






Momba: JANI Meets Python^{*}

Maximilian A. Köhl¹  (✉), Michaela Klauck¹ , and Holger Hermanns^{1,2} 

¹Saarland University, Saarland Informatics Campus, Saarbrücken, Germany

²Institute of Intelligent Software, Guangzhou, China

{koehl,klauck,hermanns}@cs.uni-saarland.de



Abstract. JANI-model [6] is a model interchange format for networks of interacting automata. It is well-entrenched in the quantitative model checking community and allows modeling a variety of systems involving concurrency, probabilistic and real-time aspects, as well as continuous dynamics. Python is a general purpose programming language preferred by many for its ease of use and vast ecosystem. In this paper, we present *Momba*, a flexible Python framework for dealing with formal models centered around the JANI-model format and formalism. *Momba* strives to deliver an integrated and intuitive experience for experimenting with formal models making them accessible to a broader audience. To this end, it provides a pythonic interface for model construction, validation, and analysis. Here, we demonstrate these capabilities.

1 Introduction

Dealing with formal models encompasses a variety of tasks which can be challenging from time to time—especially for newcomers. Everything starts with the *construction* of a model or a family thereof. Often a textual or other, more formal, description of the scenario to be modeled is already existing, such as a rough sketch of the desired behavior or a circuit diagram. Then, after a formal model has finally been conceived, one has to *validate* that the model actually adequately models what should be modeled. In this regard models are just like any other human artifact, inadequate initially but over time it gets better. Only after confidence in the model has been established, one is able to harvest the benefits by handing over the model to *analysis* tools, e. g., a model checker.

In this paper, we present *Momba*, a flexible Python framework for dealing with formal models. *Momba* strives to deliver an integrated and intuitive experience to aid the process of model construction, validation, and analysis. It provides convenience functions for the constructions of models effectively turning Python into a syntax-aware macro language enabling the construction of models in a modular fashion. *Momba*’s built-in simulation engine allows gaining

^{*} This work was partially supported by the ERC Advanced Investigators Grant 695614 (POWVER), by the German Research Foundation (DFG) under grant No. 389792660, as part of TRR 248, see <https://perspicuous-computing.science>, and by the Key-Area Research and Development Program Grant 2018B010107004 of Guangdong Province.

confidence in a model, for instance, by rapidly prototyping a tool for interactive model exploration and visualization, or by connecting it to a testing framework. Finally, thanks to the JANI-model [6] interchange format, several state-of-the-art model checkers and other tools are readily available for analysis. The latest version of Momba is always available on GitHub [1] and the evaluated artifact of this tool demo paper can be found on Zenodo [27].

Why Momba? The idea to harvest a general purpose programming environment for formal modelling is not new at all. For instance, the SVL language combines the power of process algebraic modelling with the power of the bourne shell. As part of many CADP installations [12,13], it is in daily use since its inception [11]. Many formal modeling tools also already provide Python bindings [23,10]. Momba tries not to be yet another incarnation of these ideas.

While the construction of formal models clearly is an integral part of Momba, Momba is more than just a framework for constructing models with the help of Python. Most importantly, it also provides features to work with these models such as a simulator or an interface to different model checking tools. At the same time, it is not just a binding to an API developed for another language, say C++. Momba is *tool-agnostic* and aims to provide a pythonic interface for dealing with formal models while leveraging existing tools. Momba covers the whole process from model creation through validation to analysis. To this end, it is centered around the well-entrenched JANI-model interchange format.

Why JANI? Traditionally, most analysis tools for formal models came with their own modeling languages and formats. The resulting fragmentation hindered interoperability between and comparability across different tools. JANI-model [6] has been conceived with the vision to put an end to this fragmentation. It has since been adopted by many quantitative model checkers [20,21,9] while for others translators have been developed [20,9] enabling cross-tool comparability and fostering competition within the community [22,19,7]. Recently, JANI has also been discovered by the planning community [24,25].

