EventB2Maude: Probabilistic Modeling and Statistical Model Checking for Event-B with Probabilistic Rewrite Theories and MultiVeStA

Daniel F. Osorio-Valencia
Pontificia Universidad Javeriana Cali
frafle@javerianacali.edu.co

June 9, 2023

Abstract

Computer based systems increase every day in complexity. This tendency demands more modeling and verification features from formal methods, in order to mathematically proof properties over these systems. One of these formal methods is Event-B, which has been extended in order to cope with probabilistic behavior. In order to contribute to the verification of probabilistic Event-B models, in this work a rewriting logic semantics for probabilistic Event-B has been proposed, and it allows to encode these models as probabilistic rewrite theories in PMaude. The encoding, implemented as the EventB2Maude tool, opens the door for new statistical model checking tools like MultiVeStA to be used over probabilistic Event-B specifications. Therefore, this work presents an integration between the EventB2Maude tool and MultiVeStA. Additionally, an introductory guide for using MultiVeStA with PMaude is also given. Finally, some case studies are presented to show how the integration works and validate that the integration was done correctly, by comparing the obtained results of the simulations with their expected value.

1 Introduction

1.1 General Objective

Allow the modeling of probabilistic Event-B models encoded as probabilistic rewrite theories and their statistical model checking using MultiVeStA.

1.2 Specific Objectives

1. Document how to do statistical model checking of probabilistic rewrite theories using the MultiVeStA tool.
2. Extend the probabilistic rewrite theory $\mathscr{R}_{\mathscr{M}}$ generated by the encoding in [1], to perform statistical model checking with MultiVeStA.
3. Implement case studies with probabilistic models specified in probabilistic Event-B, as encoded with the EventB2Maude tool [2] and system properties written in MultiQuaTEx [3].

1.3 Justification

The presented project aims to make the following contributions:

1. Given the lack of documentation of the MultiVeStA tool, this project will provide a detailed and comprehensive guide for using it. This will facilitate future projects that try to do statistical model checking over probabilistic rewrite theories with MultiVeStA.
2. Even though both PVeStA [4] and MultiVeStA use the QuaTEx language [5] to specify the system properties to be verified, in practice MultiVeStA allows the specification of such properties easily, compared with PVeStA. Therefore, using MultiVeStA will help to specify and verify more interesting system's properties.
3. Comparing the model checking results obtained with EventB2Maude and MultiVeStA, with the PRISM model checker, will help to verify that the results obtained are valid.
4. The use of statistical model checking can provide insight of the properties of some probabilistic models inspired by the PROMUEVA project.

2 Preliminaries

For the following definitions, the reader is assumed to have some basic notions or familiarity with Event-B, Maude and formal methods. If not, the reader can refer to articles and books like [6, 7, 8, 9] for more in depth information.

2.1 Event-B

Event-B models can be divided in two main parts: the *context* and the *machine*. The context contains the static part of the model, which are the sets and constants, together with axiomatic properties of the model [6]. It can be defined as a tuple $C = (\bar{s}, \bar{c}, A(\bar{s}, \bar{c}))$, where $\bar{s} = \{s_1 ... s_n\}$ is a set of set identifiers, $\bar{c} = \{c_1 ... c_n\}$ is a set of constant identifiers and $A(\bar{s}, \bar{c})$ is a set of axioms over $\bar{s}$ and $\bar{c}$, that define the properties of the constants and sets of the model. The machine contains the dynamic part of the model, where behavioural properties of the model are defined [6]. The following definition of Event-B machine, is the one proposed in [10]. Generally speaking, Event-B machines can be expressed as a tuple $M = (\bar{v}, I(\bar{v}), V(\bar{v}), \text{Evts}, \text{Init})$ where $\bar{v} = \{v_1 ... v_n\}$ is a set of variables, $I(\bar{v})$ is a conjunction of predicates over the variables of the system that specifies properties of the model that must hold, $V(\bar{v})$ is an expression over the variables called variant, used for proving convergence of the model, i.e. proving that the model doesn't loops forever, Evts is a set of events and Init $\in$ Evts is an initialisation event. The events of the model have the form:

$$\textbf{event } e_i \textbf{ any } \bar{t} \textbf{ where } G_i(\bar{t}, \bar{v}) \textbf{ then } S_i(\bar{t}, \bar{v}) \textbf{ end}$$

