

# Comprehensive Exam Responses

Landon Taylor – A02236752

16 January 2025

## Contents

<b>1. Questions for <i>Stochastic Model Checking</i> [1]</b>	<b>2</b>
(a) About Sections 3 and 4 . . . . .	2
(i) Purpose of infinite path and cylinder set . . . . .	2
(ii) Syntax comparison . . . . .	2
(iii) Modified example . . . . .	2
(iv) Reason for computing BSCC . . . . .	3
(v) Rewards . . . . .	3
(b) About PRISM modeling language . . . . .	4
(i) Multiple guarded commands . . . . .	4
(ii) Always block . . . . .	4
(iii) Synchronize commands in Verilog . . . . .	4
<b>2. Questions for <i>Model-checking Algorithms for CTMCs</i> [2]</b>	<b>5</b>
(a) Modified Proposition 1 . . . . .	5
(b) Formula equivalence . . . . .	5
(c) Next/until equivalence . . . . .	5
(d) Sufficiency in Theorem 1 . . . . .	5
(e) CSL checking pseudocode . . . . .	6
<b>3. Questions for <i>PrIC3</i> [3]</b>	<b>7</b>
(a) Intuitive description of PDR/IC3 . . . . .	7
(b) Paper review & challenges . . . . .	8
(c) Feasibility & limitations for CTMCs . . . . .	10
<b>4. Questions for <i>Parameter Synthesis</i> [4]</b>	<b>10</b>
(a) Terminology . . . . .	10
(b) Section 5 summary & review . . . . .	10
(c) Adaptation to CTMCs . . . . .	12
(d) Usefulness of parametric CTMC . . . . .	12
(e) PrIC3 and parameter synthesis . . . . .	12

## 1. Questions for *Stochastic Model Checking* [1]

### (a) About Sections 3 and 4

#### (i) Purpose of infinite path and cylinder set

**Infinite Path.** Defining infinite paths allows the examination of a Markov chain for an infinite number of steps or an infinite amount of time. Because paths are not required to be bounded by number of steps in a model, an effective analysis requires infinite paths to be accounted for.

**Cylinder Set.** The cylinder set allows the exploration of all infinite-length paths with a finite prefix. More importantly, it allows properties of the semi-ring topology to apply to models.

**Comparison.** In a DTMC, an infinite path and cylinder set are used to apply Theorem 1 to DTMCs. That enables the discovery of the probability measure of each path. In a CTMC, this is supplemented by an amount of time spent in a state. This means that rather than having a prefix sharing only states, there is a shared prefix by length of time.

