Thesis Dissertation

A SIMULATOR FOR REVERSING PETRI NETS BASED ON A MATRIX-EQUATION REPRESENTATION

Maria Psara

UNIVERSITY OF CYPRUS

COMPUTER SCIENCE DEPARTMENT

May 2018

UNIVERSITY OF CYPRUS
COMPUTER SCIENCE DEPARTMENT

A Simulator for Reversing Petri Nets Based on a Matrix-Equation Representation

Maria Psara

Supervisor
Dr. Anna Philippou

Thesis submitted in partial fulfilment of the requirements for the award of degree of Bachelor in Computer Science at University of Cyprus

May 2018

Acknowledgments

Primarily, I would like to express my gratitude to my supervisor Dr. Anna Philippou for her help, her support and the very good cooperation we have had throughout this diploma thesis. The incessant aid, support and guidance I have had throughout this project gave me self-confidence and strength to keep going. I want to signalize that she had provided me with opportunities and new knowledge which will be very lucrative for my future career. The occasion to get involved in this absorbing project gave me the opportunity to explore intriguing areas.

Moreover, I want to thank the PhD student, Kyriaki Psara for her unlimited help and patience during this thesis. She contributed in the construction of my project through precious advice. Her experience in this area provided me with immediate support in hard times and in major decision making.

I would also like to thank many of my fellow students and friends, because they were there for me every step of the way, and I really appreciate everything they have done for me.

Finally, a huge thank you I would like to address to my family that did not leave me alone. Each member of my family it was next to me in a different way, supported me at any time I needed it and helped me to deal with every problem I face. That's why I want to dedicate this diploma thesis to them and thank them deeply for the patience and tolerance that they have indicated to me throughout my studies.

Abstract

This diploma thesis deals with the issue of reversible computation. Computation is said to be reversible if it has the ability to execute in reverse, so that computation can go back to previous visited states or even reaching new states, as well as it can proceed in the forward direction. Different guises of reversibility can be found in systems from various fields. Such systems have been investigated by means of reversible formal languages, including reversible PNs (RPNs).

PNs have been used extensively to model systems both in graphical representation as well as the mathematical alternative. The mathematical form represents PNs in a series of matrix equations which can be used to specify and manipulate the state of the system in a simulator. Thus, we set out to study the matrix equations of RPNs by exploring the modelling of the main strategies for reversible computation.

The definition and creation of matrix equations for Reversing Petri nets has been used for the implementation of an RPN simulator. This simulator gives to users the opportunity to import a Reversing Petri net's information, which can be expressed in the form of matrices. The creation of the corresponding matrices, gives users the opportunity to execute a specific transition, both in forward and reverse direction, and to see how the marking of the net evolves in a computation. For the needs of the simulator some algorithms have been implemented in the Java programming language and are based on the matrix equations that have been defined.

After the completion of have the simulator we examined how the simulator of Reversing Petri nets and, therefore, the matrices, can be applied on a reversible system simulating the product assembly system, and whether or not it is possible to disassemble the product by using the same Reversing Petri net model and the matrices that are derived from it. Moreover, we discuss the usefulness of the approach in the context of manufacturing task planning, a field where reversibility plays a crucial role both in the distinct processes of assembly and disassembly and also as a means of recovering from failures during the assembly process.

Contents

List of Figures

Chapter 1

Introduction

Contents

1.1 Motivation

Since PNs are able to model discrete event systems, it is helpful to have a system of equations which can be used for specifying and manipulating the states of a system. Matrix representation can be mechanised very easily and it can also be used to represent the dynamic behaviour of PNs, study the coverability and reachability problems, and study properties such as boundedness, invariance, conservativeness and liveness.

The automatic generation of assembly and disassembly sequence requires proper modelling and representation of the assembly network that aims to generate the most efficient assembly sequence. Modelling a disassembly process should consider the product topology, mating relations, and precedence relations. The modelling of assembly and disassembly by Petri nets is appealing because it is simple in its application, visually comprehensible, and allows computer manipulation. Another advantage in Petri net representation is that it can be further extended or modified to accommodate specific attributes required for modelling of different systems.

A principal process during the re-manufacturing of worn-out or malfunctioning products is disassembly that enables the dumping, cleaning, repair or replacement of components as desired. It is essentially the inverse of an assembly process that decomposes products into parts or sub-assemblies. Therefore, reversible computation is by nature embedded to real life applications of assembly and disassembly. Hence, the research issues that we address in this work include the modelling of disassembly processes using RPNs that offer the flexibility to provided program control using the reversible computation strategies. Creating matrix equations helps to optimise the assembly operations, automatically recover from the errors during the execution, and visualise an assembly process in a quick and intuitive manner using computer graphics and Reversing Petri nets.

1.2 Work Purpose

The main aim of this thesis project is to present an overview of the fundamental matrix equations that provide the basis for calculating the dynamic behaviour of RPNs. We set out to study the matrix equations of RPNs by exploring the modelling of the main strategies for reversible computation, namely, of *backtracking*, *causal reversibility* and *out-of-causal-order reversibility*. Upon the development of the matrices, this thesis is called to answer the question of whether the created matrices can be used for the modelling and representation of an assembly network.

1.3 Work Methodology

In the first phase of the study, a theoretical study was conducted. Initially, an extensive study and research was carried out in various scientific papers, based on reversibility, Petri nets and Assembly/Disassembly processes, in order to fully understand and ensure the appropriate knowledge of how the different semantic models of this study have emerged. With the completion of the above, the Reversing Petri nets were extensively studied in order to better understand their operations and the changes that had been made in relation to the Petri nets model.

In the second phase of the thesis, all the knowledge we acquired from the first phase was combined to create the corresponding matrices and their equations that met the needs of the functions of the RPNs. These matrices were created to express all the information of a Reversing Petri net model. By performing some matrices equations of addition, subtraction and multiplication, we were able to perform operations corresponding to the current operation semantics of the model. The result of these equations is to take as an output the new marking of the RPN model.

In the third phase of the work, an experimental study of the matrices that were created in the previous phase, was carried out. In particular, we have checked whether these matrices can be used to model and express an assembly network. Disassembly is essentially the inverse of an assembly process that decomposes products into parts or subassemblies and therefore naturally follows the principles of reversible computation. The method delineates the dynamics of the individual tasks, and emphasises a discrete system-oriented approach.

After extensive study, the code for the implementation of the particular simulator began. The implementation of the algorithms was done in Java programming language. Initially, the three basic equations for the possible execution modes in RPNs were implemented. For the implementation of these procedures, there were other auxiliary functions that broke the basic equations into smaller ones or performing a particular calculation of the model. It also required the implementation of functions that were designed to make math operations between the matrices, namely the operations of addition, subtraction, and multiplication.

From the examples and audits we performed, we came to the conclusion that the results in this research can be applied to many other types of applications, except from assembly/disassembly planning, such as in machining and human operation modelling.

1.4 Thesis Structure

In Chapter 2 there is a historical review of reversibility as a concept and the research that has been centred on this field. Below are some areas where reversible modelling has been applied, as well as the forms of reversibility that exist. Finally, reference is made to the form and functions of Petri nets and to the Reversing Petri nets, which are a kind of extension of PN models.

Chapter 3 presents separately the four forms of execution that exist in Reversing Petri nets. In each case, once the matrix equations have been created, an example of the specific form of computation is executed on an RPN. These matrices, through a series of addition, subtraction and multiplication operations, help to calculate the new marking M of a network as well as the new emerging history.

Chapter 4 presents information on the creation and implementation of the simulator. Initially, the algorithms created for the purpose of implementing the various system functions and matrices are presented. After that, we explain the operations of the simulator via the interface that has been created in Java programming language.

Chapter 5 presents a complete example of a case study. With the use of Reversing Petri nets and matrix equations of Chapter 3, we assembly this object and disassembly it when that is necessary.

Finally, in Chapter 6 we draw a general conclusion, mention the problems we encountered and how we dealt with them, as well as some future extensions that can be made to the program to further expand it.

The complete Java code implementations for the algorithms listed in Chapter 4 as well as the implementation of the interface of the simulator, are found in Appendix A and B respectively.

Chapter 2

Related Work

Contents

2.1 Reversible Computation

Reversibility is a fundamental concept in sciences and is inherent in Mathematics, Physics, Biology, and Computer Science. In the field of Computer Science, the concept of reversibility is expressed in the form of computations. Reversible computation, in a general sense, means computing using reversible operations, that is, operations that can be easily and exactly reversed, or undone. More specifically, it is an unconventional form of computing where any executed sequence of operations can be executed in reverse at any point during computation.

The history of reversible computation starts in 1961, when physicist Rolf Landauer published a paper with title "Irreversibility and Heat Generation in the Computing Process" [4]. In this paper, Landauer, based on the fact that the most fundamental laws of physics are reversible, and correlating the logical irreversibility with the physical irreversibility, supported that the logically irreversible character of conventional computational operations affects directly the thermodynamic behaviour of the device that is executing these operations.

Reversible features of the laws of physics are based on the fact that, if you know all the information of the state of a closed system at some time, you can apply in a reverse order the laws of physics and find out the exact state of the system at any previous time.

The same fundamental reversibility is valid in quantum physics. Therefore, two different states of any physical system cannot evolve into the exact same state at some later time, as in this case it would be impossible to define the earlier state from a later one. So, to avoid this, we cannot destroy any information at the lowest level in physics, which means that we can never truly erase information in a computer.

Although, in the world of computers, when we overwrite a bit of information with a new value, the previous information is lost for all practical purposes. Instead of a physical destruction, the previous information is pushed into the machine's thermal environment, where it becomes entropy and manifests as heat.

At the time of Landauer's paper, people used to believe that the process of deleting information was a consequence of a computation process that you could not avoid. As Landauer described in his embedding [4], all irreversible operations can be transformed into a reversible operation. However, he believed that these transformation techniques could only be used to store temporarily each gate's inputs, which later on should be deleted.

In 1963, Lecerf described for the first time the reversible Turing machine, in a paper titled "Reversible Turing machines" [5]. This paper described a new method in which the computational history was saved and then decomputed away. This was Lecerf's reversal method to uncompute histories. However, since he did not focus on the thermodynamic applications, his machines did not save their outputs, so they were not very useful in the physically reversible setting.

Almost 10 years later, in 1973, a breakthrough came to light by Bennett where he defined the first universal reversible computation model. Bennett actually repeated Lecerf's reversal by adding the Bennett trick which copied the output before uncomputing the undo trail. In this way he managed to limit the conventional (irreversible) Turing machines (TMs) and to define the reversible Turing machines (RTMs). This paper proved for the first time that reversible computations can avoid entropy production, and pointed out the possibility of a physically reversible computer in which the energy leakage is arbitrarily small.

Logic, from the beginning, has had a central role in reversible computation. For example, the ideas of Landauer for reversible gates were invented as a way to decrease the heat dissipation of logic circuits.

Fredkin and Toffoli, in 1978, inspired by Landauer's and Bennett's work, reinvented reversible computing in the form of conservative logic circuits [2]. In a conservative logic circuit all logic gates must be reversible and also maintain their parity. For this reason, Fredkin and Toffoli, introduce in this paper the Fredkin gate, a gate where the output variables are expressed as explicit functions of the input variables. This functional relationship between input and output variables is invertible.

Later on, in 1980, Toffoli invented the n-bit controlled-not gates (shown in Figure 2.1), also known as Toffoli gates [9]. This is perhaps the most used class of reversible gates today. The fact that the gates are using a simple mathematical definition has rendered reversible logic synthesis much easier.

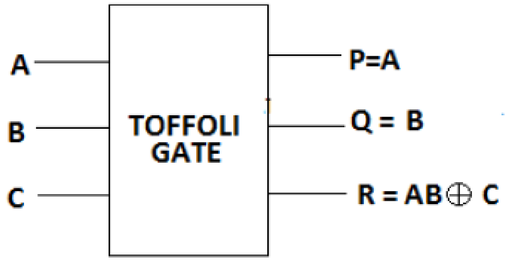


Figure 2.1: Toffoli Gates

Other than reversible circuits, in the last decade, reversibility has found many inspiring and important applications. For example, reversibility is useful in fault-tolerance, the assembly/ disassembly process, simulation, quantum computing, reversible operational semantics, causality and reversibility, and many other.

Several promising applications are enabled due to the prototype reversible circuits which are, indeed, much greater in many aspects than conventional devices. Furthermore, the inherent properties of the reversible computing paradigm can be used in the design of conventional circuits and systems in such a way that the technology will remain unaltered. One of the most basic application domains of reversible circuit design is quantum computational circuits.

Quantum computation [1] is an area which offers the promise of much more efficient computation of problems that are difficult for today's conventional computers. Any quantum operation is inherently reversible, since quantum computation is based on physics laws and as we know, the most fundamental laws of physics are reversible. By taking advantage of the physical effects of superposition and entanglement, quantum computation leads us to a qualitatively new computation example. In quantum mechanical computation models, all quantum gates are reversible, since the events occur by unitary transformations.

Reversible computation can be considered to be a subset of quantum computing, which is easier to work with and can, thus, be utilized for this purpose. One of the benefits of the exploitation of reversible circuit design in the design of conventional systems is the ability to undo any operation when an error state has been entered. Also, the full connectivity of a reversible circuit makes the detection of an error much easier, only by applying few randomly generated stimuli. In such type of circuits there is as well easy testability since reversible computation allows perfect controllability and observability.

Reversibility can also be used for the development of reliable software/systems, since, if any trouble occurs, it gives the opportunity to go back to past safe states, and from there try to explore a new direction in such a way as to avoid the problematic actions and the unsafe states. Also, it can be naturally applied in the debugging process, where when an error occurs, reversibility can help to backtrack the execution until reaching the point where the error was created.

Robotics is another area in which reversibility plays an important role. A main point of this field is that a robot performs many reversible actions in a real environment. As an example, we can consider the disassembly process that a robot performs which is actually the inverse of an assembly sequence of operations. Reversibility is a helpful high-level mechanism for describing operation sequences and may also allow someone to have a practical reversible behaviour on hand.

2.2 Reversible Modelling

In the field of mathematics and computer science, the use of the term "formal language" expresses a set of symbols or strings, combined with a set of rules that specify their use. Any formal language is a modelling language which can be used for information, knowledge or system description, either in a graphical (e.g. Behaviour trees, Petri nets) or a textual (e.g. CCS, π-calculus) form. Formal languages can be used for various systems modelling over a wide range of sciences, from biology to computer science.

There are different guises of reversibility that can be found in systems. A system is called reversible if it has the ability to reverse certain processes, and return to its initial state, without leaving net effects in any of the systems involved. Debugging systems are a kind of reversible systems, since they provide fault tolerance by allowing the user to return to previous states in the presence of faults, providing in this way high system dependability. Another type of reversible systems is assembly/disassembly systems, where the disassembly process can be done by executing the assembly process in the exact reverse order.

Both debugging and assembly/disassembly systems can be used for fault tolerance, since if you face any error problem you can reverse to previous safe states. Except from this kind of systems we can also find reversibility in natural systems, which are reversible systems in the field of biology and chemistry. A reversible chemical system is actually a chemical reaction that can go in both directions; the reactants - which are the chemicals you start with - can change into the products - which are the chemicals you end up with - and vice versa. Since many reversible systems exist, there is a need to create reversible models in order to better understand their behaviour.

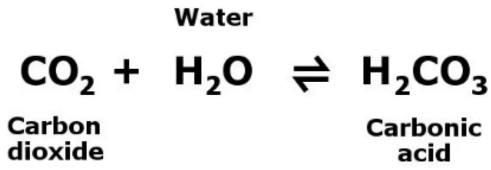


Figure 2.2: An example of reversible chemical reaction; carbonic anhydrase

In the Figure 2.2 above we can see the reactants and the products of the carbonic anhydrases. This kind of reversible reactions form a family of enzymes that catalyze the rapid interconversion of carbon dioxide and water to carbonic acid - which essentially consists of bicarbonate and protons - or vice versa. One of the functions of the enzyme in animals is to interconvert carbon dioxide and bicarbonate to maintain acid-base balance in blood and other tissues, and to help transport carbon dioxide out of tissues.

2.2.1 Forms of Reversibility

Computation is reversible if it has the ability to execute in reverse, so that computation can go back to previously visited states or even reaching new states, as well as it can proceed in the forwards direction. There are three different forms of logic reversibility that can be used in reversible models, which are backtracking, causal reversibility and out-of-causal-order reversibility.

Backtracking is the process of undoing computational steps in the inverse order to the order in which they occurred. This form of reversing can be considered as overly restrictive in the context of concurrent systems since, the reversing of computational moves in the exact order in which they were executed, causes fake causal dependencies on backward sequences of actions.

The second form of reversing is called *causal reversibility* and it is allowing events to reverse in any order assuming that they are independent. Concurrent, distributed or asynchronous computation is considered to be independent since forward steps may carry out independently of each other, and possibly at different locations. Consequently, as long as caused actions are reversed before the actions that have caused them, causal reversibility does not have to execute the exact inverse order for independent steps.

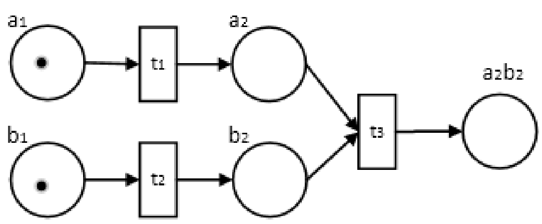


Figure 2.3: Causal Reversibility

For example consider the Petri net in Figure 2.3. We may observe that transitions t_1 and t_2 are independent from each other, as they may be executed in any order, and they are both preconditions for transition t_3. Backtracking the sequence of transitions $\langle t_1, t_2, t_3 \rangle$ would require that the three transitions should be reversed in exactly the reverse order, i.e. $\langle t_3, t_2, t_1 \rangle$. Instead, causal flexibility allows the inverse computation to reverse transition t_3 and then t_1 and t_2 in any order, but never t_1 or t_2 before t_3.

Both forms of backtracking and causal reversing are cause-respecting. However, many examples in real life are undoing things in a seemingly arbitrary order, which includes the reversing of causes before their effects are undone. This is the third form of reversibility and is called *out-of-causal-order reversibility*. This form gives us the opportunity to create new states that were not accessible by any forward-only execution path, in contrast with the other causally-respecting forms, which give us the ability to move forward and backward through previously visited states.

We can observe that assembly/disassembly systems compute in the out-of-causal-order form of reversibility. Imagine that we have a product that consists of many parts, and one of those parts is a battery. After a long period of use, the battery should either be charged or replaced. To get the battery removed from the product, we only have to remove the battery and the parts that prevent access to it, and not disassemble the entire product. The fact that we only remove certain pieces from the product without taking into account the order in which the product was assembled, means that we have performed an out-of-causal-order reversible form.

2.3 Petri nets

A Petri net - also known as a place/transition (PT) net - is one of several formal modelling languages [7]. It is a formalism that allows the modelling of systems which include concurrency, resource sharing, synchronization and conflict. The analysis of the qualitative properties of these modelling systems, permits the validation of the system's correctness.

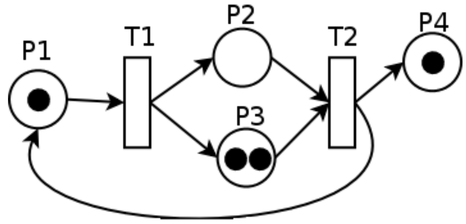


Figure 2.4: A Petri net composed of four places and two transitions.

A Petri net model is composed of a net structure that consists of nodes and arcs. The nodes represent transitions, which are events that may occur and are graphically represented by bars, and places, which are conditions of the model and are graphically represented by circles. The arcs running from a place to a transition, describe which places are pre-conditions for the corresponding transition and are called incoming arcs. The arcs running from a transition to a place describe which places are post-conditions for the corresponding transition and are called outgoing arcs. Arcs can never run between transitions or between places.

For example consider the Petri net in Figure 2.4. It is composed of four places which are the circles with names $P1$, $P2$, $P3$, $P4$, and two transitions which are the bars with names $T1$ and $T2$. Each arc that starts from a circle and ends to a bar is an incoming arc, and each arc that starts from a bar and ends to a circle is an outgoing arc. In this Petri net we have seven arcs from which the three are incoming arcs, and the rest are outgoing arcs.

The net structure is a weighted-bipartite directed graph specified as a 4-tuple:

$$N =< P, T, F, W >$$

where:

- P is a finite non-empty set of *places*.
- T is a finite non-empty set of *transitions*.
- F is a set of *directed arcs*, $F \subseteq (T \times P) \cup (P \times T)$.
- $W : F \to \mathbb{N}^+$ is a function assigning a weight to each arc.

The weight is graphically represented on the arc as a label and most of the times omitted if it is equal to one. The weight of the arc running from transition t to place p is expressed as $w(t, p)$, and $w(p,t)$ is the expression of the weight of the arc runs from place p to transition t.

Petri net graphs might contain tokens to some places. A token is one unity of a certain resource, and is graphically represented as a dot inside the place; it cannot exist somewhere else on the graph except from there.

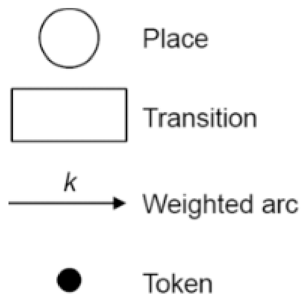


Figure 2.5: Graphical Petri net symbols

A marking of a Petri net graph is a mapping of its places P on $\mathbb{N}^+$. It represents an assignment of tokens to each place of the graph. The marking of a Petri net graph changes based on the marking evolution rule that describes how the states of the system are changing during computation.

The above rule determines whether a transition is enabled and may fire, and how the marking will change with the firing of a transition. A transition t is enabled when every input place p_i of t contains as marking at least as many tokens as specified by the weight $w(p_i,t)$. A transition t may fire, if and only if it is an enabled transition. The firing of transition t removes every $w(p_i,t)$ tokens from each input place p_i of t, and these tokens are added to each output place p_o of t.

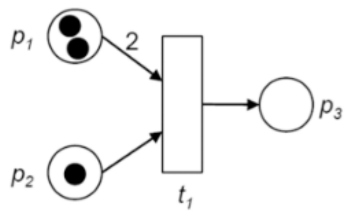


Figure 2.6: Graphical Petri net example where t_1 is an enabled transition

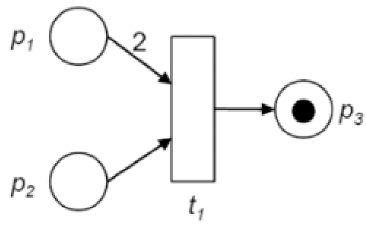


Figure 2.7: Graphical Petri net example after the firing of transition t_1 (of Figure 2.6)

The balance between the power of modelling and analyzability, is one of the things that make Petri nets an interesting conception. Petri nets can automatically determine a lot of useful information that someone might wish to know about concurrent systems. Several subclasses of the specific model have been studied which can easily model interesting classes of concurrent systems. Petri nets are characterized by some qualitative properties, which are boundedness, liveness and reversibility. The analysis of such properties associated with concurrent systems can successfully be supported by Petri nets, and that is a major strength they have [8].

2.4 Reversing Petri nets

Reversing Petri nets [6] is a reversible approach to Petri nets that introduces machinery and associated operational semantics to meet the challenges of the tree main forms of reversibility, which are backtracking, causal reversing and out-of-causal-order reversing. This model is actually an alternative of Petri nets where tokens are persistent and stand out from each other with the use of an identity. The methodology of this approach can be applied to a wide range of problems that feature reversibility.

