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

MATHEMATICAL MODELING

ASSIGNMENT - GROUP NAME: 404 NOT FOUND

PROCESS MINING USING PETRI NET

Under the guidance of:	Prof. NA Khuong
Accomplished by:	Nguyen Nho Gia Phuc - 2052214
	Nguyen Van Quoc Chuong - 1950004
	Nguyen Thanh Hieu - 1852369
	Nguyen Huy Hoang - 1852382
	Tran Pham Minh Hung - 2053067

HO CHI MINH CITY, 11/2021

Contents

1 Process mining

During the last fifty years, the humanity has come a long way in the field of technology, we have achieved many incredible advances especially with those regarding to computers. For example, the number of transistors on integrated circuits has been doubling every two years, albeit at a slower pace as predicted by Moore's law. Disk capacity as well as performance of computers per unit cost, etc. have been growing at a similar pace. Besides these magnificent advances, people and organizations depend more and more on computer devices and information sources on the internet. These reasons lead to what is referred as the *data explosion*.

However, most of the raw data is stored in the digital universe is unstructured and organizations have problems dealing with such large quantities of data. For such reasons, the information systems have been improve significantly. Many technologies such as RFID (Radio frequency Identification), GPS (Global Positioning System), and sensor networks has stimulated a further alignment of the digital universe and the physical universe. Many region-related services of many corporations like “App Store” of Apple or “Steam” of Valve have intertwined the digital universe and the physical universe by combining location-awareness with continuous internet connection.

The growth of a digital universe that is well-aligned with the processes in organizations make it possible to record and analyze events. The challenge now is to exploit event data in a meaningful way, for example to provide insights, identify bottlenecks,etc. And so that is what process mining is all about!

1.1 Definition

Process mining, which can be considered an expansion of work-flow mining, is an emerging discipline that focus on the analysis and improvements of process since the 90s using events data. It builds on process model-driven approach and data mining, therefore heavily related to these two disciplines: data science and process science.

1. **Data science**

   It is a ”concept to unify statistics, data analysis, informatics, and their related methods” in order to ”understand and analyze actual phenomena” with data. Data science is the core of many partially overlapping disciplines:

   - Algorithms, business models, data mining
   - Data transformation, storage and retrieval of information

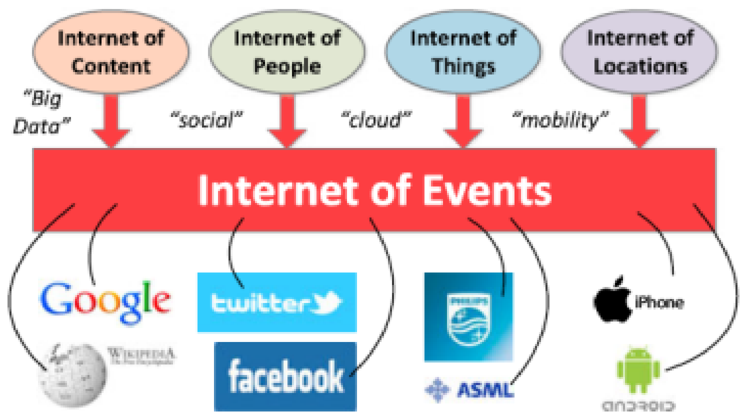
- (Computing) infrastructures: databases, distributed systems
- Privacy- security law, predictive analytics (learning, predictions)
- Statistics (modeling+ inference), can be viewed as the origin of data science (DS)...

Data science and its key component, data mining, however are data-centric, not process-centric. Data mining, Statistics, Machine learning and the likes technically do not consider end-to-end process models. Data science usually make use of classical sample data, I.e. numerical-digital observations.

2. **Process science**

A broad term aims to combine knowledge from information technology and knowledge from management sciences to run and improve operational processes. Research objective is changed [from sample numerical/digital data to event data], so the coupling methods are changed, upgraded to a more sophisticated scale, and as a result, applications are broader.

In process science the concept of **EVENT** is used, popularly called the **Internet of Events - IoE**, is a brand new structural extension of classical sample data. In general, it contains 4 dimensions, possibly overlapping, as follows.



Event data nowadays are generated from a variety of sources connected to the Internet [courtesy Wil van der Aalst (2016)]

Figure 1

Content, i.e., all information created by humans to increase knowledge on particular subjects. The IoC includes traditional web pages, articles, encyclopedia Wikipedia, YouTube, e-books, newsfeeds...

People,i.e., all data related to personal, human-being subject with their social interaction (e-mail, Facebook, Twitter, forums, LinkedIn, etc.

The Internet of Things (IoT),i.e., all physical objects connected to the network, or broadly, relevant hardware which allow an event or sub-process happens.

Locations refers to all data that have a geographical or geo-spatial dimension.

Process mining, with the key study objectives of event and event data, like other BPM (Business Process Management) approaches in process science field, is process-centric. However, unlike most BPM approaches, it is driven by factual event data rather than hand-made models. Hence, process mining can be seen as a bridge between the process science and data science.

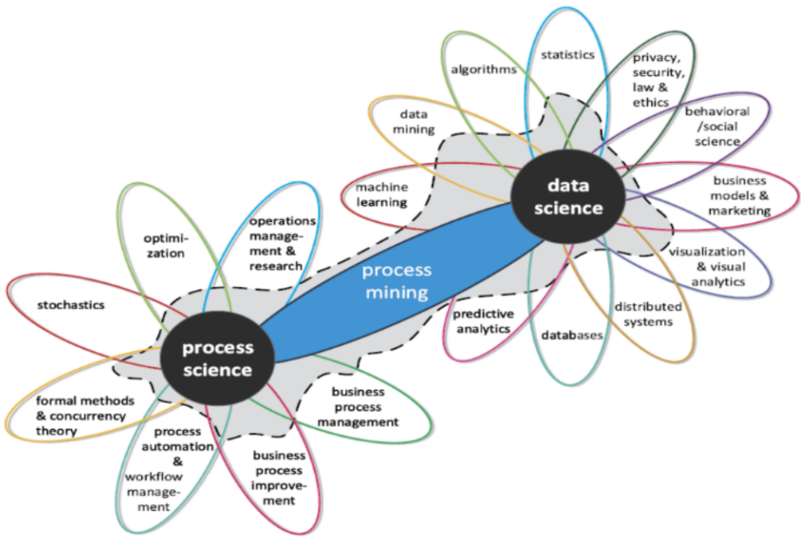

algorithms
statistics
privacy, security, law & ethics
data mining
behavioral /social science
machine learning
data science
business models & marketing
optimi-zation
operations manage-ment & research
process mining
visualization & visual analytics
stochastics
predictive analytics
distributed systems
databases
process science
formal methods & concurrency theory
business process management
process automation & workflow manage-ment
business process improve-ment

Figure 2: The link between Process Science and Data Science.

1.2 Reason behind the combination of Data Science and Process Science

Structurally, Process Mining is viewed as the core of a tree, with the branches:

- Business process management and improvement (BPM & I), business intelligence (BI), as Petri nets, Lean Six Sigma, TQM

- Formal methods and concurrency theory, machine learning
- Optimization, operations (management and research),
- Process automation and work-flow management,
- Stochastic process and analysis, as Markov models,
- Statistical inference and optimization, time series model and analysis, and so on.

Exploiting data means revisited as well as aiming to answer the following four questions:

1. Reporting: What happened?
2. Diagnosis: Why did it happen?
3. Prediction: What will happen?
4. Recommendation: What is the best that can happen?

In order to fully answer such questions, raw data and computing power is just simply not enough. Probability/statistics/stochastic, inference, and causality analysis, and visualization are essential. It requires expertise in data and process science to efficiently decode diverse complex datasets from biology, chemistry, computing, environment,... And thus, process mining emerges.

Although it only recently appeared as a sub-discipline of both data science and process science, but the corresponding techniques can be applied to any type of operational processes (organizations/systems), as:

- Analyzing treatment processes in hospitals,
- Improving customer service processes in a multinational corporation,
- Understanding the behavior of customers choosing an insurance firm,
- Improving the efficacy of a new vaccine to cope with fatal pandemics, knowing lab's experimental designs, better modeling data obtained from labs/field trips/factories, in order to perform convincing statistical analyses, and finally to make meaningful guidelines or (nearly) optimal decisions].

Conclusion: All the related workers must understand things at process level (at least at dimension of EVENT DATA) for it operate properly.

1.3 Process Mining - A brief development history and Top applications

Some notably mark:

- 2015 - Compliance monitoring in business processes: Functionalities, application, and tool-support
- 2016 - The State of the Art of Business Process Management Research: related to research methods, quality discussion, maturity, citations index, and progress in the business process management
- 2016 - Process mining in healthcare, Biomedical Informatics.
- 2018 - A systematic mapping study of process mining: maps the relationship between data mining tasks in the process mining context...

Top seven areas with $>$ 80 % publications related to PM applications:
α **Healthcare:** covering clinical path, patient treatment, or the primary processes of a hospital
β **ICT:** related to software development, IT operation services
γ **Manufacturing:** in industrial activities, realized by a factory that usually receives material and delivers partial or finished products
δ **Education:** e-learning, scientific applications, and centers with innovation process management
ϵ **Finance and** η **Logistics:** process management for profit maximization
λ **Robotics / Smart:** applications using advanced technologies related to smart buildings, industry 4.0...

1.4 Some Process Mining key research topics

1. Process Discovery
2. Process Conformance
3. Process Enhancement...

From the Quality Engineering view, including process control and improvement, process mining does not replace the traditional process improvements methods, such as Business Process Improvement (BPI), Continuous Process Improvement (CPI), Corporate Performance Management (CPM), Total Quality Management (TQM), Six Sigma, and others. However, process miners are able (a) to check compliance, diagnose deviations, point out bottlenecks, and then (b) to perform, integrate, accelerate process improvements, as well as recommend actions and, last but not least redesign systems.