where e_i is the name of the event, $\bar{t} = \{t_1 ... t_n\}$ is the set of parameters of the event, $G_i(\bar{t}, \bar{v})$ is a conjunction of predicates over the variables and parameters that specify the conditions that must hold for the event to occur, and $S_i(\bar{t}, \bar{v})$ are multiple variables assignments that represent the action of the event. Depending on the state of the model, i.e. the variable values or also called configuration, An event is *enabled* if and only if there exist parameter valuations that are a model for $G_i(\bar{t}, \bar{v})$ in the given configuration.

2.2 Non-deterministic choices in Event-B

Event-B models have 3 types of non-deterministic choices:

- **Choice of enabled events:** When multiple events are enabled for execution, one of them is chosen non-deterministically.
- **Choice of parameter values:** In an event with parameters, it is possible to have multiple valuations for parameters $\bar{t}$ such that the guard of the event $G_i(\bar{t}, \bar{v})$ is satisfied. Therefore, the parameter that will be used in the execution of the event is chosen non-deterministically
- **Non-Deterministic assignments:** There are 3 types of event assignments $S_i(\bar{t}, \bar{v})$: *Deterministic assignment*, *Predicate assignment* and *Enumerated assignment*. Both predicate and enumerated assignments, are non-deterministic for the following reason:
  - Predicate assignment $x :| Q(\bar{t}, \bar{v}, x, x')$: means that the variable x is assigned the value x' that satisfies the predicate $Q(\bar{t}, \bar{v}, x, x')$. In case there are multiple x' values that satisfy the predicate, then the new value of x is chosen non-deterministically.
  - Enumerated assignment $x :\in \{E_1(\bar{t}, \bar{v})...E_n(\bar{t}, \bar{v})\}$: means that the variable x is assigned on of the multiple expressions $E_i(\bar{t}, \bar{v})$ in the set. This choice of which expression will be chosen, is done non-deterministically.

Based on these 3 types of non-determinism present in Event-B, probabilistic Event-B aims to introduce probabilistic choices to replace non-deterministic choices

2.3 Probabilistic Event-B

In article [10], a probabilistic extension of Event-B is proposed that deals with non-deterministic choices in the following way:

- **Probabilistic choice of enabled events:** To solve the non-determinisim, an expression $W_i(\bar{v})$ over the variables represent the *weight* of a specific event e_i. Therefore, when multiple events are enabled, the probability of choosing one of them will be the ratio of its weight over the sum of all the weights of enabled events.
- **Probabilistic choice of parameter values:** In order to choose a parameter value probabilistically, a discrete uniform distribution can be used as a default choice to assign probabilities to the parameters. For that reason, the probability of choosing a parameter valuation $P(t_i) = \frac{1}{n}$ where n is the number of parameter valuations that satisfy the guard of the event.
- **Predicate probabilistic assignment:** A predicate probabilistic assignment written as $x :\oplus Q(\bar{t}, \bar{v}, x, x')$ chooses the new value of x with an uniform distribution. Hence, the probability of choosing a variable value x'_i is $P(x'_i) = \frac{1}{n}$ where n is the number of variable valuations that satisfy the predicate $Q(\bar{t}, \bar{v}, x, x')$. This probabilistic assignment replaces the non-deterministic predicate assignment.
- **Enumerated probabilistic assignment:** A enumerated probabilistic assignment written as $x := E_1(\bar{t}, \bar{v})@_{p_1} \oplus ... \oplus E_n(\bar{t}, \bar{v})@_{p_n}$ assigns a specific probability p_i to each expression

E_i, where $0 < p_i \leq 1$ and $p_1 + ... + p_n = 1$. This probabilistic assignment replaces the non-deterministic enumerated assignment.

