Input Attribution for Statistical Model Checking Using Logistic Regression

Jeffery P. Hansen(✉), Sagar Chaki, Scott Hissam, James Edmondson, Gabriel A. Moreno, and David Kyle

Carnegie Mellon University, Pittsburgh, PA, USA
{jhansen,chaki,shissam,jredmondson,gmoreno,dskyle}@sei.cmu.edu

Abstract. We describe an approach to Statistical Model Checking (SMC) that produces not only an estimate of the probability that specified properties (a.k.a. predicates) are satisfied, but also an "input attribution" for those predicates. We use logistic regression to generate the input attribution as a set of linear and non-linear functions of the inputs that explain conditions under which a predicate is satisfied. These functions provide quantitative insight into factors that influence the predicate outcome. We have implemented our approach on a distributed SMC infrastructure, DEMETER, that uses Linux Docker containers to isolate simulations (a.k.a. trials) from each other. Currently, DEMETER is deployed on six 20-core blade servers, and can perform tens of thousands of trials in a few hours. We demonstrate our approach on examples involving robotic agents interacting in a simulated physical environment. Our approach synthesizes input attributions that are both meaningful to the investigator and have predictive value on the predicate outcomes.

1 Introduction

Statistical model checking (SMC) [4,23] has emerged as a key technique for quantitative analysis of stochastic systems. Given a stochastic system $\mathcal{M}$ depending on random input x, and a predicate Φ, the primary goal of SMC is to estimate the probability $P[\mathcal{M} \models \Phi]$ that Φ is satisfied in $\mathcal{M}$ within some specified level of confidence (e.g., relative error). SMC, which is based on Monte-Carlo methods, has some major advantages over methods such as probabilistic model checking. It can be applied to larger and more complex systems, and to the actual system software rather than an abstract model of that software. Moreover, it can analyze a system as a "black box" observing only its inputs and outputs.

While estimating the probability that a predicate holds is important, it is also important to understand the factors that contribute to that estimate. We refer to this as *input attribution*. More specifically, an input attribution is a

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center, DM-0003895.

Y. Falcone and C. Sanchez (Eds.): RV 2016, LNCS 10012, pp. 185–200, 2016.
DOI: 10.1007/978-3-319-46982-9_12

human-understandable quantitative model explaining the relationship between the random inputs and the specified predicate Φ (e.g., a mathematical expression of the input variables that predicts whether Φ will be satisfied). A good input attribution must: (i) describe a relationship that actually exists in the system; (ii) be presented in a way that is quantitative, meaningful and understandable to the investigator; (iii) give the investigator new insights into the system; and (iv) be resilient to additional hidden or uncontrolled randomness (e.g., randomness due to the physics in the system not included in the input x).

In this paper, we address the input attribution problem for SMC, and make the following contributions. First, we present an approach to input attribution that builds a statistical model from the simulation data collected during SMC. Among several potential statistical modeling methods, we focus on logistic regression [14] (LR). Logistic regression is targeted at systems with a binary (or categorical) dependent variable, which is exactly the case in SMC. The result of an LR analysis is a function that predicts the probability that the dependent binary variable will be 1 as a function of the input variables. One advantage of LR over other techniques, such as linear discriminant analysis, is that it makes no assumptions on the distribution of the inputs. We show how to compute both linear and polynomial input attributions via LR.

Second, we implement our approach in a distributed SMC infrastructure, called DEMETER, that uses a dispatch and join pattern to run many simulations in parallel across a set of machines. DEMETER uses Docker [17] containers to isolate simulations from each other, and batching to avoid statistical bias in results [22]. Using six blade servers, DEMETER has to date run millions of simulations over many days, demonstrating its robustness. Finally, we validate our approach over a set of examples involving one or more agents that operate under uncertainty to achieve specific goals. Our results indicate that the LR-based approach is able to synthesize input attributions that are both meaningful to the investigator and have predictive value on the predicate outcomes.

The rest of this paper is organized as follows. In Sect. 2 we discuss related work; in Sect. 3 we discuss some basic concepts and theory of Statistical Model Checking; in Sect. 4 we discuss our approach to input attribution; in Sect. 5 we describe DEMETER; in Sect. 6 we present our results in applying our techniques to three different examples; and in Sect. 7 we conclude.

2 Related Work

SMC, developed by Younes [23], has been applied to a wide variety of system models including stochastic hybrid automata [7], real time systems [8], and Simulink models for cyber-physical systems [4]. In contrast, we apply SMC directly to the system executing in an operating environment that includes uncertainty from scheduling and communication. Our prior work [16] also presented a distributed SMC infrastructure for DMPL [3] programs, but used a manually managed set of virtual machines to isolate trials from each other logically. In contrast, DEMETER uses lighter weight Docker [17] containers for isolation, and

Rancher [1] for automated launching and failover. In addition, it is able to carry out trials involving a broader class of applications, not just those generated from DMPL programs.

The PRISMATIC [19] project investigated "counterexample generation and culprit identification" in the context of probabilistic verification. It used machine learning (specifically the Waffles tool) to construct decision trees from runs of the system. From the decision tree, one can infer the component that is most responsible for failure. Their approach has limited effectiveness when a combination of several components leads to failure. In contrast, we use LR to give numeric weights to input variables, as well as polynomial terms of such variables. This makes our approach more effective when a combination of multiple random inputs is the more likely cause of failure. In addition, the PRISMATIC tool is built on top of PRISM [15] and can analyze models, while we analyze system executables.

