Quantifying and Analyzing Uncertainty in Simulations to Enable User Understanding

Michael Spiegel, Ross Gore, and Paul F. Reynolds, Jr., University of Virginia

Abstract—Quantitative methods of analysis have progressed faster than quantitative methods of capturing, representing, propagating and analyzing uncertainty in the realm of computational thinking, adversely affecting the quality of both scientific computational analysis, and important policy decisions. Uncertainty arises from incomplete model input information (aleatory uncertainty), incomplete model structure information (epistemic uncertainty), and incomplete understanding of model dynamics. We describe a work in progress approach to support representation, propagation, and calibration of aleatory uncertainty using probability theory, probability boxes, and Dempster-Shafer theory of evidence; 2) develop reliable methodologies — algorithms, data acquisition and management procedures, software and theory — for quantifying uncertainty in computer predictions; 3) support exploration of epistemic uncertainty utilizing causal analysis, and static and dynamic program slicing to characterize the dependencies, causal relationships, and interactions of design decisions; and 4) as a way of gaining insight into uncertainties, enable subject matter experts to observe model characteristics under novel conditions of interest. These capabilities represent a revolutionary approach to capturing, representing, propagating, and analyzing quantitatively, uncertainties that arise in the process of computational thinking.

Index Terms—computer languages, emergent behavior, quantifying uncertainty, risk analysis

I. INTRODUCTION

Computational thinking is ubiquitous. Unfortunately so are the risks associated with an unsystematic management of uncertainty during the design, development and use of computational models. One only needs to study conflicting model results in nearly every scientific endeavor to appreciate the problem. Many risks arise because uncertainties are quantified in a supplementary, rather than integrative manner. There are primarily two reasons they aren’t integrated: 1) uncertainties are often epistemic and 2) no good general purpose methods exist for capturing and propagating expert characterizations of uncertainty in their models. The impact is profound. How can policy makers make informed decisions involving billions of dollars and millions of people in confidence when poor management of uncertainty pervades model development and analysis? We present several work in progress methods for capturing and propagating characterizations of uncertainty in computational thinking-based models and for exploring uncertainties that emerge during model execution.

Modeling under uncertainty has been of paramount importance in the past half century, as quantitative methods of analysis have been developed to take advantage of computational resources. Simulation is gaining prominence as the proper tool of scientific analysis under circumstances where it is infeasible or impractical to directly study the system in question. According to a February 2006 report of the NSF Blue Ribbon Panel on Simulation-Based Engineering Science (SBES): “The development of reliable methodologies – algorithms, data acquisition and management procedures, software, and theory – for quantifying uncertainty in computer predictions stands as one of the most important and daunting challenges in advancing SBES” [1]. Its daunting nature is evident in the results of epidemiology studies conducted this century. Epidemiologists have addressed the question of government level actions and reactions regarding the spread of infectious diseases such as smallpox and bird flu. Should a comprehensive vaccination program be initiated? How and to what degree should infected individuals be isolated, and for how long? The range of answers to these questions is broad and full of conflict. Recently Elderd [2] has shown analytically that just four of the potentially hundreds of critical independent variables in these studies induce extreme sensitivity in model predictions, leading to serious conflict regarding remedial approaches involving billions of dollars and millions of people.

Clearly there is a need for robust uncertainty representation and analysis methods in computational thinking so that scientists and policy makers can better understand and characterize the properties of the predictions they make based on their models. Our envisioned solution builds on the acausal modeling language Modelica, producing a language we call “RiskModelica,” by incorporating novel methods for quantifying uncertainty formally and robustly, for propagating that uncertainty through the modeling process and revealing its effects on model outcomes, for later use by scientists and policymakers. Further, semi-automated methods for exploring
unexpected model outcomes that exploit propagated uncertainties will be developed and integrated with RiskModelica.