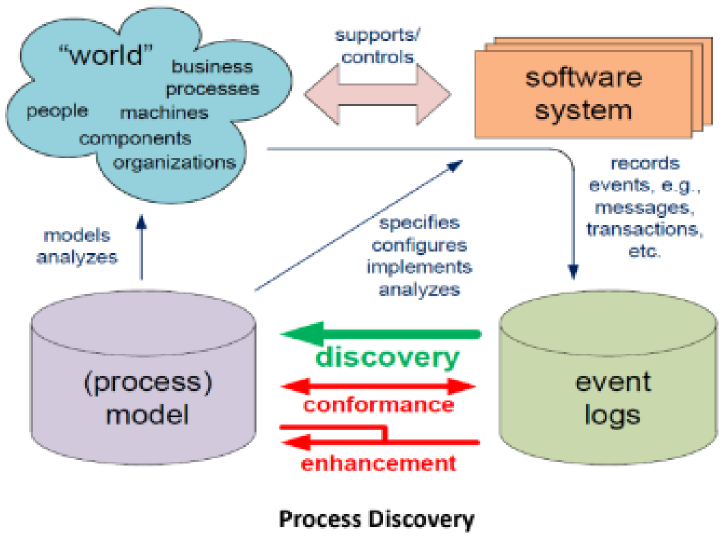


Figure 3: Process Discovery Bridge.

2 Petri net

This section aims to formally define a Petri net, study about its behavior, and extend its base knowledge to more advanced networks.

2.1 Multiset

Multiset is a core concept used in Petri nets. In regular algebra, a set contains distinguished elements (the order does not matter) and a repetition of an individual element does nothing more than confirming the existence of it. A multiset, though, takes into account that repetition.

For example: A lady opens up her purse and finds some lipsticks, some bottles of sunscreen, and a pepper spray.

A set describing her items would be $\{lipstick, sunscreen, spray\}$. A possible multiset of such items could be $[3.lipstick, 2.sunscreen, spray]$, implying that she has 3 lipsticks, 2 bottles of sunscreen, and a pepper spray. Notice the dot(.) used in the multiset: it denotes how many times its suffixed item is present in the set. This is a very compact and intuitive notation and saves time to write explicitly, say, "lipsticks" 3 times. What is more, the brackets for multisets are square brackets rather than the regular set brackets.

In a Petri net, a multiset denotes a marking of it. It tells us how many tokens are present, and the individual token count in every place. If a Petri net's place is not seen in a particular marking, then at that stage no token is in it. A more in-depth look of markings and multisets will be discussed later in this report.

2.2 Transition systems

In order to boost productivity and efficiency of a certain process, we aim to model it graphically and mathematically. A compact yet easy-to-understand model helps businesses and users make the most out of a certain process.

For example, you are asked to do the laundry. When we think of this, there is a set of activities needed to be done in a certain order, say collecting clothes, putting them in the washing machine, tumble drying, and air drying. Whenever your mother checks up on you, you should be in one of the four mentioned states. All activities can be combined into a single set, called the *state space*. A system can be in several states. Next up, we care about changing from a state to another, and aim to generalize it.

Such change is called a **transition**. Take two arbitrary states in the state space. If a state can change to the other at once then we can say that there is a transition between them. Like in the example of the previous paragraph, a transition is present when we are done putting clothes out from the washing machine and putting them back in the tumble dryer. We do not care about the time, or the activities that happen in-between those states, but rather just acknowledge the change. Mathematically, we can generalize a transition as an ordered pair (a, b), with a and b states in the state space $S(a, b \in S)$. The ordered bit tells us which state turns to which.

A transition relation contains all *possible* transitions of a system. This should not be mistaken for all the ordered pairs within the state space S, because some transitions are not possible. In our example, we cannot have a transition between the tumble dryer and the act of collecting clothes. Let the Cartesian product $S \times S$ denote all the ordered pair of states, so if we use TR as the identifier for transition relation, TR is the subset of the mentioned Cartesian product: $TR \subseteq S \times S$.

Finally, we can find the initial state of the transition system and denote it with S_0. This is where the system starts operating. To sum up, we have a definition of transition system. A transition system is a triple (S, TR, S_0), where S is the finite state space, $TR \subseteq S \times S$ is the transition relation, and $S_0 \in S$ denotes the initial state. This notion bears some similarities to the definition of a finite automaton, which have been studied in the former stages of this semester.

2.3 Definition of a Petri net

The transition system can model processes, and for good reasons: it is easy to understand, has a mathematical foundation, and can be graphically represented. However, as we define the state space S to be finite, when it becomes too large of a number, the transition relation TR consequently increases. A big transition system can be hard to grasp, especially when it comes to parallelism (concurrency). A Petri net shows up as a delicate solution to such ”state explosion” of a simple transition system.

A Petri net inherits the essence of a transition system. It can be mathematically defined, can be expressed graphically, and is easy to understand for non-experts. It does not explicitly tell all the possible state (not showing all the elements of the state space) but rather generalizes operations so succinctly that even infinite states can be reduced to just 2 Petri net components.

Rather than using a state space and a transition relation, Petri net uses concepts called **place**, **transition**, **arcs**, **flow relation**, and a special node called **token**.

In a Petri net

- Places (graphically rendered as circles or ellipses) can show states by storing tokens. It models passive components and are in discrete states.
- Transitions (graphically rendered as a rectangle, square, or black rectangle bars) can trigger the token flow of the Petri net. It models active components.
- Tokens (graphically rendered as black dots) are the representation of real world items, such as people, cargo containers, or simply the state of something. A token moves from one place to another through a transition, therefore is not contained inside a transition, but is acted upon by one.
- An arc is an arrow connecting a place and a transition or vice versa.
- Flow relation describes the connection between places and transitions, pretty much like the transition relation described in the part of transition systems.

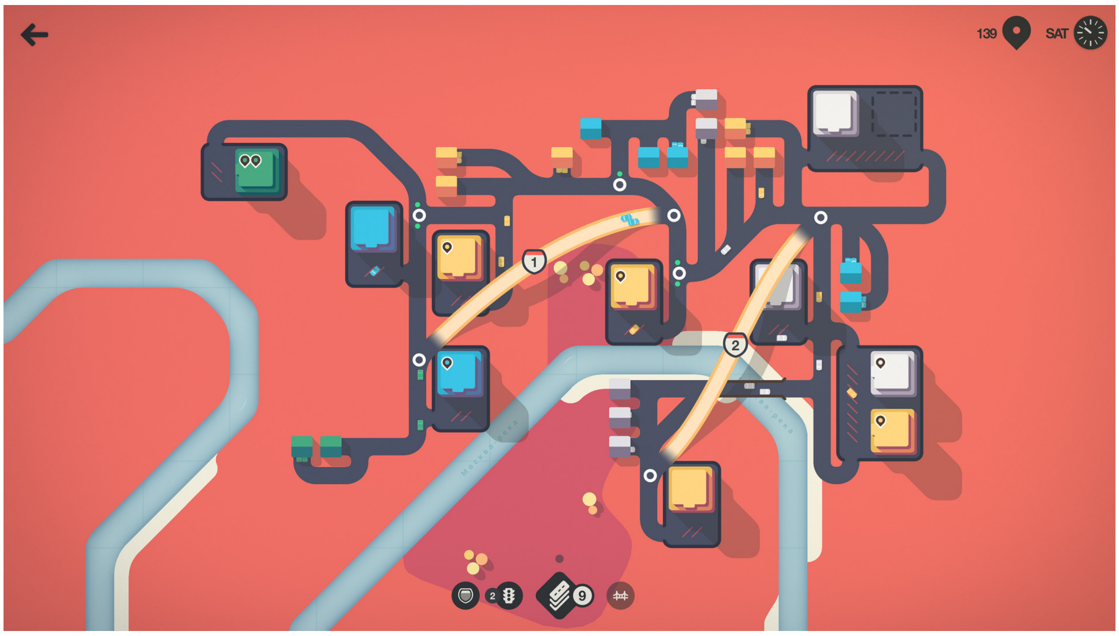


Figure 4: The game Mini Motorways can help us somewhat familiarize with places, transitions, and tokens running around (image in courtesy of Steam).

As we mentioned about the mathematical foundation of Petri nets, let us define it in mathematical terms.

A Petri net is a triple (P, T, F), where

- P is a finite set of places.
- T is a finite set of transitions.
- $F = (P \times T) \cup (T \times P)$ is a flow relation.

P contains all the places of the net. Such place are **labeled** according to their functionality. Because places store tokens and are passive components, we often use nouns or adjectives to label them.

T contains all the transitions of the net. They are also labeled, but since they are active components which acts upon the tokens inside interconnecting places, we can use verbs to label them. Such clear labeling helps when it comes to introducing a Petri net to non-experts as well as when it comes to reviewing a net's functionality.

The arcs, as mentioned, connect places to transitions and transitions to places. Therefore, there are two kinds of arcs. An arc can be represented with an ordered pair (x, y) having binary relation, and with two kinds of arcs, we have two binary relations. Take $R_1 \subseteq (T \times P)$ the relation of arcs connecting transitions to places, and $R_2 \subseteq (P \times T)$ the relation of arcs connecting places to transitions. The flow relation is therefore the union of R_1 and R_2.

We go further by introducing **input places**, **output places**, **presets**, and **postsets**. The input places of a transition $t \in T$ are all places $p \in P$ where the ordered pair (p, t) is an element of the flow relation F. Informally, we say from these input places, we can trigger a certain transition to do some tasks. Mathematically, we define the set $\cdot t = \{p|(p, t) \in F\}$ the preset of the transition t. The same goes for output places and postset. The postset $t\cdot = \{p|(t, p) \in F\}$ defines all output places of a transition t. If we think of a Petri net as a bipartite graph, the number of elements of a transition's preset and postset are the in-degree and out-degree of it, respectively.

Now, let us have an example to have a recap on all the Petri net concepts introduced so far.

Counter-Strike: Global Offensive (CS:GO) is a very popular tactical first-person shooter game developed by Valve. Some game situations can be described with a Petri net.

There are often two modes in CS:GO, bomb planting and hostage rescue, but the former is greatly more favored by players. There are 2 bombsites (A and B) and the Terrorists would plant the bomb in either site while the Counter-Terrorists would defend the sites to stop it. Let us examine a strategy called "faking" in which a player from team Terrorist tries to make the Counter-Terrorists think his team is going to attack a site, but it is actually another way around. Figure 5 showcases the net.