Momba supports all features of the JANI-model specification and some of its optional extensions. JANI is the natural foundation for a project like Momba. It provides a solid, well-established, and powerful modeling formalism for a variety of different kinds of systems involving concurrency, probabilistic and real-time aspects, as well as continuous dynamics. A JANI model is a network of interacting automata with variables. Attached to a model one can also specify various kinds of probabilistic and timed properties which can then be checked by several model checkers, e. g., ePMC [20], The Modest Toolset [21], and Storm [23]. The broad tool support for JANI models enables us to build upon existing research and to outsource computation-intensive tasks via unified interfaces.

Why Python? Python is a popular high-level programming language, preferred by many for its ease of use and ecosystem. Especially within the data-science community, Python is the go-to language for data analysis and machine learning leveraging tools such as TensorFlow [2] and scikit-learn [29]. Around these tools, scientific general purpose tools such as Jupyter [26] have emerged. Jupyter

provides a platform for documenting scientific experiments and their results in a reproducible way combining code, data, and documentation.

Our vision is to harvest Python’s ecosystem and the tools developed by the scientific community for dealing with formal models. Imagine, a Jupyter notebook documenting a model, including the code to construct it, with interactive visualizations of the model itself and various analysis results.

By basing our efforts on a popular language that is appreciated by scientists and established in the scientific community, we hope to lower the entry barrier, especially for those outside the formal methods community.

The User Perspective. In what follows, we demonstrate multiple facets of Momba using a variant of Racetrack, a well-known benchmark in autonomous AI decision making [4,31] which has recently found its use in several model checking contexts [16,3,15]. too. We go through the entire process from the construction of a family of models through their validation to their analysis. For each step, we highlight what Momba has to offer in terms of effectively supporting the process.

Originally Racetrack has been a pen-and-paper game [14]. A *track* is a two-dimensional grid comprising *start*, *goal*, *wall*, and *blank* cells (cf. Fig. 1) [4]. A vehicle starts off with some initial velocity from a start cell, with the objective to reach a goal cell as fast as possible without crashing into a wall. The vehicle is controlled by nine possible actions modifying the current velocity vector. Racetrack naturally lends itself as a benchmark for sequential decision making in risky scenarios, in particular, when extended with probabilistic noise. In a variety of such noisy forms, it has been adopted as a benchmark for *Markov Decision Process* (MDP) algorithms in the AI community [4,5,28,30,31].

For our demonstration, we consider multiple *variants* of Racetrack giving rise to a family of MDPs, studied recently [3] from a feature-oriented perspective [8]. For example, there are different tank options and fuel is consumed according to various consumption models. In addition, there are different undergrounds inducing probabilistic noise modeling slippery road conditions. Clearly, this modeling scenario is beyond what is possible with mere model parametrization, especially so because we are interested in the car’s performance on different tracks each inducing its own MDP [4].

2 Scenario-Based Model Construction

Usually, formal models are not constructed out of thin air but based on some kind of scenario description existing upfront. Such descriptions usually comprise an operational characterization of the behavior to model together with additional and sometimes more formal information about the specific case. Our use case is no exemption, here a textual description of the behavior of the car is provided together with a specific track and a specification of the variant.

Naturally, Python can be used to nicely capture the formal parts of a scenario description in various data structures. Combined with a domain-specific parser for configuration files, scenario descriptions are interchangeable and easy to interface with the code for model construction. In our case, a textual representation of the track (cf. Fig. 1) [4] is provided and parsed together with additional

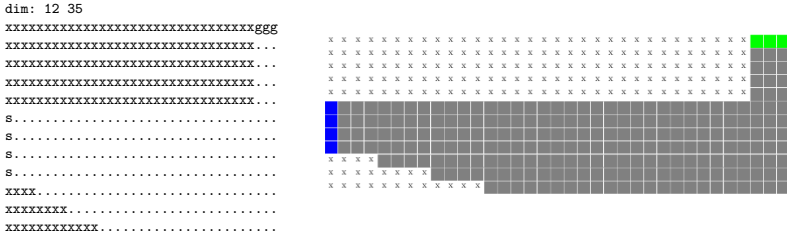


Fig. 1. Textual representation (left) and picture of a track (right): start cells in blue (s), goal cells in green (g), and wall cells marked with x.

parameters, like the size of the tank and the type of the underground, into a data structure tailored to that purpose.

