Ho Chi Minh City University of Technology
Faculty of Computer Science and Engineering

Mathematical Modeling (CO2011)

Assignment Report
Petri net

Lecturer: Nguyen An Khuong
Team name: TxtbookDestroyer
Team members: Le Phuoc Gia Loc – 2052155
Tran Quoc Bao – 2052038
Truong Tan Hao Hiep – 2011211
Nguyen Cong Duy – 2052915
Le Khanh Duy – 2052003

Ho Chi Minh City, 06 December 2021

Contents

1 Introduction

In today digital world, most business or organization processes are fully run or supported by various system with large amount of data. As the process become more and more complicated with more and more data involved, the tradition process optimization cannot cope with the challenge. The demand for a new simple model that can describe exactly how the process should be arrive.

Emerged as a new research field since 1990s, process mining is a group of techniques relating the fields of data science and process management to support the analysis of processes using event data. Process mining techniques aim to discover, monitor and improve real processes by extracting knowledge from event logs. The three main classes of process mining are:

- Process discovery: transform an event log into a process model. An event log contains a case id (a unique identifier for a particular process instance), an activity description (describing the executed activity) and a timestamp of the execution activity.
- Conformance checking: comparing an event log with an existing process model to analyze the discrepancies between them.
- Performance analysis: The model is extended with additional performance information such as processing times, cycle times, waiting times, costs, etc., to check for further improvement of the existing model.

One of the main concerns of process mining has been concurrency. In many activities of cooperation and companies, transactions are a common occurrence. A transaction is divided into two main stages: execution stage and committing stage. At the execution stage, transaction access data through a concurrency control, while at the committing stage, a commit protocol is executed to ensure failure atomicity. A transaction that requests a lock can be blocked by a committing transaction for a long time due to the delay in completing the committing procedure. The potential long delay can lead to data item cannot be access during a transaction.

In the concurrency area, this problem has led to the development of large number of concurrency control algorithms. One of the methodology and key techniques to this problem is Petri net.

Petri nets are a powerful modeling formalism in computer science, system engineering and many other disciplines. Petri nets combine a mathematical theory with a graphical representation of dynamic behavior of systems. The theory aspect allows precise modeling and analysis of the system, while the graphical one enable visualization of the modeled system states changes. Consequently, Petri nets have been used to model various kinds of dynamic event-driven systems like computers networks, communication systems, real-time computing systems, workflows, ... This wide spectrum of applications is accompanied by wide spectrum different aspects which have been considered in the research on Petri nets. Carl Adam Petri's technique was initially intended to be significantly more expansive. He was looking for a theory of information processing that was consistent with physical principles, had deep invariants in natural scientific traditions, and used formalisms to describe a human-centered, pragmatic approach to computer science's technological capabilities.

Interesting theoretical issues regarding Petri nets have been presented and addressed for years. Tools have been created and special subclasses have been investigated. There have been case studies completed and successful projects completed. Petri nets, unlike some other modeling approaches that were popular for a brief period and then disappeared, have maintained its status as a well-established modeling technique.

With the incredible growth of event data in most big companies and large-scale scientific studies nowadays, new challenges has been posed. As event logs grow, process mining techniques need to become more efficient and highly scalable. Moreover, torrents of event data need to be distributed over multiple databases and larges process mining problems need to be shared over a network of computers. In fact, today's organization are already storing terabytes of event data. Techniques

aiming only at precise results quickly become intractable when dealing with such massive event logs.

This report main purpose is to present a general approach in the name of Petri net to decompose existing process discovery and conformance checking problem. More modern processing, modeling problems will occur, thus Petri nets will become more and more essential, so it is important to understand well the concept and practice implementing it. As a part of our journey in studying Petri nets, we build a simple model and program of Petri nets for a real-life problem to express our knowledge in the field as well as widen it throughout the progress.

2 Background

2.1 Petri net

A Petri net is a graphical tool (a bipartite graph consisting of places and transitions) for the description and analysis of concurrent processes which arise in systems with many components (distributed systems). The graphics, together with the rules for their coarsening and refinement, were invented in August 1939 by Carl Adam Petri.

2.2 An intuitive approach

In this section, we will exercise practical examples to cover concepts related to Petri net and how they can be applied to construct a Petri net.

Before that, there is the state transition system – the basic principle of a discrete system. It is used to decide *which* activities need to be executed and in *what order*; be it sequential, concurrent or repeated.

Definition 2.1: State transition system

A transition system is a triplet $TS = (S, A, T)$ where S is the set of *states*, $A \subseteq \mathcal{A}$ is the set of *activities* (often referred to as *actions*), and $T \subseteq S \times A \times S$ is the set of *transitions*.

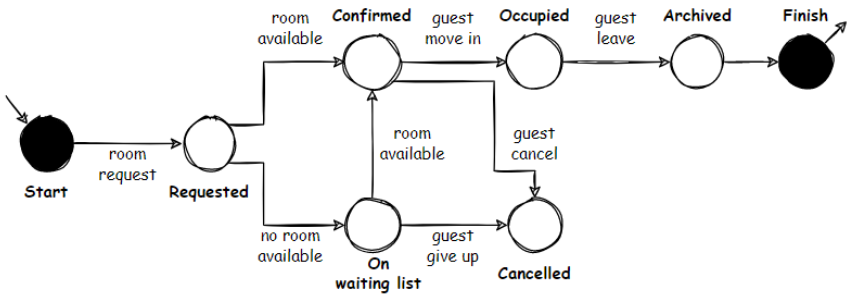

Figure 2.2.1: A hotel reservation procedure

The transition starts in one of the initial states. Any path in the graph starting in such a state corresponds to a possible execution sequence. A path terminates successfully if it ends in one of the final states (as for the example, a successfully terminated path is Start → Requested→ Confirmed → Occupied → Archived → Finish) or deadlocks if it reaches a non-final state without any outgoing transitions (for example, in Figure 2.4.4, if **Currency box** is empty while **Card holder** has a token, this triggers a deadlock).

2.3 Petri net model

Since any process model with executable semantics can be mapped onto a transition system, many notions defined for transition systems can easily be translated to higher – level languages such as Petri net ... As a matter of fact, transition systems are simple but have problems expressing concurrency succinctly, i.e. 'state explosion'. However, a Petri net can be used much more compactly and efficiently. We will delve into the details of a Petri net and how it works via the structure of an ATM.

The machine has a *card reader* and a *cash dispenser* out of which the cash come. In the *initial state* of the ATM, the card reader contains a card and the cash dispenser is empty.

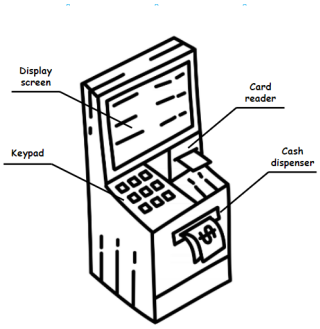

Figure 2.3.2: Example of an ATM

Consider a Petri net model of the machine (as in Figure 2.3.3a): **card reader** and **cash dispenser**, both depicted as circles, are the *places* of the Petri net. The **card reader** contains a credit card, which is a *token* of the net (in Figure 2.3.3a).

A distribution of tokens across places is a *marking* of a Petri net. Figure 2.3.3b presents the new marking which the card reader is empty, while the cash dispenser contains cash as a token.

The machine now can accept the credit card and in return, dispense the corresponding amount of cash. In a Petri net, this process is modeled as a *transition* **Process**, depicted as a square.

(a) The ATM in its initial state

(b) The ATM after transition Process

Figure 2.3.3: Example of a Petri net model

As the directions and labeling of the arrows in Figure 2.3.3a show, the credit card "flows" out of the card reader, and cash "flows" into the cash dispenser. The arrows depict the *arcs* of a Petri net.

Arcs and transitions can be labeled with *expressions*. These expressions can either be functions or variables as in Figure 2.3.3a; along with a central property: if all variables in an expression are replaced by elements, it becomes possible to evaluate the expression in order to obtain yet another element.

Definition 2.2: Formal Petri net

A Petri net is a triplet $N = (P, T, F)$ where P is a finite set of *places*, T is a finite set of *transitions* such that $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed *arcs*, called the *flow relation*.

2.4 Analyze the extended model

If we look inside transition **Process**, we will find a more detailed procedure. Since we only cover the basic concepts and behaviour of a Petri net, we will use a simple restructured Petri net model of an ATM with the sole purpose of withdrawing cash instead.

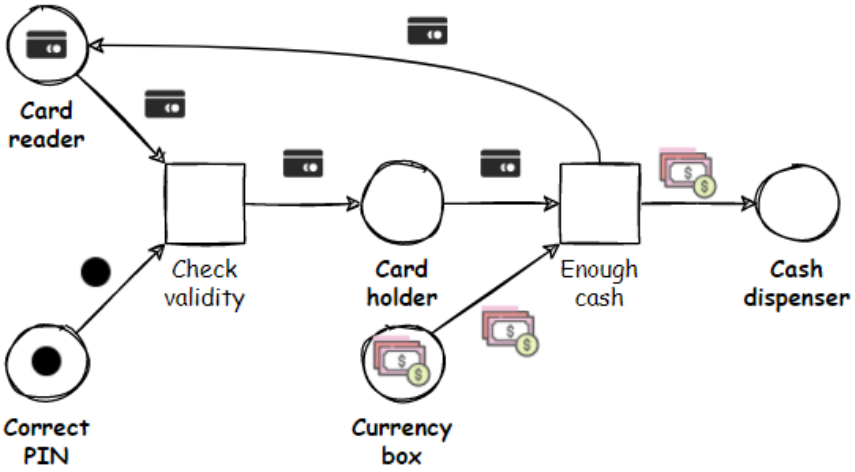

Figure 2.4.4: A more detailed look of an ATM machine

Figure 2.4.4 models the withdrawal procedure of an ATM machine as a Petri net: there is the addition of **Correct PIN** signal (initially empty if the PIN is incorrect), a **Card holder** and a **Currency box** filled with cashes.

In Figure 2.4.4, the transition **Check validity** is *enabled* because its incoming arcs start at places containing at least one token each, as required by the arc's label. Therefore, Check validity can *occur* and thereby change the current marking.

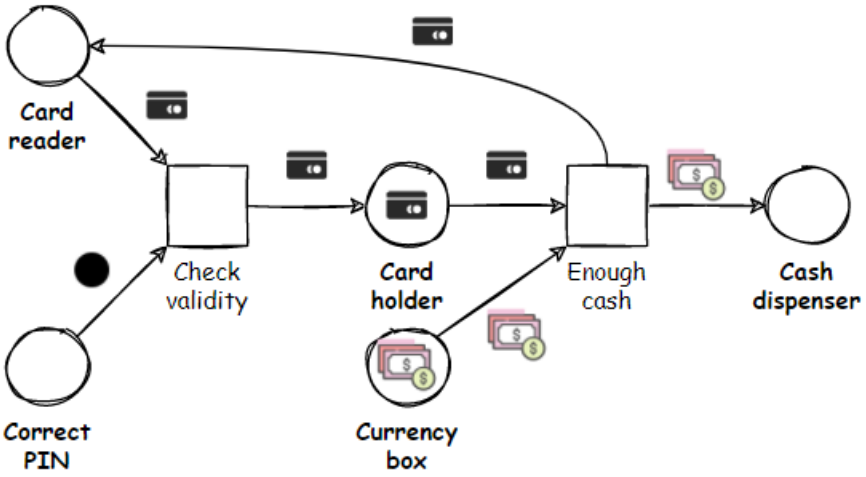

Figure 2.4.5: After the occurrence of Check validity

The occurrence of a transition is the act of *firing*, which is demonstrated from Figure 2.4.4 to Figure 2.4.5. A transition must be **enabled** to fire, thereby changing the current marking and consuming one token from each of its input places to produce one token in each of its output places.

Figure 2.4.6 represents a *marked Petri net*, which is a pair (N, M), where $N = (P, T, F)$ is a Petri net and M is a *multi-set* over P denoting the **marking** of the net.

2.5 Multi-set

In a Petri net, the tokens of a place often represent objects that we usually do not want to distinguish. In general, examples of different kinds of tokens mixed in one place are called *multi-set*, e.g., dollars, euros and yens in the **Currency box**.

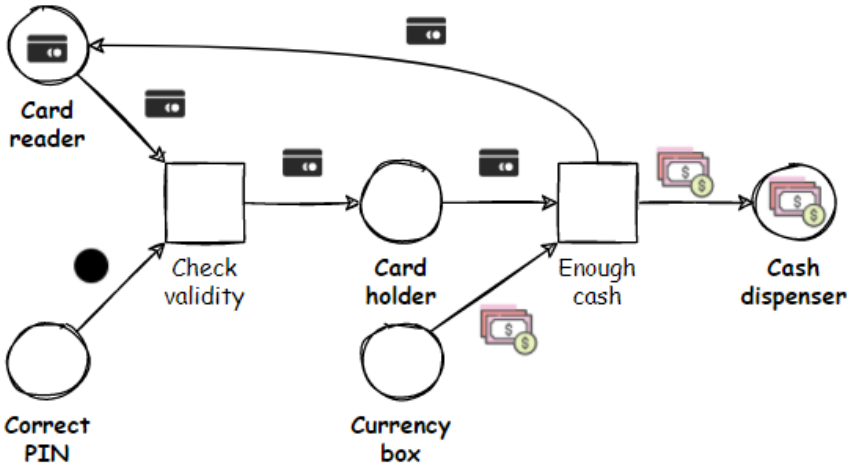

Figure 2.4.6: After the occurrence of Enough cash

Definition 2.3. Multi-set

A *multi-set* a is a mapping $a : U \to \mathbb{N}$ that maps every object u of a *universe* U to the number of its occurrences in a.