The problem of determining under which conditions a program will fail has also been explored in the context of non-stochastic software. For example, Cousot et al. [6] use abstract interpretation [5] to statically compute an expression over a function's parameters (i.e., a precondition) under which the function will always fail a target assertion. Similarly, the DAIKON system [11] dynamically constructs likely program invariants from collected execution traces using machine learning techniques. Our goals are similar, in that we want to produce artifacts that provide insight about a program's behavior, but our focus is on stochastic systems, and we use logistic regression.

3 Background

Consider a system $\mathcal{M}$ with a finite vector of random inputs x over domain D_x. The SMC problem is to estimate the probability $p = P[\mathcal{M} \models \Phi]$ that $\mathcal{M}$ satisfies a predicate Φ given a joint probability distribution f on x. Let us write $x \sim f$ to mean x has distribution f. SMC involves a series of Bernoulli trials, modeling each trial as a Bernoulli random variable having value 1 with probability p, and 0 with probability $1 - p$. For each trial i, a random vector $x_i \sim f$ is generated, and the system $\mathcal{M}$ is simulated with input x_i to generate a trace σ_i. The trial's outcome, y_i, is 1 if Φ holds on σ_i, and 0 otherwise.

Traditionally, we would assume that whether $\mathcal{M} \models \Phi$ is satisfied under a specific input x is deterministic. However, since we are considering physical simulations of agents, the physics engine itself may introduce additional randomness that is not under our control. For this reason, we weaken our deterministic output assumption and assume the outcome y_i of $\mathcal{M} \models \Phi$ for a specific input x_i is itself a Bernoulli random variable with an unknown probability $J_{\mathcal{M} \models \Phi}(x_i)$ that $\mathcal{M} \models \Phi$ is satisfied. An alternative and equivalent way to model this is to introduce a hidden random variable $u \sim \mathcal{U}(0, 1)$ to represent randomness inherent in

the simulation.[1] We then have a system with input x, u for which $\mathcal{M} \models \Phi$ is satisfied when $J_{\mathcal{M}\models\Phi}(x) \geq u$.

Define an indicator function $I_{\mathcal{M}\models\Phi} : D_x \times [0,1] \rightarrow \{0,1\}$ that returns 1 if $\mathcal{M} \models \Phi$ under input x, u, and 0 otherwise. Then, when $x \sim f$, and $u \sim \mathcal{U}(0,1)$, the probability $p = E[I_{\mathcal{M}\models\Phi}(x,u)]$ that $\mathcal{M} \models \Phi$ holds can be estimated as $\hat{p} = \frac{1}{N}\sum_{i=1}^{N} I_{\mathcal{M}\models\Phi}(x_i, u_i)$, where N is the number of trials. Note that, while we observe the values of x_i for each trial simulation, we can see only the resulting outcome $I_{\mathcal{M}\models\Phi}(x_i, u_i)$ and not the value of the hidden variable u_i itself.

The precision of $\hat{p}$ is quantified by its *relative error* $RE(\hat{p}) = \frac{\sqrt{Var(\hat{p})}}{\hat{p}}$ where $Var(\hat{p})$ is the variance of the estimator. It is known [4] that for Bernoulli trials, relative error is related to the number of trials N and the probability of the event p as $RE(\hat{p}) = \sqrt{\frac{1-p}{pN}} \approx \frac{1}{\sqrt{pN}}$. Thus, we have $N = \frac{1-p}{pRE^2(\hat{p})} \approx \frac{1}{pRE^2(\hat{p})}$.

4 Input Attribution

Statistical learning is a field of statistics that is concerned with finding a model that relates a stimulus to some response [12]. In the case of statistical model checking, the stimulus is the set of random input variables and the response is the outcome of a trial. There are two main uses for these learned models: prediction and inference. Prediction is using the model to predict the response given a stimulus, while inference is learning something about the relationship between the stimulus and the response. Input attribution is primarily concerned with inference, though we do evaluate the predictive power of the model to ensure validity of any input attribution generated by our approach.

One technique used in statistical learning is logistic regression. In logistic regression, a linear function of "predictors" is fit to the log of the odds (often called a "logit") ratio that a binary response variable holds. Here, odds are simply an alternative way of representing probability such that $p = \frac{\gamma}{1+\gamma}$ is the probability where γ is the odds. For example, if the odds of an event are 4 to 3, then $\gamma = \frac{4}{3}$ and the probability is $p \approx 0.57$.

In this paper, we take the predictors to be either the input variables, or a combination of the input variables and functions of the input variables. For simplicity we assume all random variables are continuous or countable, though it is possible to generalize these results to categorical random variables. The analysis is performed independently for each predicate defined by the investigator with a separate result for each. Let x_{ij} be the inputs over a set of trials $1 \leq i \leq N$ with predictors $1 \leq j \leq M$ for each input, and y_i be the result of each trial. Logistic regression will find a linear function of the form:

$$L : x \mapsto \beta_0 + \sum_{j=1}^{M} \beta_j x_j \tag{1}$$

[1] In this paper we use $\mathcal{U}(a,b)$ for the uniform distribution between two real numbers $a \leq b$, and $\mathcal{U}\{a,b\}$ for the uniform integer distribution between a and b, inclusive.

such that $\hat{p} = \frac{1}{1+e^{-L(x_i)}}$ is the predicted probability that we will get a response of $y_i = 1$ given input x_i. The β_j for a continuous random variable x_j represents the increase in the "logit" for each unit increase in of x_j. When interpreting the coefficients, it is sometimes useful to think of an "odds ratio", the factor by which the odds changes in response to some change. For an increase of Δ on variable x_j the odds ratio will be $e^{\beta_j \Delta}$. Note that β_j itself does not necessarily indicate the importantance of x_i as it is also dependentant on the units.