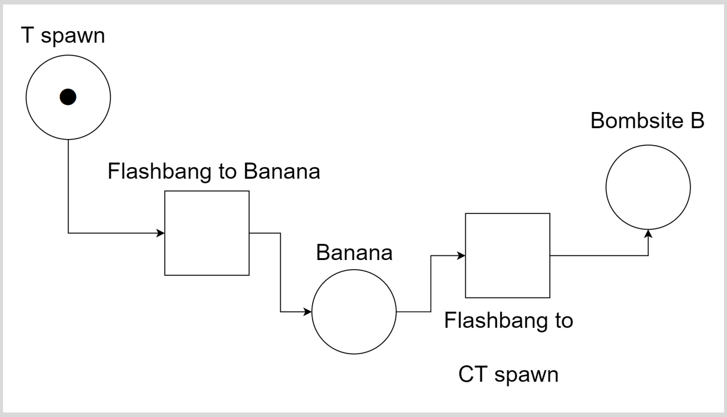


Figure 5: A Petri net showing B faking tactic on Inferno - CS:GO.

We labeled the net with legends according to the actual map. When faking B site, a terrorist moves from T spawn, banana, and then bombsite B. The set P, therefore, is {T_spawn, Banana, Bombsite_B}. While doing it, he has to throw flash bangs twice to Banana and CT Spawn. This gives us the set T, which is {Flashbang to Banana, Flashbang to CT spawn}. From T spawn, he can only go to Banana and not directly to Bombsite B. The flow relation as stated, contains ordered pair with binary relation. We can name some elements of this set like (T spawn, Flashbang to Banana).

We move on to study transition $FlashbangtoBanana$. Its input place is T_spawn, and its output place is Banana. Therefore, its preset has one element and so does its postset. Finally, the terrorist is still in T spawn, so a token at this place denotes his presence. Later on in the CS:GO round, through some transition firings, he will end up in Bombsite B (given he was not killed in action). Every state he is in will be shown through a **marking**, which is introduced in the next subsection.

2.4 Petri net behavior

Now that we have defined a Petri net as a triple (P, T, F) and studied about markings, tokens, arcs, and so on, we aim to understand the behavior of such net through the terms marking, enabled, firing, and deadlock.

First, a marking is a distribution of tokens across a Petri net. A multiset is used to describe a marking, telling us what which place how many tokens are present. This is the key to Petri nets being compact and not end up "exploding" like transition systems, because instead of listing all states, we only need to use fewer places, transitions, and let markings do their work.

A place in a Petri net can store tokens, a transition changes the marking by firing, and to fire, it first has to be enabled. If no transition is enabled, the Petri net is deadlocked. This further implies transitions being active and places being passive.

A transition is enabled when all input places of it each contains a token. An enabled transition can fire, consuming a token from every input place and producing a token at every output place.

A net deadlocks when no transition is enabled, meaning no more firing will ever occur. The marking at the deadlocked situation is called the terminal marking of the net.

2.5 From Petri net to transition system

Transition systems are amongst the most elementary systems of process modeling. We can represent a Petri net $N = (P, T, F, M_0)$ as a transition system (S, TR, S_0). Let us have a thorough look at individual components of such transition system.

S is a state space covering every possible token distribution in a Petri net's places. As a result, every possible marking M of the net will be present in S, or in other words, S is comprised of all functions that maps P to the natural number set $(P \longrightarrow \mathbb{N})$.

For the transition relation set TR, let us examine two random markings in the state space S. If a marking, say M, yields another marking M' through the firing of a transition $t \in T$, then transition (M, M') is an element of TR, and this element is unique. This means that in all other cases, no such transition exists. The definition of TR can be expressed as the set of all pairs $(M, M') \in S \times S$ such that there is a transition $t \in T$ enabled at marking M, and the firing of t results in M changing to M'.

Finally, S_0 can be determined quite easily from the initial marking M_0 in the Petri net.

2.6 Colored Petri net

Simple Petri nets are often referred to as Places-Transitions net due to their simplistic nature. A token is rendered as a black dot, and while it can move across the net with a series of transition firings, we do not know which token represent which data. The traditional Petri net can model simple business processes, but it encounters some problems.

First, it is the **Large network size**. Take a production line for example. There are two key products made here, and we can express their manufacturing processes with Petri nets. The problem arises when the products have many components, and at the same time they **share** some of the components and manufacturing steps. Because we have no distinction of places or transitions in the traditional net, the size of the nets keep on increasing. Imagine just how tiring and difficult it would be to read such a big net, say, for manufacturing two different models of Lamborghini cars. For every additional product, we need to duplicate the same net and modify just a little here and there. This is caused by the fact that tokens are not distinguishable. A token of this type may end up in another place of different type but we do not know. Therefore, to distinguish tokens we must put them in separate places.

Second, traditional nets lack **expressive power**. In such nets, a transition can fire as soon as it is enabled without further conditions. This is not ideal for a business process model. For example, in a car's production line, the transition from chassis to painting is enabled as soon as the first chassis arrives from the factory. It is inefficient, because it would be more favorable if multiple chassis are painted at once. A threshold is needed, like a guard, to make sure enough chassis are ready to get painted. A traditional Petri net does not have the capability to do this, and it comes back to the indistinguishable tokens again.

The two mentioned problems can be solved with colored Petri nets. One more extension of Petri nets is time, but since we are not going to concern this matter, we can simply understand that transition firings in a Petri net is *timeless*.

3 Practical assignment: Consulting medical specialists

It is high time we put what we know about Petri net into a practical problem. From the problem statement in the description file, let us have a short highlight of important information within the said statement.

- Scenario: in a city H, patients would consult specialists in an outpatient clinic of a hospital, and the whole process would be modeled using Petri nets.
- There are three possible states for a specialist: (1) *free* (the specialist is free and is waiting for the next patient), (2) *busy* (the specialist is treating a patient), and (3) *docu* (the specialist is documenting the patient's result).
- There are also three possible states for a patient: (1) *wait* (the patient is waiting to be treated), (2) *inside* (the patient is being treated by the specialist), and (3) *done* (the patient is done with the treatment).
- The patient goes through these states only once, while the specialist does that in an iterative fashion.
- There are three key events for this scenario: (1) event *start* when the specialist starts treating a patient, (2) event *change* when the specialist is done with the treatment and starts documenting, and (3) event *end* when the specialist finishes documenting.

From the given data and assumption, we can have some basic acknowledgements about the scenario [1]. First, a specialist or a patient can be described as **tokens** in the Petri nets. Second, we can take into account the states of specialists and patients and make them **places** in the Petri nets. Finally, an event is represented as a **transition**.

As the specialist goes through the states in a iterative fashion, while a patients does it once, it explains Figure 5.20 in the description file. In real life, this is also true because of the fact that

[1]Petri_NETWORKS Version 0.2 - page 25

a specialist (doctor) in a hospital has to work with several patients while a patient only needs to have a single treatment at a time. Figure 6 is a copy of the said Figure 5.20 in the description.

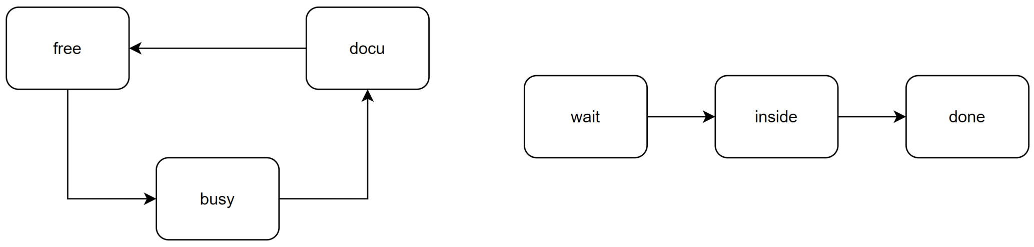


Figure 6: The transition systems of a specialist and a patient (Re-drawn for better clarity).

3.1 Petri net of the specialist

Given the Petri net N_S modeling the state of the specialist, as in Figure 7. In the displayed marking, the specialist is in state *free*.

(a) Write down states and transitions of the Petri net N_S.

(b) Could you represent it as a transition system assuming that

(i) Each place **cannot** contain more than one token in any marking; and

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

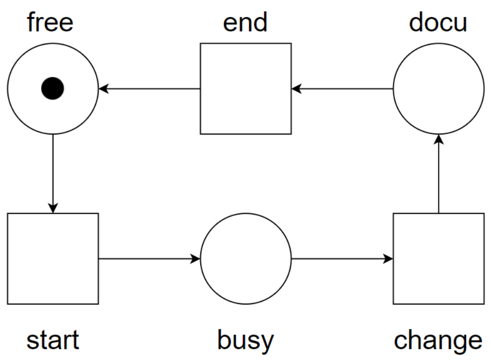


Figure 7: The Petri net of the specialist's states.

Solution:

(a) In the given Petri net N_S, there are 3 states (places) and 3 transitions.

- 3 states: *free*, *busy*, *docu*.
- 3 transitions: *start*, *change*, *end*.

(b) Before analyzing the net N_S as a transition system of the two requirements, let us define some properties of the transition system (S, TR, S_0). S is a finite state space that contains all the markings of N_S. TR is the transition relation set which contains every possible transition from S should it occur. Finally, S_0 is the initial state of the transition system, and is identical to the initial marking M_0 of such Petri net. We can describe a marking of the net as a triple (x, y, z):

- x specifying the number of tokens in place *free*.
- y specifying the number of tokens in place *busy*.
- z specifying the number of tokens in place *docu*.

(i) **Each place cannot contain more than one token in any marking**. This restricts the tokens in a place to at most one. The state space S in this case can be formalized as $S = \{(x, y, z) | x, y, z \in [0, 1]\}$. For TR, first we need to work out for individual firing of the three transitions, which marking would yield which. Respectively for three transitions *start*, *change*, and *end*, we get three sets:

$$\begin{cases} TR_1 = \{((1,0,0),(0,1,0)),((1,0,1),(0,1,1))\} \\ TR_2 = \{((0,1,0),(0,0,1)),((1,1,0),(1,0,1))\} \\ TR_3 = \{((0,0,1),(1,0,0)),((0,1,1),(1,1,0))\} \end{cases}$$

TR, therefore, should be the union of the three sets above, which is $TR = TR_1 \cup TR_2 \cup TR_3$. Corresponding to the Petri net N_S, S_0 is $(1, 0, 0)$. This case should not be mistaken for ***no more than one token in the whole net***, which is why there is a state $(1, 1, 0)$ or even $(1, 1, 1)$ in the set S.