We always assume a sufficiently large universe U that contains all examined kinds of tokens. We write the set of all multi-sets over U as $\mathcal{M}(U)$ or $\mathcal{M}$ for short if the context unambiguously identifies the universe U.

The universe U can contain an infinite number of elements, for instance, all natural numbers. A multi-set a over U can map the value $a(u) = 0$ to almost all $u \in U$. That means that u does not occur in a. Thus, a is finite if

$$a(u) \neq 0 \quad \forall u \in U_0, U_0 \subset U \text{ and } |U_0| < \infty.$$

We write a finite multi-set a with its multiple elements in square brackets [...], e.g. $[a^2 2, b^5, c]$ or short as $[2, 5, 1]$ if the order is predetermined. The *empty* multi-set is denoted by empty [ ]:

$$\forall u \in U, [\,](u) = 0$$

Two multi-sets $a, b \in M$ can be added: $\forall u \in U$, let

$$(a + b)(u) \stackrel{\text{def}}{=} a(u) + b(u)$$

They can be compared:

$$a \leqslant b \text{ iff } \forall u \in U : a(u) \leqslant b(u),$$

and b can be subtracted from a if $b \leqslant a$:

$$(a - b)(u) \stackrel{\text{def}}{=} a(u) - b(u).$$

With these notations, we can describe the dynamic behaviour of Petri net.

Definition 2.4. Marking

A marking M of a net structure (P, T, F) is a mapping

$$M : P \to \mathcal{M}$$

Means that M maps every place p to a multi-set $M(p)$.

As explained, a marking M describes a *state* of the modeled system. Given the significance of a system's initial state, the initial marking (usually denoted M_0) is often drawn into the respective net structure.

2.6 Formal definition of Firing rule

First, we need to cover some basic notations to get a better understanding of the upcoming mathematical demonstration.

- A node x is an *input node* of another node y if and only if there is a directed arc from x to y (i.e.,$(x, y) \in F$). Node x is an *output node* of y if and only if $(y, x) \in F$.
- For any $x \in P \cup T$, we define ${}^\bullet x = \{y | (y, x) \in F\}$ as the preset of x, and $x^\bullet = \{y | (x, y) \in F\}$ as the postset of x.
- Transition $t \in T$ is enabled at marking M, denoted $(N, M)[t\rangle$, if and only if ${}^\bullet t \leqslant M$.

Definition 2.5: The firing rule

Let $(N, M) \in \mathcal{N}$ be a marked Petri net with $N = (P, T, F)$ and $M \in \mathcal{M}$. The firing rule $\alpha[t\rangle\beta \subseteq \mathcal{N} \times T \times \mathcal{N}$ is the smallest relation satisfying

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

for any $(N, M) \in \mathcal{N}$ and any $t \in T$.

Since the act of firing is the occurrence of an enabled transition, we can achieve a sequence of transitions by firing continuously, which is denoted as a *firing sequence*.

Formally, a sequence $\sigma \in T^*$ is called a **firing sequence** of (N, M_0) *if and only if*, for some natural number $n \in \mathbb{N}$, there exist markings $M_1, M_2, ..., M_n$ and transitions $T_1, T_2, ..., T_n$ such that

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

and for all i with $0 \leqslant i < n$, then

$$(N, M_i)[t_i + 1\rangle \text{ and } (N, M_i)[t_i + 1\rangle(N, M_i + 1)$$

2.7 Advanced net structure

With supporting concepts mentioned in the previous sections, we can build up more sophisticated Petri net structure for various purpose. Typically, a *Petri net system* (P, T, F, M_0), which consists of a Petri net (P, T, F) and a distinguished marking M_0 - the initial marking.

Another structure is a *Labeled Petri net*, which works on the following principles:

- Transitions are often labeled by a single letter but they also have a longer label describing the corresponding activity.
- $A \in \mathcal{A}$ is a set of *activity labels*, and the map $l \in \{L : T \to A\}$ is a *labeling function*. [One can think of the transition label as the *observable action*. Sometimes one wants to express that particular transitions are **not** observable, or invisible.]
- Use the label τ for a special activity label, called 'invisible'. A transition $t \in T$ with $l(t) = \tau$ is said to be **unobservable**, **silent** or **invisible**.

2.8 Another way to represent Petri net

We can describe the behavior of a Petri net system $(N, M_0) \equiv (P, T, F, M_0)$ as a state transition system (S, TR, S_0) by showing how to determine the state space S, the transition relation TR and the initial state S_0 for the system (P, T, F, M_0).

As for the reason why we choose state transition system to represent a Petri net: the transition system represents the state space of the modeled system, thus representing all possible markings M of the net.

Before we delve into the conversion, it is crucial to define the term reachable as well as relating notations.

Definition 2.6: Reachability

A marking M is *reachable* from the initial marking M_0 **if and only if** there exists a sequence of enabled transitions whose firing leads from M_0 to M. The set of reachable markings of (N, M_0) is denoted $[N, M_0\rangle$.

Reachability graph

Let (N, M_0) with $N = (P, T, F, A, l)$ be a marked labeled Petri net. (N, M_0) defines a transition system $TS = (S, A_1, TR)$ with

$$S = [N, M_0\rangle,\ S^{start} = \{M_0\},\ A_1 = A,$$

and

$$TR = \{(M, M_1) \in S \times S \mid \exists t \in T(N, M)[t\rangle(N, M_1)\},$$

or with label $l(t)$:

$$TR = \{(M, l(t), M_1) \in S \times A \times S \mid \exists t \in T(N, M)[t\rangle(N, M_1)\},$$

TS is often referred to as the *reachability graph* of (N, M_0).

2.9 Modeling problem with Petri net

In a process, there often are many agents, they all have different business activities, as a result, their Petri nets have distinct places/ tokens and transitions. However, the dynamic of entire system (by all agents) is made up by mutual interactions, from which many places should usually have/share the same tokens, and sometimes do the transitions.

Therefore, the grand Petri net of the whole system is not simply the disjoint union of the constituents' nets, it should be the superimposition of the smaller nets.

Formally, consider a system of only two agent types, denoted N_1, N_2, to be their own Petri nets. Assume that $N_1 = (P_1, T_1, F_1, M_0)$ and $N_2 = (P_2, T_2, F_2, M_0)$, where the places P_i could be disjoint, but with the same initial marking M_0.

We can define the superimposition operator, $\oplus : T_1 \times T_2 \to T$, where $T = T_1 \cup T_2$, as follows:

- If ${}^\bullet t_1 = {}^\bullet t_2$ then $(t_1, t_2) \mapsto \oplus(t_1, t_2) = t \in T$ with ${}^\bullet t = {}^\bullet t_1$, (the presets of t_i are the same, keep one version only in the merged net). Else ${}^\bullet t_1 \neq {}^\bullet t_2$ then $(t_1, t_2) \mapsto \oplus(t_1, t_2) = \{t_1, t_2\} \subseteq T$, keep both presets.
- Namely, we can identify two transitions/events of two nets into one node of the merged Petri net $N = N_1 \oplus N_2$ if the events act on the same physical token.
- The superimposed (merged) Petri net is determined by

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

3 Exercises

3.1 Practice 5.1

Consider the Petri net figure below.

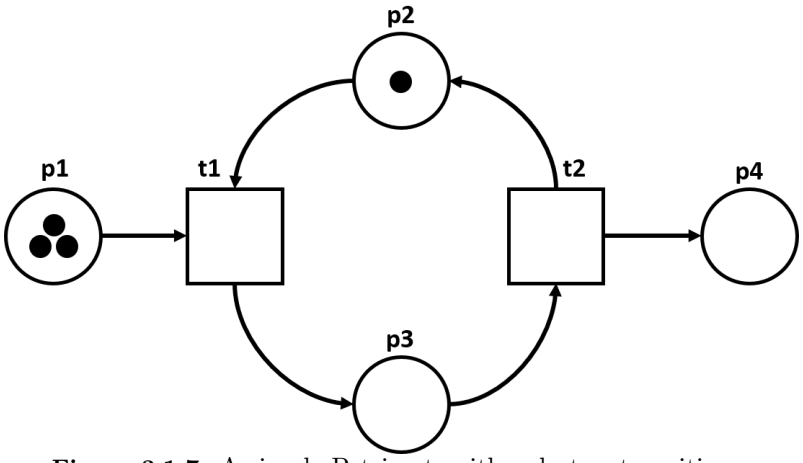

Figure 3.1.7: A simple Petri net, with only two transitions.

1. **Define the net formally as a triplet** (P, T, F).
   The Petri net above can be shown in the form $N = (P, T, F)$ with:
   - $P = \{p1, p2, p3, p4\}$
   - $T = \{t1, t2\}$
   - $F = \{(p1, t1), (t1, p3), (p3, t2), (t2, p2), (t2, p4), (p2, t1)\}$
2. **List presets and postsets for each transition.**
   - $\{p1, p2\}$ is the preset and $\{p3\}$ is the postset of transition $t1$.
   - $\{p3\}$ is the preset and $\{p2, p4\}$ is the postset of transition $t2$.
3. **Determine the marking of the net.**
   - Marking of the net: $[3, 1, 0, 0]$
4. **Are the transition t1 and t2 enabled in this net?**
   - Transition t1 is *enabled* because both of its input $p1$ and $p2$ have at least one token.
   - Transition t2 is *not enabled* because its input $p3$ do not have any token.

3.2 Practical problem 1

Given a process of a X-ray machine in which we assume the first marking in Figure 3.2.8.a shows that there are three patients in the queue waiting for an X-ray. Figure 3.2.8.b depicts the next marking, which occurs after the firing of transition enter.

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

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

1. Determine the two relations R_I and R_O, and the flow relation $F = R_I \cup R_O$.

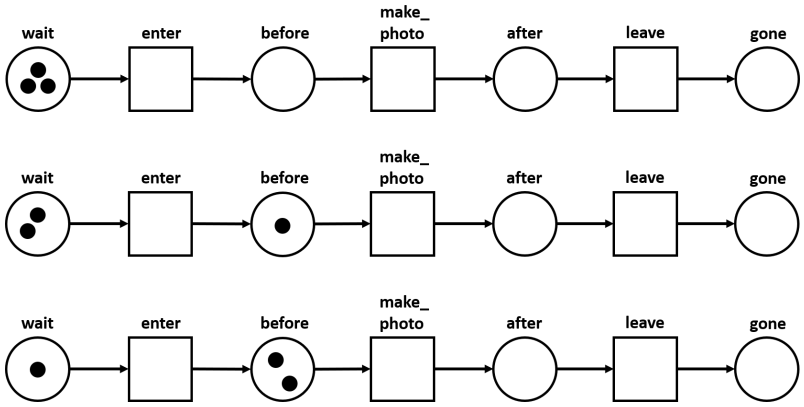

Figure 3.2.8: A Petri net model of a business process of an X-ray machine.

- $P = (wait, before, after, gone)$.
- $T = (enter, make_photo, leave)$.
- $R_I = \{(wait, enter), (before, make\_photo), (after, leave)\}$.
- $R_O = \{(enter, before), (make\_photo, after), (leave, gone)\}$.
- $F = \{(wait, enter), (enter, before), (before, make\_photo), (make\_photo, after), (after, leave), (leave, gone)\}$.

2. A patient may enter the X-ray room only after the previous patient has left the room. We must make sure that places **before** and **after** together do not contain more than one token.

   There are two possible states: the room can be **free** or **occupied**. We model this by adding theses two places to the model, to get the improved the Petri net, in Figure 3.2.9.

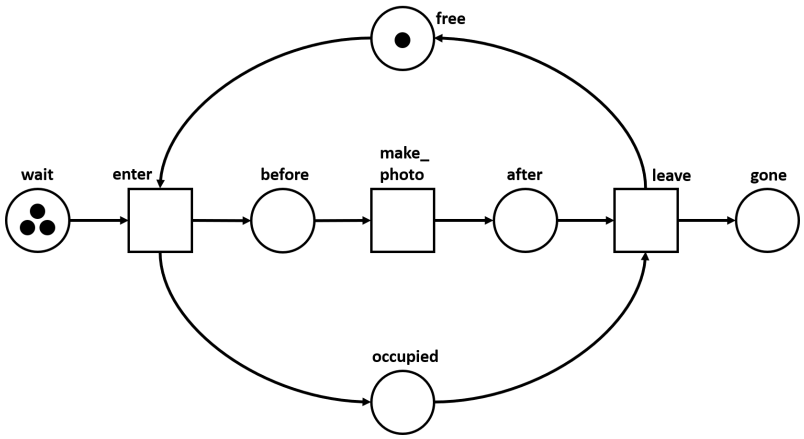

Figure 3.2.9: An improved Petri net for the business process of an X-ray machine.

   Now for this Petri net, can place **before** contain more than one token? Why? Rebuild the set P of place labels.

   - Assume that there is a token in place **before**. Place **before** obtain another token *if and only if* transition **enter** is enabled. Meaning that place **free** must contain a token which require the firing of transition **make_photo** and consume the current token in place **before** as there can only be one token in either places **before** or **after**. As a consequence, it is certain that place **before** can never contain more than one token.
   - $P = \{wait, before, after, free, occupied, gone\}$.

3. As long as there is no token in place **free** (Figure 3.2.10), can transition **enter** fire again? Explain why or why not. Remake the two relations R_I and R_O.

