongoing work on the understanding and validation of unexpected model behavior to SAIC's modeling and simulation research and development. While my proposed work will be applicable to a broader set of applications, it will also benefit SAIC's award winning Consequence Assessment Tool Set (CATS). "CATS employs a suite of hazard, casualty and damage estimation modules to estimate and analyze effects due to natural phenomena, such as hurricanes and earthquakes, and technological disasters, such as terrorist incidents, involving weapons of mass destruction, and industrial accidents. CATS depicts geographical areas of damage, probabilities and numbers of fatalities and injuries, and mitigative resource allocation. [Swia 1999]" My work will provide improved capabilities to understand and analyze the estimates and assessments from CATS.

2 RESEARCH OBJECTIVES

I have identified two research objectives that will constitute novel work in the field of modeling and simulation.

- 1. The demonstration that semi-automatic search can produce conditions of interest a user may not otherwise know how to create directly and enable users to test hypotheses about an unexpected model behavior.***
- 2. The demonstration that uncertainty representation, program slicing and causal inference procedures can be employed to provide a more precise and richer analysis of model behavior than any existing technique.***

2.1 Testing User Specified Hypotheses about Unexpected Model Behaviors

My first research objective is to demonstrate that semi-automatic search can produce conditions of interest a user may not otherwise know how to create directly and enable users to test hypotheses about unexpected model behavior. There is a difference between validating a model and validating an unexpected behavior that arises in a model. The former represents an effort to demonstrate that the model exhibits expected behavior(s) [Boeh 1984]. The latter is a demonstration of the validity of the behavior that was unexpected for a given set of conditions, or experimental frames [Zeig 2000]. Then the problem can be reframed so that the unexpected behavior becomes part of the set of behaviors considered valid. The common practice for users presented with unexpected model behaviors is to use standard debugging techniques to test hypotheses and gather insight. Debuggers provide users with the ability to modify the values of variables in an executing model but require source code level knowledge and manual modification to change the values of variables. These mechanisms do not offer a semi-automatic, higher than source code level exploration capability to test hypotheses under specified conditions.

Such an exploration capability extends a model beyond its original design and requires model adaptation. *Model adaptation* is the process of expressing new model requirements and modifying an existing model to meet the new requirements. I propose to use semi-automatic search. *Semi-automated search* is a model adaptation process that combines the use of optimization and manual modification. In this context, optimization refers to automatic objective function minimization. The objective function is met by searching across identified model input parameters and other abstractions to find a model meeting the objective function.

I will demonstrate how semi-automatic search can enable user specified hypothesis testing. Hypothesis testing will be supported by producing conditions of interest a user may not know how to create directly and allowing the user to observe the program under those conditions. The term *conditions of interest* means when an identified quantifiable condition of the program is maximized, minimized, or targeted to a specified requirement. When applied to CATS, this hypothesis testing capability will allow users to specify a casualty or damage scenario in the computer models of natural and technological hazards that they do not know how to create directly. Semi-automated search will explore the model space to find a model exhibiting the casualty or damage scenario. Then the user can then observe the model under the created scenario and observe the CATS estimates and analysis.

2.2 Precise and Rich Analysis of Unexpected Model Behavior

My second research objective is the demonstration that uncertainty representation, program slicing and causal inference procedures can be employed to provide a more precise and richer analysis of unexpected model behavior than any existing technique. *More precise* means the analysis will provide users with more possible behaviors for a fixed input to a stochastic model than any existing technique [Van 2006]. *Richer* means the analysis will include all the analysis provided by causal inference procedures and program slicing, as well as new analysis that can only be revealed by combining the two. *Program slicing* is a decomposition technique that extracts program statements relevant to a particular computation within the program [Weis 1984]. A program slice provides the answer to the question, "What program statements affect the computation of variable v at statement s ?" [Bink 1996]

An important distinction is that between static and dynamic slices. Figure 1(a) shows an example program that reads an integer input n , and computes the sum and the average of the first n positive numbers. If the sum of the first n integers is evenly divisible by n the program assigns 1 to x . Otherwise the program assigns -1 to x . The criterion for a static slice is a 2-tuple consisting of {line number of statement s , the name of variable v }, where v is the variable of interest and s is the statement of interest. Figure 1(b) shows a static slice of this program using criterion {13, x }. As shown in Figure 1(b), all computations not relevant to the final value of variable x have been "sliced away". Slices are computed by identifying consecutive sets of transitively relevant statements, according to data flow and control flow dependences [Tip 1995]. Only statically available information is used to compute slices; hence, this is referred to as a *static slice*.

