

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC BÁCH KHOA  
KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH



# BÁO CÁO

## BÀI TẬP LỚN MÔ HÌNH HÓA TOÁN HỌC

---

# PETRI NETWORKS

---

GVHD: Nguyễn An Khương  
Huỳnh Tường Nguyên  
SV thực hiện: Nguyễn Đức An – 2010102 – L01  
Trần Phúc Anh – 2010133 – L01  
Huỳnh Tấn Lộc – 2010391 – L01  
Nguyễn Quang Khánh – 2010330 – L01  
Lưu Quốc Hưng Thịnh – 2010651 – L01

Tp. Hồ Chí Minh, Tháng 12/2021

## Mục lục

<b>1</b>	<b>Mở đầu:</b>	<b>2</b>
<b>2</b>	<b>Lý thuyết:</b>	<b>2</b>
2.1	Backgrounds . . . . .	2
2.1.1	Giới thiệu về Petri net . . . . .	2
2.1.2	Transition system . . . . .	2
2.1.3	Petri net . . . . .	3
2.1.4	Một số vai trò của Petri net . . . . .	6
2.2	Behaviors . . . . .	8
2.2.1	Firing . . . . .	8
2.2.2	Labeled Petri net . . . . .	9
2.2.3	Reachability Graph . . . . .	9
2.2.4	Trả lời câu hỏi trong đề mục 5.3 . . . . .	9
2.3	Cấu trúc của Petri Net . . . . .	11
2.3.1	Causality, Concurrency và Synchronization . . . . .	11
2.3.2	Hệ quả của Concurrency . . . . .	12
2.4	Tổng kết về Petri net . . . . .	12
2.5	Các vấn đề liên quan đến mô hình hóa Petri net . . . . .	14
2.5.1	Các vấn đề cơ bản khi mô hình hóa với Petri net . . . . .	14
2.5.2	Động lực của Petri net . . . . .	15
2.5.3	Kết hợp hai Petri net . . . . .	15
2.5.4	Trả lời các câu hỏi trong đề mục 5.6 . . . . .	16
<b>3</b>	<b>Bài tập:</b>	<b>16</b>
3.1	Problem 1 . . . . .	16
3.2	Problem 2 . . . . .	17
3.3	Problem 3 . . . . .	17
3.4	Problem 4 . . . . .	18
3.5	Problem 5 . . . . .	20
3.6	Problem 6 . . . . .	21
3.7	Problem 7 . . . . .	24
3.7.1	Hướng dẫn sử dụng chương trình: . . . . .	24
3.7.2	Setting cho chương trình . . . . .	26
3.7.3	Chương trình chính . . . . .	27
3.7.4	Class TableOption . . . . .	28
3.7.5	Class TableSettingToken . . . . .	29
3.7.6	Class Problem1Solving và class Problem2Solving . . . . .	31
3.7.7	Class Problem3Solving . . . . .	34
<b>4</b>	<b>Kết luận:</b>	<b>38</b>
<b>5</b>	<b>Tài liệu tham khảo:</b>	<b>38</b>

## 1 Mở đầu:

**Petri Nets** được giới thiệu lần đầu tiên bởi C.A.Petri vào đầu những năm 1960 như là một công cụ toán học cho việc mô hình hóa hệ thống. Petri Nets đã được sử dụng thành công trong lĩnh vực xây dựng các mô hình đồng bộ, song song, giao thức truyền thông, đánh giá hiệu suất và hệ thống phát hiện lỗi

Để hiểu rõ hơn về Petri Nets cũng như những ứng dụng của nó trong việc xử lý hệ thống thông tin, nhóm tác giả đã quyết định chọn nghiên cứu đề tài này. Nội dung của bài nghiên cứu sẽ bao gồm khái niệm, cấu trúc, những đặc điểm và những ứng dụng thực tế của Petri Nets.

## 2 Lý thuyết:

### 2.1 Backgrounds

#### 2.1.1 Giới thiệu về Petri net

Petri net là một dạng biểu diễn đồ thị (đồ thị phân đôi bao gồm các trạng thái là các **places** và **transition**) để mô tả và phân tích các quá trình phát sinh đồng thời (**concurrent processes**) trong các hệ thống nhiều thành phần (**distributed systems**)

#### 2.1.2 Transition system

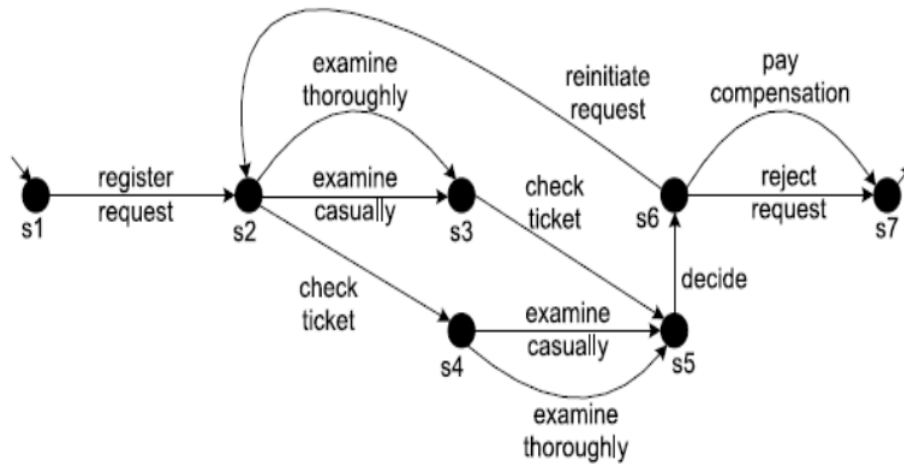
##### 1. Định nghĩa

Transition system là một bộ ba triplet  $TS = (S, A, T)$  với  $S$  là tập hợp của các **states**,  $A$  là tập hợp của các **activities**,  $T \subseteq S \times A \times S$  là tập của các **transitions**.

- $S^{start} \subseteq S$  được gọi là initial states.
- $S^{end} \subseteq S$  được gọi là final states.

Các Transition system trong trường hợp này được xem như là Finite-State Machine hoặc Finite Automaton (FA)

- Transition system là một mô hình để quyết định những hoạt động nào cần được thực hiện và thứ tự thực hiện của nó. Các hoạt động có thể được thực hiện tuần tự, các hoạt động có thể là tùy chọn hoặc đồng thời, và có thể thực hiện lặp lại cùng một hoạt động.
- Một transition bắt đầu ở một trong những trạng thái ban đầu. Bất kỳ đường đi nào trong đồ thị bắt đầu ở trạng thái như vậy thì sẽ có tương ứng một trình tự thực thi có thể xảy ra.
  - Đường đi gọi là **terminates successfully** khi nó kết thúc ở một trong những final states.
  - Đường đi gọi là **deadlocks** nếu nó kết thúc ở một trong những đỉnh non-final states mà không còn đường đi nữa.



**A transition system having one initial state and one final state**

Hình 1: Transition system

**NOTES on using transition system:**

- Bất kỳ mô hình quy trình nào có ngữ nghĩa thực thi đều có thể được ánh xạ vào một Transition system. Do đó, nhiều khái niệm được định nghĩa cho các hệ thống chuyển tiếp có thể dễ dàng được dịch sang các ngôn ngữ cấp cao hơn như Petri net ...
- Transition system đơn giản nhưng có vấn đề trong việc thể hiện các công việc đồng thời xảy ra vì cần phải sử dụng quá nhiều states để lưu trữ như trường hợp "state explosion". Nhưng Petri Net có thể được sử dụng nhỏ gọn và hiệu quả hơn.
- Ví dụ khi ta có n công việc song song độc lập với nhau, các công việc này đều cần phải thực hiện nhưng có thể thực hiện ở bất cứ thứ tự nào. Trong trường hợp này có  $n!$  trình tự thực thi. Transition system cần  $2^n$  states và  $n \times 2^{n-1}$  transitions để lưu trữ trong khi đó Petri net chỉ cần n states and n transitions để model n parallel activities.

2. Multiset

Cho miền  $D = \{x_1, x_2, \dots, x_k\}$ , phép ánh xạ  $X : D \rightarrow N$  định nghĩa một **multiset**:

$$M = \underbrace{\{x_1, \dots, x_1\}}_{m_1 \text{ times}}, \dots, \underbrace{\{x_k, \dots, x_k\}}_{m_k \text{ times}}.$$

$$M = \{x_1^{m_1}, x_2^{m_2}, \dots, x_k^{m_k}\},$$

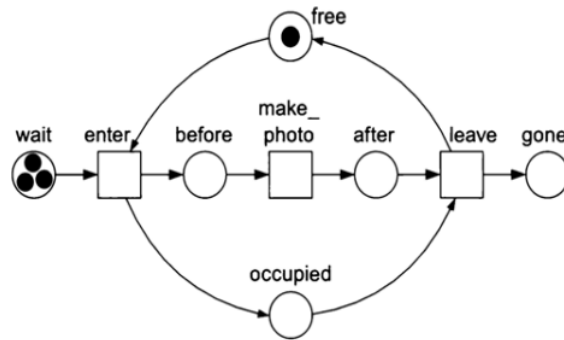
M còn được định nghĩa với một list  $[m_1, m_2, \dots, m_k]$ , với  $m_i$  là số lần xuất hiện của phần tử  $x_i$  trong multiset.

**2.1.3 Petri net**

1. Định nghĩa

Petri net được biểu diễn bởi một bộ ba  $N = (P, T, F)$ , P là tập hợp hữu hạn các places, T là tập hợp hữu hạn các transitions thỏa mãn  $P \cap T = \emptyset$  và  $F \subseteq (P \times T) \cup (T \times P)$  là tập các mối quan hệ giữa các node P và T, còn được gọi là flow relation.

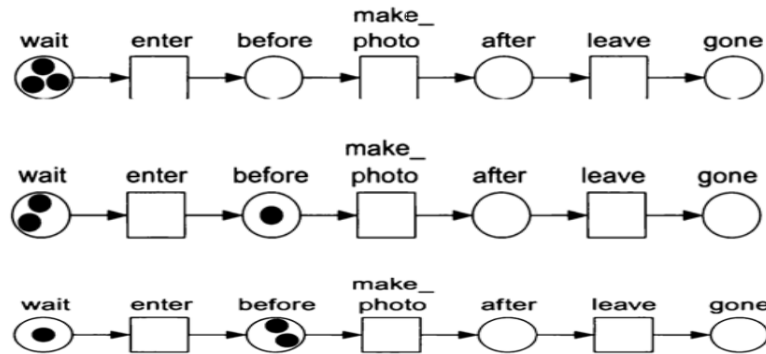
- Tokens** là một transition node đặc biệt, được kí hiệu bằng kí hiệu blackdot •. Nó đại diện cho một elements cụ thể thực tế. Places có chứa tokens, còn transition thì không.
- Một transition được gọi là **enabled** nếu **tất cả** các input tới nó đều phải chứa tokens. Một cái **Enabled transition** có thể **fire**, tiêu thị ít nhất tokens ở mỗi input và tạo ra ít nhất một tokens ở output của nó.
- Mỗi **Marking** là một tập hợp các phân phối tokens ở các places tại một thời điểm. Nó là phép ánh xạ từ tập  $P \rightarrow N$ , biến mỗi  $p \in P$  thành  $m(p)$ .
- Một petri net được kí hiệu bởi cặp  $(N, M)$  với  $N = (P, T, F)$  và  $M$  là một multiset, đại diện cho một Marking của petri net.



Hình 2: Petri net minh họa cho quá trình hoạt động của một X-ray machine

#### Trả lời Example 5.3 (Trang 17)

- $P = \{\text{wait, before, after, free, occupied, gone}\}$
- $M = [3, 0, 0, 1, 0, 0]$
- $F = \{(\text{wait, enter}), (\text{enter, before}), (\text{before, make\_photo}), (\text{make\_photo, after}), (\text{after, leave}), (\text{leave, free}), (\text{free, enter}), (\text{enter, occupied}), (\text{occupied, leave}), (\text{leave, gone})\}$



The first three markings in a process of the X-ray machine

- [top, **transition enter** not fired];
- [middle, **transition enter** fired];
- [down, **transition enter** has fired again]

Figure 5.4: Three different markings on a Petri net, modeling of a process of an X-ray machine

#### Trả lời Example 5.4 (Trang 18)

- $P = \{\text{wait, before, after, gone}\}$

- $T = \{\text{enter, make\_photo, leave}\}$
- $M_1 = [3, 0, 0, 0]$  ,  $M_2 = [2, 1, 0, 0]$  ,  $M_3 = [1, 2, 0, 0]$

2. Đầu vào và đầu ra của các trạng thái (Input is place, output is transition)

- Node x được gọi là input node của y khi có một directed arc từ x đến y kí hiệu  $(x, y)$
- Với  $x \in P \cup T$ , ta kí hiệu  $\bullet x = \{y | (y, x) \in F\}$  - gọi là **preset** x,  
 $x\bullet = \{y | (x, y) \in F\}$  - gọi là **postset** x.

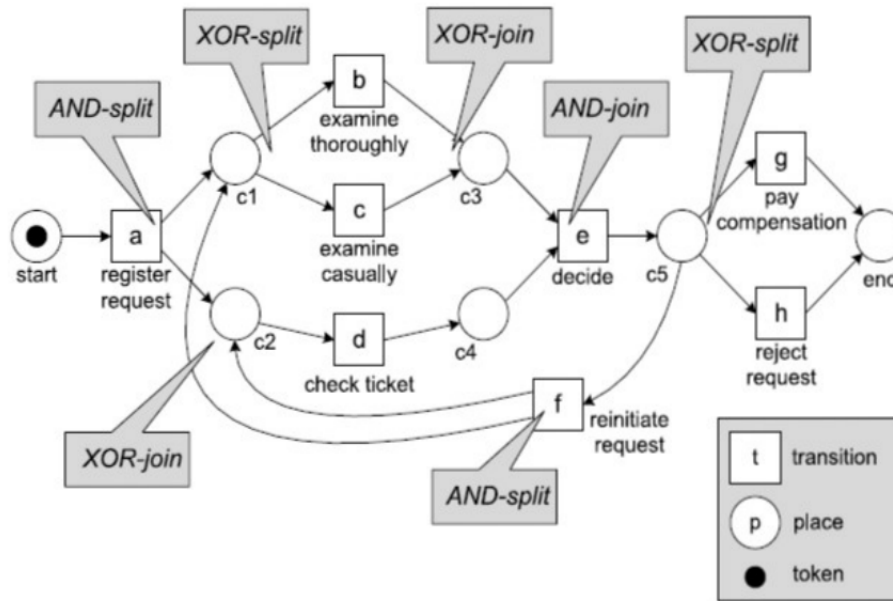


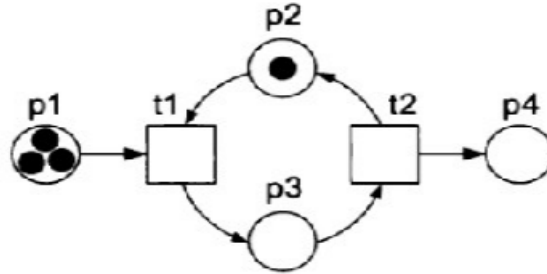
Figure 5.5: A marked Petri net with one initial token

Example:  $\bullet c1 = \{a, f\}$  và  $\bullet c5 = \{g, h\}$

3. *Enable transition và marking changes*

**Trả lời Practice 5.1 (Trang 21)**

Consider the Petri net in figure below.



