33rd European Modeling & Simulation Symposium
18th International Multidisciplinary Modeling & Simulation Multiconference

ISSN 2724-0029 ISBN 978-88-85741-57-7 © 2021 The Authors.
doi: 10.46354/i3m.2021.emss.044

Developing a Bridge from GPenSIM to NuSMV for Model Checking

Albana Roci[1,*] and Reggie Davidrajuh[1]

[1]University of Stavanger, Stavanger, Norway

[*]albana.roci@uis.no

Abstract

The general-purpose Petri Nets Simulator (GPenSIM) is an extensible, easy to use, and flexible Petri Nets modeling tool. It is used to model, simulate and evaluate the performance of discrete event systems. Nevertheless, GPenSIM does not support the formal verification of the nets. The new symbolic model checker (NuSMV) is a state-of-art model checking tool that automatically examines whether a finite transition system (TS) satisfies the property specification under consideration. This paper introduces an algorithm that implicitly utilizes NuSMV in GPenSIM. The algorithm automatically converts the static safe Petri Nets model generated by GPenSIM to the NuSMV description language.

Keywords: 1-safe Petri Nets; GPenSIM; NuSMV; Model Checking.

1. Introduction

Nowadays automatic verification techniques, such as model checking, are widely used to verify the correctness of the systems. Model checking (Baier and Katoen, 2008) provides automatic and complete verification for the concurrent systems that are otherwise achieved by testing methods, albeit not-exhaustively. Furthermore, it is a combination of temporal logic with state space (Huth and Ryan, 2017). Different available model checkers verify the validity of properties expresses in temporal logic formalities (LTL or CTL) (Ben-Ari, 2008)(Clarke and Emerson, 1981). Amid several well-established model checkers, NuSMV (Cimatti et al., 2002), (Cimatti et al., 2000) is chosen as a state-of-art symbolic tool that provides both BDD-based and SAT-based model checking (Biere et al., 1999) which supports the verification of a large state space. NuSMV is a well-structured, flexible, open-source tool and accessible to participate in developing its functionalities.

Petri Nets are widely used to model and analyze parallel, synchronous, asynchronous and distributed discrete systems (Peterson, 1981). The behavioral and structural properties are depicted through its mathematical and graphical representations (Murata, 1989) (Reisig, 2013). PROD (Varpaaniemi et al., 1997) and LoLA (Wolf, 2018) are modeling tools for Petri Nets that support the verification of specifications expressed in both CTL and LTL temporal logic combined with different reduction techniques. PEP is another Petri Nets modeling tool that supports the verification of CTL properties utilizing SMV model checker and LTL properties through SPIN model checker(Grahlmann and Best, 1996). Szpyrka et al. (Szpyrka et al., 2014) presented two different algorithms that translate the low-level Petri Nets and colored Petri Nets to NuSMV input language for TINA (Berthomieu et al., 2004) and CPN (Jensen and Kristensen, 2009) Petri Nets modeling tools. Among others, in the literature are presented several approaches that have used NuSMV to verify high-level Petri Nets models (Penczek and Półrola, 2004)(Nakahori and Yamaguchi, 2017). However, GPenSIM is a

Petri Nets modeling tool that does not support formal verification.

The paper is structured as follows. The second section presents the basic details about the Petri Net models.The third section and fourth section give an introduction to the GPenSim and NuSMV, respectively. The following section presents an algorithm that converts the static safe Petri Nets model to NuSMV description language.

2. Definitions of Petri Nets

A marked Petri Net model (PM) is a weighted bipartite graph, $PM = (P, T, F, m_0)$ where P is a finite set of places, $P = \{p_1, p_2, ..., p_n\}$, T is a finite set of transitions, $T = \{t_1, t_2, ..., t_m\}$, F is set of weighted arcs that signify the relations between P and T, $F = (P \times T) \cup (T \times P)$ and m_0 represents the initial marking of the model. Pre-function of a place is the set of all input transitions denoted as $\bullet p_i = \{t_j \in T \mid (t_j, p_i) \in F\}$. Similarly, pre-function of a transition is the set of all input places denoted as $\bullet t_j = \{p_i \in P \mid (p_i, t_j) \in F\}$. The post-functions ($p\bullet$ and $t\bullet$) are defined by following the same reasoning.

A marking $m_k = (p_0, p_1, .., p_n)$ denotes the number of tokens for each specific place $p_i \in P$. The firing of transition t_j is associated with the movement of the tokens from input places to output places. It implies the change from one marking m_k to another marking m_{k+1} denoted as $m_k \xrightarrow{t_{jk}} m_{k+1}$.