II. PREVIOUS WORK

A. Imprecise Probabilities and Model Calibration

Several different mathematical systems can be used to perform uncertainty analysis. We will focus on probability theory, probability boxes, and the Dempster-Shafer theory of evidence. Probability theory is the most traditional representation of uncertainty and the one most familiar to non-mathematicians. The use of probability theory attempts to provide a quantitative analysis to answer the following three questions: (1) what can go wrong, (2) how likely is it that will happen, and (3) if it does happen, what are the consequences [3]? Probability used as a representation of subjective belief is common in quantitative risk analysis. Safety assessments must deal with rare events and thus it is difficult to assess the relative frequencies of these events [4]. The Bayesian approach to uncertainty analysis is to specify a coherent probability measure as the current state of available knowledge, and use Bayes’ theorem to adjust probabilities as new evidence is unveiled. Imprecise probability is a generic term for any mathematical model which measures chance or uncertainty without crisp numerical probabilities. Two types of imprecise probability, probability boxes and Dempster-Shafer belief structures, offer a more flexible representation of uncertainty over the crisp probabilistic approach. According to a study by Ferson et al. [5], these two mathematical techniques provide an approach to several of the most serious problems that can arise during risk analysis, including: (i) imprecisely specified distributions; (ii) poorly known or even unknown dependencies; (iii) non-negligible measurement uncertainty; (iv) non-detects or other censoring in measurements; (v) small sample size; (vi) inconsistency in the quality of input data; (v) model uncertainty; and (vii) non-stationarity (non-constant distributions).

B. Probabilistic Programming Language

The dissertation work of Sungwoo Park of Carnegie Mellon University describes the design and implementation of PTP, a ProbabilisTic Programming language [6]. PTP is an extension of the lambda calculus that uses sampling functions to specify probability distributions. A sampling function takes as input an infinite sequence of random numbers drawn independently from U(0,1), consumes zero or more random numbers, and returns a sample with the remaining sequence. Park et al. [7] claim that PTP is the only probabilistic language with a formal semantics that has been applied to practical problems involving continuous distributions.

C. Modelica

Modelica is an object-oriented equation-based programming language. It is designed for large, complex and heterogeneous physical systems. Modelica programs are declarative, mathematical systems of equations that specify acausal relationships among state variables [8]. Acausal programming is a programming paradigm in which program data flow is not explicitly represented. The primary operator in acausal programming is the equality operator. In traditional imperative programming the primary operator is the assignment operator which has defined inputs, the right-hand side, and outputs, the left-hand side. The equality operator does not express neither input nor output. Instead it states that two expressions containing one or more variables are equivalent. Acausal programming allows for the expression of higher-order mathematical properties to be observed and preserved.

The Modelica language allows for the expression of a system of equations, and then the Modelica compiler will translate this system of equations into a traditional imperative C program. It is the purpose of the Modelica compiler to determine the appropriate data flow and control flow that will solve the system of equations. An application developer can express the problem as a system of equations and allow the programming tools to produce an executable program that will solve the equations.

Modelica also allows for the expression of hybrid discrete event and continuous systems. Hybrid differential algebraic equations allow for the expression of discontinuous changes in system state. Modelica allows us to express models that combine discrete event behavior and continuous behavior. Many real-world problems behave continuously until some threshold is crossed, at which point a sharp discontinuity in system state occurs.

D. Causality Analysis

Causal reasoning refers to the use of knowledge about cause-effect relationships in the world to support plausible inferences about events. Causal reasoning has been treated mathematically as a formal causal model and graph [9-10]. This formal representation relies on two ideas. First, the absence of causal relation is marked by independence in probability. If the outcome of one trial has no influence on the outcome of another, then the probability of both outcomes equals the product of each outcome separately. The second idea is that probability is associated with control: if variation of one feature, X, causes variation of another feature Y, then Y can be changed by an appropriate intervention that alters X. This representation captures a wide range of statistical models – regression models, logistic regression models, structural equation models, latent factor models, and models of categorical data – and captures how these models may be used in prediction [9]. This representation enables the use of algorithms for analyzing and discovering causal structure in linear and nonlinear systems, and systems with or without feedback from sample data.

E. Using slicing for program comprehension

Understanding how a certain feature is implemented is a
major research area of the program understanding community, especially when the understanding is directed to a certain goal like changing or extending the feature. Systems often appear as a large number of modules each containing hundreds of lines of code. Often it is not obvious which components implement a given feature. Eisenbarth et al. [11] have automated the process of localizing the implementation of a particular feature in the code. Their technique combines dynamic and static analyses to rapidly focus on the system’s parts urgently required for a goal-directed process of program understanding.

Dynamic information by way of execution summaries is generated by a profiler for different scenarios the user must manually create. This dynamic information is used to identify the subprograms executed when any of the given features is invoked. One scenario represents the invocation of preferably one single feature only and yields all subprograms executed for this feature. Next concept analysis is used to derive relationships between features and executed subprograms. These relationships identify subprograms jointly required by any subset of features, classify subprograms as low-level or high-level with respect to the given set of features and help to identify the subprograms that together constitute a larger component during static analysis [11].

This technique successfully uses static and dynamic analysis to help programmers gather insight into deterministic program behavior. However, Eisenbarth et al. have focused on deterministic software. RiskModelica will be applied to deterministic and stochastic models and will combine static and dynamic analysis, uncertainty analysis, and causal analysis to understand the interactions within a model that cause a specified behavior.