We use the R statistical analysis system [20] to perform the logistic regression. For each predictor x_j, R generates a maximum likelihood estimate $\hat{\beta}_j$ and a standard error $se(\hat{\beta}_j)$ of the coefficient β_j for that predictor. The standard error is used to perform a Wald test [14] on the significance of β_j against the null-hypothesis that β_j could be 0 (i.e., the hypothesis that predictor x_j is not important for determining the outcome). The Wald test involves calculating a z-value $z_j = \frac{\hat{\beta}_j - 0}{se(\hat{\beta}_j)}$ representing the number of standard deviations from zero of the $\hat{\beta}_j$ estimate, then looking up that value in the Normal distribution table to find the p-value representing the probability that the null-hypothesis could occur by chance. Typically a p-value <0.05 is considered statistically significant.

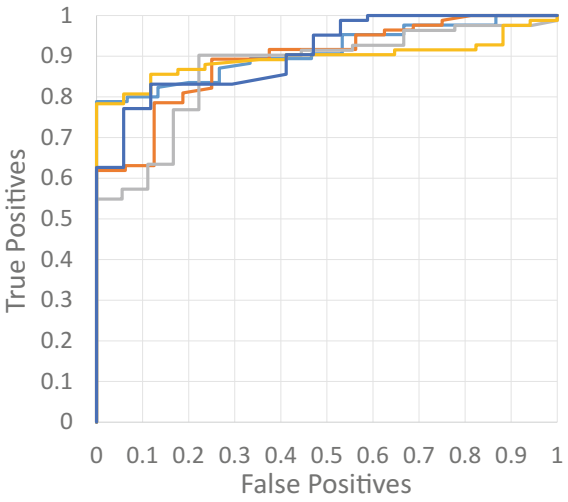
Since our goal is to discover relationships between the predictors and the predicate, the $\hat{\beta}_j$ and the associated p-values for each coefficient are the most useful for us. Low p-values tell us that a predictor is significant, and the $\hat{\beta}_j$ tells us the factor by which the log of the odds for the predicate being satisfied increases which each unit increase in predictor x_j.

4.1 Linear Input Attribution

The most straight-forward application of logistic regression is to use each input x_j as a predictor, and report those for which the p-value is below the selected threshold. Our approach includes this as one of its options, and is the easiest to use when relationships between input variables and predicates are linear.

Example. As an illustrative example consider a data set consisting of 500 samples of random vectors $x = (x_1, x_2, x_3)$ where $x_1 \sim \mathcal{U}\{1, 6\}$, $x_2 \sim \mathcal{U}\{1, 6\}$, $x_3 \sim \mathcal{U}\{1, 12\}$. Also assume there is a hidden random variable $u \sim \mathcal{U}\{1, 10\}$ that affects the outcome of the trial, but cannot be directly observed. Now assume the predicate y we are testing is 1 when $x_2 + x_3 + u > 10$ and 0 otherwise. When we apply logistic regression on this data set using R we get: $L : x \mapsto -2.8 - 0.03x_1 + 0.50x_2 + 0.64x_3$, with p-factors well below 0.01 for both x_2 and x_3, and a p-factor of 0.78 for x_1 indicating it is not statistically significant (which is expected since it was not involved in the predicate being tested). For this example, our approach generates the input attribution: $0.50x_2 + 0.64x_3$, excluding the x_1 term because it was not statistically significant. The positive coefficients for both x_2 and x_3 indicate the probability of y being 1 increases with an increase in either input, as expected.

Before accepting the result of the logistic regression analysis, we must verify that the overall logistic model fits the data. We do this using ROC (Receiver



(a)

Name	β	$se(\beta)$	p-value
x_1	16.0	2.80	$< 10^{-4}$
x_2	17.8	2.75	$< 10^{-4}$
x_1^2	-18.1	2.43	$< 10^{-4}$
x_2^2	-19.2	2.42	$< 10^{-4}$
$x_1 x_2$	4.5	2.74	0.0976

(b)

Fig. 1. (a) ROC curves for linear example; (b) Logistic regression results for polynomial example.

Operating Characteristic) analysis [13]. In ROC analysis, we consider $L(x) > T$ for some threshold T to be the prediction that is compared to the actual result of the predicate y. We then plot the true positive rate $P[y = 1|L(x) > T]$ against the false positive rate $P[y = 0|L(x) > T]$ for $-\infty < T < \infty$.

The ROC curve for our example data using 5-fold cross validation is shown in Fig. 1(a). In 5-fold cross validation, the trials are randomly partitioned into 5 chunks then 4/5 of the data are used to build the logistic model while the remaining 1/5 is used to validate the model. This process is repeated 5 times with each chunk taking its turn as the test data resulting in curves for each of the 5 folds shown in the figure. In ROC analysis, curves that approach the upper left corner are considered "good" detectors while a detector near the diagonal from the lower left to upper right are no better than random guessing. Another more succinct way to present the results of an ROC analysis is the AUC (Area Under Curve). It is known [13] that the AUC is equivalent to the probability $P[L(x_{sat}) > L(x_{unsat})]$ where x_{sat} is a randomly selected input that satisfies the predicate and x_{unsat} is a randomly selected input that does not satisfy the predicate. An AUC of 1 corresponds to a perfect detector, while an AUC of 0.5 corresponds to a detector that is no better than guessing. Using our sample data, we computed AUC for each fold yielding values be 0.87 and 0.91 with a mean of 0.90 indicating good predictive value of the model, and thus indicating our input attribution is valid.

4.2 Non-linear Input Attribution

If there are non-linear dependencies between a predictor and the predicate, the linear analysis techniques described above will fail to find a statistically significant relationship. To solve this problem, we include non-linear functions of the inputs as predictors. In effect, the non-linear functions are "guesses" on potential relationships between the inputs and the predicate. Some guesses may result in statistically significant relationships, while others may not and be discarded.