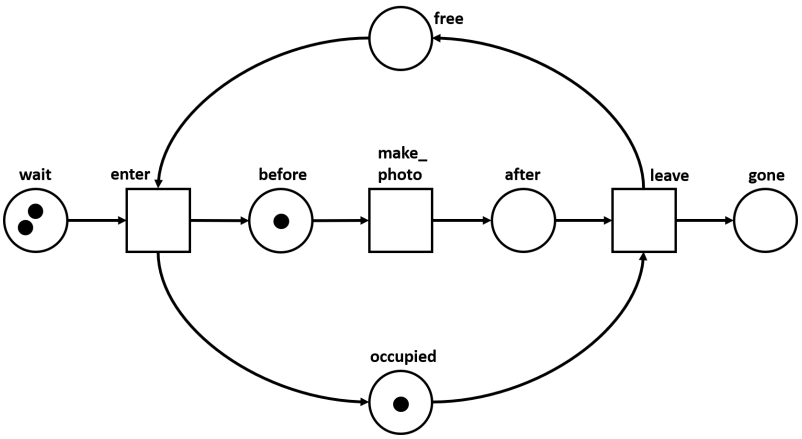

Figure 3.2.10: The marking of the improved Petri net after transition **enter** has fired.

- As long as there is no token in the place free, the transition **enter** can not fire because it is not *enabled*.
- The two relations R_I and R_O after remake:
  $R_I = \{(wait, enter), (free, enter), (before, make\_photo), (after, leave), (occupied, leave)\}$.
  $R_O = \{(enter, before), (enter, occupied), (make\_photo, after), (leave, free), (leave, gone)\}$.

3.3 PROBLEM 5.1

Explain the following terms for Petri nets, and provide a specific example for each term:

1. **"enabled transition"**: transition is enabled if each of its input places contains a token.
   Example:

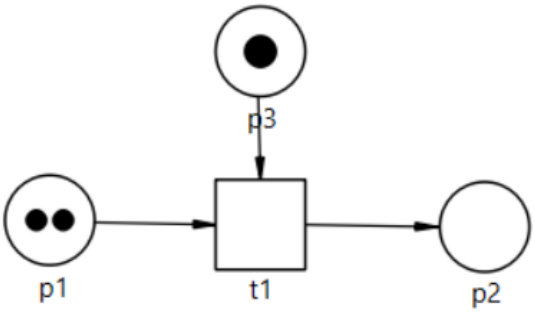

Figure 3.3.11

   In figure 3.3.7, each input places of transition t1 (p3 and p1) contains token, so t1 is enabled transition.

2. **"firing of a transition"**: An enabled transition can fire, thereby consuming (energy of) one token from each input place and producing at least one token for each output place next.
   Figure 3.3.8 is the result of figure 3.3.7 when firing transition t1, one token in p3 and one token in p1 will be consumed and produced one token in p2

3. **"reachable marking"**: A marking M is reachable from the initial marking M_0if and only if there exists a sequence of enabled transitions whose firing leads from $M_0$to M
   Ex: Figure 3.3.8 is a reachable marking from figure 3.3.7:

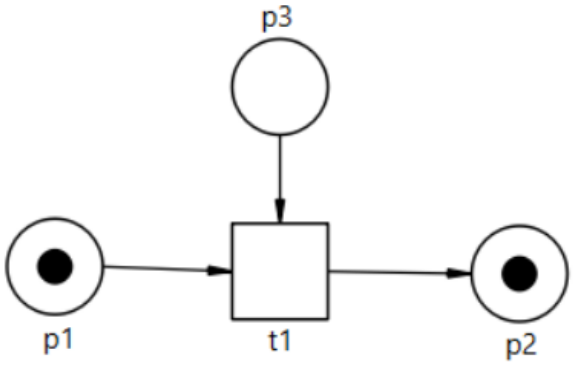

Figure 3.3.12

$M_0 = [2.p1, 1.p3]$

$M_0 = [1.p1, 1.p2]$

4. **"terminal marking"**: The transitions keep firing until the net reaches a marking that does not enable any transition. This marking is a terminal marking.

   Ex: Figure 3.3.8 is a terminal marking.

5. **"non-deterministic choice"**: When several transitions are enabled at the same moment, it is not determined which of them will fire. This situation is a non-deterministic choice.

   Example: At figure 3.3.9, both transition t1 and t2 are enabled, so it's can not determine which transition will be fired. This is a non-deterministic choice.

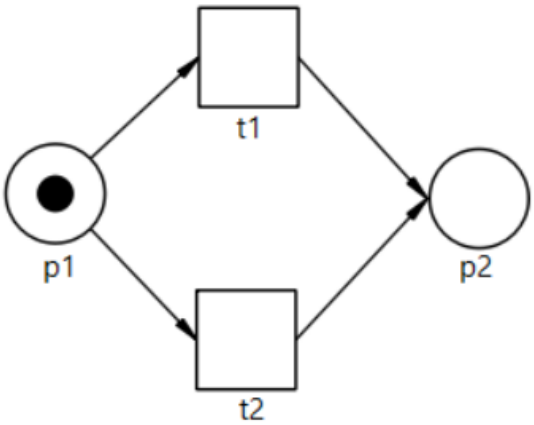

Figure 3.3.13

3.4 PROBLEM 5.2

Consider the Petri net system in figure below.

1. **Formalize this net as a quadruplet** (P, T, F, M_0)
   - $P = \{p1, p2, p3, p4\}$
   - $T = \{t1, t2, t3\}$
   - $F = \{(p1, t1); (p2, t2); (p3, t3); (p4, t3); (p4, t1)\} \cup \{(t1, p2); (t2, p3); (t3, p4); (t2, p4)\}$
   - $M_0 = [p1, p3, 2p4]$

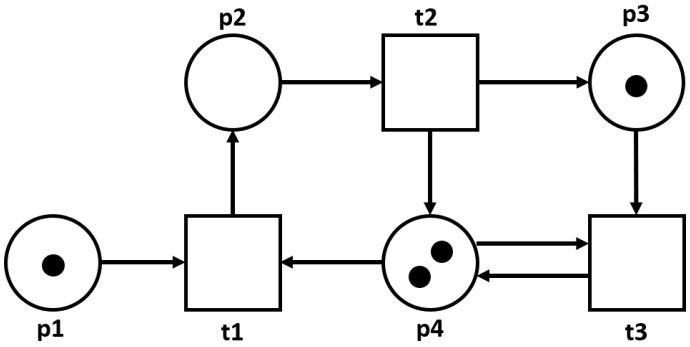

Figure 3.4.14: A Petri net with small numbers of places and transitions.

2. **Give the preset and the postset of each transition.**
   - Transition t1: $preset = \{p1, p4\}$ and $postset = \{p2\}$
   - Transition t2: $preset = \{p2\}$ and $postset = \{p3\}$
   - Transition t3: $preset = \{p3, p4\}$ and $postset = \{p4\}$
3. **Which transitions are enabled at M_0?**
   Transition t1 and t3 are enabled at M_0
4. **Give all reachable markings. What are the reachable terminal markings?** All the reachable markings:
   $M_0 = [p1, p3, 2p4]$
   $M_1 = [p2, p3, p4]$
   $M_2 = [p1, p3, 2p4]$
   $M_3 = [2p3, 2p4]$
   $M_4 = [p3, 2p4]$
   $M_5 = [2p4]$
   $M_6 = [p2, p4]$
   $M_7 = [p1, 2p4]$
   Reachable terminal markings: $M_5 = [2p4]$
5. **Is there a reachable marking in which we have a nondeterministic choice?** There are two reachable marking in which we have a nondeterministic choice:
   $M_0 = [p1, p3, 2p4]$
   $M_1 = [p2, p3, p4]$
6. **Does the number of reachable markings increase or decrease if we remove**
   (1) place $p1$ and its adjacent arcs and
   (2) place $p3$ and its adjacent arcs?

   (1) Remove place $p1$ and its adjacent arcs. Reachable markings:
   $M_0 = [p3, 2p4]$
   $M_1 = [p2, p3, p4]$
   $M_2 = [2p3, 2p4]$
   $M_3 = [p2, 2p3, p4]$
   $M_4 = [3p3, 2p4]$
   $M_5 = [p2, 3p3, p4]$

...

The number of reachable markings are increase infinitely because after t1 and t2 are fired, new token will produce in p3.

(2) Remove place $p3$ and its adjacent arcs. Reachable markings:

$M_0 = [p1, 2p4]$

$M_1 = [p2, p4]$

$M_2 = [2p4]$

The number of reachable markings are decrease.

4 Practical simulation

4.1 Objective 1: Simulating specialists

Description:

a) Write down states and transitions of the Petri net NS. [1 point]

b) Represent it as a transition system assuming that:

(i) Each place cannot contain more than one token in any marking.

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

Solution:

a) The set of states of the Petri net: S = {free, busy, docu}

The set of transitions of net: T = {start, change, end}

b) i)

The transition system is $TS = \{S, TR, s_0\}$.

Since each place can not contain more than one token, the state space of all listable markings becomes: S = {[0, 0, 0], [1, 0, 0] , [0, 1, 0] , [0, 0, 1], [0, 1, 1], [1, 1, 0], [1, 0, 1], [1, 1, 1]}

The transition relation space accordingly becomes TR = {([1, 0, 0], [0, 1, 0]), ([0, 1, 0], [0, 0, 1]), ([0, 0, 1], [1, 0, 0]), ([1, 1, 0], [1, 0, 1]), ([1, 0, 1], [0, 1, 1]), ([0, 1, 1], [1, 1, 0])}. Notice how because of the given restriction, no transition relations like $([1, 1, 1], [2, 1, 0])$ are allowed.

The initial marking: s_0 = [1, 0, 0]

We know that the whole transition system covers a lot more than what the real system expresses. Hence, a reachability graph is hereby shown:

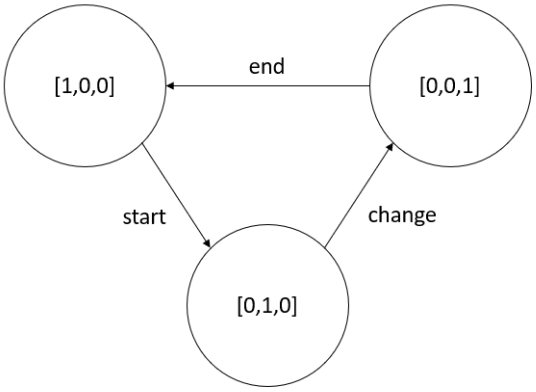

Figure 4.1.1

ii)

Let's look at an example first. Assume that the initial marking is $s_0 = M_0 =$ [2.free, busy, docu] (or [2, 1, 1] for short)

The transition system are $TS = \{S, TR, s_0\}$ with:

The state space of all possibly reachable marking S = { [2, 1, 1] , [1, 2, 1] , [2, 0, 2] , [3, 1, 0] , [0, 3, 1] , [1, 1, 2] , [2, 2, 0] , [3, 0, 1] , [0, 2, 2], [1, 3, 0] , [1, 0, 3] , [4, 0, 0] , [0, 1, 3] , [0, 4, 0] , [0, 0, 4] }

Transition relation

$TR = \{([2, 1, 1], [1, 2, 1]), ([2, 1, 1], [2, 0, 2]), ([2, 1, 1], [3, 1, 0]), ([1, 2, 1], [0, 3, 1]),$
$([1, 2, 1], [1, 1, 2]), ([1, 2, 1], [2, 2, 0]), ([2, 0, 2], [1, 1, 2]), ([2, 0, 2], [3, 0, 1]),$
$([3, 1, 0], [2, 2, 0]), ([3, 1, 0], [3, 0, 1]), ([0, 3, 1], [0, 2, 2]), ([0, 3, 1], [1, 3, 0]),$
$([1, 1, 2], [0, 2, 2]), ([1, 1, 2], [1, 0, 3]), ([1, 1, 2], [2, 1, 1]), ([2, 2, 0], [1, 3, 0]),$
$([2, 2, 0], [2, 1, 1]), ([3, 0, 1], [2, 1, 1]), ([3, 0, 1], [4, 0, 0]), ([0, 2, 2], [0, 3, 1]),$
$([0, 2, 2], [1, 2, 1]), ([1, 3, 0], [0, 4, 0]), ([1, 3, 0], [1, 2, 1]), ([1, 0, 3], [0, 1, 3]),$
$([1, 0, 3], [2, 0, 2]), ([4, 0, 0], [3, 1, 0]), ([0, 1, 3], [0, 0, 4]), ([0, 1, 3], [1, 1, 2]),$
$([0, 4, 0], [0, 3, 1]), ([0, 0, 4], [1, 0, 3])\}$

The initial marking $s_0 = [2, 1, 1]$

The reachability graph:

Back to the main problem, note that the number of states of the example reachability graph grows exponentially compared to the **i)** case just by a little tweak to the initial marking. And we also know the transition system is much larger than the reachability space. This sparks a need for a different method than naive listing to display the markings. Hence, an algebraic representation is born. We use variable x, y and z to specify the number of token in place free, busy and docu respectively with $x, y, z \in \mathbb{N}$

The state space $S = \{[\, x, y, z\,] \mid x, y, z \in \mathbb{N}\}$

Transition relation space $TR = \{((x+1,\ y,\ z),\ (x,\ y+1,\ z)) \mid x, y, z \in \mathbb{N}\}$
$\cup\ \{((x,\ y+1,\ z),\ (x,\ y,\ z+1)) \mid x, y, z \in \mathbb{N}\}$
$\cup\ \{((x,\ y,\ z+1),\ (x+1,\ y,\ z)) \mid x, y, z \in \mathbb{N}\}$

The initial marking is the same as the last case $m_0 = \{[1, 0, 0] \mid \forall m \in S\}$

By formalizing it this way, we can represent all possibilities of this net under the assumption that a place may contain an arbitrary number of tokens. In fact, the state space here is infinite and no listing can help us with the task.