III. THE RISKMODELICA FRAMEWORK

Our approach begins with recognition of the benefit of separation of concerns: separating information about uncertainty from model behavior. Separation of concerns is a method often employed in computer science strategies for information sharing. Separation of concerns involves breaking a task into distinct features that overlap as little as possible in functional responsibility [12-13]. To translate this definition into practical terms, let us imagine a scenario where an environmental scientist and a policymaker need to communicate about the uncertainties in a water quality predictive simulation. The policymaker needs to understand uncertainties in the input and structure of the model. In practice, the scientist builds a deterministic simulation using either a simulation language (such as Arena, Matlab, Simulink) or a general-purpose language (such as C, C++, Java). Substitution of stochastic behavior for the indefinite components of the simulation are typically accomplished through the use of an application library that shares the formal semantics of the underlying programming language and therefore carries no additional information that is amenable to formal program analysis. The simulation implementation becomes only a means to an end; its sole purpose is to produce numbers. The policymaker must rely on the scientist’s interpretation of simulation output because the scientist’s research artifacts do not have the capacity to represent model uncertainty independently of model behavior. There has not been a separation of concerns. We will be addressing this shortcoming.

The following tasks capture the work that would be performed in order to enable representation, propagation and posterior analysis of uncertainty, thus enabling the flow of needed, quantifiable information between scientists and policymakers.

A. Separation of Concerns

A proven vehicle for achieving separation of concerns is through domain-specific programming languages (DSLs) – languages designed for specific families of tasks [14]. DSLs allow solutions to be expressed in the idiom and at the level of abstraction of the problem domain. Consequently, domain experts can understand, validate, modify, and develop DSL programs. Several high-profile examples of DSLs include XML for extensible data management, SQL for database queries, Prolog for artificial intelligence research, and Hancock (developed by AT&T) for data mining telecommunications records.

Our DSL will be built as an extension to the Modelica programming language for the purpose of uncertainty analysis in computational risk assessment studies. Recognizing community emphasis on risk analysis in computational thinking, the language has been dubbed RiskModelica. The RiskModelica framework will consist of an anterior component for specifying model uncertainty and a posterior exploratory component for understanding model uncertainty. The anterior component will consist of a formal uncertainty semantics that will allow the precise specification of ambiguities in model parameters (aleatory uncertainty) and ambiguities in model structure (epistemic uncertainty). Aleatory and epistemic uncertainties in models, particularly stochastic models, often result in model behaviors which are unexpected and not completely understood. The posterior, exploratory component is intended to address this model output uncertainty, and is discussed further in subsection C.

B. Anterior Component

The anterior component of RiskModelica will serve as a platform for research into representation and calibration of imprecise probabilities in quantitative risk analysis simulations and for analyzing and testing imprecise probability theories (e.g. Probability Boxes, Dempster-Shafer Theory) as alternative representations of stochastic variables. Imprecise probability theories present strong alternatives for overcoming some of the weaknesses of traditional probability theory [15]. They provide the capacity to express additional, quantified information to a decision-maker engaged in risk analysis and management. The anterior component of RiskModelica will focus on two primary design goals. The first is the representation of continuous and discrete random variables as first-class citizens in a programming language. We shall employ multiple mathematical frameworks for the representation of random variables. Each mathematical
framework displays a tradeoff of relative expressive power for ease of use. Probability theory suffers from three primary weaknesses when representing uncertainty [15]. First, a precise probability value must be assigned to each element in the set of possible outcomes. It may not be possible to assign exact values or even assign reasonable approximations when little information is available. Second, probability theory imposes Laplace’s principle of insufficient reason when no information is available. When \( n \) mutually exclusive possible outcomes are indistinguishable except for their names, they must each be assigned a probability of \( 1/n \). Third, conflicting evidence cannot be represented in traditional probability theory. By assigning probabilities to individual elements, we cannot express incompatibility between multiple sources of information or a cooperative effect between multiple sources of information.

The second design goal of the anterior RiskModelica capability is the inclusion of Bayesian calibration techniques for precise and imprecise probability theories. Modeling under uncertainty implies the absence of perfect information, but often partial information exists in the form of observations on the model's expected behavior. Simulation practitioners expect to make the best possible use of the information available to them. A Bayesian engine will support the calibration of probability theory, probability boxes, and probability mass functions. The inclusion of model calibration techniques is a vital component of a simulation language that intends on making the most out of limited available data.

The explicit representation of imprecise probability theories in a domain-specific programming language will facilitate the development of efficient algorithms for expressing, computing, and calibrating imprecise probability structures, for the purpose of conducting quantitative risk analyses that are more informative than analysis using traditional probability theory.