(ii) **Each place may contain any natural number of tokens in any marking**. This case is the expansion of the previous item. Rather than restricting the number of tokens to be no more than 1, in this case, we can have infinitely more states as well as transition relations. The formalized state space S is the same with some tweaking: instead of being bounded between 0 and 1, now, x, y, z are elements of the natural number set, which yields $S = \{(x,y,z)|x,y,z \in \mathbb{N}\}$. For the transition relation TR, we can also generalize the case mentioned in the previous item. There is an observable rule. For example, take TR_1 in item (i) (corresponds to transition *start*), we can see that whenever this transition fires, it consumes a token in place *wait* and produces a token in place *inside*, and with our defined triple (x,y,z), x decreases from the former marking to the latter marking of a transition relation and vice versa for y, while z indeed does not change. This defines TR'_1 as follows: $TR'_1 = \{((x+1,y,z),(x,y+1,z))|(x,y,z) \in \mathbb{N}\}$. Similarly, we can deduce generalized form of other transitions into TR'_2 and TR'_3 as $\{((x,y+1,z),(x,y,z+1))|(x,y,z) \in \mathbb{N}\}$ and $\{((x,y,z+1),(x+1,y,z))|(x,y,z) \in \mathbb{N}\}$, respectively. We take the new transition relation set TR to be union of the three previous sets, just like in item (i). The set that remains unchanged is $S_0 = (1,0,0)$ since the initial marking is unique and indeed unchanged.

3.2 Petri net of the patient

Define N_{Pa} as the Petri net modeling the state of patients. By the similar ideas (to the specialist's net),

(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) The net N_{Pa} models the state of the patient. Therefore, in general, a token inside N_{Pa} represents a patient. Specifically, in state *inside* of this Petri net, a token indicates the following ideas:

- A patient has done waiting in the state *wait* and is being treated by the specialist in state *inside*.
- This also means that a firing occurred: a token in state *wait* is consumed and a token is produced in state *inside*.

(b) Because the patient goes through the states only once, we can think of N_{Pa} as a straight process without any looping like N_S (due to the fact that a specialist works iteratively).

Using the same idea as N_S, though, we know N_{Pa} has 3 places: *wait*, *inside*, and *done* and 2 transitions: *start* and *change* rather than 3 compared to N_S. This is because the documenting of specialists requires no presence of a patience then. Finally, along with the requirement "assuming that there are five patients in state *wait*, no patient in state *inside*, and one patient is in state *done*", we obtain $P = \{wait, inside, done\}$, $T = \{start, change\}$, $F = \{(wait, start), (start, inside), (inside, change), (change, done)\}$, and $M_0 = [5.wait, done]$.

Fig. 8 illustrates N_{Pa}.

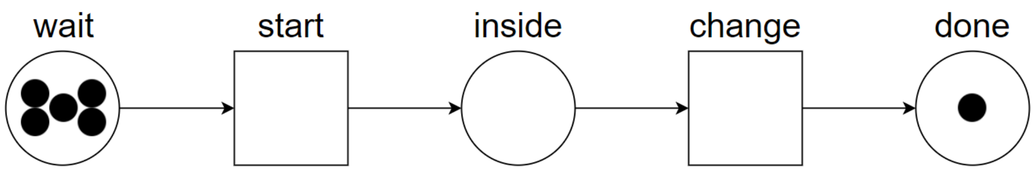


Figure 8: Petri net N_{Pa} with the described initial marking.

3.3 Superimposed Petri net

Determine the superimposed (merged) Petri net model $N = N_S \bigoplus N_{Pa}$

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:

The two nets N_S and N_{Pa} share the same two transitions *start* and *change*, while transition *end* is exclusive for the specialist's net. We can design N such that it shows the dynamic of the whole outpatient clinic's operation but still preserve the constituent nets.

The superimposed (merged) Petri net is determined by: $N = N_S \bigoplus N_{Pa} = (P_S \cup P_{Pa}, T, F_S \cup F_{Pa}, M_0)$. As stated, the two component nets share two transitions, but in each net they have different presets. By the superimposing operation, we keep both presets and gradually form a single set T. Finally, when explicitly written, the superimposed net's components are: $P =$

$\{wait, inside, done, free, busy, docu\}$, $T = \{start, change, end\}$, $F = \{(wait, start), (start, inside),$ $(inside, change), (change, done), (free, start), (start, busy), (busy, change), (change, docu),$ $(docu, end), (end, free)\}$, and $M_0 = [4.wait, 1.free, 1.done]$. The initial marking corresponds to the requirement "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*". Figure 9 showcases the net N.

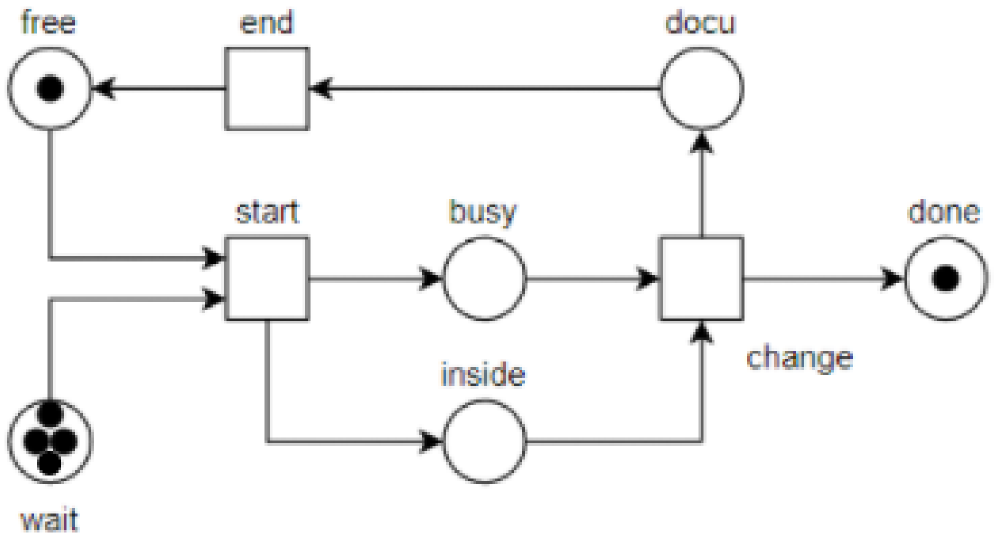


Figure 9: Petri net N

We can see that this net indeed preserves the operations of its two component nets. The patient still travels through the places in one visit, and the specialist still works in an iterative way. The treatment starts **only** when the patient and the specialist is ready to do that (transition *start* is enabled when there are tokens in both places *free* and *wait*).

3.4 Superimposed Petri net (cont.)

Consider an initial marking $M_0 = [3.wait, done, free]$ in the grand net $N = N_S \bigoplus N_{Pa}$.

Which markings are reachable from M_0 by firing one transition once? Why?

Solution:

At M_0 of the merged Petri net, only transition *start* is enabled. Its preset has two places *wait* and *free*, and with the given initial marking, *wait* has 3 tokens and *free* has 1 token. So by firing one transition once, the only marking reachable from M_0 is $M = [2.wait, busy, inside, done]$. As only *start* fires, a token from *free* and a token from *wait* is consumed.

In the resulting marking, only *change* is an enabled transition. The whole process is illustrated in Figure 10.

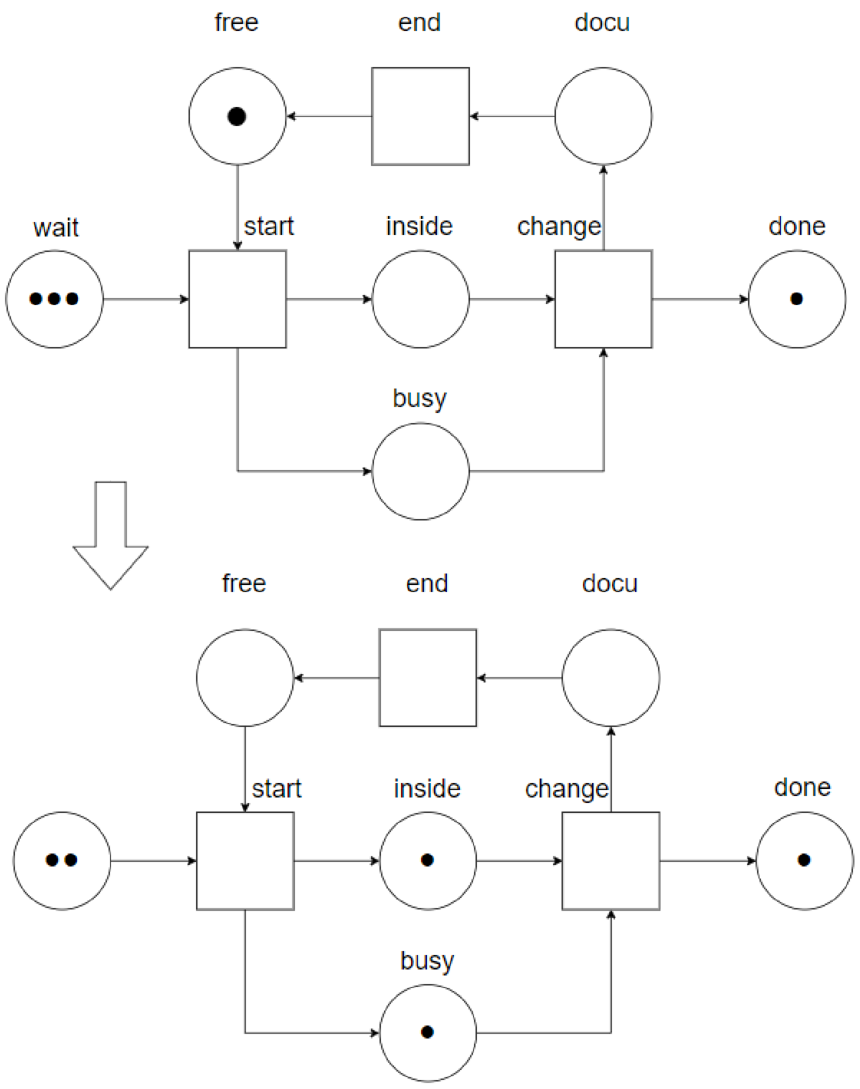


Figure 10: Firing one transition once.