**A Petri net showing transitions t1 and t2.**

**Figure 5.6: A simple Petri net, with only two transitions**

1. Define the net formally as a triple  $(P, T, F)$ .
2. List presets and postsets for each transition.
3. Determine the marking of this net.
4. Are the transitions t1 and t2 enabled in this net.

- (a)
- $P = \{p1, p2, p3, p4\}$
  - $T = \{t1, t2\}$
  - $F = \{(p1, t1), (t1, p3), (p3, t2), (t2, p2), (p2, t1), (t2, p4)\}$
- (b)
- $t1 = \{p1, p2\}$       $t1 \bullet = \{p3\}$
  - $t2 = \{p3\}$       $t2 \bullet = \{p2, p4\}$
- (c)  $M_1 = [3, 1, 0, 0]$
- (d)
- t1 enabled vì cả hai input của t1 là p2, p1 đều chứa ít nhất 1 tokens.
  - t2 không enabled vì input p3 của t2 không chứa tokens.

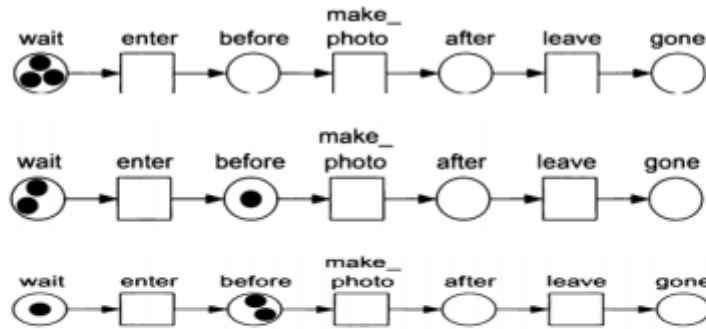
#### 2.1.4 Một số vai trò của Petri net

- Business process
- Information system
- Task ordering principles

Petri net bao gồm hai loại arcs:

1.  $R_I \subseteq P \times T$  bao gồm tất cả các arcs kết nối transition và input places.
2.  $R_O \subseteq T \times P$  bao gồm tất cả các arcs kết nối transition và output places.

## Practical Problem 1 (Trang 25)



The first three markings in a process of the X-ray machine

- a) [top, **transition enter** not fired]; b) [middle, **transition enter** fired]; and  
c) [down, **transition enter** has fired again]

Figure 5.8: A Petri net model of a business process of an X-ray machine

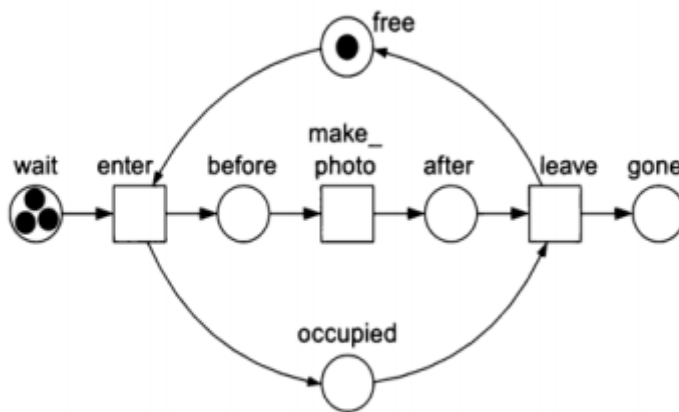
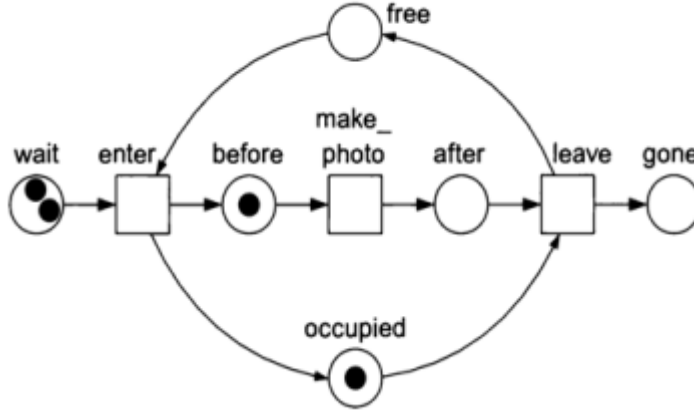


Figure 5.9: An improved Petri net for the business process of an X-ray machine

- $R_I = \{(wait, enter), (before, make\_photo), (after, leave)\}$
  - $R_O = \{(enter, before), (make\_photo, after), (leave, gone)\}$
  - $F = \{(wait, enter), (before, make\_photo), (after, leave), (enter, before), (make\_photo, after), (leave, gone)\}$
- before không thể chứa nhiều hơn 1 tokens vì lúc này trong free chỉ có 1 tokens.  
Ban đầu, free và wait đều có chứa ít nhất 1 tokens nên enter được enable. Sau khi firing thì ở wait và free mỗi place tiêu thụ 1 token, đồng thời sinh ra ở before và occupied mỗi place một token  
 $\Rightarrow$  Lúc này free không còn token và enter bị disabled, không thể firing nữa  $\rightarrow$  before không thể chứa nhiều hơn 1 tokens. Xem hình minh họa.





3. enter không thể fire again vì lúc này input free không có tokens  $\rightarrow$  enter bị disabled.

## 2.2 Behaviors

### 2.2.1 Firing

1. *Firing rule*:

Giả sử ta có một *marked Petri net*  $(N, M) \in \mathcal{N}$  với  $N = (P, T, F)$  &  $M \in \mathcal{M}$ . Khi đó:

- Một *transition* gọi là *enabled* nếu mọi *place* đầu vào của *transition* ấy đều có ít nhất một *token*.
- Ký hiệu: Nếu *transition*  $t \in T$  đang *enabled* tại trạng thái mà lưới có đánh dấu là  $M$ , ta ký hiệu là:  $(N, M)[t]$
- Firing rule*:  $\alpha[t]\beta \subseteq \mathcal{N} \times T \times \mathcal{N}$  là quan hệ bé nhất thỏa mãn:

$$(N, M)[t] \Rightarrow \underbrace{(N, M)[t]}_{\alpha} \underbrace{(N, (M \setminus \bullet t) \uplus t \bullet)}_{\beta}$$

với mọi  $(N, M) \in \mathcal{N}$  và  $t \in T$ .

**Nhận xét:**

- Chỉ thực hiện *firing* một *transition* khi nó đang *enabled*.
- Khi một *enabled transition* thực hiện *fires*, nó sẽ làm cho *Petri net* chuyển trạng thái đánh dấu  $M$  sang  $M_1$ . Khi đó, ứng với mỗi *place* đầu vào của *transition* sẽ bị mất đi một *token* và ứng với mỗi *place* đầu ra của *transition* sẽ tăng thêm một *token*.

2. *Firing sequence*:

Giả sử ta có một *marked Petri net*  $(N, M_0) \in \mathcal{N}$  với  $N = (P, T, F)$  và  $M_0 \in \mathcal{M}$ . Khi đó:

- Một chuỗi  $\sigma \in T^*$  được gọi là *firing sequence* khi và chỉ khi với  $n \in \mathbb{N}$ , tồn tại các *markings*  $M_1, M_2, \dots, M_n$  và các *transition*  $t_1, t_2, \dots, t_n$  thỏa

$$\sigma = (t_1, t_2, \dots, t_n) \in T^*$$

và với mọi  $i \leq n$ , thì  $(N, M_i)[t_{i+1}]$  và  $(N, M_i)[t_{i+1}](N, M_{i+1})$ .

- Một *marking*  $M$  được gọi là *reachable* từ một *initial marking*  $M_0$  khi tồn tại một chuỗi các *enabled transition* có thể *fires* từ  $M_0$  đến  $M$ .
- Ta ký hiệu tập hợp các *markings* được gọi là *reachable* từ *initial marking*  $M_0$  là:  $[N, M_0]$ .
- Ta ký hiệu  $(P, T, F, M_0)$  thể hiện cho một *Petri net*  $(P, T, F)$  có trạng thái đánh dấu ban đầu (*initial marking*) là  $M_0$ .

Từ một *initial marking*  $M_0$  ta có thể thực hiện *fires* các *enable transition* để phân phối lại các *token* và đến được một *marking* khác. Nếu khi nào đến một *marking*  $M$  mà trong *Petri net* không còn tồn tại bất cứ *enabled transition* nào nữa thì  $M$  được gọi là *terminal marking*.

**Lưu ý:** Một *Petri net* có thể không tồn tại *terminal marking*.

### 2.2.2 Labeled Petri net

Trong một *Petri net*, *transition* có thể được định danh bởi một chữ cái, nhưng chúng ta có thể thực hiện gán nhãn cho các *transition* để miêu tả các hoạt động một cách rõ ràng.

Ta định nghĩa một *Labeled Petri net* là bộ năm  $N = (P, T, F, A, l)$  trong đó bộ ba  $(P, T, F)$  đã được định nghĩa ở mục trước và:

- $A \subseteq \mathcal{A}$  là tập các nhãn của các hoạt động và ánh xạ  $l \in \{L : T \rightarrow A\}$  có chức năng gán các nhãn cho các *transition*.
- Ta sử dụng nhãn  $\tau$  để biểu diễn cho cái hoạt động gọi là *invisible*. Một *transition* mà được gán nhãn  $\tau$  được gọi là *unobservable*, *silent* hoặc *invisible*.

### 2.2.3 Reachability Graph

Như chúng ta đã biết, từ một *initial marking*  $M_0$  ta có thể tìm ra các *reachable markings*. Từ đó, ta có một số vấn đề cần phải giải quyết:

- Có tổng cộng bao nhiêu *reachable marking*.
- Nhận dạng được một *marking* có *reachable* hay không.
- Có tồn tại *terminal marking* không.

Đối với những *Petri net* đơn giản thì những vấn đề trên ta có thể giải quyết một cách trực giác. Nhưng đối với những *Petri net* quá phức tạp thì ta không thể dùng trực giác để giải quyết được. Do đó, để giải quyết vấn đề trên ta cần đến *reachability graph*.

Ý tưởng chính của *reachability graph* là ta sẽ miêu tả lại các hành vi của *Petri net* trên một *transition system* có các *state* là các *reachabled marking*.

**Định nghĩa:** Giả sử ta có một *marked Petri net*  $(N, M_0)$  với  $N = (P, T, F, A, l)$ . Khi đó, ta có thể định nghĩa định một *transition system*  $TS = (S, A_1, TR)$  với:

- $S = [N, M_0], S^{start} = \{M_0\}$
- $A_1 = A$
- $TR = \{(M, M_1) \in S \times S \mid \exists t \in T : (N, M)[t](N, M_1)\}$  trường hợp có nhãn:  
 $TR = \{(M, l(t), M_1) \in S \times A \times S \mid \exists t \in T : (N, M)[t](N, M_1)\}$

ta nói  $TS$  chính là *reachability graph* của *marked Petri net*  $(N, M_0)$ .

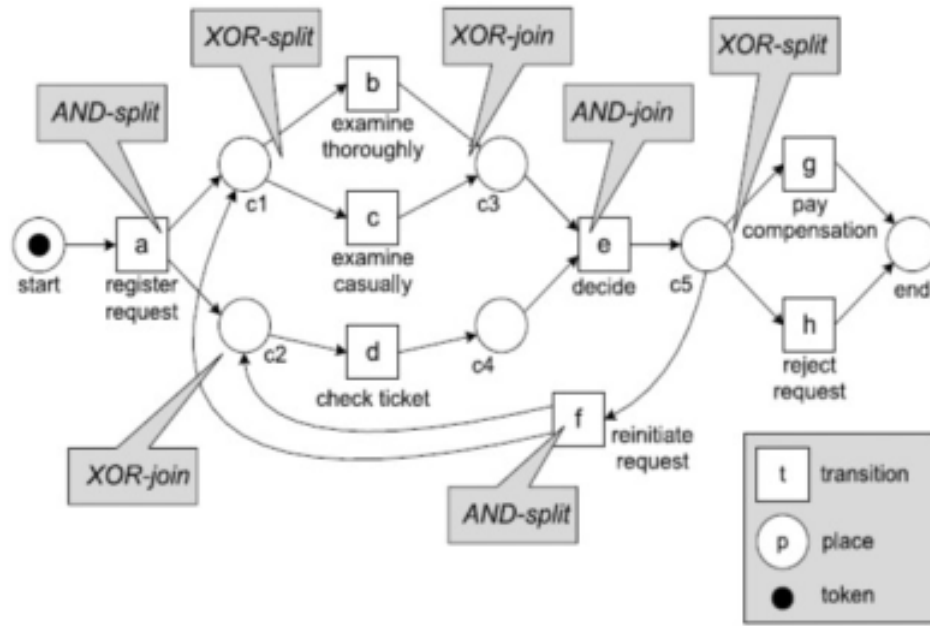
**Lưu ý:** Không nhất thiết phải định nghĩa tập  $S^{end}$  cho *transition system* thể hiện cho *reachability graph*.

### 2.2.4 Trả lời câu hỏi trong đề mục 5.3

#### 1. QUESTION 5.1: (Trang 32)

- Câu 1: Để tính được số *marking* có thể đạt đến từ *marked Petri net* ban đầu, ta có thể thực hiện vẽ *reachability graph* để xác định.
- Câu 2: Để xác định được một *marking* có thể đạt đến được hay không, ta phải tìm được một *firing sequence* từ *initial marking* đến *marking* hiện tại cần xác định.
- Câu 3: Một *marked Petri net* có thể không tồn tại *terminal marking*, do đó ta phải dựa trên *reachability graph* để xác định.

#### 2. PRACTICE 5.2: (Trang 36)

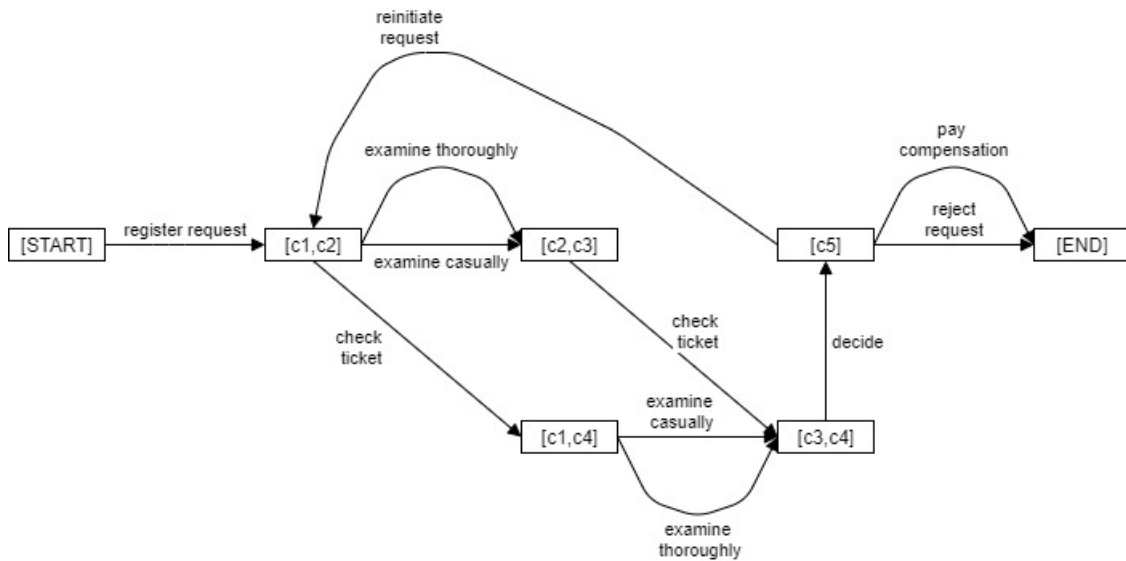


Hình 3: A sample marked Petri net

Ta sẽ xây dựng một *transition system*  $TS = (S, A_1, TR)$  để thể hiện cho *reachability graph* của Petri net trên.