2.4 Maude

This definition is based on [8, 1]. Maude is a modelling language for distributed systems based on rewriting logic [11]. In Maude and rewriting logic, data types are defined algebraically as *equations*. This equations correspond to the static part of a Maude model, and are similarly defined like how functions are defined in functional programming languages.The dynamic part of a Maude model is defined by *rewriting rules*. These rewriting rules define the model transitions from one state to another. Formally defined, a specified *rewrite theory* in Maude, is a tuple $\mathscr{R} = (\Sigma, E, L, R)$ where Σ is an algebraic signature (a set of typed operators to build the terms of the language), E is a set of equations over the terms built from Σ and typed variables X (written as $\mathscr{T}_\Sigma(X)$), L is a set of labels and R is a set of labeled rewrites rules, both *unconditional* and *conditional*. Unconditional labeled rewrite rules are written as:

$$l : t \to t'$$

And conditional labeled rewrite rules have the form:

$$l : t \to t' \textbf{ if } \text{cond}$$

where $l \in L$, t and t' are terms in $\mathscr{T}_\Sigma(X)$, and cond is a conjunction of rewrite conditions.

2.5 PMaude, MultiVeStA and MultiQuaTEx

A *probabilistic rewrite theory* [5] incorporates functions π_r function for each rule $r \in R$. Each of this functions determines a probability distribution that can be evaluated with the different state configuration of the model. Depending on the value of the function, that expresses the specific probability of a rule execution, a rule may or may not be executed. The general form of probabilistic rewrite rules, for unconditional and conditional respectively is:

$$\begin{aligned} l &: t(\overrightarrow{x}) \to t'(\overrightarrow{x}, \overrightarrow{y}) \text{ if } C(\overrightarrow{x}) \textbf{ with probability } \overrightarrow{y} := \pi_r(\overrightarrow{y}) \\ l' &: t(\overrightarrow{x}) \to t'(\overrightarrow{x}, \overrightarrow{y}) \textbf{ with probability } \overrightarrow{y} := \pi_r(\overrightarrow{y}) \end{aligned}$$

Where $\overrightarrow{x}$ is the set of variables of the current model state, $\overrightarrow{y}$ is the set of new variables accessible in the following model state and $C(\overrightarrow{x})$ is the conjunction of conditions over the set $\overrightarrow{x}$. PMaude models can be verified with statistical model checking tools like MultiVeStA [3] using the MultiQuaTEx [3] query language that allows the specification of quantitative temporal expressions to verify properties over PMaude models.

2.6 Rewriting Logic Approach to Probabilistic Event-B

The rewriting logic approach to probabilistic Event-B [1], introduces a method for translating probabilistic Event-B models to PMaude models that are executable in PVeStA. Formally speaking, it is a map $[\![\cdot]\!] : (\mathscr{C}, \mathscr{M}) \to \mathscr{R}_{\mathscr{M}}$ from a probabilistic Event-B model to a rewrite theory. There are two main steps for this transformation:

1. Specify a rewrite theory $\mathscr{R}$ to encode the static parts of the model, i.e. the context and the initialization of the machine.
2. Then, $\mathscr{R}$ is extended with equations and rewrite rules that correspond to the events in the machine $\mathscr{M}$.

The first step is represented by the term:

$$\langle\ [\![\mathscr{C}]\!]_{id} : Context \mid sets : [\![\mathscr{C}]\!]_{sets},\ constants : [\![\mathscr{C}]\!]_{constants}\ \rangle$$
$$\langle\ [\![\mathscr{M}]\!]_{id} : Machine \mid variables : [\![\mathscr{M}]\!]_{initVars}\ \rangle$$
$$\langle\ events : Events \mid state : [\![\mathscr{M}]\!]_{initEvtSt}\ \rangle$$

where $[\![\mathscr{C}]\!]_{id}$ and $[\![\mathscr{M}]\!]_{id}$ encode the context and machine identifier respectively, $[\![\mathscr{C}]\!]_{sets}$ encodes the deferred sets, $[\![\mathscr{C}]\!]_{constants}$ encodes the constants, $[\![\mathscr{M}]\!]_{initVars}$ encode the initial value of each one of the variables in the machine, and $[\![\mathscr{M}]\!]_{initEvtSt}$ sets the initial state of all the events (which can be *unknown*, *blocked* or *enabled* with a specific weight) to *unknown*. To refer to this term, the following notation will be used:

$$\mathfrak{C}\ \mathfrak{M}_0\ \mathfrak{E}$$

Where $\mathfrak{C}$ is the term that represents the context, $\mathfrak{M}_0$ is the term that represents the machine in the initial state, and $\mathfrak{E}$ is the term that represents the events' states in the current system configuration. The rest of the system states, are symbolized with $\mathfrak{C}\ \mathfrak{M}_i\ \mathfrak{E}$, and are obtained with the application of equations and rewrite rules, defined in $\mathscr{R}_{\mathscr{M}}$, over the initial state $\mathfrak{C}\ \mathfrak{M}_0\ \mathfrak{E}$. These rules and equations interact with the initial term as follows:

1. The system is initialized with term $\mathfrak{C}\ \mathfrak{M}_\mathfrak{o}\ \mathfrak{E}$ in $\mathscr{R}_{\mathscr{M}}$:

$$\mathfrak{C}\ \mathfrak{M}_\mathfrak{o}\ \mathfrak{E}$$

2. For any state of the model, symbolized by $\mathfrak{M}_\mathfrak{i}$, the event guards are evaluated using the equation $evalSt$, which changes the events states depending on the guards and the current configuration of the system.

$$\mathfrak{C}\ \mathfrak{M}_\mathfrak{i}\ \langle state : ev(e_1, unknown)...ev(e_n, unknown)\rangle \xrightarrow{evalSt} \mathfrak{C}\ \mathfrak{M}_\mathfrak{i}\ \langle state : ev(e_1, st_1)...ev(e_n, st_n)\rangle$$

3. The blocked events are filtered out and the next event is chosen probabilistically from all the enabled events, based on the event's weights:

$$\mathfrak{C}\ \mathfrak{M}_\mathfrak{i}\ \langle state : ev(e_1, st_1)...ev(e_n, st_n)\rangle \xrightarrow{chooseEvt} \mathfrak{C}\ \mathfrak{M}_\mathfrak{i}\ \langle state : ev(e_i, execute)\rangle$$

4. The rule associated to the encoded version of the chosen event e_i in the previous step is executed. The resulting change of the variable values, is captured in the rewrite $\mathfrak{M}_\mathfrak{i} \xrightarrow{execEvt_{e_i}} \mathfrak{M}_\mathfrak{j}$. Moreover, the states of the events are initialized again.

$$\mathfrak{C}\ \mathfrak{M}_\mathfrak{i}\ \langle state : ev(e_i, execute)\rangle \xrightarrow{execEvt_{e_i}} \mathfrak{C}\ \mathfrak{M}_\mathfrak{j}\ \langle state : ev(e_1, unknown)...ev(e_n, unknown)\rangle$$

5. Repeat steps 2, 3 and 4 until deadlock (i.e. none of the events can be executed since the guards of the events are unsatisfiable), or until the rule $execEvt_{e_i}$ has been executed a fixed number of times (this value is represented as a constant named `MAX-STEPS` in $\mathscr{R}_{\mathscr{M}}$).

2.7 MultiVeStA and $\mathscr{R}_{\mathscr{M}}$

First and foremost, the MultiVeStA tool was developed by Andrea Vandin, and it is available to download in a public repository in GitHub [12]. The main components of the tool are:

- **Maude-3.0+yices2:** A folder that contains the Maude distribution used by the tool. It is important to note, that MultiVeStA is not integrated with Maude. It only uses this folder to run the simulations.
- **apmaude.maude:** A Maude file that contains the necessary sorts, equations and rules to extend any PMaude specification, so it can be checked using the tool.
- **multivesta.jar:** A java executable that runs the simulations using the specified PMaude model, the MultiQuaTEx query, and the simulation parameters.

To run simulations of a specific PMaude specification, like the probabilistic rewrite theory $\mathscr{R}_{\mathscr{M}}$, it is necessary to define:

- **model.maude:** A Maude file that contains the translated version of the Event-B model in Maude.
- **query.multiquatex:** A MultiQuaTEx file, with the quantitative temporal logic formula written in the MultiQuaTEx language, that will be used to verify the model.
- **Simulations Parameters:** There are multiple parameters to take into account when performing simulations with the tool:
  - α and δ, used to calculate define the confidence interval.
  - bs, which defines the number of simulations that each batch will contain.