In this paper, we restrict our guesses to 2^{nd} order polynomials over the input variables. These are important because in sets of random variables describing points in space, distances between fixed points, or between pairs of random points can be described as 2^{nd} order polynomials. Note that it is not necessary to actually include every possible polynomial of the inputs, we need only include the monomial building blocks of the possible polynomials. More specifically, in addition to the linear input terms $\{x_1, \ldots, x_M\}$, we include the additional predictors $\{x_j^2 \mid 1 \leq j \leq M\}$, and the predictors $\{x_j x_k \mid 1 \leq j < k \leq M\}$.

Factored Polynomials. In order to present relationships that are easy to interpret, our algorithm further attempts to find factored polynomials where possible. Two types of factors are considered: single variable and two variable. In each case, we look for subsets of the log odds expression (1) that can be factored. In the single variable case, we look for predictors $\beta_a x_j^2$ and $\beta_b x_j$ where both have a low p-value and factor by completing the squares as:

$$\beta_a(x_j + \frac{\beta_b}{2\beta_a})^2 + C = \beta_a x_j^2 + \beta_b x_j \tag{2}$$

where C is the constant needed to complete the square. But since our goal is to show relationships, our algorithm only outputs the $\beta_a(x_i + \frac{\beta_b}{2\beta_a})^2$ part.

Factored Two-Variable Polynomials. Similarly, in the two variable case, we look for predictors $\beta_a x_j^2$, $\beta_b x_j x_k$ and $\beta_c x_k^2$ with low p-values, and factor as:

$$\beta_a x_j^2 + \beta_b x_j x_k + \beta_c x_k^2 = \beta_a(x_j + (h+g)x_k)(x_j + (h-g)x_k) \tag{3}$$

where $h = \frac{\beta_b}{2\beta_a}$ and $g = \frac{\sqrt{\beta_b^2 - 4\beta_a\beta_c}}{2\beta_a}$ when g is real. Since all of the β_i are approximations, our algorithm suggests the simpler factoring:

$$\beta_a x_j^2 + \beta_b x_j x_k + \beta_c x_k^2 \approx \beta_a(x_j + h x_k)^2 \tag{4}$$

when $|\beta_c - \beta_a h^2| < K se(\beta_c)$ (i.e., when the polynomial coefficient of the x_k^2 term in $\beta_a(x_j + h x_k)^2$ is within K standard error of its original value). We use $K = 3$ in this paper.

Example. Now consider an example with 500 trials with inputs $x = (x_1, x_2)$ where $x_1 \sim \mathcal{U}(0,1)$ and $x_2 \sim \mathcal{U}(0,1)$ are uniformly distributed between 0 and 1 and represent coordinates in a 2D square. Define the predicate y as being true when the point x is within distance $u \sim \mathcal{U}(0, 0.5)$ of the center $(0.5, 0.5)$ of the square but with u being a hidden random variable.

When we first try a linear analysis, we find that the p-factors for both x_1 and x_2 are not significant. When we repeat the analysis using polynomial terms we get the results shown in Table 1(b). The 5-fold cross validation for this example results in an average AUC of 0.85 indicating reasonable predictive strength for the model. Since the p-value for the x_1x_2 term is greater than 0.05, it is not statistically significant and so we only look for single variable polynomial expressions. Completing the squares for the x_1 and x_2 gives us the input attribution: $-18.1(x_1 - 0.44)^2 - 19.2(x_2 - 0.46)^2$. We see from the form of this expression that the analysis synthesizes an expression for distance between (x_1, x_2) and $(0.44, 0.46)$ which is close to the expected distance from the center point at $(0.5, 0.5)$. We also see this expression is negative meaning that the further (x_1, x_2) is from the center point, the lower the probability that y is satisfied.

5 SMC Infrastructure: DEMETER

We have implemented our approach in a distributed SMC infrastructure called DEMETER (Distributed Execution of Multiple Experiments and Transfer of Empirical Results). DEMETER uses a dispatch and join pattern for parallel processing across a set of machines. Dispatch is managed by an *SMC Master* (see Fig. 2) which queues SMC *jobs*. A job is described via a `.smc` file, which includes the system $\mathcal{M}$, the input variables x_i and their probability distribution f, target predicate(s) Φ, and the target relative error for each predicate $RE(\hat{p})_\Phi$.

A job is conducted by the Master as a series of Bernoulli trials. Each trial in the series is allocated to an *SMC Runner* which simulates the system $\mathcal{M}$ with the trial inputs, and reports the outcome. The system $\mathcal{M}$ can be any arbitrary piece of software that can be invoked from a shell script dispatched by the Runner, potentially with multiple communicating processes. The trial input to the Runner is an instance of a random input vector, $x_i \sim f$, generated by the Master for that trial. The outcome, produced by $\mathcal{M}$, is either 1 if Φ holds on σ_i, and 0 otherwise for each Φ described by the job. The Master records the input

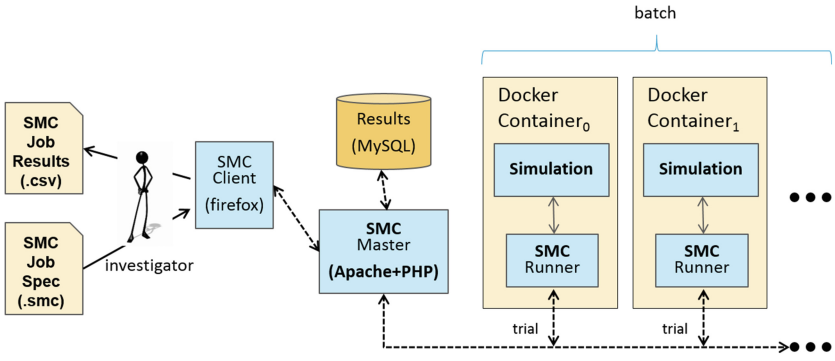


Fig. 2. The Master-Runner architecture of DEMETER.

and outcome for each trial in the *Results* database and the Master will continue to perform trials until the target relative error computed for each $\hat{p}_\Phi$ is reached.

