PRISM for Discrete Time Markov Chains

Andreas Plank

January 30, 2018

Contents

Recapitulation of Markov Chain Theory

We can describe a discrete Markov chain (DTMC) as a Tuple D $=$ (S,s_0,**P**,AP,L) where:

- S is a set of states;
- s_0 is the initial state;
- **P** is the transition probability matrix
- AP is a set of atomic propositions
- L is a labelling function

Properties of Markov Chains

Important and often focussed properties are transient and steady-state behaviours for DTMCs.

As an example we can compute the stationary distribution of an ergodic (irreducible, non periodic) Markov chain by solving the linear System

$$\pi = \pi A$$

with the normalization property

$$\sum_{i \in E} \pi_i = 1$$

Probabilistic Computation Tree Logic

Probabilistic model checking tools like PRISM extend our knowledge to properties over paths. This is done via the PCTL (Probabilistic Computation Tree Logic) that extends the temporal logic CTL by probabilistic concepts.

Definition

$\Phi ::= true \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid P_{\ p}[\phi]$
$\phi ::= X\,\Phi \mid \Phi U^{\leq k}\Phi \mid \Phi\ U\ \Phi$

where a is an atomic proposition, $\ \in <, \leq, \geq, >$, $p \in [0,1]$ *and* $k \in \mathbb{N}$

Probabilistic Computation Tree Logic

With s$\models$ Φ we denote that a property Φ is true/satisfied for state s.

The operator $P\ _p[\phi]$ indicates if a path formula ϕ is true in a state satisfying the given bound p. This can be formally described as

Definition

$$s \models P_{\sim p}[\phi] \iff Prob(s, \phi) \sim p$$

where $Prob(s,\phi) = Pr_s\ \{\omega \in Path(s) \mid \omega \models \phi\}$

Example

$$P_{>0.60}[\neg fail \text{ U } success]$$

means "is the probability that a task does not fail before it succeeds >60?".

PCTL model checking

Definition

Sat(ϕ) $= \{s \in S \mid s \models \Phi\ \}$ = *set of states satisfying* Φ

For the non-probabilistic operators of our PCTL model we have

- $\mathrm{Sat}(\mathrm{true}) = \mathrm{S}$
- $\mathrm{Sat}(\mathrm{a}) = \{\mathrm{s} \in \mathrm{S} \mid \mathrm{a} \in \mathrm{L}(\mathrm{s})\}$
- $\mathrm{Sat}(\neg\ \Phi) = \mathrm{S} \setminus \mathrm{Sat}(\Phi)$
- $\mathrm{Sat}(\Phi_1 \vee \Phi_2) = \mathrm{Sat}(\Phi_1) \cup \mathrm{Sat}(\Phi_2)$

PCTL model checking

For $P_{\ p}[\phi]$ (e.g. with next operator X)we need to compute probabilities for all states s $\in$ S.

$$Sat(P_{\ p}[\phi]) = \{s \in S \mid Prob(s, X\Phi) \sim p\}$$

$Prob(s, X\Phi)$ can be computed by

$$Prob(s, X\Phi) = \sum_{s' \in Sat(\Phi)} P(s, s')$$

We can also compute the vector **Prob**(XΦ) which contains the probabilities for all states s

$$\mathbf{Prob}(\mathrm{X}\Phi) = \mathbf{P} \cdot \Phi$$

with Φ a vector with entries $\in \{0,1\}$ over S with $\Phi(s) = 1 \iff s \models \Phi$

Example for PCTL next

Example

Given is a property that returns true if the probability that we do not land in try or success doing one step is ≥ 90:

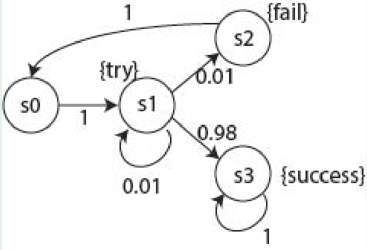
$$P_{\geq 0.9}[X(\neg try \vee success)]$$

for a Markov Chain with transition matrix

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Example for PCTL next

Example



In order to compute the vector **Prob**(XΦ) we first need to compute

$$\begin{aligned} Sat(\neg try \vee success) &= (S \setminus Sat(try)) \cup Sat(success) \\ &= (\{s_0, s_1, s_2, s_3\} \setminus \{s_1\}) \cup \{s_3\} = \{s_0, s_2, s_3\} \end{aligned}$$

