Towards a Probabilistic Extension to Non-Deterministic Transitions in Model-Based Checking

Sergey Staroletov[1]

Polzunov Altai State Technical University,
Lenin avenue 46, Barnaul, 656038, Russia
serg_soft@mail.ru

Abstract. The more the software systems complexity increases, the harder to describe models for them; in addition, these models should be adequate. For many types of systems, we can create models by replacing complex algorithms of interaction with occurrences of certain non-deterministic events. The paper is considered the problem of modeling probabilistic transitions on Promela language (SPIN verification tool) and technique of enhancing its grammar for the purpose of probabilistic verification. The extension is based on non-deterministic choice operator in Promela.

Keywords: Model Checking, Promela, Probabilistic Model Checking

1 Research objectives

Software engineering world has already developed sufficient amount of testing technologies as processes of searching inconsistencies between programs and requirements for them, and they are applicable well when creating typical desktop applications or web sites with their business logic. However, for systems that need to work for critical industries and technologies, such as embedded control systems, aircraft management entities, operating system components, the testing process does not guarantee sufficient output quality of the product and the appearance of an error can lead to costly consequences. It is because testing is not able to prove the absence of errors over all possible states and only can show their lack on a particular test.

Formal verification process (in comparison with testing) enables us to prove the infallibility of some formal model in relation to the requirements of the system, expressed in a predicate logic, and we can talk about the correctness of the model on all data inputs and all specified methods of interaction of model entities.

This study applies formal models in Promela (*"protocol meta-language"*) for SPIN verification tool (*"simple Promela interpreter"*) [1, 2], and system requirements are defined in terms of linear time temporal logic (LTL).

The more the software systems complexity increases, the harder to describe models for them; in addition, these models should be adequate. For many types of systems, we can create models by replacing complex algorithms of interaction with occurrences of certain non-deterministic events.

The goal of the paper is to move from the non-determinism to probabilities. The two words mean some variations of unpredicted behaviour, but when we talk about the

last one, we operate with probability values as guards of program actions, and they can be either *a priori* and *a posteriori* (pre-defined before the execution and calculated after some cases of it).

Promela language already includes *non-deterministic transitions in the IF clause*, but in the simulation mode the corresponding actions are selected randomly and equiprobably. It can lead to a problem of simulation of complex protocols, distributed and network software, with given initial requirements with probabilities.

In this work, an approach to extend Promela language with probability-based constructions is presented for simulating behaviours with given probabilities in the system description. To solve this problem, some SPIN verifier internals are studied and described. Moreover, a method to probability driven programming is shown, and some further steps to move to Probabilistic model checking are described.

2 Related work

2.1 Background of the proposed method

In the articles [3, 4] we considered creating a model of interoperational programs on different levels of abstraction: from the code level to the logic level – the interaction between components of a distributed system, constructed with a goal to its verification and testing. As well as we proposed methods of software development (according to the Model-driven developing approach), when the development begins with a model, then the code is generated and can be verified subsequently. For the verification, we provided the methods for transforming models into Promela language. This language was chosen because the programs on it are created in accordance with the *"Actors"* approach [5] that involves the interoperation between low coupling processes that interact through messages. This approach allows modeling distributed systems, multi-threaded systems, multi-component systems that interact through API calls.

The developed model has probabilistic transitions between states of an extended finite state machine. In Promela language there are not probabilistic transitions explicitly, but there are non-deterministic conditions, which became the basis for creating the technology proposed here for transformation non-deterministic model into probabilistic one. In [3] it was considered a way to generate additional duplicate code to satisfy a given guard probability in the simulation, but it is a very naive approach.

2.2 Probabilistic model checking

According to [6] the Probabilistic model checking problem is stated as follows:
Given a property ϕ, the Problem consists of checking whether a stochastic system satisfies ϕ with a probability greater than or equal to a certain threshold θ.