- $S = \{[start], [c1, c2], [c2, c3], [c1, c4], [c3, c4], [c5], [end]\}$   
 $S^{start} = \{[start]\}$   
 $S^{end} = \{[end]\}$
- $A_1 = \{\text{register request, examine thoroughly, examine casually, check ticket, decide, reinitiate request, pay compensation, reject request}\}$

Khi đó, ta có mô hình hoàn chỉnh là:



Hình 4: Mô hình hoàn chỉnh của transition system

## 2.3 Cấu trúc của Petri Net

Places và transitions trong một Petri net được nối nhau bởi các đường *arcs*. Việc kết nối các nodes quyết định hành vi của mạng. Nói cách khác, cách mà các transitions được kết nối sẽ quyết định thứ tự chúng được *fire*. Đầu tiên, ta ôn lại một số khái niệm, quy luật và các mối liên hệ trong một Petri net  $N = (P, T, F)$

1. Khi một transition  $t$  được fire, số các tokens được sinh ra trong place  $p$  sẽ bằng với số tokens ban đầu **trừ đi** số tokens được tiêu thụ và cộng với số tokens được tạo ra.
2. Tổng số tokens trong mạng sẽ thay đổi nếu số lượng places đầu vào của transition  $t$  **không** bằng số lượng places đầu ra của nó. Theo đó, việc *firing* của một transition có thể làm tăng hoặc làm giảm tổng số tokens.
3. Khi có nhiều transitions được *enabled* đồng thời, nó **không** thể xác định transition nào sẽ được fire. Trường hợp này gọi là Lựa chọn không xác định **nondeterministic choice**. Mặc dù chúng ta không thể biết được transition nào sẽ được fire nhưng chúng ta biết được chắc chắn một trong số chúng sẽ fire.

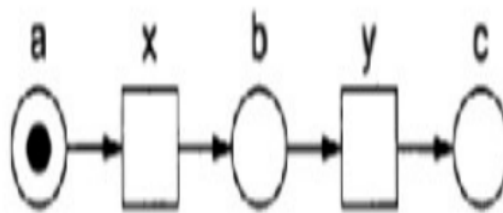
### 2.3.1 Causality, Concurrency và Synchronization

#### MỘT SỐ KHÁI NIỆM

- **Causality** được hiểu là mối quan hệ giữa 2 sự kiện phải được diễn ra theo một trình tự nào đó trong hệ thống. Trong một **Petri net**, chúng ta có thể biểu diễn mối quan hệ này bằng hai transition và một place trung gian
- **Concurrency** (đồng bộ, song song) là một đặc tính quan trọng của hệ thống thông tin. Trong một hệ thống đồng bộ, nhiều sự kiện có thể **diễn ra đồng thời**. Ví dụ, nhiều người dùng có thể sử dụng một hệ thống thông tin như là database cùng một thời điểm.

Nhìn chung thì một transition sẽ đại diện cho một sự kiện. Giả sử chúng ta có 3 sự kiện là  $x, y$  và  $z$ . Đầu tiên chúng ta xem xét một cấu trúc mạng diễn hình để chỉ ra rằng:

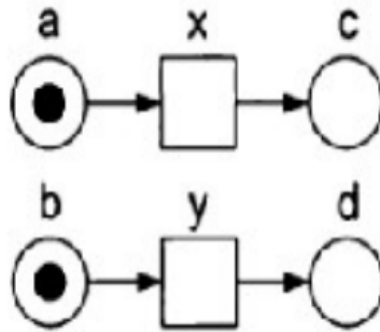
1. Sự kiện  $y$  xảy ra sau sự kiện  $x$
2. Sự kiện  $x$  và  $y$  diễn ra đồng thời.
3. Sự kiện  $z$  diễn ra sau sự kiện  $x$  và  $y$



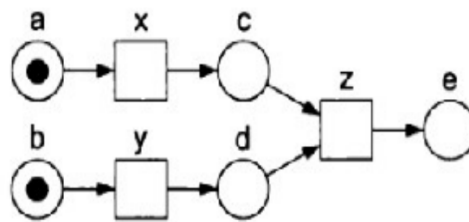
**Hình 5:** Causality trong một mạng  $N$ . Transition  $y$  chỉ có thể được fire sau khi  $x$  đã được fire

- Ở hình thứ nhất, đây là trường hợp biểu diễn mối quan hệ lẫn nhau **causality**
- Ở hình thứ hai, transition  $x$  và  $y$  có thể được fire một cách độc lập lẫn nhau. Ta nói, không có một mối quan hệ nào (**no causal relationship**) giữa việc firing  $x$  và  $y$ . Với mô hình network như ở hình thứ hai, ta có thể xây dựng mô hình đồng bộ.

Ta có thể xây dựng ô hình đồng bộ trong *Petrinet* khi một transition có ít nhất hai input places. Ở hình sau, transition  $z$  có hai place đầu vào và chỉ có thể fire khi transitions  $x$  và  $y$  đã được fire. Trong bất cứ quy trình nào, giả sử transition  $x$  và  $y$  biểu diễn hai bước thực hiện đồng thời. Transition  $z$  sẽ biểu diễn bước mà chỉ có thể thực thi chỉ khi có kết quả của hai bước trước.



Hình 6: Concurrency trong một mạng  $N$ . Transitions  $x$  và  $y$  xuất hiện đồng thời



Hình 7: Synchronization. Transition  $z$  xảy ra sau khi  $x$  và  $y$  đồng thời xảy ra

### 2.3.2 Hệ quả của Concurrency

**Problem :** Liệu chúng ta có thể ước lượng tính đồng bộ cho một quy trình hay một *Petrinet* hay không?

Câu trả lời là có, bằng cách sử dụng transition system. Một transition system được biểu diễn bởi bộ ba

$$TS = (S, A, T)$$

Trong đó,

- $S$  là tập hợp các trạng thái states.
- $A$  là tập hợp các hoạt động
- $T$  là tập hợp các transition

**Concurrency thông qua Transition system:** nếu một mô hình chứa nhiều concurrency hoặc có nhiều tokens cùng nằm trong một place. Khi đó, transition system  $TS$  sẽ lớn hơn rất nhiều lần Petri net  $N = (P, T, F)$

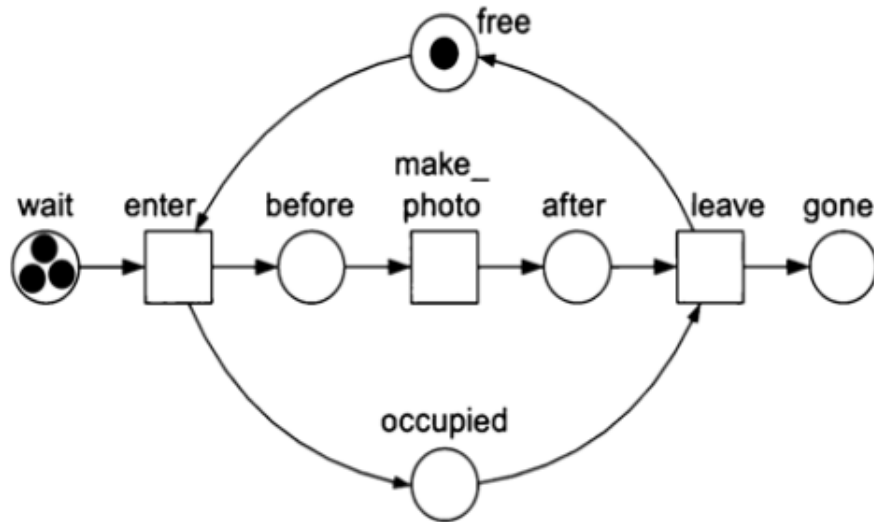
Nhìn chung, một marked Petri net  $(N, M_0)$  có thể có vô hạn các states có thể đến được.

## 2.4 Tổng kết về Petri net

**Problem 5.1 :** Giải thích các thuật ngữ sau trong Petri nets và cho ví dụ mỗi khái niệm.

1. "enabled transition": Một transition được gọi là enabled nếu mọi place đầu vào của transition ấy đều có ít nhất một token
2. "firing of a transition": khi một enabled transition thực hiện fire thì ứng với mỗi place đầu vào của transition sẽ bị mất đi một token và ứng với mỗi place đầu ra của transition sẽ tăng thêm một token
3. "reachable marking": một marking  $M$  được gọi là reachable từ một initial marking  $M_0$  khi tồn tại một chuỗi các enabled transition có thể fire từ  $M_0$  đến  $M$

- "terminal marking": từ một initial marking  $M_0$  phân phối lại các token đến các marking khác, nếu khi nào đến một marking  $M$  mà trong Petri net không còn tồn tại bất cứ enabled transition nào nữa thì  $M$  được gọi là terminal marking. Một Petri net có thể **không** tồn tại terminal marking
- "nondeterministic choice": trường hợp này xảy ra khi có nhiều transition được enabled đồng thời, ta không thể xác định được transition nào sẽ được fire tiếp theo. Tuy nhiên, ta biết được một trong số những transition đó sẽ được fire.

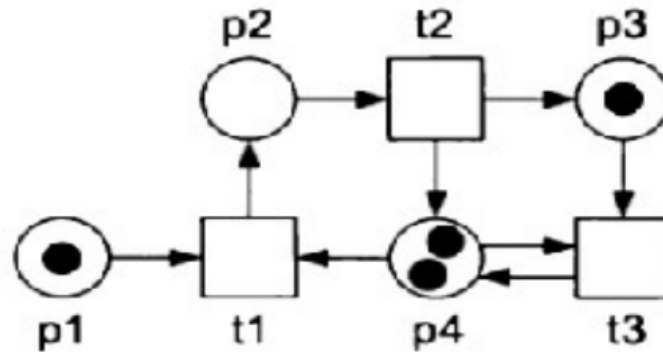


Theo hình minh họa trên:

- transition *enter* được enabled bởi vì 2 place đầu vào là *wait* và *free* đã có đủ token.
- transition *enter* do đã được enabled nên sẽ có thể fire và chuyển 1 token sang place *before*
- Ở initial marking ta có  $M_0 = [wait^3, free]$ . Sau khi transition *enter* được fire thì ta có một **reachable marking** mới là  $M = [wait^2, before, occupied]$
- Từ hình trên, ta có thể thấy được quá trình xử lý sẽ được dừng lại khi không còn token trong transition *wait*. Từ đó, ta có thể suy ra được **terminal marking** của Petri net này là  $M = [free, gone^3]$

**Problem 5.2** : Xét Petri net theo mô hình sau

- Petri net trên được biểu diễn dưới dạng công thức như sau:  $N = (P, T, F, M_0)$   
Trong đó:
  - $P = \{p1, p2, p3, p4\}$
  - $T = \{t1, t2, t3\}$
  - $F = \{(p1, t1), (t1, p2), (p2, t2), (t2, p3), (t2, p4), (p3, t3), (t3, p4), (p4, t1), (p4, t3)\}$
  - $M_0 = [p1, p4^2, p3]$
- Preset* và *Postset* của mỗi transition:
  - $t1$  có *preset* là  $p1$  và  $p4$ , có *postset* là  $p2$
  - $t2$  có *preset* là  $p2$ , có *postset*  $p3, p4$
  - $t3$  có *preset* là  $p3, p4$ , có *postset* là  $p4$
- Ở marking  $M_0$  thì transition  $t1$  và  $t3$  được enabled vì input place của 2 transition này đều có đủ token.



4. Reachable marking:

- $M = [p2, p4, p3]$
- $M = [p2, p4]$
- $M = [p4^2, p3]$
- $M = [p4^2]$ , marking này cũng chính là **terminal marking** của Petri net trên

5. Petri net trên có **nondeterministic choice** vì ở  $M_0$  ta có 2 transition  $t1$  và  $t3$  đều được enabled nên ta không thể quyết định xem transition nào sẽ *fire*.

## 2.5 Các vấn đề liên quan đến mô hình hóa Petri net

### 2.5.1 Các vấn đề cơ bản khi mô hình hóa với Petri net

Đối với một *token* thì chúng có thể biểu diễn cho:

- Một đối tượng vật lí như sản phẩm của quá trình sản xuất, thuốc hoặc con người,...
- Một đối tượng chứa thông tin như tin nhắn, tín hiệu,...
- Đối tượng là tập hợp các đối tượng như xe tải chở hàng,...
- Một trạng thái chỉ dẫn.
- Một điều kiện.

Đối với *place*, với vai trò là nơi chứa các *token*, chúng là phải thể hiện cho các sự vật có liên kết với *token*:

- Một bộ đệm như kho hàng, ngăn xếp hay thùng gửi thư,...
- Một phương tiện giao tiếp như đường dây điện thoại, người trung gian, mạng lưới liên lạc,...
- Một vị trí địa lí như một nơi nào đó trong nhà kho, trong văn phòng, trong bệnh viện,...
- Một trạng thái khả thi hay không.

Đối với *transition* là thành phần gây ra sự biến đổi trong *Petri net*, chúng sẽ thể hiện cho:

- Một sự kiện như bắt đầu một ca phẫu thuật, chuyển đổi màu đèn giao thông, sự thay đổi mùa,...
- Một sự biến đổi của đối tượng như sửa chữa các sản phẩm, cập nhật lại cơ sở dữ liệu,...
- Vận chuyển một đối tượng như vận chuyển hàng hóa, gửi một tệp tin,...

### 2.5.2 Động lực của Petri net

Một số tính chất chung ta cần quan tâm:

1. Một *marked Petri net*  $(N, M_0)$  được gọi là **k-bounded** nếu không có *place* nào có nhiều hơn **k** tokens:

$$\forall p \in P, \forall M \in [N, M_0] : M(p) \leq k$$

2. Một *marked Petri net* được gọi là **safe** khi và chỉ khi nó là **1-bounded**.
3. Một *marked Petri net* được gọi là **bounded** khi và chỉ khi tồn tại số tự nhiên **k** sao cho nó là một **k-bounded**.

Để miêu tả về động lực của một *Petri net* thì chúng ta hãy nói đến những định nghĩa sau:

1. Một *marked Petri net* được gọi là **deadlock free** nếu với mỗi *reachable marking* có ít nhất một *transition* được *enabled*:

$$\forall M \in [N, M_0], \exists t \in T : (N, M)[t].$$

2. Một *transition t* trong một *marked Petri net* được gọi là **live** nếu từ mọi *reachable marking* nó có thể làm cho *transition* ấy trở nên *enable*:

$$\forall M \in [N, M_0], \exists M_1 \in [N, M] : (N, M_1)[t].$$

3. Một *marked Petri net* được gọi là **live** nếu mọi *transition* của nó đều **live**.

**Lưu ý:** Một *Petri net* **dealock free** không nhất thiết cũng **live**.

### 2.5.3 Kết hợp hai Petri net

Trên thực tế, trong các quy trình thường bao gồm các tác nhân khác nhau và chúng sẽ thực hiện các hoạt động khác nhau nên khi mô hình hóa bằng *Petri net* thì các tác nhân trên sẽ có các *place*, *token* và *transition* khác biệt nhau. Tuy nhiên động lực thật sự của toàn bộ hệ thống (bao gồm tất cả các tác nhân trong quy trình) được tạo ra từ sự tương tác giữa chúng: các *place* có thể có cùng *token* hoặc có thể thực hiện các hoạt động có liên quan nhau dẫn đến có cùng các *transition*. Do đó, để mô hình hóa cả hệ thống lớn từ các mô hình *Petri net* của các quy trình riêng biệt thì ta không chỉ đơn giản là hợp lại các mạng lưới riêng lẻ, mà chúng ta phải thực hiện chồng chất các mô hình riêng lẻ nhỏ hơn với nhau. Công việc đó gọi là *superimposition*.