A job may require thousands of trials to reach the target $RE(\hat{p}_\Phi)$. The Master dispatches trials in parallel as a sequence of *batches* allowing all trials in a batch to complete before starting the next batch to avoid bias [22]. The default batch size equals the number of available Runners, but this can be controlled via the "weight" specification for the job (discussed later in *Load Management*). Once a batch is dispatched, the Master waits for all trials in that batch to complete before dispatching the next batch. A batch is complete if every trial in it completes with either 0 or 1 for all predicates defined for the job. Any other result (e.g., an infrastructure error) is considered an error for the trial, and the entire batch is discarded to avoid bias. Trials from a complete batch are used to compute $RE(\hat{p}_\Phi)$ for each predicate.

For our experiments, we deployed DEMETER over six Dell PowerEdge blades with each blade having 128 Gb of RAM and 2 Intel Xeon E5-2687W 3.1 GHz processors with each processor having 10 processing cores and 2 cpu threads per core. A pool of 216 Runners (36 *Runners* $*$ 6 *blades*[2]) were used. We now discuss how DEMETER achieves isolation between trials.

Logical Isolation is achieved by running each trial within a separate Docker [17] instance. Docker has the following twofold advantage over other virtual machine approaches: (i) startup, shutdown and failover for each Runner can be managed by open source tools like Rancher [1] with low learning curve; and (ii) Docker's overhead is low compared to other full operating system virtualization technologies [21].

Network Isolation. Processes in Docker containers can communicate at the network level with each other. By default, multicast messages (used in our simulations) cross container boundaries. This can result in processes from one simulation receiving messages from another simulation. Using Docker's `icc=false` configuration directive disables such inter-container communication, but also disables Rancher's ability to manage the Runners. As such, `iptables` was used in conjunction with `icc=true` to drop all multicast traffic emanating from the `docker0` interface. This minimalistically isolates Runners from each other, avoiding network interference within the simulation, while allowing Rancher management.

Load Management. DEMETER allows a numeric "weight" to be specified for the job. Intuitively, the weight w is the number of CPU cores required to execute each trial. Note that w can be greater than 1 if, for example, the application is distributed and consists of several nodes executing in parallel. The Master executes at most $\lfloor \frac{216}{w} \rfloor$ trials in parallel at a time. Thus, CPU overload can be avoided by specifying a suitably large value of w.

[2] This leaves a few CPU threads for host processing independent of simulation activities.

6 Results

We validated our approach on three scenarios with increasing complexity. In each experiment, one or more agents are realized by a quadcopter model in the physics simulator V-REP [9]. An additional Linux process is used as a controller for each agent, communicating over a socket with V-REP. For each scenario we show that DEMETER constructs effective input attributions.

6.1 Pursuer/Evader Scenario

Scenario Overview. The goal of this example was to validate the effectiveness of non-linear input attribution via logistic regression. It consists of two agents—a pursuer P and an evader E—moving on a 2-dimensional 20×20 grid of cells. Each cell is identified by its coordinate, with the cell at the lower-left corner of the grid being $(0, 0)$ and the cell at the upper-right corner being $(19, 19)$. Each trial runs as follows: (i) the pursuer starts in a random initial cell (x_p, y_p) and the evader starts in a random initial cell (x_e, y_e) such that x_p, y_p, x_e and y_e are all uniformly selected from $\mathcal{U}\{6, 13\}$; note that this means that initially P and E are located in the sub-grid whose lower-left cell is $(6, 6)$ and upper-right cell is $(13, 13)$; (ii) the evader moves toward the grid corner away from the pursuer with velocity v, and the pursuer moves toward the evader with velocity kv where $k > 1$ is the factor by which P is faster than E; (iii) if P is able to reach within distance d of E by time $t_{\max}$ then P wins and the trial results in 0; otherwise, E wins and the trial results in 1. The constants v and $t_{\max}$ are such that E can never reach a grid corner by time $t_{\max}$, and hence always has space to move. Intuitively, the result of a trial depends on the initial distance between E and P, i.e., on $(x_e - x_p)^2 + (y_e - y_p)^2$. Moreover, this is a polynomial and hence requires non-linear input attribution. Purely linear logistic regression will not be able to detect this dependency. Our results, discussed next, confirm these intuitions.

Analysis of Results. Using DEMETER, we estimated the probability $\hat{p}$ that the evader escapes the pursuer with a result of $\hat{p} = 0.214$. This was estimated to a target relative error of 0.01 and required 36,960 trials, of which 7,900 satisfied the predicate. Total run time was 5 h and 20 min with 120 trials per batch. When we perform the Logistic Regression for the linear analysis, we get the results shown in Table 1(a). Only x_e results in a p-value less than 0.05. However, an ROC analysis with 5-fold cross validation results in an average AUC of 0.51, indicating that the predictive value of the model is no better than chance. For this reason, we do not accept the results of the linear analysis.

When we include the polynomial terms, we get the results shown in Table 1(b) (only terms for which the p-value was below 0.05 are shown). The AUC for the 5-fold cross validation including the polynomial terms was 0.77, considerably better than that including only linear terms. When we apply the factoring heuristics from Sect. 4.2, we get the following polynomial input attribution expressions: $0.0602(x_e - 1.03x_p)^2$ and $0.0561(y_e - 1.09y_p)^2$. These expressions, generated automatically from just the simulation data, are very close to our

Table 1. Logistic analysis results of pursuer/evader experiment.