A transition fires if it is enabled where each incoming place is assigned with at least as many tokens as the weight of the arc between the place and the transition. The enabled transitions of a marking m_k are defined as $T_{st} = (m_k \rightarrow)$. The set of markings $\mathcal{M} = (m_0, m_1, ..., m_r)$, differently named as the state space of PM, is a matrix with N^n rows.

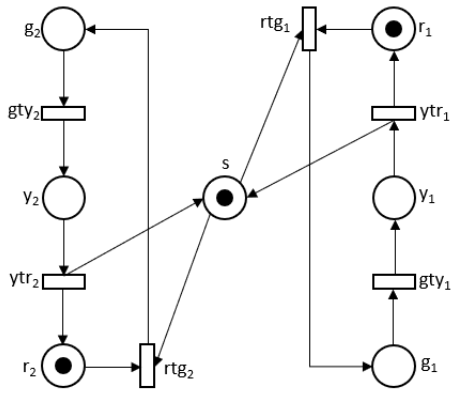

Figure 1. Marked Petri Nets model *PM*

An execution is the interleaving sequence of markings and transitions. The reachability graph defines the executions of the model, which is essential in studying the dynamic behaviors of the Petri Nets model. Reachability graph is a directed graph denoted as RG = (V, E) where V = $\mathcal{M}$ and E = $\{t \mid t \in T \text{ and } m_k \xrightarrow{t} m_{k+1} \text{ where } (m_k, m_{k+1}) \subseteq \mathcal{M} \text{ and } t \in (m_k \rightarrow)\}$. The Petri Nets model is safe if each place in the model has at most one token at each state, $m_k(p) \leq 1$. Fig. 1 illustrates a safe Petri Nets model of two synchronized traffic lights in an intersection.

3. GPenSim

GPenSIM (Davidrajuh, 2018) is used to model, simulate, and performance analysis of discrete event systems (Mutarraf et al., 2018). It is developed as a MATLAB toolbox where it provides integration with the other toolboxes of the MATLAB platform. The reasons for the acceptance of GPenSIM is its simplicity in learning and using and its flexibility to add newer functionality (Cameron et al., 2015). GPenSim is utilized to model and solve many industrial large-scale discrete problems (Skolud et al., 2016; Jyothi, 2012) where among others, it is used to reduce the Petri Nets model for model checking (Davidrajuh et al., 2020) (Davidrajuh and Roci, 2018).

The newest version of GPenSIM (v10) allows the designing and implementation of the modular Petri Nets. The modular development of Petri Nets increases the flexibility (ability to add or modify functionality) of the models and their comprehensibility. It reduces large Petri Net models' development time, as modelers can separately develop different modules simultaneously. Therefore, the modules can run faster while utilizing parallel computers (Davidrajuh, 2020).

A Petri Net model developed with GPenSIM consists of several files. The *main simulation file MSF* is the file that will be run directly by the MATLAB command. Except for the main simulation file, there will be one or more *Petri net definition files PDF* where is described the definition of a Petri net graph (static details). If the Petri Nets model is divided into many modules, then each of them defines separate PDFs files. While PDF has the static details, the main simulation file contains the dynamic information (such as initial tokens in places, firing times of transitions) of the Petri net. For further information, refer to (Davidrajuh, 2018).

Listing 1 illustrates the main simulation file *MFS* and Listing 2 depicts the Petri Nets definitions file *PDF* file of the exemplified Petri Nets model Fig. 1 in GPenSIM.

4. NuSMV

NuSMV automatically verifies whether a finite transition system satisfies the property under consideration. Therefore, NuSMV takes as input the properties expressed in temporal logical formulas and the model described in the input language of NuSMV. If the model

Listing 1. MSF file of the traffic light

```
function MSF

global PNs;
% build the static model
pns = pnstruct('semafor_pdf');
% combine PN with m0
pni = initialdynamics(pns, PNs.dyn);
```