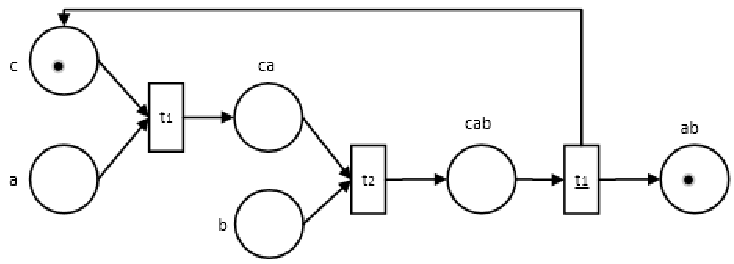


Figure 2.8: Standard Petri Net example

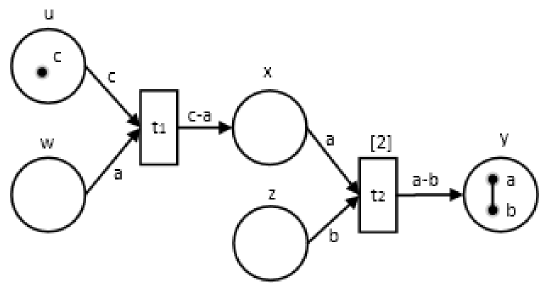


Figure 2.9: Reversible Petri Net example (same example with Figure 2.8)

The *Reversible Petri net* (RPN) structure is specified as a tuple:

$$N =< A, P, B, T, F >$$

where:

- A is a finite set of *bases* or *tokens* ranged over by a, b,.... $\overline{A} = \{\overline{a} \mid a \in A\}$ contains a "negative" instance for each token and we write $\mathscr{A} = A \cup \overline{A}$.
- P is a finite set of *places*.
- $B \subseteq A \times A$ is a set of *bonds* ranged over by β, γ,.... We use the notation $a-b$ for a bond $(a,b) \in B$. $\overline{B} = \{\overline{\beta} \mid \beta \in B\}$ contains a "negative" instance for each bond and we write $\mathscr{B} = B \cup \overline{B}$.
- T is a finite set of *transitions*.
- $F : (P \times T \cup T \times P) \to 2^{\mathscr{A} \cup \mathscr{B}}$ is a set of directed *arcs*.

Places and *transitions* are understood in a standard way as in Petri nets. Places are indicated by circles and transitions by bars where bases are indicated by • and bonds by lines between tokens. Arcs $l = F(p,t)$ or $l = F(t,p)$ contain each token at most once, either as a or $\overline{a}$ and if a bond $(a,b) \in l$ then $a,b \in l$ and for $l = F(t,p)$ then the following holds $l \cap (\overline{A} \cup \overline{B}) = \emptyset$. For $t \in T$ we introduce: $\circ t = \{p \in P \mid F(p,t) \neq \emptyset\}$,

$t\circ = \{p \in P \mid F(t,p) \neq \emptyset\}$ to be the sets of incoming and outgoing places of t respectively, and $\mathsf{pre}(t) = \bigcup_{p\in P} F(p,t)$ as well as $\mathsf{post}(t) = \bigcup_{p\in P} F(t,p)$ which are the unions of labels of the incoming/outgoing arcs of t.

A marking is a distribution of tokens and bonds across places, $M : P \to A \cup B$, where for $p \in P$: if $a{-}b \in M(p)$ then $a, b \in M(p)$. A *history* assigns a memory to each transition, $h : T \to \varepsilon \cup \mathbb{N}$ and is represented over the respective transition as $[k]$, where $k = H(t)$. If $k = \varepsilon$ it means that t has not been executed yet or it has been reversed, while if $k \in \mathbb{N}$ indicates that t was the k-th transition executed and has not been reversed. H_0 denotes the initial history where $H_0(t) = \varepsilon$ for every $t \in T$. A *state* is a pair $\langle M, H\rangle$ of a marking and history.

Every RPN is well-formed, acyclic and for all $a \in A$, $|\{p \mid a \in M_0(p)\}| = 1$.

A RPN is *well-formed*, when for all $t \in T$:

1. $A \cap \mathsf{pre}(t) = A \cap \mathsf{post}(t)$,
2. If $a{-}b \in \mathsf{pre}(t)$ then $a{-}b \in \mathsf{post}(t)$,
3. $F(t,p) \cap F(t,q) = \emptyset$ for all $p, q \in P, p \neq q$,

For $a \in A$ and $C \subseteq A \cup B$ the set of tokens and bonds connected with a according to the set C is denoted by $\mathsf{con}(a,C)$.

$$\mathsf{con}(a,C) = (\{a\} \cap C) \cup \{\beta, b, c \mid \exists w \text{ s.t. } \mathsf{path}(a,w,C), \beta \in w, \text{and } \beta = (b,c)\}$$

where $\mathsf{path}(a,w,C)$ if $w = \langle \beta_1, \ldots, \beta_n\rangle$, and for all $1 \leq i \leq n$, $\beta_i = (a_{i-1}, a_i) \in C \cap B$, $a_i \in C \cap A$, and $a_0 = a$. We also write $\mathsf{con}(S,C)$, where $S \subseteq A$, for $\bigcup_{a\in S} \mathsf{con}(a,C)$.

2.4.1 Forward execution

Definition 1 Consider a RPN (A,P,B,T,F), a transition $t \in T$, and a state $\langle M, H\rangle$. We say that t is *forward enabled* in $\langle M, H\rangle$ if:

1. if $a, \beta \in F(x,t)$ for some $x \in \circ t$, then $a, \beta \in M(x)$,
2. if $\overline{a}, \overline{\beta} \in F(x,t)$ for some $x \in \circ t$ then $a, \beta \notin M(x)$,
3. if $a \in F(t,y_1)$, $b \in F(t,y_2)$, $y_1 \neq y_2$ then $b \notin \mathsf{con}(a, M(x))$ for all $x \in \circ t$, and
4. if $\beta \in F(t,x)$ for some $x \in t\circ$ and $\beta \in M(y)$ for some $y \in \circ t$ then $\beta \in F(y,t)$.

A transition t is enabled if all tokens and bonds on incoming arcs are available according to marking M in $\mathsf{pre}(t)$, forks do not duplicate tokens and all repeated bonds appear on the incoming places.

Definition 2 Given a RPN (A,P,B,T,F), a state $\langle M,H\rangle$, and a transition t enabled in $\langle M,H\rangle$, we write $\langle M,H\rangle \xrightarrow{t} \langle M',H'\rangle$ where:

$$M'(x) \quad = \quad \begin{cases} M(x) - \bigcup_{a\in F(x,t)} \mathsf{con}(a,M(x)) & \text{if } x \in \circ t \\ M(x) \cup F(t,x) \cup \bigcup_{a\in F(t,x), y\in\circ t} \mathsf{con}(a,M(y)) & \text{if } x \in t\circ \\ M(x), & \text{otherwise} \end{cases}$$

and $H'(t') = \max\{k \mid k = H(t''), t'' \in T\} + 1$, if $t' = t$, and $H(t')$, otherwise.

After the execution of t all tokens and bonds occurring in $\mathsf{pre}(t)$ are transferred from the input to the output places of t. Moreover, the history function H is changed by assigning the the next available integer number to the transition.

2.4.2 Backtracking

A transition is backward enabled if it was the last executed transition.

Definition 3 Consider RPN $N = (A,P,B,T,F)$, a state $\langle M,H\rangle$ and a transition $t \in T$. We say that t is *bt-enabled* in $\langle M,H\rangle$ if $H(t) = k \in \mathbb{N}$ with $k \geq k'$ for all $k' \in \mathbb{N}$, $k' = H(t')$ and $t' \in T$.

The effect of backtracking a transition in a RPN is as follows:

Definition 4 Given a RPN (A,P,B,T,F), a state $\langle M,H\rangle$, and a transition t *bt*-enabled in $\langle M,H\rangle$, we write $\langle M,H\rangle \overset{t}{\rightsquigarrow}_b \langle M',H'\rangle$ where:

$$M'(x) \quad = \quad \begin{cases} M(x) \cup \bigcup_{y\in t\circ, a\in F(x,t)\cap F(t,y)} \mathsf{con}(a,M(y) - \mathsf{eff}(t)), & \text{if } x \in \circ t \\ M(x) - \bigcup_{a\in F(t,x)} \mathsf{con}(a,M(x)), & \text{if } x \in t\circ \\ M(x), & \text{otherwise} \end{cases}$$

and $H'(t') = \varepsilon$, if $t' = t$, and $H(t)$ otherwise.

After reversing t all tokens and bonds, as well as their connected components, are transferred from the outgoing places to the incoming places. Note that newly-created bonds are broken and the history function H of t is altered to ε.

2.4.3 Causal-Order reversibility

A transition is causally enabled if all its effects are reversed or not executed yet.

Definition 5 Consider RPN $N = (A,P,B,T,F)$, a state $\langle M,H\rangle$ and a transition $t \in T$. Then that t is *co-enabled* in $\langle M,H\rangle$ if $H(t) \in \mathbb{N}$, and, for all $a \in F(t,p)$, if $a \in M(q)$ for some q and $\mathsf{con}(a,M(q)) \cap \mathsf{pre}(t') \neq \emptyset$ for some $t' \in T$ then either $H(t') = \varepsilon$ or $H(t') \leq H(t)$.

Reversing a transition in a causally-respecting manner is implemented in exactly the same way as in backtracking.

2.4.4 Out-of-Causal-Order reversibility

In out-of-causal-order reversibility all executed transitions are enabled.

Definition 6 Consider RPN $N = (A,P,B,T,F)$, a state $\langle M,H \rangle$ and a transition $t \in T$. We say that t is *o-enabled* in $\langle M,H \rangle$, if $H(t) \in \mathbb{N}$.

The following notion defines the last transition that uses tokens of C which helps to define the effect of out-of-causal-order reversing.

Definition 7 Given RPN $N = (A,P,B,T,F)$, an initial marking M_0, a current marking M, a history H, and a set of bases and bonds C we write:

$$\mathsf{last}(C,H) \quad = \quad \begin{cases} t, & \text{if } \exists_t \ \mathsf{post}(t) \cap C \neq \emptyset,\ H(t) \in N \\ & \quad \nexists t', \mathsf{post}(t') \cap C \neq \emptyset, H(t') \in \mathbb{N}, H(t') > H(t) \\ \bot, & \text{otherwise} \end{cases}$$

Thus, $\mathsf{last}(C,H)$ is defined as follows: If the component C has been manipulated by some previously-executed transition, then $\mathsf{last}(C,H)$ is the last executed such transition. Otherwise, if no such transition exists (e.g. because all transitions involving C have been reversed), then $\mathsf{last}(C,H)$ is undefined ($\bot$). Transition reversal in an out-of-causal order can thus be defined as follows in Definition 8.

Definition 8 Given a RPN (A,P,B,T,F), an initial marking M_0, a state $\langle M,H \rangle$ and a transition t that is o-enabled in $\langle M,H \rangle$, we write $\langle M,H \rangle \stackrel{t}{\rightsquigarrow}_o \langle M',H' \rangle$ where H' is defined as in Definition 4 and we have:

$$\begin{aligned} M'(x) \quad = \quad & M(x) - \mathsf{eff}(t) - \ \{C_{a,x} \mid \exists a \in M(x), x \in t'\circ, t' \neq \mathsf{last}(C_{a,x},H')\} \\ & \cup \{C_{a,y} \mid \exists a,y,\ a \in M(y), \mathsf{last}(C_{a,y},H') = t', F(t',x) \cap C_{a,y} \neq \emptyset\} \\ & \cup \{C_{a,y} \mid \exists a,y,\ a \in M(y), \mathsf{last}(C_{a,y},H') = \bot, C_{a,y} \subseteq M_0(x)\} \end{aligned}$$

where we use the shorthand $C_{b,z} = \mathsf{con}(b, M(z) - \mathsf{eff}(t))$ for $b \in A$, $z \in P$.

When reversing t in out-of-causal order all bonds produced by t are broken. If the destruction of a bond divides a component into smaller connected components, then those components should be transferred to the outgoing places of their last transition as defined by 7, or to the places in their initial marking.

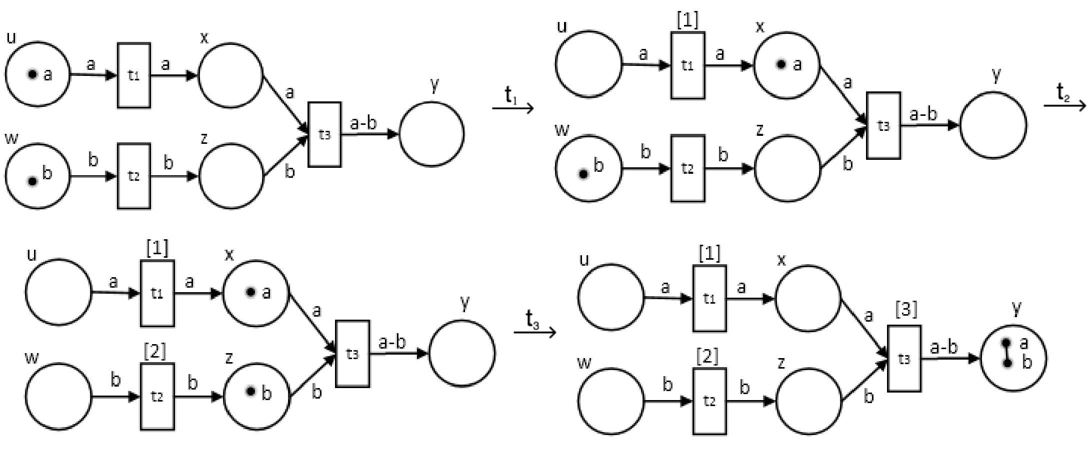


Figure 2.10: Forward Execution

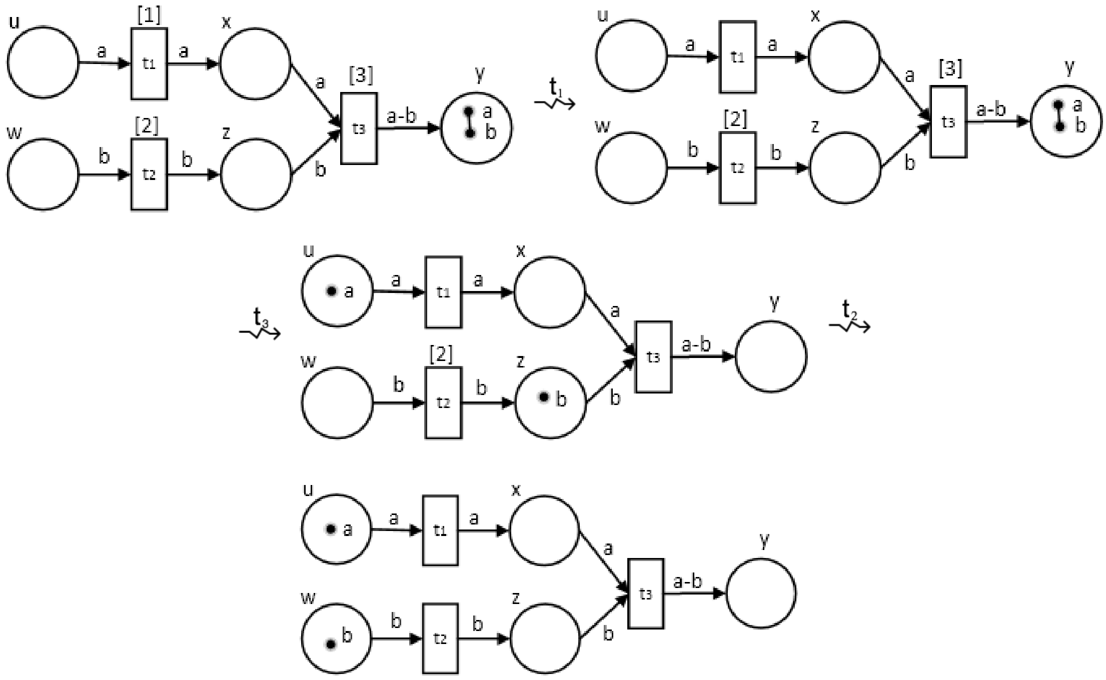


Figure 2.11: Reversing in Out-of-Causal-Order

An example of forward transitions can be seen in Figure 2.10 where transitions t_1, t_2 and t_3 take place with the histories of the two transitions becoming [1], [2] and [3], respectively. In Figure 2.11 we observe transitions t_1, t_3 and t_2 being reversed respectively, with the histories of the three transitions being eliminated. As we can see in second step of Figure 2.11, after the reversing of transition t_1 the marking of the RPN does not change since the transition t_3 it is the last transition using the token a, and it does not let it go. After the reversing of t_3, in third step, we can see that token a return back to its initial place since there are no other transitions that are using a. On the other hand, token b return in place z since t_2 is using b and has not been reversed yet.

Chapter 3

Matrix Semantics

Contents

Since Petri nets are presented as a model for discrete-event systems, it is helpful to have a system of equations which can be used to specify and manipulate the state of the system in a simulator. An alternative approach to the representation and analysis of Petri nets is based on matrix equations. In this approach matrix equations are used to represent the dynamic behaviour of Petri nets. We set out to study the matrix equations of RPNs by exploring the modelling of the main strategies for reversible computation, namely, of *backtracking*, *causal reversibility* and *out-of-causal-order reversibility*.

After the completion of this chapter we will be able to express the correlation between transitions and places of a RPN through matrices, as well as execute some transitions by using the equations between the matrices, which are defined below. The result of these operations will be the export of the new marking matrix M and the new history matrix H. These matrix equations will be used for the subsequent implementation of the simulator.

3.1 Forward execution

3.1.1 Matrices description

We now begin to describe how the matrix equations for the forward mode of computation are calculated. Starting with matrix FT which indicates the executing transition.

Definition 9 Transition matrix FT is a matrix of dimensions $1 \times \eta$, where $\eta = |T|$, which contains the number 1 in the position of the transition we are going to execute.

The following two definitions describe the matrices of the marking and history. Starting with matrix M which indicates the current marking of the RPN model.

Definition 10 Current marking matrix M is a matrix of dimensions $\eta \times \theta$, where $\eta = |T|$ and $\theta = |P|$, which contains the distribution of tokens and bonds across the places P. Each position of the matrix, represent a place, and includes a set of tokens and bonds.

We now define the matrix H which indicates the current history of the RPN model.

Definition 11 History matrix H is a matrix of dimensions $1 \times \eta$, where $\eta = |T|$, which contains the assignment of a memory to each transition. Each position in this matrix consists of a number from 0 to infinity (based on the times of the reversing).

We now proceed to describe the equations that operate on matrices consisting of multisets of bases and bonds. The fact that Reversing Petri nets use tokens and bonds in their directed arcs, and that each place in the network may contain a set of tokens and bonds, led us to the decision to use sets of tokens and bonds for each entry instead of quantities.

Such matrices that consist of sets of objects require redefining matrix operations. Firstly, we need to define the subtraction operation which will take one-by-one all the sets of tokens of each position in a matrix, and will remove all the elements that are in the corresponding position in the second matrix. Thus, we define the notion of subtraction operator as follows:

Definition 12 Given two matrices A and B of dimensions $n \times m$, we define the subtraction operator $C = A \ominus B$ such that:

$$C[i][j] = A[i][j] - B[i][j]$$

For the addition operation we need to define a second operator which will take one-by-one all the sets of tokens of each position in a matrix, and will add all the elements that are in the corresponding position in the second matrix. Thus, we define the notion of addition operator as follows:

Definition 13 Given two matrices A and B of dimensions $n \times m$, we define addition operator $C = A \oplus B$ such that:

$$C[i][j] = A[i][j] \cup B[i][j]$$

The last operator that we need to define is an operator for multiplication. It is based on the rationale of matrices multiplication as defined in the field of mathematics. The difference in our approach is that the first matrix of the multiplication operator is a matrix which contains 1s and 0s. When you multiply a set of tokens with 1 then the result it is equal with the set, and when you multiply a set of tokens with 0 then the result it is an empty set. Thus, we define the notion of multiplication operator as follows:

Definition 14 Given two matrices A and B of dimensions $n \times m$, we define multiplication operator $C = A \otimes B$ such that:

$$C[i][j] = \bigcup_{A[i][k]=1} B[k][j]$$

The following two definitions show the sets of tokens and bonds which are present on the directed arcs. If an arc goes from a place to a transition then it means that it is an incoming arc and its tokens will be added in the matrix D^{-}, as defined below:

Definition 15 Given an RPN $N = (A, P, B, T, F)$ we write D^{-} the matrix of the incoming arcs :

$$D^{-}[t][p] \quad = \quad F(p,t)$$

If an arc goes from a transition to a place then it means that it is an outgoing arc and its tokens will be added in matrix D^{+}, as defined below:

Definition 16 Given an RPN $N = (A, P, B, T, F)$ we write D^{+} the matrix of the outgoing arcs :

$$D^{+}[t][p] \quad = \quad F(t,p)$$

Since we have defined all the auxiliary matrices we are now ready to define how to execute a transition in forward direction. When executing t all tokens and bonds occurring in $\mathsf{pre}(t)$ are transferred from the input to the output places of t. Moreover, the history function H is changed by assigning the the next available integer number to the transition.

Definition 17 Given an RPN $N = (A, P, B, T, F)$, a transition matrix FT (with an enabled transition t as defined in Def. 1), a history matrix H, and a current marking matrix M, we write $\langle M, H\rangle \xrightarrow{t} \langle M', H'\rangle$ for M' and H':

$$M' = M \oplus CD^{+} \oplus TD^{+} \ominus CD^{-}$$
$$\text{and} \quad H' = H \oplus (max\{k | k = H(t), t \in T\} + 1) \times FT$$

where:
$TD^{+} = FT \otimes D^{+}$
$TD^{-} = FT \otimes D^{-}$
$CD^{+}[i] = \bigcup_{a \in TD^{+}[i], p \in P} \mathsf{con}(a, M(p))$
$CD^{-}[i] = \bigcup_{a \in TD^{-}[i], p \in P} \mathsf{con}(a, M(p))$

After the multiplication of matrices FT (Definition 9) and D^+ (Definition 16) we store the results in matrix TD^+. Matrix TD^+ contains the outgoing arcs of the executed transition, and has dimensions $1 \times p$, where $p = |P|$. Similarly, after the multiplication of matrices FT and D^- we store the results in matrix TD^-. Matrix TD^- contains the incoming arcs of the executed transition, and has dimensions $1 \times p$, where $p = |P|$.

The created matrices CD^- and CD^+ have dimensions $1 \times p$, where $p = |P|$. These matrices contain sets of tokens and bonds which are directly or indirectly connected with the given elements of each place of matrices TD^- and TD^+, respectively.