Example for PCTL next

Example

Now

$$\mathbf{Prob}(\mathrm{X}(\neg try \vee success)) = \mathbf{P} \cdot (\neg\ \mathbf{try} \vee \mathbf{succ}) =$$

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.99 \\ 1 \\ 1 \end{pmatrix}$$

which leads to

$$\mathbf{Prob}(X(\neg try \vee success)) = [0, 0.99, 1, 1]$$

and

$$Sat(P_{\geq 0.9}[X(\neg try \vee success)]) = \{s_1, s_2, s_3\}$$

PCTL for the bounded until operator

When we use the bounded until operator we need to do more work to compute the probabilities:

$$Sat(P_{\sim p}[\phi_1 U^{\leq k} \phi_2]) = \{s \in S \mid Prob(s, \phi_1 U^{\leq k} \phi_2) \sim p\}$$

First we identify the trivial states

- $S^{yes} = \mathrm{Sat}(\phi_2)$
- $S^{no} = \mathrm{S} \setminus (\mathrm{Sat}(\phi_1) \cup \mathrm{Sat}(\phi_2))$

For the expression $Prob(s, \phi_1 U^{\leq k} \phi_2)$ we get

$$Prob(s, \phi_1 U^{\leq k} \phi_2) = \begin{cases} 1 & \text{if s} \in S^{yes} \\ 0 & \text{if s} \in S^{no} \\ 0 & \text{if s} \in S^{?}, k = 0 \\ \sum_{s' \in S} P(s, s') \cdot Prob(s, \phi_1 U^{\leq k-1} \phi_2) & \text{if s} \in S^{?}, k > 0 \end{cases}$$

PCTL for the bounded until operator

The vector **Prob**$(\phi_1 \ U^{\leq k} \ \phi_2)$ can be computed simultaneous via computing the probabilities for all states s $\in$ S, or by using an iterative method:

- **Prob**$(\phi_1 \ U^{\leq 0} \ \phi_2) = \boldsymbol{\phi_2}$
- **Prob**$(\boldsymbol{\phi_1} \ \boldsymbol{U^{\leq k}} \ \boldsymbol{\phi_2}) =$ **P'** $\cdot$ **Prob**$(\boldsymbol{\phi_1} \ \boldsymbol{U^{\leq k-1}} \ \boldsymbol{\phi_2})$

where

$$\mathbf{P'}(s,s') = \begin{cases} \mathbf{P}(s,s') & \text{if s} \in S^{?} \\ 1 & \text{if s} \in S^{yes} \\ 0 & \text{if s} \in S^{no} \end{cases}$$

PCTL for the until operator

Again we identify the trivial states

- $S^{yes} = Sat(P_{\geq 1}[\phi_1 \text{ U } \phi_2])$
- $S^{no} = Sat(P_{\leq 0}[\phi_1 \text{ U } \phi_2])$

This two sets are computed with two extra algorithms giving the following advantages:

- gives exact results for the states in these sets
- no further computations are needed for these states
- reduces the number of states that need to be computed via a numeric solver

PCTL for the until operator

Algorithm0

Computation of $S^{no} = Sat(P_{\leq 0}[\phi_1 \text{ U } \phi_2])$

- compute Sat($P_{>0}[\phi_1 \text{ U } \phi_2]$).
  This means we want to find states that reach a state satisfying ϕ_2 through states satisfying ϕ_1 with positive probability
- this can be done by graph-based algorithms
- the result is subtracted from S

PCTL for the until operator

Algorithm1

Computation of $S^{yes} = Sat(P_{\geq 1}[\phi_1 \text{ U } \phi_2])$

- compute Sat($P_{<1}[\phi_1 \text{ U } \phi_2]$), using S^{no}
  This means we want to find states that reach a state in S^{no} through states satisfying ϕ_1 with positive probability
- again, this can be done by graph-based algorithms
- the result is subtracted from S

PCTL for the until operator

From this we can compute the probabilities $\text{Prob}(s, \phi_1 \text{U} \phi_2)$ as the solution of a system of linear equations:

$$Prob(s, \phi_1 U \phi_2) = \begin{cases} 1 & \text{if s} \in S^{yes} \\ 0 & \text{if s} \in S^{no} \\ \sum\limits_{s' \in S} P(s, s') \cdot Prob(s, \phi_1 U \phi_2) & \text{if s} \in S^{?} \end{cases}$$