**Định nghĩa:** Giả sử rằng một hệ thống lớn chỉ có hai loại tác nhân, *Petri net* của chúng lần lượt là  $N_1 = (P_1, T_1, F_1, M_0)$  và  $N_2 = (P_2, T_2, F_2, M_0)$  với điều kiện  $P_1, P_2$  là các tập rời nhau nhưng có cùng *initial marking*  $M_0$ . Khi đó:

- Ta định nghĩa toán tử *superimposition*,  $\oplus : T_1 \times T_2 \longrightarrow T$  với  $T = T_1 \cup T_2$  như sau:
  - Nếu  $\bullet t_1 = \bullet t_2$  thì  $(t_1, t_2) \mapsto \oplus(t_1, t_2) = t_1$ .
  - Ngược lại,  $(t_1, t_2) \mapsto \oplus(t_1, t_2) = \{t_1, t_2\} \subseteq T$

Chúng ta có thể gộp hai *transition* cùng tên của hai mạng lưới thành một khi trộn hai lưới lại với nhau nếu sự kiện xảy ra của chúng trên cùng các *token* có cùng tính chất.

- Một *Petri net* được kết hợp lại từ  $N_1, N_2$  là:

$$N = N_1 \oplus N_2 = (P_1 \cup P_2, T, F_1 \cup F_2, M_0)$$



#### 2.5.4 Trả lời các câu hỏi trong đề mục 5.6

##### 1. QUESTION 5.3 (Trang 47)

Để xây dựng một *Petri net* từ một hệ thống lớn mà không làm mất các thông tin cần thiết và hữu dụng thì ta nên chia hệ thống lớn ấy thành các quy trình nhỏ hơn của các nhân tố trong hệ thống ấy và thực hiện mô hình hóa các quy trình ấy thành các *Petri net*, sau đó thực hiện trộn các *Petri net* nhỏ ấy lại thành một *Petri net* lớn thể hiện được động lực đúng của hệ thống.

### 3 Bài tập:

#### 3.1 Problem 1

(a) Gọi  $P$  là tập chứa các states,  $T$  là tập chứa các transition của  $N_S$

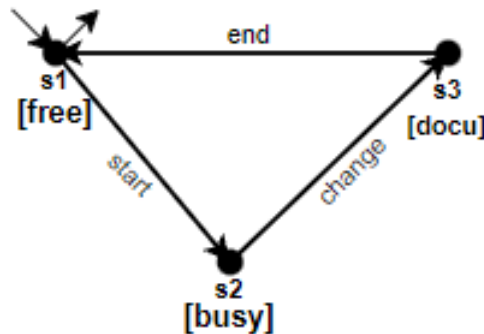
- $P = \{\text{free, docu, busy}\}$
- $T = \{\text{start, end, change}\}$

(b) Trả lời:

i. **Each place cannot contain more than one token in any marking** Ta có *initial marking* của Petri net  $N_S$  là  $M_0 = [\text{free}]$

- Đầu tiên, **start** được enabled  $\rightarrow$  firing start, giả sử  $(N, M)[\text{start}](N, M_1) \Rightarrow M_1 = [\text{busy}]$
- Tiếp theo, **change** được enabled  $\rightarrow$  firing change, giả sử  $(N, M_1)[\text{change}](N, M_2) \Rightarrow M_2 = [\text{docu}]$
- Sau đó, **end** được enabled  $\rightarrow$  firing end, giả sử  $(N, M_2)[\text{end}](N, M_0) \Rightarrow M_0 = [\text{free}]$ , và tiếp tục chu trình như trên.

Transition system tương ứng với petri net  $N_S$  :



Hình 8: Transition system tương ứng với petri net  $N_S$

ii. **Each place may contain any natural number of tokens in any marking**

Dựa vào mô hình *Petri net* miêu tả các trạng thái của chuyên gia, ta có thể thấy chuyên gia thực hiện các công việc thông qua 3 trạng thái **free**, **busy**, **docu** trong một vòng lặp. Do đó, đây là *Petri net* không có *terminal marking* và mỗi *place* đều có thể đạt được tối đa số lượng *token* có trong *Petri net*.

Ta quy ước bộ ba  $(x, y, z)$  với  $x$  là số *token* trong *place free*, tương tự với  $y, z$  thể hiện cho *busy, docu*.

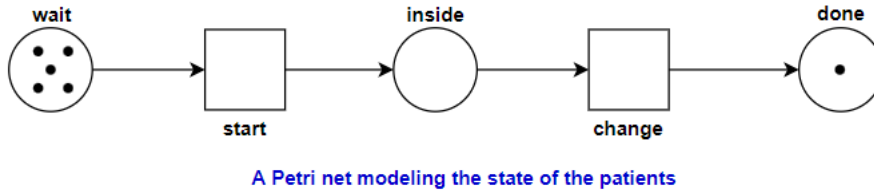
Giả sử có  $k$  *token* trong hệ thống trên, ta thực hiện xây dựng *transition system*  $TS = (S, A, TR)$  thể hiện cho quá trình trên. Trong đó:

- $S = \{(x, y, z) \mid x, y, z \in \mathbb{N} : x+y+z=k\}$
- $A = \{\text{start, end, change}\}$
- Tập  $TR \subseteq S \times A \times S$ . Ta định nghĩa tập này như sau: Giả sử  $s = (x_0, y_0, z_0) \in S, a \in A$ , khi đó:

- Nếu  $a = \text{start}$  và  $x_0 \neq 0$  thì  $(s, \text{start}, (x_0 - 1, y_0 + 1, z_0)) \in TR$
- Nếu  $a = \text{change}$  và  $y_0 \neq 0$  thì  $(s, \text{change}, (x_0, y_0 - 1, z_0 + 1)) \in TR$
- Nếu  $a = \text{end}$  và  $z_0 \neq 0$  thì  $(s, \text{end}, (x_0 + 1, y_0, z_0 - 1)) \in TR$

### 3.2 Problem 2

- (a) Token ở trong state **inside** của Petri net  $N_{Pa}$  đại diện cho những bệnh nhân (**patients**) đang được điều trị bởi các chuyên gia (**specialists**).
- (b) Xây dựng Petri net  $N_{Pa}$



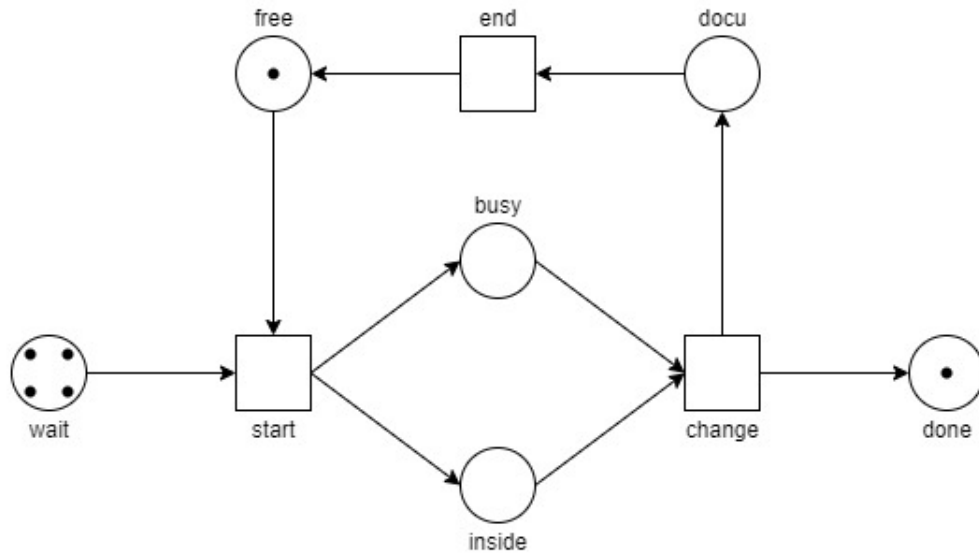
Hình 9: Transition system tương ứng với petri net  $N_{Pa}$

### 3.3 Problem 3

Gọi  $N = N_S \oplus N_{Ps} = (P, T, F)$  là Petri net miêu tả quá trình khám chữa bệnh của chuyên gia với các bệnh nhân (bằng cách kết hợp giữa Petri net của hai nhân tố là chuyên gia và bệnh nhân). Khi đó, theo định nghĩa thì ta sẽ xây dựng được các tập tương ứng:

- $P = \{\text{free, busy, docu, wait, inside, done}\}$
- $T = \{\text{start, end, change}\}$
- $F = \{(\text{start, busy}), (\text{busy, change}), (\text{change, docu}), (\text{docu, end}), (\text{end, free}), (\text{free, start}), (\text{wait, start}), (\text{start, inside}), (\text{inside, change}), (\text{change, done})\}$

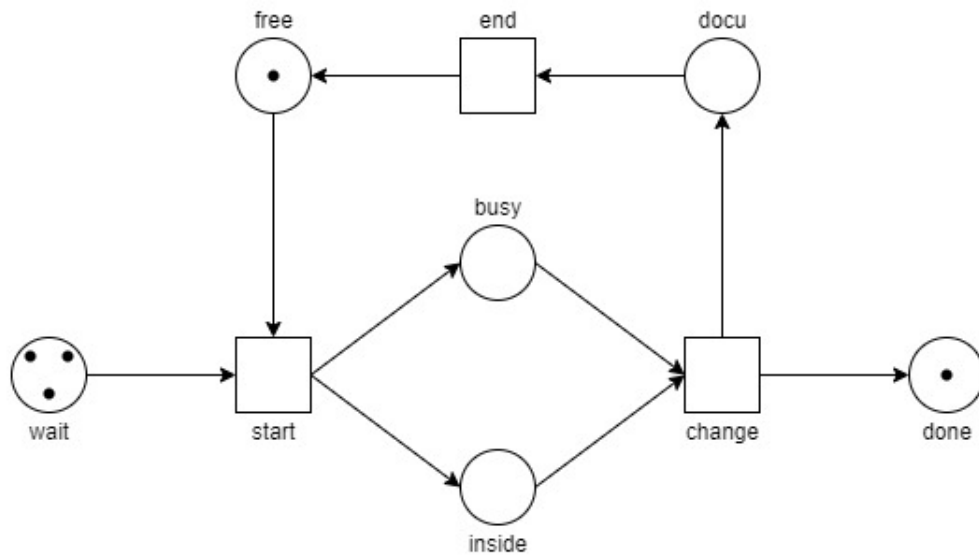
Theo như yêu cầu, ta có 4 bệnh nhân đang chờ khám bệnh, 1 bệnh nhân đã hoàn thành việc khám và bác sĩ đang ở trạng thái **free**. Do đó, *initial marking* của Petri net là  $M_0 = [\text{wait}^4, \text{done}, \text{free}]$   
Khi đó, ta có mô hình Petri net như bên dưới:



Hình 10: Mô hình Petri net tương ứng với initial marking  $M_0$

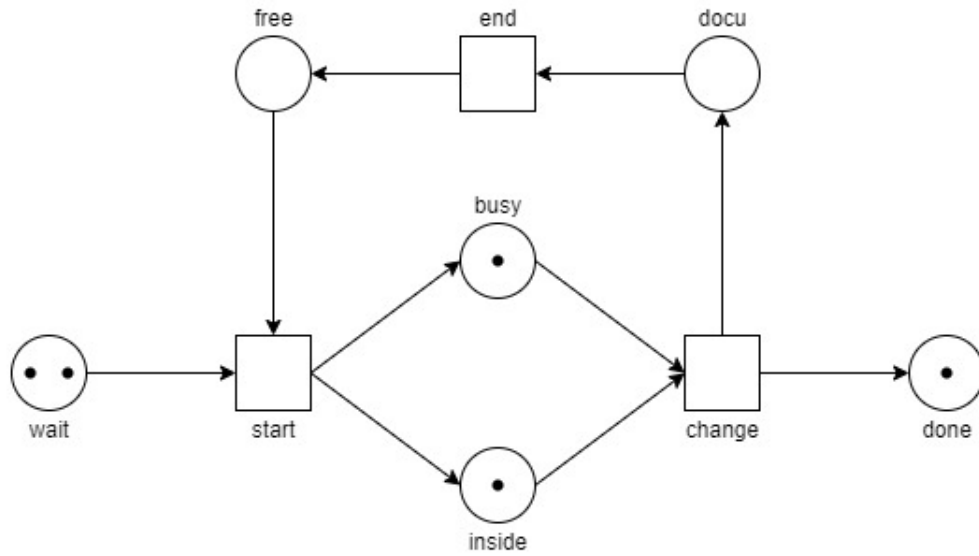
### 3.4 Problem 4

Ta có *initial marking* của Petri net là  $M_0 = [\text{wait}^3, \text{done}, \text{free}]$ . Khi đó, ta có mô hình Petri net như bên dưới:



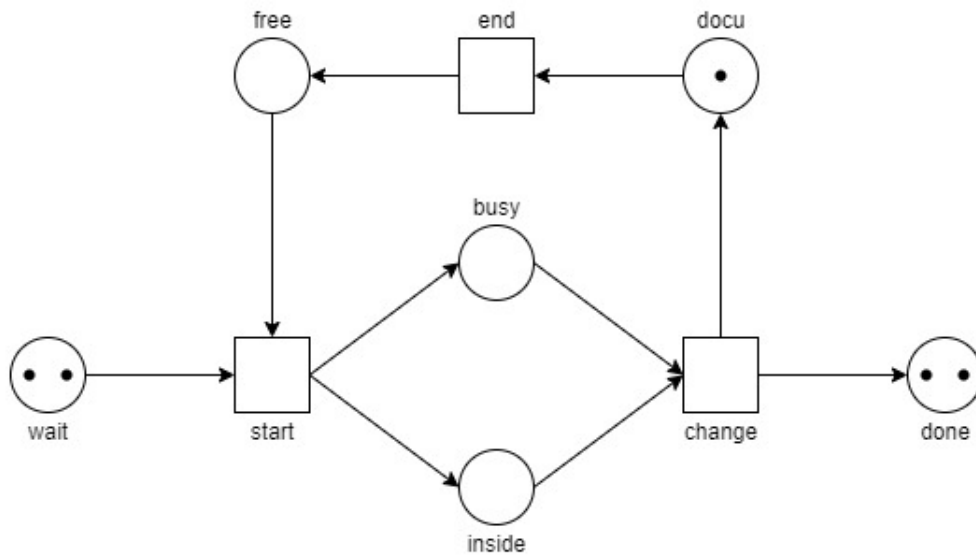
Hình 11: Mô hình Petri net tương ứng với initial marking  $M_0$

Ta thấy trong mô hình chỉ có *transition* **start** là được *enabled*. Do đó nếu từ *initial marking*  $M_0$  mà chỉ fire duy nhất một *transition* thì ta chỉ thu được một *marking* duy nhất là  $M = [\text{wait}^2, \text{busy}, \text{inside}, \text{done}]$ :