Herein we consider to LTL properties ϕ. For an example to understand the problem, refer to the *Roundabout Maneuver* [7] – is a behavior of two aircraft to make *Collision Avoidance*, and it is a subject to cooperation in air traffic control. The avoidance of collision is achieved by an agreement on some common angular velocity ω_{xy} and common centre c_{xy} around which both can fly by the circle safely without coming close to

each other not more than R_{safe} [7]. There is the following precondition to the entry procedure of the Maneuver and the safe property to verify:

$$isSafe ::= (x_1 - y_1)^2 + (x_2 - y_2)^2 \leqslant R_{safe}^2 \tag{1}$$

where $x = (x_1, x_2)$ is a first planar position, $y = (y_1, y_2)$ is a second planar position.

Model checking approach to this problem can be applied by specifying an LTL property (for example, in SPIN notation):

$$[\,]\,(isSafe == 1) \tag{2}$$

where "[]" is the "Globaly" LTL operator and 1 is an alias for true. (2) in natural language means "during execution in all the states of the program model (1) will be preserved as true".

When we move to the Probabilistic model checking, we can specify a PLTL property like

$$[\,]_{\,\theta>90\%}\,(isSafe == 1) \tag{3}$$

when we like to verify that the system satisfies the safe property with probability of 90% or higher (that means the system will be safe in 90% cases for some reasons when it start working from an initial state and finish in a final state; or: in 10% cases or less two aircraft can be closer than R_{safe}).

2.3 Probabilistic languages and models to verify

In the work [8] the Probabilistic model checking problem was faced and an approach was presented to ensure that a given labeled Markov chain satisfies a given LTL property with a given probability. In [9] the approach is extended to verify these things effectively using an LTL-BDD representation of formulas.

The authors of the publications [10, 11] state that Markovs' models cannot be useful at the system description level since they are not able to describe parallel processes. So, to evaluate aspects such as the probability of system failure they construct Discrete Time Markov Chain (DTMC) models from the composition of the Probabilistic Component Automata (PCA) representations of each component and then analyse them in tools such as PRISM [12].

In the work [13] the SMC (Statistical model checking) approach has been introduced, and the tool Uppaal [14] has been extended to check system properties using an extended automaton model. The tool offers probabilistic simulation and verification by specifying in the user interface probability distributions that drive the timed behaviors, and the engine offers computing an estimate of probabilities and comparison between estimated probabilities without actually computing them.

In the book [16] the author decided to implement classes in Scala language to allow developers to make complex probabilistic and statistical programs.

In the paper [17] the authors decided to implement from scratch a language similar to Promela with some probabilistic additions with the probabilistic model checking

goals. They used SOS-rules in the Plotkin style [15] to describe the language formally. For example, the language offers to write code with PIF (probability IF clause):

$$\textbf{PIF}\ [p1] \Rightarrow P1 ... [pn] \Rightarrow Pn\ \textbf{FIP} \tag{4}$$

That means: probabilities p1..pn lead the code P1..Pn to execute. In the current paper, we are going to follow this approach, but without creating a "yet another" language and a tool, because now Promela language and SPIN tool are well-designed and community-approved, and new language creation instead of extension of an existing one should be well justified. Of course, it is possible to declare some inductive rules for a theorem prover for making the evidence of particular problems like it is done in [18], but extending a modeling language to some conventional structures and rules can involve additional engineers to the process of formal proof of software.

2.4 How to create non-deterministic transitions in Promela

It is known that the programs in Promela are modeled in the form of instructions in a special language, at the same time like C and like functional languages such as Erlang (it refers to "Actors" approach). The language contains conditions in the form

```
if
  :: boolean expression1 -> actions1
  :: boolean expression2 -> actions2

  :: boolean expressionN -> actionsN
fi
```