PRISM solves this by applying one of several available iterative method

PCTL for the until operator

Example

Given is a property that returns true if we stay in state "try" until we reach state "success" with probability bigger than 0.90

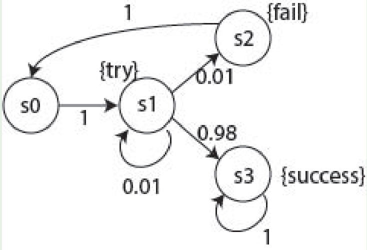
$$P_{>0.9}[X(try \text{ U } success)]$$

for the Markov Chain with transition matrix

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

PCTL for the until operator

Example



- Sat(try)=$\{s_1\}$, and Sat(success)=$\{s_3\}$
- $S^{no} = Sat(P_{\leq 0}[try \text{ U } success]) = \{s_0, s_2\}$
- $S^{yes} = Sat(P_{\geq 1}[try \text{ U } success]) = \{s_3\}$
- $S^{?} = \{s_1\}$

PCTL for the until operator

Example

This leads to the following linear system

- $x_0 = 0$
- $x_1 = 0.01 \cdot x_1 + 0.01 \cdot x_2 + 0.98 \cdot x_3$
- $x_2 = 0$
- $x_3 = 1$

This yields
- **Prob**(try U success) $=$ [0,98/99,0,1]
- $Sat(P_{>0.90}[try \text{ U } success]) = \{s_1, s_3\}$

Rewards for DTMCs

Definition

For a given DTMC (S,s_0,$\mathbf{P}$,L) *we define a reward structure as a pair*(ρ, ι) *where*

- $\rho : S \rightarrow \mathbb{R}_{\geq 0}$ *is the state reward function*
- $\iota : S \times S \rightarrow \mathbb{R}_{\geq 0}$ *is the transition reward function*

Example

- steps: ρ is 1 for all states; ι is 0 for all transitions
- power consumption: ρ is the power consumption per time unit in each state; ι is the power cost of each transition

PCTL for Rewards

We now extend the PCTL with the operator R, which works similar to the operator P

$$\Phi ::= \ldots \mid R_{\sim p}[I^{=k}] \mid R_{\sim r}[C^{\leq k}] \mid R_{\sim r}[F\Phi]$$

where

- r $\in \mathbb{R}_{\geq 0}, \sim \in \{<, \leq, \geq, >\}, k \in \mathbb{N}$ and
- $R_{\sim r}[\cdot]$ describes the mean value of $\cdot$ satisfying $\sim$ r

Types of Rewards

Instantaneous: $R_{\sim r}[I^{=k}]$

- indicates if the expected state reward at step/time k is $\sim$ r
- example: expected occupied servers in the system after 30 steps
- computation: can be reduced to bounded until probabilities

Cumulative: $R_{\sim r}[C^{\leq k}]$

- indicates if the expected reward up to step/time k is $\sim$ r
- example: total power consumption up to next month
- computation: can be computed similar to bounded until probabilities

Reachability: $R_{\sim r}[F\Phi]$

- indicates if the expected cumulated reward until a state satisfies Φ is $\sim$ r
- example: total power consumption until the system fails - computation: can be computed similar to until probabilities.

Numerical Solution

In PRISM the the systems of linear equations can be created by several different engines.
The available engines in PRISM are

- MTBDD
- sparse
- hybrid
- explicit

The first three engines use a (at least to some extend) symbolic representation of the data structure (BDDs, MTBDDs,...), and the fourth engines uses explicit data structures.

Changing the engine will not alter the results however depending on the problem the engines can vary in memory usage and time.

Numerical Solution

The hybrid engine

The hybrid engine is the default engine used in PRISM. It combines symbolic and explicit state data structures. Although it needs slightly more time than other engines, in general it provides the best compromise between time and memory usage.

The sparse engine

Good method for smaller models that require more time for model checking. This engine is faster than the hybrid engine however it requires significantly more memory.