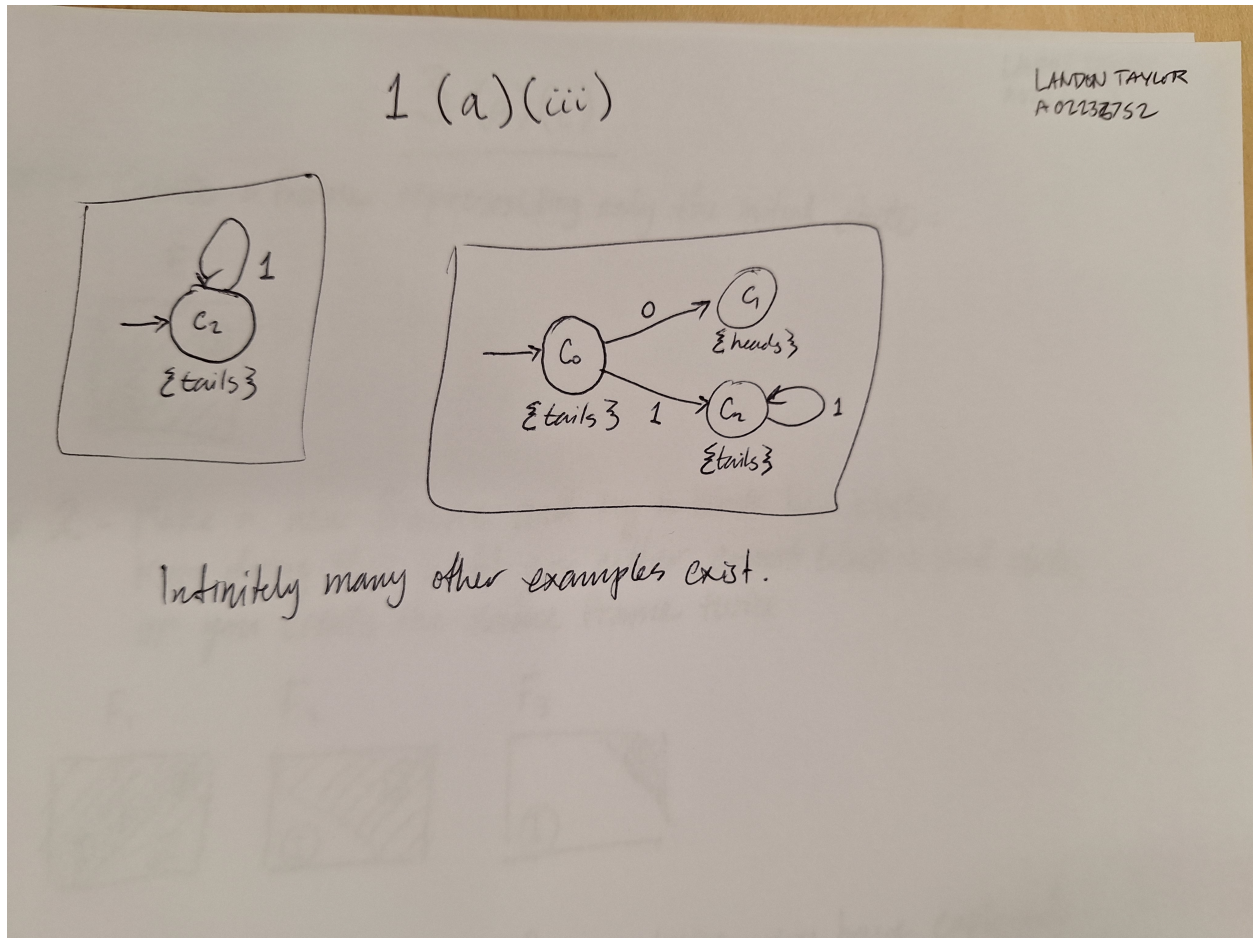
#### (ii) Syntax comparison

The syntax of CSL is very similar to PCTL. The major differences are as follows:

- CSL has a steady-state operator, which is a major difference. Adding an equivalent steady-state operator to PCTL for DTMC analysis would potentially duplicate the unbounded until operator, as it is possible to specify behavior in the long-run with the condition “property holds until ‘false’”
- CSL is based on time intervals, while PCTL is based on number of steps. They share very similar operations; however, in CSL a property might read “the probability of failure after 10 seconds is under 0.5”, where in PCTL it would read “the probability of failure after 10 *steps* is under 0.5”. Specifically, PCTL has a step-bounded until operation, where CSL has a time-bounded until.

#### (iii) Modified example

It is possible to modify the example in such a way that the probability of “eventually tails” is at least one, or tails will always eventually hold. A major modification is to remove all states except “tails”, make “tails” the initial state, then add a self-loop with a probability of 1. Because I imagine this is not the intent of the question, other variations are shown on the scanned page labeled 1(a)(iii).



#### (iv) Reason for computing BSCC

For steady-state analysis, we are effectively searching for “sinks” in the model. Using the metaphor of probability flowing through the state space like water, we are looking for the places where puddles will form. A BSCC represents a portion of the state graph that, once entered, cannot be exited. In the metaphor, the water gets stuck in the BSCC (puddle), and it cannot flow to any other areas. At that point, we only care about where *within* the BSCC the probability mass settles. In other words, if the model enters a state or region that it is likely to leave in a short amount of time, it does not make sense to account for that behavior in the long run. In contrast, if a model is likely to be stuck in a state, that is impactful to the long run analysis.

#### (v) Rewards

A reward is extremely versatile, but it effectively allows the model to accrue “points” for being in certain states or taking certain transitions. In a DTMC, rewards are given based on (1) number of time-steps spent in a state, and (2) which transitions are taken. In a CTMC, it is similar, based on (1) *amount of time* spent in a state, and (2) which transitions are taken.

In practice, it can be particularly helpful in determining “side effects” of actions. For example, a network analysis may check that the probability of failure is low, which can be done with normal CSL or PCTL checking. The user might add rewards to certain transitions to model packets being

transmitted, which allows not only the probability check, but the answer to “how many packets can we expect to fire before the network fails?”