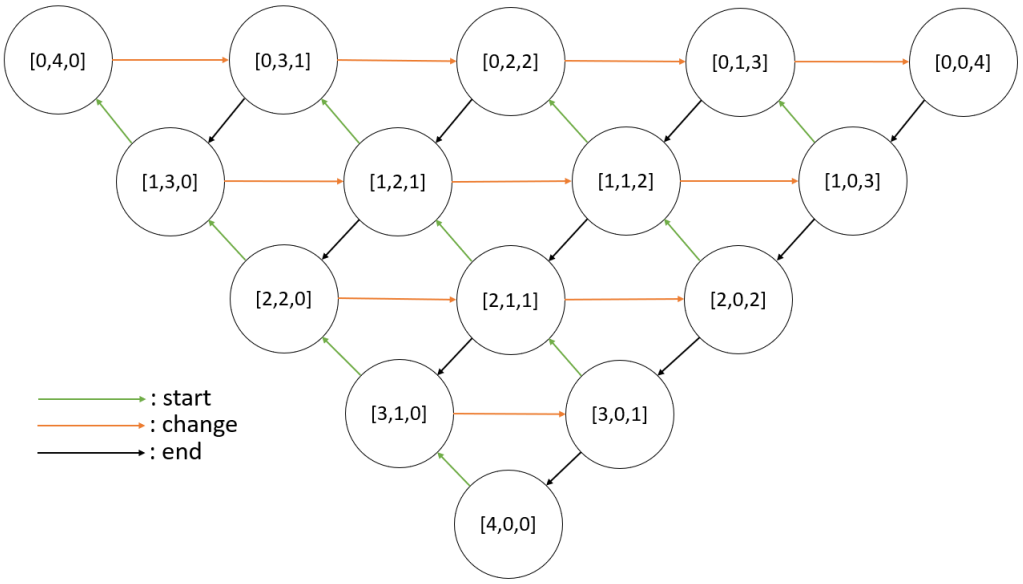

Figure 4.1.2

Provided that the net contains three places and three transitions forming a loop, let's assume the x, y, z symbols represent the numbers of tokens in the three places, respectively. When a firing occurs, a token that stands for a specialist would change his/her working status, making the marking transfer to any of these following states:

- Free and ready to treat another patient (the extra token is in place "free": $M = [x + 1, y, z]$).

- Busy treating a patient (the extra token is in place "busy": $M = [x, y + 1, z]$).

- Busy documenting the previous patient's status (the extra token is in place "docu": $M = [x, y, z + 1]$).

As already discussed, the transition system can not be listed due to its size, while the reachability graph remains the same because the initial marking stays unchanged with its reachability set. Therefore, this problem has been resolved completely.

4.2 Objective 2: Simulating patients

Description:

a) Explain the possible meaning of a token in state inside of the net N_{pa};

b) Construct the Petri net N_{pa}, assuming that there are five patients in state wait, no patient in state inside, and one patient is in state done.

Solution:

a) Token in state inside could mean patient is inside the room currently in treating by the specialist.

b) **Define N_{pa} as the Petri net modeling the state of patients:**

By the similar ideas of Petri net N_s, we need to define Petri net N_{pa} for modeling the state of patients.

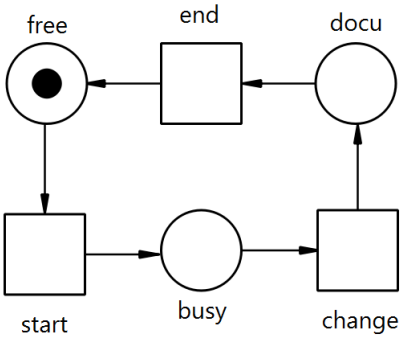

Figure 4.2.1: Petri net N_s - Modeling the state of the specialist

As we can see from Figure 4.2.1, there are three states of the specialist:

- The specialist is free and waits for the next patient ("free")
- The specialist is busy treating a patient ("busy")
- The specialist is documenting the result of the treatment ("docu")

Initially, transition "start" is enabled and when fired, the token in place "free" will be consumed and transferred to place "busy". Following that, the "change" transition now is enabled and fired, delivering the token from "busy" to "docu". Finally, the "end" transition is activated, which then finishes the cycle by moving the content of "docu" through to the initial working state, "free". From the above information of Petri net N_s, we can construct a Petri net N_{Pa} for the patients using similar idea with 3 states:

- The patient is waiting specialist ("wait")
- The patient is treated by the specialist ("inside")
- The patient has been treated by the specialist ("done")

For both nets N_s and N_{Pa}, we consider that transitions "start" and "change" represent two possible events. Figure 4.2.2 shows the basic Petri net N_{Pa}.

- In net N_s, the "start" transition leads to the "busy" place, while in net N_{Pa} for the patients the transition "start" leads to state "inside".
- In net N_s for the Specialist the transition "change" leads to state "docu", while in net N_{Pa} for the patients the transition "change" leads to state "done".

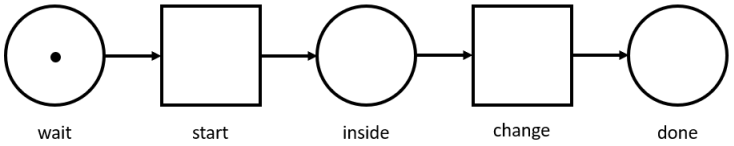

Figure 4.2.2: Basic Petri net N_{Pa}

Construct the Petri net N_{pa}:

According to the given assumtion that there are five patients in state wait, no patient in state "inside", and one patient is in state "done". We consider that there are two cases for this problem.

- Case 1: There are more than one patient can be treated at the same time.

  The Petri net $N_{Pa} = (P, T, F)$ with:

  $N_{pa} = (P, T, F)$ with:

  $P = \{\text{wait, inside, done}\}$

  $T = \{\text{start , change}\}$

  $F = \{(\text{wait, start}), (\text{start, inside}), (\text{inside, change}), (\text{change, done})\}$

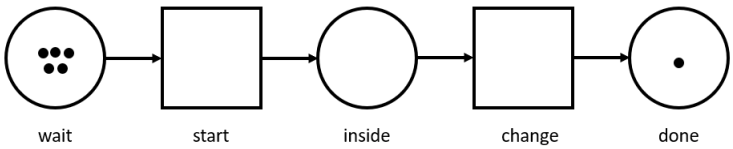

Figure 4.2.3

Shown in Figure 4.2.3 is the Petri net N_{Pa} for case 1, in which an arbitrary number of tokens can be consumed for firing at each place as long as it does not exceed the maximum cap in that place.

- Case 2: Only one patient can be treated at the same time.

  It seems more reasonable in this case, because the number of patients is restricted. But if we reuse the Petri net in Figure 4.2.3, we can not limit the number of consuming tokens in state "wait". So that we need to construct another N_{Pa} for more information.

  To ensure one token in place "wait" will be consumed when transition "start" fires, we can add two extra places, "free" and "occupied" to the net as control stations. Figure 4.2.4 demonstrates this idea.

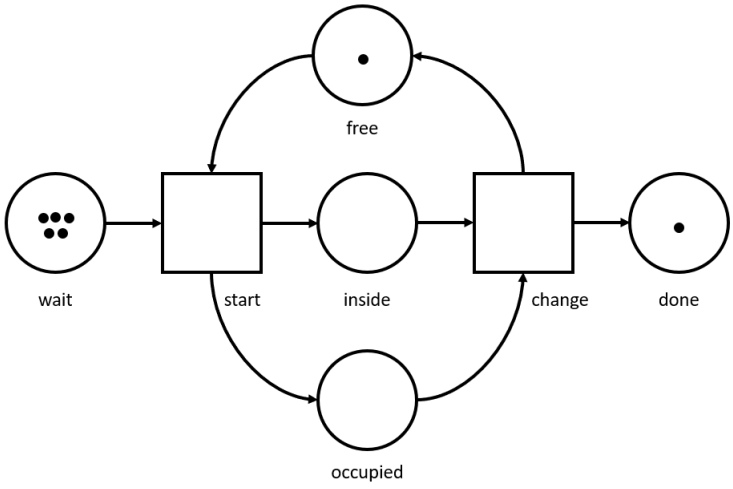

Figure 4.2.4

This way, transition "start" only enables when there exists a token in both states "free" and "wait". When transition "start" fires, one token in place "wait" (one patient) and one token in place "free" are consumed and produced in places "inside" and "occupied", respectively.

Now, there is no token in the "free" state, so the "start" transition is disabled and can not fire. That means only one patient can be treated at that time.

When that patient finishes his/her treatment session, transition "change" will fire, consume one token from each of "inside" and "occupied", and transfer them to "done" and "free"

accordingly. Now, if there are waiting patients left (tokens in the "wait" state), the system will loop again till the waiting room is empty and the doctors are free.

We define this Petri net $N_{pa} = (P, T, F)$ with:

P = {wait, inside, done, free, occupied}

T = {start, change}

F = {(wait, start), (start, inside), (inside, change), (change, done), (free, start), (start, occupied), (occupied, change), (change, free)}

4.3 Objective 3: A complete model

Description: Determine the superimposed (merged) Petri net model $N = N_S \oplus N_{Pa}$ allowing a specialist treating patients, assuming there are four patients are waiting to see the specialist/ doctor, one patient is in state done, and the doctor is in state free. (The model then describes the whole course of business around the specialist).

Solution:

We have seen in our solution to **Objective 2** a method that helps control the flow of patients through the medical center, once at a time. We added two fixed places, "free" and "occupied", to represent the state of work inside the clinic and wire them around the transitions "start" and "change", as suggested in Figure 4.2.4. This behavior is analogous to a doctor accompanying their patient into the clinic, and thus we can generalize the Petri net in Figure 4.2.4 to account for the specialists and the patients simultaneously. In details, the "occupied" state resembles an on-duty interval, or the "busy" state in the specialists net. So we will not change but rather rename it to fit the context. On the other hand, the "free" itself does not encompass the extra procedure named "docu" that every specialist has to perform after an examination on a patient. Thus, we replace that single place with a place - transition - place sequence, i.e. "Free" ⟵ "End" ⟵ "Docu", respectively. The result is captured in Figure 4.3.1.

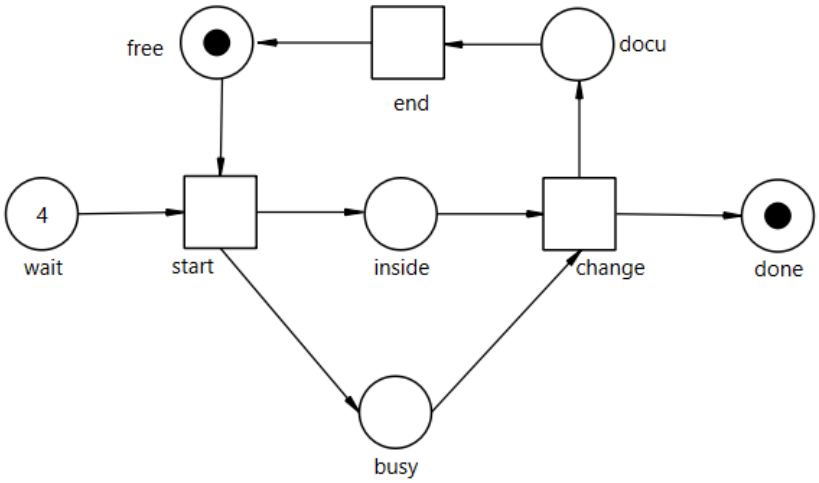

Figure 4.3.1

4.4 Objective 4: Simulating the complete model

Description: Consider an initial marking M_0 = [3.wait, done, free] in the grand net $N = N_S \oplus N_{Pa}$. Which markings are reachable from M_0 by firing one transition once? Why?

Solution:

We have M_0 = [3.wait, 0.inside, done, free, 0.busy, 0.docu] = [3, 0, 1, 1, 0, 0]

Initial marking M_0 has a token in place "wait" and "free", therefore transition "start" is enabled as the patient enter the room and the specialist ready to treat him (or her):

$$(N, M_0)[\text{start}\rangle(N, M_1 = [2, 1, 1, 0, 1, 0])$$

After the first firing, marking M_1 is reached with a new token in place "inside" and "busy", thus the treatment is in process, enabling transition "change":

$$(N, M_1)[\text{change}\rangle(N, M_2 = [2, 0, 2, 0, 0, 1])$$

At this marking, the patient has been treated and gone to place "done", while the specialist is now documenting the result, enabling transition "end":

$$(N, M_2)[\text{end}\rangle(N, M_3 = [2, 0, 2, 1, 0, 0])$$

As the specialist is now done with the first patient and is free (token in place "free"), he (or her) continues the treatment for the other two patients (the remaining two tokens in place "wait"):

$$(N, M_3)[\text{start}\rangle(N, M_4 = [1, 1, 2, 0, 1, 0])$$

$$(N, M_4)[\text{change}\rangle(N, M_5 = [1, 0, 3, 0, 0, 1])$$

$$(N, M_5)[\text{end}\rangle(N, M_6 = [1, 0, 3, 1, 0, 0])$$

$$(N, M_6)[\text{start}\rangle(N, M_7 = [0, 1, 3, 0, 1, 0])$$

$$(N, M_7)[\text{change}\rangle(N, M_8 = [0, 0, 4, 0, 0, 1])$$

$$(N, M_8)[\text{end}\rangle(N, M_9 = [0, 0, 4, 1, 0, 0])$$

Throughout the whole process, there is no two marking with the exact same number of token on each place as the number of waiting patients as well as the number of patients done getting treated change, thus resulting in 10 different total marking.

At marking M_9, the specialist is free but there is no patient in waiting anymore, so not enabling any transition. We call M_9 the terminal marking.