3.5 Superimposed Petri net (cont.)

Consider the net N.

Is it deadlock free? Explain properly.

Solution:

A marked Petri net is deadlock free "if at every reachable marking at least one transition is enabled". We can check if N is deadlock free by verifying the given condition. Otherwise, if we can find a terminal marking where the net does not have any enabled transition, then the Petri net is not deadlock free.

The preset of the transition *start* has two elements: place *wait* and place *free*. A crucial observation here is that the number of tokens in place *wait* cannot increase, but instead only decreases after every firing of *start*. Therefore, if *wait* is depleted then surely the transition *start* would not be enabled again, resulting in a deadlock. But what about the other place in its preset, *free*? Take the previous exercise for example, after *start* fires for the first time, a token is produced in places *busy* and *inside*, thus enabling *change*, and when *change* fires, a token is produced in place *docu*. Eventually, *free* will contain a token again.

If we start from the initial marking when we merge the nets N_S and N_{Pa}, i. e. $M_0 = [5.wait, 1.free, 1.done]$, then after a series of transition firing, we would reach the terminal marking $M = [1.free, 6.done]$. This implies that N is **not** deadlock free. Figure 11 illustrates this marking.

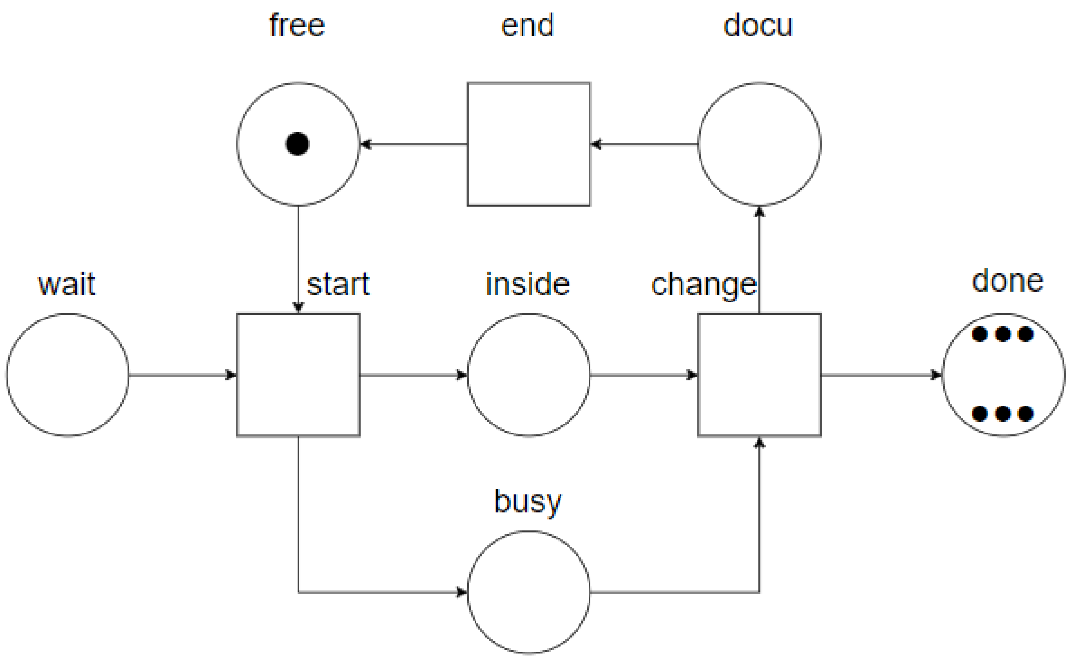


Figure 11: Terminal marking of the merged Petri net N.

In real life, a similar case would be when the outpatient clinic is not treating anyone, all the treating and documenting is completed, and there is no more patient waiting for their turn.

3.6 A similar net

Propose a similar Petri net

With two specialists already for treating patients, with explicitly explained construction.

Solution:
The originally proposed Petri net has a problem when it comes to determining token type. We can tell how a patient or a specialist moves through the net, but when there are plenty of them, we do not know explicitly **where** they end up. Some patients could end up in the place *busy* rather than *inside*, or a specialist may be sitting in place *inside* instead. Therefore, we can introduce a new place *treatment_room* to contain both specialists and patients, ditch the two old places, and extend the superimposed net to a **Colored Petri net**.

We have already built a merged Petri net with both the specialist and the patients before in exercises 3 and 4. However, to specifically explain how the medical examination and treatment with 2 specialists, a simple net like this is not enough. Therefore, we have constructed a colored Petri net CPN$_C$ from the original merged Petri net to represent it with some basic concepts only .

We keep some of the places and the transitions from the merged Petri net. However, to show how this health care model works more clearly, we have make some changes with the places, the transitions as well as the arcs.

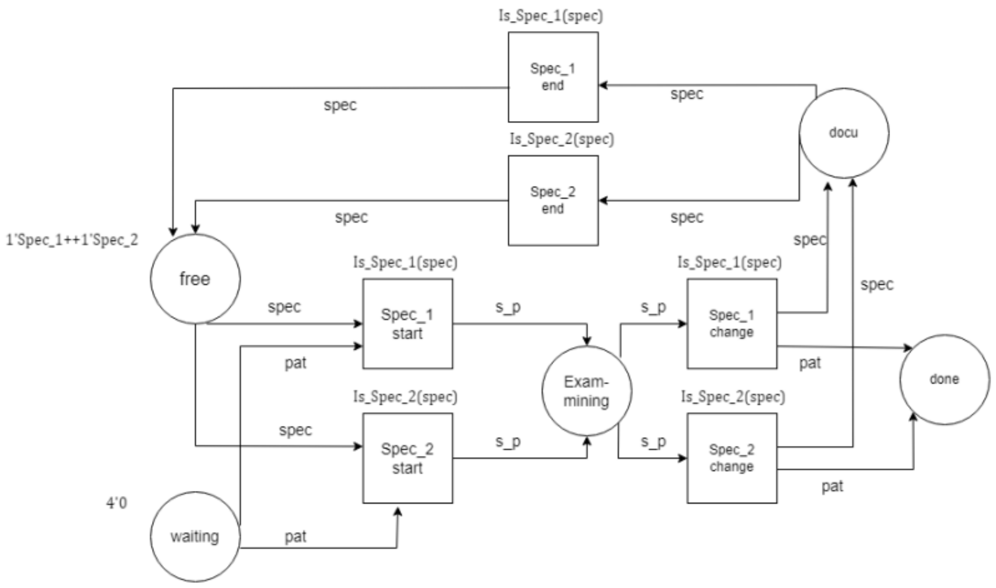


Figure 12: New Petri net N.

Some places like **free, wait, done, docu** are kept. The two places busy and inside is merged and create a new place called examining. The main reason we use the colored Petri net is to represent the two specialists and also the patients distinguishingly. To do that, we add a type for each place that decide what type of tokens it can store. The places like free and docu only for the specialists, wait and done only for the patients and examining can contain both of them.

The types of patients and specialists are being declared as:

- *Color_set* Specialist: union of *Spec_1* and *Spec_2*: Place with this types can contains both specialist 1 and specialist 2.

- *Color_set* Patients = integer: the patient type is a integer represent the number of patient.

With the declaration of the types, the initail markings are also changed. The initial state(

place) **free**, which type is specialist, contains 1 specialist $Spec\_1$ and 1 specialist $Spec\_2$. The initial state(place) **wait**, which type is patiens now contain a number of patients waiting, for our example figure is 4 patients. Since each evaluation of the arc expression must yield a multi-set over the colour set that is attached to the corresponding place, we must assign them with one of the types that match the type of place the tokens are firing to. Therefore, we declared them as variables with types; spec with type specialists, pat with type patient and s_p for both specialists an patients.

We have also doubling some places base on the idea of healthcare facilities in daily life. Mostly the hospitals or clinics we visit every day contain many faculties. Every faculties have their own spaces and rooms but usually, many specialists is in one same big room with their own tables. In our previous Petri net, we can see the set of transitions {**start**,**change**, **end**} as a single table for a single specialist, for this exercises, with two specialists to represent, why don't we give each of them a separated table to work on?

To have one specialists examining you, the patient only need to step in the table of the specialist or through the $Spec\_1's$ transition and $Spec\_2's$ transition.

For every transitions, we need a guard. For our Petri net, we use a function with the return type of Boolean as a guard: $Is\_Spec\_1(spec)$ for taking the specialist $Spec\_1$ and Is_Spec_2 (spec) for taking the specialist $Spec\_2$. With these guards, the enable binding of transitions for the two specialist is well distinguish.

In our opinion, this Petri net is well designed for the purpose of representing the process with two specialists distinguishablely at a time since each marking has only one enabled transition and one enabled binding.

3.7 Implementing Item 1, 2, 3, 4 with codes

Write a computational package to realize (implement) Items 1, 2, 3, 4 above

The program should allow input with max 10 patients in place *wait*.

Solution:

Because a Petri net can be understood as a bipartite graph, along with the flow relation F being the subset of the two Cartesian products $T \times P$ and $P \times T$, we can build the computational program for Petri net using an incidence matrix. The flow relation now can be referred to as a 2D square matrix with the size being the sum of the number of places and transitions in the net. We explore the concept of **weight function** w as a function that maps to either 0 or 1. For an ordered pair (x, y), if $(x, y) \in F$, or in other words, from x we have an arc going to y, then $w((x, y)) = 1$, otherwise $w((x, y)) = 0$.

We go further to talk a bit differently about marking, enabledness, and firing. A marking can be denoted as a function m which assigns a natural number to a place $p \in P$. For all places in the preset of a transition $t \in T$, if they each contains at least one token then the transition t is enabled at that particular marking. Mathematically, a transition t is enabled if and only if $\forall p \in \cdot t, m(p) > 0$. When it comes to firing, a transition consumes token from its input places and produces to every of it output places a token. We are able to calculate the successor marking M' from the initial marking M with the formula incorporating the concept of weight functions mentioned earlier:

$$m'(p) = m(p) - w((p, t)) + w((t, p)) \tag{1}$$

The number of consumed tokens is equal to 1 if p is an input place of t and 0 else. The same goes for the number of produced tokens.