## (b) About PRISM modeling language

### (i) Multiple guarded commands

In Prism, the guard does not synchronize commands or require sequential execution. Synchronization is achieved with an action label (see Prism Manual, but sequential operation is determined by the model’s specification rather than Prism’s semantics.

In short, Prism does not execute commands sequentially. If two guards are simultaneously enabled in a DTMC, it is a probabilistic choice, and in a CTMC, it is a race condition.

### (ii) Always block

Blocking assignments can be achieved in Prism, but to my knowledge, it requires a bit of extra information in the model (rather than in Prism’s syntax). A step counter, if the model is relatively simple, suffices. Essentially, if a Verilog program can be implemented as a probabilistic model, the Prism language can model it. A simple example follows, with every action being completely deterministic (probability 1) in the (rather uninteresting) Prism DTMC.

Verilog Syntax (approximate):

```
...
always @(*) begin
    a = 3;
    b = 4;
end
...
```

Prism Syntax (approximate):

```
...
step : [0..1] init 0;
[] step = 0 -> 1.0 : a' = 3 + 1.0 : step' = 1
[] step = 1 -> 1.0 : b' = 4 + 1.0 : step' = 0
...
```

### (iii) Synchronize commands in Verilog

It is possible to achieve similar behavior in Verilog, potentially using an Always block. For example, many commands in Verilog are commonly synchronized using a shared clock variable. A simple example follows:

```
module m1
...
always @(clk) begin
    a <= 3;
end
endmodule
```

```

module m2
  ...
  always @(clk) begin
    b <= 3;
  end
endmodule

```

In the example, `clk` is used to synchronize the setting of `a` and `b`, allowing for small differences in the amount of time a physical hardware implementation would take to achieve each task. For the sake of the abstract Verilog model, `a` and `b` are set at the same time.

## 2. Questions for *Model-checking Algorithms for CTMCs* [2]

### (a) Modified Proposition 1

It appears that the transition delay does not need to be 1, and it can be capped at an arbitrary  $k$ . This is because the proof relies on the fact that multiplying probabilities always yields a probability, or  $\mathbb{R} \in [0, 1]$ . It is important to the proof, however, that the sum of all  $t$  values converges, so  $k$  would need to allow for convergence.

### (b) Formula equivalence

Yes, according to the semantics description in Section 3.1, the CTL and CSL formulas are equivalent. One indicates that in every path in  $\mathcal{M}$ ,  $\varphi$  holds, and the other indicates that in every state in  $\mathcal{M}$ ,  $\varphi$  holds.

### (c) Next/until equivalence

No, the statements are not equivalent. The time-bounded next state query checks that  $\Phi$  holds *in the next state* within 1 time unit. The time-bounded until query checks that  $\Phi$  holds within 1 time unit, but it does not care if that happens in the next state. For example, the former allows only one transition to  $s \models \Phi$ , where the latter allows an arbitrary number of transitions (with delays summing to less than one time unit), and only requires  $\Phi$  to hold before the end of the interval  $[0, 1]$ .

### (d) Sufficiency in Theorem 1

Assuming the remainder of the theorem remains unchanged, requiring only  $s \models \Phi$  creates a conflict between the second and third case. I assume this conflict would not exist for the remainder of my response.

The second case, where  $\Phi \wedge \neg\Psi$  holds represents the chance of going to a satisfying state within the time bound. In contrast, the third case requires extra information: if  $a > 0$  and  $\Phi \wedge \Psi$  holds, we are interested in the probability of going to a satisfying state, but we need to add the sum of the probability to stay in the next (satisfying) state as well. Thus, this case also considers the chance of exiting a satisfying state before reaching the lower time bound.

### (e) CSL checking pseudocode

At a very high level, this algorithm splits the time-bounded until CSL problem into two (plus a bonus case for  $I = [t, \infty)$ ) cases. In the case where  $I = [0, t]$ , we simply find the sum over all uniformized time steps of the probability of being in a post-until state at or before time  $t$ . In the case where  $I = [t, t']$ , we split the problem into two parts: the probability of staying in a pre-until state until time  $t$ , and the probability of moving to a post-until state within  $t' - t$  time units.

