# FunSpec4DTMC – A Tool for Modelling Discrete-Time Markov Chains Using Functional Specification

Frederik Hauser, Dominik Krauß, and Michael Menth

University of Tuebingen, Chair of Communication Networks,
Sand 13, 72076 Tuebingen, Germany
{frederik.hauser,menth}@uni-tuebingen.de
johannes-dominik.krauss@student.uni-tuebingen.de

**Abstract.** We present a tool for the analysis of finite discrete-time Markov chains (DTMCs). As a novelty, the tool offers functional specification of DTMCs and implements forward algorithms to compute the stationary state distribution $x_s$ of the DTMC or derive its transition matrix $P$ [19]. In addition, we implement nine direct and indirect algorithms to compute various metrics of DTMCs based on $P$ including an algorithm to determine the period of the DTMC. The tool is intended for both production purposes and as platform for teaching the functional specification of DTMCs. It is published under GPLv3 [3] on Github [2].

## 1 Introduction

Discrete-time Markov chains (DTMCs) are a widely applied concept for system modelling. Typically, DTMCs are defined by a stochastic matrix $P$ that holds probabilities for transitions among system states. The vector $x_n$ describes the state distribution of a system after $n$ transitions. Consecutive distribution vectors $x_n$ are calculated by $x_{n+1} = x_n \cdot P$. The stationary state distribution fulfills $x_s = x_s \cdot P$. It reflects the average state distribution after multiple transitions and is a useful base for the derivation of further specific performance metrics. Theoretical background of DTMCs is described in [21,22]. There are many scientific analysis tools [6,13,14,16–18,23] and libaries for the field of teaching [6,20]. All utilize the transition matrix $P$ as the base for analysis.

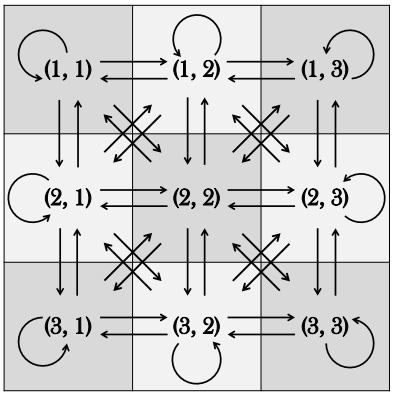
In this work, we present a tool for modelling DTMCs with a finite state space using the novel functional specification suggested in [19]. We introduce the functional specification by an example. We consider a two-dimensional constraint random walk on a grid with coordinates $(a, b)$ and integer values $a \in \{A_{min}, ..., A_{max}\} = \mathcal{A}$ and $b \in \{B_{min}, ..., B_{max}\} = \mathcal{B}$. The walk starts at position $(A_{min}, B_{min})$. In any transition, the position may change horizontally by integer values $\mathcal{H} = \{H_{min}, ..., H_{max}\}$ and vertically by $\mathcal{V} = \{V_{min}, ..., V_{max}\}$, all with equal probability. We consider the system with $A_{min} = B_{min} = 1$, $A_{max} = B_{max} = 3$, $H_{min} = V_{min} = -1$, and $V_{max} = H_{max} = 1$. It can be modelled by a two-dimensional state space $\mathcal{X} = \mathcal{A} \times \mathcal{B}$ and a factor space $\mathcal{Y} = \mathcal{H} \times \mathcal{V}$ with $\mathcal{H} = \mathcal{V} = \{-1, 0, 1\}$. The random variables $X_n = (A_n, B_n) \in \mathcal{X}$ describe the position of the walk after $n$ transitions. Given a random factor for the move $Y = (H, V) \in \mathcal{Y}$, the next position $(A_{n+1},\ B_{n+1}) \in \mathcal{X}$ of the walk is determined by

$$A_{n+1} = min(A_{max}, max(A_{min}, A_n + H)) \tag{1}$$

$$B_{n+1} = min(B_{max}, max(B_{min}, B_n + V)). \tag{2}$$

This is a stochastic recursive equation that constitutes the state transition function of the system $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{X}$. Together with the distribution $y$ of the factor space $\mathcal{Y}$ this function constitutes a DTMC [19]. A so-called forward algorithm may be used to compute consecutive state distributions $x_n$ based on this description without the use of a transition matrix $P$. The transition matrix $P$ can be derived by a similar algorithm so that other analytical methods can be applied to it.

Figure 1(a) shows all states of the random walk with potential transitions. Figure 1(b) illustrates the state transition matrix. The stationary state distribution yields $x_s(i) = \frac{1}{9}$ for any $i \in \mathcal{X}$.




