Mochy : a tool for the modeling of concurrent hybrid systems

Loïc Hélouët, Antoine Thébault

Mochy : a tool for the modeling of concurrent hybrid systems

Loïc Hélouët and Antoine Thébault

Inria, Rennes, France

Abstract. This paper introduces MOCHY, a tool designed for the modeling of concurrent systems with variants of stochastic, timed and hybrid Petri nets. Beyond modeling, the tool serves as a platform for fast simulation, and can be used for statistical verification of properties, controller testing, and learning of control rules. The targeted models are variants of stochastic and timed nets where tokens can be continuous quantities depicting trajectories of moving objects. The architecture of the tool is designed to be as adaptive as possible, and allow the redefinition of objects behaviors or transitions firing through the refinement of a few semantic rules. The framework also allows for the integration of controllers. For any model variant, MOCHY can perform fast simulation, and perform statistical verification, evaluate some quantitative properties of a model, or learn control rules for reachability or quantitative objectives.

1 Introduction

This paper introduces a new tool called MOCHY, tailored for the modeling of systems with timed variants of Petri nets, and for fast simulation. The origin of MOCHY stems from the need of fast simulation tools to test traffic management policies for metro networks [Kec19]. We rapidly came to the conclusion that transport networks had so many specificities that time Petri nets, timed Petri nets, or most of their variants were not adapted to the design of such models. First of all, even if models such as TPNs are Turing powerful, and can hence simulate most systems, using this expressive power in practice forces to loose the graphical and concurrent nature of nets and results in complex models that are hard to simulate, and cannot be understood by humans. A way to circumvent these issues were to tune existing models to obtain ad-hoc variants of Petri nets. However, this was not satisfactory either, because every transport network comes with its own traffic management policy, i.e. a light form of control that is used to mitigate effects of incidents and delays, that affects the semantics of the model. With this additional constraint, every transport networks can have its own ad-hoc semantics, and is hence a new kind of model. An example of variation point for instance is whether a metro network follows a fixed block policy allowing at most one train in each track segment, or a moving block policy that allows several trains in a segment provided they preserve safety headways.

The main principles of MOCHY are the following: we consider timed models that can depict trajectories of objects in a bounded environment, such as trains

on a track, cars on a road lane, manufactured objects on a conveyor,... The simulation scheme of the tool is designed to be as generic as possible. To reach this objective, the semantics of a model is given in a few generic operational rules depicting how the state of a network transforms upon occurrence of a discrete event, or when time elapses. There rules are repeated within a simulation loop, that may use a controller to select the next actions or delays allowed. Controllers also maintain a memory that can be used for further decisions.

This paper is organized as follows: we first describe the general architecture of the tool, the common features of models that can be simulated by MOCHY, the rules used by its adaptive semantics, and the way runs of MOCHY models are simulated. We then explain how MOCHY has been used to model a metro line in Rennes, and to learn a controller which aim is to help a metro recovering from a bunching situation. We conclude with related work, and perspectives on the future development of the tool.

2 Mochy Description

MOCHY is designed to be as modular and adaptable as possible. Its architecture decouples semantics, interfaces, simulation scheme and control. This architecture allow redefinition of a semantics attached to a particular project, modification of a controller to guide choices of delays and actions during simulation, etc. This approach was proved particularly useful when considering models for metro networks: several train management policies have been implemented by simply changing the controller part of the tool.

2.1 Architecture

Mochy's architecture is composed of four main parts : an interface, and three modules describing a Physical Model, a Controller and a Simulator loop. The Physical Model contains the data structures needed to describe the Petri net variant that will be simulated, its initial configuration (mainly initial contents of its places) a net type, which will be used to load a class implementing the semantics rule of the net and a controller if needed. The physical model is loaded from an input project file. The Core part is composed of semantics rules and of a controller. This part is composed of classes that are loaded once the type of net is known from the input project file. The controller and semantic classes define the effects of possible actions and delays on configurations. The Simulator part implements a simulation loop, i.e. it handles semantics rules and control to animate a fixed number of semantics steps set from the interface. The interface allows interaction with these modules, by displaying current configuration of a loaded model, and providing access to the simulation functions of the tool and to standard functions (load, save...).