The following pseudocode (presented primarily in plain language due to differences in notation between papers) represents an algorithm for checking a CSL time-bounded until property in an arbitrary CTMC. It is based on a combination of methods presented in [2] and [1]. This response assumes the model and property are already parsed and prepared to analyze.

Input: CTMC  $\mathcal{M}$

- Initialize transition rate matrix and state probability vector
- Given interval  $I$ , if  $I = [0, t]$ :
  - Determine the least solution of the set of integral equations on Page 25 of [1], which [2] proves is equivalent to transient analysis:

$$Prob(s, \Phi \mathcal{U}^{[0,t]} \Psi) = \sum_{s'' \models \Psi} \pi^{\mathcal{M}[\neg\Phi \wedge \neg\Psi]}(s, s'', t)$$

To get this value, perform transient analysis as described below, implementing the formula on the bottom of page 25 of Stochastic Model Checking:

- For each uniformized time step  $i$ , accumulate the sum of the following matrices:
  - \* Find the uniformized matrix such that satisfying states are treated as absorbing
  - \* Multiply that matrix by itself  $i$  times, where  $i$  represents the number of time-steps in the uniformized CTMC
  - \* Multiply the result by both the  $i$ th Poisson probability, or the probability that  $i$  steps occur within time  $t$ , and  $\underline{\Psi}$ , or the vector with a 1 representing an absorbing state and 0 representing all other states.
- Given interval  $I$ , if  $I = [t, t']$ :
  - Two values need to be obtained: the probability of staying in a  $\Phi$  state until time  $t$ , then the probability of reaching a  $\Psi$  state without first leaving a  $\Phi$  state with in time  $t' - t$ .
  - To get the former, perform transient analysis as described above. To get the latter, treat  $\neg\Phi$  as an absorbing state and perform transient analysis as described above. Specifically:
    - For each uniformized time step  $i$ , accumulate the sum of the following matrices into  $M$ :
      - \* Find the uniformized matrix such that satisfying states are treated as absorbing
      - \* Multiply that matrix by itself  $i$  times, where  $i$  represents the number of time-steps in the uniformized CTMC
      - \* Multiply the result by the  $i$ th Poisson probability, or the probability that  $i$  steps occur within time  $t' - t$ , then multiply that result by  $\underline{\Psi}$ , or the vector with a 1 representing an absorbing state and 0 representing all other states.
    - For each time step  $i$ :
      - \* Find the uniformized matrix such that states satisfying  $\neg\Phi$  are treated as absorbing
      - \* Multiply that matrix by itself  $i$  times, where  $i$  represents the number of time-steps in the uniformized CTMC
      - \* Multiply the result by the  $i$ th Poisson probability, or the probability that  $i$  steps occur within time  $t'$ , then multiply that result by  $M$ , which we calculated earlier.

- Given interval  $I$ , if  $I = [t, \infty]$ :
  - Compute as if  $I = [t, t']$ , removing bounds on the Until operator.

### 3. Questions for *PrIC3* [3]

#### (a) Intuitive description of PDR/IC3

*Property-Directed Reachability* (PDR or IC3) is an extremely powerful tool that enables checking properties of systems with very many states over an unknown number of steps. Intuitively, PDR explores regions of states that can be reached from an initial state, one step at a time. To check if a bad state is reachable, it keeps exploring until it cannot explore any more regions of the state space, or until it finds an example of failure.

Where many model checking and verification tools use a state-based approach to analyzing a model, this becomes impractical when models have a very large set of states. PDR's major revolution is the fact that it groups states into regions (i.e., rather than representing infinitely many states, it represents the set of states where  $x > k$ ).

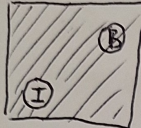
An illustrated example is provided on the scanned paper labeled 3(a)(i). This very simple example shows PDR creating an initial frame with the bad states blocked, then iteratively adding new frames and attempting to block the bad state. If after F3, the bad state cannot be blocked, the user will get a counterexample path, or a path from I to a bad state. If the next frame is constructed and matches F3 exactly, then it is an inductive proof of correctness: since we cannot take a single step to leave the good states in F3 to F4, then we cannot take any single step from any good state to any bad state.