Now, how does Momba support the construction of models from such data structures? A distinguishing feature of Momba is that it effectively turns Python into a syntax-aware macro language enabling the modular construction of models. For our Racetrack use case different fuel consumption models can be captured as macros from JANI expressions to JANI expressions:

```
linear = lambda dx, dy: expr("abs($dx) + abs($dy)", dx=dx, dy=dy)
quadratic = lambda dx, dy: expr("$linear ** 2", linear=linear(dx, dy))
```

A macro is simply a Python function. Upon execution, these macros construct JANI expressions using a straightforward syntax inspired by Python expressions. In this case, both functions take expressions for the current velocity of the vehicle in x and y dimension and return an expression for the resulting fuel consumption which is either *linear* or *quadratic* in the velocity. In contrast to how macros work in languages like C, syntax-aware macros using Momba's `expr` function prevent surprises from mere text-based expansion. Being Python functions, macros can be easily passed around and used elsewhere:

```
assignments = {
    "fuel": expr(
        "min(TANK_SIZE, max(0, fuel - floor($consumption)))",
        consumption=fuel_model(car_dx, car_dy),
    )
}
```

Here, we update the fuel level by taking whatever macro has been provided for computing the fuel consumption. This code is part of constructing an edge for the tank automaton in a modular fashion in the sense that the consumption model is exchangeable. Momba provides further functions, for instance, for declaring variables, like `fuel`, and constructing automata, networks, as well as other model objects. Most of these functions provide all kinds of comforts, for instance, directly checking the types of the involved expressions.

Using syntax-aware macros and Momba's other convenience functions, we arrive at a Python script `racetrack.py` [27] generating a collection of JANI models from scenario descriptions comprising a track and specifying a variant. Iterating over possible scenario descriptions, hundreds of JANI models can be generated fully automatically and consequently be analyzed.

3 Validation by Simulation

Having our models ready, we have to somehow gain confidence that they actually model what we want them to, before handing them over to analysis tools. One way of gaining confidence into a model is by simulating its behavior and manually checking it for consistency with the own understanding of what the model should do. Just like any kind of debugging, this can be a tedious and frustrating process, especially with text-based traces generated by some generic simulator. Momba instead comprises a built-in simulation engine, enabling rapid development of interactive visualizations. This effectively allows us to steer a vehicle through a track thereby exploring a model's behavior, testing edge cases as in a racing game, and ultimately gaining confidence in the model.

Momba's built-in simulation engine supports the simulation of a variety of different JANI models including timed models. It has been written completely from scratch with easy accessibility from Python in mind. Non-determinism can be resolved by uniform random sampling or by querying an external oracle such as, in the case of our interactive visualization, the user, a testing framework, or even a neural network as done for DSMC [16]. For each step, the simulator provides all the necessary information like the binding of variables to values, the locations the various automata of a network are in, and the possible actions (and time delays for timed models) that can be taken. This information can then be extracted and used to display whatever is of interest for understanding and investigating the behavior of the model under scrutiny.

Fig. 2 shows a simple interactive visualization of the Racetrack example based on Momba's simulation engine where the user can steer the vehicle (indicated by the yellow asterisk) through the track by entering acceleration values. Certainly, there is ample room for beautification of this simulator (see TraceVis [15] for example) but for rapid model development this is not needed. After playing around with the interactive simulation for a while and testing various edge cases, we are confident that the model is adequate.

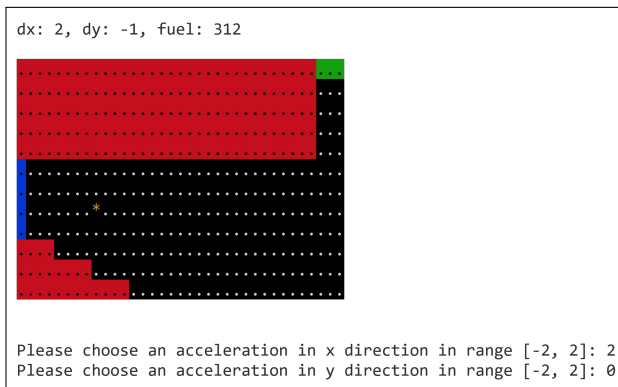


Fig. 2. Interactive visualization using Momba's simulation engine.

