VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY

HO CHI MINH UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

MATHEMATICAL MODELING (CO2011)

Assignment

Symbolic and Algebraic Reasoning in Petri Nets

Instructor: Dr. Van-Giang Trinh

Ho Chi Minh City, 12/2025

Members and Workload

STT	Full name	Students ID	Task	Class
1	Lưu Nguyễn Thanh Bình	2310927	Implement the 1st task and 2nd task: *Reading Petri nets from PNML files* and *Explicit computation of reachable markings*	L02
2	Trần Hoàng Bá Huy	2311249	Implement the 3rd task: *Symbolic computation of reachable markings by using BDD*	L01
3	Xà Gia Khánh	2311543	Implement the 4th task: *Deadlock detection by using ILP and BDD*	L04
4	Đoàn Duy Khanh	2311488	Implement the 5th task: *Optimization over reachable markings*	L04
5	Dương Khôi Nguyên	2312333	Conceptualize the structure; writing, editing, formatting the final report and incorporating all co-author feedback and data inputs	L04

Table of Contents

1 Background Theory

1.1 Petri Net

According to Murata (1989), a Petri net is defined as a tuple $N = (P, T, F, M_0)$, where P is a finite set of places, T is a finite set of transitions, and F represents the flow relation, allowing us to visualize the causal dependencies between events. The state of a Petri net is defined by its **marking** M, which is a vector of non-negative integers of size $|P|$. The dynamic behavior is governed by the firing rule: a transition t is enabled if $M(p) \geq 1$ for all input places $p \in t$. When t fires, the new marking M' is given by the state equation:

$$M' = M + C \cdot u$$

where C is the incidence matrix and u is the firing vector.

This assignment specifically focuses on **1-safe Petri nets**. A net is 1-safe if for every reachable marking $M \in Reach(M_0)$, and for every place $p \in P$, the number of tokens is at most 1:

$$\forall M \in Reach(M_0), \forall p \in P : M(p) \in \{0, 1\}$$

This property allows the global state to be encoded as a Boolean vector, facilitating the use of symbolic logic.

1.2 Binary Decision Diagrams

Binary Decision Diagrams (BDDs) are a canonical directed acyclic graph (DAG) data structure used to represent Boolean functions Bryant (1986). A Boolean function $f : \{0, 1\}^n \to \{0, 1\}$ is represented as a rooted graph where non-terminal nodes are labeled with variables x_i and terminal nodes represent values 0 (False) and 1 (True).

The power of BDDs lies in their **canonical form** (Reduced Ordered BDD - ROBDD). For a fixed variable ordering, any Boolean function has a unique BDD representation. This allows for constant-time equivalence checking and efficient logical operations with complexity proportional to the graph size rather than the exponential state space.

In the context of this assignment, we utilize BDDs to solve the state space explosion problem inherent in concurrent systems. Since the Petri net is **1-safe** (Cheng, Esparza,

& Palsberg, 1995), the number of tokens in any place p_i is strictly bounded to $\{0, 1\}$. Thus, a global marking M can be encoded as a Boolean vector $(x_1, x_2, \ldots, x_{|P|})$, where $x_i = 1$ if and only if place p_i contains a token. The set of all reachable markings is represented by a characteristic function χ_{Reach}.

1.3 Integer Linear Programming

Integer Linear Programming (ILP) is a mathematical optimization technique where the objective is to minimize or maximize a linear function subject to linear equality and inequality constraints, with the variables restricted to integers. The standard form is:

$$\text{Maximize } c^T x \quad \text{subject to } Ax \leq b,\ x \in \mathbb{Z}^n$$

In Petri nets, ILP is often used for structural analysis because the fundamental state equation ($M = M_0 + A \cdot \sigma$, where A is the incidence matrix) is a linear system.

This assignment requires a hybrid approach, combining the "exact" state space from BDDs with the "structural" equations of ILP. For **Deadlock Detection (Task 4)**, we formulate linear constraints representing "dead" states (where all transitions are disabled) and intersect this condition with the BDD-computed reachability set to identify genuine reachable deadlocks. For **Optimization (Task 5)**, the objective to maximize $c^\top M$ is addressed by directing the search for the optimal marking within the valid solution space encoded by the BDD, ensuring the result is both optimal and reachable within the 1-safe Petri net.