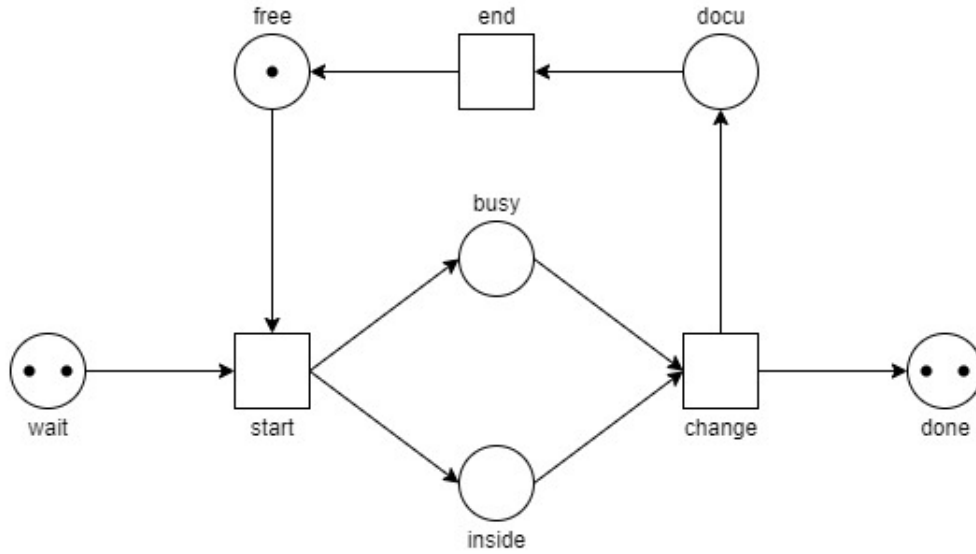
**Hình 12:** Mô hình Petri net sau khi firing transition start

Tiếp theo, chỉ có transition change là đang *enabled*, do đó marking tiếp theo ta có thể đạt tới là  $M = [wait^2, docu, done^2]$ :



**Hình 13:** Mô hình Petri net sau khi firing transition change

Hiện tại, chỉ có transition end là có thể *firing*, do đó marking tiếp theo mà Petri net đạt được là:  $M = [wait^2, free, done^2]$ :



Hình 14: Mô hình Petri net sau khi firing transition end

Đến đây, ta lặp lại việc *transition start* được *enabled*. Do đó, ta được các *marking* tiếp theo là:

- $(N, [wait^2, free, done^2]) [start] (N, [wait, busy, inside, done^2])$
- $(N, [wait, busy, inside, done^2]) [change] (N, [wait, docu, done^3])$
- $(N, [wait, docu, done^3]) [end] (N, [wait, free, done^3])$
- $(N, [wait, free, done^3]) [start] (N, [busy, inside, done^3])$
- $(N, [busy, inside, done^3]) [change] (N, [docu, done^4])$
- $(N, [docu, done^4]) [end] (N, [free, done^4])$

**Kết luận:** Ta có tổng cộng 9 *marking* có thể đạt đến nếu có *initial marking* là  $M_0 = [wait^3, done, free]$  (không tính *initial marking*).

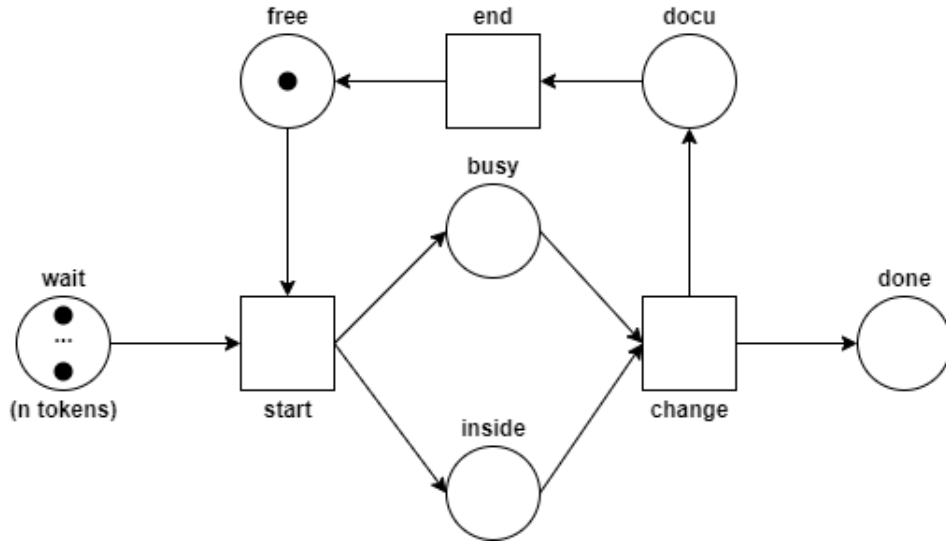
### 3.5 Problem 5

Một *marked Petri net* được gọi là **deadlock free** nếu với mỗi *reachable marking* có ít nhất một *transition* được *enabled*:

$$\forall M \in [N, M_0], \exists t \in T : (N, M)[t]$$

Hay nói cách khác, nếu như *marked Petri net N* **deadlock free** thì sẽ không tồn tại *terminal marking*.

Giả sử với 1 *initial marking* tổng quát là  $M_0 = [n.wait, free]$ , đồng nghĩa với việc có  $n$  bệnh nhân đang đợi (trạng thái **wait**) và 1 chuyên gia đang sẵn sàng cho bệnh nhân tiếp theo (trong trạng thái **free**).

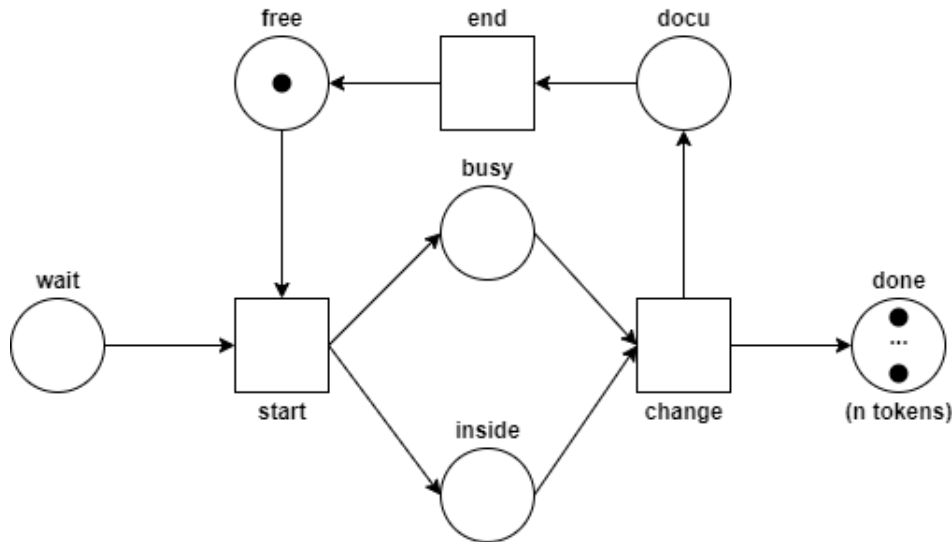


Hình 15: Petri net  $N$  với một initial marking tổng quát

Ở trạng thái đầu, chỉ có *transition* **start** là được cho phép (*enabled*). Khi khai hoả *transition* **start**, ta thu được *marking* là:  $M_1 = [(n-1).wait, busy, inside]$

Do chỉ có 1 chuyên gia, nên sau đó, *transition* duy nhất được *enabled* là **change**. Khi khai hoả *transition* **change**, ta thu được *marking* là:  $M_2 = [(n-1).wait, docu, done]$

Sau đó, khai hoả **end** ta thu được *marking* là:  $M_3 = [(n-1).wait, free, done]$ . Tương tự, ở mỗi *marking* ta đến được (trước khi *place* **wait** hết *token* và *place* **free** có 1 *token*) chỉ có duy nhất một *transition* được *enabled*. Sau tổng cộng  $3 * n$  lần khai hoả từ *initial marking*, *marking* ta thu được là  $M_{3n} = [n.done, free]$ .



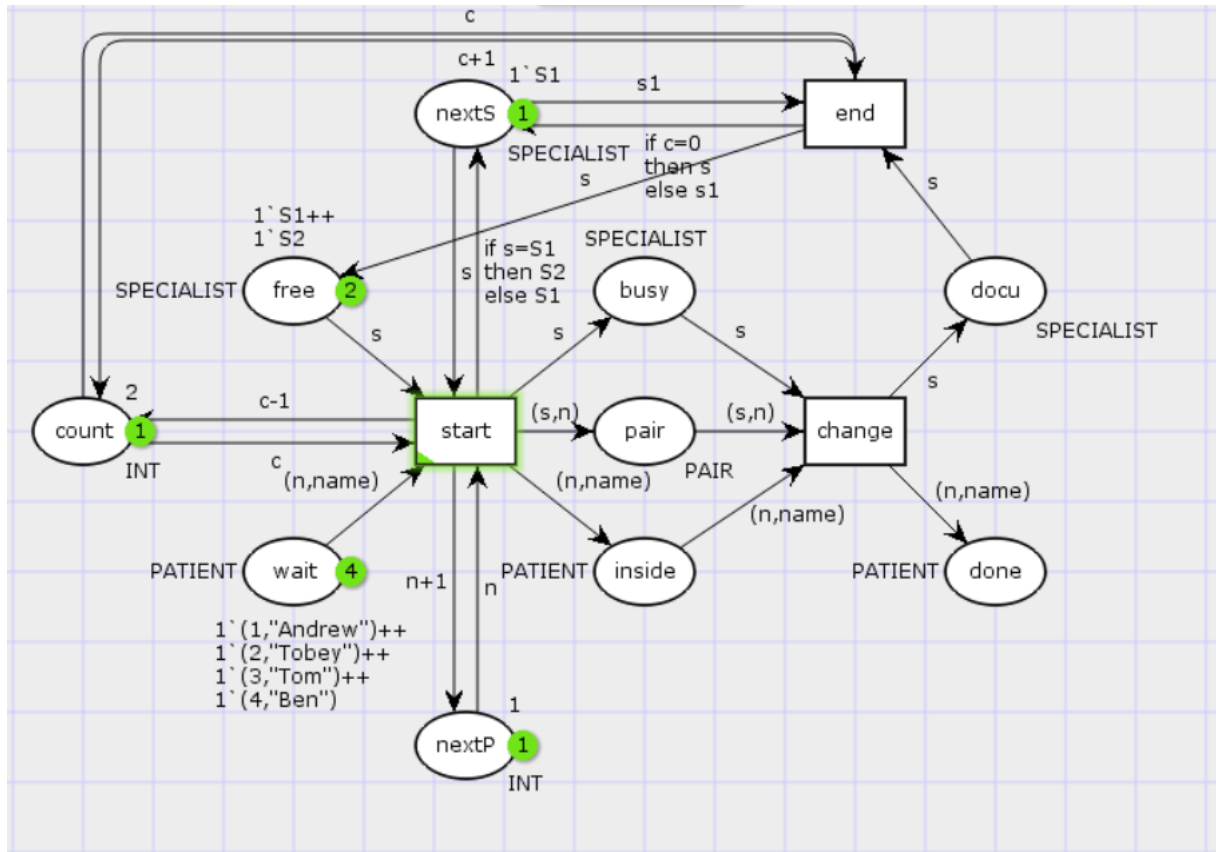
Hình 16: Petri net  $N$  sau khi khai hoả  $3n$  lần

Trong trạng thái trên, Petri net  $N$  với *marking*  $M_{3n}$  không có *transition* nào được *enabled*. Đây chính là *terminal marking* của  $N$ .

Vậy có thể kết luận rằng: Petri net  $N$  không có tính chất **deadlock free**.

### 3.6 Problem 6

Ta sẽ dùng khái niệm *colored Petri net* để đề xuất một Petri net tương tự với 2 chuyên gia sẵn sàng điều trị bệnh nhân.



Hình 17: Colored Petri net  $N'$  được đề xuất

```

▶ colset INT
▶ colset STRING
▼ colset SPECIALIST = with S1|S2;
▼ colset PATIENT = product INT * STRING;
▼ colset PAIR = product SPECIALIST * INT;
▼ var s, s1 : SPECIALIST;
▼ var n, c : INT;
▼ var name : STRING;

```

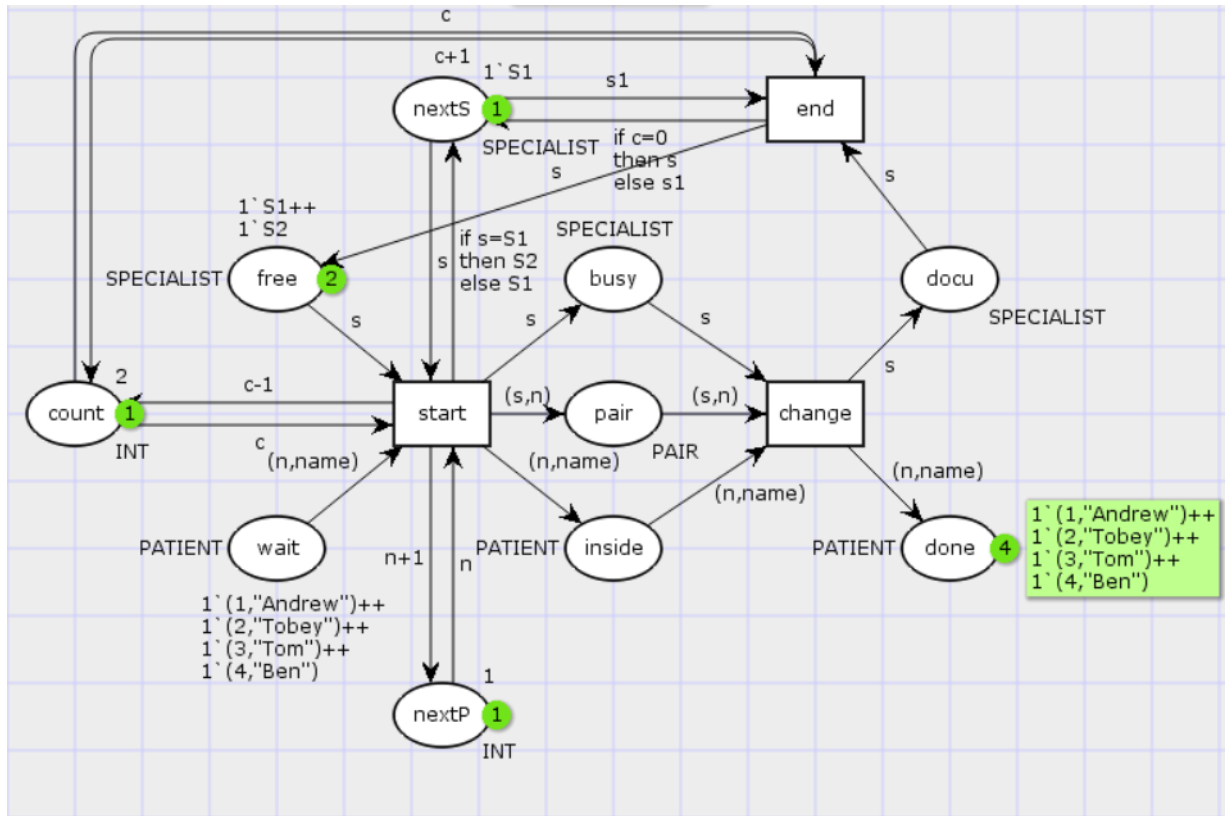
Hình 18: Các khai báo cho colored Petri net  $N'$

### Giải thích:

- Về khai báo:
  - Kiểu dữ liệu INT: số nguyên (ví dụ: 4).
  - Kiểu dữ liệu STRING: chuỗi (ví dụ: "Andrew").
  - Kiểu dữ liệu SPECIALIST: chuyên gia (với 2 giá trị tương ứng 2 chuyên gia là S1, S2).
  - Kiểu dữ liệu PATIENT: bệnh nhân, là 1 cặp giá trị gồm số nguyên (đại diện cho số thứ tự khám bệnh) và chuỗi (đại diện cho tên của bệnh nhân).
  - Kiểu dữ liệu PAIR: đại diện cho việc ghép cặp bệnh nhân với chuyên gia điều trị tương ứng, là 1 cặp giá trị gồm chuyên gia (biến SPECIALIST) và số nguyên (số thứ tự bệnh nhân).
  - Biến s, s1: chuyên gia (thuộc kiểu SPECIALIST).