3.1.2 Execution example

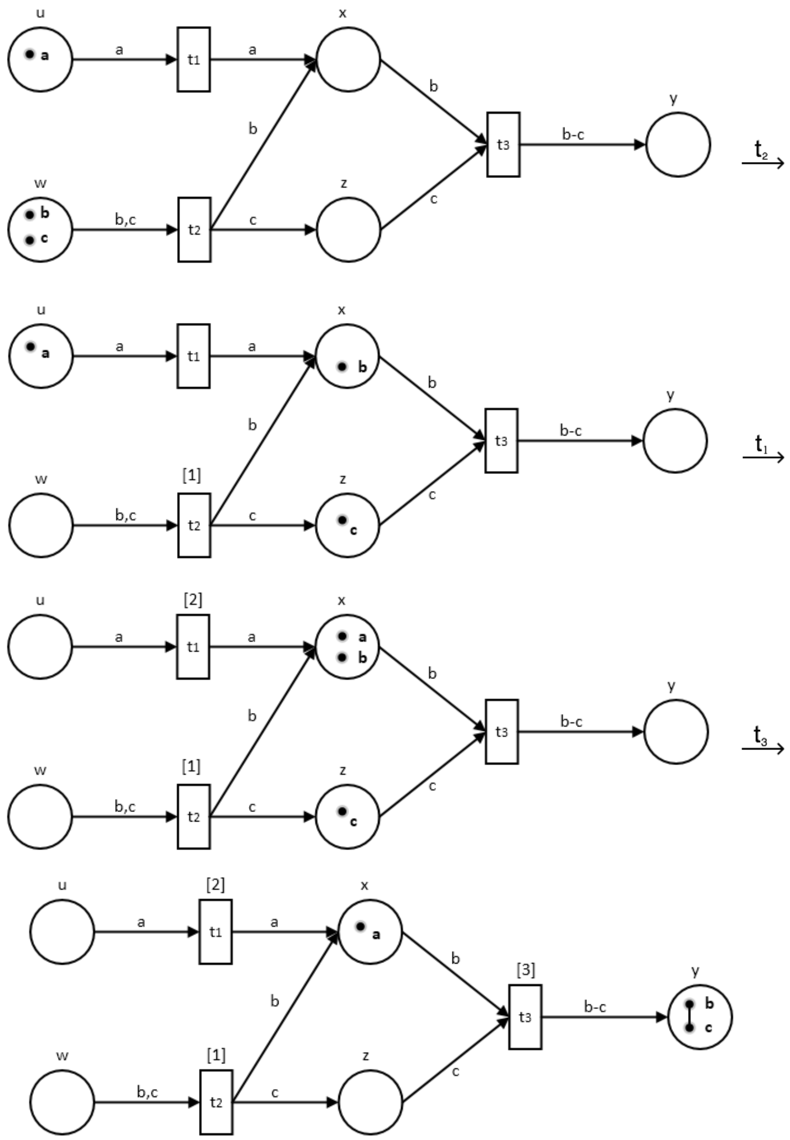


Figure 3.1: Forward execution

Example 1

An example of forward transitions can be seen in the steps of Figure 3.1 where transitions t_2, t_1 and t_3 take place with the histories of the three transitions becoming [1], [2] and [3], respectively. Note that to avoid overloading figures, we omit writing the bases of bonds on the arcs of an RPN and recall that within places we indicate bases by • and bonds by lines between relevant bases.

Any RPN can be represented as an incidence matrix. The reversing Petri net of the first scheme in Figure 3.1 can be specified in matrix form as follows:

1. Matrix D^- based on the incoming arcs of Fig. 3.1.

$$D^- = \begin{pmatrix} \{a\} & 0 & 0 & 0 & 0 \\ 0 & \{b,c\} & 0 & 0 & 0 \\ 0 & 0 & \{b\} & \{c\} & 0 \end{pmatrix}$$

2. Matrix D^+ based on the outgoing arcs of Fig. 3.1.

$$D^+ = \begin{pmatrix} 0 & 0 & \{a\} & 0 & 0 \\ 0 & 0 & \{b\} & \{c\} & 0 \\ 0 & 0 & 0 & 0 & \{b-c\} \end{pmatrix}$$

To represent the firing transition of the RPN we create a transition matrix FT. Assuming that transition t_2 is firing, we have:

$$FT = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$$

For the representation of the current marking of the RPN we create a marking matrix M. Assuming that at the beginning, matrix M is the same with matrix M_0, which contains the initial marking we have:

$$M = M_0 = \begin{pmatrix} \{a\} & \{b,c\} & 0 & 0 & 0 \end{pmatrix}$$

To determine the marking of the reversing Petri net after the firing of the transition specified in the transition matrix, we perform the following steps:

Step 1: Calculate matrix TD^+.

$$\begin{aligned} TD^+ = FT \otimes D^+ &= \begin{pmatrix} 0 & 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & \{a\} & 0 & 0 \\ 0 & 0 & \{b\} & \{c\} & 0 \\ 0 & 0 & 0 & 0 & \{b-c\} \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & \{b\} & \{c\} & 0 \end{pmatrix} \end{aligned}$$

Step 2: Calculate matrix TD^-.

$$\begin{aligned} TD^- = FT \otimes D^- &= \begin{pmatrix} 0 & 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} \{a\} & 0 & 0 & 0 & 0 \\ 0 & \{b,c\} & 0 & 0 & 0 \\ 0 & 0 & \{b\} & \{c\} & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & \{b,c\} & 0 & 0 & 0 \end{pmatrix} \end{aligned}$$

Step 3: Calculate matrix CD^+.

$$CD^+ = \begin{pmatrix} 0 & 0 & \{b\} & \{c\} & 0 \end{pmatrix}$$

Step 4: Calculate matrix CD^{-}.

$$CD^{-} = \begin{pmatrix} 0 & \{b,c\} & 0 & 0 & 0 \end{pmatrix}$$

Step 5: Calculate the new marking matrix M'.

$$\begin{aligned} M' &= M \oplus CD^{+} \oplus TD^{+} \ominus CD^{-} \\ &= \begin{pmatrix} \{a\} & \{b,c\} & 0 & 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} 0 & 0 & \{b\} & \{c\} & 0 \end{pmatrix} \\ &\quad \oplus \begin{pmatrix} 0 & 0 & \{b\} & \{c\} & 0 \end{pmatrix} \ominus \begin{pmatrix} 0 & \{b,c\} & 0 & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} \{a\} & 0 & \{b\} & \{c\} & 0 \end{pmatrix} \end{aligned}$$

Step 6: Calculate the new history matrix H'. (Initial history matrix H contains only 0s since no transition yet fire)

$$\begin{aligned} H' &= H \oplus (max\{k|k = H(t), t \in T\} + 1) \times FT \\ &= \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} \oplus 1 \times \begin{pmatrix} 0 & 1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 1 & 0 \end{pmatrix} \end{aligned}$$

The other two transitions, t_1 and t_3, of the example in Figure 3.1 are firing in the same way as transition t_2, starting from the state of the RPN as shown in the second scheme of figure.

3.2 Backtracking

3.2.1 Matrices description

Matrix E is a matrix of dimensions 1 $\times$ places, which contains the current marking tokens of each place, without the effect of the transition we are reversing.

Definition 18 Given a current marking matrix M (with dimensions $1 \times |P|$), and the transition t we are reversing, we write:

$$E[i][j] = M[i][j] - \mathsf{eff}(t)$$

where:
$\mathsf{eff}(t) = \mathsf{post}(t) - \mathsf{pre}(t)$.

After reversing t all tokens and bonds, as well as their connected components, are transferred from the outgoing places to the incoming places. Note that newly-created bonds are broken and the history function H of t is altered to ε.

Definition 19 Given an RPN $N = (A, P, B, T, F)$, a transition matrix FT (with a *bt*-enabled transition t as defined in Def. 3), a history matrix H, and a current marking matrix M, we write $\langle M, H \rangle \overset{t}{\rightsquigarrow}_b \langle M', H' \rangle$ for M' and H':

$$\begin{aligned} &M' = M \oplus CD^{-} \ominus CD^{+} \\ \text{and} \quad &H' = H \ominus (\{k \mid k = H(t), t \in T, FT(t) = 1\} \times FT) \end{aligned}$$

where:
$TD^{+} = FT \otimes D^{+}$

$TD^- = FT \otimes D^-$
$CD^+[i] = \bigcup_{a \in TD^+[i], p \in P} \mathsf{con}(a, M(p))$
$CD^-[i] = \bigcup_{a \in TD^-[i], p \in P} \mathsf{con}(a, E(p))$

After the multiplication of matrices FT and D^+ we store the results in matrix TD^+. Matrix TD^+ contains the outgoing arcs of the executed transition, and has dimensions $1 \times p$, where $p = |P|$. Similarly, after the multiplication of matrices FT and D^- we store the results in matrix TD^-. Matrix TD^- contains the incoming arcs of the executed transition, and has dimensions $1 \times p$, where $p = |P|$.

The created matrices CD^- and CD^+ have dimensions $1 \times p$, where $p = |P|$. These matrices contain sets of tokens and bonds which are directly or indirectly connected with the given elements of each place of matrices TD^- and TD^+, respectively.

3.2.2 Execution example

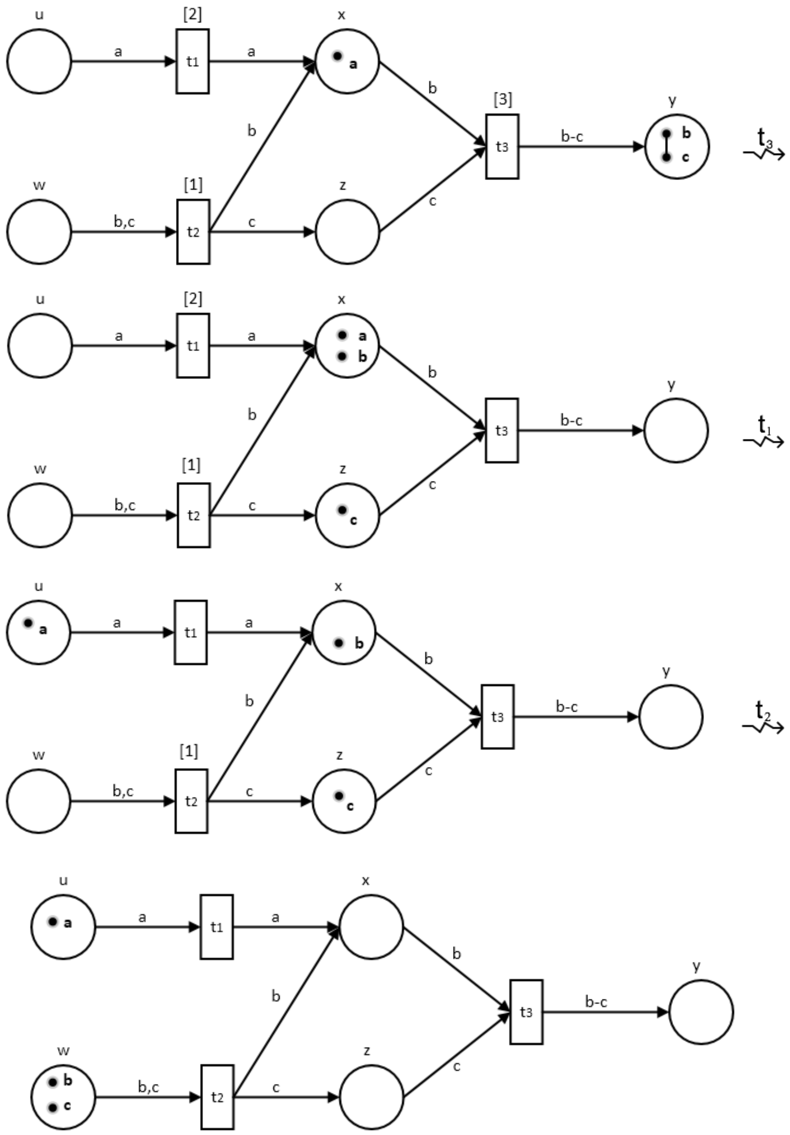


Figure 3.2: Backtracking execution

Example 2 After the forward execution of Figure 3.1, where transitions t_1, t_2 and t_3 were executed in the order t_2, t_1, t_3, in the example of Figure 3.2 we observe the same transitions being backtracked with their histories being eliminated.

To represent the transition of the RPN that executed in reverse we create a transition matrix FT where, assuming we are in the state of the first scheme of Figure 3.2 and that transition t_3 is reversing, we have:

$$FT = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$$

The matrix E represent the current marking of the RPN, without the effect of the transition we are reversing. In this case matrix E has the following values:

$$E = \begin{pmatrix} 0 & 0 & \{a\} & 0 & \{b,c\} \end{pmatrix}$$

The matrix M which represent the current marking of the RPN and the history matrix H, after the forward execution of transitions t_2, t_1 and t_3 respectively has take the following values.

$$M = \begin{pmatrix} 0 & 0 & \{a\} & 0 & \{b-c\} \end{pmatrix}$$
$$H = \begin{pmatrix} 2 & 1 & 3 \end{pmatrix}$$

To determine the marking of the reversing Petri net after the reverse execution of the transition specified in the transition matrix, we perform the following steps:

Step 1: Calculate matrix TD^+.

$$\begin{aligned} TD^+ = FT \otimes D^+ &= \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & \{a\} & 0 & 0 \\ 0 & 0 & \{b\} & \{c\} & 0 \\ 0 & 0 & 0 & 0 & \{b-c\} \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & 0 & 0 & \{b-c\} \end{pmatrix} \end{aligned}$$

Step 2: Calculate matrix TD^-.

$$\begin{aligned} TD^- = FT \otimes D^- &= \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} \{a\} & 0 & 0 & 0 & 0 \\ 0 & \{b,c\} & 0 & 0 & 0 \\ 0 & 0 & \{b\} & \{c\} & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & \{b\} & \{c\} & 0 \end{pmatrix} \end{aligned}$$

Step 3: Calculate matrix CD^+.

$$CD^+ = \begin{pmatrix} 0 & 0 & 0 & 0 & \{b-c\} \end{pmatrix}$$

Step 4: Calculate matrix CD^- (using con function on matrix E).

$$CD^- = \begin{pmatrix} 0 & 0 & \{b\} & \{c\} & 0 \end{pmatrix}$$

Step 5: Calculate the new marking matrix M'.

$$\begin{aligned} M' &= M \oplus CD^- \ominus CD^+ \\ &= \begin{pmatrix} 0 & 0 & \{a\} & 0 & \{b-c\} \end{pmatrix} \oplus \begin{pmatrix} 0 & 0 & \{b\} & \{c\} & 0 \end{pmatrix} \\ &\quad \ominus \begin{pmatrix} 0 & 0 & 0 & 0 & \{b-c\} \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & \{a,b\} & \{c\} & 0 \end{pmatrix} \end{aligned}$$

Step 6: Calculate the new history matrix H'.

$$\begin{aligned} H' &= H \ominus (\{k \mid k = H(t), t \in T, FT(t) = 1\} \times FT) \\ &= \begin{pmatrix} 2 & 1 & 3 \end{pmatrix} \ominus 3 \times \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 2 & 1 & 0 \end{pmatrix} \end{aligned}$$

The other two transitions, t_1 and t_2, of the example in Figure 3.2 are reversing respectively following the same steps as t_3 did and proceed from the first to the second RPN scheme of Figure 3.2.

3.3 Causal-Order reversibility

3.3.1 Matrices description

Reversing a transition in a causally-respecting manner is implemented in exactly the same way as in backtracking.

Definition 20 Given a RPN (A,P,B,T,F), and a *co*-enabled transition t (as defined in Def. 5) in $\langle M,H\rangle$ we write $\langle M,H\rangle \overset{t}{\rightsquigarrow}_c \langle M',H'\rangle$ for M' and H' as in Definition 19.

3.3.2 Execution example

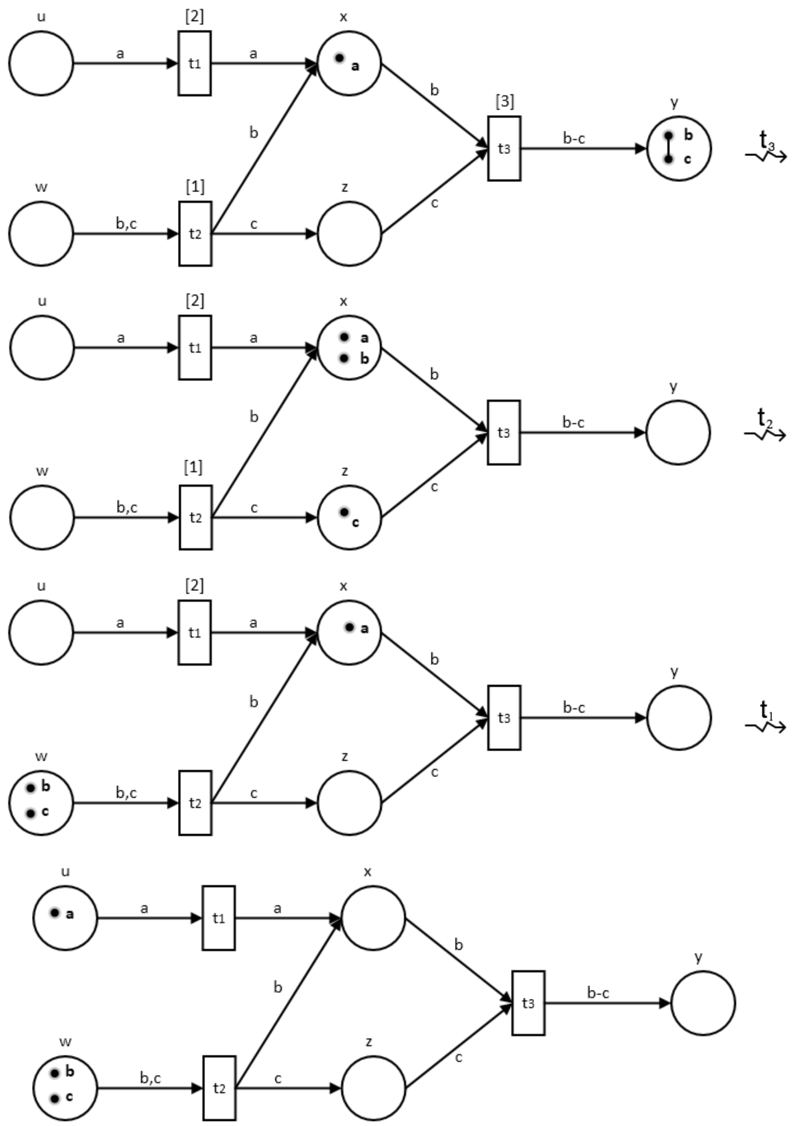


Figure 3.3: Causal Order execution

Example 3 An example of causal-order reversibility can be seen in Figure 3.3. Here we have two independent executions, t_1 and t_2, or t_1 and t_3. Assuming that transitions were executed in the order t_2, t_1, t_3 (as shown in Figure 3.1), the example demonstrates a causally-ordered reversal where t_3 is reversed, followed by the reversal of t_2 and t_1. These can be reversed in any order although in the example t_2 is reversed before t_1.

To represent the transition of the RPN that executed in reverse we create a transition matrix FT where, assuming that transition t_3 has already reverse and now transition t_2 is reversing, we have:

$$FT = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$$

In this case where transition t_2 is reversing, matrix E has the same values with the current marking matrix M, since the effect of t_2 is the empty set.

So marking matrix M, E and the history matrix H, after the forward execution of transitions t_2, t_1 and t_3 respectively and the reverse execution of transition t_3, has take the following values.

$$M = E = \begin{pmatrix} 0 & 0 & \{a,b\} & \{c\} & 0 \end{pmatrix}$$
$$H = \begin{pmatrix} 2 & 1 & 0 \end{pmatrix}$$

To determine the marking of the reversing Petri net after the reverse execution of the transition specified in the transition matrix, we perform the following steps:

Step 1: Calculate matrix TD^+.

$$\begin{aligned} TD^+ = FT \otimes D^+ &= \begin{pmatrix} 0 & 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & \{a\} & 0 & 0 \\ 0 & 0 & \{b\} & \{c\} & 0 \\ 0 & 0 & 0 & 0 & \{b-c\} \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & \{b\} & \{c\} & 0 \end{pmatrix} \end{aligned}$$

Step 2: Calculate matrix TD^-.

$$\begin{aligned} TD^- = FT \otimes D^- &= \begin{pmatrix} 0 & 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} \{a\} & 0 & 0 & 0 & 0 \\ 0 & \{b,c\} & 0 & 0 & 0 \\ 0 & 0 & \{b\} & \{c\} & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & \{b,c\} & 0 & 0 & 0 \end{pmatrix} \end{aligned}$$

Step 3: Calculate matrix CD^+.

$$CD^+ = \begin{pmatrix} 0 & 0 & \{b\} & \{c\} & 0 \end{pmatrix}$$

Step 4: Calculate matrix CD^- (using con function on matrix E).

$$CD^- = \begin{pmatrix} 0 & \{b,c\} & 0 & 0 & 0 \end{pmatrix}$$

Step 5: Calculate the new marking matrix M'.

$$\begin{aligned} M' &= M \oplus CD^- \ominus CD^+ \\ &= \begin{pmatrix} 0 & 0 & \{a,b\} & \{c\} & 0 \end{pmatrix} \oplus \begin{pmatrix} 0 & \{b,c\} & 0 & 0 & 0 \end{pmatrix} \\ &\quad \ominus \begin{pmatrix} 0 & 0 & \{b\} & \{c\} & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & \{b,c\} & \{a\} & 0 & 0 \end{pmatrix} \end{aligned}$$

Step 6: Calculate the new history matrix H'.

$$\begin{array}{rcl} H' & = & H \ominus (\{k \mid k = H(t), t \in T, FT(t) = 1\} \times FT) \\ & = & \begin{pmatrix} 2 & 1 & 0 \end{pmatrix} \ominus 3 \times \begin{pmatrix} 0 & 1 & 0 \end{pmatrix} \\ & = & \begin{pmatrix} 2 & 0 & 0 \end{pmatrix} \end{array}$$

The transition t_1 of the example in Figure 3.3 are causally reversing following the same steps that t_2 execute above.

3.4 Out-of-Causal-Order reversibility

3.4.1 Matrices description

When reversing t in out-of-causal order all bonds produced by t are broken. If the destruction of a bond divides a component into smaller connected components, then those components should be transferred to the outgoing places of their last transition as defined by 7, or to the places in their initial marking.

Definition 21 Given an RPN $N = (A, P, B, T, F)$, a transition matrix FT (with a o-enabled transition t as defined in Def. 6), a history matrix H, and a current marking matrix M, after computing matrix E (as defined in Def. 18), we write $\langle M, H \rangle \overset{t}{\rightsquigarrow}_o \langle M', H' \rangle$ for M' and H':

$$M' = E \ominus CL^- \oplus CL^+$$
$$\text{and} \quad H' = H \ominus (\{k \mid k = H(t), t \in T, FT(t) = 1\} \times FT)$$

where:
$CL^+[i] = \bigcup_{a \in L^+[i], p \in P} \mathsf{con}(a, E(p))$
$CL^-[i] = \bigcup_{a \in L^-[i], p \in P} \mathsf{con}(a, E(p))$

The created matrices CL^- and CL^+ have dimensions $1 \times p$, where $p = |P|$. These matrices contain sets of tokens and bonds which are directly or indirectly connected with the given elements of each place of matrices L^- and L^+, respectively.