(a) Linear Analysis

Name	β	$se(\beta)$	p-value
x_e	-0.0178	0.0055	0.0013
y_e	0.0106	0.0055	0.0554
x_p	0.0026	0.0056	0.6458
y_y	-0.0009	0.0055	0.8689

(b) Polynomial Analysis

Name	β	$se(\beta)$	p-value
$x_e x_p$	-0.124	0.0027	$< 10^{-4}$
$y_e y_p$	-0.122	0.0027	$< 10^{-4}$
x_e^2	0.060	0.0031	$< 10^{-4}$
y_e^2	0.056	0.0031	$< 10^{-4}$
x_p^2	0.056	0.0031	$< 10^{-4}$
y_p^2	0.056	0.0031	$< 10^{-4}$

expectation that the probability of escape for the evader depends on the initial distance between the pursuer and evader. The positive leading coefficient on both of the expressions tell us that the probability of escape increases as the initial distance increases, which is also what we expect.

6.2 Target/Threat Scenario

Scenario Overview. This scenario involves self-adaptive behavior by an agent that must fly a pre-planned route over a 2D grid at constant forward speed, detecting as many targets on the ground as possible. Target detection is done using a downward-looking sensor whose performance is inversely proportional to the distance from the ground. There are also threats along the route, but the probability of being destroyed by a threat is inversely proportional to the height of the agent. Clearly, when deciding at what height the agent should fly there is a tradeoff between detecting targets and avoiding threats.

The number and location of targets and threats is random and unknown. However, the agent has a forward-looking sensor that observes the environment ahead with some finite horizon to detect threats and targets, albeit with false positive and false negative rates. The agent self-adapts proactively [18] to this uncertainty by changing its altitude as it flies. It aims to maximize the number of targets detected, taking into account that if it is destroyed, no more targets are detected and the mission fails. Specifically, using the forward-looking sensor information, the agent periodically constructs at run time a probabilistic model of the environment, which is then used to make the adaptation decision—either stay at the current altitude, or increase/decrease altitude by one level.

In this scenario, mission success is defined as detecting at least 50 % of the existing targets without being destroyed. We break this down into three predicates with the predicate Φ_t being "at least 50 % of the existing targets are detected", Φ_s being the "the agent survives to end of run", and $\Phi_m = \Phi_t \wedge \Phi_s$ being the predicate for mission success. The random variables upon which these results depend are shown in Table 2.

Analysis of Results. The estimated probabilities produced by DEMETER for the predicates tested are $\hat{P}[\Phi_t] = 0.361$, $\hat{P}[\Phi_s] = 0.618$ and $\hat{P}[\Phi_m] = 0.308$. Thus, while the agent survived most trials, the mission success rate is low because of

Table 2. Inputs for target/threat scenario.

Name	Dist.	Description
N_E	$\mathcal{U}\{10, 40\}$	Total number of existing targets
N_T	$\mathcal{U}\{5, 20\}$	Total number of existing threats
d_{LA}	$\mathcal{U}\{1, 5\}$	Number of cells in front of the agent scanned by the forward-looking sensor, and decision horizon for proactive adaptation
p_{EFP}	$\mathcal{U}(0, 0.5)$	False positive rate for target detection with the forward-looking sensor
p_{EFN}	$\mathcal{U}(0, 0.5)$	False negative rate for target detection with the forward-looking sensor
p_{TFP}	$\mathcal{U}(0, 0.5)$	False positive rate for threat detection with the forward-looking sensor
p_{TFN}	$\mathcal{U}(0, 0.5)$	False negative rate for threat detection with the forward-looking sensor
r_E	$\mathcal{U}\{1, 5\}$	Downward-looking sensor range (i.e., maximum height from which a target can possibly be detected)
r_T	$\mathcal{U}\{1, 3\}$	Threat range (i.e., maximum height at which agent can be destroyed)

failure to detect the required number of targets. The target relative error was 0.01, and 22,560 trials were completed in 10 h and 6 min, with 120 trials per batch. The 5-fold cross validated mean AUCs for the linear and polynomial versions of the logistic regression analysis for each of the three predicates were approximately equal, ranging between 0.891 and 0.926. Since the polynomial model provides no additional predictive quality, we focus our analysis on the linear results shown in Table 3 (coefficients that were not statistically significant for a predicate are not shown). Notable conclusions from these results are:

(a) The most important variables affecting Φ_m are r_E, r_T, N_T, d_{LA} and p_{TFP}. Mission success $\hat{P}[\Phi_m]$ increases as r_E and d_{LA} increase, but decreases as the other input variables increase.
(b) As the target false-positive rate p_{TFP} increases, $P[\Phi_s]$ increases but $P[\Phi_t]$ decreases. The increase in $P[\Phi_s]$ is explained by the fact that falsely detecting a threat causes the agent to fly at a higher altitude on average. This results in it being at a higher than necessary altitude when it actually encounters a threat, thus increasing its probability of survival. On the other hand, being higher than necessary causes it to miss targets, thus lowering $P[\Phi_t]$.
(c) Increasing the number of targets N_T results in decreases to all three predicates. While it is not surprising that increasing the number of targets makes it more difficult to meet the 50 % requirement, the effect on the survival probability $P[\Phi_s]$ is less obvious. A possible explanation for this is that detections of a potential target cause the agent to take more risk flying at lower altitude and thus increasing its chances for being destroyed.