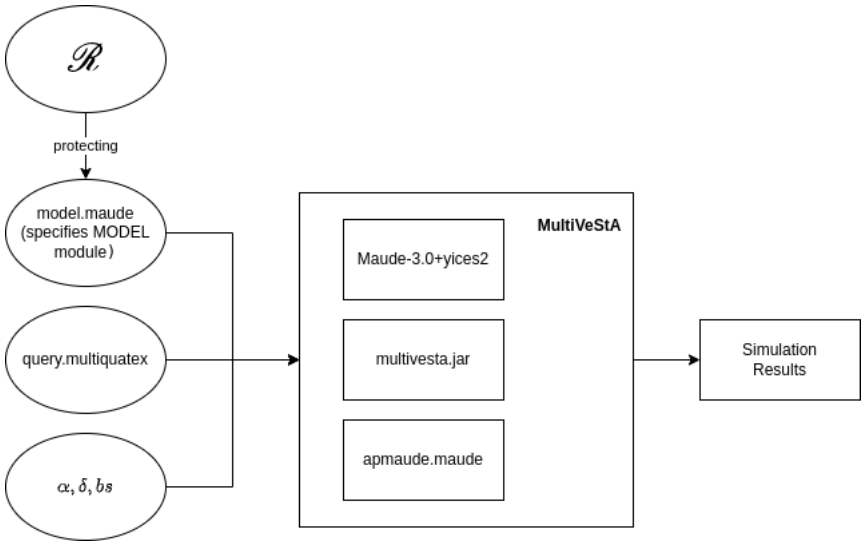

Figure 1: MultiVeSta's components

With these components, it is then possible to run the simulations and obtain the results of the query,

as seen in Figure 1. The way MultiVeStA obtains these simulation results can be defined in the following steps:

1. MultiVeStA performs bs number of simulations and calculates the confidence interval based on the results obtained in each one of the simulations. Each simulation consists of:
   (a) Get the initial state of the system $\mathfrak{C}\ \mathfrak{M}_0\ \mathfrak{E}$ by running *model.maude*, which includes the theory specified in *apmaude.maude*, using the *Maude-3.0+yices2* distribution.
   (b) Evaluate the MultiQuaTEx formula in *query.multiquatex* over $\mathfrak{C}\ \mathfrak{M}_0\ \mathfrak{E}$. If the formula evaluates to true, then the simulation ends and the result of the query is returned. Otherwise, MultiVeStA performs one step in the simulation, to get the next state $\mathfrak{C}\ \mathfrak{M}_i\ \mathfrak{E}$.
   (c) Repeat step b with the new state $\mathfrak{C}\ \mathfrak{M}_i\ \mathfrak{E}$, until reaching a state where the formula is true and the results are returned.

   The result of a MultiQuaTEx formula is a floating point value. Therefore, with each one of the values returned by the query in each simulation, the confidence interval of the batch is calculated.
2. If the confidence interval satisfies the parameter α and δ i.e. the real value of the query x has a probability $(1-\alpha) * 100\%$ of being in the interval $[\bar{x} - \frac{\delta}{2}, \bar{x} + \frac{\delta}{2}]$, where $\bar{x}$ is the current value calculated for the query using the simulations, then MultiVeStA returns the results and ends it execution. If not, then MultiVeStA will send another batch of simulations until the confidence interval is finally reached.

3 Simulation Results

In total there are 3 case studies: two probabilistic programs using dices and coins, a bounded retransmission protocol, and a program that simulates a modal operator in positive negative logic. The repository that contains the code of this integration can be found in reference: `https://github.com/dfosorio/EventB2Maude-MultiVeStA`. Furthermore, this repository contains the code of the probabilistic Event-B models, their translated versions in PMaude, and the results of the simulations of the case studies. A short guide on how to use the software can also be found in the repository.

3.1 Dice Programs

The dice programs, based on the Knuth & Yao paper [13], were found in the PRISM model checker web page [14]. The first query verifies the probability that the dice lands in a specific value from 1 to 6. Using the parameters $\alpha = 0.01, \delta = 0.01$ and $bs = 28$, the results of the simulation using MultiVeStA are:

Property	ObtainedValue	Variance	CI
PDice1(obs1)	0.169194084235345	0.140571212336999	0.00999772484533201
PDice1(obs2)	0.165127336320909	0.137864066309953	0.00999898248710578
PDice1(obs3)	0.166322650733297	0.138663192561158	0.00999737034957979
PDice1(obs4)	0.164898271712973	0.137710597576251	0.00999724079076279
PDice1(obs5)	0.166095890410959	0.138511810325858	0.00999571303811207
PDice1(obs6)	0.167870619946092	0.139693840237627	0.00999651791545913