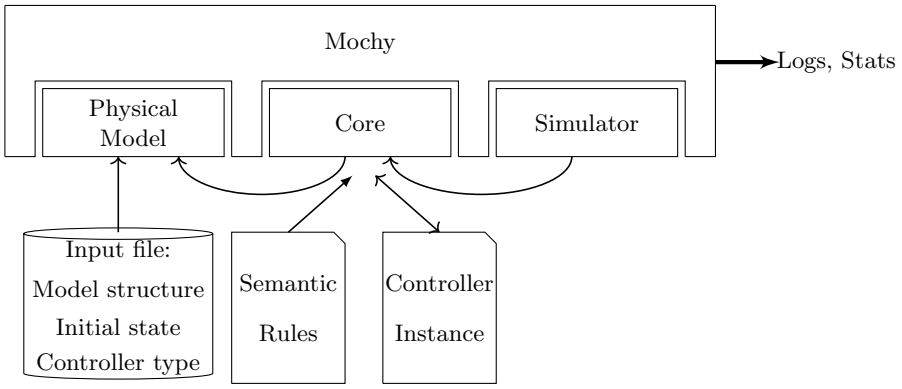

Fig. 1. Mochy architecture

The Physical Model The goal of MOCHY is to provide a generic tool for the analysis and simulation of Petri nets variants. The models used as input for MOCHY share some common characteristics, namely notions of events represented by transitions of a net, notions of resources represented by places, and the flows of resources consumed/produced by transitions, and timing information. The common elements of MOCHY models are hence close to those of a stochastic timed Petri-Net. The variation points are the contents of places, the way they evolve over time, the firing rules of transitions, and the way a firing of a transition affects place contents. The contents of places can be simple tokens, or more complex objects evolving in a multi dimensional space. We have specialized this generic description to address models for transport networks.

Structure of nets The models used by MOCHY are variants of Petri nets with time. They share common features, such as the notions of places, transitions, flows, and time intervals. **Places** are contents holders for quantities that may evolve over time. For instance, they can be used as containers for tokens, i.e. integral numbers that are not affected by time elapsing, but only by discrete transition firings. Conversely, places can represent a physical space such as track portion with boundaries, where objects have trajectories. **Transitions** represent classes of events. As usual in Petri nets, they have a preset, i.e. a set of places depicting resources needed for an occurrence of the transition, a postset, i.e. a set of places depicting resources impacted by the firing of a transition. A transition t can be triggered upon conditions that depend on time, and on the contents of places in the preset and in the postset of t. These conditions and the effect of a transition firing vary depending on the semantics rules.

The core structure of a net is hence a tuple $\mathcal{N} = (P, T, A, I)$, where P is a set of places, T is a set of transitions representing events in a system, $A \subseteq (P \times T) \cup (T \times P)$ is a set of arcs connecting places to transitions, and transitions to places. Map $I : T \to \mathbb{Q} \times \mathbb{Q} \times DF$ associates a time interval $[\alpha(t), \beta(t)]$ and a distribution function $f_t : [\alpha(t), \beta(t)] \to [0, 1]$ to each transition $t \in T$.

Place contents Standard timed variants of Petri nets (Time Petri nets, timed arc Petri nets, stochastic nets...) manipulate tokens that are put in places, and moved by each transition firing. In the models addressed by MOCHY, we allow for the definition of more complex place contents. For instance, we have used MOCHY to design models for metro networks, where trains can move at several speeds on a track segment as soon as they respect some safety headways. In one of the studied models, called *trajectory nets*, some places represent track segments, and their contents are trajectories of trains in a track portion depicted by *space-time* diagrams (see [Kec19] for a complete description of this model). Figure 2 shows an example of a configuration of a trajectory net. Place p_1 contains a space-time diagram with two train trajectories, depicting how the remaining distance to arrival evolves over time for each train. As one can imagine, as time elapses, the remaining distance to arrival decreases, which modifies the contents of places. When the remaining distance of a trajectory is zero, this trajectory is moved to another place by firing of a transition. The duration of the newly created trajectory is sampled from an interval attached to the transition that will consume it, and its initial distance is obviously the size of the track segment represented by the place entered. The semantics of the model also enforce safety of trains by allowing only trajectories preserving a sufficient headway between trains. This type of model was successfully implemented by instantiating the high-level semantics rules specified below.