4 Harvesting the Benefits

Having constructed the models and gained confidence in their adequacy, we are now ready to harvest the benefits of formal modeling and to apply various state-of-the-art analysis tools, exploiting the JANI-model interchange. Again, Momba provides the necessary functions to define properties and hand our models, with the respective properties attached to them, over to common analysis tools.

Imagine that we are interested in the property $P_{max}(\Diamond \text{ on_goal} \wedge \text{fuel} > 0)$, i. e., the maximal probability of reaching a goal cell with a non-empty tank from a given start cell. Using Momba's syntax-aware macros, we first construct a disjunction over all goal cells and then define the property using the concise syntax provided by Momba's `prop` function:

```
on_goal = reduce(lor, (expr("car_pos == $g", g=g) for g in goal_cells), False)
define_property(
    prop("min({ Pmax(F($on_goal and fuel > 0)) | initial })", on_goal=on_goal),
    name="goalProbabilityFuel",
)
```

After generating a model with the vehicle starting from position (0,7) on the track depicted in Fig. 1 and with sand as underground, the value iteration engine `mcsta` [18] of The Modest Toolset calculates a probability of 87.5% taking 153s when invoked by Momba with the model. Momba also cross-checks the results for us, by invoking Storm's `dd` engine [9] (the fastest engine for this model) and obtains the same result in 107s. These experiments have been carried out on a standard laptop with an Intel Core i7 at 2.7 GHz.

5 Conclusion

We presented Momba, a Python framework for dealing with quantitative models covering the whole process of model creation, validation, and analysis providing an integrated and intuitive experience. In a user story on Racetrack, we demonstrated how Momba's capabilities can be used throughout all stages of the development process of cyber-physical models.

We demonstrated how Momba enables scenario-based model construction with Python code in a concise and modular way with syntax-aware macros. Using Momba's simulation engine, we were able to rapidly prototype an interactive visualization thereby gaining confidence in our models and, finally, thanks to JANI-model, we demonstrated how to analyse our models with state-of-the-art model checkers directly invoked and cross-checked by Momba.

By basing Momba on Python, we aim to harvest the tools developed by the data-science community. Especially, when combined with Jupyter [26], Momba enables literate programming [32] combining code, data, and documentation for reproducible experiments and process documentation.

We hope that Momba helps to open up the world of formal modeling towards a broader community by lowering or removing barriers otherwise obstructing the application of formal models. Momba's infrastructure is implemented in such a way that it can easily be extended into other directions and for connections to other research areas, e. g., model checking policies machine learned with Python libraries [16,17].