- Biến  $n, c$ : số thứ tự bệnh nhân và biến đếm (thuộc kiểu INT).
- Biến `name`: tên bệnh nhân (thuộc kiểu STRING).
- Về các *place, transition*:
  - **count**: là *place* đại diện cho số lượng chuyên gia hiện có thể điều trị, thuộc kiểu INT và được khởi tạo với 1 *token* là 2.
  - **free**: là *place* đại diện cho những chuyên gia hiện có thể điều trị, thuộc kiểu SPECIALIST và được khởi tạo với 2 *token* là S1 và S2.
  - **wait**: là *place* đại diện cho những bệnh nhân đang chờ được điều trị, thuộc kiểu PATIENT và được khởi tạo với 4 *token* tương ứng 4 bệnh nhân với tên Andrew, Tobey, Tom và Ben.
  - Các *transition* **start, change, end** là các sự kiện tương tự như của *Petri net*  $N$ .
  - **nextS, nextP**: là các *place* đại diện cho chuyên gia và số thứ tự bệnh nhân kế tiếp, có kiểu SPECIALIST và INT với *token* khởi tạo tương ứng là S1 và 1.
  - Các *place* **busy, docu, inside, done** là các trạng thái của chuyên gia, bệnh nhân tương tự như của *Petri net*  $N$ , với kiểu lần lượt là SPECIALIST, SPECIALIST, PATIENT, PATIENT. Tất cả đều được khởi tạo rỗng.
  - **pair**: là *place* dùng để chứa thông tin của cặp chuyên gia và bệnh nhân tương ứng, thuộc kiểu PAIR và được khởi tạo rỗng.
- Về cách hoạt động:
  - Khác với *Petri net* thông thường, nhờ có các *arc expression, colored Petri net* có thể quản lý và phân phối *token* hiệu quả hơn.
  - Đối với *transition* **start**: sẽ chỉ được *enabled* khi **nextS** và **free** có *token* kiểu SPECIALIST với cùng giá trị (có chuyên gia tiếp theo đang sẵn sàng), **count** có *token*, **nextP** có *token* chỉ số thứ tự và **wait** có *token* kiểu PATIENT với cùng số thứ tự tương ứng (có bệnh nhân tiếp theo đang chờ). Khi được khai hoả, **start** sẽ giảm giá trị trong **count** đi 1; tiêu thụ *token* trong **free, wait**; thay đổi giá trị chuyên gia và bệnh nhân tiếp theo trong **nextS, nextP**; tạo *token* chuyên gia, bệnh nhân tương ứng trong **busy, inside** và gửi thông tin cặp chuyên gia, bệnh nhân đến **pair**.
  - Đối với *transition* **change**: sẽ chỉ được *enabled* khi có chuyên gia  $s$  trong **busy**, bệnh nhân có số thứ tự  $n$  trong **inside** và có cặp  $(s, n)$  lưu trữ trong **pair**. Khi được khai hoả, **change** sẽ tiêu thụ *token* tương ứng trong các *place* đầu vào trên, đồng thời tạo *token* kiểu SPECIALIST đến **docu** và kiểu PATIENT đến **done**.
  - Đối với *transition* **end**: sẽ chỉ được *enabled* khi có *token* trong **count, nextS** và có chuyên gia  $s$  ở trạng thái **docu**. Khi được khai hoả sẽ cộng giá trị ở **count** lên 1; tiêu thụ *token* SPECIALIST từ **docu**. Nếu hiện tại không có chuyên gia nào đang rảnh ( $c=0$ ) thì chuyên gia tiếp theo (**nextS**) sẽ là chuyên gia vừa hoàn thành **docu**, ngược lại sẽ là chuyên gia đang sẵn sàng trước đó.





**Hình 19:** *Terminal marking của colored Petri net  $N'$  (trạng thái cuối cùng khi đã điều trị tất cả bệnh nhân)*

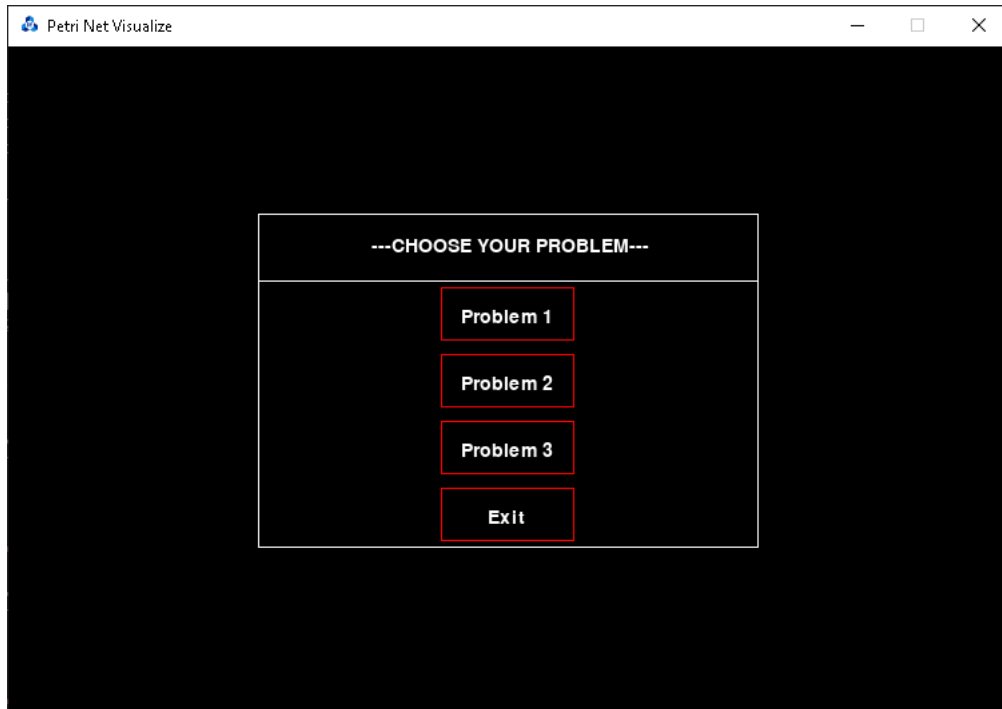
### 3.7 Problem 7

Nhóm sẽ dùng ngôn ngữ lập trình Python để hiện thực hoá các Petri Nets ở các bài tập 1, 2, 3, 4.

### 3.7.1 Hướng dẫn sử dụng chương trình:

Trong folder **PetriNet\_Code** có chứa ba file *PetriNet.exe*, *PetriNet.py* và *hcmut.png*. Người dùng nhấp chuột chọn vào file *PetriNet.exe* để chạy chương trình.

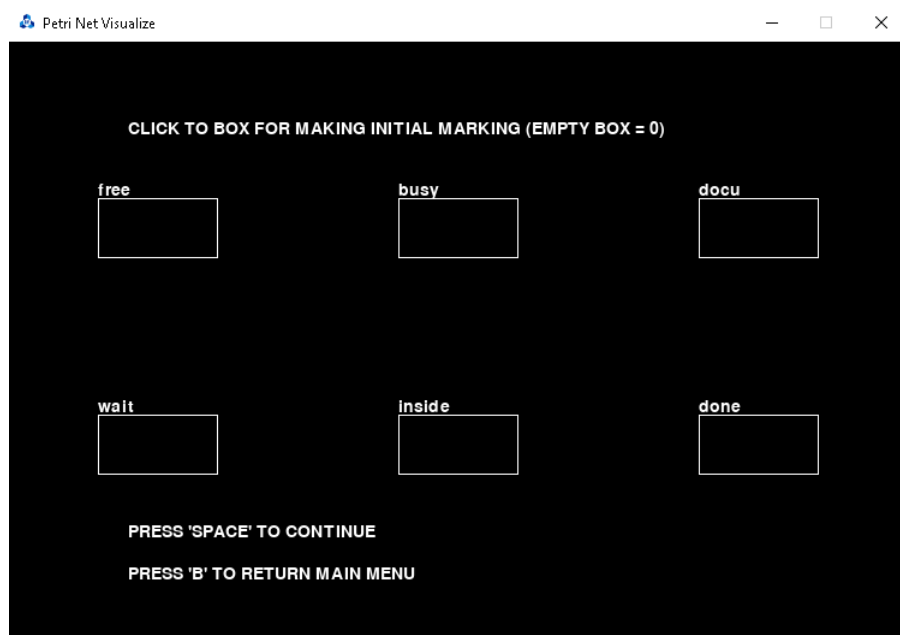
Giao diện sau khi chạy chương trình sẽ như sau:



Sau đó, người dùng có thể chọn 1 trong 4 tùy chọn để thao tác:

- **Problem 1:** Realize bài tập 1
- **Problem 2:** Realize bài tập 2
- **Problem 3:** Realize bài tập 3 và 4
- **Exit:** Thoát khỏi chương trình

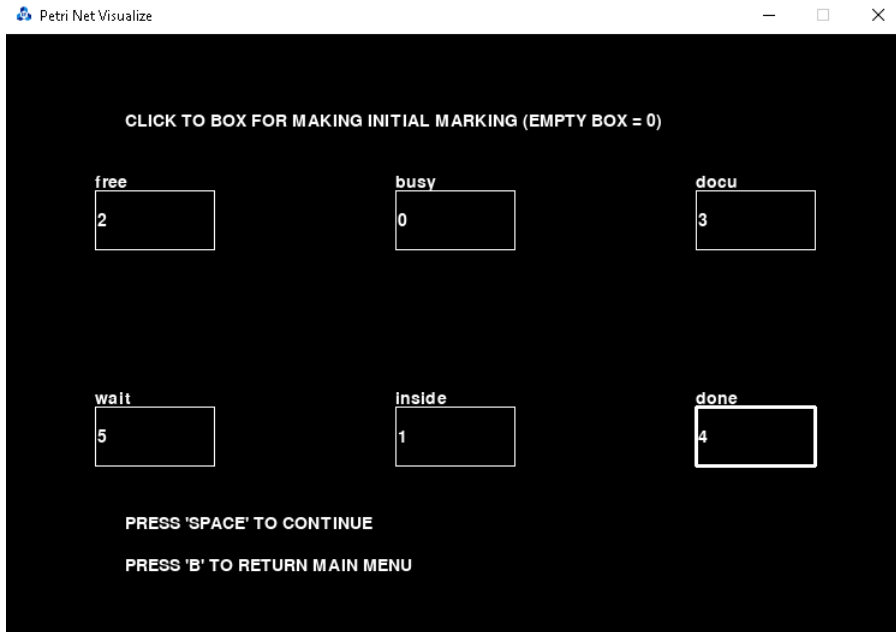
Giả sử người dùng chọn "**Problem 3**", chương trình sẽ hiển thị như sau:



**Hình 20:** Giao diện tương tác người dùng

Người dùng sẽ ấn click vào những ô trên màn hình, sau đó nhập những con số, tương ứng với số token tại transition trong Petri Nets.

Giả sử người dùng nhập các con số như hình sau:



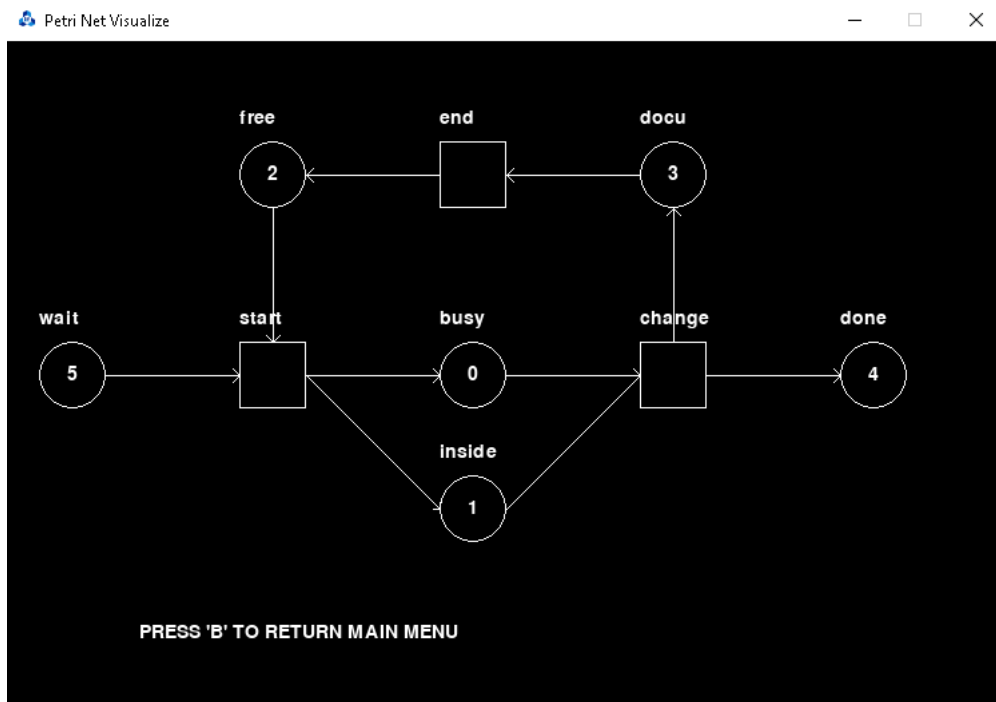
Petri Net Visualize

CLICK TO BOX FOR MAKING INITIAL MARKING (EMPTY BOX = 0)

free 2	busy 0	docu 3
wait 5	inside 1	done 4

PRESS 'SPACE' TO CONTINUE  
PRESS 'B' TO RETURN MAIN MENU

Sau khi nhấn phím "Space", chương trình sẽ hiển thị Petri Nets tương ứng với các số liệu đã chọn:



Tiếp theo, người dùng chỉ cần nhấp chuột vào **transition** tương ứng cần **fired** thì chương trình sẽ realize **token** di động chuyển từ **place** này sang **place** khác được nối bởi **transition** đó.