2 Python Implementation with Example

The team proposes this Petri net as an example for our Python implementation:

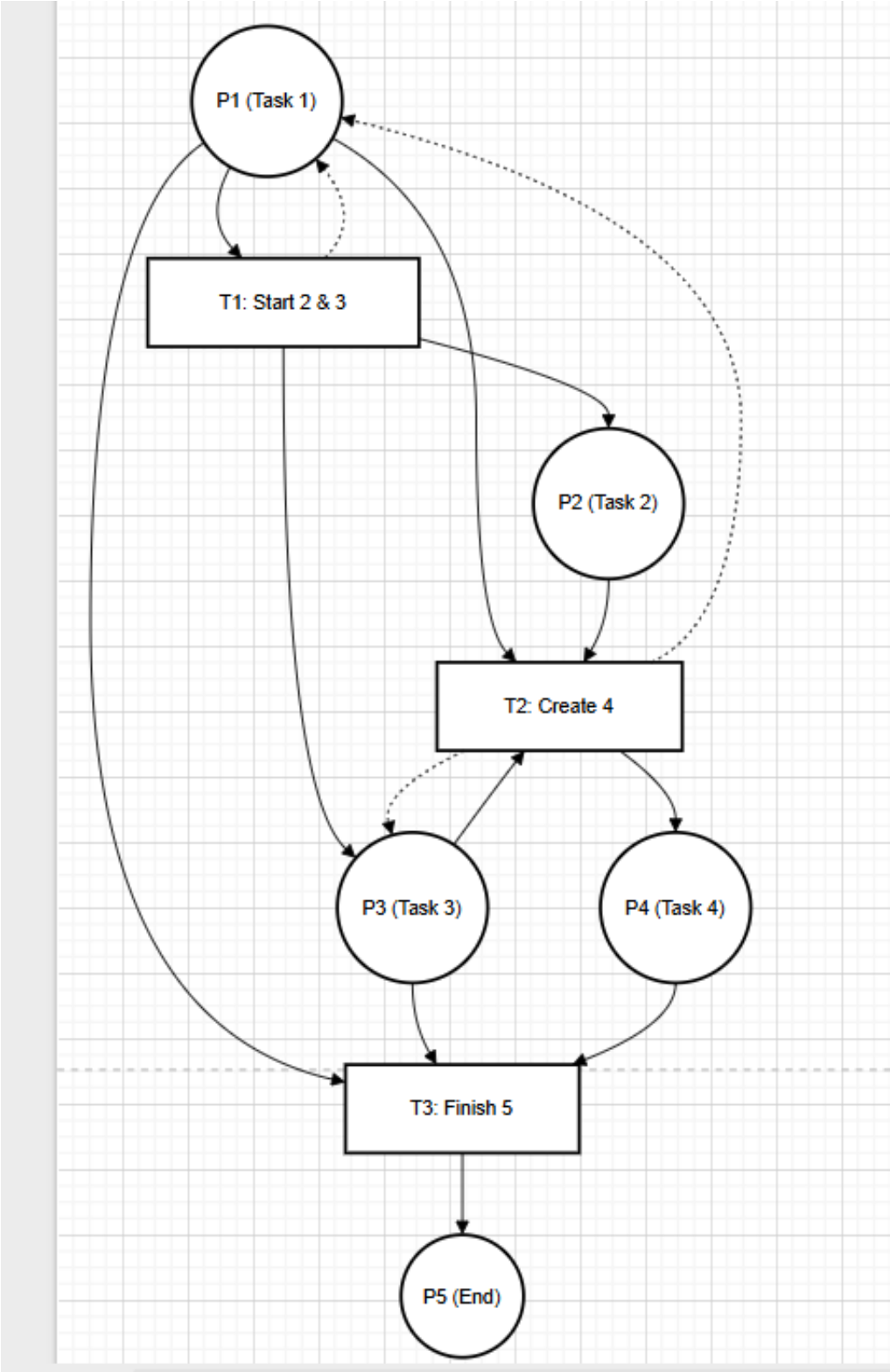

Figure 1: Example of Petri net

2.1 Task 1 - Reading Petri nets from PNML files

The Petri net class is defined in Python as follows:

```
class PetriNet:
    # Identifier for places and transitions
    self.place_ids = place_ids
    self.trans_ids = trans_ids
    # Labels for places and transititons
    self.place_names = place_names
    self.trans_names = trans_names
    self.I = I # Input matrix of size T x P
    self.O = O # Output matrix same size of input matrix
    self.M0 = M0 # Initial marking
    # Class method is defined elsewhere in the source code
```

In order to read a Petri net from a `.pnml` file, we define a function that can:

1. **XML Parsing**: Uses `xml.etree.ElementTree` to parse the file. It includes a `get_tag` helper function to ignoring namespaces commonly found in PNML files.