RiskModelica will be designed as a language extension to 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 \( QT \) [16]. Type qualifiers can be used as a mechanism for implementing a variety of compile-time and run-time semantic properties. One popular example is Splint, the static analysis tool of Evans and Larochelle, which uses program annotations to implement type qualifiers that can check for memory usage errors at compile-time [17]. Another example is the CCured translator, built by Necula et al., that extends C’s type system with pointer qualifiers [18]. Pointer qualifiers are type qualifiers that modify pointer types. CCured uses a combination of static analysis and run-time checking to add memory safety guarantees to C programs.

RiskModelica will extend the rich type qualifier system available in the Modelica language. Modelica contains twelve type qualifiers, preserving semantic properties at compile-time such as encapsulation [public/private], causality [input/output], and variability [constant/parameter/discrete], etc. RiskModelica will extend Modelica as necessary to support (i) probability bounds analysis, (ii) Dempster-Shafer belief structures, (iii) Bayesian inference, and (iv) model calibration in Dempster-Shafer theory. RiskModelica will maintain the type properties of Modelica in order for the new language features to integrate well with the host language. Modelica is statically typed, explicitly typed, and type safe for a subset of the language [19].

The RiskModelica language will be implemented as a superset of the Modelica language, and we will create a RiskModelica compiler to translate RiskModelica into an executable program. The output of our RiskModelica compiler will be a Modelica program, along with special instructions for the RiskModelica execution framework. The execution framework consists of the Modelica compiler, the Monte Carlo engine, and the Bayesian engine. Therefore it is the function of the RiskModelica compiler to translate RiskModelica-specific language constructs, and pass along the remaining Modelica constructs downstream to the Modelica compiler. The RiskModelica compiler will function primarily as a compiler frontend, as an intermediate language representation will not be generated and code optimization will not be performed. The RiskModelica compiler will build from the grammar specification that is available from the OpenModelica project, which uses ANTLR [20], a predicated LL(k) parser generator, to construct the Modelica parser. As to the Modelica compiler, we shall use either the open source OpenModelica compiler designed by Programming Environment Laboratory of Peter Fritzson [8], or the commercial Dymola compiler produced by Dynsim AB of Sweden [21]. The Modelica compiler translates the Modelica source into C code which is subsequently compiled and executed.

The remaining units of the RiskModelica execution framework, the Monte Carlo engine and the Bayesian engine, are responsible for directing execution flow in order to analyze the uncertainty information that has been encoded in the original RiskModelica program. The Monte Carlo engine and the Bayesian engine receive instructions from the RiskModelica compiler, and use these directions to control the execution of the compiled Modelica program. The Monte Carlo (MC) engine will be responsible for setting up multiple iterations of the model, and consequently the MC engine will be responsible for managing the sampling expression \( S \) and whatever sampling constructs will be created for probability belief structures. The Bayesian engine will be responsible for applying Bayes’ Law or the variations of Dempster’s rule of combination to calibrate RiskModelica models. When the Bayesian engine has been engaged, the output of the RiskModelica execution framework will not consist of model output. Instead the Bayesian engine will output a calibrated RiskModelica program which can be subsequently compiled and executed at a later time. The Bayesian engine thus operates as a one-time iterative example of self-modifying code. The Bayesian engine has only iteration by default, as the output of the engine is a calibrated program that has turned off Bayesian inference, although this inference could be manually turned back on when additional calibration data became available.
C. Posterior Component

Unexpected model outputs can reflect valid behaviors arising from seemingly unrelated phenomena, or they can reflect errors in model assumptions, design, or implementation. The posterior explanatory component of the RiskModelica framework will utilize the anterior component along with multiple complimentary analysis techniques to richly improve methods for exploring and understanding the behavior of models containing uncertainty. Increased insight gained from posterior analysis will contribute to reduced epistemic uncertainty which in turn will increase scientist and policymaker confidence in model results and predictions.

The need to understand unexpected model behaviors generally requires an exploration capability that extends a model beyond its intended use. The authors have previously published an exploratory approach using semi-automated search that allows a user to test hypotheses about unexpected behaviors as a simulated phenomenon is driven towards a condition of interest [22]. The posterior, exploratory component of RiskModelica will build upon this work and add an integrated multidimensional analysis capability of a model and its outputs. The multidimensional analysis will combine uncertainty representation, causality analysis, and static and dynamic program slicing to gather insight in a disciplined manner into the interactions within the model that cause unexpected model behavior.