The following two definitions show the sets of tokens which must be added or removed from the current marking matrix. For the creation of this matrices we have to use the last function as this defined in Definition 7. Matrix L^+ contains the sets of tokens that must be added to the current marking matrix in specific positions. Let us define the matrix L^+ contents as follows:

Definition 22 Given an RPN $N = (A, P, B, T, F)$ a history matrix H, initial marking matrix M_0 and a current marking matrix M we write:

$$L^+[x] \quad = \quad L^+[x] \cup \begin{cases} \{a\} & \text{if } \exists a, y,\ a \in M(y), \mathsf{last}(C_{a,y}, H') = t', F(t', x) \cap C_{a,y} \neq \emptyset \\ \{a\} & \text{if } \exists a, y,\ a \in M(y), \mathsf{last}(C_{a,y}, H') = \perp, C_{a,y} \subseteq M_0(x) \\ \emptyset, & \text{otherwise} \end{cases}$$

Matrix L^- contains the sets of tokens that must be removed from the current marking matrix from specific positions. Let us define the matrix L^- contents as follows:

Definition 23 Given an RPN $N = (A, P, B, T, F)$ a history matrix H and a current marking matrix M we write:

$$L^{-}[x] \quad = \quad L^{-}[x] \cup \begin{cases} \{a\} & \text{if } \exists a \in M(x), x \in t'\circ, t' \neq \mathsf{last}(C_{a,x}, H') \\ \emptyset, & \text{otherwise} \end{cases}$$

3.4.2 Execution example

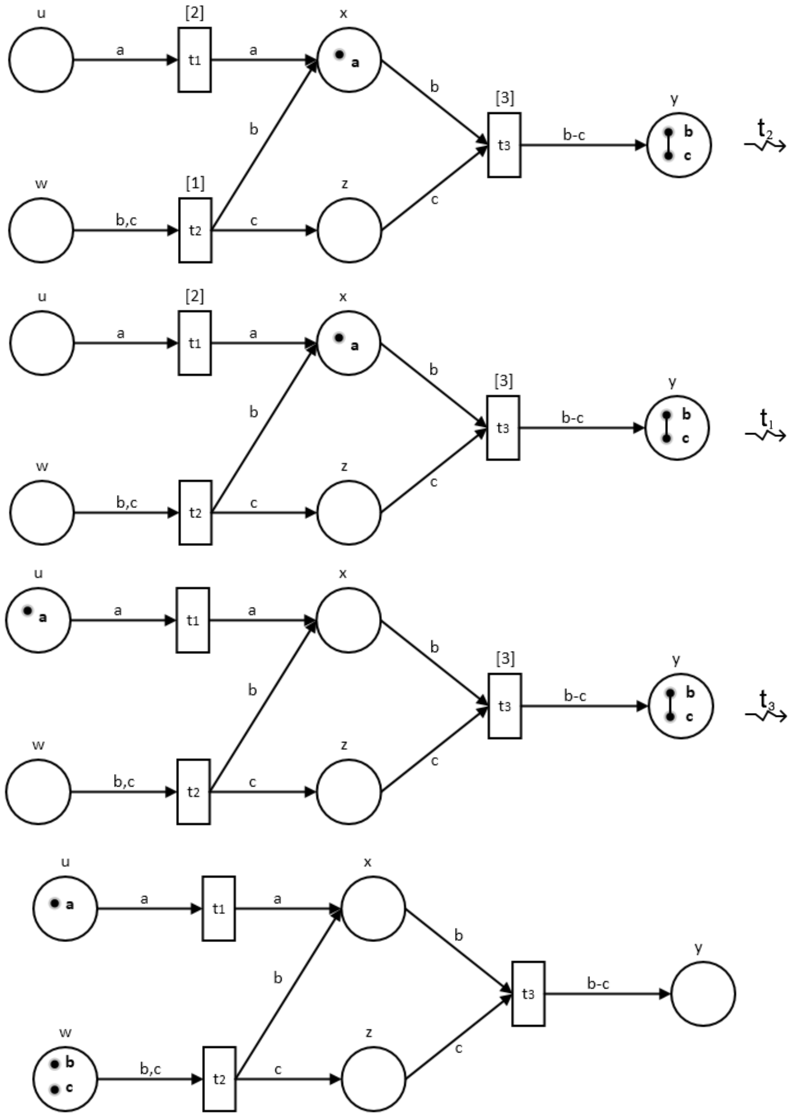


Figure 3.4: Out-Of-Causal-Order execution

Example 4 An example of out-of-causal-order reversal can be seen in Figure 3.4. In the net of Figure 3.1, we see that t_2, t_1 and t_3 have been executed in this order and now b,c tokens are in place y, and a token is in place x. The example demonstrates an out-of-causal-order reversal where all the transitions are reversed in the exact order they fired.

To represent the transition of the RPN that is reversed out of order we create a transition matrix FT where, assuming that transition t_2 is reversing, we have:

$$FT = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$$

When transition t_2 is reversing, matrix E has the same values with the current marking matrix M, since the effect of t_2 is the empty set.

So, marking matrices M, E and the history matrix H, after the forward execution of transitions t_2, t_1 and t_3 respectively, have the following values.

$$M = E = \begin{pmatrix} 0 & 0 & \{a\} & 0 & \{b-c\} \end{pmatrix}$$
$$H = \begin{pmatrix} 2 & 1 & 3 \end{pmatrix}$$

To determine the marking of the reversing Petri net after the reverse execution of the transition specified in the transition matrix, we perform the following steps:

Step 1: Calculate matrix L^+.

$$L^+ = \begin{pmatrix} 0 & 0 & \{a\} & 0 & \{b,c\} \end{pmatrix}$$

Step 2: Calculate matrix L^-.

$$L^- = \begin{pmatrix} 0 & 0 & \{a\} & 0 & 0 \end{pmatrix}$$

Step 3: Calculate matrix CL^+.

$$CL^+ = \begin{pmatrix} 0 & 0 & \{a\} & 0 & \{b-c\} \end{pmatrix}$$

Step 4: Calculate matrix CL^-.

$$CL^- = \begin{pmatrix} 0 & 0 & \{a\} & 0 & 0 \end{pmatrix}$$

Step 5: Calculate the new marking matrix M'.

$$\begin{aligned} M' &= E \ominus CL^- \oplus CL^+ \\ &= \begin{pmatrix} 0 & 0 & \{a\} & 0 & \{b-c\} \end{pmatrix} \ominus \begin{pmatrix} 0 & 0 & \{a\} & 0 & 0 \end{pmatrix} \\ &\quad \oplus \begin{pmatrix} 0 & 0 & \{a\} & 0 & \{b-c\} \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & \{a\} & 0 & \{b-c\} \end{pmatrix} \end{aligned}$$

Step 6: Calculate the new history matrix H'.

$$\begin{aligned} H' &= H \ominus (\{k \mid k = H(t), t \in T, FT(t) = 1\} \times FT) \\ &= \begin{pmatrix} 2 & 1 & 3 \end{pmatrix} \ominus 1 \times \begin{pmatrix} 0 & 1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 2 & 0 & 3 \end{pmatrix} \end{aligned}$$

Even when transition t_2 reverses, the current marking of the RPN did not change because the transition t_3, which is using tokens b and c has not reverse yet.

Now let us assume that transition t_1 is reversing next, we have:

$$FT = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$$

When transition t_1 is reversing, matrix E has the same values with the current marking matrix M, since the effect of t_1 is the empty set.

To determine the marking of the reversing Petri net after the reverse execution of the transition specified in the transition matrix, we repeat again the above steps.

Step 1: Calculate matrix L^+.

$$L^+ = \begin{pmatrix} \{a\} & 0 & 0 & 0 & \{b,c\} \end{pmatrix}$$

Step 2: Calculate matrix L^-.

$$L^- = \begin{pmatrix} 0 & 0 & \{a\} & 0 & 0 \end{pmatrix}$$

Step 3: Calculate matrix CL^+.

$$CL^+ = \begin{pmatrix} \{a\} & 0 & 0 & 0 & \{b-c\} \end{pmatrix}$$

Step 4: Calculate matrix CL^-.

$$CL^- = \begin{pmatrix} 0 & 0 & \{a\} & 0 & 0 \end{pmatrix}$$

Step 5: Calculate the new marking matrix M'.

$$\begin{aligned} M' &= E \ominus CL^- \oplus CL^+ \\ &= \begin{pmatrix} 0 & 0 & \{a\} & 0 & \{b-c\} \end{pmatrix} \ominus \begin{pmatrix} 0 & 0 & \{a\} & 0 & 0 \end{pmatrix} \\ &\quad \oplus \begin{pmatrix} \{a\} & 0 & 0 & 0 & \{b-c\} \end{pmatrix} \\ &= \begin{pmatrix} \{a\} & 0 & 0 & 0 & \{b-c\} \end{pmatrix} \end{aligned}$$

Step 6: Calculate the new history matrix H'.

$$\begin{aligned} H' &= H \ominus (\{k \mid k = H(t), t \in T, FT(t) = 1\} \times FT) \\ &= \begin{pmatrix} 2 & 0 & 3 \end{pmatrix} \ominus 2 \times \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & 3 \end{pmatrix} \end{aligned}$$

After t_2 and t_1 have reversed we assume that transition t_3 is reversing next, so we have:

$$FT = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$$

In this case, where transition t_3 is reversing (and since the effect of t_3 is the bond $(b-c)$), matrix E has the following values :

$$E = \begin{pmatrix} \{a\} & 0 & 0 & 0 & \{b,c\} \end{pmatrix}$$

To determine the marking of the reversing Petri net after the reverse execution of the transition specified in the transition matrix, we do:

Step 1: Calculate matrix L^+.

$$L^+ = \begin{pmatrix} \{a\} & \{b,c\} & 0 & 0 & 0 \end{pmatrix}$$

Step 2: Calculate matrix L^-.

$$L^- = \begin{pmatrix} 0 & 0 & 0 & 0 & \{b,c\} \end{pmatrix}$$

Step 3: Calculate matrix CL^+ (using con function on matrix E).

$$CL^+ = \begin{pmatrix} \{a\} & \{b,c\} & 0 & 0 & 0 \end{pmatrix}$$

Step 4: Calculate matrix CL^-(using con function on matrix E).

$$CL^- = \begin{pmatrix} 0 & 0 & 0 & 0 & \{b,c\} \end{pmatrix}$$

Step 5: Calculate the new marking matrix M'.

$$\begin{aligned} M' &= E \ominus CL^- \oplus CL^+ \\ &= \begin{pmatrix} \{a\} & 0 & 0 & 0 & \{b,c\} \end{pmatrix} \ominus \begin{pmatrix} 0 & 0 & 0 & 0 & \{b,c\} \end{pmatrix} \\ &\quad \oplus \begin{pmatrix} \{a\} & \{b,c\} & 0 & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} \{a\} & \{b,c\} & 0 & 0 & 0 \end{pmatrix} \end{aligned}$$

Step 6: Calculate the new history matrix H'.

$$\begin{aligned} H' &= H \ominus (\{k \mid k = H(t), t \in T, FT(t) = 1\} \times FT) \\ &= \begin{pmatrix} 0 & 0 & 3 \end{pmatrix} \ominus 3 \times \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} \end{aligned}$$

Now we can see that after the reverse execution of t_3 the tokens b and c have return in their initial marking since the transition t_2 has already been reversed before.

Chapter 4

Simulator

Contents

4.1 Requirements Specification

The key of any well organised and coherent project is to simultaneously keep track of the requirements and project development, since normally requirements will drive the software design. Requirements are the objectives that must necessarily be met, they define the functions that the project is ultimately supposed to provide. The requirements specification document contains the instructions which describe the behavioural characteristics of the system that is going to be developed.

4.1.1 Aims

The theoretical semantics of Reversing Petri nets will find practical application in a user-friendly software, simulating some algorithms that will be implemented based on the matrix equations, defined in Chapter 3 which execute transitions in forward and reverse computational order. The software product will give to users the opportunity to give a Reversing Petri net's information, express this information in the form of matrices, and after the creation of the required matrices, users can execute a specific transition and see how the new marking of the net has changed.

4.1.2 Objectives

In order to apply the principles of Reversing Petri net in practice several algorithms are going to be constructed for the representation of the RPN information. The project aims:

- To outline the theoretical aspects of bi-directional computing after extensive research. The scientific principles on which the software is based are demonstrated in previous chapters.
- The information of Reversing Petri net will be then translated into matrices, and from that form into the corresponding algorithms which are using matrix equations. A formal description of the logic was created in a previous chapter to define how the RPN semantics can be expressed in the form of matrices.
- To prepare the stages of the software development by identifying the technical and engineering background of the product.
- To construct a Graphical User Interface (G.U.I.) that allows users to interact with the system, to understand Reversing Petri nets and observe the forward and reverse execution of a chosen transition.
- To develop software product that demonstrates the changes of RPN marking after the execution of transitions by showing how a chosen transition can change marking by executing in forward or reverse order.

4.1.3 Specifications

The project objectives are going to be met, by starting with proposing ideas to express Reversing Petri net information in the simulator environment in the form of an algorithm. The key consideration of those ideas will be correctness at first, independent of the amount of memory needed, and then efficiency.

Consequently, the project will start with finding the best structure to use for the representation of the matrices information in the simulator environment. We have to choose a structure that allows us to easily change the tokens within a set, since our system will have to add or remove tokens from marking pretty often.

4.2 Implementation Programming Language

The developed software system simulates transition execution of RPNs both in forward and reverse direction implemented in Java. Java is an object oriented language, that provides programmers the opportunity to develop user friendly interfaces which are easy to understand and once the code runs on one platform it does not need to be recompiled on another. In addition by using the programming features that Java offers, memory space is no longer a programming concern since they provide an easy way to clean up temporary results and variables.

4.3 Simulator Manual

When the program starts the user can choose whether it should read the RPN information from a file or from the user (as shown in Fig. 4.1). The information given is the names of tokens A, places P, transitions T, directed arcs F - from a place to a transition or from a transition to a place, with a set of tokens and/or bonds -, and initial marking M_0.

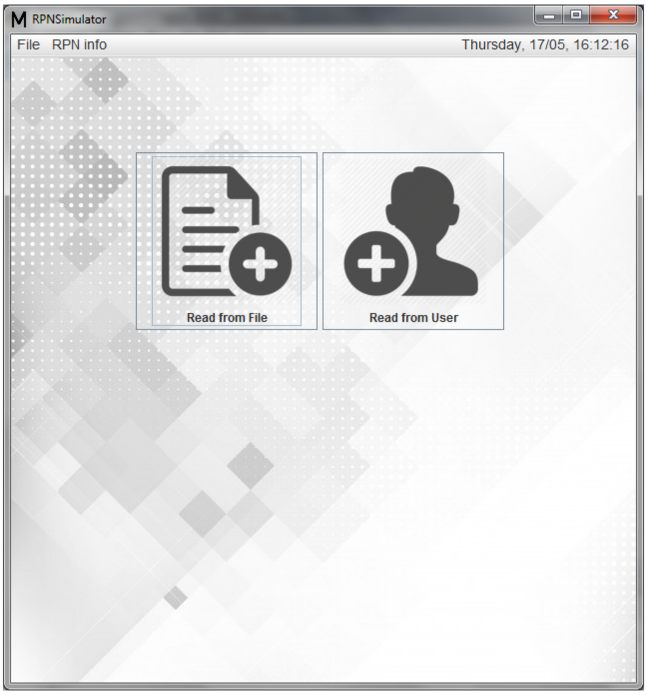


Figure 4.1: The appearance of the interface when you start the simulator

If the user presses the “Read from File” button then the appearance of the interface will change and the user will be asked to insert the name of the file that contains the information of the model. If the name of the file is correct then by pressing the button “Read..”, a window will appear on the screen with the information extracted from the file.

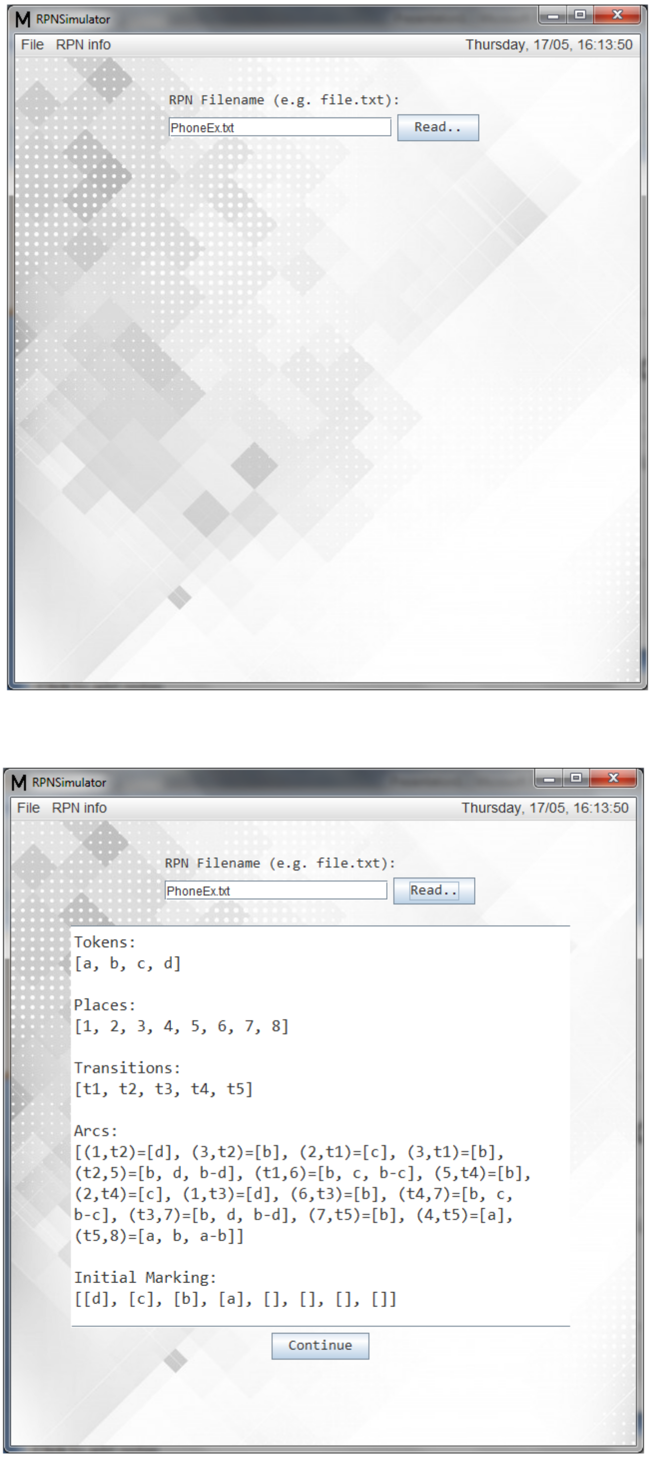


Figure 4.2: After the successful reading of the RPN information from the file PhoneEx.txt, the information is being displayed on the screen

The imported file containing the information of the RPN model should have the same form as the file in Fig. 4.3. The first line in the file should contain the tokens of the model separated by comma. The next line should contain the places of the model, followed by the transitions of the RPN in the next line, both of them being separated by comma. All the directed arcs should follow in the fourth line in the form $F(a,b) = \{c\}$, where a and b can be a transition or a place of the model, and c it is a set of tokens and/or bonds (separated by comma). If a has the value of a transition that means that b should have the value of a place, and vice versa. The initial marking M_0 of the model should follow by adding the tokens or/and bonds of each place, line by line (in the same order that places P have been inserted in the second line). If a place does not contain any tokens in the initial state, then the user should add to the corresponding line the value 0.

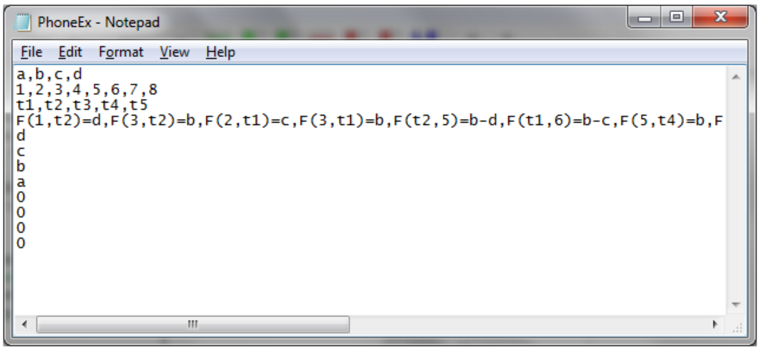
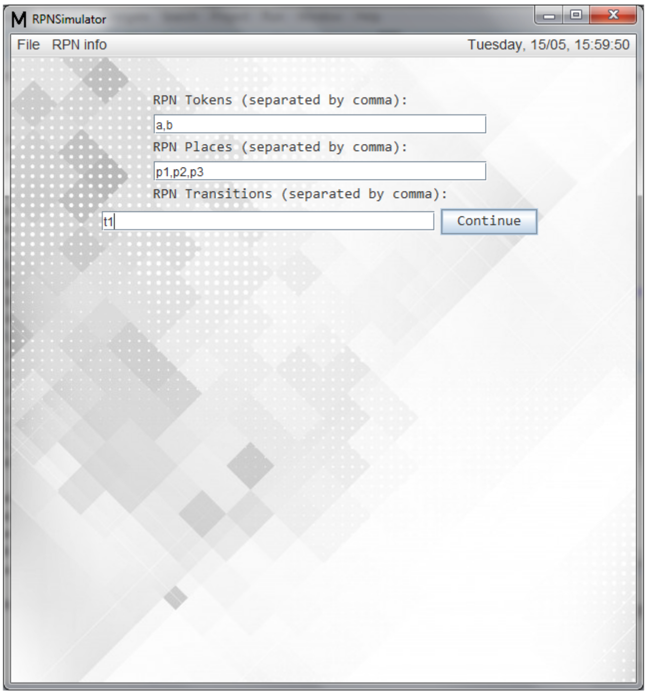


Figure 4.3: The form of the file that the user should import if he/she choose the "Read from File" mode

If the user presses the "Read from User" button then the appearance of the interface will change and the user will be asked to insert the tokens A, the places P and the transitions T of the model. After that, when the user chooses to continue, the system will ask from him/her to add the directed arcs and the initial marking of each place of the RPN model.



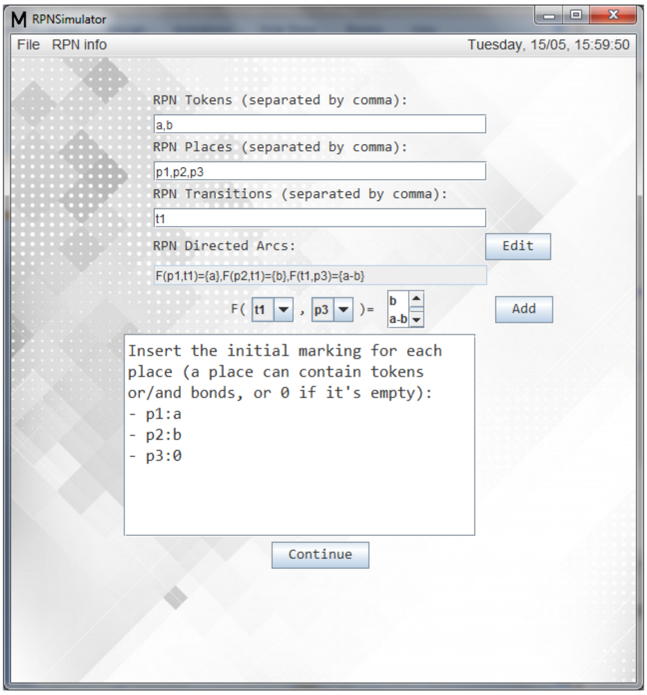
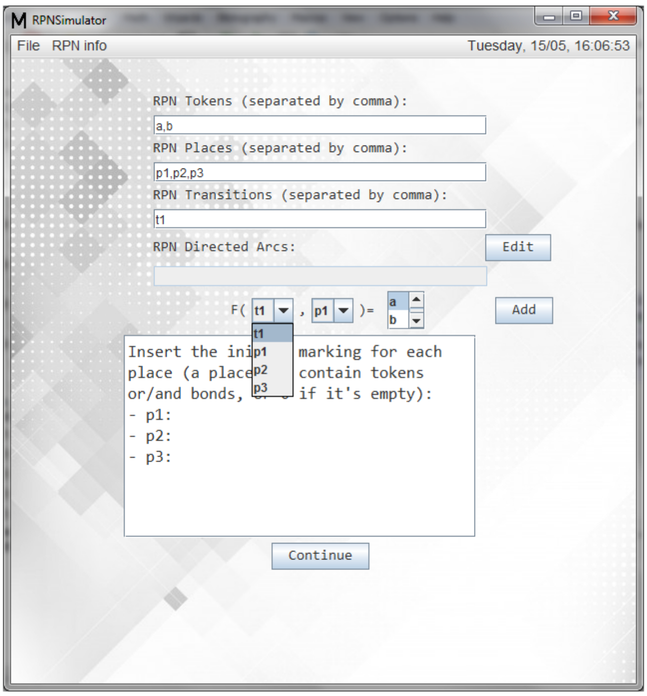


Figure 4.4: The information that the user should import if he/she choose the "Read from User" mode

The user can add a directed arc as follows. From the existing drop-down lists the user can choose the transition or place from which the arc begins and ends, and also the tokens or bonds that are contained on the arc. By pressing the "Add" button the selected values will be added in directed arcs. If the user wants to delete or edit the values of an arc that is already in the list of arcs then he/she can just press the "Edit" button.