Table 1: Results of the first dice program

The number of simulations performed to obtain the results were 37324, and the time it took to run them was 2521.757 seconds (42 minutes). From this simulation, it is safe to say that the obtained results are close to the expected real value of $\frac{1}{6}$. In the second program two dice are used, and the property that wants to be checked is the probability of obtaining a specific value from 2 to 12, which corresponds to the sum of the two dices. Using the same parameters the results where:

Property	ObtainedValue	Variance	CI
PDice2(obs1)	0.03	0.0291037312475958	0.00995116917577923
PDice2(obs2)	0.0510077519379845	0.0484097138712153	0.00997972279089514
PDice2(obs3)	0.0853365384615385	0.0780579664517895	0.00997985024064457
PDice2(obs4)	0.114200743494424	0.101162694374703	0.00999035979240436
PDice2(obs5)	0.138710691823899	0.119473792835145	0.00998550057004364
PDice2(obs6)	0.164328767123288	0.137328585846037	0.00999266014168248
PDice2(obs7)	0.135176848874598	0.116907827497064	0.00998823325270086
PDice2(obs8)	0.110306513409962	0.0981427467677965	0.0099897800415653
PDice2(obs9)	0.0811616161616162	0.0745781747981354	0.00999816628115168
PDice2(obs10)	0.0551798561151079	0.0521387905863524	0.00997746359029961
PDice2(obs11)	0.0282191780821918	0.0274266131408506	0.0099855441351178

Table 2: Results of the second dice program

In total, the number of simulations performed to obtain the results were 36500, and the time it took to run them was 4583.059 seconds (76 minutes). In the end, the results of the simulation are very close to the expected probability for each one of the possible values of the sum of the two dice.

3.2 Bounded Re-transmission Protocol

The bounded re-transmission protocol is based on the sixth refinement of the file transfer protocol in Chapter 6 of Abrial's book [7]. The property that wants to be checked is the probability of success of the transmission protocol is, with different number of maximum retries (represented with the parameter MAX). Using parameters $\alpha = 0.01, \delta = 0.01$, $bs = 100$, a fixed size for the file $N = 100$, and multiple values for the maximum number of retries MAX, the results of the simulation are:

File size	*MAX*	Simulations	Time in seconds	Property	ObtainedValue	Variance	CI
100	0	100	7.96	PRetrans5(obs1)	1	0	0
				PRetrans5(obs2)	0	0	0
100	5	100	19.633	PRetrans5(obs1)	1	0	0
				PRetrans5(obs2)	0	0	0
100	10	4200	4695.782	PRetrans5(obs1)	0.984285714285714	0.0154710305174701	0.0098873988006215
				PRetrans5(obs2)	0.0154761904761905	0.0152403066489754	0.00981339506242592
100	15	13200	49320.0	PRetrans5(obs1)	0.636742424242424	0.231319033581516	0.0215658109731058
				PRetrans5(obs2)	0.359772727272727	0.230353763026125	0.0215207679775556
100	20	8900	51120.0	PRetrans5(obs1)	0.212111111111111	0.167138558605277	0.0222005711076664
				PRetrans5(obs2)	0.786111111111111	0.168159116445037	0.0222682469512274
100	25	7700	49560.0	PRetrans5(obs1)	0.0507792207792208	0.0482069521594135	0.0128901078154959
				PRetrans5(obs2)	0.949090909090909	0.0483236311681564	0.0129056978494018
100	30	2800	19700.462	PRetrans5(obs1)	0.0103571428571429	0.0102535344255602	0.00985836415788304
				PRetrans5(obs2)	0.989642857142857	0.0102535344255602	0.00985836415788304

Table 3: Results of the bounded re-transmission protocol

Where each `PRetrans5(obs1)` corresponds to the probability of failure, and `PRetrans5(obs2)` corresponds to the probability of success. As seen in Table 3, the probability of success is directly proportional to the maximum number of retries. Conversely, the probability of failure is inversely proportional to the maximum number of retries. This is the expected behavior, since incrementing the maximum number of retries of the protocol should increase the probability that the file is sent completely. It is important to note, that not all of the simulations reached the desired confidence interval ($CI < 0.01$). The reason behind this, is that the time it was taking to reach the desired confidence interval was overly long, as seen in the previous table. Hence, some of the simulations were stopped before reaching the desired confidence interval. Nevertheless, the obtained confidence interval for the stopped simulations is good enough ($CI < 0.023$) to approximate the probability of reaching a success or failure state.