It is considered that in order to perform a certain action is necessary to the corresponding logical expression was true. However, if multiple logical expressions are true in the same if constructions, it is considered that the next step is one of the non-deterministic choices of them. Consider the code snippet on Promela, modeling the *Leader Selection algorithm* (Dolev, Klawe & Rodeh [19]):

```
if      /* non-deterministic choice */
  :: Ini[0] == 0 && N >= 1 -> Ini[0] = I
  :: Ini[1] == 0 && N >= 2 -> Ini[1] = I
  :: Ini[2] == 0 && N >= 3 -> Ini[2] = I
  :: Ini[3] == 0 && N >= 4 -> Ini[3] = I
  :: Ini[4] == 0 && N >= 5 -> Ini[4] = I
  :: Ini[5] == 0 && N >= 6 -> Ini[5] = I
fi
```

with N equal to, for example, 3 and the zero value of Ini, it has four variants of non-deterministic steps in the Promela model to continue at this point. In the simulation mode of SPIN, a necessary option can be selected by a user in the interactive dialog or enabled by using a random number generator with a given initial value.

If there is a software system that works in some way and we have calculated the probability of occurrence of certain events in it, for example, transitions to different

states, then to simulate such a system on Promela we can propose using a technology to increase the probability: for events that are more likely proceed to simply increase the number of identical conditions and actions.

For example in this case the probabilities of both actions are identical (50 and 50 percents):

```
if
  :: boolean expression1 -> actions1
  :: boolean expression2 -> actions2
fi
```

If we want to increase the likelihood of action1, the following code

```
if
  :: boolean expression1 -> actions1
  :: boolean expression1 -> actions1
  :: boolean expression2 -> actions2
fi
```

does not change the logic of the transition, it changes the likelihood of the first step in the model. And the probabilities of selection this actions equal to 66.(6) and 33.(3) percents. To check this assumption, we can write a program in Promela, which counts the number of execution of one of the branches into p variable:

```
mtype = {s1, s2, s3};
active proctype main() {
mtype state;
int p = 0;
int count = 0;
do
::{
  state = s1;
  do :: {
    if
    :: (state == s1) -> {
      if
      :: true ->  {state = s2;  p = p + 1}
      :: true ->  {state = s2;  p = p + 1}
      :: true -> state = s3
      fi
    }
    :: (state == s2) -> state = s3;
    :: (state == s3) -> break;
    fi;}
  od
  count = count + 1;
  }
  :: (count >= 100) -> goto fin;
```

```
od
fin:
printf ("p = %d", p);
}
```

As result, starting the model execution with different values of the random number generator, we will receive the value of p = 60 ... 70.

In the current SPIN implementation, a next running node in a non-deterministic transition is selected using seed-based random generator, see run.c:

```
/* CACM 31(10), Oct 1988 */
Seed = 16807*(Seed%127773) - 2836*(Seed/127773);
if (Seed <= 0) Seed += 2147483647;
return Seed;
```

If we need to create some emulation of probabilistic transition with a given probability, it is possible to generate a large number of repetitive conditions [3]. However, the disadvantages of this approach are: duplicated code, generation a large number of conditions, difficulties with manual testing this code.

3 The proposed solution

We propose first to extend the Promela grammar with an annotation of probabilistic transition:

```
if
  :: [prob = value %] condition -> actions
...
fi
```

where the "prob value" (probability value) is an integer of [0..100]. In this case, we can get rid of unwanted repetitive conditions, and perform the transition with a probability within the SPIN source code of the analysis of transitions with using abstract syntax tree.

SPIN is shipped in the source code, now available in Github repository. Promela language parser is implemented using Yacc tools. A fragment of the grammar, introducing the changes described here (modified source from spin.y):

```
options : option
| option options
;
option  : SEP
prob
sequence OS
;

prob    :/* empty */
| '[' PROB ASGN const_expr '%' ']'
;
```

where prob – is the new grammar rule for probability transition, SEP is the "::" terminal, PROB – is the new terminal symbol "prob", ASGN is the assignment ("=") and const_expr is Promela's non-terminal (grammar rule) for constant expression that is calculated at the parsing time. This grammar extension can now allow the previous code snippet as well as usual Promela constructions.