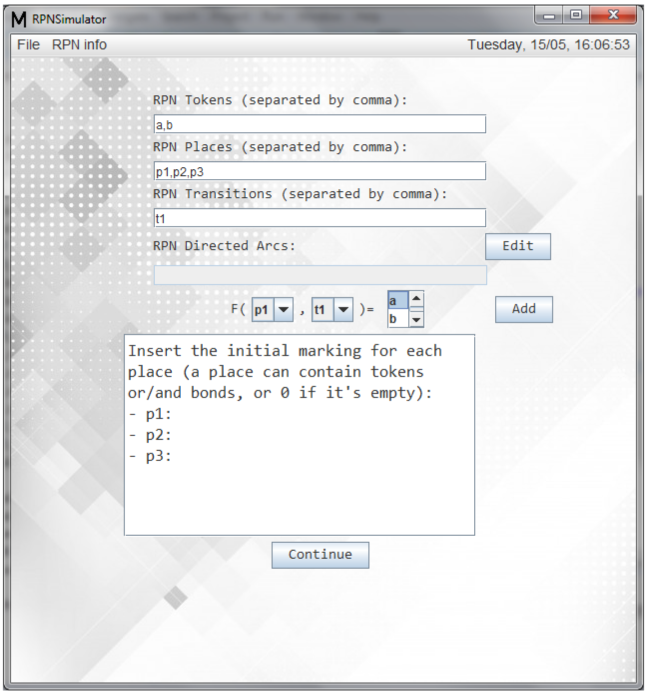
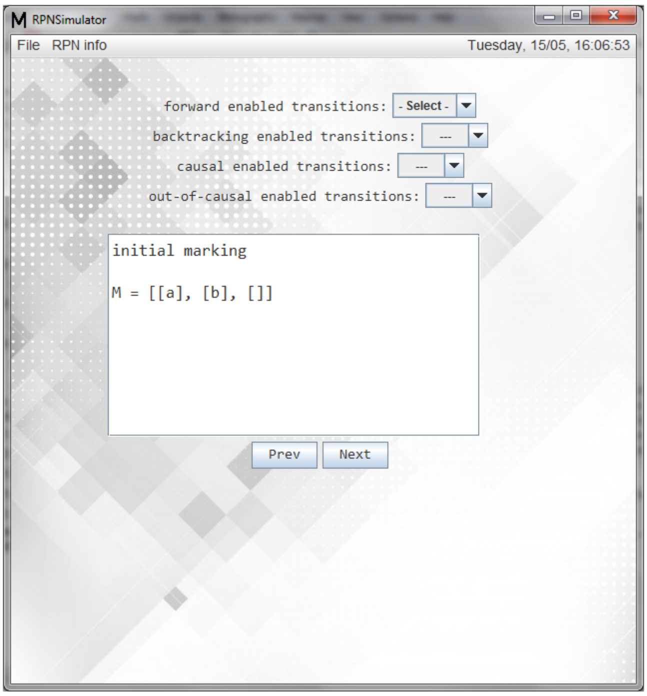


Figure 4.5: The way we insert the information of an arc

The tokens and bonds that appear on an arc can be more than one. The third list of Figure 4.5 shows us that token *a* is selected. That means this token is the only one that exists on the current arc. This list gives us the opportunity to select more than one item. By using the up and down arrows of the list we can move through the list and select all the tokens and bonds that exist on a specific arc. After the completion of the import of all the necessary information needed for representing an RPN the user is able to press the "Continue" button in order to proceed to the next form.



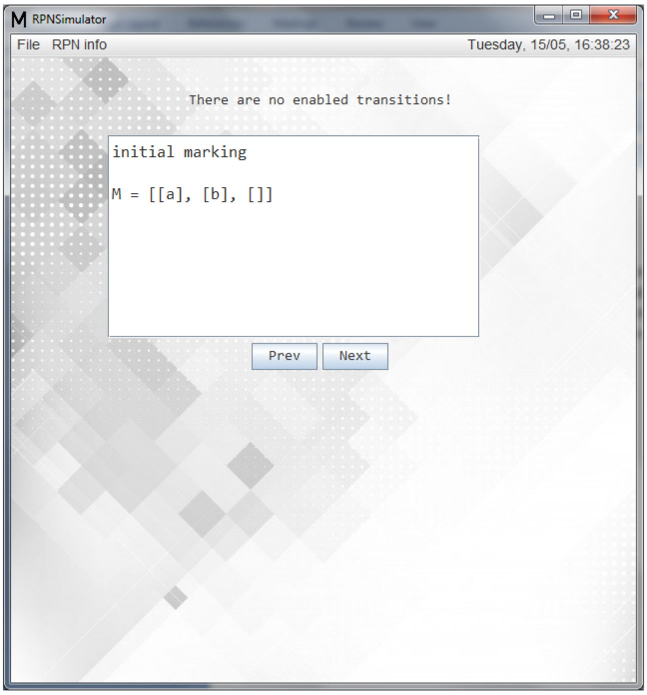
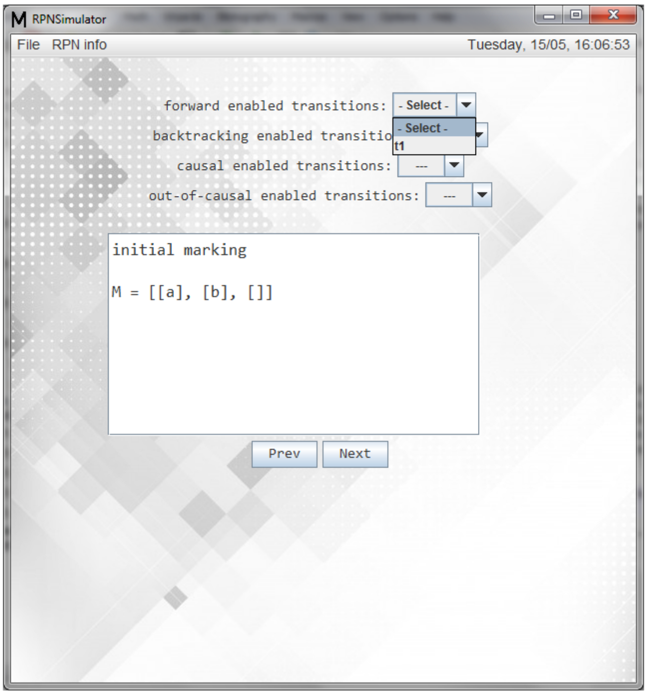


Figure 4.6: The initial appearance of the simulator before we start the transitions execution.

In the bottom form of Fig. 4.6 we can see the enabled transitions of the given model. We can all find four categories of enabled transitions in a drop-down list. We have the forward, backtracking, causal, and out-of-causal-order enabled transitions. If the first item of a drop-down list is the “-Select-” value, this means that the specific list is not empty and there are transitions that are enabled in this order. Otherwise, if the first - and only - value of a drop-down list is “—”, the current list is empty, and there are no transitions that can be executed in this order. If the imported model does not have any enabled transitions then the simulator would display a relevant message (as shown in the bottom form of the above Figure).

Under the four categories of the enabled transitions in Fig. 4.6 there is a text box where the current marking matrix M occurs. At the beginning, the current marking it is the initial marking of the model, and it is presented in the form of a matrix with dimensions $1 \times P$ (where P is the number of places of the RPN model).

Once the user chooses to execute an enabled transition from a drop-down list, the form of the simulator will change. The text box of the form will now contain the new marking, as well as, the transition that the user executed and the category from which he chose the specific transition. The system after the calculation of the new current marking and history matrix, will automatically calculate the new enabled transitions of each category. For that reason, after the execution of a transition, the drop-down lists with enabled transitions will change.



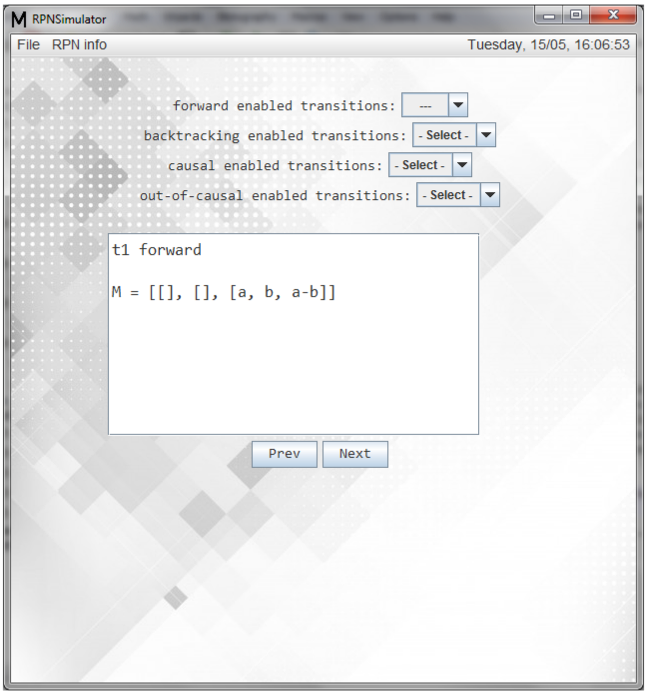
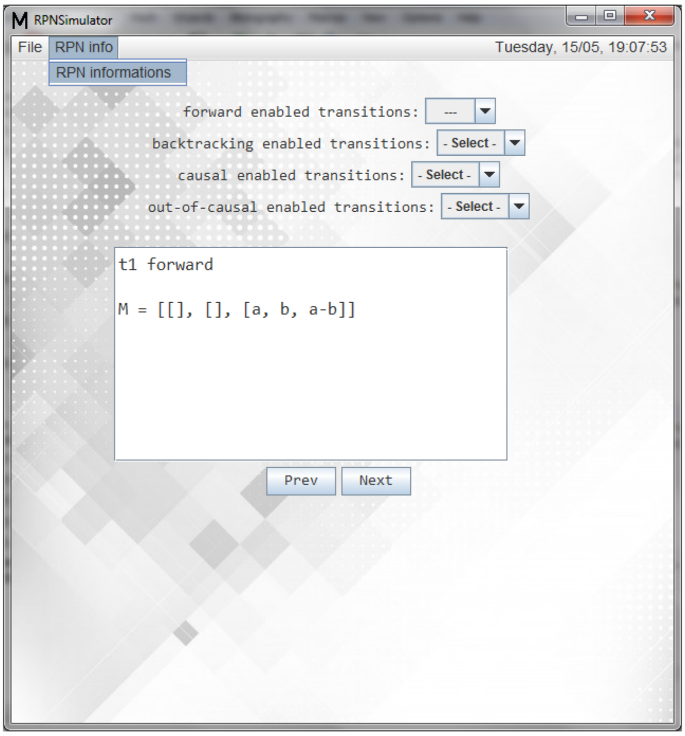


Figure 4.7: After the execution of the forward enabled transition t_1 (in the first form), the simulator will automatically calculate the new marking and the new enabled transitions of each category (as shown in the second form)

Under the text box of the forms in Fig. 4.7 there are two buttons; “Prev” and “Next”. These buttons give to the user the ability to move backward and forward, so he/she can see the previous markings of the model and which transitions have been executed from the beginning of the simulation until now.

There are two tabs on the menu bar of the simulator. The “File” tab contains the “Close” and “Restart” button, and the “RPN info” tab contains the “RPN information” button. As you can see in Figure 4.8, by pressing the “RPN information” button on the menu bar in the “RPN info” tab, the system will display in the form the initial information of the RPN model that have been read. This information contains the tokens A, the places P, the transitions T, the directed arcs F and the initial marking M_0 of the Reversing Petri net. The user can proceed from the point where he was before, choosing the “RPN information”, simply by pressing the “Continue” button. If the system has not yet read any RPN model then this button will not display any information and will therefore not change the current user form.



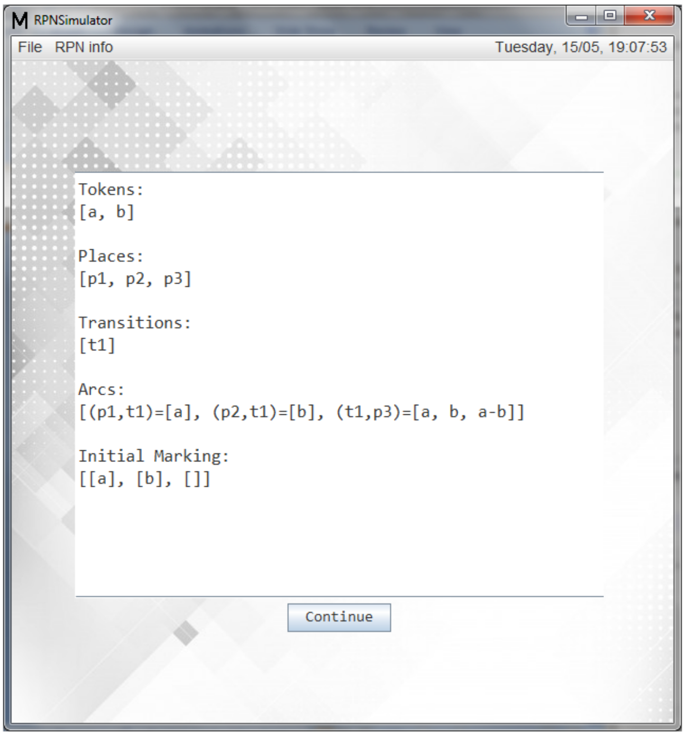
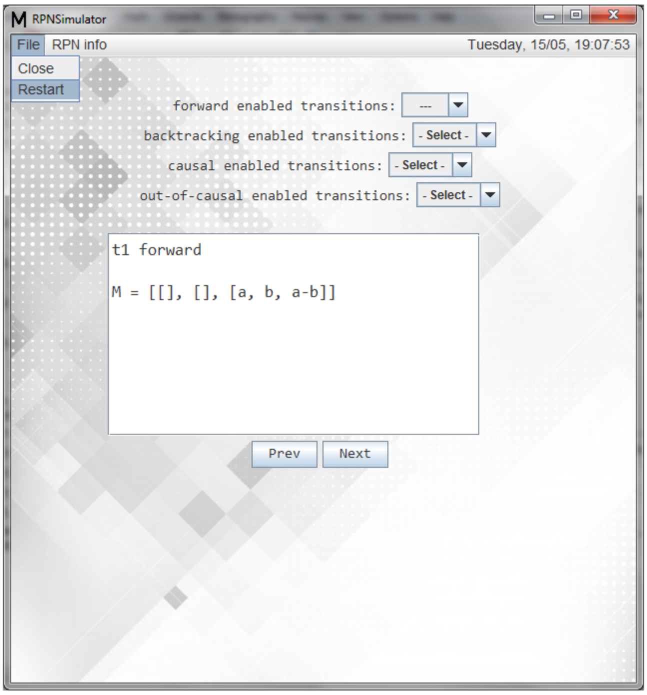


Figure 4.8: By pressing the “RPN information” button (left form), the system displays the initial information of the RPN model (right form).

By pressing the “Restart” button on the menu bar in the “File” tab, the system will give you the opportunity to start again the execution of enabled transitions of the model from the initial marking. It is equivalent to never having execute any enabled transitions from the beginning of the system.

The “Close” button will give the user the opportunity to insert an other Reversing Petri net model, either by using the “Read from User” mode, or by using the “Read from File” mode. Therefore, this button will display the first form that appeared when the user start the simulator, as it shown in Fig. 4.1.



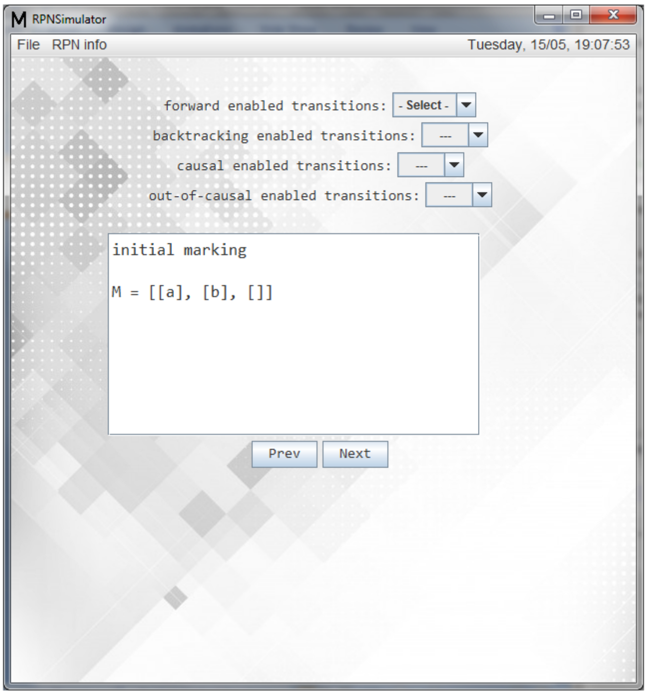


Figure 4.9: By pressing the “Restart” button (left form), the system starts again from the initial marking of the RPN model (right form).

4.4 Simulator Functions

The program and the simulator was implemented on the Java programming language. Each token, transition and place of the Reversing Petri net model was expressed in the program as a String. The ArrayList structure was used to represent the matrices in Java. At the beginning of the program, when the simulator reads the RPN model, its information is automatically stored in an one-dimension ArrayList of Strings. For all the other calculated matrices, the system is using a three-dimension ArrayList of Strings. These lists are three-dimension because we need one dimension to represent the rows of a matrix, one dimension for the representation of the columns of a matrix and the third dimension is a list which is used for the representation of a set of tokens and bonds in each place.

Other than the information of tokens, transitions and places, the simulator also reads information about the directed arcs of the RPN. To store each of these arcs in the system, a new structure was created, as shown below:

```
public class Arc {
    String from;            // the place or transition from where the arc starts
    String to;              // the place or transition where the arc ends
    ArrayList<String> with; // the tokens and/or bonds on the specific arc

    Arc(String f, String t, String w) {…}
}
```

The structure of the directed arc (`Arc`) consists of three elements. The first element is a String named `from`, which is the name of a transition or a place from where the specific arc is beginning. The second element is a String named `to`, corresponding to the name of a transition or a place where the directed arc ends. The third element is a list of Strings named `with` that shows the set of tokens and/or bonds which are labelled on the arc and represent the consumptions or productions of the arc.

4.4.1 Connected Component Method

The method `con(a,b,c)` is aimed at finding the connected components of a given token a in a specific place b. It is a recursive function which takes as first argument a String. This string represents the name of a token, from the set of tokens A, and tries to find all the bonds between the specific token a and any other token in place b, where b is the second argument of the function and it is an ArrayList.

Before checking whether or not there is a bond between a and another token, this function checks whether the token a is already included in the list c. If it is not included then token a is added in the list c before the calculation of its connected components; otherwise the function returns in the previous recursion. The ArrayList c which is the third argument of this function, it helps us to avoid stick in a loop. This function returns an ArrayList which contains all the tokens that are directly or indirectly linked with the given token a.

Algorithm 1: Connected Component Algorithm in the form of pseudo-code

```
Input: a is the token for which we want to find its connected tokens
Mp is a list with the current marking in the place where we want to search for the
   connections of a
C is an auxiliary list which includes the tokens that have already calculated by the
   recursive function con (it helps us prevent the loops)
Output: I is a list which contains all the bases and bonds that are directly and indirectly
   connected to the given token
begin
    foreach element e in Mp do
        if e is a bond x − y then
            if (x = a) then
                insert a in C
                if C does not contain y then
                    insert con(y, Mp, C) in I
                end
                insert e in I
            end
        end
        else if e = a then
            insert e in I
            insert e in C
        end
    end
end
```

4.4.2 Enabled Transitions Method

We have four categories of enabled transitions; forward, backtracking, causal and out-of-causal-order. For each category we have create a method which takes no arguments and returns an ArrayList which contains all the names of the transitions that are enabled in this category.

Method `fenabled()` was created for the forward enabled transitions as described in Definition 1. To identify all the transitions that are forward enabled, this method needs all the information of the RPN, and also the current marking M.

Method `benabled()` was created for the backtracking enabled transitions as described in Definition 3. To identify all the transitions that are backtracking enabled, this method needs the information of the transitions of the RPN, and also the current history H.

Method `coenabled()` was created for the causal enabled transitions as described in Definition 5. To identify all the transitions that are causally enabled, this method needs all the information of the RPN, and also the current history H, and the current marking M.

Method `oenabled()` was created for the out-of-causal-order enabled transitions as described in Definition 6. To identify all the transitions that are out-of-causal enabled, this method needs the information of the transitions of the RPN, and also the current history H.

4.4.3 Addition between Matrices Method

The method `addMatrix(a,b)` calculates the new table created by the addition of the two matrices, *a* and *b*, which are inserted in the function as arguments. This function considers that the three-dimensions ArrayLists, *a* and *b*, have the same number of columns and rows. By taking each position of the first matrix, and the relevant position of the second matrix, it unites the two sets of tokens and/or bonds of the current positions. The created union is stored in the corresponding position of a new matrix, which is a three-dimensions ArrayList that will be returned as soon as the function completes.

Algorithm 2: Matrices Addition Operation Algorithm in the form of pseudo-code

Input: $N = (A, P, B, T, F)$ is the RPN structure
$A[n][m]$ is a matrix which contains sets of tokens and bonds in each position, where $n = |T|$ and $m = |P|$
$B[n][m]$ is a matrix which contains sets of tokens and bonds in each position, where $n = |T|$ and $m = |P|$
Output: $C[n][m]$ is the matrix which contains the addition of the two matrices A and B, where $n = |T|$ and $m = |P|$
begin
    foreach *row x in A* **do**
        foreach *column y in A* **do**
            $C(x,y) \leftarrow A(x,y) \cup B(x,y)$
        end
    end
end

4.4.4 Subtraction between Matrices Method

The method `subMatrix(a,b)` calculates the new table created by the subtraction of the two matrices, *a* and *b*, which are inserted in the function as arguments. This function considers that the three-dimensions ArrayLists, *a* and *b*, have the same number of columns and rows. It removes from each position of the first matrix, all the tokens and bonds which are included in the relevant position of the second matrix. The set remaining after deduction is stored in the corresponding position of a new matrix, which is a three-dimension ArrayList that will be returned as soon as the function completes.