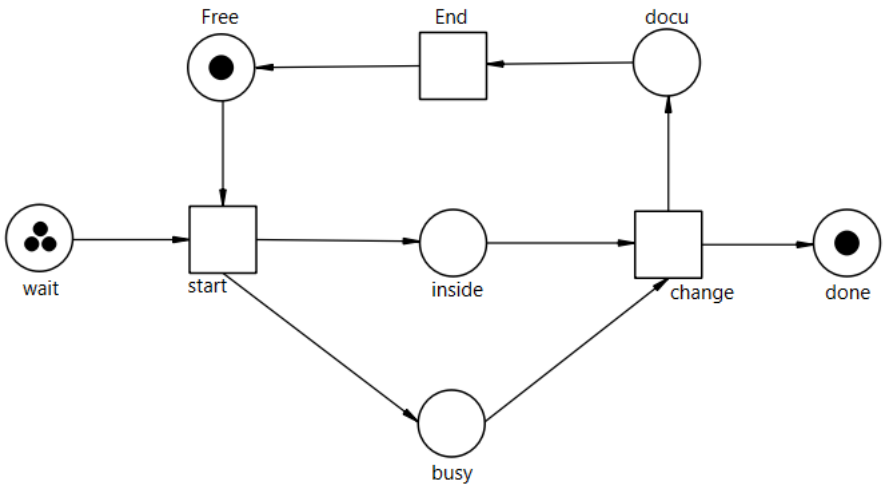

Figure 4.4.1

4.5 Objective 5: An important property

Description: Is the superimposed Petri net M deadlock free? Explain properly.

Solution:

A marked Petri net (N, M_0) is **deadlock free** if at every reachable marking at least one transition is enabled. Formally, for any $M \in [N, M_0\rangle$ there exists a transition $t \in T$ such that $(N, M)[t\rangle$.

In this exercise we have the superimposed Petri net $N = N_S \oplus N_{Pa}$ with an initial marking $M_0 =$ [4.wait, done, free]. Figure 4.5.1 demonstrate that:

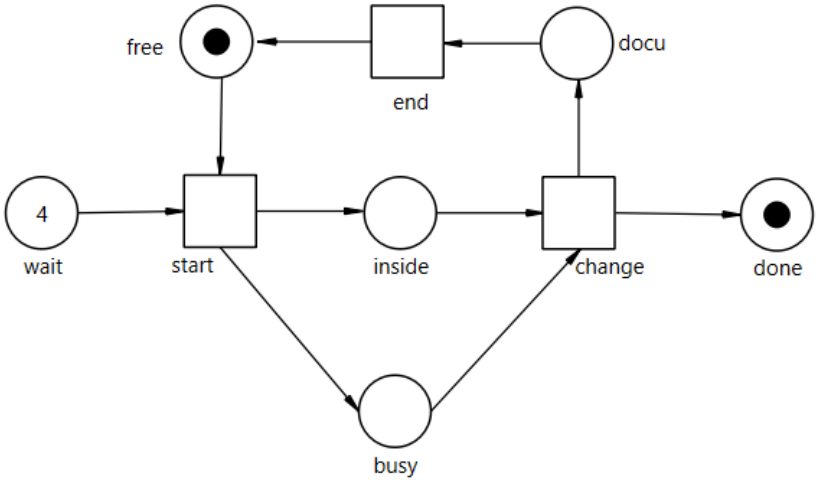

Figure 4.5.1

At the beginning, the transition "start" is enabled. Firing at this transition, one token in "wait" and "free" will be consumed and produced in "inside" and "busy" state, respectively.

$$(N, M_0)[\text{start}\rangle(N, M_1 = [3.wait, inside, busy, done])$$

Then, the transition "change" is enable and firing. One token in "inside" and one in "busy" will be consumed and produced in "done" and "docu" state, respectively. Now, the number tokens in "done" increase one.

$$(N, M_1)[\text{change}\rangle(N, M_2 = [3.wait, docu, 2.done])$$

The transition "end" firing and we get the marking:

$$(N, M_2)[\text{end}\rangle(N, M_3 = [3.wait, free, 2.done])$$

At this time, the token is produced in "free" state, so the "start" transition is enabled again. So we have the similar procedure.

$$(N, M_3)[\text{start}\rangle(N, M_4 = [2.wait, inside, busy, 2.done])$$

$$(N, M_4)[\text{change}\rangle(N, M_5 = [2.wait, docu, 3.done])$$

$$(N, M_5)[\text{end}\rangle(N, M_6 = [2.wait, free, 3.done])$$

According to these above description, we consider that the petri net N play a role as a "pump" to transport tokens from "wait" to "inside" and from "inside" to "done". It leads to a result that the number of tokens from "wait" after a procedure will decrease by one token to "done". So that, everytime "wait" decreases one token, it will no longer have tokens leading to the "start" transition is not enabled at this time. So we have a direction of these tokens in the Figure 4.5.2:

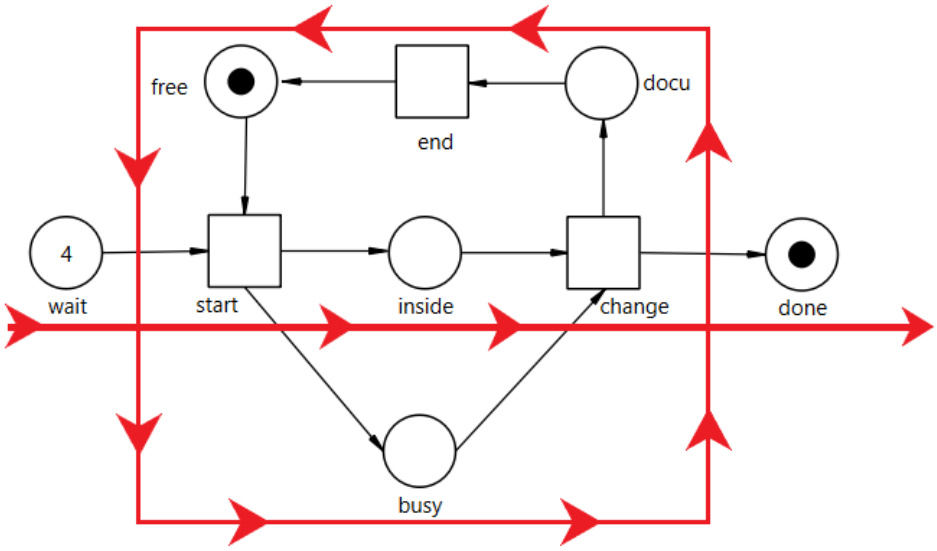

Figure 4.5.2

In practial problems, we can see that the number of patients is not infinite, so that the number of patient will be empty, at this time, these specialists also will not do their task. And the above model simulated accurately.

Continue the above example, the Petri net N will have the terminate marking, $M = [5.\text{done}, \text{free}] \in [N, M_0\rangle$ and at this time there does not exist a transition is enabled.

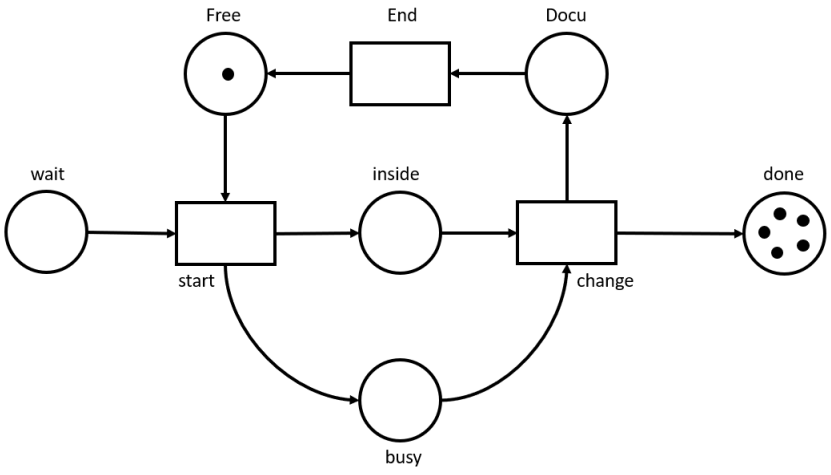

Figure 4.5.3

At this marking M, there is no token in "wait" state so transition "start" is not enabled and does not fire. The "inside" and "busy" state are not have token so the "change" transition is not enabled and so on the "End" transition is the same. Therefore, this Petri net N is not deadlock free.

In another way, when the number of patient waiting for specialist equal zero, this Petri net N will stop. So that it is not deadlock free.

4.6 Objective 6: An alternative structure

Description: Propose a similar Petri net with two specialists already for treating patients, with explicitly explained construction.

Solution:

In this exercise, the number of specialists is two. If we use the basic Petri net like the previous exercises, it is not sufficient for us to distinguish the data between each token in the same place. So, we need to use Colored Petri net in this exercise for extending with color and time.

A colored Petri net is a Petri net in which every place has a type, and every token has a value (color) that complies to the place type. An arc in a colored Petri net may have an arc inscription which is an expression with some variables that evaluate to a multi set. A transition can have guard which is a Boolean expression, and it may have variables in exactly the same way than arc inscriptions have.

4.6.1 Determining place types

Firstly, we need to set the value for the token in place "wait" represents a waiting patient, and token in place "free" represents a specialist already treating patients.

Patient = patientID x name x dateOfBirth x gender.
Specialist = specialistID x name x exYear.

For example:

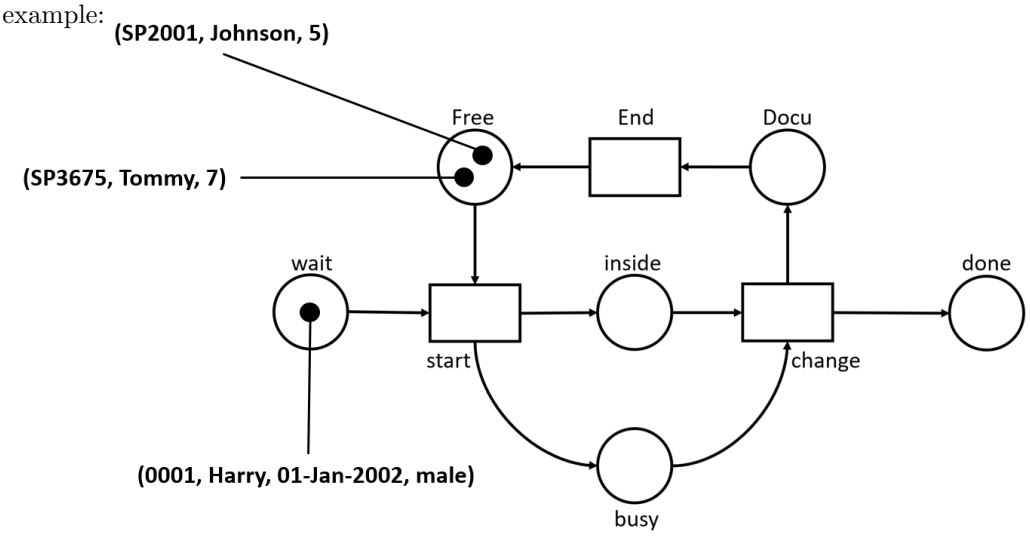

Figure 4.6.1

As we can see in Figure 4.6.1, a token in place "wait" has a value (0001, Harry, 01-Jan-2002, male). This value represents the information of the patient. This patient name's Harry, ID number is 0001, Harry is male and was born on 01-Jan-2002.

Similarly, two tokens in place "free" have these value (SP2001, Johnson, 5), (SP3675, Tommy, 7). The first value represents the information of the first specialist. His name is Johnson, specialist ID is SP2001, and he has 5 years of experience. And it's similar to the second specialist.

4.6.2 Arc inscription and Guard

Figure 4.6.2 depicts a colored Petri net net of the patient and specialist extended with arc inscription and guard. The arc inscription contains two variables x and y. Variable x is of type patient and variable y is of type specialist.

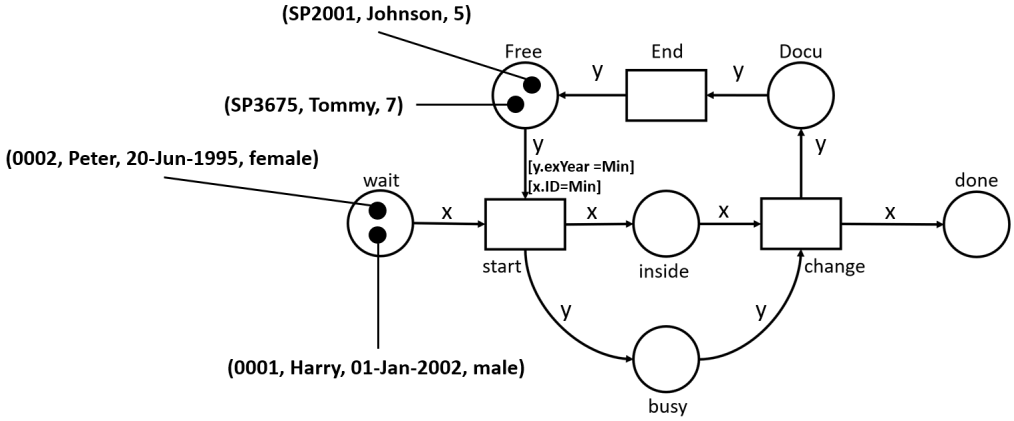

Figure 4.6.2

To distinguish the order of the patient, specialist, we need some additional conditions. So, we can use a transition guard. In Figure 4.6.3, the guard of transition "Start" define at two constraints:

[y.exYear = Min]: It specifies that the value of exYear in variable y must be minimum.

[x.ID = Min]: It specifies that the value of patientID in variable x must be minimum.

It also means that the patient whose ID is smallest will be treated first and the specialist whose experiment year is higher will treat the patient later.