<pre> 1 read(n); 2 i := 1; 3 x := 0; 4 sum := 0; 5 average := 0; 6 while i<= n 7 sum := sum + i; 8 i := i + 1; 9 end 10 if (sum mod n == 0) 11 x := 1; 12 else 13 x :=-1; 14 print (x); 15 average := sum/n; 16 print (average); </pre>	<pre> 1 read(n); 2 i := 1; 3 x := 0; 4 sum := 0; 6 while i<= n 7 sum := sum + i; 8 i := i + 1; 9 end 10 if (sum mod n == 0) 11 x := 1; 12 else 13 x :=-1; 14 print (x); </pre>	<pre> 1 read(n); 2 i := 1; 3 x := 0; 4 sum := 0; 6 while i<= n 7 sum := sum + i; 8 i := i + 1; 9 end 12 x :=-1; 13 print (x); </pre>
(a)	(b)	(c)

Figure 1: (a) An example program. (b) A static slice of the program using criterion {13, x }. (c) A dynamic slice of the program using criterion { $n = 4$, 13, x }.

In the case of dynamic program slicing, only the dependences that occur in a specific execution of the program are taken into account. A dynamic slicing criterion specifies the input, and distinguishes between different occurrences of a statement in the execution history; it consists of {input, line number of statement s , name of

variable v }. The difference between static and dynamic slicing is that dynamic slicing assumes fixed input for a program, whereas static slicing does not make assumptions regarding the input. Figure 1(c) shows a dynamic slice of the program in Figure 1(a) using the criterion $\{n = 4, 13, x\}$, where 13 denotes the first occurrence of statement 12 in the execution history of the program. Note that for input $n = 4$, the assignment $x := -1$ is executed, and the assignment $x := 1$ is not executed. The “if (sum mod $n = 0$)” branch of statement 10, and statement 11 in Figure 1(a) may be omitted from the dynamic slice because the assignment of $x := 1$ is not executed. The resulting dynamic program slice of the program in Figure 1(a) is shown in Figure 1(c).

2.2.1 Precise Unexpected Model Behavior Analysis through Uncertainty Representation

One shortcoming of program slicing is the inability to precisely analyze models with stochastic behavior. *Precision* is measured by the number of dynamic slices for a fixed input provided by the analysis divided by the number of possible dynamic slices for a fixed input [Van 2006]. My work will provide users with a means to capture more program slices for a criterion than any existing analysis technique and it will provide users with the likelihood of each behavior for the given criterion. Figures 2(a), 2(b) and 2(c) elucidate the imprecision in existing program slicing tools. From the programmer’s point of view the behavior of the example program in Figure 2(a) is stochastic. Because a static program slice only uses static information to compute the program slice the static program slice of the program in Figure 2(a) using criterion $\{7, x\}$ consists of the entire program. However, for any execution of the program at least 1 of the 7 statements will not be executed. Thus, the analysis offered by the static program slice does not accurately capture the behavior of the program for any execution. While the analysis offered by a dynamic program slice provides the user with one accurate program slice for a given input, it does not necessarily capture all of the possible program slices of a program for a given input. In this sense the analysis offered by dynamic program slicing is imprecise. Furthermore, the developer does not know the likelihood that the provided dynamic slice represents the behavior of the program. Ideally, all possible slices for a fixed input would be given and the developer would know the likelihood of a given slice for the fixed input. While Figure 2(a) is a small example of a program with stochastic behavior, it is representative of many stochastic models.

<pre> 1 read(n); 2 x = 0; 3 rand := randomNumber(0, 1); 4 if (rand >= .998 && rand <= .999) 5 x := rand + n; else 6 x := n; 7 print(x); </pre>	<pre> 1 read(n); 2 x = 0; // (rand >= .998 && // rand <= .999) == false 6 x := n; 7 print(x); </pre>	<pre> 1 read(n); 2 x = 0; 3 rand := randomNumber(0, 1); // (rand >= .998 && // rand <= .999) == true 5 x := rand + n; 7 print(x); </pre>
(a)	(b)	(c)

Figure 2: (a) A stochastic program. (b) One possible dynamic slice of the program using criterion $\{n=13,7,x\}$. (c) Another possible dynamic slice of the program using the same criterion.