LANDON TAYLOR  
A02236752

3 (a)(i)

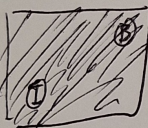
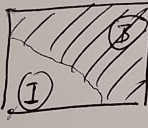
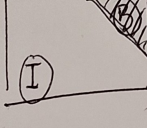
Step 1 - Create a frame representing only the initial state:

$F_1$



Step 2 - Make a new frame and try to block bad states.  
Keep doing this until you either cannot block a bad state,  
or you create the same frame twice:

$F_1$        $F_2$        $F_3$

If you create the same frame twice, you have explored everything you can, and the model is OK.  
If you can't block a bad state, find a path to the bad state and report it to the user.

### (b) Paper review & challenges

**Summary.** The paper's primary contribution is the extension of PDR into the probabilistic domain. This involves overcoming four major challenges, described below. The paper presents three nested algorithms: the main PrIC3 loop, the Strengthening procedure, and the overall counterexample-generation and oracle refinement loop. The paper is successful at meeting several of the challenges, and room is left for significant future development, especially of the heuristic guiding the strengthening procedure.

**Technical Contributions.** The primary contribution of PrIC3 is the adaptation from PDR (which is built for deterministic systems) into the probabilistic domain. This comes with several challenges, which the paper addresses throughout. Discussion of these challenges follows. PrIC3 appears to be a powerful algorithm for models that are well-suited for it. The paper mentions particularly promising results for models that are relatively simple but have prohibitively large state spaces, as PrIC3 appears to outperform model checking tools on this task.



**Challenges.** The following is an analysis of how the paper addressed its four presented challenges.

- *Challenge 1, Leaving the Boolean Domain.* Moving from PDR’s qualitative (Boolean) analysis into the quantitative (probabilistic) space is the major challenge this paper addresses. Frames in traditional PDR are (potentially bounded) sets of states. In PrIC3, a property is written as “the maximal probability of reaching a bad state is at most  $\lambda$ ”, as opposed to a property in PDR, which is written “a bad state is not reachable”. Recovery statements throughout the paper demonstrate equivalence to PDR in the case when the maximum allowed probability  $\lambda$  is zero. Because of this shift to probabilistic analysis, models effectively become infinitely large due to the addition of real-valued probabilities. Thus, frames are treated as real-valued functions that evaluate a  $k$ -step reachability probability for a given state. The paper’s contribution is strongest for this challenge, but this task creates additional challenges (2-4).
- *Challenge 2, Counterexamples  $\neq$  single paths.* In traditional PDR, a counterexample is simply a path demonstrating reachability from the initial state to a target/bad state. In a probabilistic setting, a single path tends to be an insufficient counterexample to show the negation of the kind of properties PrIC3 is interested in. Thus, PrIC3 returns a sub-MDP containing the portion of the model that caused the calculated probability to exceed the bound  $\lambda$ . The authors address this challenge throughout the paper, emphasizing both the benefits and shortcomings of the presented counterexample generation strategy. While the counterexample sub-MDP is usually smaller and thus easier to pass to a probabilistic model checker for verification, it is not always guaranteed to be a true counterexample. That is, PrIC3 sometimes returns a sub-MDP that is not demonstrative of the reachability probability exceeding  $\lambda$ .
- *Challenge 3, Strengthening.* While traditional PDR strengthens frames by pruning regions of states from them, PrIC3 strengthens frames by decreasing the maximal probability of states within frames. This step is the reason a good heuristic is so important; it is desirable to decrease probabilities to be as close to their true values as possible. However, in order to effectively do that, a full model analysis would be required and PrIC3 would become moot. The paper leaves much of this challenge to the reader, though the artifact at <https://github.com/moves-rwth/pric3>, specifically in the folder at <https://github.com/moves-rwth/PrIC3/blob/master/pric3/oracles> includes some examples of useful oracles (which are used to generate the heuristics). The authors address this challenge thoroughly, but it is clear that because of the nature of heuristics, this will likely remain a design challenge for PrIC3 development.
- *Challenge 4, Generalization.* PDR’s scalability is greatly impacted by its ability to generalize states. While this is not necessary for correctness, it is a major scalability challenge. PrIC3 has to not only generalize a *reachable* set of states, but it has to generalize *probabilities* for those states. This presents a scalability challenge due to the complexity of calculating probabilities, and the authors rely primarily on interpolation to attack it. While several methods of interpolation are presented, the paper appears to lack conclusive evidence for any particular method.