For the sake of simplicity, we are going to implement the Petri nets in this assignment with Python. The source code comes in .py files in the compressed submission file, or can be found in this Pastebin folder, or this GitHub repo. The weight functions in the code segments are implemented with 2D lists, and the function for firing is key for the exercise. It follows closely Formula 1.

```python
def fire(originalMarking, weight1, weight2):
  return originalMarking - weight1 + weight2;
```

Other than that, it is pretty much the same thing over the Petri nets. The essentials are pretty much taken care of, and in each source code, we only need to correctly identify which matrix entry belongs to which place or transition.

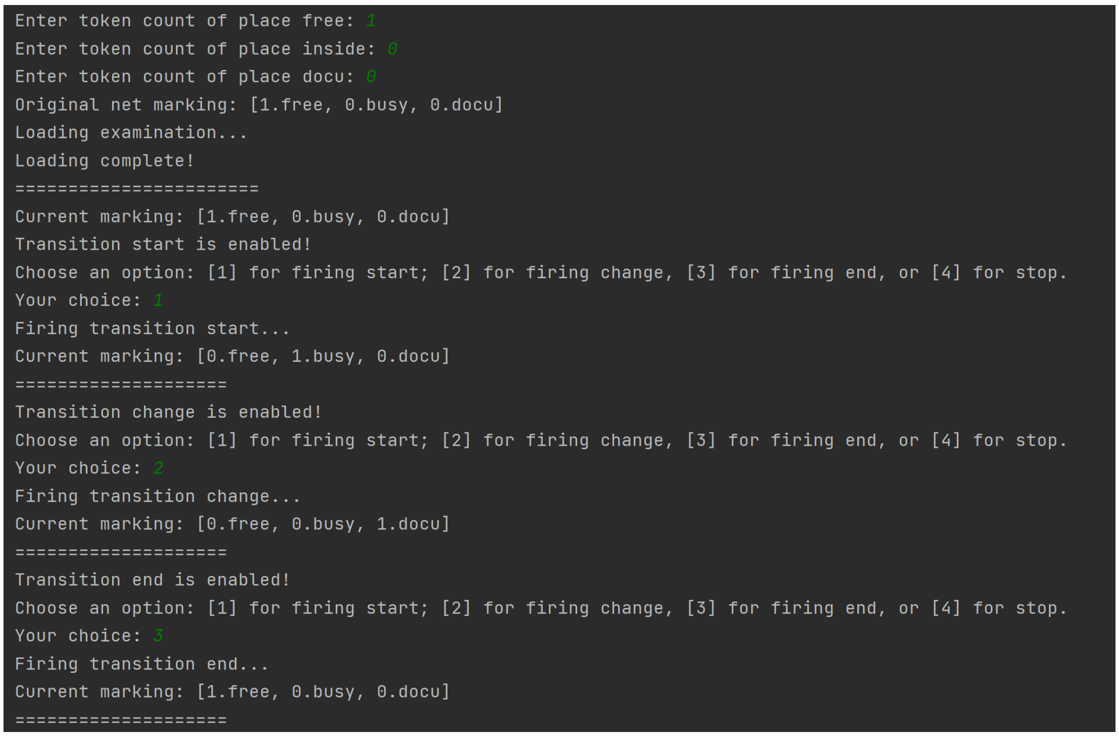
3.7.1 Implementing specialist's net N_S

```python
weight_matrix = [
    [0, 1, 0, 0, 0, 0],
    [0, 0, 1, 0, 0, 0],
    [0, 0, 0, 1, 0, 0],
    [0, 0, 0, 0, 1, 0],
    [0, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 0]
]
# left-to-right fashion
# row-wise: free, start, busy, change, docu, end
# column-wise: free, start, busy, change, docu, end
# eg: a[0][1] == 1: free -> start.
# eg: a[3][4] == 1: change -> docu.
# eg: a[0][2] == 0: free --x--> busy
# place1 --x--> place2

marking = [0, 0, 0]


def fire(originalMarking, weight1, weight2):
    return originalMarking - weight1 + weight2


marking[0] = int(input("Enter token count of place free: "))
marking[1] = int(input("Enter token count of place inside: "))
marking[2] = int(input("Enter token count of place docu: "))
print("Original net marking: " + "[" + str(marking[0]) + ".free" + ", " + str(marking[1]) + ".busy" + ", " + str(
    marking[2]) + ".docu" + "]")

#  we move on to set the initial enabledness
if marking[0] > 0:
    start = True
    #  marking[0] represents place free. If it has token then transition start is enabled (i.e. Boolean true)
else:
    start = False

if marking[1] > 0:
    change = True
    #  marking[1] represents place busy. If it has token then transition change is enabled (i.e. Boolean true)

else:
    change = False

if marking[2] > 0:
```

```python
    end = True
    #  marking[2] represents place docu. If it has token then transition end is enabled (i.e. Boolean true)
else:
    end = False
print("Loading examination...")
print("Loading complete!")
print("======================")
print("Current marking: " + "[" + str(marking[0]) + ".free" + ", " + str(marking[1]) + ".busy" + ", " + str(
    marking[2]) + ".docu" + "]")
premature = False
while start == True or change == True or end == True:
    if start:
        print("Transition start is enabled!")
    if change:
        print("Transition change is enabled!")
    if end:
        print("Transition end is enabled!")

    print("Choose an option: [1] for firing start; [2] for firing change, [3] for firing end, or [4] for stop.")
    choice = int(input("Your choice: "))
    if choice == 1:
        if not start:
            print("Can't fire, transition start is not enabled")
        else:
            print("Firing transition start...")
            marking[0] = fire(marking[0], weight_matrix[0][1], weight_matrix[1][0])
            marking[1] = fire(marking[1], weight_matrix[2][1], weight_matrix[1][2])
            print("Current marking: " + "[" + str(marking[0]) + ".free" + ", " + str(marking[1]) + ".busy" + ", " + str(
                marking[2]) + ".docu" + "]")
    elif choice == 2:
        if not change:
            print("Can't fire, transition change is not enabled")
        else:
            print("Firing transition change...")
            marking[1] = fire(marking[1], weight_matrix[2][3],
weight_matrix[3][2])
            marking[2] = fire(marking[2], weight_matrix[4][3], weight_matrix[3][4])
            print("Current marking: " + "[" + str(marking[0]) + ".free" + ", " + str(marking[1]) + ".busy" + ", " + str(
                marking[2]) + ".docu" + "]")
    elif choice == 3:
        if not end:
            print("Can't fire, transition end is not enabled")
        else:
            print("Firing transition end...")
            marking[2] = fire(marking[2], weight_matrix[3][4], weight_matrix[4][3])
            marking[0] = fire(marking[0], weight_matrix[0][5], weight_matrix[5][0])
            print("Current marking: " + "[" + str(marking[0]) + ".free" + ", " + str(marking[1]) + ".busy" + ", " + str(
                marking[2]) + ".docu" + "]")
    elif choice == 4:
        premature = True
        print("Done examining the net.")
        print("Current marking is: " + "[" + str(marking[0]) + ".free" + ", " + str(marking[1]) + ".busy" + ", " + str(
            marking[2]) + ".docu" + "]")
        break
    else:
        print("Invalid choice!")
    if marking[0] > 0:
        start = True
    else:
        start = False
    # ====
    if marking[1] > 0:
        change = True
    else:
        change = False
    if marking[2] > 0:
        end = True
    else:
        end = False
    print("====================")
```

```
if not premature:
    print("Done.")
    print("Terminal marking is: " + "[" + str(marking[0]) + ".free" +
    ", " + str(marking[1]) + ".busy" + ", " + str(
        marking[2]) + ".docu" + "]")
```

With the initial marking of exercise 1 being $[1.free]$, we run an exact example with our source code, illustrated in Fig. 13. This is a closed loop, so it would never reach a terminal marking. We included in the source code a choice to end examining, because if not then the simulation would run forever.



```
Enter token count of place free: 1
Enter token count of place inside: 0
Enter token count of place docu: 0
Original net marking: [1.free, 0.busy, 0.docu]
Loading examination...
Loading complete!
=======================
Current marking: [1.free, 0.busy, 0.docu]
Transition start is enabled!
Choose an option: [1] for firing start; [2] for firing change, [3] for firing end, or [4] for stop.
Your choice: 1
Firing transition start...
Current marking: [0.free, 1.busy, 0.docu]
====================
Transition change is enabled!
Choose an option: [1] for firing start; [2] for firing change, [3] for firing end, or [4] for stop.
Your choice: 2
Firing transition change...
Current marking: [0.free, 0.busy, 1.docu]
====================
Transition end is enabled!
Choose an option: [1] for firing start; [2] for firing change, [3] for firing end, or [4] for stop.
Your choice: 3
Firing transition end...
Current marking: [1.free, 0.busy, 0.docu]
====================
```