Algorithm 3: Matrices Subtraction Operation Algorithm in the form of pseudo-code

```
Input: N = (A,P,B,T,F) is the RPN structure
A[n][m] is a matrix which contains sets of tokens and bonds in each position, where
  n = |T| and m = |P|
B[n][m] is a matrix which contains sets of tokens and bonds in each position, where
  n = |T| and m = |P|
Output: C[n][m] is the matrix which contains the subtraction of matrix B from matrix A,
  where n = |T| and m = |P|
begin
    foreach row x in A do
        foreach column y in A do
            C(x,y) ← A(x,y) − B(x,y)
        end
    end
end
```

4.4.5 Multiplication between Matrices Method

The method `mulMatrix(a,b)` calculates the new table created by the multiplication of the two matrices, a and b, which are inserted in the function as arguments. This function considers that the first matrix has dimensions $1 \times j$, and consists of integer numbers, 0 or 1, and the second matrix has dimensions $j \times k$, and consists of tokens and/or bonds. By taking each position $j_i \times k_i$ of the second matrix, checks whether the integer in position $1 \times j_i$ of the first matrix is the number 1, and if that is true then the set of the second matrix in position $j_i \times k_i$ is stored in position $1 \times k_i$ of a new matrix. This function returns the new matrix, which is a three-dimensions ArrayList with bonds and/or tokens.

Algorithm 4: Matrices Multiplication Operation Algorithm in the form of pseudo-code

```
Input: N = (A,P,B,T,F) is the RPN structure
A[n] is a matrix which contains 0s and 1s in each position, where n = |T|
B[n][m] is a matrix which contains sets of tokens and bonds in each position, where
  n = |T| and m = |P|
Output: C[1][m] is the matrix which contains the multiplication of the two matrices A
  and B, where m = |P|
begin
    foreach column x in B do
        foreach row y in B do
            if A(y) = 1 then
                C(1,x) ← C(1,x) ∪ B(y,x)
            end
        end
    end
end
```

4.4.6 D Matrices Calculation Method

The function `calcDmatrices()` puts values in the public lists of the program, D^- and D^+. Each of these lists is a three-dimensions $t \times p$ ArrayList, where t is the number of transitions of the model, and p is the number of places of the model. For each directed arc of the RPN, the system checks whether there is an incoming arc or an outgoing arc. An incoming arc, from a place i to a transition j, is added in matrix D^- in position $[j,i]$; an outgoing arc, from a transition j to a place i, is added in matrix D^+ in position $[j,i]$.

4.4.7 Connected Component Matrix Method

The method `conMatrix(a,b)` is aimed at finding the connected components of each token that exist in a given matrix. The first argument of this function is a three-dimensions ArrayList with tokens, and it is the list that will be used for the process of function. As we know from the Connected Component Method which has been described in section 4 we can find the tokens that are directly or indirectly connected with a given token in a specific place. In some cases we want to find the connected tokens after deducting an effect from a specific place. The second argument b is an integer which takes values -1 or 1. In case of a -1 value the connected component will be calculated from a specific marking, after removing the effect of the transition that is being executed at the moment. Otherwise, the connected component will be calculated directly from the current marking without any deduction. This function returns the new three-dimensions ArrayList which created with all connected components of the inserted matrix.

4.4.8 Forward Execution Method

Method `ForwardExec()` was created for the forward execution of a transition as described in Definition 17. After the step-by-step calculation of all matrices that is needed - $TD^-, TD^+, CD^- and CD^+$ -, this function will find the new marking M, and also the new history H of the system.

For a new marking value the function is adding the current marking and the productions of the executed transition, with all the connected tokens and bonds that come with it, and later it subtracts from the new matrix the consumptions of the executed transition. For new history value, the function, after finding the maximum value of the history, increases this value by one, and puts it in place of the transition executed. This function does not return anything since all changes are made directly on matrices M and H.

4.4.9 Backtracking Execution Method

Method `Backtracking()` was created for the backtracking and causal execution of a transition as described in Definition 19. After the step-by-step calculation of all matrices that is needed - $TD^-, TD^+, CD^- and CD^+$ -, this function will find the new marking M, and also the new history H of the system.

For new marking value the function is adding current marking and the consumptions of the executed transition, and later is subtract from the new matrix the productions of the executed transition, with all the connected tokens and bonds that come with it. For new history value, the function, after finding the value of the history in position of the executed transition, removes this value from that position. This function does not return anything since all changes are made directly on matrices M and H.

4.4.10 Out-of-Causal-Order Execution Method

Method `OutOfCausalExec()` was created for the out-of-causal-order execution of a transition as described in Definition 21. After the step-by-step calculation of all matrices that is needed - $L^{-}, L^{+}, CL^{-} and CL^{+}$ -, this function will find the new marking M, and also the new history H of the system.

For new marking value the function is subtract from each place of matrix E - current marking without the effect of executed transition - all tokens and bonds which are not in the right place, and later is adding in each place the tokens and bonds that should be in each position, based on the changes caused by the reversed transition. When calculating the new history value, the function finds the value of the history in position of the executed transition and removes this value from that position. This function does not return anything since all changes are made directly on matrices M and H.

Algorithm 5: Forward Execution Algorithm in the form of pseudo-code

Input: M is the current marking matrix
H is the current history matrix
FT is the current executed transition matrix
$N = (A,P,B,T,F)$ is the RPN structure
Output: M is the new marking matrix
H is the new history matrix
Variables: $D^+[n][m]$ is the matrix of outgoing arcs, where $n = |T|$ and $m = |P|$
$D^-[n][m]$ is the matrix of incoming arcs, where $n = |T|$ and $m = |P|$
$TD^+[m]$ is the matrix of outgoing arcs for the executed transition, where $m = |P|$
$TD^-[m]$ is the matrix of incoming arcs for the executed transition, where $m = |P|$
$CD^+[m]$ is the matrix with connected component of TD^+, where $m = |P|$
$CD^-[m]$ is the matrix with connected component of TD^-, where $m = |P|$
$MC^+[m]$, $MD^+[m]$ are intermediate matrices used, where $m = |P|$
max is the maximum number of matrix H
begin
    foreach *arc* $F(x,y)$ **do**
        if $F(x,y)$ *is an incoming arc* **then**
            $D^-(y,x) \leftarrow$ tokens on $F(x,y)$
        end
        else
            $D^+(x,y) \leftarrow$ tokens on $F(x,y)$
        end
    end
    $TD^+ \leftarrow mulMatrix(FT, D^+)$
    $TD^- \leftarrow mulMatrix(FT, D^-)$
    foreach *position* p *in* TD^+ **do**
        foreach *token* a *in* p **do**
            foreach *place* q *in* M **do**
                $CD^+(p) \leftarrow CD^+(p) \cup con(a, M(q))$
            end
        end
    end
    foreach *position* p *in* TD^- **do**
        foreach *token* a *in* p **do**
            foreach *place* q *in* M **do**
                $CD^-(p) \leftarrow CD^-(p) \cup con(a, M(q))$
            end
        end
    end
    $MC^+ \leftarrow addMatrix(M, CD^+)$
    $MD^+ \leftarrow addMatrix(MC^+, TD^+)$
    $M \leftarrow subMatrix(MD^+, CD^-)$
    $max \leftarrow$ the maximum number in matrix H
    foreach *position* i *in* H **do**
        $H(i) \leftarrow H(i) + (max + 1) \times FT(i)$
    end
end

Algorithm 6: Backtracking and Causal Execution Algorithm in the form of pseudo-code

```
Input: M is the current marking matrix
H is the current history matrix
FT is the current executed transition matrix
N = (A, P, B, T, F) is the RPN structure
Output: M is the new marking matrix
H is the new history matrix
Variables: D+[n][m] is the matrix of outgoing arcs, where n = |T| and m = |P|
D−[n][m] is the matrix of incoming arcs, where n = |T| and m = |P|
TD+[m] is the matrix of outgoing arcs for the executed transition, where m = |P|
TD−[m] is the matrix of incoming arcs for the executed transition, where m = |P|
CD+[m] is the matrix with connected component of TD+, where m = |P|
CD−[m] is the matrix with connected component of TD−, where m = |P|
MC−[m] is intermediate matrix used, where m = |P|
eff is the effect of transition that is reversing
E is the current marking matrix without the eff variable
executed is the number exist in position j where FT(j) = 1
begin
    t ← the transition where FT(t) = 1
    eff ← pre(t) − post(t)
    foreach place w in M do
        E(w) ← M(w) − eff
    end
    foreach arc F(x,y) do
        if F(x,y) is an incoming arc then
            D−(y,x) ← tokens on F(x,y)
        end
        else
            D+(x,y) ← tokens on F(x,y)
        end
    end
    TD+ ← mulMatrix(FT, D+)
    TD− ← mulMatrix(FT, D−)
    foreach position p in TD+ do
        foreach token a in p do
            foreach place q in M do
                CD+(p) ← CD+(p) ∪ con(a, M(q))
            end
        end
    end
    foreach position p in TD− do
        foreach token a in p do
            foreach place q in M do
                CD−(p) ← CD−(p) ∪ con(a, E(q))
            end
        end
    end
    MC− ← addMatrix(M, CD−)
    M ← subMatrix(MC−, CD+)
    executed ← the number exist in position j where FT(j) = 1
    foreach position i in H do
        H(i) ← H(i) − executed × FT(i)
    end
end
```

Algorithm 7: Out-of-Causal-Order Execution Algorithm in the form of pseudo-code

```
Input: M is the current marking matrix
H is the current history matrix
FT is the current executed transition matrix
N = (A, P, B, T, F) is the RPN structure
Output: M is the new marking matrix
H is the new history matrix
Variables: L^+[m] is the matrix which contains the tokens to be added to the marking in
  the corresponding places, where m = |P|
L^-[m] is the matrix which contains the tokens to be removed from the marking from the
  corresponding places, where m = |P|
CL^+[m] is the matrix with connected component of L^+, where m = |P|
CL^-[m] is the matrix with connected component of L^-, where m = |P|
MC^-[m] is intermediate matrix used, where m = |P|
eff is the effect of transition that is reversing
E is the current marking matrix without the eff variable
executed is the number exist in position j where FT(j) = 1
begin
    t ← the transition where FT(t) = 1
    eff ← pre(t) − post(t)
    foreach place w in M do
        E(w) ← M(w) − eff
    end
    L^+, L^- = calcLmatrices()
    foreach position p in L^+ do
        foreach token a in p do
            foreach place q in M do
                CL^+(p) ← CL^+(p) ∪ con(a, E(q))
            end
        end
    end
    foreach position p in L^- do
        foreach token a in p do
            foreach place q in M do
                CL^-(p) ← CL^-(p) ∪ con(a, E(q))
            end
        end
    end
    ML^- ← subMatrix(E, CL^-)
    M ← addMatrix(ML^-, CL^+)
    executed ← the number exist in position j where FT(j) = 1
    foreach position i in H do
        H(i) ← H(i) − executed × FT(i)
    end
end
```

4.4.11 Last Transition Calculation Method

The method `last(a)` was created as described in Definition 7. It aims to find the last transition of the model that has been executed, and contains the given set of tokens a on its outgoing arc. This method takes as an argument an ArrayList that represents a set of tokens and bonds for which we want to find out their last position in the model. The function returns the position of the transition in the transition set T for which the above restrictions are true. If there are no transitions for which these restrictions are true then it returns the value -1.

Algorithm 8: Last Transition Calculation Algorithm in the form of pseudo-code

Input: C is the set of tokens and bonds for which we want to find their last executed transition
M is the current marking matrix
H is the current history matrix
FT is the current executed transition matrix
$N = (A, P, B, T, F)$ is the RPN structure
Output: *lastt* is the position of the last executed transition that is using the given set, in T
Variables: *last_history_value* is the history value of the temporary last transition
begin
    $lastt \leftarrow -1$
    $last\_history\_value \leftarrow -1$
    foreach *transition t in T* **do**
        if $(H(t) \neq 0)$ *and* $(\mathsf{post}(t) \cap C \neq \emptyset)$ *and* $(H(t) > last\_history\_value)$ **then**
            $lastt \leftarrow t$
            $last\_history\_value \leftarrow H(t)$
        end
    end
end

4.4.12 L Matrices Calculation Method

The function `calcLmatrices()` puts values in the public lists of the program, L^- and L^+. Each of these lists is a three-dimension ArrayList with dimensions $1 \times p$, where p is the number of places of the model. L^- list contains in each position the tokens that are in the wrong place on the model and must be removed from the specific place of marking. L^+ list contains in each position the tokens that must be added in the specific place of marking. After the calculation of the connected component of each token, this function calculates the last place of each connected component, and finds out the two lists, L^- and L^+.

Algorithm 9: L Matrices Calculation Algorithm in the form of pseudo-code

```
Input: M0 is the initial marking matrix
M is the current marking matrix
FT is the current executed transition matrix
N = (A, P, B, T, F) is the RPN structure
Output: L+ is the matrix which contains the tokens to be added to the marking in the
  corresponding places
L− is the matrix which contains the tokens to be removed from the marking from the
  corresponding places
Variables: eff is the effect of transition that is reversing
E is the current marking matrix without the eff variable
conn is the set of tokens that are directly or indirectly connected with a specific token
lastt is the last executed transition that is using a specific token
begin
    t ← the transition where FT(t) = 1
    eff ← pre(t) − post(t)
    foreach place w in M do
        E(w) ← M(w) − eff
    end
    foreach place p in P do
        foreach token a in p do
            foreach place q in M do
                conn ← conn ∪ con(a, E(q))
            end
            lastt ← last(conn)
            if lastt ≠ ⊥ then
                foreach arc F(x, p) do
                    if (F(x, p) ∩ conn ≠ ∅) and (lastt = x) then
                        L+(p) ← L+(p) ∪ a
                    end
                end
            end
            else if conn ⊆ M0(p) then
                L+(p) ← L+(p) ∪ a
            end
            if a ∈ M(p) then
                foreach arc F(z, p) do
                    if lastt ≠ z then
                        L−(p) ← L−(p) ∪ a
                    end
                end
            end
        end
    end
end
```

Chapter 5

Case Study

Contents

In this chapter we are going to examine how the matrices of Reversing Petri nets can be applied on an example given from the product assembly process, and whether or not it is possible to disassemble the product by using the same Reversing Petri net model and the matrices that are derived from this.

5.1 Assembly and Disassembly

In a time of information revolution, assembly, one of the oldest forms of industrial production, and its twin task, disassembly, have experienced enormous modernization. Assembly is a productive function of the composition of some individual parts, subassemblies and substances in a predetermined quantity and within a predetermined period of time. The assembly process is one of the most expensive and time-consuming activities in the area of manufacturing.

Disassembly is defined as the set of all processes that decompose the structure of geometrically defined bodies over a given period of time. The aspects of disassembly must be taken into account in various steps of the product life cycle, during both the design process of the product and in the process of designing the disassembly of the end-of-life products. Maintenance, remanufacturing, recycling or disposal of end-of-life products, are some of the basic objectives of disassembly process.

While assembly procedures have existed since ancient history, disassembly has become famous during the last decades as a response to society's needs for recycling and remanufacturing. As the complexity of products and production systems increases, the need for models that deal with assembly and disassembly aspects is becoming greater. Specifically, the kind of models that would be more useful for this kind of representations are the reversible models, since the disassembly process is actually the reversing order of the assembly execution process.

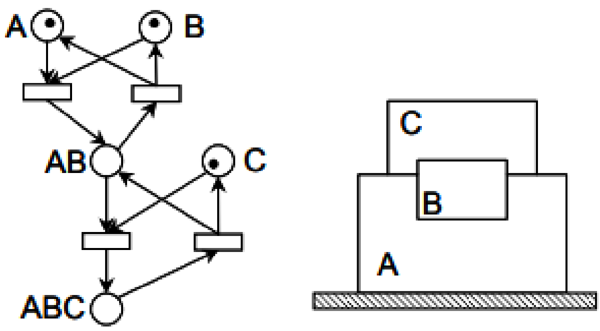


Figure 5.1: The Petri net model (shown in left) capturing the relations between the parts of an assembly product (shown in right).

As a kind of reversible model, Petri nets facilitate the representation of the subtasks into which an assembly model can be decomposed by taking into account the preconditions and post-conditions which are used for the specification of the feasible sequences.

Figure 5.1 illustrates the Petri net model of a very simple assembly product. This product consists of 3 parts, A, B and C. For each of these parts, a new place has been added to the network. Within each place that represents a part of the product there is a token, this represents the fact that every part of the product exists only once. From the Petri net that exists in the left scheme of figure we can notice which parts can be connected together. For example, we can see that C can be added to the product only when parts A and B have already been connected together.

As we mention in previous chapters, Petri net models, give the opportunity to study the correctness of a system using the qualitative analysis they provide. In this case, when a Petri net used for assembly/disassembly modelling [8], the correctness of this system can be studied by using the qualitative analysis of the net. Since Petri nets are ideal models for the representation of assembly and disassembly processes, this means that Reversing Petri nets are equally appropriate.

5.2 Ballpoint pen Case Study

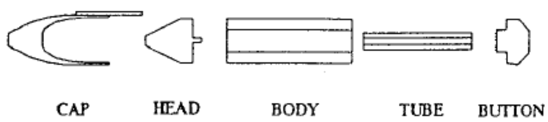


Figure 5.2: Ballpoint pen [3]

In Fig. 5.2 above we can see the sub-pieces of a ballpoint pen. We have five pieces; cap, body, tube, head and button. In the product's (ball point pen) reversing Petri net (RPN) model below (Fig. 5.3) we can see that for each of the above five sub-pieces we assign a token in the RPN model. We use C for Cap, O for Body, T for Tube, H for Head and U for Button.

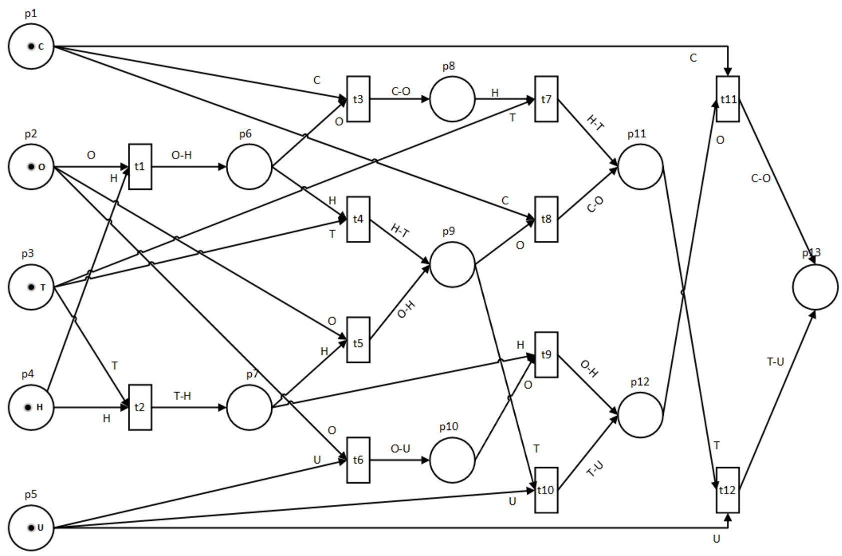


Figure 5.3: Pen Assembly/disassembly demonstration in Reversing Petri Nets

The RPN contains 13 places and 12 transitions. These numbers will be used later for the creation of this Reversing Petri net's matrices in simulator algorithms. Below we are going to demonstrate the assembly and disassembly of this product by using any of the four types of execution (that is enabled) within RPNs - forward execution, backtracking, causal order and out-of-causal-order execution. In addition to the execution of the simulator, the matrix equations that are executed through the simulator algorithms are shown below.

The reversing Petri net in Figure 5.3 can be specified in matrix form as follows:

1. Find D^- matrix with the incoming arcs of Fig. 5.3.

$$D^- = \begin{pmatrix}
0 & \{O\} & 0 & \{H\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \{T\} & \{H\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\{C\} & 0 & 0 & 0 & 0 & \{O\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \{T\} & 0 & 0 & \{H\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \{O\} & 0 & 0 & 0 & 0 & \{H\} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \{O\} & 0 & 0 & \{U\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \{T\} & 0 & 0 & 0 & 0 & \{H\} & 0 & 0 & 0 & 0 & 0 \\
\{C\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{O\} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \{H\} & 0 & 0 & \{O\} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \{U\} & 0 & 0 & 0 & \{T\} & 0 & 0 & 0 & 0 \\
\{C\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{O\} & 0 \\
0 & 0 & 0 & 0 & \{U\} & 0 & 0 & 0 & 0 & 0 & \{T\} & 0 & 0
\end{pmatrix}$$

2. Find D^+ matrix with the outgoing arcs of Fig. 5.3.

$$D^+ = \begin{pmatrix}
00000 & \{O-H\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
00000 & 0 & \{T-H\} & 0 & 0 & 0 & 0 & 0 & 0 \\
00000 & 0 & 0 & \{C-O\} & 0 & 0 & 0 & 0 & 0 \\
00000 & 0 & 0 & 0 & \{H-T\} & 0 & 0 & 0 & 0 \\
00000 & 0 & 0 & 0 & \{O-H\} & 0 & 0 & 0 & 0 \\
00000 & 0 & 0 & 0 & 0 & \{O-U\} & 0 & 0 & 0 \\
00000 & 0 & 0 & 0 & 0 & 0 & \{H-T\} & 0 & 0 \\
00000 & 0 & 0 & 0 & 0 & 0 & \{C-O\} & 0 & 0 \\
00000 & 0 & 0 & 0 & 0 & 0 & 0 & \{O-H\} & 0 \\
00000 & 0 & 0 & 0 & 0 & 0 & 0 & \{T-U\} & 0 \\
00000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{C-O\} \\
00000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{T-U\}
\end{pmatrix}$$

At the beginning, marking matrix M is the same with the matrix M_0, which contains the initial marking.

Marking matrix for Fig. 5.3:

$$M = M_0 = \begin{pmatrix} \{C\} & \{O\} & \{T\} & \{H\} & \{U\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

At this moment the only enabled transitions are transitions t_1, t_2 and t_6, which are forward enabled. Let us assume that we are going to fire transition t_1, using **forward execution**.

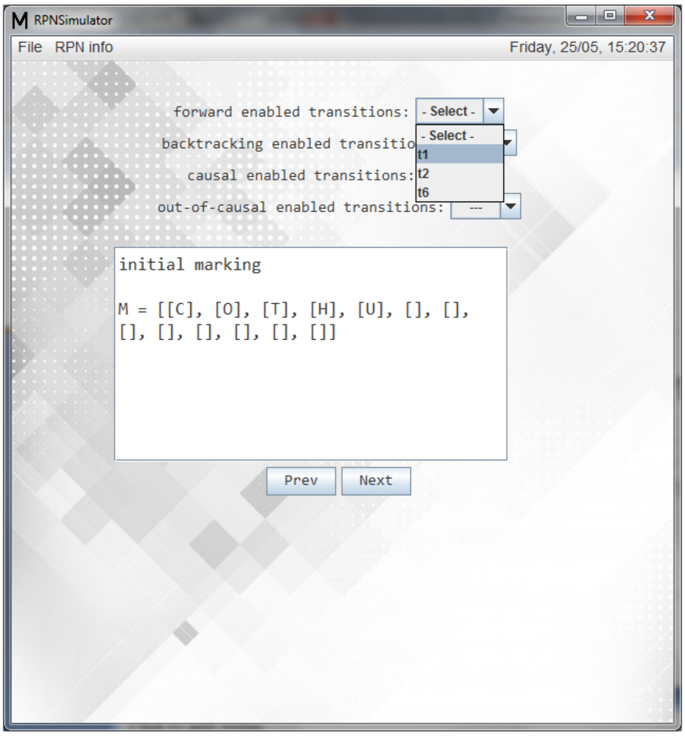


Figure 5.4: The initial marking of the Reversing Petri net and the forward enabled transitions

Transition matrix for Fig. 5.3:

$$FT = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

To determine the new marking of the RPN after the firing of the transition specified in the transition matrix, we create the following matrices.

$$TD^{+} = FT \otimes D^{+} \;=\; \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & \{O-H\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$TD^{-} = FT \otimes D^{-} \;=\; \begin{pmatrix} 0 & \{O\} & 0 & \{H\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Matrix CD^{+} is the same with TD^{+}, and matrix CD^{-} is the same with TD^{-}.