**Shortcomings in Presented Techniques.** The paper is clear and explicit about the shortcomings of the algorithms. The primary weakness seems to be consistent throughout the paper; without an adequate heuristic for invariant strengthening, the tool can sometimes fall flat. A related shortcoming is the treatment of counterexamples. Specifically, a counterexample returned by PrIC3 may not truly be a counterexample (or, a good model can produce a false positive for failure). The paper addresses this weakness by implementing a series of oracle refinements during verification, but this seems to trade runtime for accuracy and correctness. The presented algorithms, while they make a significant advance in PDR by moving it into the probabilistic domain, still requires an

additional probabilistic model checking tool to check counterexamples.

### (c) Feasibility & limitations for CTMCs

As Katoen commented shortly after the release of PrIC3, adapting PrIC3 to the continuous-time domain poses formidable challenges. Even limiting the discussion to vector addition systems, continuous-time systems require explicit state space enumeration for complete analysis, so a CTMC adaptation of PDR would likely only be able to estimate probabilities.

One potential direction for exploration is discovering probability bounds on regions of a state space using the embedded DTMC (potentially leveraging desirable properties from strongly-connected components and other model features), then refining the analysis for only regions that seem to increase the observed probability of reaching a target. Frames' time steps could potentially be represented by a single uniformized time step in the CTMC.

Separating the probabilistic and the deterministic sides of models has shown promise: in Ragtimer and Wayfarer, for example, paths to a target are collected in a probability-agnostic setting, then probabilistic model checking is used to find a probability lower bound. PDR is potentially useful for discovering these paths, but more efficient methods for VAS path discovery seem to already exist.

## 4. Questions for *Parameter Synthesis* [4]

### (a) Terminology

**Feasibility Problem** is simply the question “is there any parameter instantiation satisfying some property  $\varphi$ ?” It is the fundamental question for parameter synthesis.

**Exact Partitioning Problem** attempts to break a region into two complimentary sub-regions, one that contains only members satisfying some  $\varphi$ , one satisfying  $\neg\varphi$ .

**Approximate Partitioning Problem** has the same goal as the exact partitioning problem, except it allows for a margin of error. This allowance is added primarily for efficiency. In the case that no error is allowed, this is equivalent to the exact partitioning problem.

**Parameter Synthesis**, broadly speaking, is the art of finding parameters or specifications for a model that allow the model to satisfy a property. In traditional verification, we simply ask whether or not a model is safe. Parameter synthesis allows us to ask “What do I have to do to the model to make it safe?”

**CEGAR** is an abstraction-refinement approach that creates an abstract model, then iteratively uses counterexamples to refine the abstraction (i.e., make it as close as possible to the original model). An ideal CEGAR algorithm produces an abstraction that saves on computation and memory but accurately reflects all of the original model's properties of interest.

### (b) Section 5 summary & review

**Summary.** Section 5 provides algorithms for parameter synthesis problems. It describes exact partitioning, region verification, and approximate partitioning, as well as feasibility checking methods, describing each in detail.

Exact partitioning is essentially the process of exactly solving a linear system of equations whose coefficients are rational functions. This poses a significant challenge when systems become very large, as the computation is exponential in the number of parameters. Cutting-edge tools use state elimination on the Markov model before doing this calculation to save computational effort. Experiments show this method offers perfect representation of the model in exchange for significant runtime.

Region verification partitions a model into regions then exploits SMT solving to check if the parameters produce a satisfiable formula. This is exact and complete, but it does not scale. By relaxing parameter dependencies and considering only worst cases, this method becomes more viable. Region abstraction further benefits from refinement.

Approximate Parameter Space Partitioning is very similar to exact partitioning, but it allows for unknown regions. As shown in Figure 10, the method is similar to CEGAR in that it creates regions as an abstraction, then iteratively refines the result space. The challenges arising here include implementing effective refinement strategies, and this becomes similar to a heuristic problem. This appears to be the most scalable method presented.

Feasibility Checking benefits from mathematical optimization, as it is an ETR-complete problem. Essentially, feasibility is treated as an optimization problem, enabling the ETR-formula to be represented as a problem with a quadratic objective, which can then be turned into LP problems. Further, integrating PMC techniques into the approach once the initial point is feasible allows more efficiency.