2. **Raw Data Extraction:** Iterates to find:

   - `place`: Extracts ID, Name, and `initialMarking` (initial token count).
   - `transition`: Extracts ID and Name.
   - `arc:` Extracts source and target to define connections.

   then mapping IDs to indices. This is essential for populating the `numpy` matrices.

3. **Matrix Construction:** Initializes matrices with zeros. The matrices are designed with dimensions Transitions $\times$ Places ($T \times P$). Then the function iterates through the list of arcs:

   - If the arc goes from **Place** $\rightarrow$ **Transition**: Adds to the **Input Matrix** (I).
   - If the arc goes from **Transition** $\rightarrow$ **Place**: Adds to the **Output Matrix** (O).

Finally, the function returns an instance of the `PetriNet` class with fully processed data.

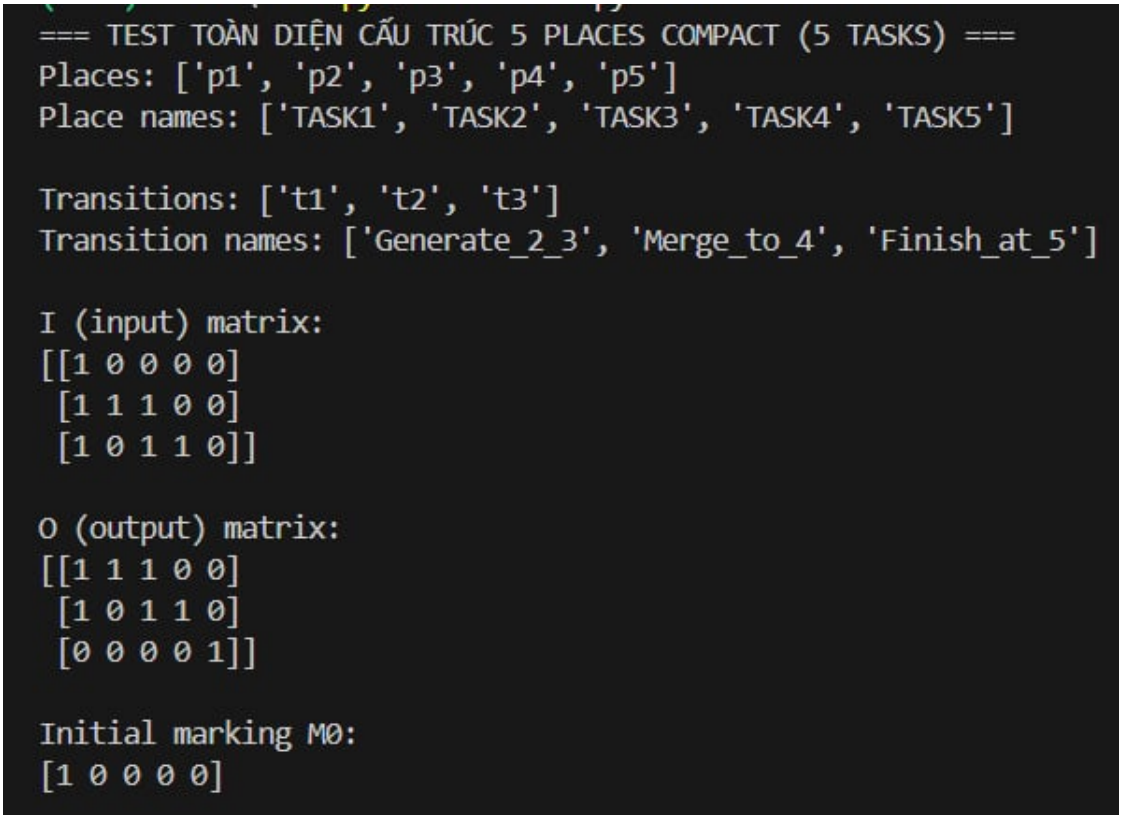

Figure 2: Reading Petri net from `.pnml` file

2.2 Task 2 - Explicit computation of reachable markings

The team chooses the BFS to for the task. For convenience, we also implement a DFS function. But since the assignment requires implementing either BFS or DFS only, the other approach is not shown here. The result of BFS is shown in Figure 3.