4.6.3 Time stamp and delay

We must extend the time stamp and the delay for the Colored Petri net in Figure 4.6.2, so that it is more systematic. Figure 4.6.3 depicts the colored Petri net with time-stamp and delay extending.

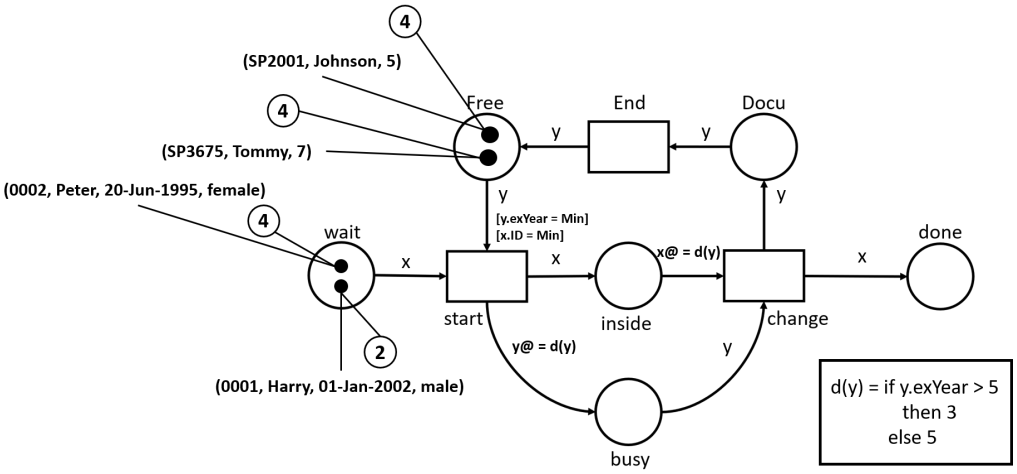

Figure 4.6.3

The tokens in place "wait" with time stamp 2 represents a patient who arrives at time 2 and similar for time stamp 4. The tokens in place "free" with time stamp 4 specifies that two specialists are available at time 4.

The delay is noted by '@' sign. There are two delay time at two arc inscriptions:

$$y@ = d(y),\ x@ = d(y)$$

It depends on $d(y)$, that is, the Experience year of the specialist. If the Experience Year is greater than 5, the delay is equal to 3-time units. Otherwise, the delay is equal to 4-time units.

4.6.4 Construct the clored Petri net

Reusing all the data from subsection 1, 2, 3 and adding one patient (adding one token in place "wait") we have Figure 4.6.4 and Table 1.

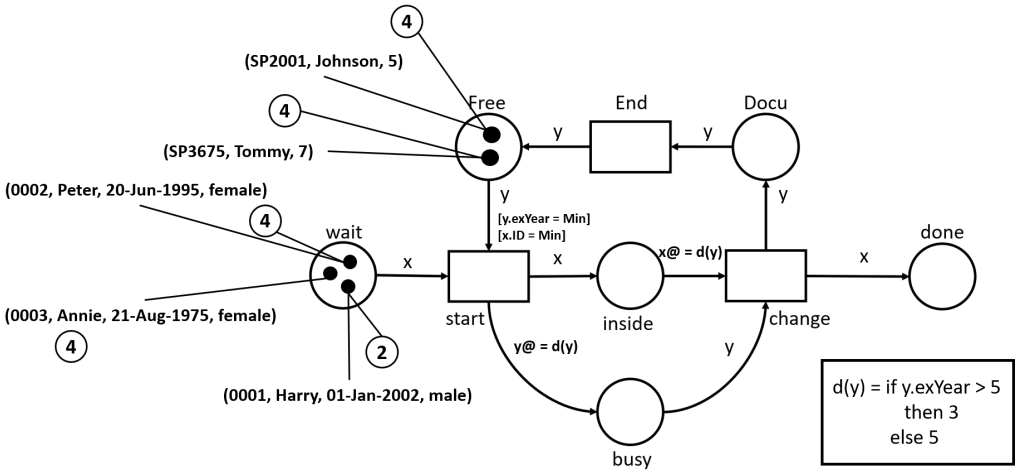

Figure 4.6.4

The starting point is the marking of Table 1. Transition Start is enabled, with enabled time for the first pair $(x, y) = ((0001, Harry, 01-Jan-2002, male), (SP2001, Johnson, 5))$ equal 4 and second $(x, y) = ((0002, Peter, 20-jun-1995, female), (SP2001, Johnson, 5))$ equal 4. Firing transition "Start" produces two tokens in place "inside" and two tokens in place "busy".

Place	Time stamp	Value
wait	2	(0001, Harry, 01-Jan-2002, male)
	4	(0002, Peter, 20-jun-1995, female)
	4	(0003, Annie, 21-Aug-1975, female)
Free	4	(SP2001, Johnson, 5)
	4	(SP3675, Tommy, 7)
Busy		
Inside		
Docu		
Done		

Table 1

Two tokens in place busy have the delay, the first token $((SP2001, Johnson, 5))$ has $exYear = 5$ so the delay equals 4-time units. And the second token $((SP3675, Tommy, 7))$ has $exYear = 7$ so the delay equals 4-time units. Table 2 shows the result marking.

Place	Time stamp	Value
wait	4	(0003, Annie, 21-Aug-1975, female)
Free		
Busy	7	(SP3675, Tommy, 7)
	8	(SP2001, Johnson, 5)
Inside	4	(0001, Harry, 01-Jan-2002, male)
	4	(0002, Peter, 20-jun-1995, female)
Docu		
Done		

Table 2

Now, transition "Change" fires, at tokens in place "Inside" have the delay equal $d(y)$, so the first tokens $((0001, Harry, 01 - Jan - 2002, male))$ has delay equal 4 and the second token $((0002, Peter, 20 - june - 1995, female))$ has the delay equal 3. Table 3 shows the result marking.

Place	Time stamp	Value
wait	4	(0003, Annie, 21-Aug-1975, female)
Free		
Busy		
Inside		
Docu	7 8	(SP3675, Tommy, 7) (SP2001, Johnson, 5)
Done	7 8	(0002, Peter, 20-jun-1995, female) (0001, Harry, 01-Jan-2002, male)

Table 3

After that, transition "End" fires, and table 4 shows the result making. Then, transition "Start"

Place	Time stamp	Value
wait	4	(0003, Annie, 21-Aug-1975, female)
Free	7 8	(SP3675, Tommy, 7) (SP2001, Johnson, 5)
Busy		
Inside		
Docu		
Done	7 8	(0002, Peter, 20-jun-1995, female) (0001, Harry, 01-Jan-2002, male)

Table 4

fires again, and the token $((SP2001, Johnson, 5))$ will be consumed because the $exYear = 5$ is minimum. The pair $(x, y) = ((0003, Annie, 21-Aug-1975, female), (SP2001, Johnson, 5))$. The delay in the "busy" place is equal 4. Table 5 shows the result marking.

Place	Time stamp	Value
wait		
Free	7	(SP3675, Tommy, 7)
Busy	12	(SP2001, Johnson, 5)
Inside	8	(0003, Annie, 21-Aug-1975, female)
Docu		
Done	7 8	(0002, Peter, 20-jun-1995, female) (0001, Harry, 01-Jan-2002, male)

Table 5

Transition "Change" fires and the delay of token $((0003, Annie, 21 - Aug - 1975, female))$ equal 4. Table 6 shows the result.

Place	Time stamp	Value
wait		
Free	7	(SP3675, Tommy, 7)
Busy		
Inside		
Docu	12	(SP2001, Johnson, 5)
Done	7 8 12	(0002, Peter, 20-jun-1995, female) (0001, Harry, 01-Jan-2002, male) (0003, Annie, 21-Aug-1975, female)

Table 6

And the terminate marking is shown in Table 7.

Place	Time stamp	Value
wait		
Free	7 12	(SP3675, Tommy, 7) (SP2001, Johnson, 5)
Busy		
Inside		
Docu		
Done	7 8 12	(0002, Peter, 20-jun-1995, female) (0001, Harry, 01-Jan-2002, male) (0003, Annie, 21-Aug-1975, female)

Table 7

5 Objective 7: Implementing a graphic package

5.1 Introduction

In the previous sections, we have constructed and simulated a practical business process in form of a Petri net. It is now logical that we further our research and aim for a more general method to simulate an arbitrary Petri net, not just any specific models. In this section, our team shall present a graphic package[1] that supports visualizing Petri nets and performs computations to analyse them.

Visualization is a great tool to demonstrate Petri net, which plays an important role in process mining, data mining, chemistry, etc. In a student's perspective, implementing a visual representation of Petri net can help facilitate understanding of the subject, deconstruct any flow work structure, and tackle relevant practical problems such as business modelling, chemical processes, ... or solving reachability, boundedness problems and many more yet to be covered. With that vision in mind, we hope that this section could serve as an excellent addition to the report, not merely an answer to a problem, but a little journey to give a taste about our research into the GUI[2] realm, which entails interesting concepts and advanced topics of Graph theory, and of course more on Petri net. As a result, this section might stand out as a separate report because of its length. Reviewers might ignore some discussions irrelevant to the main Petri net topic. We also deliver the build up in a less formal tone compared to other sections in pursuit of better presentation for tough subjects like Graph and Process mining.

5.2 Description

Write a computational package to realize (implement) Items 1,2,3 and 4 above. You could employ any programming language (mathematical- statistical like R, Matlab, Maple, Singular, or multi-purposes language like C or Python), that you are familiar with, to write computational programs. The programs of your package should allow input with max 10 patients in place wait.

5.3 Framework

Our implementation was done in `C++`, using the Qt Framework. Since `C++` is the main programming language throughout our university lives, it is reasonable to keep using the language in advanced projects and to hop on a framework that fully supports both `C++` and Graphics, in this case Qt, with their rich set of visualization tools. We shall now delve into the details of Qt and introduce some features crucial to our implementation.

[1] Graphic packages are application software that can be used to create and manipulate images on a computer

[2] The graphical user interface, or GUI, is a form of user interface that allows users to interact with electronic devices through graphical icons and audio indicator such as primary notation, instead of text-based user interface, typed command labels or text navigation

Qt

Qt is a cross-platform application development framework for desktop, embedded and mobile. It consists of several key modules called Qt Essentials that provide complete tools for building GUIs. Also, the framework goes with its famous Integrated Development Environment called Qt creator IDE, which we utilized to implement the graphic package.

Qt is not a programming language, but a framework that supports `C++`. For this reason, every key feature of `C++` is maintained and enhanced further in Qt, including Object-oriented programming (OOP) model. Objects in Qt are stored hierarchically, meaning each object has a parent, or some hook like a screen. When we go out of scope, the parent object is freed in sequence with its children. So one does not have to worry about destructor calls during coding session with Qt.

Figure 5.3.1: Qt logo

QWidget and QObject

QWidget is an important module that provides all sorts of required interface items, or widgets, to manage a full-fledged GUI with menubar, icons, main view, docks and many more inside an application. We use its child classes, of which noteworthy are QMainWindow and QWidgetItem, to manifest our software interface, as shown in Figure 4.3.1.

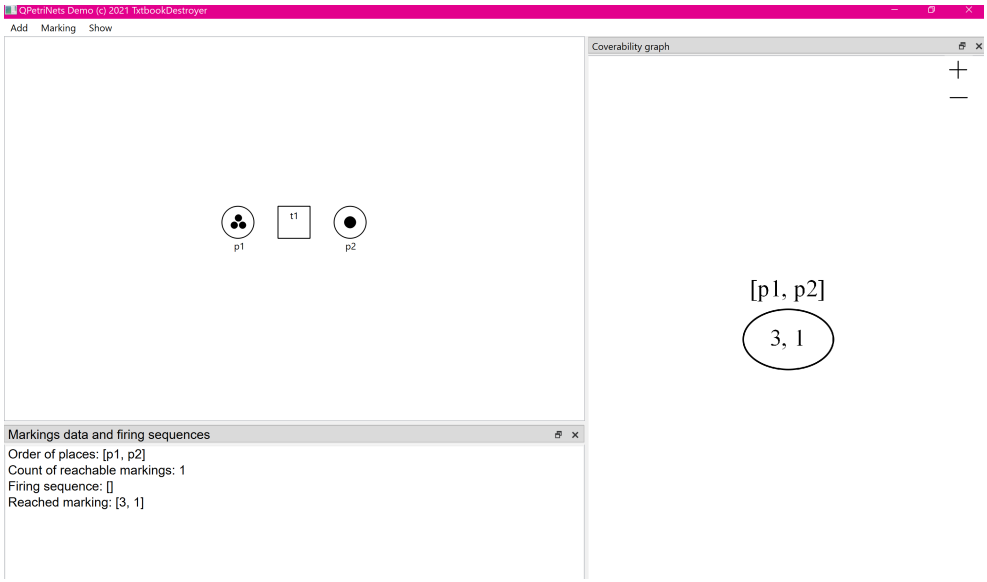

Figure 5.3.2: Petri net GUI

However, onscreen widgets are just static images. They do not have functionalities yet. Therefore, we have to, by some means, assign to these widgets duties and animations to give life to the interface. Qt Framework provides the QObject module which helps track onscreen events such

as mouse movement and clicking. It also allows for SIGNAL and SLOT, a distinctive mechanism that only Qt possesses. Using SIGNAL and SLOT, we can trigger certain functions (SLOTs) upon some events (SIGNALs). For instance, when we click a selectable object, Qt would signal a change in the system and call for functions that link with that change. These functions then manifest their effect on the interface by animation, like shading the area of the object, or popping up a declarative window. This is one of the most important feature of Qt GUI framework. In our implementation, the SLOT and SIGNAL mechanism plays a huge role in invoking animation upon interactions between users and the screen, mostly for clicking utility and object manipulation. We will find out more about the options one has with the application in the next few sections.