References

1. Momba on GitHub, <https://github.com/koehlma/momba>
2. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X.: Tensorflow: A system for large-scale machine learning. In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. p. 265283. OSDI'16, USENIX Association, USA (2016)
3. Baier, C., Dubslaff, C., Hermanns, H., Klauck, M., Klüppelholz, S., Köhl, M.A.: Components in probabilistic systems: Suitable by construction. In: *Proceedings of the 9th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation. X by Construction*. (2020)
4. Barto, A.G., Bradtke, S.J., Singh, S.P.: Learning to act using real-time dynamic programming. *Artificial Intelligence* **72**(1), 81 – 138 (1995). [https://doi.org/10.1016/0004-3702\(94\)00011-O](https://doi.org/10.1016/0004-3702(94)00011-O)
5. Bonet, B., Geffner, H.: Labeled RTDP: improving the convergence of real-time dynamic programming. In: *ICAPS*. pp. 12–21 (2003)
6. Budde, C.E., Dehnert, C., Hahn, E.M., Hartmanns, A., Junges, S., Turrini, A.: JANI: Quantitative model and tool interaction. In: Legay, A., Margaria, T. (eds.) *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part II. Lecture Notes in Computer Science*, vol. 10206, pp. 151–168 (2017). https://doi.org/10.1007/978-3-662-54580-5_9
7. Budde, C.E., Hartmanns, A., Klauck, M., Kretinsky, J., Parker, D., Quatmann, T., Turrini, A., Zhang, Z.: On Correctness, Precision, and Performance in Quantitative Verification (QComp 2020 Competition Report). In: *Proceedings of the 9th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation. Software Verification Tools*. (2020)
8. Chrszon, P., Dubslaff, C., Klüppelholz, S., Baier, C.: Profeat: feature-oriented engineering for family-based probabilistic model checking. *Formal Aspects Comput.* **30**(1), 45–75 (2018). <https://doi.org/10.1007/s00165-017-0432-4>, <https://doi.org/10.1007/s00165-017-0432-4>
9. Dehnert, C., Junges, S., Katoen, J., Volk, M.: A storm is coming: A modern probabilistic model checker. In: Majumdar, R., Kuncak, V. (eds.) *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II. Lecture Notes in Computer Science*, vol. 10427, pp. 592–600. Springer (2017). https://doi.org/10.1007/978-3-319-63390-9_31
10. Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., Xu, L.: Spot 2.0a framework for ltl and ω -automata manipulation. In: *International Symposium on Automated Technology for Verification and Analysis*. pp. 122–129. Springer (2016)
11. Fernandez, J., Garavel, H., Kerbrat, A., Mounier, L., Mateescu, R., Sighireanu, M.: CADP - A protocol validation and verification toolbox. In: Alur, R., Henzinger, T.A. (eds.) *Computer Aided Verification, 8th International Conference, CAV '96, New Brunswick, NJ, USA, July 31 - August 3, 1996, Proceedings. Lecture Notes in Computer Science*, vol. 1102, pp. 437–440. Springer (1996). https://doi.org/10.1007/3-540-61474-5_97

12. Garavel, H., Lang, F., Mateescu, R., Serwe, W.: CADP 2011: a toolbox for the construction and analysis of distributed processes. *Int. J. Softw. Tools Technol. Transf.* **15**(2), 89–107 (2013). <https://doi.org/10.1007/s10009-012-0244-z>
13. Garavel, H., Lang, F., Mounier, L.: Compositional verification in action. In: Howar, F., Barnat, J. (eds.) *Formal Methods for Industrial Critical Systems - 23rd International Conference, FMICS 2018, Maynooth, Ireland, September 3-4, 2018, Proceedings. Lecture Notes in Computer Science*, vol. 11119, pp. 189–210. Springer (2018). https://doi.org/10.1007/978-3-030-00244-2_13
14. Gardner, M.: Mathematical games. *Scientific American* **229**, 118–121 (1973)
15. Gros, T.P., Groß, D., Gumhold, S., Hoffmann, J., Klauck, M., Steinmetz, M.: TraceVis: Towards Visualization for Deep Statistical Model Checking. In: *Proceedings of the 9th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation. From Verification to Explanation.* (2020)
16. Gros, T.P., Hermanns, H., Hoffmann, J., Klauck, M., Steinmetz, M.: Deep statistical model checking. In: Gotsman, A., Sokolova, A. (eds.) *Formal Techniques for Distributed Objects, Components, and Systems - 40th IFIP WG 6.1 International Conference, FORTE 2020, Held as Part of the 15th International Federated Conference on Distributed Computing Techniques, DisCoTec 2020, Valletta, Malta, June 15-19, 2020, Proceedings. Lecture Notes in Computer Science*, vol. 12136, pp. 96–114. Springer (2020). https://doi.org/10.1007/978-3-030-50086-3_6
17. Gros, T.P., Höller, D., Hoffmann, J., Wolf, V.: Tracking the race between deep reinforcement learning and imitation learning. In: Gribaudo, M., Jansen, D.N., Remke, A. (eds.) *Quantitative Evaluation of Systems - 17th International Conference, QEST 2020, Vienna, Austria, August 31 - September 3, 2020, Proceedings. Lecture Notes in Computer Science*, vol. 12289, pp. 11–17. Springer (2020). https://doi.org/10.1007/978-3-030-59854-9_2
18. Hahn, E.M., Hartmanns, A.: A comparison of time- and reward-bounded probabilistic model checking techniques. In: Fränzle, M., Kapur, D., Zhan, N. (eds.) *Dependable Software Engineering: Theories, Tools, and Applications - Second International Symposium, SETTA 2016, Beijing, China, November 9-11, 2016, Proceedings. Lecture Notes in Computer Science*, vol. 9984, pp. 85–100 (2016). https://doi.org/10.1007/978-3-319-47677-3_6
19. Hahn, E.M., Hartmanns, A., Hensel, C., Klauck, M., Klein, J., Kretínský, J., Parker, D., Quatmann, T., Ruijters, E., Steinmetz, M.: The 2019 comparison of tools for the analysis of quantitative formal models - (QComp 2019 competition report). In: Beyer, D., Huisman, M., Kordon, F., Steffen, B. (eds.) *Tools and Algorithms for the Construction and Analysis of Systems - 25 Years of TACAS: TOOLympics, Held as Part of ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part III. Lecture Notes in Computer Science*, vol. 11429, pp. 69–92. Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_5
20. Hahn, E.M., Li, Y., Schewe, S., Turrini, A., Zhang, L.: iscasmc: A web-based probabilistic model checker. In: Jones, C.B., Pihlajasaari, P., Sun, J. (eds.) *FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings. Lecture Notes in Computer Science*, vol. 8442, pp. 312–317. Springer (2014). https://doi.org/10.1007/978-3-319-06410-9_22
21. Hartmanns, A., Hermanns, H.: The Modest Toolset: An integrated environment for quantitative modelling and verification. In: Ábrahám, E., Havelund, K. (eds.) *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014.*