Listing 2. PDF file of the traffic light

```
function [png] = semafor_pdf()

global PNs;
PNs.dyn.m0 = {'s', 1, 'r1', 1, 'r2', 1};

png.PN_name = 'Traffic Light example!';

png.set_of_Ps = {'s', 'g1', 'g2', 'r1', ...
'r2', 'y1', 'y2'};

png.set_of_Ts = {'rtg1', 'rtg2', 'ytr1',...
                  'ytr2','gty1', 'gty2'};

png.set_of_As = {...
   's', 'rtg1', 1, 'r1', 'rtg1', 1, ...
   'rtg1', 'g1', 1,...  %rtg1
   's', 'rtg2', 1, 'r2', 'rtg2', 1,...
   'rtg2', 'g2', 1,...  %rtg2
   'g1', 'gty1', 1, 'gty1', 'y1', 1, ...  %gty1
   'g2', 'gty2', 1, 'gty2', 'y2', 1, ...  %gty2
   'y1', 'ytr1', 1, 'ytr1', 's', 1, ...
   'ytr1', 'r1', 1,...  %ytr1
   'y2', 'ytr2', 1, 'ytr2', 's', 1,...  '
   ytr2', 'r2', 1,...  %ytr2
   };
```

satisfies the property specification, it generates a *true* statement, while the property is evaluated as false, it provides counterexamples exemplified by executable traces.

The finite transition system is a directed graph denoted as TS = (S, R, I, L) where S is a set of states, $R \subseteq s_i \times s_{i+1}$ is a transition relation between states, $I \subseteq S$ is a set of initial states and $L : S \rightarrow 2^{AP}$ is a labelling function over the atomic propositions AP. In NuSMV, the labeling function is not explicitly defined. The values of the variables define the labeling of each state. The reachability graph generated by the Petri Nets model represents a transition system.

NuSMV evaluates the specifications expressed in temporal logic LTL or CTL. LTL and CTL specifications are identified using the keywords LTLSPEC and CTLSPEC(SPEC), respectively. The specifications are described in the corresponded semantics, where each expression is a combination of different logical connectives. Another important feature is that SMV restricts the evaluation of the specification to a set of executions under the fairness assumptions.

The following briefly indicates some of the keywords used in the proposed algorithms. For further information, refer to (NuSMV). The cross-product of the values of the declared variables determines the domain of the state space. Accordingly, the states are fully connected. The input variables are specified using *VAR* declaration. The transition relations are determined while considering the values of variables in the current state and defining the values in the next state. *TRANS* and *ASSIGN* constraints are two different methods that restrict the transition relation between states by using proposition formulas. Note that SMV input language describes a specific model in different ways.

5. From static safe Petri Net to NuSMV

This section describes how a static safe marked Petri Nets model PM is expressed in NuSMV input language. Let consider the Petri Nets model in Fig. 1 as an example that will be converted to NuSmv input language. Fig. 2 depicts the reachability graph of exemplified net and Fig. 3 illustrates the graph of TS generated by the NuSMV shown in Listing 3. The primary purpose of this algorithm is to avoid the generation of the reachability graph in the GPenSim, so it directly converts the static model to the NuSMV input language. In NuSMV, the transition systems are represented either in BDD or SAT, supporting transition systems with much larger state space (J.R.Burch et al., 1992).

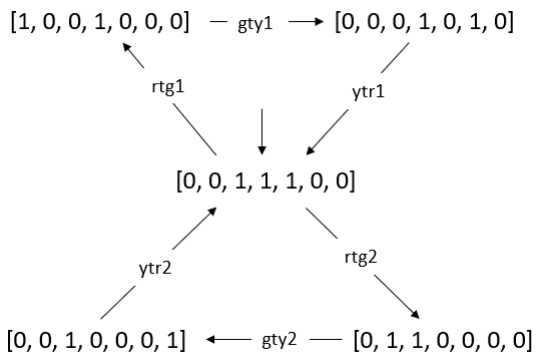

Figure 2. Reachability Graph *RG* of PM

In NuSMV, the declaration of the variables determines the state space of transition system TS and the transition relations are defined as a pair of the current state and the next state. Since the transitions of TS cannot be explicitly defined, each state is composed of places of the Petri Nets model and an enabled transi-

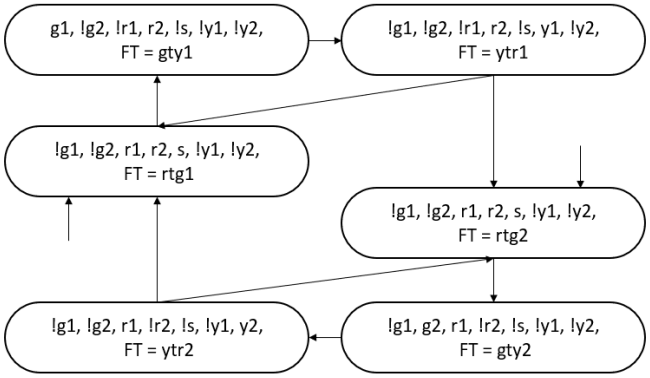

Figure 3. TS generated by NuSMV

tion, shown in Fig 3. For a safe Petri Nets model, a marking corresponds to a vector where each place is assigned with at most one token. Therefore, each place $p_i \in P$ is declared of type Boolean. Since the firing of the transitions in PM is associated with the states' changes, to each state s_k is added the enabled transition that will fire. Accordingly, the variable *FT* is declared as a scalar variable where the set of transitions are assigned as symbolic values, $FT = T \uplus \tau$. τ corresponds to the states without enabled transitions.