Uncertainty representation will be provided by anterior RiskModelica and will be used to produce an expression of model output using multiple mathematical frameworks of uncertainty. Causal analysis refers to the use of knowledge about cause-effect relationships to support plausible inferences about events. Causal analysis analyzes model input and output to construct causal inference directed graphs that enable visual and automated analysis of causal relationships among variables [9-10]. Data for causal analysis can be obtained, applying existing COERCE technology, by collecting samples from semi-automated search used to create conditions of interest for user hypothesis tests [22]. Backward program slicing is performed on the simulation source code to efficiently determine the causal chain from the target variables representing the unexpected behavior to the source program statements causing the behavior for all inputs [23].

Static program slicing and causality analysis will be used to give a user insight into model behavior for all inputs, and suggest new hypotheses for a user to test. The causal analysis layer will indicate (with given confidence intervals) causal relationships between variables. This new information is not specified by static analysis due to its conservative nature. Furthermore causal diagrams give formal semantics to the notion of intervention, providing explicit formulas for estimating how the values of other variables will change given a specified change to a variable included in the causal diagram [10].

Uncertainty representation and dynamic program slicing can be used together to gather insight and understand the behavior of the model for specific inputs. Given a model exhibiting a specific condition of interest, dynamic program slicing will provide an accurate causal chain from the target variables to the source program statements for one possible run of a model with uncertain inputs. Uncertainty representation will allow users to observe two new characteristics of the unexpected behavior at conditions of interest: 1) the likelihood of the given output for the model and 2) the range of possible outputs. Understanding the likelihood of the output and the range of the possible outputs enables users to determine the utility of the dynamic program slice of a particular run. The analysis information for each variable at each condition of interest will be stored in a relational database that can be queried. The queries allow users to create specific filters to view the analysis data. Storing the analysis results for each condition of interest will provide both scalability and flexibility.

Posterior RiskModelica will not be tied to a source code level of detail. While it can be used at the source code level, users may find it too difficult to discern useful information at such a detailed level. Instead, users may perform the multidimensional analysis at the level of model abstractions. When constructing a model, abstractions inevitably must be selected in order to reduce complexity, improve performance, or provide estimations for unknown information. When developing simulations a SME identifies a set of abstraction opportunities and alternatives for each model abstraction. Model abstractions have been studied and formalized in [24]. The multidimensional analysis on this formalized level will gather insight into the interactions between abstractions that cause unexpected behavior. This provides users with a configurable approach to handling uncertainty that arises due to unexpected behaviors during model execution.

The program understanding community has successfully used dynamic and static analysis to help programmers gather insight into program behavior [11]. However, the program understanding community has focused on deterministic software. Utilizing these program slicing techniques together with anterior RiskModelica, uncertainty analysis, and causal analysis, can extend the state of the art. The posterior RiskModelica component will supply users with a tool to reduce uncertainty as to whether a particular behavior is valid by understanding the interactions within the model that cause unexpected behavior.

IV. CONCLUSION

Simulation has become the tool of scientific analysis under circumstances where it is infeasible or impractical to study a system directly. Uncertainty is an inevitable part of such studies. Large scale analyses clearly affected by uncertainty include the Yucca Mountain nuclear waste repository studies, large scale epidemiology studies, the four assessment reports of the Intergovernmental Panel on Climate Change, and in a related sense, guidelines from the Office of Budget and Management that recommend formal quantitative uncertainty analysis for federal regulations involving annual economic effects of $1 billion or greater. With impact on this scale methods for capturing uncertainty must be employed.

Methods exist for representing uncertainty but they have not been integrated in a structured, general manner with the tools
of computational thinkers. Best concepts and practices from computer science, probability and statistics, and risk management should be employed in a novel framework for quantitative uncertainty management. Results will be of two varieties: 1) new tools that quantify, represent and propagate user knowledge about uncertainty and 2) new tools and methods that support exploration of uncertainties that arise in the form of emergent behaviors as a model executes. The first result represents a front-end (anterior) process in which a user captures uncertainty quantitatively using best known methods (e.g. probability boxes and Dempster-Shafer theory). The second represents a back-end (posterior) process in which the inevitable surprises that arise in model exploration can be dealt with, and epistemic uncertainty can be reduced. Future work remains in determining the correct method for using uncertainty representation provided by anterior RiskModelica to better the causal analysis performed in posterior RiskModelica.

ACKNOWLEDGMENT

We gratefully acknowledge support from our colleagues in the Modeling and Simulation Technology Research Initiative (MaSTRI) at the University of Virginia as well as from Dr. Jack Cosby from the Department of Environmental Sciences, UVa.

REFERENCES