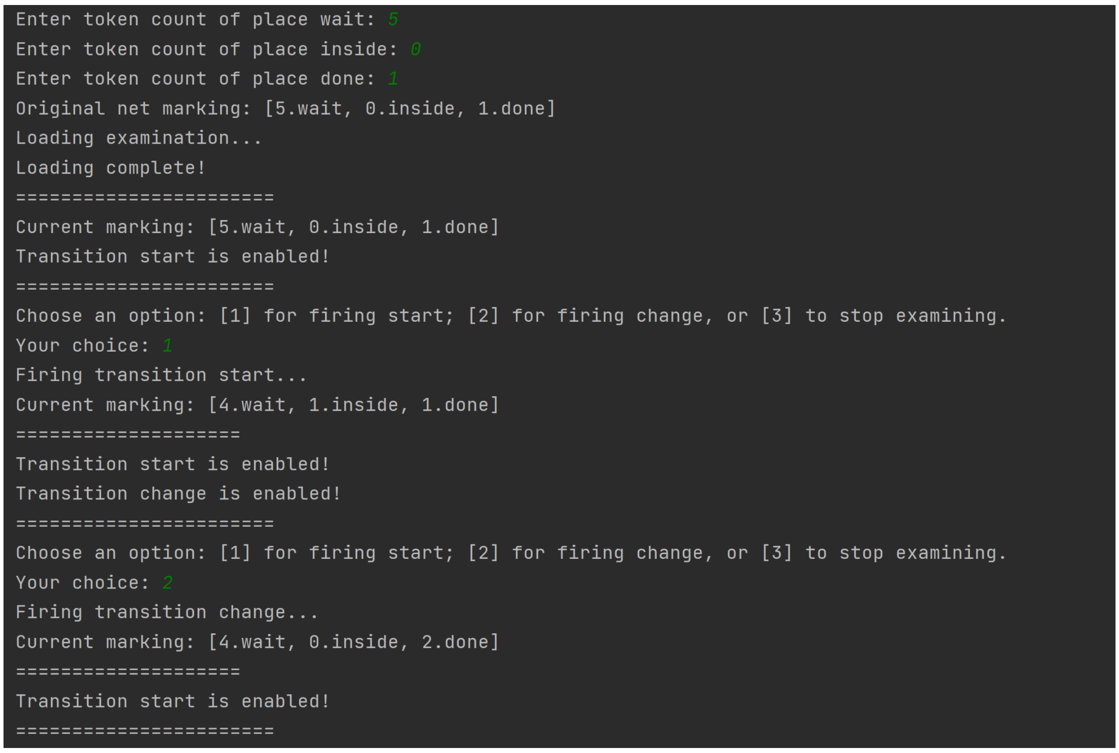
Figure 13: Demonstration of the N_S net, with $[1.free]$ as initial marking.

3.7.2 Implementing patient's net N_{Pa}

```
weight_matrix = [
    [0, 1, 0, 0, 0],
    [0, 0, 1, 0, 0],
    [0, 0, 0, 1, 0],
    [0, 0, 0, 0, 1],
    [0, 0, 0, 0, 0]
]


# left-to-right fashion
# row-wise: wait, start, inside, change, done
# column-wise: wait, start, inside, change, done
# eg: a[0][1] == 1: wait -> start.
# eg: a[3][4] == 1: change -> done.
# eg: a[0][2] == 0: wait -x-> inside
# place1 --x--> place2
marking = [0, 0, 0]


def fire(originalMarking, weight1, weight2):
    return originalMarking - weight1 + weight2


marking[0] = int(input("Enter token count of place wait: "))
marking[1] = int(input("Enter token count of place inside: "))
marking[2] = int(input("Enter token count of place done: "))
print("Original net marking: " + "[" + str(marking[0]) + ".wait" + ",
    " + str(marking[1]) + ".inside" + ", " + str(
    marking[2]) + ".done" + "]")
if marking[0] > 0:
    start = True
    #  marking[0] represents place wait. If it has token then
    transition start is enabled (i.e. Boolean true)
else:
    start = False
# ====
if marking[1] > 0:
    change = True
    #  marking[1] represents place inside. If it has token then
    transition change is enabled (i.e. Boolean true)
else:
    change = False
print("Loading examination...")
print("Loading complete!")
print("======================")
print("Current marking: " + "[" + str(marking[0]) + ".wait" + ", " +
    str(marking[1]) + ".inside" + ", " + str(
    marking[2]) + ".done" + "]")
```

```python
premature = False  # used to track the premature end of examination, so that it prints "current marking"
# instead of "terminal marking"
while start == True or change == True:
    if start:
        print("Transition start is enabled!")
    if change:
        print("Transition change is enabled!")
    print("=======================")
    print("Choose an option: [1] for firing start; [2] for firing change, or [3] to stop examining.")
    choice = int(input("Your choice: "))
    if choice == 1:
        if not start:
            print("Can't fire, transition start is not enabled!")
        else:
            print("Firing transition start...")
            marking[0] = fire(marking[0], weight_matrix[0][1], weight_matrix[1][0])
            marking[1] = fire(marking[1], weight_matrix[2][1], weight_matrix[1][2])
            print(
                "Current marking: " + "[" + str(marking[0]) + ".wait" + ", " + str(marking[1]) + ".inside" + ", " + str(
                    marking[2]) + ".done" + "]")
    elif choice == 2:
        if not change:
            print("Can't fire, transition change is not enabled!")
        else:
            print("Firing transition change...")
            marking[1] = fire(marking[1], weight_matrix[2][3], weight_matrix[3][2])
            marking[2] = fire(marking[2], weight_matrix[4][3], weight_matrix[3][4])
            print(
                "Current marking: " + "[" + str(marking[0]) + ".wait" + ", " + str(marking[1]) + ".inside" + ", " + str(
                    marking[2]) + ".done" + "]")
    elif choice == 3:
        premature = True
        print("Done examining the net.")
        print(
            "Current marking is: " + "[" + str(marking[0]) + ".wait" + ", " + str(marking[1]) + ".inside" + ", " + str(
                marking[2]) + ".done" + "]")
        break
    else:
        print("Invalid choice!")
    if marking[0] > 0:
        start = True
    else:
        start = False
    # ====
    if marking[1] > 0:
        change = True
    else:
        change = False
    print("====================")
if not premature:
    print("Done examining the net.")
    print("Terminal marking is: " + "[" + str(marking[0]) + ".wait" + ", " + str(marking[1]) + ".inside" + ", " + str(
        marking[2]) + ".done" + "]")
```

With the initial marking of exercise 2 being $[5.wait, 1.done]$, we run an exact example with our source code, illustrated in Fig. 14.



```
Enter token count of place wait: 5
Enter token count of place inside: 0
Enter token count of place done: 1
Original net marking: [5.wait, 0.inside, 1.done]
Loading examination...
Loading complete!
=======================
Current marking: [5.wait, 0.inside, 1.done]
Transition start is enabled!
=======================
Choose an option: [1] for firing start; [2] for firing change, or [3] to stop examining.
Your choice: 1
Firing transition start...
Current marking: [4.wait, 1.inside, 1.done]
====================
Transition start is enabled!
Transition change is enabled!
=======================
Choose an option: [1] for firing start; [2] for firing change, or [3] to stop examining.
Your choice: 2
Firing transition change...
Current marking: [4.wait, 0.inside, 2.done]
====================
Transition start is enabled!
=======================
```

Figure 14: Demonstration of the N_{Pa} net, with $[5.wait, 1.done]$ as initial marking.