(a) State space and potential transitions.

| | (1,1) | (1,2) | (1,3) | (2,1) | (2,2) | (2,3) | (3,1) | (3,2) | (3,3) |
|---|---|---|---|---|---|---|---|---|---|
| (1,1) | $\frac{4}{9}$, | $\frac{2}{9}$, | 0, | $\frac{2}{9}$, | $\frac{1}{9}$, | 0, | 0, | 0, | 0 |
| (1,2) | $\frac{2}{9}$, | $\frac{2}{9}$, | $\frac{2}{9}$, | $\frac{1}{9}$, | $\frac{1}{9}$, | $\frac{1}{9}$, | 0, | 0, | 0 |
| (1,3) | 0, | $\frac{2}{9}$, | $\frac{4}{9}$, | 0, | $\frac{1}{9}$, | $\frac{2}{9}$, | 0, | 0, | 0 |
| (2,1) | $\frac{2}{9}$, | $\frac{1}{9}$, | 0, | $\frac{2}{9}$, | $\frac{1}{9}$, | 0, | $\frac{2}{9}$, | $\frac{1}{9}$, | 0 |
| (2,2) | $\frac{1}{9}$, | $\frac{1}{9}$, | $\frac{1}{9}$, | $\frac{1}{9}$, | $\frac{1}{9}$, | $\frac{1}{9}$, | $\frac{1}{9}$, | $\frac{1}{9}$, | $\frac{1}{9}$ |
| (2,3) | 0, | $\frac{1}{9}$, | $\frac{2}{9}$, | 0, | $\frac{1}{9}$, | $\frac{2}{9}$, | 0, | $\frac{1}{9}$, | $\frac{2}{9}$ |
| (3,1) | 0, | 0, | 0, | $\frac{2}{9}$, | $\frac{1}{9}$, | 0, | $\frac{4}{9}$, | $\frac{2}{9}$, | 0 |
| (3,2) | 0, | 0, | 0, | $\frac{1}{9}$, | $\frac{1}{9}$, | $\frac{1}{9}$, | $\frac{2}{9}$, | $\frac{2}{9}$, | $\frac{2}{9}$ |
| (3,3) | 0, | 0, | 0, | 0, | $\frac{1}{9}$, | $\frac{2}{9}$, | 0, | $\frac{2}{9}$, | $\frac{4}{9}$ |

(b) State transition matrix P.

Fig. 1: Random walk example.

The functional specification might appear more complex, but provides many benefits. First, it allows intuitive modelling of systems with event-triggered state transitions. Events are modelled by factors whose probabilities are described by the factor distribution. The system's state transition in case of particular events is described by the transition function. Therefore, the functional specification is close to the system's behaviour which facilitates modelling of complex systems with even multi-dimensional state spaces. Second, the functional specification allows the modelling of very large DTMCs. The conventional specification requires the transition matrix $P$ which scales quadratically with the number of states. For very large DTMCs, the resulting size of $P$ may be so large (multiple Terabytes) that DTMC analysis based on $P$ may become infeasible. Sparse matrix representation may reduce memory requirements, but its effectiveness depends on the specific use case. With the functional specification, the transition matrix is not needed for the analysis of the DTMC and memory requirements are reduced to the state and factor distribution. The memory requirement for the state transition function is generally small. In [19] further optimization methods are described to speed up the convergence of the consecutive state distributions based on the functional description.

FunSpec4DTMC implements the functional specification and the forward algorithm to calculate consecutive state distributions $x_n$ and the transition matrix $P$. Besides, the

tool offers various direct and iterative computation methods to calculate metrics for DTMCs that are based on $P$. In particular, the period of finite DTMCs can be derived and methods for output visualization are provided.

The paper is structured as follows. In the next section, we present the core idea and features of FunSpec4DTMC. Section 3 describes the architecture and implementation.

## 2 Tool Description

FunSpec4DTMC consists of a library implementing the functionality and a graphical user interface (GUI) that allows users to analyse DTMCs in an interactive process. The four phases of FunSpec4DTMC's analysis process for DTMCs are depicted in Figure 2.