The initial history matrix H of Fig. 5.3 contains only 0s since no transition yet fire.
History matrix for Fig. 5.3:

$$H = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

After the calculation of the above matrices we can now calculate the new marking matrix M':

$$\begin{aligned}
M' &= M \oplus CD^{+} \oplus TD^{+} \ominus CD^{-} \\
&= \begin{pmatrix} \{C\} & \{O\} & \{T\} & \{H\} & \{U\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \oplus \\
&\quad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & \{O-H\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \oplus \\
&\quad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & \{O-H\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \ominus \\
&\quad \begin{pmatrix} 0 & \{O\} & 0 & \{H\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
&= \begin{pmatrix} \{C\} & 0 & \{T\} & 0 & \{U\} & \{O-H\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}
\end{aligned}$$

and the new history matrix H':

$$\begin{aligned}
H' &= H \oplus (max\{k|k = H(t), t \in T\} + 1) \times FT \\
&= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \oplus \\
&\quad 1 \times \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}
\end{aligned}$$

The simulator algorithms will calculate the above matrix equations and as a result the simulator will show to the user the following screen (as shows in Figure 5.5):

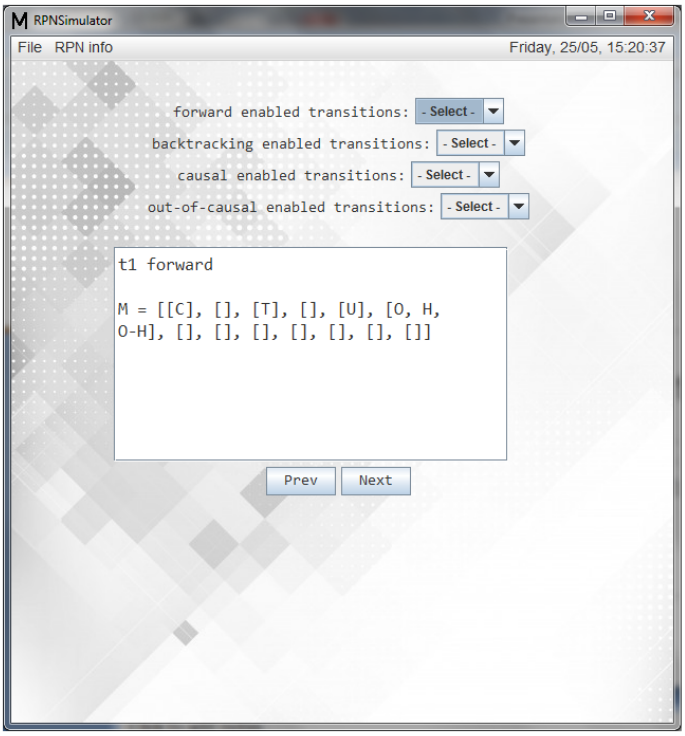


Figure 5.5: Simulator appearance after the forward execution of t_1

Graphically the appearance of the Reversing Petri net model will change, since the tokens will change places and new bonds will be created as shown below:

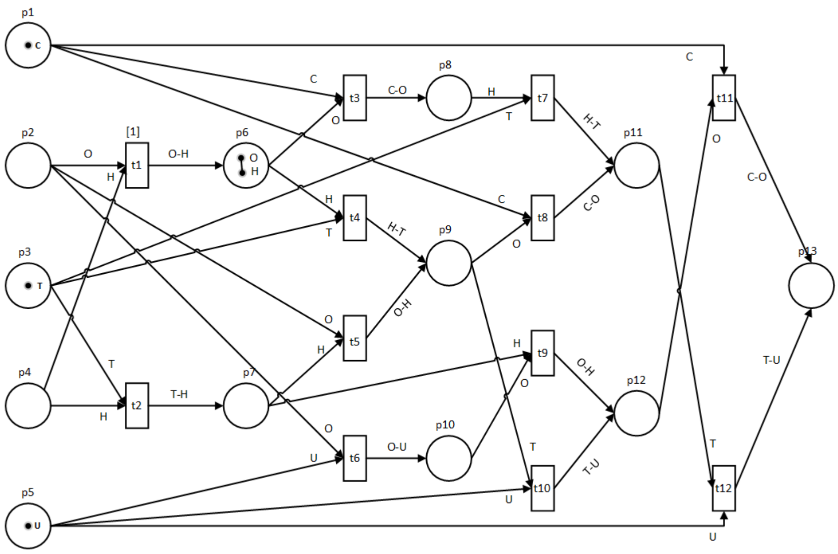


Figure 5.6: Pen Assembly/disassembly RPN after the forward execution of t_1

After the calculation of the new marking and history matrices the new enabled transitions sets will change. According to the new changes the forward enabled transitions now are t_3 and t_4.

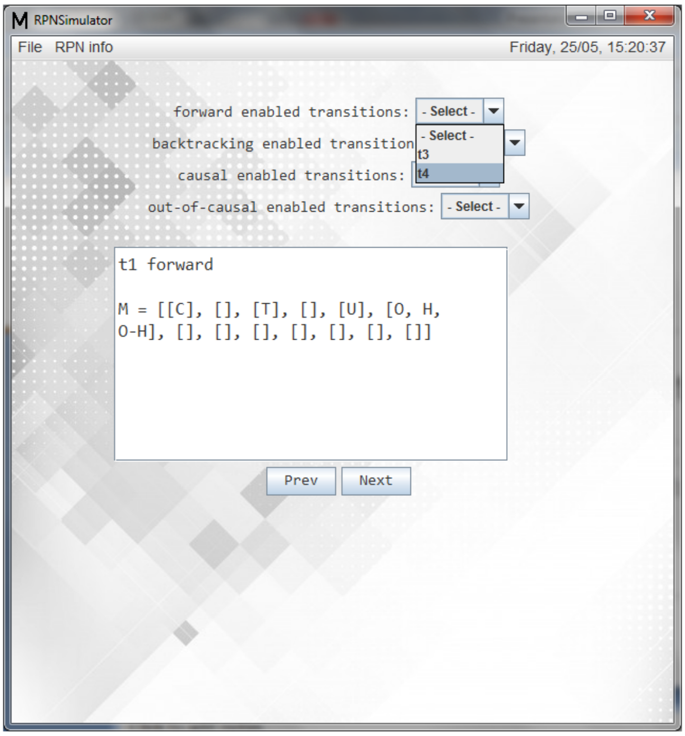


Figure 5.7: The current marking of the Reversing Petri net and the forward enabled transitions

Let us assume that we are going to fire transition t_4.

Transition matrix for Fig. 5.6:

$$FT = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

To determine the new marking of the RPN after the firing of the transition specified in the transition matrix, we create the following matrices.

$$TD^+ = FT \otimes D^+ \;=\; \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{H-T\} & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$TD^- = FT \otimes D^- \;=\; \begin{pmatrix} 0 & 0 & \{T\} & 0 & 0 & \{H\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Matrices CD^+ and CD^- contains the connected component of each base of matrices TD^+ and TD^-, respectively.

$$CD^+ \;=\; \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{O-H,T\} & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$CD^- \;=\; \begin{pmatrix} 0 & 0 & \{T\} & 0 & 0 & \{O-H\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

After the calculation of the above matrices we can calculate the new marking matrix M':

$$\begin{aligned}
M' &= M \oplus CD^{+} \oplus TD^{+} \ominus CD^{-} \\
&= \begin{pmatrix} \{C\} & 0 & \{T\} & 0 & \{U\} & \{O-H\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \oplus \\
&\quad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{O-H,T\} & 0 & 0 & 0 & 0 \end{pmatrix} \oplus \\
&\quad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{H-T\} & 0 & 0 & 0 & 0 \end{pmatrix} \ominus \\
&\quad \begin{pmatrix} 0 & 0 & \{T\} & 0 & 0 & \{O-H\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
&= \begin{pmatrix} \{C\} & 0 & 0 & 0 & \{U\} & 0 & 0 & 0 & \{O-H,H-T\} & 0 & 0 & 0 & 0 \end{pmatrix}
\end{aligned}$$

and the new history matrix H':

$$\begin{aligned}
H' &= H \oplus (max\{k|k = H(t), t \in T\} + 1) \times FT \\
&= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \oplus \\
&\quad 2 \times \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}
\end{aligned}$$

The simulator algorithms will calculate the above matrix equations and as a result the simulator will show to the user the following screen (as shows in Figure 5.8):

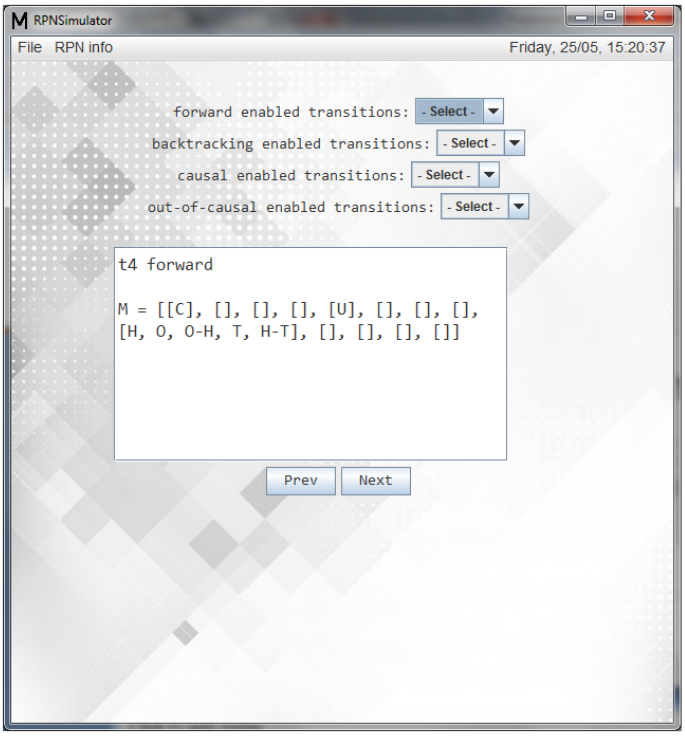


Figure 5.8: Simulator appearance after the forward execution of t_4

Graphically the appearance of the Reversing Petri net model will change, since the tokens will change places and new bonds will be created as shown below:

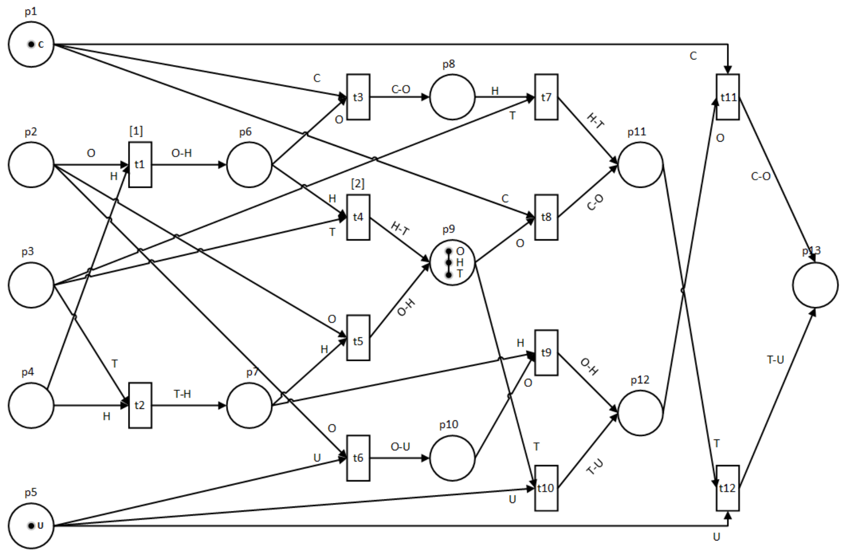


Figure 5.9: Pen Assembly/disassembly RPN after the forward execution of t_4

Let us assume that after the forward execution of transition t_4, the transition t_{10} has been executed also in forward. So the simulator appearance, as well as the RPN model has been changed as shown in Figure 5.10 and Figure 5.11, respectively.

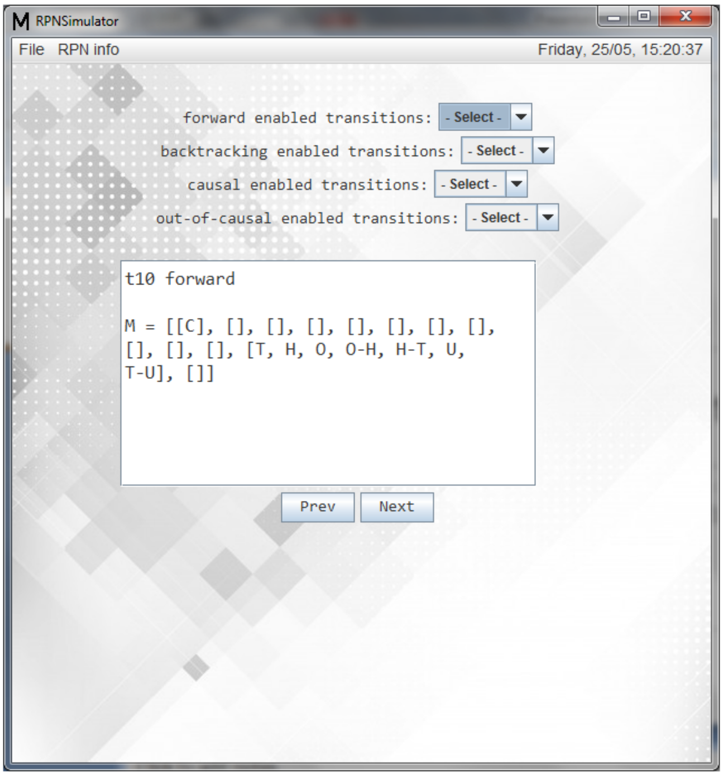


Figure 5.10: Simulator appearance after the forward execution of t_{10}

The current marking matrix M' is:

$$M' = \begin{pmatrix} \{C\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{O-H, H-T, T-U\} & 0 \end{pmatrix}$$

and the current history matrix H':

$$H' = \begin{pmatrix} 1 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 \end{pmatrix}$$

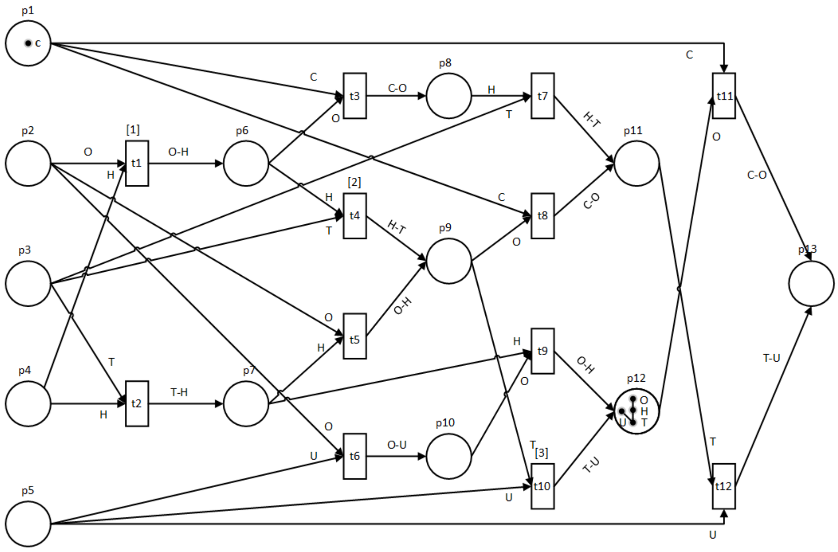


Figure 5.11: Pen Assembly/disassembly RPN after the forward execution of t_{10}

Let's assume that we are going to reverse transition t_{10}, using **out-of-causal-order execution**.

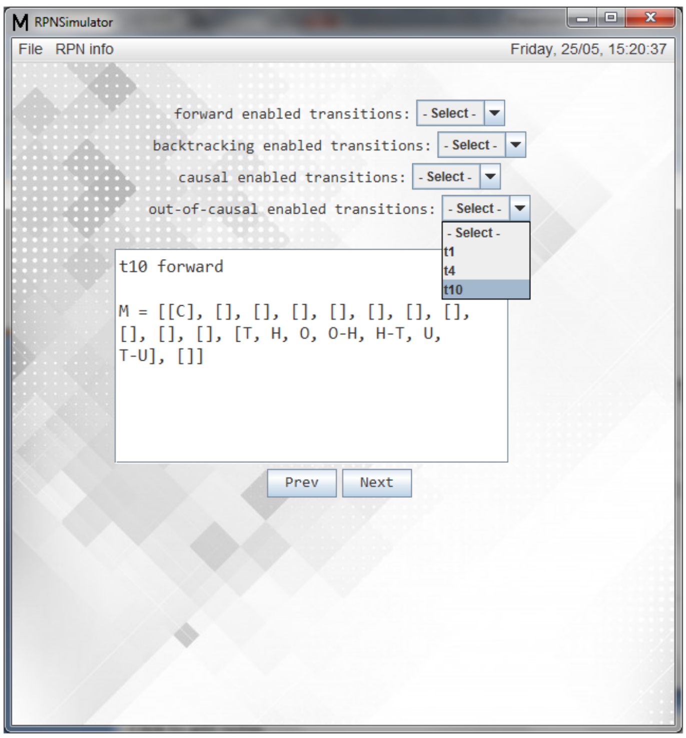


Figure 5.12: The current marking of the Reversing Petri net and the out-of-causal-order enabled transitions

Transition matrix for Fig. 5.11:

$$FT = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

When transition t_{10} is reversing, and since the effect of t_{10} is the bond $(T-U)$), matrix E has the following values :

$$E = \begin{pmatrix} \{C\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{O-H,H-T,U\} & 0 \end{pmatrix}$$

To determine the new marking of the RPN after the reversing of the transition specified in the transition matrix, we create the following matrices.

$$L^{+} = \begin{pmatrix} \{C\} & 0 & 0 & 0 & \{U\} & 0 & 0 & 0 & \{O,H,T\} & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$L^{-} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{O,H,T,U\} & 0 \end{pmatrix}$$

Matrices CL^{+} and CL^{-} contains the connected component of each base of matrices L^{+} and L^{-}, respectively.

$$CL^{+} = \begin{pmatrix} \{C\} & 0 & 0 & 0 & \{U\} & 0 & 0 & 0 & \{O-H,H-T\} & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$CL^{-} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{O-H,H-T,U\} & 0 \end{pmatrix}$$

After the calculation of the above matrices we can calculate the new marking matrix M':

$$\begin{aligned} M' &= E \ominus CL^{-} \oplus CL^{+} \\ &= \begin{pmatrix} \{C\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{O-H,H-T,U\} & 0 \end{pmatrix} \ominus \\ &\quad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{O-H,H-T,U\} & 0 \end{pmatrix} \oplus \\ &\quad \begin{pmatrix} \{C\} & 0 & 0 & 0 & \{U\} & 0 & 0 & 0 & \{O-H,H-T\} & 0 & 0 & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} \{C\} & 0 & 0 & 0 & \{U\} & 0 & 0 & 0 & \{O-H,H-T\} & 0 & 0 & 0 & 0 \end{pmatrix} \end{aligned}$$

and the new history matrix H':

$$\begin{aligned} H' &= H \ominus (\{k \mid k = H(t), t \in T, FT(t) = 1\} \times FT) \\ &= \begin{pmatrix} 1 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 \end{pmatrix} \ominus \\ &\quad 3 \times \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{aligned}$$

The simulator algorithms will calculate the above matrix equations and as a result the simulator will show to the user the following screen (as shows in Figure 5.13):

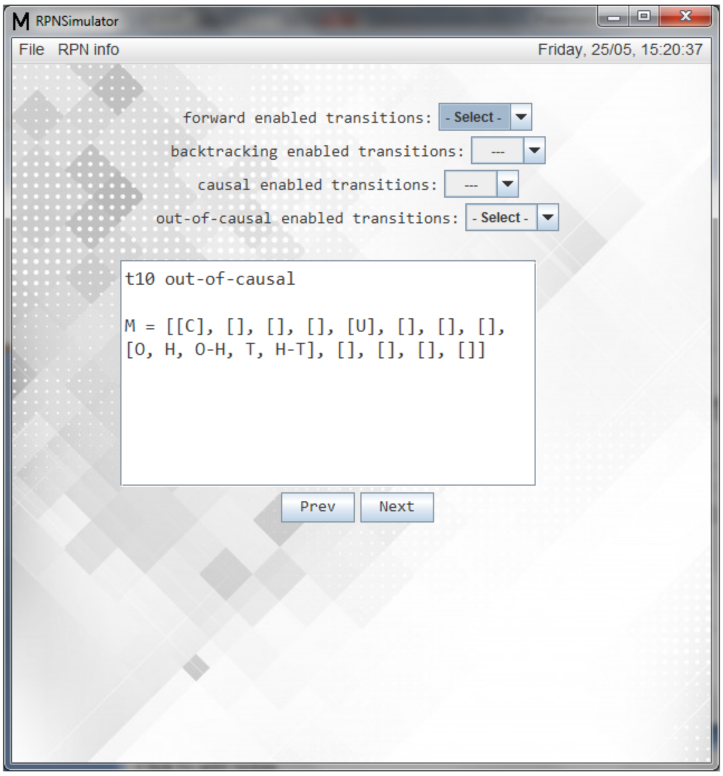


Figure 5.13: Simulator appearance after the out-of-causal-order execution of t_{10}

Graphically the appearance of the Reversing Petri net model will change, since the tokens will change places and new bonds will be created as shown below:

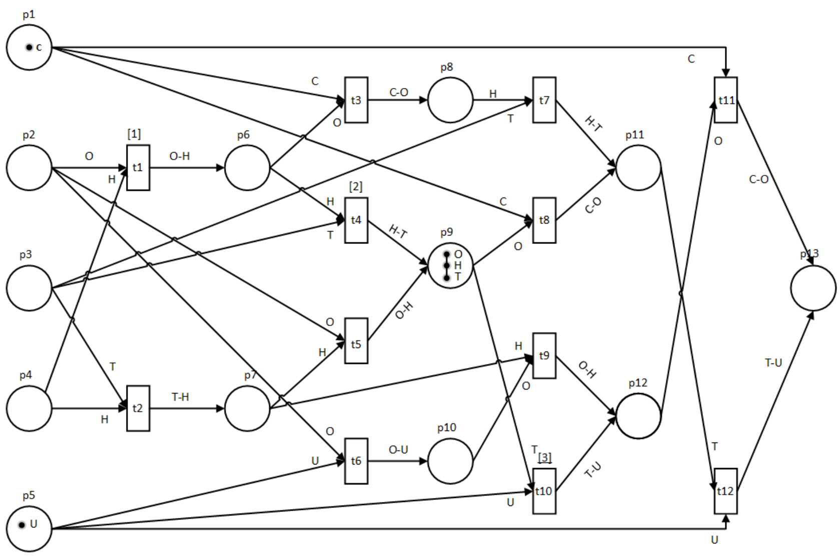


Figure 5.14: Pen's Graphical RPN model after the out-of-causal-order execution of t_{10}

Let us assume that after the out-of-causal-order execution of transition t_{10}, transitions t_8 and t_{12} have been executed in forward, respectively. So the simulator appearance, and the graphical representation of RPN model has been changed as shown in Fig. 5.15 and Fig. 5.16, respectively.