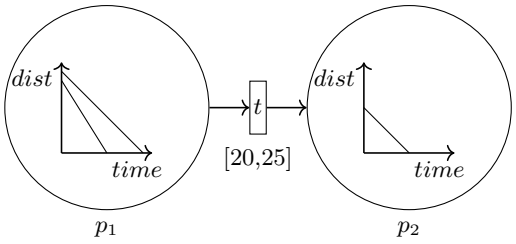

Fig. 2. A simple Trajectory Net.

2.2 Semantics

The current state of a model is called a *configuration*. Configurations are denoted by $C_1, C_2, \ldots$. The semantics of models designed with MOCHY are defined in terms of *timed moves* of the form $C_i \xrightarrow{\delta} C_{i+1}$ which describe how a system evolves when a certain amount of time δ elapses, and *discrete moves* of the form $C_i \xrightarrow{ev} C_{i+1}$, that describe how a system evolves when a discrete event ev (usually the firing of a transition) occurs. We assume a sampling semantics, that is when a transition gets enabled, the time before its urgent firing is chosen according to the current configuration and never changed. In the context of time Petri nets for instance, this corresponds to sampling a duration δ_t within a

interval $[\alpha, \beta]$ for every newly enabled transition t when using a discrete move, and considering t as urgent δ_t time units later if it was not disabled before.

In a configuration C, a transition is *urgent* if it **has to** fire or be disabled before some time elapses. In urgent semantics timed moves are forbidden in a configuration C if C has urgent transitions. The notion of urgency may vary from a variant of a model to another. In trajectory nets [Kec19], transitions are urgent if a trajectory of an object has reached a border of the physical space depicted by a place, and a transition is ready to move this object to another part of the net. To allow for the specification of many models, the semantics of a model in MOCHY is given by redefinition of two configuration transformation rules and two functions to test place contents or check values of clocks:

- $R_1(C, \delta)$: depicts the transformation occurring in a configuration C when δ time units elapse
- $R_2(C, ev)$: depicts the transformation occurring in a configuration C when event ev occurs (mainly firing of a transition)
- $F_1(C)$: returns the time that can elapse before a transition becomes urgent
- $F_2(C)$: returns the list of transitions that are firable in C.

2.3 Simulation

Configurations and transformation rules differ for every type of model, but providing functions R_1, R_2, F_1, F_2 is sufficient to implement the simulation loop proposed in Algorithm 1 next page. Creating a model with a new semantics in MOCHY hence boils down to coding these rules and functions. Then simulation of a model with MOCHY consists in repeatedly deciding which transition to fire or which delay to elapse.

Algorithm 1 Rule-base Operational semantics for MOCHY

```
set a number of steps n
set an initial configuration C_0
i ← 1
while i ≤ n do
   L = F_2(C)
   if L = ∅ then
      δ = F_1(C)
      Use rule R_1(C, δ) : C --δ--> C'
   else
      Use controller to choose t ∈ L and update controller's memory
      Use rule R_2(C, t) : C --t--> C'
   end if
   i++
end while
```

Let us detail the role of controllers during during simulation. When several transitions are firable, a controller is used to choose which transition fires.

Controllers can be any program making such choice. The most basic controllers chose randomly a transition among firable ones. More involved controllers can be equipped with memory, with a schedule to follow, etc. and can implement complex strategies. We detail in Section 3 a controller designed to fix speed and dwell time of metros to recover from bunching situations.

2.4 Inputs-Outputs

MOCHY takes as input **project files** that contain : the class of model used by the project (it is a compiled Java class that implements the semantics rules), a description of the structure of an instance of the loaded model: the places, transitions, and flow relations of the net, the time intervals attached to transitions and the associated distributions. Depending on the specialization of the model, the project file can also provide pointer to additional features : a controller, schedules, etc.). This approach allowed for the modeling of metro networks equipped a traffic management algorithm designed to adhere to a timetable.

Once a file is loaded, users can play with the specification in an interactive manner, or run a simulation for several steps, following the simulation procedure of Algorithm 1. At each stop of the simulator, the contents of places is displayed. The net can be reset to its initial configuration at every stop. During simulation, the tool generates **logs**, remembering dates and transitions fired, and a selection of a few variables for further statistics computed from the saved logs.

2.5 User Interface

Figure 3 describes the main window of MOCHY. It is divided in three horizontal panels. The top of the window shows the structure of the simulated net, and its current configuration. The middle of the window in a control panel for simulation, that allows launching a simulation for a given number of steps, letting time elapse, or firing a single transition. The bottom part of the interface contains two consoles to display simulation logs, warnings and statistics such as the mean duration of a simulation after an extensive simulation campaign, or any measure of a performance indicator.