**Review.** The following lists advantages and shortcomings of each section:

Exact partitioning has the following advantages:

- Medium-sized functions can be computed relatively efficiently
- The method is exact, so an exact solution is known
- State elimination is effective and gives a good efficiency boost

Exact partitioning has the following shortcomings:

- Solution functions can be too large
- Optimally ordering elimination is NP-hard, requiring heuristics and advanced knowledge
- Models often need to be decomposed to make checking possible

Region verification has the following advantages:

- SMT solving is relatively easy to implement
- Some models can benefit from a transformation into an MDP
- The result is exact and complete, unlike some other methods

Region verification has the following major shortcoming:

- The method is very expensive and only works on very small models

Approximate Parameter Space Partitioning has the following advantages:

- The algorithms are more scalable for very large models because of abstraction
- The result is probably “good enough” for many models

Approximate Parameter Space Partitioning has the following shortcomings:

- The algorithms are more complicated to implement and can introduce errors more easily
- Finding region candidates requires sampling and is not exact

### (c) Adaptation to CTMCs

A parametric MC for CTMCs is relatively straightforward to translate, but algorithms would be significantly more complex, as is usually the case with CTMCs. The following definition is my adaptation:

**Definition.** A CTpMC  $\mathcal{C}$  is a tuple  $\langle S, s_I, X, R \rangle$ , with a finite set  $S$  of states, an initial state  $s_I$ , a finite set  $X$  of real-valued variables (parameters), and a transition rate function  $R : S \times S \rightarrow \mathbb{R}[X]$ . The parametric transition rate is given by  $R(s, t)$ .

### (d) Usefulness of parametric CTMC

I believe a CTpMC has potential to be useful. There are many cases where the desired outcome (property) is known, but the challenge lies in designing the system to satisfy that property. For example, if a synthetic biologist desires to produce a certain amount of a product within a certain timeframe, the model could be encoded as a CTpMC in which the parameters are impacted by changes made to the rate of each reaction. In a simpler example, if a manufacturing company is trying to determine how many employees to hire, they may use parameters to manipulate the number of units each employee produces per day based on factors like experience, fatigue, and other job responsibilities.

### (e) PrIC3 and parameter synthesis

The techniques in PrIC3 have the potential to help with parameter synthesis in two primary ways: (1) as an alternative to PMC or SMT solving, and (2) to iteratively generate and refine bounds on variables using the existing PrIC3 heuristics.

As an alternative to PMC, PrIC3 may (for models that are well-suited for it) be a faster way to check that a change to a parameter satisfies the property. As a tightly-integrated method, it may be possible for PrIC3 to quickly find and refine  $k$ -step bounds on probabilities to help guide parameter synthesis.

Additionally, PrIC3 and even traditional PDR may be particularly helpful for generating abstracted regions. Since one of PDR's strong suits is finding reachable regions, extended with probabilities in PrIC3, it may be able to boost the heuristics implemented in existing parameter synthesis tools.

## References

- [1] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic Model Checking," in *Formal Methods for Performance Evaluation*, M. Bernardo and J. Hillston, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, vol. 4486, pp. 220–270, series Title: Lecture Notes in Computer Science. [Online]. Available: [http://link.springer.com/10.1007/978-3-540-72522-0\\_6](http://link.springer.com/10.1007/978-3-540-72522-0_6)
- [2] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen, "Model-checking algorithms for continuous-time Markov chains," *IEEE Transactions on Software Engineering*, vol. 29, no. 6, pp. 524–541, Jun. 2003, conference Name: IEEE Transactions on Software Engineering. [Online]. Available: <https://ieeexplore.ieee.org/document/1205180>
- [3] K. Batz, S. Junges, B. L. Kaminski, J.-P. Katoen, C. Matheja, and P. Schröder, "PrIC3: Property Directed Reachability for MDPs," in *Computer Aided Verification*, S. K. Lahiri and C. Wang, Eds. Cham: Springer International Publishing, 2020, pp. 512–538.
- [4] N. Jansen, S. Junges, and J.-P. Katoen, "Parameter Synthesis in Markov Models: A Gentle Survey," in *Principles of Systems Design: Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*, J.-F. Raskin, K. Chatterjee, L. Doyen, and R. Majumdar, Eds. Cham: Springer Nature Switzerland, 2022, pp. 407–437. [Online]. Available: [https://doi.org/10.1007/978-3-031-22337-2\\_20](https://doi.org/10.1007/978-3-031-22337-2_20)