Further, we propose to extend this idea, putting the probability of not only as a constant number but the value of a variable. In this case, the probability may dynamically be recalculated during program execution on Promela. So, the following Promela grammar addition is presented:

```
prob    :/* empty */
| '[' PROB ASGN const_expr '%' ']'
| '[' PROB ASGN expr ']'
;
```

where expr – is a grammar rule for the Promela expression.

Thus we will be able, for example, simulate and verify various training and recognition algorithms. A new probability value will be received from a different process, changed in code based on a given probabilistic transition or other different cases. So, with simple grammar additions and some changing to semantic of SPIN code, we can speak of *"Probability driven programming"* in Promela.

To continue this idea, we can propose some "syntactic sugar" for probabilistic actions with Promela channels. To simulate some networking protocols it is required to specify a loss/reliability value for a channel, that means: with a given probability a channel lose/guarantee some messages and protocols should correctly handle this. Our new additions for the channels:

```
one_decl: vis TYPE chan_loss osubt var_list
...
;
chan_loss    : /* empty */
| '[' LOSS ASGN const_expr '%' ']'
| '[' RELY ASGN const_expr '%' ']'
;
Special : varref RCV
  rargs
| varref SND msg_loss
  margs
;
msg_loss    : /* empty */
| '[' LOSS ASGN const_expr '%' ']'
| '[' RELY ASGN const_expr '%' ']'
| '[' RELY ASGN expr ']'
| '[' LOSS ASGN expr ']'
;
```

where chan_loss is the new grammar rule to define channels, msg_loss – is the new rule to annotate the message send Promela operator (SND terminal); LOSS – is our terminal

symbol 'loss' (to specify channel loss values after it) and RELY – is our terminal symbol 'rely' (to specify channel reliability values after it).

So, after the modifications, the extended Promela grammar should accept the following test code with new annotations:

```
mtype = {s1, s2, s3};
int count = 0;
/* now define a normal chan */
chan a = [1] of { short };
/* now define a chan with 30% loss value */
chan [loss = 30%] b = [1] of { short };
/* now define a chan with 70% rely value (or 30% loss) */
chan [rely = 70%] c = [1] of { short };

int pp = 70;

active proctype main() {
  mtype state = s1;
  count = pp - 10;
  /* send to a normal chan */
  a ! 1
  /* send to a 30% loss chan */
  b ! count;
  /* send to a chan with re-defined 10% loss */
  c ! [loss = 10%] 3;
  /* send to a chan with 70% reliability */
  c ! 2;
  /* send to a chan with re-defined 99% reliability */
  c ! [rely = 99%] 4;
  /* send to a chan with re-defined loss with count value */
  b ! [loss = count ] 1;
  if
    :: true -> state = s2;
    /* 10% prob transition */
    :: [prob = 10%] true -> state = s3;
    /* dynamic prob transition */
    :: [prob = pp] true -> state = s3;
    /* for the others prob will be */
       calculated as 100% - all specified probs */
    :: true -> state = s2;
    :: true -> state = s3;
  fi
  printf ("state = %d", state);
}
```

4 Status and perspectives of the solution

Currently, we implemented the extension to the grammar with goal to extend the semantics and understanding of the SPIN internal logic. The advanced program simulation in Promela as well as verification code generation are in progress. Adding probabilities to the transitions and channels with losses will allow SPIN users to model some complex interactions and probabilistic algorithms. We are making some additions on internal nodes (at the parsing time), some – on dynamic values calculation after SPIN processes are scheduled.

With regard to verification, making the addition to Promela with transitions for a given value of probability should allow us to use probabilities in the system requirements as LTL predicates, after which it would be possible to prove the statements of the forms "if the transition probability is given, it satisfied the requirement ...", "if a requirement is satisfied, then the probability is ..."; moreover, it is possible to show a counterexample as a sequence of states that can be performed too frequently or too infrequently.