3 Case Studies

MOCHY has been tested successfully on several models of metro networks, with various semantics. In this section, we present an first experiment that successfully used MOCHY to test accuracy of a traffic management algorithms, and a second one that allowed to train a controller to avoid train bunching.

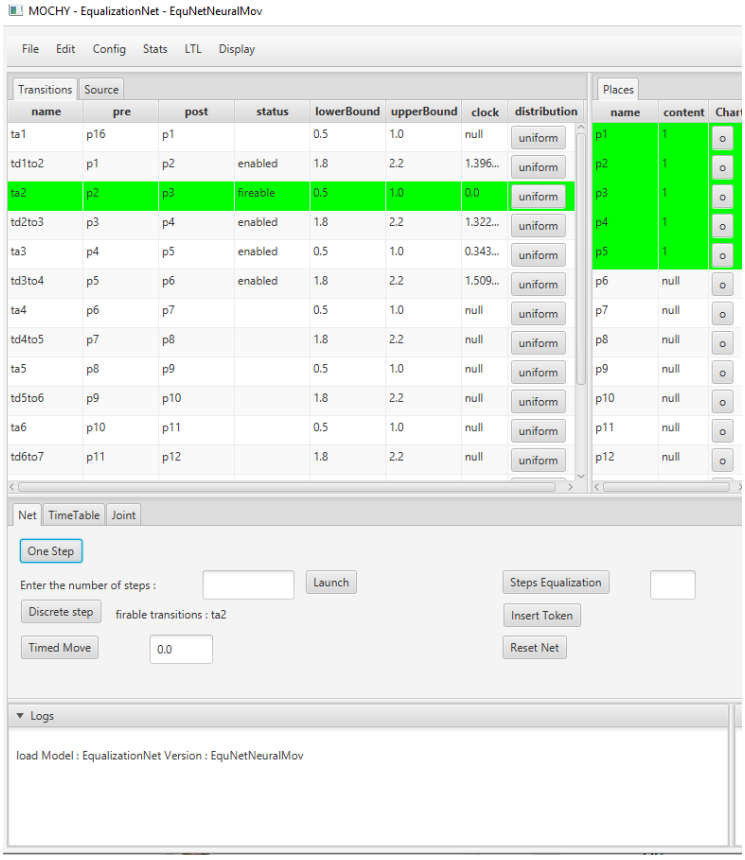

Fig. 3. The main window

A Metro line in Rennes The Metro line A in Rennes has 15 stations and a length of 9 kilometres. The model developped with MOCHY uses a controller that implements a traffic management policy which goal is to give dwell duration and speed advices to trains in order to adhere to a given timetable. The timetable describes planned operations for 4 hours, and contains dates for more than 3000 events. The net model is composed of 56 places, 63 transitions. MOCHY showed good performance for the simulation of this metro network : simulation of 10 runs (i.e. sequences of at least 3000 discrete moves) can be performed in 1.5 minutes on an average laptop.

Regulation by Equalization The good performance of the tool allows for its use for applications that require intesive simulation campaigns, such as statistical model-checking or learning techniques. We have used MOCHY to train a neural netwrok in charge of controlling a metro network to recover from a bunching

situation. Bunching is a situation where all trains are not well distributed on a network, causing long periods without service in stations, followed by arrival of many trains in a short amount of time. This situation is depicted in Figure 4.

We have considered a simple network, namely a loop of 20 kilometers, modeled by a net with 60 places and 60 transitions equipped with a controller that aims at equalizing distances between trains in the network (see [HFT22] for more details on the experiment). Several traffic management algorithms have been tested. For each of them, the decision taken was to choose an appropriate speed and dwell time for trains stopped at a quay. The first tested approach was based on optimization of a quadratic function considering distances of a train wrt to its predecessor and its successor. The second approach tested was a neural network, with similar parameters as input, and trained with a genetic learning approach [PHU05], selecting mutations improving the controller's statistics during intensive simulation campaigns realized with MOCHY. The tool allowed to simulate runs of duration of up to 2 hours for fleet sizes ranging from 5 to 50 trains in less than 15 seconds. Both controller types were evaluated w.r.t the time needed to return to an equilibrate distribution of trains and to the average speed of trains after this equalization. Figure 5 shows the performance of a neural network controller trained with MOCHY. The black curve represents the average time needed to recover from the worst possible bunching situation, and the blue line the average speed (in meters/min, with an objective of a 500 m/min) once the equalization is performed.