Table 3. Results for target/threat scenario

Name	Φ_s			Φ_t			Φ_m		
	β	$se(\beta)$	p-value	β	$se(\beta)$	p-value	β	$se(\beta)$	p-value
r_E				1.46	0.0195	$<10^{-4}$	1.33	0.0194	$<10^{-4}$
r_T	−2.37	0.0308	$<10^{-4}$	−1.189	0.0195	$<10^{-4}$	−1.57	0.0288	$<10^{-4}$
d_{LA}	0.377	0.0137	$<10^{-4}$	0.194	0.0137	$<10^{-4}$	0.233	0.0140	$<10^{-4}$
N_T	−0.0792	0.0041	$<10^{-4}$	−0.0943	0.0043	$<10^{-4}$	−0.0892	0.0043	$<10^{-4}$
N_E	−0.0296	0.0021	$<10^{-4}$						
p_{EFP}	−17.8130	1.3026	$<10^{-4}$						
p_{TFP}	32.7410	1.3363	$<10^{-4}$	−10.0390	1.3358	$<10^{-4}$	−3.2583	1.3569	0.0163

(d) Increasing d_{LA} increases all three predicates. This happens for two reasons. First, the agent accumulates observations done with the forward-looking sensor as it flies. Thus, the larger the look-ahead, the more time a target/threat will be within the sensor range and the more times it will be sensed. Second, using a longer horizon for the adaptation decision allows the agent to consider not only immediate, but also upcoming needs (e.g., to start increasing altitude to avoid a threat likely present three cells ahead).

(e) The target and threat sensor false-negative rates – p_{EFN} and p_{TFN} – have no predictive value on the outcome. This is surprising since the corresponding false-positive rates are predictive, and we expect the input-attribution to be symmetric. However, we validated these results by repeating our experiments with different combinations of (high and low) values of these rates. Our results showed that while changing p_{EFP} and p_{TFP} changed the $\hat{p}$ significantly, changing p_{EFN} and p_{TFN} had no effect. This demonstrates the effectiveness of our approach in producing counter-intuitive input-attributions, and its predictive value.

6.3 Paparazzi Scenario

Scenario Overview. This scenario involves multiple collaborating autonomous agents attempting to protect another agent from being clearly photographed. There is one paparazzi photographer (P) agent, one famous celebrity (C) agent, and one or more unmanned autonomous quadcopter guardian (G) agents. All agents start at random initial locations on a 3D map. Guardians must position themselves between P and C, while P moves around C to get a clear shot.

The guardians G execute an "onion-defense" formation between C and P. An onion-defense is a layered formation with the number of layers being a function of the number of guardians, and each layer forming an arc around C, resembling the layers of an onion as it is peeled. The goal is to provide redundant line-of-sight blocking for any direction P might move to attempt to get a picture of C.

The more guardians G in the formation, the more protected C is from a nimble P that tries to flank members of G. Once in stable formation, members of G try to maintain a spacing buffer $S_G = 1$ between each other.

Each guardian G has a block radius B_G which is the range around its center-of-mass that it blocks effectively. Each trial has a random number of guardians $N_G \sim \mathcal{U}\{1, 14\}$, each having a random $B_G \sim \mathcal{U}(1, 4)$ (note that $B_G \geq S_G$). P has a minimal distance D_P that he must be from C to take a useful photograph, and an initial distance I_P that he starts north from C. Once P reaches D_P from C, he moves counter-clockwise around C until he either has line of sight (success) or a 300 s timeout occurs (failure). We keep D_P a constant, but $I_P \sim \mathcal{U}(D_P, 1.5D_P)$. To prevent guardians from blocking all possible photograph angles on initialization, each member of G has an initial distance $I_G \sim \mathcal{U}(0.4D_P, 1.2D_P)$ from C in a random direction $\theta_G \sim \mathcal{U}(0, 360)$. The farther any G is from C and other G, the longer it takes to get into formation and the better chance P should have to get a good photograph of C once the actual distance A_P between P and C equals the useful photograph distance D_P.

Our experiments are built atop the Group Autonomy for Mobile Systems [2] toolkit (which provides the onion defense algorithm), the Multi-Agent Distributed Adaptive Resource Allocation [10] middleware, and V-REP.

Analysis of Results. DEMETER estimated the probability that P photographs C as $\hat{p} = 0.0023$, with a target relative error of 0.05, using 170,424 trials of which 400 satisfied the predicate (i.e., the photographer succeeded 0.23 % of the time). Total run time was 1 day 17 h and 37 min with 120 trials per batch. The Logistic Regression for linear and polynomial analysis is shown in Table 4. The ROC analysis for linear analysis had a good predictive value with an AUC of 0.73. However, the ROC for polynomial analysis was a more distinctive curve with average AUC of 0.87. Consequently, we will focus on the results of the polynomial analysis over the linear analysis.

Table 4(b) shows results for polynomial terms with p-values below 0.0002. Applying the factoring heuristics from Sect. 4.2 results in the input attribution expressions: $0.0208(N_G - 1.53\theta_1)(N_G + 2.96\theta_1)$ and $0.0939(\theta_1 - 7.82)^2$. This result illuminates the peculiarities of the onion-defense, which gets more protective with more defenders (N_G). The first defender (G_1) is especially important because it is located directly between P and C. G_2 is then placed to its left in an arc around C and G_3 to the immediate right of G_1. Each subsequent member builds

Table 4. Logistic analysis results for Paparazzi experiment.

(a) Linear Analysis

Name	β	$se(\beta)$	p-value
N_G	0.1166	0.0133	0.0000
θ_1	0.1792	0.0284	0.0000
θ_2	-0.1452	0.281	0.0000
I_P	-1.6228	0.3524	0.0000

(b) Polynomial Analysis

Name	β	$se(\beta)$	p-value
θ_2^2	0.0939	0.0170	0.0000
θ_1^2	0.0939	0.0173	0.0000
N_G^2	-0.0208	0.0044	0.0000

outward from G_1 and new layers are added behind it as N_G gets larger. θ_1 is important because P always starts due north of C, and if θ_1 is more northward, then G_1 gets to its assigned position quickly to block P. θ_2 is also important because G_2 is to the left of G_1, and if it can get into position quickly, it can block the counter-clockwise movement of P.

