VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

MATHEMATICAL MODELING - CO2011

Symbolic and Algebraic Reasoning in Petri Nets

Version 0.0.1

HO CHI MINH CITY, 10/2025

1 Introduction

Petri nets are among the most fundamental and elegant mathematical models for describing **concurrent, distributed, and event-driven systems** [18, 15]. They provide a graphical and formal way to represent how *conditions* (places) and *events* (transitions) interact through the flow of *tokens*, enabling a precise analysis of system behavior. Since their introduction by Carl Adam Petri in the early 1960s [19], Petri nets have become a cornerstone of **formal methods** [3, 20], **system verification** [6, 20], **workflow modeling** [22, 21], and **biological network analysis** [4, 1].

From a theoretical perspective, Petri nets occupy a unique position at the intersection of **graph theory**, **discrete dynamical systems**, **linear algebra**, and **logic-based reasoning** [18, 15, 14, 3, 12]. Many core questions in computer science—such as *reachability*, *liveness*, and *deadlock-freedom*—can be formally stated and analyzed within the Petri net framework. However, these problems are also computationally challenging: even for small systems, the *state space explosion* caused by concurrency can lead to an exponential number of reachable markings [15, 3].

From a practical perspective, Petri nets offer a bridge between **modeling and computation** [18]. They are used to design and analyze manufacturing systems [16], communication protocols [9], concurrent programs [8], and even biological regulatory networks [13]. In these applications, the ability to compute and analyze the *reachability graph*—a directed graph representing all possible system states and their transitions—is crucial for verifying correctness and discovering hidden behaviors such as deadlocks or unsafe states [15, 22].

To address the scalability challenge, symbolic representations such as **Binary Decision Diagrams (BDDs)** [2] have been introduced to compactly encode large state spaces. Meanwhile, **Integer Linear Programming (ILP)** [7] provides a flexible optimization-based framework to reason about properties of these state spaces. They have been separately applied to efficiently analyze Petri nets [17, 12]. Combining these two techniques has the potential to enable both efficient state-space exploration and formal property checking.

In this assignment, students shall build a small-scale application integrating this idea:

1. Use BDDs to symbolically construct the of reachable markings of a given 1-safe Petri net (see [5] for more details about 1-safe Petri nets).
2. Apply ILP formulations to detect **deadlocks** [15, 17] in combination with the resulting BDD.

Through this exercise, students will not only deepen their understanding of the mathematical foundations of Petri nets but also gain practical experience with computational modeling techniques widely used in formal verification, optimization, and AI [7, 18, 3, 11, 10]. The goal is to foster both **theoretical insight** and **hands-on skills** in bridging abstract models with algorithmic analysis.

2 Tasks

1. **Reading Petri nets from PNML files:** Implement a parser that reads a 1-safe Petri net from a standard PNML file[1] and constructs the internal representation of places, transitions, and flow relations. The program should verify consistency (e.g., no missing arcs or nodes).

2. **Explicit computation of reachable markings:** Implement a basic breadth-first search (BFS) or depth-first search (DFS) algorithm to enumerate all reachable markings starting from the initial marking.

3. **Symbolic computation of reachable markings by using BDD:** Encode markings symbolically using Binary Decision Diagrams (BDDs). Construct the reachability set iteratively by symbolic image computation. Return a BDD representing the set of all reachable markings. Report the total number of reachable markings and compare performance with the explicit approach (time and memory).

4. **Deadlock detection by using ILP and BDD:** Combine ILP formulation and the BDD obtained in Task 3 to detect a deadlock if it exists. More specifically, output one deadlock if found, otherwise report none. Note that a *dead marking* is a marking where no transition is enabled, whereas a *deadlock* is a dead marking that is reachable from the initial marking [17]. Report the running time on some example models.

5. **Optimization over reachable markings (20%):** Given a linear objective function

$$\textbf{maximize } c^\top M, \quad M \in \mathrm{Reach}(M_0),$$

   where $\mathrm{Reach}(M_0)$ denotes the set of markings reachable from the initial marking M_0 and c assigns integer weights to places, determine a reachable marking if it exists that optimizes the objective function. If there is no such a marking, report none. Report the running time on some example models.

6. **Report quality:** The report must be clear, concise, and logically structured. It should describe:

   - Theoretical background for each method (explicit, symbolic, ILP-based)

[1] https://www.pnml.org/

- Implementation design and data structures
- Experimental results and performance discussion
- Challenges encountered and possible improvements (may receive **bonuses** if the findings are significant)

Figures and tables are encouraged to illustrate results.

3 Instructions

All of the aspects related to this assignment will be included in some questions of the final exam of the course. Therefore, each group (3-4 students) should work together so that all members understand all of the aspects of the assignment. The group leader should organize the group so that this requirement will be met. During the work, if students have any question about the assignment, please post it on the forum for the assignment on BK-eLearning or write an email to their instructor. Part of the in-class time will also be allocated for students to discuss this assignment directly with their instructor.

3.1 Requests

- **Implementation language:** Students may use Python, C++, or Java, but BDD and ILP libraries (e.g., PyEDA, CUDD, Gurobi, PuLP) should be clearly referenced in the report.
- The code must run with provided input files and reproduce the reported results.
- It can assume that the testing models are already 1-safe. Students do not need to check the 1-safe property.
- The objective of this assigment is not to push the state-of-the-art, hence students should not care much about the performance. The testing models should be of small or medium size, and **be the same for all the tasks**.