3.3 Balance Property in Signed Frames

Positive negative logic or *PNL* is a modal logic introduced in [15, 16], used to define properties over a specific class of network. One of the extensions proposed to PNL in the article [15], are the modalities $\langle /\!\!/\!\!\backslash + \rangle$ and $\langle /\!\!/\!\!\backslash - \rangle$, which can be explained as adding a positive and negative link somewhere in the network. One way to represent the previously explained behavior, is with a dynamic system that takes an initial *signed frame* [15, 16], and non-deterministically adds positive and negative links until reaching a terminating state, i.e. all the links between the nodes of the graph have been added. Based on this model, the property that wants to be verified is the probability of reaching the *balance property* [15, 16], when all the links in the graph have been added. Two experiments where made:

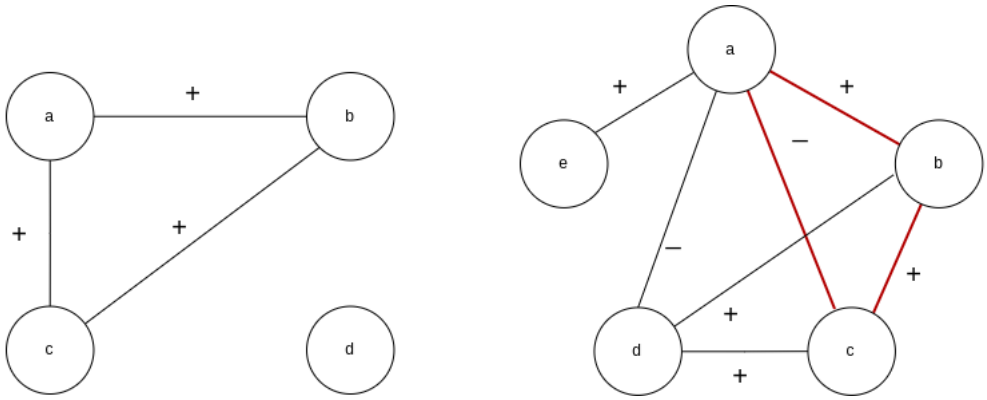

Figure 2: Signed frames of experiment 2 (left) and experiment 3 (right)

For the first experiment (left side of Figure 2), the chances of reaching a terminating state that is balanced are two out of eight (25% chance), and for experiment 2 (right side of Figure 2) the probability is 0%, since the initial configuration already includes a triangle that is not balanced, which is highlighted in red. The obtained results with parameters $\alpha = 0.01, \delta = 0.01$ and $bs = 100$ are:

Experiment	Time in seconds	Simulations	ObtainedValue	Variance	CI
1	2639.935	46400	0.254341252699784	0.189655876114114	0.010426528417539
2	5.631	100	0	0	0

The results obtained in the simulations are in accordance with the real expected value of the queries. For the first experiment, the `NaN` value was obtained before reaching the desired confidence interval, therefore the estimation of the expected value of the query was done with the value obtained in the previous batch of simulations. Still, the confidence interval is good enough for an approximation of the expected value of the query.

4 Conclusions and future work

This work provided a functional implementation that integrates the probabilisitic Event-B to PMaude translation process, known as EventB2Maude, with the statistical model checking tool MultiVeStA. By implementing different case studies, it was possible to demonstrate the functionality of MultiVeStA and the EventB2Maude tool, by obtaining simulation results that match the real expected values of the MultiQuaTEx queries. Unfortunately, it is important to note that there exists some difficulties with the current state of the integration:

- As the reader might have noticed in the verification results using the MultiVeStA tool, the time to reach the desired confidence interval might take too long, when the required confidence interval determined by the δ parameter is difficult to obtain. For example, a value of $\delta = 0.01$ proved to be challenging in terms of time for some of the simulations, specially for the bounded re-transmission protocol. For the moment, we do not have a concrete explanation for the large amount of time it takes to verify this particular model, but it seems to be related to the large number of events that are executed in each simulation.