We were surprised by θ_1 being more important than I_1 (the initial distance from C). Through this analysis, we were able to give guidance to the onion-defense designer on potential fixes to deal with P being detected very close to C. For instance, instead of the current algorithm using fixed formations based on indices of agents, the algorithm could use intercept times to assign agent positions in formations to protect C from P. This highlights the usefulness of our approach to diagnose and fix issues in stochastic systems.

7 Conclusion

We have presented an approach for input-attribution in SMC and have implemented it in DEMETER, a distributed SMC infrastructure. We have shown that our approach synthesizes input attributions that satisfy the four conditions we stated are necessary for a good input attribution in Sect. 1 as follows: (i) by showing that the generated models have predictive power, we demonstrated that synthesized expressions correspond to actual relationships in the system; (ii) synthesized attributions are numeric in nature and backed up with confidence scores on individual coefficients and on the overall predictive power of the model; (iii) results from our experiments were able to validate our hypotheses and in some cases such as in the paparazzi scenario resulted in new and unexpected insights; and (iv) all of these results were obtained despite substantial noise from other hidden and explicit random variables in the system.

Note that while in this paper we focused on inputs, we believe that it is also possible to "watch" internal variables in the system and include them in the attribution. We also expect that other relationships, besides polynomial, could also be found by adding predictors to the logistic regression analysis, and believe this to be an important area for future work.

References

1. A platform for operating docker in production. http://github.com/rancher/rancher
2. Dukeman, A., Adams, J.A., Edmondson, J.: Extensible collaborative autonomy using GAMS. In: Proceedings of IRMAS (2016)
3. Chaki, S., Kyle, D.: DMPL: programming and verifying distributed mixed-synchrony and mixed-critical software. Technical report CMU/SEI-2016-TR-005, Software Engineering Institute, Carnegie Mellon University, Pittsburgh (2016). http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=464254
4. Clarke, E.M., Zuliani, P.: Statistical model checking for cyber-physical systems. In: Bultan, T., Hsiung, P.-A. (eds.) ATVA 2011. LNCS, vol. 6996, pp. 1–12. Springer, Heidelberg (2011). doi:10.1007/978-3-642-24372-1_1

5. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of POPL (1977)
6. Cousot, P., Cousot, R., Fähndrich, M., Logozzo, F.: Automatic inference of necessary preconditions. In: Giacobazzi, R., Berdine, J., Mastroeni, I. (eds.) VMCAI 2013. LNCS, vol. 7737, pp. 128–148. Springer, Heidelberg (2013). doi:10.1007/978-3-642-35873-9_10
7. David, A., Du, D., Guldstrand Larsen, K., Legay, A., Mikučionis, M.: Optimizing control strategy using statistical model checking. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 352–367. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38088-4_24
8. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 349–355. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22110-1_27
9. Rohmer, E., Signgh, S.P.N., Freese, M.: V-REP: a versatile and scalable robot simulation framework. In: Proceedings of IROS (2013)
10. Edmondson, J., Gokhale, A.: Design of a scalable reasoning engine for distributed, real-time and embedded systems. In: Xiong, H., Lee, W.B. (eds.) KSEM 2011. LNCS (LNAI), vol. 7091, pp. 221–232. Springer, Heidelberg (2011). doi:10.1007/978-3-642-25975-3_20
11. Ernst, M.D., Cockrell, J., Griswold, W.G., Notkin, D.: Dynamically discovering likely program invariants to support program evolution. In: Proceedings of ICSE (1999)
12. James, G., Witten, D., Hastie, T., Tibshirani, R.: An Introduction to Statistical Learning, 6th edn. Springer, New York (2015)
13. Hanley, J., McNeil, B.: The meaning and use of the area under a receiver operating characteristic (ROC) curve. Radiology **143**(1), 29–36 (1982)
14. Hosmer, D., Lemeshow, S.: Applied Logistic Regression, 3rd edn. Wiley, Hoboken (2013)
15. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22110-1_47
16. Kyle, D., Hansen, J., Chaki, S.: Statistical model checking of distributed adaptive real-time software. In: Bartocci, E., Majumdar, R. (eds.) RV 2015. LNCS, vol. 9333, pp. 269–274. Springer, Heidelberg (2015). doi:10.1007/978-3-319-23820-3_17
17. Merkel, D.: Docker: lightweight Linux containers for consistent development and deployment. Linux J. http://dl.acm.org/citation.cfm?id=2600239.2600241
18. Moreno, G.A., Cámara, J., Garlan, D., Schmerl, B.: Efficient decision-making under uncertainty for proactive self-adaptation. In: Proceedings of ICAC (2016, to appear)
19. Musliner, D.J., Engstrom, E.: PRISMATIC: unified hierarchical probabilistic verification tool. Technical report AFRL-RZ-WP-TR-2011-2097 (2011)
20. R Development Core Team: R: A Language and Environment for Statistical Computing (2008). http://www.R-project.org
21. Seshachala, S.: Docker vs VMs. http://devops.com/2014/11/24/docker-vs-vms
22. Younes, H.L.S.: Ymer: a statistical model checker. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 429–433. Springer, Heidelberg (2005). doi:10.1007/11513988_43
23. Younes, H.L.S.: Verification and planning for stochastic processes with asynchronous events. Ph.D. thesis, CMU, Technical report no. CMU-CS-05-105 (2005)