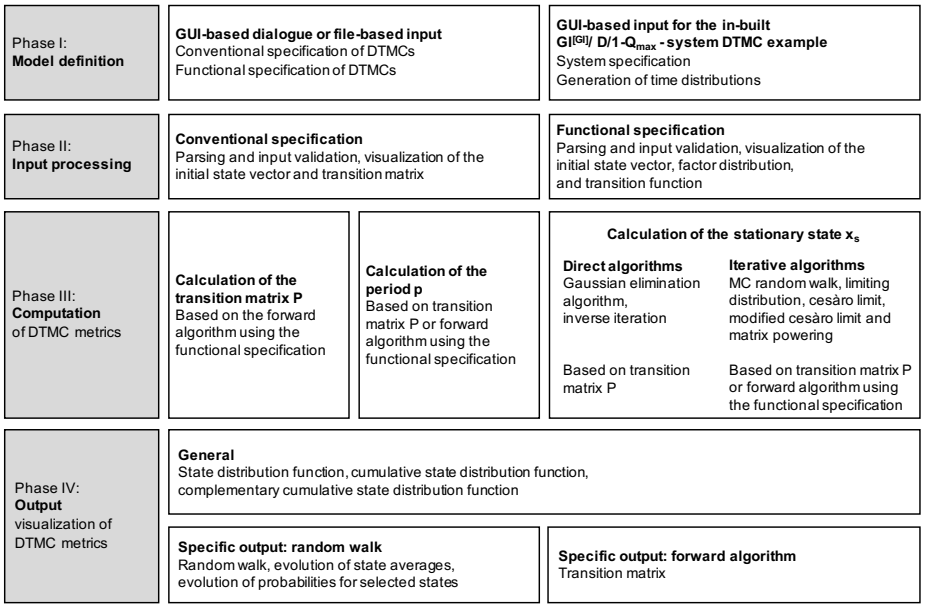



Fig. 2: Four phases of FunSpec4DTMC's analysis process for DTMCs.

In the first phase (I), the DTMC model is defined by the user. DTMCs can be either defined in a GUI-based input dialogue or imported from JSON project files. As an example for intuitive modelling of DTMCs, our tool offers a system specification dialogue for a $GI^{[GI]}/D/1-Q_{max}$ queuing system.

In the second phase (II), the DTMC model input is parsed and validated against mistakes in the specification, e.g., state probability vectors that do not sum up to 1. In addition, aspects of the DTMC model such as the initial state vector can be visualized.

In the third phase (III), metrics for DTMCs are calculated. If the DTMC is defined in a conventional way using the transition matrix $P$, multiple direct and iterative computation algorithms can be applied. The former are accurate and fast but require a large amount of memory. Examples are the Gaussian elimination algorithm and the inverse iteration. The latter requires less memory but lots of iterations to compute the stationary

state distribution with high accuracy. The tool offers the following iterative methods to approximate the stationary state distribution $x_s$:

- DTMC random walk (simulation)
- calculation of the limiting distribution ($\lim_{n\to\infty} x_n$) (applicable to aperiodic DTMCs)
- matrix powering ($\lim_{n\to\infty} \boldsymbol{P}^n$) (applicable to aperiodic DTMCs and to DTMCs with a period of $2^n$, $n \in \mathbb{N}_0$)
- calculation of the Cesàro limit ($\lim_{n\to\infty} \frac{1}{n+1} \sum_{i=0}^{n} x_i$)
- modified calculation of the Cesàro limit ($\lim_{n\to\infty} \frac{1}{p} \sum_{n\leq i<n+p} x_i$) as introduced in [19]. To that end, the tool analyzes transition structures of the DTMC and computes its period $p$.

With a functional specification of a DTMC, the tool computes consecutive state distributions $x_n$ and DTMC simulations without the state transition matrix P and uses for this purpose the forward algorithm or just the state transition function $f$, respectively. Moreover, the state transition matrix P can be derived from the functional specification based on another forward algorithm [19].

In the fourth phase (IV), the output of the DTMC analysis can be visualized. That includes the visualization of general metrics, e.g., the stationary state distribution, and the visualizations of particular computation algorithm specifics, e.g., the random walk.

## 3 Architecture and Implementation

The architecture of FunSpec4DTMC is based on the model-view-controller (MVC) pattern that separates its functionality from the GUI. We designed an object-oriented class hierarchy and applied design patterns, e.g., the observer or strategy pattern [15], and language constructs, e.g., the signal-and-slot approach [10].