- Related with the previous point, if more decimal precision is needed for the obtained value of the MultiQuaTEx query, then the δ parameter must be decreased. Thus, having a decimal precision for more than three decimal places implies a $\delta < 0.01$, which translates into a large amount of simulation time in the current condition of the tool, using only one thread for execution, i.e. no parallelism, and the value $\alpha = 0.01$.
- For the time being, there is not a satisfactory answer that explains why in some cases the confidence interval reaches the `NaN` value and stops the simulations. Nevertheless, the answer for the specific query can be obtained from the result of the previous batch of simulations.

Regarding future work, there are multiple things that can be done:

- Investigate and understand why in some of the simulations the value of the confidence interval reaches the `NaN` value.
- Find ways to reduce the overall simulation time when the model has many system transitions or the value for the confidence interval is difficult to reach, due to the δ parameter.
- Implement a more user friendly user interface that allows to make translations of probabilistic Event-B models and verification with MultiVeStA with more ease.
- Extend the EventB2Maude tool to allow the specification of more elements of the Event-B language, like the machine invariants or context axioms, defined with first order logic.
- Implement the EventB2Maude tool as a functional Rodin plug in.
- Explore how the model checking tool in Maude can be used to check the deterministic part of the translated Event-B models.
- Use MultiVeStA's option for using parallelism to run simulations on multiple threads, and verify if the simulation time is substantially reduced.

References

[1] C. Olarte, C. Rocha, and D. Osorio, "A rewriting logic semantics and statistical analysis for probabilistic event-b," 2022.

[2] C. Olarte, D. Osorio, and C. Rocha, "Eventb2maude." `https://github.com/carlosolarte/EventB2Maude`, 2022.

[3] A. Vandin and S. Sebastio, "Multivesta: Statistical model checking for discrete event simulators," 01 2014.

[4] M. AlTurki and J. Meseguer, "Pvesta: A parallel statistical model checking and quantitative analysis tool," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6859 LNCS, pp. 386–392, 2011.

[5] G. Agha, J. Meseguer, and K. Sen, "Pmaude: Rewrite-based specification language for probabilistic object systems," *Electronic Notes in Theoretical Computer Science*, vol. 153, pp. 213–239, 5 2006.

[6] M. Butler, A. S. Fathabadi, and R. Silva, "Event-b and rodin," *Industrial Use of Formal Methods: Formal Verification*, pp. 215–245, 1 2013.

[7] J. R. Abrial, *Modeling in event-b: System and software engineering*, vol. 9780521895569. Cambridge University Press, 1 2011.

[8] P. C. Ölveczky, *Designing Reliable Distributed Systems*. Springer London, 2017.

[9] M. Clavel, F. Durán, S. Eker, S. Escobar, P. Lincoln, N. Martí-Oliet, J. Meseguer, R. Rubio, and C. Talcott, "Maude manual (version 3.2.1)," 2022.

[10] M. A. Aouadhi, B. Delahaye, and A. Lanoix, "Moving from event-b to probabilistic event-b," *Proceedings of the ACM Symposium on Applied Computing*, vol. Part F128005, pp. 1348–1355, 4 2017.

[11] R. Bruni and J. Meseguer, "Semantic foundations for generalized rewrite theories," *Theoretical Computer Science*, vol. 360, pp. 386–414, 2006.

[12] A. Vandin, "Multivesta." `https://github.com/andrea-vandin/MultiVeStA/wiki/Integration-with-PMaude-specification`, 2022.

[13] D. Knuth, "The complexity of nonuniform random number generation," *Algorithms and Complexity, New Directions and Results*, pp. 357–428, 1976.

[14] prismmodelchecker.org, "Dice programs." `https://www.prismmodelchecker.org/casestudies/dice.php`.

[15] M. Young Pedersen, S. Smets, and T. Ågotnes, "Modal Logics and Group Polarization," *Journal of Logic and Computation*, vol. 31, pp. 2240–2269, 10 2021.

[16] Z. Xiong and T. Ågotnes, "On the logic of balance in social networks," *Journal of Logic, Language and Information*, vol. 29, no. 1, pp. 53–75, 2020.