Qt Graphics View Framework

Beside the interface widgets, we also need to manage graphical items that users can create and interact with. These items belong to another framework called the Qt Graphics View Framework, of whose sub-modules we chose `QGraphicsView`, together with `QGraphicsScene` as our rendering engine. `QGraphicsScene` is a class which provides a surface for managing a large number of 2D graphical items. It serves as a container for `QGraphicsItem`, which can be visualized together with `QGraphicsView`.

`QGraphicsItem` is the base class for all graphical items in a `QGraphicsScene`. It provides a lightweight foundation for implementing custom items. That includes methods for altering their geometry, collision detection and more. The shape can be altered by re-implementing their paint function.

The main use of the rendering engine is to create graphical items (which inherits from a `QGraphicsItem`) and add them to the scene with the `QGraphicsScene.addItem(QGraphicsItem*)` method. Once the scene is in possession of some item, all of its rendering will be handled by the scene. That includes repainting when necessary. The item in question must have a paint method (usually overridden by the user) which will be used by the scene when a repainting is scheduled.

If the user wants to alter the visual representation of the item, they must call the `QGraphicsItem.update()` method. This will schedule the item for repainting. In case the shape of the item is about to be altered, the method called `QGraphicsItem.prepareGeometryChange()` must be called. This is to ensure the whole item is repainted, since `QGraphicsView` does not repaint the whole area the items cover, if it is not necessary.

Another noteworthy point is that graphical items also belong to the parent-child hierarchy. It allows the inherited items to update their geometry with respect to their parent position. This is something called relative coordinate system in Qt.

5.4 Petri net Objects

In this section, we will detail our method of representing a Petri net items graphically.

Place and Transition

We follow conventions and shape places as circles and transitions as squares. Their containers are of `QGraphicsPathItem`, a subclass of `QGraphicsItem`. This class allows to represent graphical items in form of paths that you can draw on and fill in color via the paint method. It also provides methods like `setPath()` and `addShape()` to adjust the path of a graphic item, thereby affecting the updated drawing. As a result, with Qt we can easily carry out geometric changes on a place or transition when inserting tokens or moving objects.

Gate and Connection

An important aspect of Petri net is the connection between objects. In our software, we use a directed arrow to connect one place and one transition, and only place-transition type connections exist. A problem arises as we have to determine which point on each object to set the arrow hooks. This is mainly because we render items with 2D geometry, which theoretically possess infinite points on their boundaries. To handle this, we introduce the concept of gate, or port. On each place and transition, we define a set of convenient points to nail the arrows. These points are called gates. Because each place has circular shape, we distribute 8 equally-spaced gates around its boundary, as shown in Figure 5.4.1.

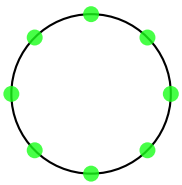

Figure 5.4.1

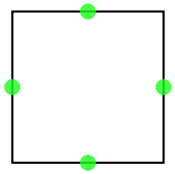

Figure 5.4.2

For transitions, we opt for 4 gates around their square shape boundary, shown in Figure 5.4.2. Moreover, gates are children objects of places and transitions. With the invention of gate, connections go from joining places and transitions to only linking gates of different parent type together. When the mouse hovers close to the position of a gate, its content glows green just like the above demonstrations to signal a selectable and draggable connection. One can finish the linking by releasing the mouse at the position of a qualified gate, otherwise the updated connection disappears. A complete connection between a place and a transition is shown in Figure 4.4.3.

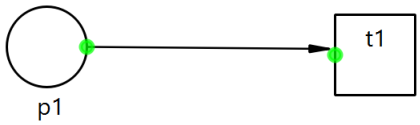

Figure 5.4.3

At the onset of our program, we do not label and assign weights to connections, thus implicitly conveying that connection weight is always 1 and firing takes up only a single token at each input place. Weight feature might be updated in the future to bring about more practical value.

Interaction

Now that we have established the graphic representations for places, transitions and their connections. The question is then how to create such objects? In the software, we provide a menu option named “Add” at the top-left corner of the application window.

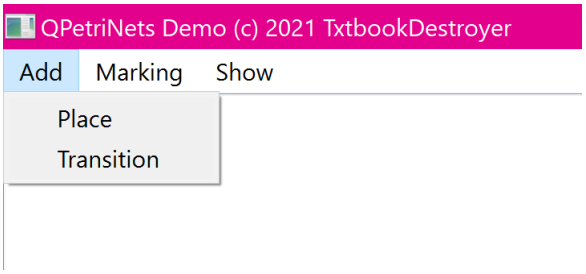

Figure 5.4.4

Clicking the push button, a list of objects appears, which includes "place" and "transition". If one go with "place", the following dialog pops up and prompt him/her to enter the place's name and its number of tokens.

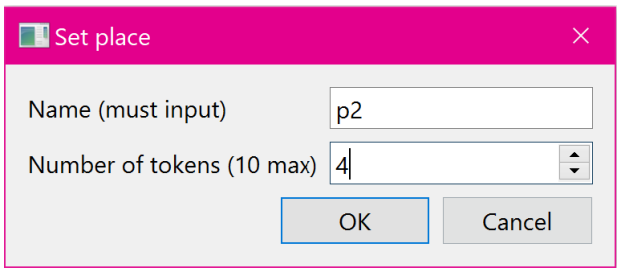

Figure 5.4.5

Similarly, if the user chooses "transition", another dialog pops up but this time it only requires a name, since a transition does not store tokens.

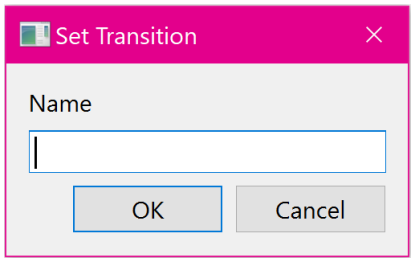

Figure 5.4.6

After the user has inputted, the GUI will manifest the objects on screen with the accurate labels and tokens (if any). One can also double click on the objects to change their information.

Firing

A graphic package can not lack its utmost delectable flavor, animation. Our program provides interaction between users and artificial Petri nets, with the most prominent feature known as firing simulation. In detail, if a transition is enabled, its occupied area becomes clickable. When one presses the mouse on an enabled transition, a firing animation carries out, which transfers tokens from one or several places, through that transition, to reach some output places. The following video captures the firing animation of one Petri net example.

The underlying work is up to the `QAnimation` module, whose objects can be assigned to perform specific animation on some `QGraphicsItem` when receiving SIGNALs. Moreover, firing in the program occurs sequentially, partly to not mess up with the animation, whereas concurrent firing is not our main interest here.

5.5 Analyzing graphical Petri nets

Having constructed a fine GUI with notable features, our next goal was to implement some algorithm that helps analyse any graphical Petri net. Our general approach was:

1. Compute and generate onscreen data about reachable markings and their firing sequences.
2. Display reachability graphs of Petri nets.
3. Perform superimposition on any two arbitrary Petri nets.

Of the three stated goals, we achieved the first two and intend to develop on the third one as our future project. We hereby present our ideas for each goal and how we set about implementing them.

5.5.1 Preliminaries

Reachability Graph

In the first goal, to generate data about reachable markings and their firing sequences, our intention was to perform a graph traversal, e.g. BFS, DFS, etc. on the equivalent reachability graph and acquire the markings data. Our task thus boils down to a main quest that relates directly to our second goal, i.e. obtaining a reachability graph. Solving this first shall help us hit two birds with one stone.

The difficulty with constructing a reachability graph is that we do not know whether it is viable to do so. This has been shown in **Section 2**, that a reachability set would be infinite if its associated Petri net $N = (P, T, F, M_0)$ is unbounded, i.e. $\forall n \in \mathbb{N}, \exists M \in [N, M_0\rangle$ and $\exists p \in P : M(p) > n$. To resolve the problem, we introduce a new notion *Coverability Graph*, which is a finite approximation of *Reachability Graph*. Before delving into details, more key terminologies are necessary.

Extended natural set

To make a reachability graph finite, it is necessary to restrict the representation of unbounded markings to a broad, convenient notion. This is where the name *extended natural set* comes in, because we will add to it an arbitrarily large element.

Definition 4.1. Extended natural set

Given $\mathbb{N}$ as the set of all natural numbers, including 0, then the set $\mathbb{N}_\omega \overset{\text{def}}{=} \mathbb{N} \cup \{\omega\}$ is an extension of $\mathbb{N}$, which not only broadens the latter's size, but also its operations. With natural numbers of their original set, the operations rules do not change. In the presence of ω, for some $p, q \in \mathbb{N}, q \geq 1$:

$$\omega + \omega \overset{\text{def}}{=} \omega$$
$$\omega - \omega \overset{\text{def}}{=} \omega$$
$$p - \omega \overset{\text{def}}{=} 0$$
$$0 \cdot \omega \overset{\text{def}}{=} \omega \cdot 0 \overset{\text{def}}{=} 0$$
$$\omega \pm p \overset{\text{def}}{=} \omega$$
$$\omega \cdot q \overset{\text{def}}{=} q \cdot \omega \overset{\text{def}}{=} \omega$$

The ordering on $\mathbb{N}_\omega$ is also updated by defining $n \leq \omega$ for all $n \in \mathbb{N}$.

Extended multi-set

Having obtained extended natural set, it is logic that we extend the notion of *multi-set* next to account for unbounded markings, since markings are multi-sets.

Definition 4.2. Extended multiset

An *extended multiset* M over a finite set P is a function $M : P \to \mathbb{N}_\omega$. Any subset of P may be viewed as as a multiset over P, and a multiset may always be considered as an extended multiset. For some $p \in P$, we write $p \in M$ iff $M(p) \geq 0$. For two extended multisets M_1 and M_2 over P, we write $M_1 \leq M_2$, or M_2 *covers* M_1, if $\forall p \in P : M_1(p) \leq M_2(p)$. Also, $M_1 < M_2$ if $M_1 \leq M_2$ and $M_1 \neq M_2$.

We shall then define over *extended multiset* basic operations like addition $(+)$, subtraction $(-)$ and scalar multiplication $(\cdot)$, respectively. Formally, $\forall p \in P$ and two extended multisets M_1 and M_2 over P:

$$(M_1 + M_2)(p) \stackrel{\text{def}}{=} M_1(p) + M_2(p)$$
$$(M_1 - M_2)(p) \stackrel{\text{def}}{=} max\{0, M_1(p) - M_2(p)\}$$
$$(n \cdot M_1)(p) \stackrel{\text{def}}{=} n \cdot M_1(p),\ n \in \mathbb{N}$$

Notice how these definitions are almost the same with that of normal multiset in **Section 2**. The exception we made is the range of the multiset, N_ω. Lastly, to properly display markings, we decide that for every set P that contains place names, P is sorted alphabetically. If we denote all elements in P as $p_1, p_2, ..., p_n$, its associated marking is $[M(p_1), M(p_2), ..., M(p_n)]$, for some $M \in \mathcal{M}$. This representation is consistent with what we meant by choosing to write "$p \in M$ iff $M(p) \geq 0$" and not ">", previously. Since we know the place set P already, readability would be enhanced significantly should we have P sorted and omit place names when dumping the data onscreen. In fact, displaying markings like suggested helps avoid the conventional naming redundancy of writing $p_1^{M(p_1)}$, $p_2^{M(p_2)}$, etc. at the expense of extra 0s, e.g. $[1, 0, 0, 0, 1, 0]$.

Monotonicity

The word *monotonicity* itself refers to an unvarying condition. In the context of Petri net, it implies the preservation of behaviors when adding tokens to the system. Formally, given M and M' are some markings of a Petri net $\mathcal{N}$ and a firing sequence $\sigma \in T^*$ such that $M[\sigma\rangle M'$, then by *monotonicity*, $(M + \zeta)[\sigma\rangle(M' + \zeta)$, therein ζ is an arbitrary extended multiset over $P_\mathcal{N}$. Increasing the number of tokens in all places can only increase the number of enabled transitions, which is why this condition is associated to the word *monotonicity*.

The property essentially implies that in a transition system where there exist some markings M and M' satisfying $M < M'$, its reachablility set becomes unbounded if we fire the sequence from M to M' infinitely many times. In other words, let σ denote the sequence that satisfies $M[\sigma\rangle M'$, then $M[\sigma^k\rangle(M' + (k-1)(M' - M))$ for any $k \in \mathbb{N}^*$. Further assuming $L_k = kM' + (k-1)M$, then $\forall p \in P_\mathcal{N}, \forall k \in \mathbb{N}^* : L_k(p) \geq M(p)$. Also, $\forall n \in \mathbb{N}, \exists k_0 \in \mathbb{N}^*$ and $\exists q \in P_\mathcal{N} : L_{k_0}(q) > n$. We can thus label the marking at q as ω, meaning q is unbounded, hence the logic behind *Coverability Graph*.

Coverability Graph [1]

Definition 4.3. Coverability graph

A *Coverability Graph* $CG = (V, A, \mu, v_0)$ for a Petri net $\mathcal{N} = (P, T, F, M_0)$ has a set of nodes V, with v_0 being the intial node. A is the set of directed arrows with labels in T. μ is a function that assigns an extended multiset to each vertex as a label. A t-labelled arc from u to v is denoted as $u \xrightarrow{t} v$. We also write $u \rightsquigarrow_A v$ to indicate that there is a path/sequence from u to v with labels in A.

To construct a coverability graph, **Algorithm 1** is given.