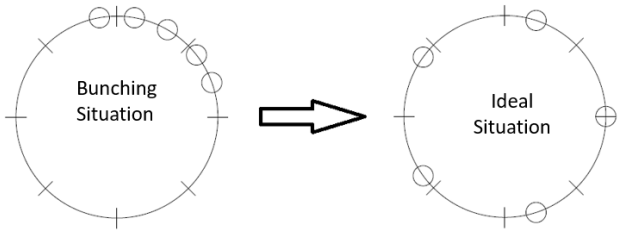

Fig. 4. From a bunching situation to a good distribution of trains in a Metro network.

4 Conclusion

The MOCHY toolbox can be freely downloaded at the following url:https://adt-mochy.gitlabpages.inria.fr/mochy/. The available packages contain implementation of semantic rules for several variants of nets, including waiting nets [HA22] and trajectory nets [Kec19] and examples of models for metro networks. Future distributions will include a statistical model checker for Signal LTL, and Machine Learning techniques to train controllers.

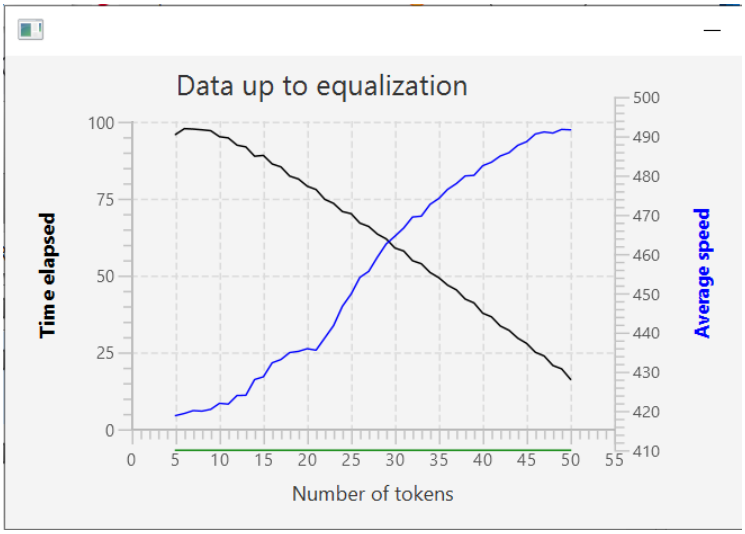

Fig. 5. Time to equalization guided by a neural network for varying fleet sizes.

Several tools dedicated to Petri nets and their variants exist (see [PNT] for an extensive list). Some of them can handle time, complex firing rules or hybrid variables needed to model trajectories of objects. Many tools are dedicated to time(d) Petri nets and their variants. Romeo [LRST09] is a verification tool for time Petri nets (TPNs). In addition to the standard urgent semantics of TPNs, ROMEO allows for the specification of ad-hoc firing rules, read/inhibitor arcs, and for parameters synthesis. ORIS [CPV22] is close to ROMEO, but is taylored for transient analysis of stochastic timed Petri nets. Tapaal [BS09] targets verification of timed arcs Petri nets. It allows inhibitor/read arcs. TINA [BV06] is an analyser for TPNs extended with read arcs, inhibitor arcs, open intervals and Data. Several other tools can perform simulation of timed and stochastic variants of nets. ARP analyzes and simulates nets where transitions are attached an interval and a distribution over this interval. Petrisim allows for the simulation of Petri nets with delay between token production and consumption.

Some tools target net variants with colors, a way to introduce variables and data in nets. Alpha/Sim allows for the simulation of stochastic, timed, attributed or colored Petri nets. Great SPN is a tool for generalized (colored) Stochastic Petri nets, that allows for timed simulation. ExSpect is dedicated to the design of business processes and is formally based on colored Petri nets. CPN tools allows for the definition of colored High-level nets and nets with time, and can simulate them to analyse performance of the modeled systems. PnetLab is a tool dedicated to the control of High-level coloured Petri net and allows in addition side management of time to test scheduling strategies when transitions are given a service time. TimeNET is a tool for the modelling and analysis of stochastic

Petri nets with non-exponentially distributed firing times. It supports graphical modeling of uncolored and colored Petri nets as well as Markov chains. Numerous performance evaluation and structural analysis algorithms are available as well as an interactive token game.