Of course, it is easy to implement our approach with probabilities as given constants - however, much more interesting is to represent the implementation of probability values, given by the variables in run time. We would like to experiment with it after doing some programming stuff with SPIN sources, and we are going to compare the resulting approach with existing probabilistic model checking approaches which have been mentioned in the paper.

The advantage of this approach is an attempt to made solving the problems of Probabilistic model checking by using the existing community-approved instrument with relatively little efforts of additions; this approach also allows verification engineers to extend the class of verifiable systems with the SPIN verifier.

References

1. Holzmann, Gerard J. "The model checker SPIN." IEEE Transactions on software engineering 23.5 (1997): 279-295.
2. Spin - Formal Verification. http://spinroot.com
3. S. Staroletov. Model of a program as multi-threaded stochastic automaton and its equivalent transformation to Promela model. Ershov informatics conference. PSI Series, 8th edition. International workshop on Program Understanding. Proceedings.At: Novosibirsk, Russia. 2011
4. S. Staroletov. Model Driven Developing & Model Based Checking: Applying Together / Tools & Methods of Program Analysis TMPA-2014:Kostroma, 2014. - pp. 215-220. ISBN 975-5-8285-0719-1. Presentation: http://www.slideshare.net/IosifItkin/model-driven-developingmodelbasedchecking
5. Hewitt, Carl, Peter Bishop, and Richard Steiger. "Session 8 formalisms for artificial intelligence a universal modular actor formalism for artificial intelligence." Advance Papers of the Conference. Vol. 3. Stanford Research Institute, 1973.
6. Clarke, Edmund, Alexandre Donz, and Axel Legay. "On simulation-based probabilistic model checking of mixed-analog circuits." Formal Methods in System Design 36.2 (2010): 97-113.
7. Platzer, André, "Differential dynamic logic for hybrid systems", Journal of Automated Reasoning, vol. 41, no. 2, pp. 143189, 2008

8. Courcoubetis, Costas, and Mihalis Yannakakis. "The complexity of probabilistic verification." Journal of the ACM (JACM) 42.4 (1995): 857-907.
9. J. Eliosoff. Calculating the probability of an LTL formula over a labeled Markov chain. McGill University, Montreal, 2003
10. Rodrigues, Pedro, Emil Lupu, and Jeff Kramer. "LTSA-PCA: Tool support for compositional reliability analysis." Companion Proceedings of the 36th International Conference on Software Engineering. ACM, 2014.
11. Lupu, E. C., P. Rodrigues, and Jeff Kramer. "Compositional reliability analysis for probabilistic component automata."
12. M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in CAV11, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, pp. 585591.
13. Sen, Koushik, Mahesh Viswanathan, and Gul Agha. "Statistical model checking of black-box probabilistic systems." International Conference on Computer Aided Verification. Springer, Berlin, Heidelberg, 2004.
14. David, Alexandre, et al. "Uppaal SMC tutorial." International Journal on Software Tools for Technology Transfer 17.4 (2015): 397-415.
15. G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.
16. Pfeffer, Avi. Practical probabilistic programming. Manning Publications Co., 2016.
17. Baier, Christel, Frank Ciesinski, and M. Grosser. "PROBMELA: a modeling language for communicating probabilistic processes." Proceedings. Second ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2004. MEMOCODE'04.. IEEE, 2004.
18. Shishkin, Evgeniy. "Construction and formal verification of a fault-tolerant distributed mutual exclusion algorithm." Proceedings of the 16th ACM SIGPLAN International Workshop on Erlang. ACM, 2017.
19. Dolev, Danny, Maria Klawe, and Michael Rodeh. "An O (n log n) unidirectional distributed algorithm for extrema finding in a circle." Journal of Algorithms 3.3 (1982): 245-260.
20. Hahn, Ernst Moritz, et al. "PARAM: A model checker for parametric Markov models." International Conference on Computer Aided Verification. Springer, Berlin, Heidelberg, 2010.