The initial values of the variables are assigned using the INIT constraint. Thus, the initial values are assigned as:

$$INIT := (\bigvee_{t \in (m_0 \rightarrow)} FT = t) \wedge \bigwedge_{p_i \in P} (p_i) = a \text{ if } m_0(p_i) = a$$

$$\text{where } p_i \in P \text{ and}$$

$$t = \{t_j \mid t_j \in T \text{ and } t \in (m_0 \rightarrow) \text{ and } m_0 \in \mathcal{M}\}$$

The transition relations are restricted if they satisfy the propositional formulas described in ASSIGN and TRANS constraints. The ASSIGN constraint exemplifies the change in the number of tokens for a specific place p if a certain transition FT fires. The firing of all transitions but the incoming and outgoing ones does not affect the number of tokens in the place p. Therefore, the number of tokens in a specific place *p* changes if either the incoming transitions or outgoing transitions are fired. In a safe net, when the incoming transition $t \in \bullet p$ fires, all the incoming places of transition $\bullet t$ should have one token in the current state s_c. Intuitively, all outgoing places of transition $t\bullet$ should not have any token, so the value of each outgoing place should be FALSE in the current state. Indeed, even the current value of the specified place p should be FALSE. Thus the next value for each place p is determined as:

Listing 3. NuSMV input language

```
MODULE main
VAR
    FT : {gty1, gty2, rtg1, rtg2, ytr1, ytr2};
    g1 :  boolean;
    g2 :  boolean;
    r1 :  boolean;
    r2 :  boolean;
    y1 :  boolean;
    y2 :  boolean;
    s :  boolean;
INIT
   (FT = rtg1 | FT= rtg2) &
   (g1 = 0 & g2 = 0 & r1 = 1 & r2 = 1
    & y1 = 0 & y2 = 0 & s = 1)
ASSIGN
    next(g1) := case
       FT = gty1 & !y1 & g1 :  FALSE;
       FT = rtg1 & r1 & s & !g1:  TRUE;
       TRUE : g1;
    esac;
    next(r1) := case
       FT = rtg1 & s & !g1 & r1 :  FALSE;
       FT = ytr1 & y1 & !s & !r1:  TRUE;
       TRUE : r1;
    esac;
    next(y1) := case
       FT = ytr1 & !r1 & !s & y1 :  FALSE;
       FT = gty1 & g1 & !y1:  TRUE;
       TRUE : y1;
    esac;
    next(s) := case
       FT = rtg1 & r1 & !g1 & s :  FALSE;
       FT = rtg2 & r2 & !g2 & s :  FALSE;
       FT = ytr1 & y1 & !r1 & !s:  TRUE;
       FT = ytr2 & y2 & !r2 & !s:  TRUE;
       TRUE : s;
    esac;
              .
              .
              .
TRANS
   next(FT) = gty1 & next(g1) & !next(y1) |
   next(FT) = gty2 & next(g2) & !next(y2) |
   next(FT) = rtg1 & next(r1) &
    next(s) & !next(g1) |
   next(FT) = rtg2 & next(r2) &
   next(s) & !next(g2) |
   next(FT) = ytr1 & next(y1) &
   !next(s) & !next(r1) |
   next(FT) = ytr2 & next(y2) &
   !next(s) & !next(r2)
```

$$next(p) := TRUE \; if \; FT = t \wedge \bigwedge_{p_i \in \bullet t} p_i \wedge \bigwedge_{p_i \in t\bullet} !p_i \wedge !p$$

$$where \; t = \{t_j \mid t_j \in T \; and \; t_j \in \bullet p\} \; and \; \bullet t \neq t\bullet$$

Similarly, the firing of the outgoing transition $t \in p\bullet$ changes the current value $p := TRUE$ to $next(p) := FALSE$, if all incoming places of transition are assigned with TRUE value and all outgoing transitions are assigned with a FALSE value. The formula for this instance is:

$$next(p) := FALSE \; if \; FT = t \wedge \bigwedge_{p_i \in \bullet t} p_i \wedge \bigwedge_{p_i \in t\bullet} !p_i \wedge p$$

$$where \; t = \{t_j \mid t_j \in T \; and \; t_j \in p\bullet\} \; and \; \bullet t \neq t\bullet$$

Listing 3 Lines [14-36] illustrate the assignment of place variables in the next state of one of the traffic lights and the synchronizer place. The assignment of the place variables of the second traffic light is similar to the first one.