Algorithm 1: Construct a CG of a Petri net $\mathcal{N}$

Input : $\mathcal{N} = (P, T, F, M_0)$
Output: $CG = (V, A, \mu, v_0)$

1 $\mu[v_0] \longleftarrow M_0$
2 $(V, A) \longleftarrow (\{v_0\}, \varnothing)$
3 $U \longleftarrow \{v_0\}$ ▷ Add v_0 to queue
4 **while** $U \neq \varnothing$ **do**
5 $\quad v \longleftarrow U.\texttt{Dequeue()}$ ▷ BFS. Other traversing techniques are viable
6 $\quad$ **foreach** $t \in T : \mu[v]\,[t\rangle$ **do**
7 $\quad\quad$ Denote $\zeta \in \mathcal{M} : \mu[v]\,[t\rangle\,\zeta$
8 $\quad\quad w \longleftarrow$ **new** Node
9 $\quad\quad$ **if** $\exists u \in V : u \rightsquigarrow_A v$ **and** $\mu[u] < \zeta$ **then** ▷ This step needs elaboration
10 $\quad\quad\quad$ **foreach** $p \in P$ **do**
11 $\quad\quad\quad\quad$ **if** $\mu[u](p) < \zeta(p)$ **then** ▷ Unbounded
12 $\quad\quad\quad\quad\quad \mu[w](p) = \omega$
13 $\quad\quad\quad\quad$ **else**
14 $\quad\quad\quad\quad\quad \mu[w](p) = \zeta(p)$
15 $\quad\quad\quad\quad$ **end**
16 $\quad\quad\quad$ **end**
17 $\quad\quad$ **else**
18 $\quad\quad\quad \mu[w] = \zeta$
19 $\quad\quad$ **end**
20 $\quad\quad$ **if** $\nexists u \in V : \mu[u] = \mu[w]$ **then**
21 $\quad\quad\quad V \longleftarrow V \cup \{w\}$
22 $\quad\quad\quad U.\texttt{Enqueue}(w)$
23 $\quad\quad$ **else**
24 $\quad\quad\quad w \longleftarrow u$
25 $\quad\quad$ **end**
26 $\quad\quad A \longleftarrow A \cup \{v \xrightarrow{t} w\}$
27 $\quad$ **end**
28 **end**

In this algorithm, we specifically do a graph traversal with Breath First Search (BFS) on line 5. However, this is not necessary and any order of traversing the unprocessed set (U) is feasible and should produce a unique coverability graph. Though the graphs identities might vary, they hold certain common properties as follows [2]:

- The algorithm always terminates, and thus the coverability graph is finite.
- Every node in the constructed coverability graph is unique in the sense that no pair of nodes shares the same label/marking, as indicated by the condition on lines 20 and 23.
- All firing sequences of $\mathcal{N}$ are present in CG. For any firing sequence in form of $M_0[t_1\rangle M_1[t_2\rangle ... M_{n-1}[t_n\rangle M_n$, there are arcs $v_0 \xrightarrow{t_1} w_1, v_1 \xrightarrow{t_2} w_2 ... v_{n-1} \xrightarrow{t_n} w_n$ in CG such that:

$$\mu[w_i] = \mu[v_i] \text{ for } i \in \{1, 2, ..., n-1\}$$
$$M_i \leq \mu[v_i] \text{ and } M_n \leq \mu[w_n] \text{ for } i \in \{0, 1, 2, ..., n-1\}$$

The algorithm works by using the monotonicity property. In lines 9-15, the algorithm introduces ω in markings if growth is found. This is because we can repeat the transitions of a monotonic sequence arbitrarily many times to generate as many tokens as possible in some places.

We have delineated the condition check on line 9 as an ambiguous interpretation. In fact, symbolic expressions and conditions might look neater than what they actually represent in practice. In the real algorithm, checking for a qualified path based on the condition on line 9 is complicated since

we are managing a graph, and every of its node might have several parent nodes. As a result, we need to track down all paths that lead to v, which then necessitates the storage of v's parents. This can be done in a recursive manner with a good choice of data structure, hash map for example, and will not be discussed further in the scope of this report for simplicity purposes. More details are provided in the `QGraph` library implementation located inside the "Source Code" folder.

5.5.2 Implementing the first and second goals

Having set up the actual algorithm that generates a coverability graph, we will have to somehow capitalize on it to tackle the first two major goals. Notice that the first goal and **Algorithm 1** both require a graph traversal. We can thus obtain firing information by simply adding some extra lines of code that extract and gather the data to some container during the traversal. Our choice with Qt is to chronically append string-converted markings and firing sequences stored in the graph nodes to an object of a class called `QStringList`, which functions quite similarly to `C++` stringstream with extra flexibility.

The main difficulty at this stage, however, is the second goal. If we run **Algorithm 1** over the graphical Petri net, what we obtain is simply the data of the transition system, which by no means are able to manifest themselves on screen. Therefore, we have to render the data and perform a graph layout algorithm to them. This is excessively tough, considering that graph drawing is a major topic in Graph theory. Nonetheless, there is a clever way to overcome the hardship, given that we only need to render and lay out a graph, not necessarily interacting with that graph. Our solution is to employ an external graphics library to help us lay out the graph, hierarchically if possible, and *Graphviz* just fits our need perfectly.

Graphviz

Graphviz is an open source graph visualization software, also used as a library, specified for DOT scripts having the file name extension "gv". It consists of a set of tools used to render DOT files such `dot`, `neato`, `circo`, etc. One can find out specific functionalities of those tools in the *Graphviz* documentation.

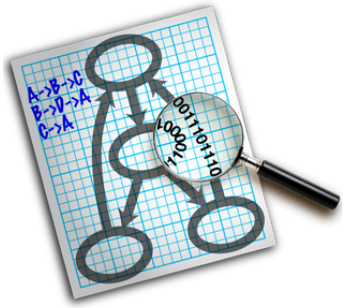

Figure 5.5.1: Graphviz logo

In our case, the `dot` description just rings the loudest bell as a layered-drawing rendering choice for directed graphs. So we decide to settle on the `dot` engine, but a tough task still remains, that is, to generate a DOT file from raw data and render it with *Graphviz* `dot` engine via scripting. The first step is not difficult since we can learn the DOT language and make some functions in the sourcode to translate the data we gathered into a gv file written in DOT. The second step is what bothers us, to integrate the *Graphviz* rendering commands to Qt GUI programs, in particular `C++`. We do not have much time to write a wrapper for *Graphviz* in `C++`, so it is more reasonable to seek for an external library that does the job. And we found the `cgraph` library that supports writing *Graphviz* codes in `C`. Documents of its use and how to link key libraries/plugins from *Graphviz* to Qt can be found at [4] and [3].

The result

Based on the information from the last section, we have successfully implemented 66% of our intention in the software, whose feat users can enjoy via the "Marking" pushbuton on the menu bar, shown in Figure 5.5.2

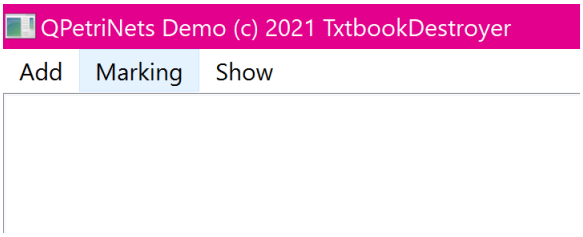

Figure 5.5.2

Let's consider an example Petri net in Figure 5.5.3. It is a structure without loops so we can guarantee its transition system to be bounded, or finite. Therefore, the respective marking data and the coverability graph are finite as well.

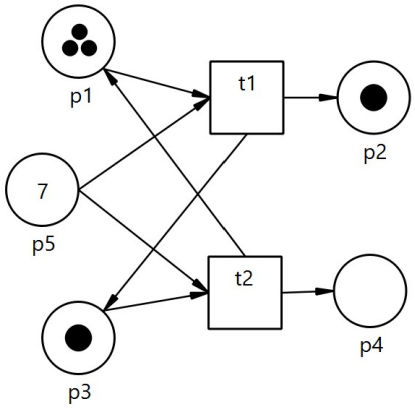

Figure 5.5.3: An example Petri net

By pressing the "Marking" pushbutton, data will be flushed to the screen like Figure 5.5.4.

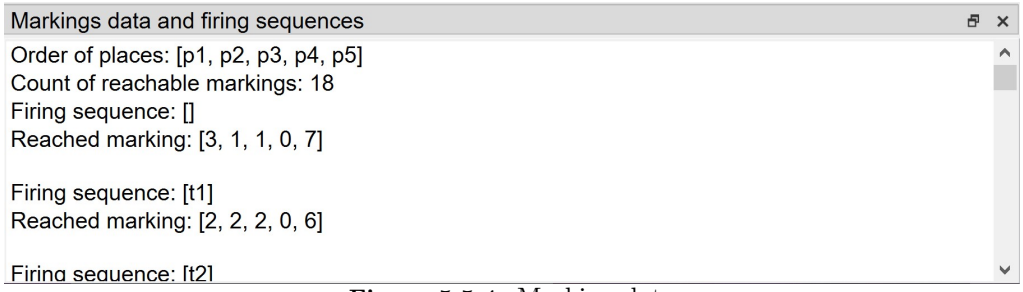

Figure 5.5.4: Marking data

The corresponding coverability graph is given in Figure 5.5.5. In the program, we constrain the maximum of states for the coverability graph to be under 30, for readability purposes. The graph is laid out hierarchically to give a sense of chronological order.

As the coverability set subsumes the reachability set, a bounded Petri net like that in Figure 5.5.3 produces a coverability graph that is also a reachability graph. In our example graph, each layer corresponds to a shrink state by one token of place "p5", making the confirmation for deadlocks fast enough since the net can not fire when "p5" is out of tokens and receives a 0 marking. The final layer shows the deadlocks for the given Petri net.

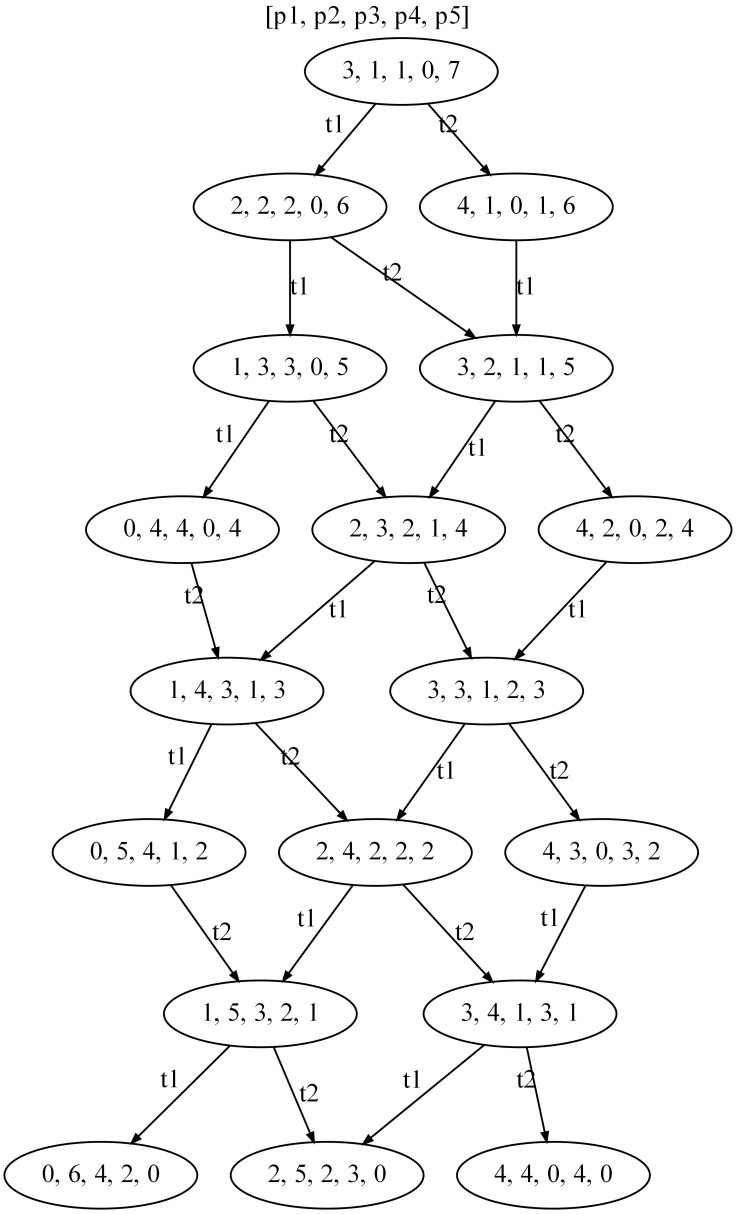

Figure 5.5.5

5.6 Testing

Reviewers can test the program by entering the "Executable" folder and running the `QPetriNets.exe` file. To inspect the source code, however, requires Qt Creator and correct library path configuration in the `.pro` file. We recommend inspecting the files one by one using Visual Studio.

References

[1] Kleijn, H. C. M., and F. M. Spieksma. "Coverability and extended petri nets." (2013).

[2] Richard M. Karp and Raymond E. Miller. Parallel program schemata. Journal of Computer and System Sciences, 3(2):147 - 195, 1969.

[3] Gansner, Emden R. "Using Graphviz as a Library (cgraph version)." Graphviz Library Manual (2014).

[4] North, Stephen C., and Emden R. Gansner. "Cgraph Tutorial." (2014).

[5] Van Der Aalst, Wil. "Process mining." Communications of the ACM 55.8 (2012): 76-83.

[6] Reisig, Wolfgang. Understanding petri nets: modeling techniques, analysis methods, case studies. Heidelberg: Springer, 2013.

[7] Van der Aalst, Wil, and Christian Stahl. Modeling business processes: a petri net-oriented approach. MIT press, 2011.