Figure 2(b) and 2(c) show the two possible dynamic program slices using criterion $\{n = 13, 7, x\}$ for the program in Figure 2(a). Figure 2(b) shows the dynamic program slice, when the random number generator does not generate a number between .998 and .999. Figure 2(c) shows the dynamic program slice when the random number generator does generate a random number between .998 and .999. Assuming a uniform random number generator, the dynamic program slice of Figure 2(a) is the program slice shown in Figure 2(b) 99.9% of the time and the program slice shown in Figure 2(c) 0.1% of the time. Without additional analysis capabilities a developer using a single dynamic program slice of Figure 2(a) cannot capture all the possible behaviors for criterion $\{n = 13, 7, x\}$.

Uncertainty in model input information (*aleatory uncertainty*) and uncertainty in model design information (*epistemic uncertainty*) result in stochastic model behaviors that are often unexpected. Aleatory and epistemic uncertainty have been studied with resulting theory and algorithms to support uncertainty representation in software. *Uncertainty representation* is the first class representation in software of aleatory and epistemic uncertainty, reflected through continuous and discrete random variables [Spie 2007], [Park 2005], [Park 2006]. The *uncertain type* is the data type used for the first class representation of continuous and discrete random variables. This representation of uncertainty enables more precise analysis of models with stochastic behavior than existing tools. The analysis will provide users with more behaviors for a dynamic program slicing criterion than any existing tool and determine the likelihood of each behavior. For example, the GIS component of the CATS system

takes ranges of spatial and temporal data and converts this data to probability distributions to provide estimates for the number of persons affected, properties damaged, and amount of resources required to mitigate the destruction [Swia 1999]. The use of ranges of data and probability distributions result in stochastic estimates that cannot be precisely analyzed by existing tools. If these ranges of data and probability distributions are represented in a language with uncertainty representation, my precise analysis of program behavior would provide CATS with the capability to provide all the possible estimates and corresponding dynamic program slices for a particular scenario. It would also provide the likelihood of each estimate and each corresponding dynamic program slice. Further details on how this analysis will be designed and performed are in Section 3.

2.2.2 Rich Analysis of Model Behavior Through Causal Inference Procedures

Researchers in the medical, social science and economics communities employ causal inference procedures to identify the causal structure of deterministic and stochastic systems. These procedures use the Causal Markov Condition to produce a causal theory explaining the cause-effect relationship of the variables of interest. The Causal Markov Condition is that “a variable X is independent of every other variable except X’s effects conditional on all of X’s direct causes [Spir 2001].” A causal theory consists of a causal model and a set of parameters which specify how each variable is influenced in the causal model. A causal model is a directed acyclic graph, with a 1-1 mapping between vertices in the graph and variables of interest. A variable X is said to have a causal influence on a variable Y if and only if a directed path exists from vertex X to vertex Y in the causal model. The causal model serves as the basis for the causal theory. Each edge in the causal model is a 1-1 mapping with an element of the set of parameters associated with the causal theory. Each parameter specifies the strength of the causal influence (the probability that X has a causal influence on Y) induced by the corresponding link [Spir 2001], [Pear 2000].

The difference between the analysis produced by causal inference procedures and the analysis produced by existing program slicing tools is that a causal inference procedure produces an analysis of model behavior for all inputs of the model where each cause-effect relationship is probabilistically weighted. Due to the different nature of these types of analysis I expect they can be combined. Data from causal inference can be used to identify the strongest causal relationships in a static program slice, and program slices can identify dependencies in program statements and abstractions that causal inference procedures may not have enough data to identify. The result will be a new causal analysis of model behavior that is richer than the analysis produced by either of these techniques alone. *Richer* means the analysis will include all the analysis provided by causal inference procedures and program slicing, as well as new analysis that can only be revealed by combining the two. This richer analysis will provide CATS with the ability to determine the input variables and program statements in the computer models of natural and technological hazards which have the greatest influence on the estimates of the persons affected, properties damaged, and amount of resources required to mitigate the destruction and its aftermath.

3 RESEARCH PLAN

1. Identify existing case study models exhibiting unexpected behavior. Previous efforts of my research group have provided me with contacts in the departments of environmental science, materials science, mechanical and aerospace engineering, physics and neurological surgery at the University of Virginia. I will use these contacts along with contacts at SAIC to find developing models simulations with unexpected behaviors that will benefit from my work. Several of the case studies will be written in languages with uncertainty representation, the other case studies will be written in programming languages without uncertainty representation.