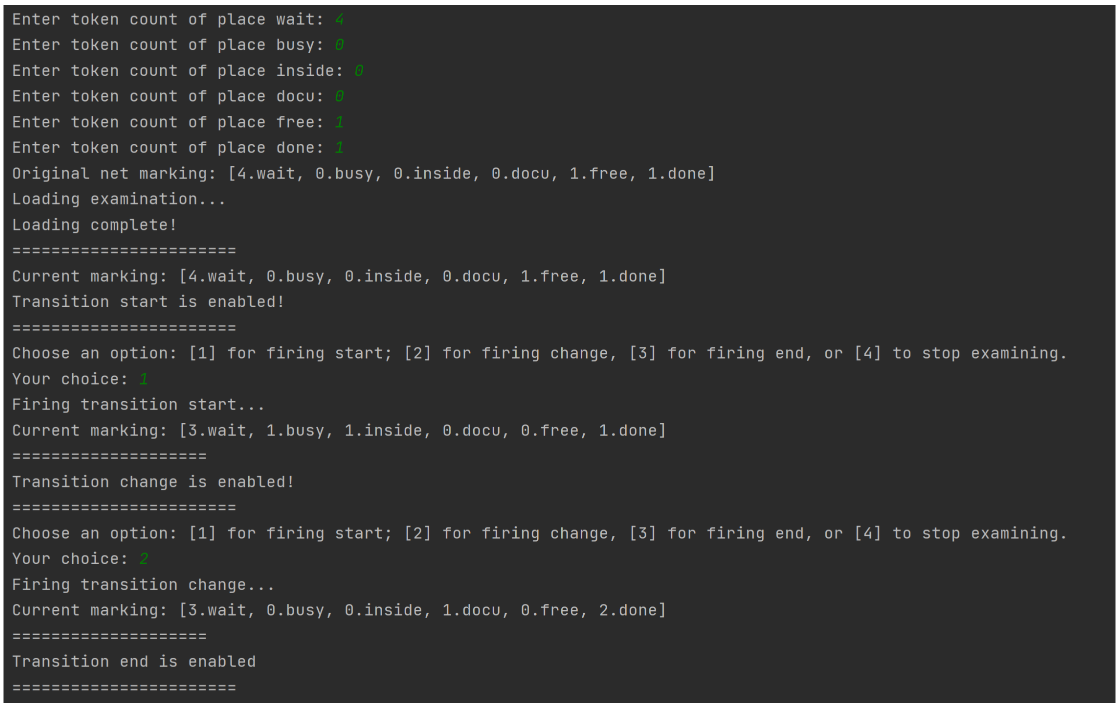
3.7.3 Implementing superimposed net

```python
weight_matrix = [
    [0, 1, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 1, 0, 0, 0, 0, 1, 0],
    [0, 0, 0, 1, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 0, 0, 0, 1],
    [0, 0, 0, 0, 0, 1, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 1, 0, 0],
    [0, 1, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0]
]

# left-to-right fashion
# row-wise: wait, start, busy, change, docu, end, free, inside, done
# column-wise: wait, start, busy, change, docu, end, free, inside, done
# eg: a[0][1] == 1: wait -> start.
# eg: a[3][4] == 1: change -> docu.
# eg: a[0][2] == 0: wait -x-> busy
# place1 --x--> place2
marking = [0, 0, 0, 0, 0, 0]  # there are 6 places now


def fire(originalMarking, weight1, weight2):
    return originalMarking - weight1 + weight2


marking[0] = int(input("Enter token count of place wait: "))
marking[1] = int(input("Enter token count of place busy: "))
marking[2] = int(input("Enter token count of place inside: "))
marking[3] = int(input("Enter token count of place docu: "))
marking[4] = int(input("Enter token count of place free: "))
marking[5] = int(input("Enter token count of place done: "))
print("Original net marking: " + "[" + str(marking[0]) + ".wait" + ", 
    " + str(marking[1]) + ".busy" + ", " + str(
    marking[2]) + ".inside" + ", " + str(marking[3]) + ".docu" + ", "
    + str(marking[4]) + ".free" + ", " + str(
    marking[5]) + ".done" + "]")
if marking[0] > 0 and marking[4] > 0:
    start = True
    #  marking[0] and marking[4] represent places wait and free,
    respectively. If they both have tokens then transition start is
    enabled (i.e. Boolean true)
else:
    start = False

if marking[1] > 0 and marking[2] > 0:
    change = True
    #  marking[1] and marking[2] represent places busy and inside,
    respectively. If they both have tokens then transition start is
    enabled (i.e. Boolean true)
else:
    change = False

if marking[3] > 0:
    end = True
    #  marking[3] represents place docu. If it has token then
    transition start is enabled (i.e. Boolean true)
```

```
else:
    end = False

print("Loading examination...")
print("Loading complete!")
print("======================")
print("Current marking: " + "[" + str(marking[0]) + ".wait" + ", " + str(marking[1]) + ".busy" + ", " + str(
    marking[2]) + ".inside" + ", " + str(marking[3]) + ".docu" + ", " + str(marking[4]) + ".free" + ", " + str(
    marking[5]) + ".done" + "]")
premature = False  # used to track the premature end of examination, so that it prints "current marking"
# instead of "terminal marking"
while start == True or change == True or end == True:
    if start:
        print("Transition start is enabled!")
    if change:
        print("Transition change is enabled!")
    if end:
        print("Transition end is enabled!")
    print("======================")
    print(
        "Choose an option: [1] for firing start; [2] for firing change, [3] for firing end, or [4] to stop examining.")
    choice = int(input("Your choice: "))
    if choice == 1:
        if not start:
            print("Can't fire, transition start is not enabled!")
        else:
            print("Firing transition start...")
            marking[0] = fire(marking[0], weight_matrix[0][1], weight_matrix[1][0])
            marking[1] = fire(marking[1], weight_matrix[2][1], weight_matrix[1][2])
            marking[2] = fire(marking[2], weight_matrix[7][1], weight_matrix[1][7])
            marking[4] = fire(marking[4], weight_matrix[6][1], weight_matrix[1][6])
            print(
                "Current marking: " + "[" + str(marking[0]) + ".wait"
    + ", " + str(marking[1]) + ".busy" + ", " + str(
                    marking[2]) + ".inside" + ", " + str(marking[3]) +
     ".docu" + ", " + str(marking[4]) + ".free" + ", " + str(
                    marking[5]) + ".done" + "]")
    elif choice == 2:
        if not change:
            print("Can't fire, transition change is not enabled!")
        else:
            print("Firing transition change...")
            marking[1] = fire(marking[1], weight_matrix[2][3],
    weight_matrix[3][2])
            marking[2] = fire(marking[2], weight_matrix[7][3],
    weight_matrix[3][7])
            marking[3] = fire(marking[3], weight_matrix[4][3],
    weight_matrix[3][4])
            marking[5] = fire(marking[5], weight_matrix[8][3],
    weight_matrix[3][8])
            print(
                "Current marking: " + "[" + str(marking[0]) + ".wait"
    + ", " + str(marking[1]) + ".busy" + ", " + str(
                    marking[2]) + ".inside" + ", " + str(marking[3]) +
     ".docu" + ", " + str(marking[4]) + ".free" + ", " + str(
                    marking[5]) + ".done" + "]")
    elif choice == 3:
        if not end:
            print("Can't fire, transition end is not enabled!")
        else:
            print("Firing transition end...")
            marking[3] = fire(marking[3], weight_matrix[4][5],
    weight_matrix[5][4])
            marking[4] = fire(marking[4], weight_matrix[6][5],
    weight_matrix[5][6])
            print(
                "Current marking: " + "[" + str(marking[0]) + ".wait"
    + ", " + str(marking[1]) + ".busy" + ", " + str(
                    marking[2]) + ".inside" + ", " + str(marking[3]) +
     ".docu" + ", " + str(marking[4]) + ".free" + ", " + str(
                    marking[5]) + ".done" + "]")
    elif choice == 4:
        premature = True
        print("Done examining the net.")
```

```python
        print(
            "Current marking is: " + "[" + str(marking[0]) + ".wait" + ", " + str(marking[1]) + ".busy" + ", " + str(
                marking[2]) + ".inside" + ", " + str(marking[3]) + ".docu" + ", " + str(marking[4]) + ".free" + ", " + str(
                marking[5]) + ".done" + "]")
        break
    else:
        print("Invalid choice!")
    if marking[0] > 0 and marking[4] > 0:
        start = True
        #  marking[0] represents place wait. If it has token then transition start is enabled (i.e. Boolean true)
    else:
        start = False

    if marking[1] > 0 and marking[2] > 0:
        change = True
        #  marking[1] represents place inside. If it has token then transition start is enabled (i.e. Boolean true)
    else:
        change = False

    if marking[3] > 0:
        end = True
        #  marking[3] represents place inside. If it has token then transition end is enabled (i.e. Boolean true)
    else:
        end = False
    print("====================")
if not premature:
    print("Done examining the net.")
    print("Terminal marking is: " + "[" + str(marking[0]) + ".wait" + ", " + str(marking[1]) + ".busy" + ", " + str(
        marking[2]) + ".inside" + ", " + str(marking[3]) + ".docu" + ", " + str(marking[4]) + ".free" + ", " + str(
        marking[5]) + ".done" + "]")
```

Let us run an example with the given initial marking from Item 4. Figure 15 showcases the execution of the program.

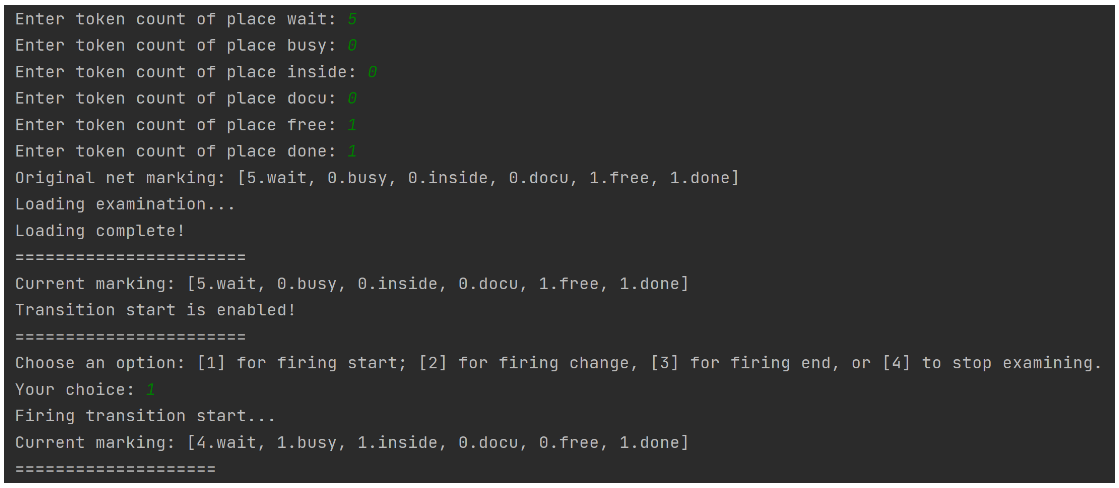


```
Enter token count of place wait: 4
Enter token count of place busy: 0
Enter token count of place inside: 0
Enter token count of place docu: 0
Enter token count of place free: 1
Enter token count of place done: 1
Original net marking: [4.wait, 0.busy, 0.inside, 0.docu, 1.free, 1.done]
Loading examination...
Loading complete!
=======================
Current marking: [4.wait, 0.busy, 0.inside, 0.docu, 1.free, 1.done]
Transition start is enabled!
=======================
Choose an option: [1] for firing start; [2] for firing change, [3] for firing end, or [4] to stop examining.
Your choice: 1
Firing transition start...
Current marking: [3.wait, 1.busy, 1.inside, 0.docu, 0.free, 1.done]
====================
Transition change is enabled!
=======================
Choose an option: [1] for firing start; [2] for firing change, [3] for firing end, or [4] to stop examining.
Your choice: 2
Firing transition change...
Current marking: [3.wait, 0.busy, 0.inside, 1.docu, 0.free, 2.done]
====================
Transition end is enabled
=======================
```

Figure 15: Merged Petri net with initial marking $[4.wait, 1.free, 1.done]$.

3.7.4 Explore Item 4 with codes

The result shown in Figure 16 conforms to the written answer in Item 4 above. After firing transition *start*, a new marking is enabled. Initially, only *start* is enabled.



```
Enter token count of place wait: 5
Enter token count of place busy: 0
Enter token count of place inside: 0
Enter token count of place docu: 0
Enter token count of place free: 1
Enter token count of place done: 1
Original net marking: [5.wait, 0.busy, 0.inside, 0.docu, 1.free, 1.done]
Loading examination...
Loading complete!
=======================
Current marking: [5.wait, 0.busy, 0.inside, 0.docu, 1.free, 1.done]
Transition start is enabled!
=======================
Choose an option: [1] for firing start; [2] for firing change, [3] for firing end, or [4] to stop examining.
Your choice: 1
Firing transition start...
Current marking: [4.wait, 1.busy, 1.inside, 0.docu, 0.free, 1.done]
====================
```

Figure 16: Item 4 illustrated with code.

4 Conclusion

The Petri net with its application can be the key to modeling business processes very effectively. We understand that in this assignment we are just scratching the surface, but the power of Petri nets can be extended quite a lot with color, and even time. It took a lot of effort to not just solve 7 exercises, but also to understand Petri nets clearly. We hope that it has set a foundation to a bright Final exam as well as our own knowledge, our vision into how a business process should be modeled and understood.

References

[1] W.M.P Aalst, van der (2016) *Part I of PROCESS MINING*, 2nd edition, Springer.

[2] Cleiton dos Santos Garcia et. al. (2019) *Process mining techniques and applications- A systematic mapping study, in Expert Systems With Applications*, Vol 133 260-295, Elsevier.

[3] Kurt Jensen (1992) *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use*, volume 1, Springer.

[4] Wil van der Aalst and Christian Stahl (2011) *Modeling Business Processes - A Petri Net-Oriented Approach*, The MIT Press, England