Numerical Solution

The MTBDD engine

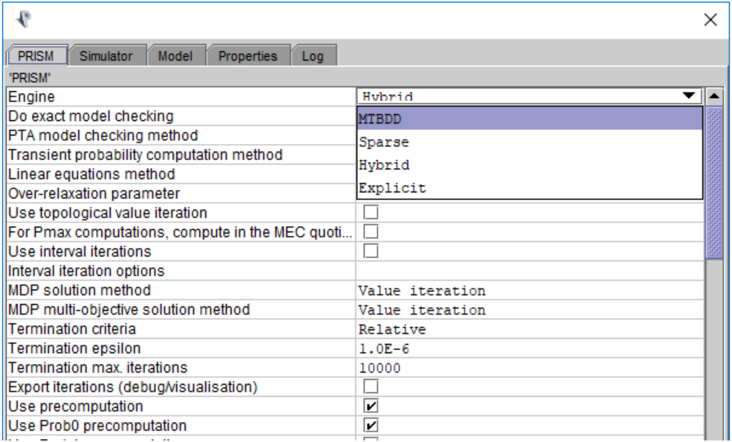
The MTBDD is mostly used for very large well structured models with few distinct probabilities/rates. Therefore this engine is mostly applied to MDP model and much less to CTMC.

The explicit engine

Similar to the sparse engine the explicit engine is mostly used for small models. However the engine uses only explicit data structure which can give advantages in some special cases. (e.g. large state space where only few states are reachable)

Numerical Solution

Setting the engine in PRISM



Iterative Solvers

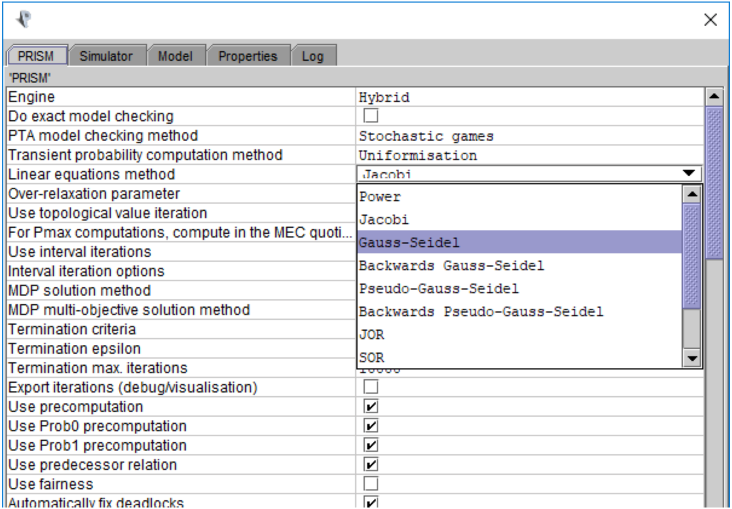
PRISM provides several different methods for solving the linear equations.

- Power method
- Jacobi-Iteration
- Gauss-Seidel
- backwards Gauss-Seidel method
- Jacobi over relaxation (JOR)
- Successive Over-Relaxation (SOR)
- Backwards SOR

However, not every method is available for every engine. (e.g. for the MTBDD engine the Gauss-Seidel and SOR methods are not available)

Iterative Solvers

Setting the solver in PRISM



Convergence of the methods

PRISM checks the convergence of the methods by comparing the maximum difference of successive solutions to a given threshold.
The value of this threshold can be altered by the user however the default value is 10^{-6}.

It is also possible to set an upper limit for the number of iterations performed by a method(default value = 10,000). If a computation reaches this upper limit an error message will be triggered.

References

- http://www.prismmodelchecker.org/lectures/
- http://www.prismmodelchecker.org/manual
- [KNP10c] Marta Kwiatkowska, Gethin Norman and David Parker. Advances and Challenges of Probabilistic Model Checking. In Proc. 48th Annual Allerton Conference on Communication, Control and Computing, pages 1691-1698, IEEE Press. Invited paper. October 2010. http://www.prismmodelchecker.org/papers/allerton10.pdf
- [KNP11] Marta Kwiatkowska, Gethin Norman and David Parker. PRISM 4.0: Verification of Probabilistic Real-time Systems. In Proc. 23rd International Conference on Computer Aided Verification (CAV'11), volume 6806 of LNCS, pages 585-591, Springer. July 2011. http://www.prismmodelchecker.org/papers/cav11.pdf
- Lecture Markov Chains by Assoz. Univ.-Prof Dr. Dmitry Efrosinin http://www.jku.at/stochastik/content/e140956/e199111

Thank you for your attention