2. Demonstrate in several of the identified case studies how semi-automatic search can be used to test hypotheses about unexpected model behavior under specified conditions. When constructing a model, abstractions inevitably must be selected in order to reduce complexity, improve performance, or provide estimations for unknown information. *Abstraction opportunities* are those places where a case study collaborator can choose among possible alternatives for the abstraction. My research group has developed a language and supporting tools for a case study collaborator to identify abstraction opportunities and alternatives for model abstractions. With alternate bindings present in the source code, a model adaptation strategy employing semi-automatic search becomes possible. Semi-automatic search will be used to test case study collaborator hypotheses about an unexpected behavior by efficiently creating user specified conditions of interest. The case study collaborator will gather insight by observing an unexpected behavior under the conditions of interest. If the observed behavior matches the user’s hypothesis it will pass the hypothesis test, otherwise it will fail.

3. Identify new types of causal analysis data that can be extracted from a model written in a language with uncertainty representation. *RiskModelica* is the first programming language to represent uncertainty as a first-

class citizen. It is designed as a language extension to the programming language *Modelica*. *Modelica* is an object-oriented equation-based programming language designed for large, complex and heterogeneous physical systems [Frit 2000]. RiskModelica extends Modelica by introducing novel primitive types and *type qualifiers* to the Modelica language. A type qualifier is a form of subtyping where a supertype T is combined with a qualifier Q such that some semantic property is enforced on all instances of the subtype Q T [Fost 1999]. Type qualifiers allow the necessary semantic properties for stochastic computation to be enforced. I will demonstrate how type qualifiers can be used to identify properties of variables of uncertain type that enable more precise analysis of stochastic model behaviors. I will apply static analysis techniques aggressively to identify possible model behaviors for a fixed input in RiskModelica. When it is impossible to use static analysis, I will revert to runtime checking.

4. Produce static and dynamic program slicing tools for models written in languages with uncertainty representation. The new program slicing tools will provide the same analysis as existing program slicing tools but will also include analysis data identified in Task (3). My enhanced method for performing causal analysis of model behavior will be a superset of the analysis performed by existing tools.

5. Demonstrate in several of the identified case study programs how causal inference procedures can be used to provide causal program analysis data that cannot be provided by existing tools or the tools developed in Task (4). Recall, a causal theory consists of a causal model and a set of parameters describing the cause-effect relationship between variables of interest. When applied to the analysis of model behavior *variables of interest* can range from program variables or program statements to other identified abstractions in the model's source code. The result is causal analysis of model behavior for all inputs of the model where each cause-effect relationship of the variables of interest identified is probabilistically weighted. Static and dynamic program slicing cannot provide this type of analysis. Dynamic program slicing can only describe the causal structure of the model behavior for a fixed input, where causal inference procedures describe the causal structure of the model for all inputs. Static program slicing produces conservative analysis about dependencies between program statements. Conservative analysis determines dependencies that may exist between identified program statements; it does not determine the strength of the dependency or if the dependency exists in certain program runs. Due to the different nature of these types of analysis I expect they can be combined. Data from causal inference can be used to identify the strongest causal relationships in a static program slice, and static program slices can identify dependencies in program statements and abstractions that causal inference procedures may not have enough data to identify.

4 PLANNED PUBLICATIONS

I will target three avenues of publication for the results of this work. The first set includes journals and conferences in the modeling and simulation domain, such as ACM Transactions on Modeling and Computer Simulation (ACM TOMACS), Proceedings of International Workshop on Principles of Advanced and Distributed Simulation (PADS), Proceedings of the Winter Simulation Conference (WSC) and Virginia Modeling and Simulation (VMASC) Student Capstone Symposium. The second set includes avenues from the field of computer science, such as Journal of Programming Languages, and International Conference on Software Engineering (ICSE). My final set will be the conferences and journals of my external collaborators in their application domain.