### 3.7.2 Setting cho chương trình

Các câu lệnh dùng cho khởi tạo chương trình ban đầu:

```
1  import pygame
2
3  pygame.init()
4  size_screen = (750, 500)
5  screen = pygame.display.set_mode(size_screen)
6  pygame.display.set_caption('Petri Net Visualize')
7  icon = pygame.image.load('hcmut.png')
8  pygame.display.set_icon(icon)
9
10 width = 1
11 backGround, color = (0, 0, 0), (255, 255, 255)
12 font = pygame.font.SysFont('Times New Roman', 14)
13 set_FPS = pygame.time.Clock()
```

Hình 21: Khởi tạo cho chương trình

- **import pygame**: Thực hiện thêm thư viện **pygame**.
- **pygame.init()**: Thực hiện thêm các module cần thiết cho chương trình.
- **size\_screen = (750, 500)**: Biến **size\_screen** chứa kích thước của cửa sổ chương trình chính.
- **screen = pygame.display.set\_mode(size\_screen)**: Biến **screen** là màn hình chính của chương trình, ta sẽ thực hiện biểu diễn các hoạt động của chương trình tại đây.
- **pygame.display.set\_caption('Petri Net Visualize')**: Thực hiện đặt đề mục cho chương trình.
- **icon = pygame.image.load('hcmut.png')**: **icon** chứa hình ảnh được load.
- **pygame.display.set\_icon(icon)**: Thực hiện đặt biểu tượng chính cho chương trình.
- **width = 1**: Biến **width** thể hiện cho độ rộng cạnh của các hình khi thực hiện mô hình.
- **backGround, color = (0, 0, 0), (255, 255, 255)**: Biến **backGround** và **color** chứa giá trị màu cho nền và cho các mô hình.
- **font = pygame.font.SysFont('Times New Roman', 14)**: Cài đặt font chữ cho chương trình.

### 3.7.3 Chương trình chính

Biến **user\_Choice\_Problem** dùng để chứa lựa chọn của người dùng khi họ tương tác với chương trình tương ứng với: Nếu người dùng muốn thực hiện **Problem 1** thì **user\_Choice\_Problem** có giá trị là 1 và tương tự với các trường hợp còn lại.

**table\_option** là đối tượng của class **TableOption** chứa các thuộc tính và các thao tác của bảng lựa chọn (ta sẽ nói rõ hơn ở phần sau).

Ta thực hiện vòng lặp để cập nhật hình ảnh của bảng chọn sau mỗi lần người dùng tương tác với bảng. Bên trong vòng lặp **while**, ta có thêm một vòng lặp **for** để lấy các sự kiện xảy ra đối với chương trình, cụ thể ở đây ta chỉ xét đến sự di chuyển chuột của người dùng và hành động nhấp chuột.

- **table\_option.interact(pygame.mouse.get\_pos())**: Ta gọi phương thức **interact()** của **table\_option** để cập nhật lại màu sắc của các button nếu con trỏ chuột ở vị trí của button.
- **event.type == pygame.QUIT**: Câu lệnh để xác định có sự kiện bấm nút X của cửa sổ hay không, nếu có thì sẽ thoát chương trình.
- **event.type == pygame.MOUSEBUTTONDOWN**: Câu lệnh để xác định có sự kiện nhấp chuột của người dùng hay không, nếu có thì ta sẽ tiếp tục xét xem người dùng có nhấp vào các button hay không bằng phương thức **interact()** của **table\_option**. Nếu người dùng click vào các

```

547 user_Choice_Problem = -1
548 table_option = TableOption()
549 done = False
550 while not done:
551     for event in pygame.event.get():
552         table_option.interact(pygame.mouse.get_pos())
553         if event.type == pygame.QUIT:
554             done = True
555         elif event.type == pygame.MOUSEBUTTONDOWN:
556             user_Choice_Problem = table_option.interact(event.pos)
557             if user_Choice_Problem == 1:
558                 ob = TableSettingToken(("free", "busy", "docu"))
559                 done = ob.display()
560             elif user_Choice_Problem == 2:
561                 ob = TableSettingToken(("wait", "inside", "done"))
562                 done = ob.display()
563             elif user_Choice_Problem == 3:
564                 ob = TableSettingToken(("free", "busy", "docu", "wait", "inside", "done"))
565                 done = ob.display()
566             elif user_Choice_Problem == 4:
567                 done = True
568     if done:
569         break
570 table_option.display()

```

Hình 22: Chương trình chính

button và chọn các **Problem** thì ta sẽ tạo ra đối tượng **ob** của class **TableSettingToken** và gọi phương thức **display()** của **ob** để chuyển hướng chương trình sang bước nhập vào **initial marking** của **Petri Net**.

### 3.7.4 Class TableOption

```

504 class TableOption:
505     def __init__(self):
506         self.name = "---CHOOSE YOUR PROBLEM---"
507         self.attribute = ("Problem 1", "Problem 2", "Problem 3", "Exit ")
508         self.size_item = (100, 40)
509         self.color_item = (255, 0, 0)
510         self.size_table = (int(size_screen[0] / 2), int(size_screen[1] / 2))
511         self.opt_item = []
512         k1 = int((self.size_table[0] - self.size_item[0]) / 2)
513         k2 = int((self.size_table[1] / 5) + (self.size_table[1] / 5 - self.size_item[1]) / 2)
514         for i in self.attribute:
515             tmp = [i, 1, (k1, k2)]
516             self.opt_item.append(tmp)
517             k2 += self.size_table[1] / 5
518
519     def display(self):
520         screen.fill(backGround)
521         my_table = pygame.Surface(self.size_table)
522         pygame.draw.rect(my_table, color, (0, 0) + self.size_table, width)
523         my_table.blit(font.render(self.name, True, color), (90, 17))
524         pygame.draw.line(my_table, color, (0, self.size_table[1] / 5), (self.size_table[0], self.size_table[1] / 5))
525         for i in self.opt_item:
526             pygame.draw.rect(my_table, self.color_item, i[2] + self.size_item, i[1])
527             my_table.blit(font.render(i[0], True, color), (int(i[2][0]) + 20, int(i[2][1]) + 15))
528         screen.blit(my_table, ((size_screen[0] - self.size_table[0]) / 2, (size_screen[1] - self.size_table[1]) / 2))
529         pygame.display.update()

```

Hình 23: Class TableOption

**Class TableOption** chính là class thể hiện cho giao diện chính của chương trình, nơi người dùng tương tác với các button để lựa chọn các **Problem** mong muốn.

#### 1. Các thuộc tính:

- **name:** Tên của bảng.

- **attribute**: Tên các button.
  - **size\_item**: Kích thước của các button.
  - **color\_item**: Màu viền của button.
  - **size\_table**: Kích thước của bảng chọn.
  - **opt\_item**: Là list chứa các list thuộc tính của từng button theo dạng [<Tên>, <Kích thước viền>, <Vị trí>].
2. Phương thức **\_\_init\_\_()**: Khởi tạo các giá trị của thuộc tính trên.
  3. Phương thức **display()**: Thực hiện in màu cho màn hình bằng lệnh **fill()**. Sau đó tạo 1 surface để thể hiện cho bảng tương tác, chứa các button. Sau đó in surface đó lên màn hình screen.
  4. Phương thức **interact(pos)**: Thực hiện nhận vào vị trí của con trỏ chuột và xác định xem con trỏ chuột có nằm trong vị trí của các button hay không. Nếu có sẽ xác định lại giá trị <Kích thước viền> của button ấy và trả về thứ tự tương ứng của button ấy, ngược lại trả về -1.

```

531     def interact(self, pos):
532         pos = list(pos)
533         pos[0] -= (size_screen[0] - self.size_table[0]) / 2
534         pos[1] -= (size_screen[1] - self.size_table[1]) / 2
535         count = 1
536         for i in self.opt_item:
537             if i[2][0] <= pos[0] <= (int(i[2][0]) + self.size_item[0]) and\
538                 i[2][1] <= pos[1] <= int(i[2][1]) + self.size_item[1]:
539                 i.__setitem__(1, 2)
540                 return count
541             else:
542                 i.__setitem__(1, 1)
543                 count += 1
544         return -1

```

### 3.7.5 Class TableSettingToken

```

429 class TableSettingToken:
430     def __init__(self, problem):
431         self.user_text = {"free": [False, "", 1], "busy": [False, "", 1], "docu": [False, "", 1],
432                             "wait": [False, "", 1], "inside": [False, "", 1], "done": [False, "", 1]}
433         self.problem = problem
434         self.size_input_box = (100, 50)
435         k1 = int((size_screen[0] - self.size_input_box[0] * 3) / 6)
436         k2 = 130
437         self.position = []
438         for i in range(2):
439             tmp = k1
440             for j in range(3):
441                 self.position.append((tmp, k2))
442                 tmp += k1 * 2 + self.size_input_box[0]
443                 k2 += self.size_input_box[1] + k2

```

Hình 24: Class TableSettingToken

Class **TableSettingToken** là class thể hiện cho giao diện nhập vào **initial marking**.

1. Các thuộc tính:
  - **user\_text**: Là một dictionary chứa cặp key:value là tên của **place** và list có dạng [<1>, <2>, <3>]:
    - <1>: Thể hiện cho việc người dùng có đang chọn input box này hay không.
    - <2>: Chuỗi nhập vào hiện tại của input box.

– <3>: Độ lớn viền của input box.

- **problem**: Chứa tên các **place**.
- **size\_Input\_box**: Kích thước của input box.
- **position**: là list của cái vị trí của các input box.

2. Phương thức **\_\_init\_\_()**: Khởi tạo các giá trị của thuộc tính.

3. Phương thức **display()**: Thực hiện vòng lặp **while** để thể hiện giao diện nhập vào **initial marking** của người dùng. Trong mỗi vòng lặp ta cần lấy ra các sự kiện hiện tại và thực hiện, cụ thể ta chỉ quan tâm đến sự kiện nhấn nút trên bàn phím và nhấn chuột. Nếu người dùng nhấn phím B thì ta sẽ thoát vòng lặp và quay về bảng chọn chính, nếu nhấn **space** thì ta sẽ chuyển hướng chương trình sang thực hiện các mô hình và nếu người dùng nhấn vào các nút số từ 0 đến 9 thì ta sẽ thực hiện tìm kiếm input box nào đã được kích hoạt cho việc nhập bằng phương thức **find\_box\_active()**, nếu có tồn tại input box được kích hoạt thì ta cập nhật lại **user\_text**. Nếu người dùng nhấn nút thì ta sẽ gọi phương thức **check\_in\_box()** để kiểm tra người dùng có nhấp vào các input box hay không.

```
445 def display(self):
446     stop = False
447     while not stop:
448         for event_setting in pygame.event.get():
449             if event_setting.type == pygame.QUIT:
450                 return True
451             elif event_setting.type == pygame.KEYDOWN:
452                 if event_setting.key == pygame.K_SPACE:
453                     obj = Problem3Solving(self.user_text)
454                     if len(self.problem) == 3:
455                         if self.problem[0] == "free":
456                             obj = Problem1Solving(self.user_text, self.problem)
457                         else:
458                             obj = Problem2Solving(self.user_text, self.problem)
459                     if obj.display():
460                         return True
461                     return False
462                 elif event_setting.key == pygame.K_b:
463                     return False
464                 tmp = self.find_box_active()
465                 if tmp != -1:
466                     if event_setting.key == pygame.K_BACKSPACE:
467                         self.user_text[tmp][1] = self.user_text[tmp][1][:1]
468                     else:
469                         c = event_setting.unicode
470                         if '0' <= c <= '9':
471                             self.user_text[tmp][1] += c
472                 elif event_setting.type == pygame.MOUSEBUTTONDOWN:
473                     self.check_in_box(event_setting.pos)
474
475     screen.fill(backGround)
476     for i in range(len(self.problem)):
477         pygame.draw.rect(screen, color, self.position[i] + self.size_Input_box,
478                          self.user_text[self.problem[i]][2])
479         screen.blit(font.render(self.problem[i], True, color), (self.position[i][0], self.position[i][1] - 14))
480         screen.blit(font.render(self.user_text[self.problem[i]][1], True, color),
481                      (self.position[i][0] + 2, self.position[i][1] + 18))
482     screen.blit(font.render("CLICK TO BOX FOR MAKING INITIAL MARKING (EMPTY BOX = 0)", True, color), (100, 65))
483     screen.blit(font.render("PRESS 'SPACE' TO CONTINUE", True, color), (100, size_screen[1] - 100))
484     screen.blit(font.render("PRESS 'B' TO RETURN MAIN MENU", True, color), (100, size_screen[1] - 65))
485     pygame.display.update()
486
487 def check_in_box(self, pos):
488     for i in range(len(self.problem)):
489         if self.position[i][0] <= pos[0] <= self.position[i][0] + self.size_Input_box[0] and self.position[i][1] <= \
490            pos[1] <= self.position[i][1] + self.size_Input_box[1]:
491             self.user_text[self.problem[i]][2] = 3
492             self.user_text[self.problem[i]][0] = True
493         else:
494             self.user_text[self.problem[i]][2] = 1
495             self.user_text[self.problem[i]][0] = False
496
497 def find_box_active(self):
498     for i in self.problem:
499         if self.user_text[i][0]:
500             return i
501     return -1
```

4. Phương thức **check\_in\_box(pos)**: Thực hiện kiểm tra xem vị trí con trỏ chuột có nằm trong input box hay không và thực hiện cập nhật lại các giá trị của **user\_text**.

5. Phương thức **find\_box\_active()**: Thực hiện kiểm tra xem có input box nào đang được người dùng chọn để nhập hay không, nếu có thì trả về input box ấy, ngược lại trả về -1.

### 3.7.6 Class Problem1Solving và class Problem2Solving

Đây là hai class dùng để biểu diễn Problem 1 và 2 (Petri net cho chuyên gia và bệnh nhân) tương ứng.

Đối với mỗi class, ta tạo 2 biến là *Places* và *Transitions*. 2 biến này thuộc kiểu *Dictionary* trong Python, với trường *key* là tên của node place/transition và trường *value* là 1 danh sách các thông số của node đó như sau:

- Đối với *Places*: trường *value* theo thứ tự sẽ là: 1 số nguyên thể hiện *token* của place và 1 cặp số nguyên thể hiện toạ độ tâm của place đó khi biểu diễn trên khung hình.
- Đối với *Transitions*: trường *value* theo thứ tự sẽ là: 1 danh sách chứa tên các *input place*, 1 danh sách chứa tên các *output place* và 1 cặp số nguyên thể hiện toạ độ tâm của transition đó khi biểu diễn trên khung hình.

```

16 class Problem1Solving:
17     def __init__(self, user_text, problem):
18         self.size, self.length = 25, 100
19         self.places = {"free": [0, (250, 150)], "busy": [0, (400, 300)], "docu": [0, (550, 150)]}
20         self.transitions = {"start": (["free"], ["busy"], (250, 300)),
21                             "change": (["busy"], ["docu"], (550, 300)),
22                             "end": (["docu"], ["free"], (400, 150))}
23         for i in problem:
24             if user_text[i][1] != "":
25                 self.places[i][0] = int(user_text[i][1])
26
27     def display_places(self):
28         for place, info in self.places.items():
29             token = font.render(str(info[0]), True, color)
30             screen.blit(token, (info[1][0] - 4, info[1][1] - 8))
31             name = font.render(place, True, color)
32             screen.blit(name, (info[1][0] - self.size, info[1][1] - self.size * 2))
33             pygame.draw.circle(screen, color, info[1], self.size, width)
34
35     # display transitions
36     def display_transitions(self):
37         for transition, info in self.transitions.items():
38             name = font.render(transition, True, color)
39             screen.blit(name, (info[2][0] - self.size, info[2][1] - self.size * 2))
40             pygame.draw.rect(screen, color,
41                             (info[2][0] - self.size, info[2][1] - self.size, self.size * 2, self.size * 2), width)
42