Modeling of transport systems calls for mechanisms that can encode objects movements, road/track bounds, safety distances. Obviously, this can be simulated by Turing powerful models such as all colored variants, time Petri nets, etc. but at the cost of low-level encodings of objects movements, that are more intuitively captured by continuous or hybrid variants of nets. QPME is a tool that implements Queueing Petri nets. This type of nets/tools is of great interest for the design and analysis of transport systems, but does not allow for the modeling of constraints among the moving objects. As for queuing theory, analyses lead either to optimistic or pessimistic performance evaluation wrt the actual behavior of a train network. Batch Petri nets [Dem01], or Differential Petri nets [HMM09], and the tool Hisim can handle mixed discrete and continuous tokens/ transitions, where places contain quantities that evolve according to differential equations. Discrete transitions firings have the usual semantics, and continuous firings are allowed when place contents exceed some threshold, and moves some quantities of token per time unit. Hisim simulates hybrid nets via a simulation loop that : fires immediate transitions, computes the next event date, progress time to this date, and iterates. Simulation in MOCHY is based on a similar simulation loop.

Fluid-survival-tool [PRHG14] considers Hybrid Petri nets to model systems with discrete and continuous quantities, and provides tools to compute the probability to be in a given state at a certain time, or to verify Stochastic Timed Logic. Time is handled by attaching constant firing times to discrete transitions, distributions on firing times to stochastic transitions, and firing speed to continuous ones.

References

[BS09] K.Y Byg, J.and Jørgensen and J. Srba. TAPAAL: editor, simulator and verifier of timed-arc petri nets. In *Proc. of ATVA 2009*, volume 5799 of *LNCS*, pages 84–89, 2009.

[BV06] B. Berthomieu and F. Vernadat. Time petri nets analysis with TINA. In *Proc. of (QEST 2006*, pages 123–124. IEEE Computer Society, 2006.

[CPV22] L. Carnevali, M. Paolieri, and E. Vicario. The ORIS tool: app, library, and toolkit for quantitative evaluation of non-markovian systems. *SIGMETRICS Perform. Evaluation Rev.*, 49(4):81–86, 2022.

[Dem01] I. Demongodin. Generalised batches petri net: Hybrid model for high speed systems with variable delays. *Discret. Event Dyn. Syst.*, 11(1-2):137–162, 2001.

[DK06] I. Demongodin and N.T. Koussoulas. Differential petri net models for industrial automation and supervisory control. *IEEE Trans. Syst. Man Cybern. Syst.*, 36(4):543–553, 2006.

[HA22] L. Hélouët and P. Agrawal. Waiting nets. In Luca Bernardinello and Laure Petrucci, editors, *PETRI NETS 2022*, volume 13288 of *LNCS*, pages 67–89, 2022.

[HFT22] L. Hélouët, E. Fabre, and A. Thébault. Optimization of traffic management with learning machines. *hal-03777459*, 2022.

[HMM09] F. Hamdi, N. Messai, and N. Manamanni. Design of switched observer using timed differential petri nets: A dwell time approach. In *Proc. of European Control Conference, ECC 2009*, pages 4641–4646. IEEE, 2009.

[Kec19] Karim Kecir. *Performance Evaluation of Urban Rail Traffic Management Techniques. (Évaluation de Performances pour les Techniques de Régulation du Trafic Ferroviaire Urbain)*. PhD thesis, University of Rennes 1, France, 2019.

[LRST09] D. Lime, O.H. Roux, C. Seidner, and L-M Traonouez. Romeo: A parametric model-checker for petri nets with stopwatches. In *Proc. of TACAS 2009*, volume 5505 of *LNCS*, pages 54–57, 2009.

[PHU05] P.P Palmes, T. Hayasaka, and S. Usui. Mutation-based genetic neural network. *IEEE Trans. Neural Networks*, 16(3):587–600, 2005.

[PNT] Petri nets tools database quick overview. https://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html. Accessed: 2023-02-01.

[PRHG14] B.F. Postema, A. Remke, B.R. Haverkort, and H. Ghasemieh. Fluid survival tool: A model checker for hybrid petri nets. In *Proc. of Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance - 17th International GI/ITG Conference, MMB & DFT 2014*, volume 8376 of *LNCS*, pages 255–259, 2014.