5 CONCLUSION

Everyday public policy debates surrounding models such as Episims [Cha 2005], hinge on the ability to understand model behaviors. How can policy makers make informed decisions involving billions of dollars and millions of people in confidence when poorly understood unexpected model behaviors are pervasive? How can developers produce models of systems shrouded in uncertainty without supporting tools? Methodology to understand unexpected behaviors is needed; this motivates my work. I will develop a hypothesis testing technique by producing conditions of interest a user may not otherwise know how to create directly. This will enable user specific insight to be gained into unexpected behaviors. As a complement to the hypothesis testing, I will combine analysis from causal inference procedures and existing program slicing tools to provide developers with a richer description of the causal interactions in model behavior than is currently available. For models written in languages with uncertainty representation the tool will provide users with more behaviors for a dynamic program slicing criterion than any existing tool and determine the likelihood of each behavior. Each of these capabilities will improve the understanding and analysis of estimates from SAIC's award winning Consequence Assessment Tool Set (CATS). My approach will be evaluated with several measures of effectiveness including the measuring the usability and practicality of my work with my external collaborators through the MILCS paradigm [Shne 2006]. My advisor, Dr. Paul F. Reynolds, Jr. fully endorses this proposal.

6 REFERENCES

- [Baci 2005] Baci, A., A. Anason, K. Stratton, and B. Strom, The Smallpox Vaccination Program: Public Health in an Age of Terrorism, Institute of Medicine of the National Academies, Washington, D.C., 2005.
- [Bink 1996] Binkley, D. and K. B. Gallagher, Advances in Computers: Program Slicing, Academic Press 1996.
- [Boeh 1984] Boehm, B. W., Verifying and validating software requirements and design specifications, IEEE Software 1:(1), pp. 7-88, January 1984.
- [Carn 2006] Carnahan, J. C., Language Support for the Coercible Software Domain, PhD Proposal. School of Engineering and Applied Science, Department of Computer Science. University of Virginia., Charlottesville, VA, 2006.
- [Cha 2005] Cha, A. E., Computers Simulate Terrorism's Extremes, Washington Post, pp. A1, July 4, 2005.
- [Euba 2004] Eubank, S., H. Guclu, A. Kumar, M. V. Marathe, A. Srinivasan, Z. Toroczkai, and N. Wang, Modeling disease outbreaks in realistic urban social networks, Nature 2541:(429), pp. 180-184, November 2004.
- [Frit 2004] Fritzson, P., Principles of Object-Oriented Modeling and Simulation with Modelica 2.1, IEEE Press and Wiley-Interscience, New York, NY, 2004.
- [Fost 1999] Foster, J., M. Fahndrich, and A. Aiken, A theory of type qualifiers, ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI) 34(5), pp. 192-203, May 1999.
- [Park 2005] Park, S., A Programming Language for Probabilistic Computation, PhD thesis, School of Computer Science. Carnegie Mellon University., Pittsburgh, PA, August 2005.
- [Park 2006] Park, S., F. Pfenning, and S. Thrun., A probabilistic language based on sampling functions, Proceedings of the 32nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL), pp. 171-182, 2006.
- [Pear 2000] Pearl, J., Causality: Models, Reasoning, and Inference, Cambridge University Press, New York, NY, 2000.
- [Shne 2006] Shneiderman, B. and C. Plaisant. Strategies for evaluating visualization tools: multi-dimensional in-depth long-term case studies, Proceedings of the 2006 AVI workshop on BEyond time and errors: novel evaluation methods for information visualization, pp. 1-7, 2006.
- [Spie 2007] Spiegel, M., A proposal for computing with imprecise probabilities: A framework for multiple representations of uncertainty in simulation software, Technical Report CS-2007-16, Department of Computer Science, University of Virginia, 2007.
- [Spir 2001] Spirtes, P., C. Glymour, and R. Scheines, Causation, Prediction, and Search, Springer-Verlag, New York, NY, 2001.
- [Swia 1999] Swiatek, J. A. and D. C. Kaul, Crisis Prediction Disaster Management, Technical Report, SAIC, Hazards Assessment and Simulation Operation Technology Analysis and Applications Group, June 24, 1999.
- [Tip 1995] Tip, F., A Survey of Program Slicing Techniques, Journal of Programming Languages 3:(3), pp. 121-189, 1995.
- [Van 2006] Van der Walt, C.M. and E. Barnard, Data characteristics that determine classifier performance, In Proceedings of the 17th Annual Symposium of the Pattern Recognition Association of South Africa, pp. 160-165, November 2006.
- [Zeig 2000] Zeigler, B. P., H. Praehofer, and T.G. Kim, Theory of Modeling and Simulation 2nd Edition, Academic Press, Burlington, MA, 2000.