On the other hand, the formulas described in the TRANS block define the next values of the transitions of PM. Once more, it is necessary to determine only the incoming and outgoing places of a specified transition t because other places do not affect the transition t. Listing 3 Lines [40 -46] exemplifies the TRANS constraint generated by the presented algorithm for the PM model. Each expression resembles the assignment of particular variables in the next state, where it includes the enabled transition that will fire and the place that are affected by its firing. In a safe PM, a transition $t \in T$ fires if all the incoming places have one token. Intuitively, neither of the outgoing places of t have token. Thus, it is formulated as follows:

$$E_j := (next(FT) = t) \wedge \bigwedge_{p_i \in \bullet t} next(p_i) \wedge \bigwedge_{p_i \in t\bullet} !next(p_i)$$

A Petri Net model PM is composed of m transitions, so TRANS declaration has at least m different expressions. Some of the nets have sink places and in such cases there is no enabled transition. Therefore, the FT variable is assigned with τ value, and the expression is a conjunction of all sink places such as:

$$E_m := (next(FT) = \tau) \wedge \bigwedge_{p_i \in P_s} next(p_i) if \; P_s \subset P$$

$$where \; P_s = \{p_i \mid p_i \in P \; and \; p_i\bullet = \emptyset\}$$

TRANS constraint is composed as a disjunction of the mentioned expressions such as:

$$TRANS := E_0 \vee E_1 \vee \ldots \vee E_m$$

The domain of the state space and fully connected transition relations are restricted to the ones that satisfy the propositional formulas mentioned above. The proposition formulas of both constraints consist of a transition system that emulates the executions of reachability graph.

Algorithm 1 (The complete code, 2021) converts the static safe Petri Nets to an SMV input language. For a given text that describes the Petri Nets model expressed in GPenSIM describing language and the specifications expressed in the temporal logic, it outputs the same results as NuSMV.

6. Conclusion

This paper presents an algorithm that converts a safe Petri Nets model described in GPenSim to NuSMV description language. The algorithm converts directly the static structure of the model without generating the state-space of the model. As a result, the Petri Nets model are automatically verified in GPenSim. Future Work: The algorithm is going to be adapted for high-level Petri Nets models and considering other Petri Nets tools.

References

Baier, C. and Katoen, J.-P. (2008). *Principles of Model Checking*. MIT PRESS.

Ben-Ari, M. (2008). Principles of the spin model checker. *Springer-Verlag London*.

Berthomieu, B., Ribet, P., and Vernadat, F. (2004). The tool tina – construction of abstract state spaces for petri nets and time petri nets. *International Journal of Production Research*, 42:2741–2756.

Biere, A., Cimatti, A., Clarke, E., and Zhu, Y. (1999). Symbolic model checking without bdds. *Cleaveland W.R. (eds) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 1999.*, 1579.

Cameron, A., Stumptner, M., Nandagopal, N., Mayer, W., and Mansell, T. (2015). Rule-based peer-to-peer framework for decentralised real-time service oriented architectures. *Science of Computer Programming*, 97:202 – 234.

Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A. (September 2002). Nusmv 2: An opensource tool for symbolic model checking. *Brinksma E., Larsen*