```

Hình 25: Khởi tạo các biến *Places*, *Transitions* và phương thức để biểu diễn các *place* và *transition*



```

43 # display arrow
44 def display_arrow(self, x1, y1, x2, y2):
45     if x1 == x2:
46         if y1 > y2:
47             pygame.draw.line(screen, color, (x1, y1 - self.size), (x2, y2 + self.size), width)
48             pygame.draw.line(screen, color, (x2 - 5, y2 + self.size + 5), (x2, y2 + self.size), width)
49             pygame.draw.line(screen, color, (x2 + 5, y2 + self.size + 5), (x2, y2 + self.size), width)
50         else:
51             pygame.draw.line(screen, color, (x1, y1 + self.size), (x2, y2 - self.size), width)
52             pygame.draw.line(screen, color, (x2 - 5, y2 - self.size - 5), (x2, y2 - self.size), width)
53             pygame.draw.line(screen, color, (x2 + 5, y2 - self.size - 5), (x2, y2 - self.size), width)
54     else:
55         if x1 > x2:
56             pygame.draw.line(screen, color, (x1 - self.size, y1), (x2 + self.size, y2), width)
57             pygame.draw.line(screen, color, (x2 + self.size + 5, y2 + 5), (x2 + self.size, y2), width)
58             pygame.draw.line(screen, color, (x2 + self.size + 5, y2 - 5), (x2 + self.size, y2), width)
59         else:
60             pygame.draw.line(screen, color, (x1 + self.size, y1), (x2 - self.size, y2), width)
61             pygame.draw.line(screen, color, (x2 - self.size - 5, y2 - 5), (x2 - self.size, y2), width)
62             pygame.draw.line(screen, color, (x2 - self.size - 5, y2 + 5), (x2 - self.size, y2), width)
63
64 # display arcs
65 def display_arcs(self):
66     for transition, info in self.transitions.items():
67         for input_place in info[0]:
68             x1, x2, y1, y2 = self.places[input_place][1][0], info[2][0], self.places[input_place][1][1], info[2][1]
69             self.display_arrow(x1, y1, x2, y2)
70         for output_place in info[1]:
71             x1, x2, y1, y2 = info[2][0], self.places[output_place][1][0], info[2][1], self.places[output_place][1][1]
72             self.display_arrow(x1, y1, x2, y2)

```

Hình 26: Các phương thức để biểu diễn dây cung

Ở đây ta có các phương thức dùng để biểu diễn các place, transition và arc lên khung hình:

- Phương thức *display\_places*: biểu diễn các place lưu trong biến *Places* thành các hình tròn (tâm là toạ độ đã nêu trên) cùng các văn bản thể hiện tên và số lượng *token* của place đó.
- Phương thức *display\_transitions*: biểu diễn các transition lưu trong biến *Transitions* thành các hình vuông (tâm là toạ độ đã nêu trên) cùng văn bản thể hiện tên của transition đó.
- Phương thức *display\_arrow* và *display\_arcs*: dùng để biểu diễn các dây cung (các mũi tên có hướng) lên khung hình. Trong đó, *display\_arrow* có chức năng vẽ các mũi tên (gồm 3 đoạn thẳng) từ toạ độ tâm (x1, y1) đến (x2, y2) nhận vào (lưu ý: do đặc thù của Petri net đang hiện thực nên hàm chỉ vẽ được các mũi tên nằm dọc/ nằm ngang). Tiếp theo, *display\_arcs* sẽ lặp qua các transtion và vẽ các dây cung từ *input place* và các dây cung đến *output place*.

```

74 # check enable
75 def check_enable(self, transition):
76     for input_place in self.transitions[transition][0]:
77         if self.places[input_place][0] <= 0:
78             return False
79     return True
80
81 # check click for transition
82 def check_trans(self, coordinate):
83     x, y = coordinate[0], coordinate[1]
84     for transition, info in self.transitions.items():
85         if (info[2][0] - self.size <= x <= info[2][0] + self.size) and (
86             info[2][1] - self.size <= y <= info[2][1] + self.size):
87             return transition
88     return None

```

Hình 27: Các phương thức để kiểm tra enable và transition được chọn

Tiếp theo, ta có phương thức *check\_enable* để kiểm tra 1 transition có *enable* hay không. Phương thức hoạt động bằng cách lặp qua tất cả *input place* của transition và kiểm tra *token* của

nó. Nếu bất cứ place nào có  $token \leq 0$ , trả về *False*. Ngược lại thì transition được *enable* nên trả về *True*.

Phương thức *check\_trans* được dùng để xác định transition được chọn từ người dùng dựa trên tọa độ đưa vào. Phương thức sẽ trả về tên của transition tương ứng. Nếu tọa độ không thuộc về transition nào thì trả về *None*.

```
90 # firing animation
91 def animation(self, x1, y1, x2, y2):
92     while (x1 != x2) or (y1 != y2):
93         screen.fill(backGround)
94         self.display_places()
95         self.display_transitions()
96         self.display_arcs()
97         pygame.draw.circle(screen, color, (x1, y1), 3, 3)
98         pygame.display.update()
99         set_FPS.tick(120)
100        if x1 == x2:
101            if y1 > y2:
102                y1 -= 2
103            else:
104                y1 += 2
105        else:
106            if x1 > x2:
107                x1 -= 2
108            else:
109                x1 += 2
```

Hình 28: Phương thức để biểu diễn hoạt ảnh khai hoả

Phương thức *animation* có chức năng biểu diễn hoạt ảnh khai hoả trong khung hình. Hoạt ảnh này được tạo ra bằng cách vẽ liên tiếp các chấm tròn (đại diện cho *token*) dọc theo các dây cung từ *input place* đến transition và từ transition đến *output place*.

```

111 def display(self):
112     done_1 = False
113     while not done_1:
114         screen.fill(backGround)
115         for event_1 in pygame.event.get():
116             if event_1.type == pygame.QUIT:
117                 done_1 = True
118             elif event_1.type == pygame.KEYDOWN:
119                 if event_1.key == pygame.K_b:
120                     return False
121             elif event_1.type == pygame.MOUSEBUTTONDOWN:
122                 co = event_1.pos
123                 trans = self.check_trans(co)
124                 if trans is not None and self.check_enable(trans):
125                     for in_place in self.transitions[trans][0]:
126                         self.places[in_place][0] -= 1
127                         self.animation(self.places[in_place][1][0], self.places[in_place][1][1],
128                                     self.transitions[trans][2][0], self.transitions[trans][2][1])
129                     for out_place in self.transitions[trans][1]:
130                         self.animation(self.transitions[trans][2][0], self.transitions[trans][2][1],
131                                     self.places[out_place][1][0], self.places[out_place][1][1])
132                         self.places[out_place][0] += 1
133                 self.display_places()
134                 self.display_transitions()
135                 self.display_arcs()
136                 screen.blit(font.render("PRESS 'B' TO RETURN MAIN MENU", True, color), (100, size_screen[1] - 65))
137                 pygame.display.update()
138     return True

```

Hình 29: Phương thức chính biểu diễn Petri net

Phương thức *display* của mỗi class là phương thức chính hiện thực Petri net của mỗi Problem lên khung hình. Một vòng lặp sẽ được chạy và chỉ dừng khi người dùng đóng cửa sổ (nhấn nút X ở góc trên bên phải) hoặc nhấn phím B. Trong vòng lặp, ta sẽ kiểm tra thao tác nhấp chuột lên các transition bằng phương thức *check\_trans* và kiểm tra *enable* bằng phương thức *check\_enable*. Nếu transition được *enable*, ta sẽ biểu diễn hoạt ảnh khai hoả đồng thời thay đổi *token* trong các *input place* (-1) và *output place* (+1).

### 3.7.7 Class Problem3Solving

Giống với 2 class trên. Tuy nhiên do Petri net này có các dây cung nằm chéo khi biểu diễn và hoạt ảnh khai hoả cần điều chỉnh nên có những sự khác biệt sau:

- Phương thức *display\_arrow* được mở rộng để có thể biểu diễn mũi tên từ transition *start* đến place *inside* và từ place *inside* đến transition *change*.
- Tạo thêm 2 phương thức *animation1* và *animation2* chuyên dùng để biểu diễn hoạt ảnh của transition *start* và *change* tương ứng. Giống với *animation*, nhưng ở đây 2 phương thức này giúp biểu diễn 2 *token* (2 chấm đen) chuyển động đồng thời với nhau qua lại giữa các place thông qua transition.
- Phương thức *display* cũng đã được điều chỉnh riêng biệt để mô phỏng Petri net hợp thành này.

```
255 # display arrow
256 def display_arrow(self, x1, y1, x2, y2):
257     if x1 == x2:
258         if y1 > y2:
259             pygame.draw.line(screen, color, (x1, y1 - self.size), (x2, y2 + self.size), width)
260             pygame.draw.line(screen, color, (x2 - 5, y2 + self.size + 5), (x2, y2 + self.size), width)
261             pygame.draw.line(screen, color, (x2 + 5, y2 + self.size + 5), (x2, y2 + self.size), width)
262         else:
263             pygame.draw.line(screen, color, (x1, y1 + self.size), (x2, y2 - self.size), width)
264             pygame.draw.line(screen, color, (x2 - 5, y2 - self.size - 5), (x2, y2 - self.size), width)
265             pygame.draw.line(screen, color, (x2 + 5, y2 - self.size - 5), (x2, y2 - self.size), width)
266     elif y1 == y2:
267         if x1 > x2:
268             pygame.draw.line(screen, color, (x1 - self.size, y1), (x2 + self.size, y2), width)
269             pygame.draw.line(screen, color, (x2 + self.size + 5, y2 + 5), (x2 + self.size, y2), width)
270             pygame.draw.line(screen, color, (x2 + self.size + 5, y2 - 5), (x2 + self.size, y2), width)
271         else:
272             pygame.draw.line(screen, color, (x1 + self.size, y1), (x2 - self.size, y2), width)
273             pygame.draw.line(screen, color, (x2 - self.size - 5, y2 - 5), (x2 - self.size, y2), width)
274             pygame.draw.line(screen, color, (x2 - self.size - 5, y2 + 5), (x2 - self.size, y2), width)
275     elif y1 < y2:
276         pygame.draw.line(screen, color, (x1 + self.size, y1), (x2 - self.size, y2), width)
277         pygame.draw.line(screen, color, (x2 - self.size, y2 - 5), (x2 - self.size, y2), width)
278         pygame.draw.line(screen, color, (x2 - self.size - 5, y2), (x2 - self.size, y2), width)
279     else:
280         pygame.draw.line(screen, color, (x1 + self.size, y1), (x2 - self.size, y2), width)
281         pygame.draw.line(screen, color, (x2 - self.size, y2 + 5), (x2 - self.size, y2), width)
282         pygame.draw.line(screen, color, (x2 - self.size - 5, y2), (x2 - self.size, y2), width)
```

Hình 30: Phương thức `display_arrow` mới

```
332 # firing animation ("start" trans only)
333 def animation1(self):
334     (x1, y1), (x2, y2) = (200, 100), (50, 250)
335     while x1 != x2:
336         screen.fill(backGround)
337         self.display_places()
338         self.display_transitions()
339         self.display_arcs()
340         pygame.draw.circle(screen, color, (x1, y1), 3, 3)
341         pygame.draw.circle(screen, color, (x2, y2), 3, 3)
342         pygame.display.update()
343         set_FPS.tick(120)
344         y1 += 2
345         x2 += 2
346     x1, y1, x2, y2 = 225, 250, 225, 250
347     while y2 - y1 != 100:
348         screen.fill(backGround)
349         self.display_places()
350         self.display_transitions()
351         self.display_arcs()
352         pygame.draw.circle(screen, color, (x1, y1), 3, 3)
353         pygame.draw.circle(screen, color, (x2, y2), 3, 3)
354         pygame.display.update()
355         set_FPS.tick(120)
356         x1 += 2
357         x2 += 2
358         y2 += 2
```

Hình 31: Phương thức *animation1* biểu diễn hoạt ảnh khai hoả cho transition start

```
360 # firing animation ("change" trans only)
361 def animation2(self):
362     x1, y1, x2, y2 = 375, 250, 375, 350
363     while y1 != y2:
364         screen.fill(backGround)
365         self.display_places()
366         self.display_transitions()
367         self.display_arcs()
368         pygame.draw.circle(screen, color, (x1, y1), 3, 3)
369         pygame.draw.circle(screen, color, (x2, y2), 3, 3)
370         pygame.display.update()
371         set_FPS.tick(120)
372         x1 += 2
373         x2 += 2
374         y2 -= 2
375     x1, y1, x2, y2 = 500, 250, 500, 250
376     while x2 - x1 != 150:
377         screen.fill(backGround)
378         self.display_places()
379         self.display_transitions()
380         self.display_arcs()
381         pygame.draw.circle(screen, color, (x1, y1), 3, 3)
382         pygame.draw.circle(screen, color, (x2, y2), 3, 3)
383         pygame.display.update()
384         set_FPS.tick(120)
385         y1 -= 2
386         x2 += 2
```

Hình 32: Phương thức *animation2* biểu diễn hoạt ảnh khai hoả cho *transition change*

## 4 Kết luận:

Qua bài báo cáo, ta thấy được **Petri Nets** là một công cụ hứa hẹn cho việc miêu tả và nghiên cứu các hệ thống như hệ thống đồng bộ, không đồng bộ, song song, không xác định hay ngẫu nhiên. Vì là một công cụ đồ họa, Petri Nets có thể đóng vai trò như một công cụ hỗ trợ bằng hình ảnh tương tự như flow charts, block diagrams hay mạng net works. Ngoài ra, còn có thêm một khái niệm là tokens được dùng để mô phỏng trạng thái các hoạt động và tính đồng bộ của hệ thống. Như một mô hình toán học, ta có thể thiết lập các phương trình trạng thái, phương trình đại số và các phép toán bằng cách sử dụng các behavior của hệ thống.

Tóm lại, **Petri Nets** có thể nói là một công cụ mô hình hóa hữu hiệu cho nhiều loại hệ thống thông tin. Kể từ khi Petri Nets lần đầu tiên được giới thiệu vào năm 1960, đã có rất nhiều bài viết về những nhánh ứng dụng tiềm năng của chủ đề này. Do việc có rất nhiều chủ đề cần quan tâm khi nói đến Petri Nets, nên ở bài báo cáo này, nhóm tác giả chỉ mô tả một cách tổng quan nhất về các khái niệm và những ứng dụng phổ biến nhất của mạng Petri.

## 5 Tài liệu tham khảo:

1. Wil van der Aalst. 2003. Applications and Theory of Petri Nets 2003.
2. Wil van der Aalst. 1998. The application of Petri nets to workflow management
3. Wil van der Aalst. 2013. Decomposing Petri nets for process mining: A generic approach
4. RWTH Aachen University. 2021. A Python Extension to Simulate Petri nets in Process Mining
5. Mogens Nielsen and Vladimiro Sassone. 1991. Petri Nets and Other Models of Concurrency
6. C. Ou - Yang and H. Winarjo. 2010. Petri net integration - An approach to support multi - agent process mining
7. B.F. van Dongen, A.K. Alves de Medeiros and L. Wen. 2009. Process Mining: Overview and Outlook of Petri Net Discovery Algorithms
8. Jensen, K. 1981. Colored Petri nets and the invariant-method, Theoretical Computer Science
9. Jensen, K., 1997. Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use
10. Peterson, J. L. 1981. Petri Net Theory and the Modeling of Systems
11. Ajmone Marsan, M. 1990. Stochastic Petri nets: an elementary introduction. Advances in Petri Nets
12. Ramamoorthy, C., and G. Ho. 1980. Performance evaluation of asynchronous concurrent systems using Petri nets
13. E. BEST AND R. DEVILLERS. 1987. Sequential and Concurrent Behaviour in Petri Net Theory.
14. Tsai, J., S. Yang, and Y. Chang. 1995. Timing constraint Petri nets and their application to schedulability analysis of real-time system specifications