We chose Python in version 3.6.3 [8] as programming language. We use Matplotlib [4] to generate plot figures and SciPy [11] to import common distributions. To apply SciPy's continuous distributions on DTMCs, we implemented mechanisms for discretization and normalization. We implemented the GUI using PyQt5 [7], the Python bindings to the widely applied GUI framework Qt [9]. It is platform-independent and allows the creation of more advanced graphical surfaces compared to simple approaches like Tkinter [12]. Python is an interpreted programming language, i.e., source code is translated at runtime. To compensate performance drawbacks, computational-intense functions are implemented in C and called at runtime. External libraries like NumPy [5] adopt this principle and use efficient implementations, e.g., for vector-matrix and matrix-matrix multiplications. We used the Cython [1] extension to implement the forward algorithm's interleaved loops and the model-specific transition functions. Cython allows to implement CPU-intensive modules as C-extensions in a Python-like syntax with additional annotations such as static type declarations. Afterwards, the Cython source code is transformed into C code, compiled, and called at runtime from within Python. Our tool automatically integrates the model-specific transition function defined by the users into the forward algorithm that is a static part of the tool. Analyzing large DTMCs is

limited by the memory on the host system. The functional specification may mitigate but not solve the memory problems that arise with a large number of states. We applied the memory-to-disk swapping mechanism of NumPy so that computing data is stored in a file on the hard disk which can be accessed in small segments.

Within the tool's GUI, users can define multiple projects. For each project, multiple DTMCs can be specified either in the functional or conventional specification within an interactive dialogue or by importing a JSON file. DTMC models and the output of the calculation algorithms can be visualized in multiple plot views. Projects can be stored as files in JSON format and be imported.

## References


1. Cython: C-Extensions for Python. `http://cython.org/`
2. Github: FunSpec4DTMC. `https://github.com/uni-tue-kn/funspec4dtmc`
3. GNU General Public License 3. `https://www.gnu.org/licenses/gpl-3.0.en.html`
4. Matplotlib 2.1.0. `https://matplotlib.org/`
5. NumPy - Scientific Computing with Python. `http://www.numpy.org/`
6. PyPI: discreteMarkovChain. `https://pypi.python.org/pypi/discreteMarkovChain`
7. PyQt5. `http://www.numpy.org/`
8. Python 3.6.3. `https://www.python.org/downloads/release/python-363/`
9. Qt. `https://www.qt.io/`
10. Qt5: Signals & Slots. `http://doc.qt.io/qt-5/signalsandslots.html`
11. SciPy. `https://www.scipy.org/`
12. tkinter. `https://docs.python.org/3.6/library/tkinter.html`
13. Benoit, Anne et al.: The Peps Software Tool. In: Proc. from the 13th Int. Conf. on the Technology of Object-Oriented Languages and Systems (TOOLS '13). Springer (2003)
14. Bini, D. A. et al.: Structured Markov Chains Solver: Software Tools. In: Proc. from the Workshop on Tools for Solving Structured Markov Chains (SMCtools '06). ACM (2006)
15. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley, Boston, MA, USA (1995)
16. Hermanns, Holger et al.: A Set of Performance and Dependability Analysis Components for CADP. In: Proc. of the 9th Int. Conf. on Theory and Practice of Software (TACAS '03). Springer (2003)
17. Katoen, J.P. et al.: The Ins and Outs of the Probabilistic Model Checker MRMC. In: Proc. of the 6th Int. Conf. on the Quantitative Evaluation of Systems (QUEST '09). IEEE Computer Society Press (2009)
18. Kwiatkowska, M et al.: PRISM 4.0: Verification of Probabilistic Real-time Systems. In: Proc. from the 23rd Int. Conf. on Computer Aided Verification (CAV '11). Springer (2011)
19. Menth, M.: Description and analysis of Markov chains based on recursive stochastic equations and factor distributions. World Journal of Modelling and Simulation, World Academic Union, UK Vol. 7(No. 1, pp. 3-15) (2011)
20. Spedicato, G.A., Kang, T.S., Yalamanchi, S.B., Yadav, D.: The markovchain Package: A Package for Easily Handling Discrete Markov Chains in R
21. Stewart, W.J.: Introduction to the Numerical Solution of Markov Chains. Princeton University Press, Princeton, New Jersey (1994)
22. Stewart, W.J.: Probability, Markov Chains, Queues, and Simulation: The Mathematical Basis of Performance Modeling. Princeton University Press, Princeton, New Jersey (2009)
23. Timmer, M. et al.: Efficient Modelling and Generation of Markov Automata. In: Proc. of the 23rd Int. Conf. on Concurrency Theory (CONCUR '12). Springer (2012)


All online resources were accessed on Nov 6, 2017.