- Proceedings. Lecture Notes in Computer Science, vol. 8413, pp. 593–598. Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_51
22. Hartmanns, A., Klauck, M., Parker, D., Quatmann, T., Ruijters, E.: The Quantitative Verification Benchmark Set. In: Vojnar, T., Zhang, L. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6–11, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11427, pp. 344–350. Springer (2019). https://doi.org/10.1007/978-3-030-17462-0_20
 23. Hensel, C., Junges, S., Katoen, J., Quatmann, T., Volk, M.: The probabilistic model checker storm. CoRR **abs/2002.07080** (2020), <https://arxiv.org/abs/2002.07080>
 24. Hoffmann, J., Hermanns, H., Klauck, M., Steinmetz, M., Karpas, E., Magazzeni, D.: Let’s learn their language? A case for planning with automata-network languages from model checking. In: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7–12, 2020. pp. 13569–13575. AAAI Press (2020)
 25. Klauck, M., Steinmetz, M., Hoffmann, J., Hermanns, H.: Bridging the gap between probabilistic model checking and probabilistic planning: Survey, compilations, and empirical comparison. J. Artif. Intell. Res. **68**, 247–310 (2020). <https://doi.org/10.1613/jair.1.11595>
 26. Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B.E., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J.B., Grout, J., Corlay, S., et al.: Jupyter notebooks—a publishing format for reproducible computational workflows. In: ELPUB. pp. 87–90 (2016)
 27. Köhl, M.A., Klauck, M., Hermanns, H.: (TACAS21 Artifact) Momba: JANI Meets Python. <https://doi.org/10.5281/zenodo.4431780>
 28. McMahan, H.B., Gordon, G.J.: Fast exact planning in Markov decision processes. In: ICAPS. pp. 151–160 (2005)
 29. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in python. the Journal of machine Learning research **12**, 2825–2830 (2011)
 30. Pineda, L.E., Lu, Y., Zilberstein, S., Goldman, C.V.: Fault-tolerant planning under uncertainty. In: IJCAI. pp. 2350–2356 (2013)
 31. Pineda, L.E., Zilberstein, S.: Planning under uncertainty using reduced models: Revisiting determinization. In: Chien, S.A., Do, M.B., Fern, A., Ruml, W. (eds.) Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21–26, 2014. AAAI (2014)
 32. Ruys, T.C., Brinksma, E.: Experience with literate programming in the modelling and validation of systems. In: Steffen, B. (ed.) Tools and Algorithms for Construction and Analysis of Systems, 4th International Conference, TACAS ’98, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS’98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1384, pp. 393–408. Springer (1998). <https://doi.org/10.1007/BFb0054185>

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