3.2 Submission

Each group must prepare a ZIP file containing:

- Source code with README instructions
- A concise report ($\leq$ 15 pages, PDF)
- Test PNML files (if applicable)

This ZIP file must be named by the format "*Assignment-CO2011-CSE251-{All member ID numbers separated by hyphens}.zip*" and submitted to the assignment section on the site of the course on BK-eLearning. Deadline for submission is 23h00, December 05, 2025 (HCMC time (GMT+7)). Noting that for each group, **only the leader should make the submission.**

4 Evaluation and cheating judgement

This assignment will make up 20% for the final evaluation of CSE learners, and few questions in the final exam could be based on it.

4.1 Evaluation

The grading follows the weighting scheme below:

Task no.	Component	Weight
1	PNML parsing	5%
2	Explicit reachability	5%
3	BDD-based reachability	40%
4	ILP + BDD deadlock detection	20%
5	Reachable optimization	20%
6	Report quality	10%

4.2 Cheating judgement

Each group must develop its own code and report. While discussion of general ideas is allowed, direct sharing of code, ILP formulations, or experimental data between groups is strictly prohibited. A group will be considered as cheating if:

- There is an abnormal similarity among the reports. In this case, ALL submissions that are similar are considered as cheating. Therefore, the students of a group have to defend the work of their own group.
- They do not understand some parts written in their reports. Students can consult from any source, but make sure that they understand the meaning of everything they will have written. This rule also applies to source code.

If the work is found as cheating, students will be judged according to the university's regulations.

Positive advice: You are encouraged to discuss high-level modeling ideas with your peers and instructors. However, ensure that your implementation, experiments, and report are genuinely your own work. Understanding and explaining what you have built is more important than producing perfect results.

5 Changelogs

References

[1] Mary Ann Blätke, Monika Heiner, and Wolfgang Marwan. Biomodel engineering with Petri nets. In *Algebraic and Discrete Mathematical Methods for Modern Biology*, pages 141–192. Elsevier, 2015.

[2] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[3] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: 10ˆ20 states and beyond. *Inf. Comput.*, 98(2):142–170, 1992.

[4] Claudine Chaouiya, Aurélien Naldi, Elisabeth Remy, and Denis Thieffry. Petri net representation of multi-valued logical regulatory graphs. *Nat. Comput.*, 10(2):727–750, 2011.

[5] Allan Cheng, Javier Esparza, and Jens Palsberg. Complexity results for 1-safe nets. *Theor. Comput. Sci.*, 147(1&2):117–136, 1995.

[6] Javier Esparza and Mogens Nielsen. Decidability issues for Petri nets - a survey. *Bull. EATCS*, 52:244–262, 1994.

[7] Jack E. Graver. On the foundations of linear and integer linear programming I. *Math. Program.*, 9(1):207–226, 1975.

[8] Matthias Heizmann, Dominik Klumpp, Lars Nitzke, and Frank Schüssele. Petrification: Software model checking for programs with dynamic thread management. In *VMCAI*, pages 3–25. Springer, 2024.

[9] Lasse Jacobsen, Morten Jacobsen, Mikael H. Møller, and Jirí Srba. Verification of timed-arc Petri nets. In *SOFSEM*, pages 46–72. Springer, 2011.

[10] Chihyun Jung and Tae-Eog Lee. An efficient mixed integer programming model based on timed Petri nets for diverse complex cluster tool scheduling problems. *IEEE Trans. Semicond. Manuf.*, 25(2):186–199, May 2012.

[11] HENRY KAUTZ and JOACHIM P. WALSER. Integer optimization models of ai planning problems. *The Knowledge Engineering Review*, 15(1):101–117, 2000.

[12] Victor Khomenko and Maciej Koutny. Verification of bounded Petri nets using integer programming. *Formal Methods Syst. Des.*, 30(2):143–176, 2007.

[13] Xuefei Lin, Xiao Chang, Yizheng Zhang, Zhanyu Gao, and Xu Chi. Automatic construction of Petri net models for computational simulations of molecular interaction network. *npj Syst. Biol. Appl.*, 10(1), November 2024.

[14] José Meseguer and Ugo Montanari. Petri nets are monoids. *Inf. Comput.*, 88(2):105–155, 1990.

[15] Tadao Murata. Petri nets: Properties, analysis and applications. *Proc. IEEE*, 77(4):541–580, 1989.

[16] Hafiz Zahid Nabi and Tauseef Aized. Performance evaluation of a carousel configured multiple products flexible manufacturing system using Petri net. *Oper. Manage. Res.*, 13(1–2):109–129, March 2020.

[17] Enric Pastor, Jordi Cortadella, and Oriol Roig. Symbolic analysis of bounded Petri nets. *IEEE Trans. Computers*, 50(5):432–448, 2001.

[18] James Lyle Peterson. *Petri net theory and the modeling of systems.* Prentice Hall PTR, 1981.

[19] Carl Petri. *Kommunikation mit Automaten.* PhD thesis, TU Darmstadt, 1962.

[20] Wolfgang Reisig. *Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies.* Springer, 2013.

[21] Wil M. P. van der Aalst. *Process Mining - Data Science in Action, Second Edition.* Springer, 2016.

[22] Wil M. P. van der Aalst, Kees M. van Hee, Arthur H. M. ter Hofstede, Natalia Sidorova, H. M. W. Verbeek, Marc Voorhoeve, and Moe Thandar Wynn. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects Comput.*, 23(3):333–363, 2011.