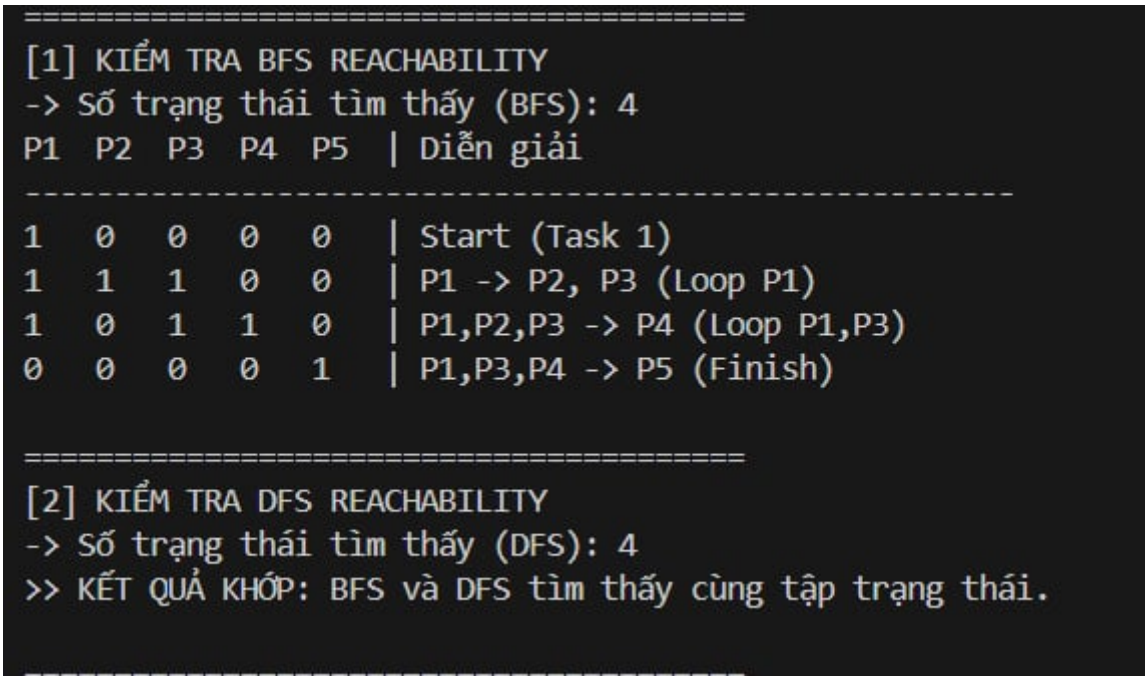

Figure 3: BFS approach result

2.3 Task 3 - Symbolic computation of reachable markings by using BDD

The process begins by validating the Petri net structure, checking for zero places, and normalizing the dimensions of input/output matrices to handle inconsistencies. The initial marking M_0 is then encoded into a boolean formula, where the presence of a token represents a `true` state ($X[i]$) and its absence represents `false`. The core logic involves constructing a global transition relation. For each transition, a **"guard"** condition is created to verify that input tokens exist and target output places are empty, strictly enforcing the 1-safe constraint. Simultaneously, an **"effect"** formula defines the next state (X_p) by handling token consumption and production, using XNOR operations to maintain the state of unaffected places. All valid transitions are combined into a single relation using the OR operator.

Finally, the algorithm employs **fixed-point iteration** to compute the full state space. It repeatedly applies the transition relation, smoothes out current state variables, and renames next-state variables until the reachable set converges. The function returns a BDD representing the set of all reachable markings.

2.4 Performance Comparison

2.5 Task 4 - Deadlock detection by using ILP and BDD

2.6 Task 5 - Optimization over reachable markings

References

Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, *35*(8), 677–691.

Cheng, A., Esparza, J., & Palsberg, J. (1995). Complexity results for 1-safe nets. *Theoretical Computer Science*, *147*(1–2), 117–136.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, *77*(4), 541–580.