Algorithm 1 Static Petri Nets to SMV input language

```
1: procedure convert(PM, formula)
2:     file.write(MODULE main)
3:     file.write(VAR) – Declaration of the variables
4:     file.write(FT : tau)
5:     for each t ∈ T do
6:         file.write(t)
7:     end for
8:     for each p ∈ P do
9:         file.write(p : boolean)
10:    end for
11:    file.write(INIT) – INIT constraint
12:    for each t ∈ m0 → do
13:        file.write(FT = t)
14:    end for
15:    for each p ∈ P do
16:        if m0(p) == 1 then
17:            file.write(& p)
18:        else
19:            file.write(& !p)
20:        end if
21:    end for
22:    file.write(ASSIGN) – ASSIGN constraint
23:    for each p ∈ P do
24:        file.write(next(p) := case)
25:        for each t ∈ •p do
26:            file.write(FT = t)
27:            if •t ≠ t• then
28:                file.write(⋀ •t)
29:                file.write(⋀ !t•)
30:            end if
31:            file.write(p : FALSE)
32:        end for
33:        for each t ∈ •p do
34:            file.write(FT = t)
35:            if •t ≠ t• then
36:                file.write(⋀ •t)
37:                file.write(⋀ !t•)
38:            end if
39:            file.write(!p : TRUE)
40:        end for
41:    end for
42:    myfile.write(TRANS) – TRANS constraint
43:    for each t ∈ T do
44:        myfile.write(next(FT) = t)
45:        for each p ∈ •t do
46:            myfile.write(& next(p))
47:        end for
48:        for each p ∈ t• do
49:            myfile.write(& !next(p))
50:        end for
51:    end for
52:    myfile.write(formula)
53: end procedure
```


K.G. (eds) Computer Aided Verification. CAV 2002. Lecture Notes in Computer Science, 2402.

Cimatti, A., Clarke, E., Giunchiglia, F., and Roveri, M. (2000). Nusmv: A symbolic model checking. *International Journal on Software Tools for Technology Transfer*, 2:410–425.

Clarke, E. M. and Emerson, A. E. (1981). Design and synthesis of synchronization skeletons using branching time temporal logic. *Logic of Programs*, 31:52–73.

Davidrajuh, R. (2018). *Modeling discrete-event systems with gpensim: An introduction*. Springer.

Davidrajuh, R. (2020). Extracting petri modules from large and legacy petri net models. *IEEE Access*, 8:156539–156556.

Davidrajuh, R. and Roci, A. (2018). Performance of static slicing algorithms for Petri nets. *International Journal of Simulation–Systems, Science & Technology*, 20:15.1–15.7.

Davidrajuh, R., Skolud, B., and Krenczyk, D. (2020). Incorporating automatic model checking into gpensim. *Modelling and Performance Analysis of Cyclic Systems*, pages 175–187.

Grahlmann, B. and Best, E. (1996). Pep — more than a petri net tool. *Lecture Notes in Computer Science*, 1055.

Huth, M. and Ryan, M. (2017). *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, United Kindom.

Jensen, K. and Kristensen, L. (2009). *Coloured Petri Nets — Modeling and Validation of Concurrent Systems*. Springer-Verlag, Berlin.

J.R.Burch, E.M.Clarke, and McMillan, K. (June 1992). Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98:142–170.

Jyothi, S. D. (2012). Scheduling flexible manufacturing system using Petri-nets and genetic algorithm. *Department of Aerospace Engineering, Indian Institute of Space Science and Technology: Thiruvananthapuram, India*.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of IEEE*, 77:541–580.

Mutarraf, U., Barkaoui, K., Li, Z., Wu, N., and Qu, T. (2018). Transformation of business process model and notation models onto Petri nets and their analysis. *Advances in Mechanical Engineering*, 10(12):1 – 21.

Nakahori, K. and Yamaguchi, S. (2017). A support tool to design iot services with nusmv. *IEEE International Conference on Consumer Electronics (ICCE)*.

NuSMV. Nusmv: a new symbolic model checker. Technical report.

Penczek, W. and Półrola, A. (2004). Specification and model checking of temporal properties in time petri nets and timed automata. *Applications and Theory of Petri Nets 2004*, 3099.

Peterson, J. (1981). *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, NJ, USA.

Reisig, W. (2013). *Understanding Petri Nets: Modeling*

Techniques, Analysis Methods, Case Studies. Springer-Verlag Berlin Heidelberg, Germany.

Skolud, B., Krenczyk, D., and Davidrajuh, R. (2016). Solving repetitive production planning problems. an approach based on activity-oriented Petri nets. In *International Joint Conference SOCO'16-CISIS'16-ICEUTE'16*, pages 397–407. Springer.

Szpyrka, M., Biernacka, A., and Biernacki, J. (2014). Methods of translation of petri nets to nusmv language.

The complete code (2021). The converter from gpensim to nusmv. Technical report.

Varpaaniemi, K., Heljanko, K., and Lilius, J. (1997). Prod 3.2 an advanced tool for efficient reachability analysis. *Application and Theory of Petri Nets and Concurrency, PETRI NETS 2018*, 1254.

Wolf, K. (2018). Petri net model checking with lola 2. *Application and Theory of Petri Nets and Concurrency, PETRI NETS 2018*, 10877.