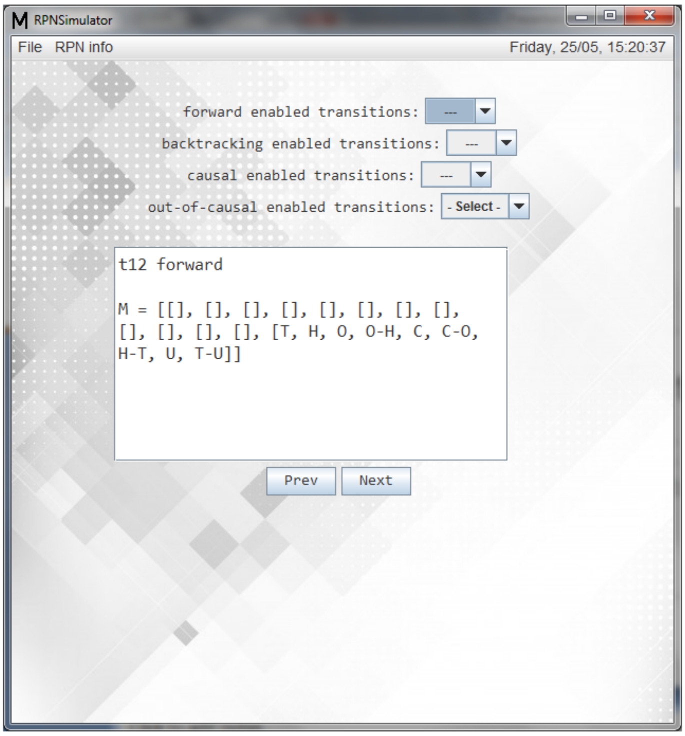


Figure 5.15: Simulator appearance after the forward execution of t_{12}

The current marking matrix M' is:

$$M' = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{C-O, O-H, H-T, T-U\} \end{pmatrix}$$

and the current history matrix H':

$$H' = \begin{pmatrix} 1 & 0 & 0 & 2 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 4 \end{pmatrix}$$

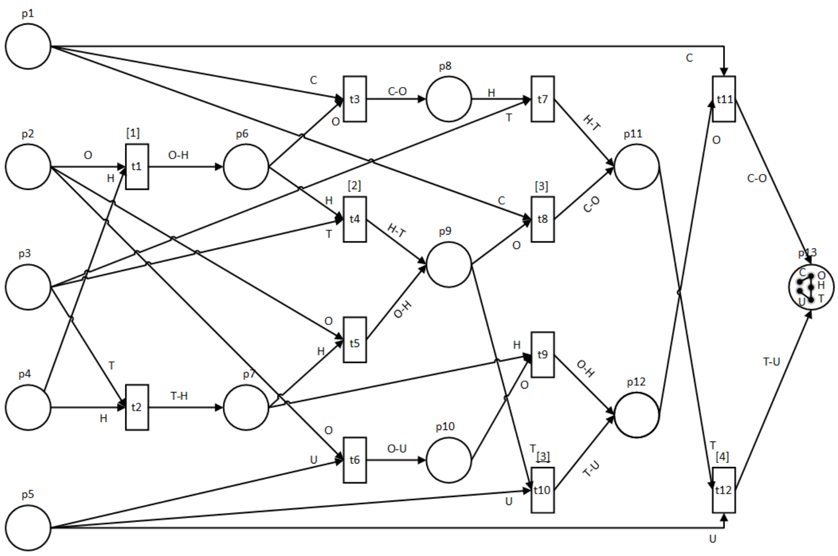


Figure 5.16: Pen Assembly/disassembly RPN after the forward execution of t_{12}

Let us assume that we are going to reverse transition t_8, using **out-of-causal-order execution**.

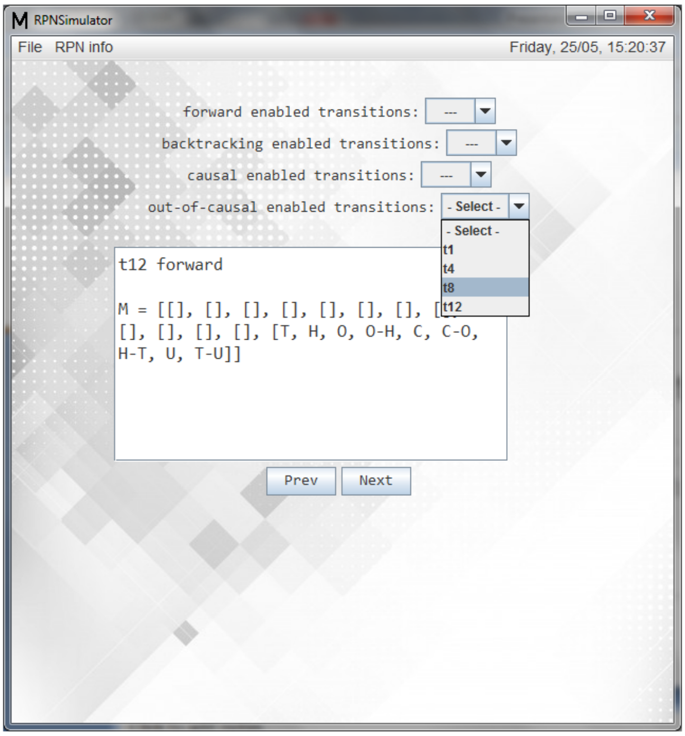


Figure 5.17: The current marking of the Reversing Petri net and the out-of-causal-order enabled transitions

Transition matrix for Fig. 5.16:

$$FT = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

When transition t_8 is reversing, and since the effect of t_8 is the bond $(C-O)$), matrix E has the following values :

$$E = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{C, O-H, H-T, T-U\} \end{pmatrix}$$

To determine the new marking of the RPN after the reversing of the transition specified in the transition matrix, we create the following matrices.

$$L^{+} = \begin{pmatrix} \{C\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{O, H, T, U\} \end{pmatrix}$$

$$L^{-} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{C\} \end{pmatrix}$$

Matrices CL^{+} and CL^{-} contains the connected component of each base of matrices L^{+} and L^{-}, respectively.

$$CL^{+} = \begin{pmatrix} \{C\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{O-H, H-T, T-U\} \end{pmatrix}$$

$$CL^- = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{C\} \end{pmatrix}$$

After the calculation of the above matrices we can calculate the new marking matrix M':

$$\begin{aligned} M' &= E \ominus CL^- \oplus CL^+ \\ &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{C, O-H, H-T, T-U\} \end{pmatrix} \ominus \\ &\quad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{C\} \end{pmatrix} \oplus \\ &\quad \begin{pmatrix} \{C\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{O-H, H-T, T-U\} \end{pmatrix} \\ &= \begin{pmatrix} \{C\} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \{O-H, H-T, T-U\} \end{pmatrix} \end{aligned}$$

and the new history matrix H':

$$\begin{aligned} H' &= H \ominus (\{k \mid k = H(t), t \in T, FT(t) = 1\} \times FT) \\ &= \begin{pmatrix} 1 & 0 & 0 & 2 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 4 \end{pmatrix} \ominus \\ &\quad 3 \times \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \end{pmatrix} \end{aligned}$$

The simulator algorithms will calculate the above matrix equations and as a result the simulator will show to the user the following screen (as shows in Figure 5.18):

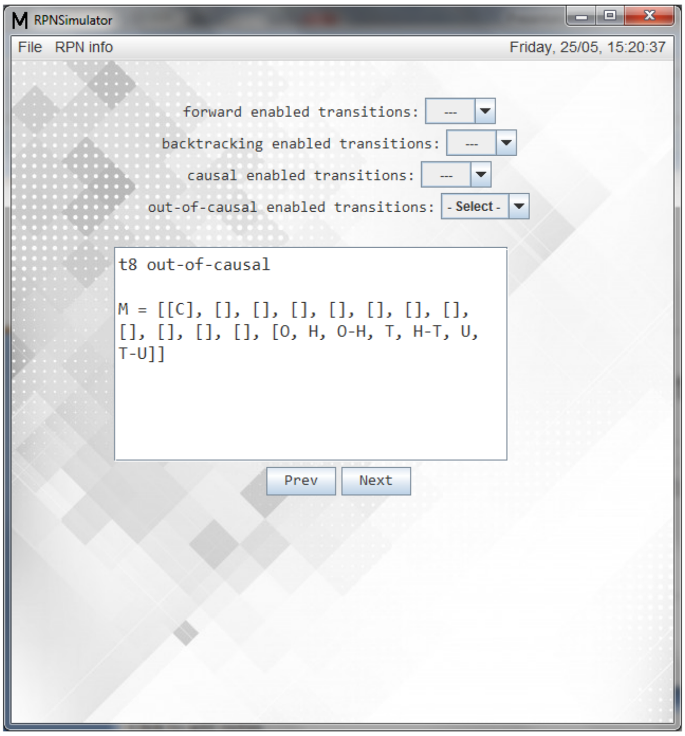


Figure 5.18: Simulator appearance after the out-of-causal-order execution of t_8

Graphically the appearance of the Reversing Petri net model will change, since the tokens will change places and new bonds will be created as shown below:

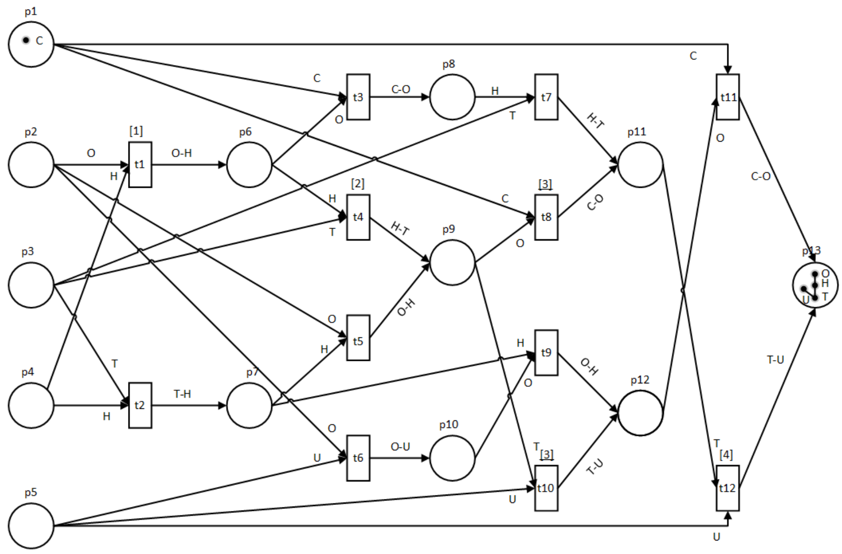


Figure 5.19: Pen Assembly/disassembly RPN after the out-of-causal-order execution of t_8

Chapter 6

Conclusion

Contents

6.1 Summary

In this work we have studied the matrix representation of RPNs which can be used for specifying and manipulating the dynamic behaviour of RPNs as realised by *backtracking*, *causal reversibility* and *out-of-causal-order reversibility*. The developed matrix equations have been used to create simulating algorithms in the Java programming language. This simulator enables the user to give the information of a Reversing Petri net model, and execute one-by-one some transitions in the system. As a result, the simulator displays on the screen the new marking of the model - in the form of a matrix - after performing the specific transitions in that order.

We have noticed, when using Reversing Petri nets to represent the various subtasks that assemble a product we are able to decompose it whilst taking into account the associated pre-conditions and post-conditions, which determine the various feasible sequences. After extensive research and experimentation, we have observed that when using Reversing Petri nets we are able to disassemble a product in all the three different manners of reversibility. So we have used the developed simulator in order to model the automatic generation of assembly and disassembly by delineating the dynamics of the individual tasks, and emphasising a discrete system-oriented approach. During the experimentation stage, various examples have shown that we can indeed use Reversing Petri nets to efficiently simulate assembly and disassembly planning.

6.2 Challenges

Because this diploma thesis was the first major research I did, I encountered several difficulties in all phases of the work. Initially, I had to devote time to understanding basic terms that played a crucial role in this study. There were also difficulties in selecting the appropriate articles to study since many of the articles I met were similar to the subject I was working on, and although they helped me to better understand the basic concepts, they were not ideal for this study.

A particular difficulty I encountered when creating the matrix equations. By looking at various examples and different models of Petri net (e.g. Coloured Petri nets, Time Petri nets etc.), I noticed that for matrix representations of these different models has been used matrices containing 0s and 1s. After various attempts to represent the Reversing Petri nets information by matrices containing 0s and 1s we have instead decided to use sets of bases and bonds. We were led to this decision by the fact that our model information is mainly concerned with the marking of the RPN in the form of tokens and bonds located in each place.

Various difficulties were also encountered when creating the simulator. Initially there was some difficulty when deciding how to represent the structure replicating the matrices in the Java programming language. The choice of the particular structure - ArrayList with Strings - is based on the fact that the representation of each token and bond, as a String would make it much easier to add and subtract elements from a set.

One of the parts that created the greatest concern was the decision whether we should allow the alternation between the three forms of reversibility - i.e. *backtracking*, *causal reversibility*, *out-of-causal-order reversibility* - during the execution of an example. After extensive study, we realized that the *out-of-causal-order reversibility* form contains all three forms of reversibility, and the *causal reversibility* form contains the *backtracking* form. So if we could classify the three forms of reversibility from the smaller to the larger, we would have, *backtracking*, *causal reversibility*, *out-of-causal-order reversibility*. So we concluded that the order of execution can be made from the smallest to the larger set but not the opposite. Thus, once the user selects one of the three forms of reversibility then he can not perform any of the other forms, since they belong to a smaller set.

6.3 Future Work

RPNs are appealing because they can be mechanised very easily and therefore as a future work we could develop a tool that uses computer graphics in order to visually represent the assembly/disassembly process and allow computer manipulation in a quick and intuitive manner. The developed matrix equations can also be used to study the coverability and reachability problems as well as study properties such as boundedness, invariance, conservativeness and liveness. The results in this research can be applied to many other types of applications such as in machining and human operation modelling. Another direction could be to implement a simulation that demonstrates the assembling/disassembling of a product based on the three forms of reversibility. This simulation could compare the various ways of disassembling a product in order to identify the most efficient one depending on the system's needs.

Bibliography

[1] Quantum theory, the church–turing principle and the universal quantum computer. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 400(1818):97–117, 1985.

[2] E. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3):219–253, Apr 1982.

[3] T. Kanehara, T. Suzuki, A. Inaba, and S. Okuma. On algebraic and graph structural properties of assembly petri net - searching by linear programming. In *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 1993, Tokyo, Japan, July 26 - 30, 1993*, pages 2286–2293, 1993.

[4] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191, 1961.

[5] Y. Lecerf. Récursive insolubilité de l'équation générale de diagonalisation de deux monomorphisms de monoïdes libres ϕ x=ψ x. *Comptes rendus de l'Académie des Sciences Paris*, 257:2940–2943.

[6] A. Philippou and K. Psara. Reversible computation in petri nets. *CoRR*, abs/1804.04607, 2018.

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

[8] J. Rosell. Assembly and task planning using petri nets: A survey. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 218(8):987–994, 2004.

[9] T. Toffoli. Reversible computing. In *Automata, Languages and Programming, 7th Colloquium, Proceedings*, pages 632–644, 1980.

Appendix A

Arc structure

```java
import java.util.ArrayList;

public class Arc {
  String from; // the place or transition from where the arc starts
  String to; // the place or transition where the arc ends
  ArrayList<String> with; // the tokens and/or bonds on the specific arc

  /**
  * The constructor of this structure.
  *
  * @param f
  *            the name of the position from where the arc starts
  * @param t
  *            the name of the position where the arc ends
  * @param w
  *            the tokens and bonds which are on the arc
  */
  Arc(String f, String t, String w) {
    String[] split, s;
    from = f;
    to = t;
    with = new ArrayList<String>();
    split = w.split("|");
    if (split[0].equals("{") || split[0].equals("[")) {
      split = w.substring(1, w.length() - 1).split(",");
      for (int i = 0; i < split.length; i++) {
        s = split[i].split("-");
        if (s.length > 1) {
          if (!with.contains(s[0]))
            with.add(s[0]);
          if (!with.contains(s[1]))
            with.add(s[1]);
        }
```

```java
          if (!with.contains(split[i]))
            with.add(split[i]);
        }
      } else {
        s = w.split("-");
        if (s.length > 1) {
          if (!with.contains(s[0]))
            with.add(s[0]);
          if (!with.contains(s[1]))
            with.add(s[1]);
        }
        if (!with.contains(w))
          with.add(w);
      }
    }

  public String toString() {
    String s;
    s = "(" + from + "," + to + ")" + "=" + with;
    return s;
  }

  /**
  * This function checks whether a directed arc is included in
     the given list
  * with arcs.
  *
  * @param list
  *            a list with Arcs
  * @return
  */
  public boolean includedIn(ArrayList<Arc> list) {
    for (int i = 0; i < list.size(); i++) {
      if (list.get(i).from.equals(this.from) && list.get(i).to.
          equals(this.to)
         && (list.get(i).with.contains(this.with.get(0))
         || list.get(i).with.contains(mainbody.rev(this.with.get
            (0)))))
          return true;
    }
    return false;
  }
}
```

Appendix B

Simulator Interface Functions

```
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.util.List;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Calendar;
import javax.imageio.ImageIO;
import javax.swing.Box;
import javax.swing.DefaultListSelectionModel;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextField;
import javax.swing.JTextPane;
import javax.swing.SwingConstants;

public class Interface extends JFrame implements ActionListener
    {
  BufferedImage image;
  MyPanel contentPane = new MyPanel();
  String date = "";
  String day = "";
  String month = "";
  String dat = "";
```

```java
    String clc = "";
    List<String> Mhistory = new ArrayList<String>();
    int mode;

    Interface() {
      BufferedImage image;
      try {
        image = ImageIO.read(getClass().getResource("/matrix.png")
            );
        this.setIconImage(image);
      } catch (IOException e) {
        e.printStackTrace();
      }
      this.setTitle("RPNSimulator");
    }

    /**
    * Set action to each item of the menu bar
    */
    @Override
    public void actionPerformed(ActionEvent evt) {
      String btnLabel = evt.getActionCommand();
      if (btnLabel.equals("Close")) {
        this.getContentPane().removeAll();
        mainbody.tokens.clear();
        mainbody.places.clear();
        mainbody.transitions.clear();
        mainbody.arcs.clear();
        mainbody.Mo.clear();
        mainbody.Dplus.clear();
        mainbody.Dmin.clear();
        mainbody.tokens = new ArrayList<String>();
        mainbody.places = new ArrayList<String>();
        mainbody.transitions = new ArrayList<String>();
        mainbody.arcs = new ArrayList<Arc>();
        mainbody.Mo = new ArrayList<ArrayList<ArrayList<String
            >>>();
        mainbody.M = new ArrayList<ArrayList<ArrayList<String>>>()
            ;
        mainbody.Dplus = new ArrayList<ArrayList<ArrayList<String
            >>>();
        mainbody.Dmin = new ArrayList<ArrayList<ArrayList<String
            >>>();
        Mhistory.clear();
        mainbody.history.clear();
        mainbody.exectrans.clear();
        mainbody.M.clear();
        mainbody.history = new ArrayList<Integer>();

        mainbody.exectrans = new ArrayList<Integer>();
```

```java

      for (int p = 0; p < mainbody.transitions.size(); p++) {
        mainbody.exectrans.add(0);
      }

      for (int t = 0; t < mainbody.transitions.size(); t++) {
        mainbody.history.add(0);
      }
      mode = 0;
      this.getContentPane().add(MENUForm());
      this.revalidate();
      this.repaint();
      this.pack();
    } else if (btnLabel.equals("Restart")) {
      this.getContentPane().removeAll();
      // delete all the extra list with the marking
      Mhistory.clear();
      mainbody.history.clear();
      mainbody.exectrans.clear();
      mainbody.M.clear();
      mainbody.history = new ArrayList<Integer>();
      mainbody.M = new ArrayList<ArrayList<ArrayList<String>>>()
          ;
      mainbody.exectrans = new ArrayList<Integer>();
      mainbody.M.add(new ArrayList<ArrayList<String>>());
      for (int m1 = 0; m1 < mainbody.Mo.get(0).size(); m1++) {
        mainbody.M.get(0).add(new ArrayList<String>());
        for (int m2 = 0; m2 < mainbody.Mo.get(0).get(m1).size();
            m2++) {
          mainbody.M.get(0).get(m1).add(mainbody.Mo.get(0).get(
              m1).get(m2));
        }
      }
      for (int p = 0; p < mainbody.transitions.size(); p++) {
        mainbody.exectrans.add(0);
      }

      for (int t = 0; t < mainbody.transitions.size(); t++) {
        mainbody.history.add(0);
      }
      mode = 0;
      this.getContentPane().add(whileForm("initial n", 0));
      this.revalidate();
      this.repaint();
      this.pack();
    } else if (btnLabel.equals("RPN information")) {
      if (!mainbody.transitions.isEmpty()) {
        this.getContentPane().removeAll();
        this.getContentPane().add(infoForm());
        this.revalidate();
```

```java
        this.repaint();
        this.pack();
      }
    }
  }

  private class MyPanel extends JPanel {
    private BufferedImage image;

    public MyPanel() {
      try {
        image = ImageIO.read(MyPanel.class.getResource("/
            whitegrey.jpg"));
      } catch (IOException ioe) {
        ioe.printStackTrace();
      }
    }

    @Override
    public Dimension getPreferredSize() {
      return image == null ? new Dimension(400, 300) : new
          Dimension(image.getWidth(), image.getHeight());
    }

    @Override
    protected void paintComponent(Graphics g) {
      super.paintComponent(g);
      g.drawImage(image, 0, 0, this);
    }
  }

  /**
  * The initial form of the RPN simulator.
  *
  * @return
  */
  private JPanel MENUForm() {
    JPanel MENU = new JPanel();

    getContentPane().removeAll();
    setJMenuBar(menu());

    JButton button = new JButton("Read from File", new ImageIcon
        (getClass().getResource("/readfilesmall.png")));
    button.setVerticalTextPosition(SwingConstants.BOTTOM);
    button.setHorizontalTextPosition(SwingConstants.CENTER);
    button.setOpaque(false);
    button.setContentAreaFilled(false);
    MENU.add(button);
    button.addActionListener(new ActionListener() {
```

```java
            public void actionPerformed(ActionEvent e) {
                try {
                    getContentPane().removeAll();
                    getContentPane().add(ReadfromFileForm());
                    revalidate();
                    repaint();
                    pack();
                } catch (Exception er) {
                    // Ignore the error and continues
                }
            }
        });

        JButton button2 = new JButton("Read from User", new
            ImageIcon(getClass().getResource("/readusersmall.png")));
        button2.setVerticalTextPosition(SwingConstants.BOTTOM);
        button2.setHorizontalTextPosition(SwingConstants.CENTER);
        button2.setOpaque(false);
        button2.setContentAreaFilled(false);
        MENU.add(button2);
        button2.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    getContentPane().removeAll();
                    getContentPane().add(ReadfromUserForm());
                    revalidate();
                    repaint();
                    pack();
                } catch (Exception er) {
                    // Ignore the error and continues
                }
            }
        });

        revalidate();
        repaint();
        pack();

        MENU.setBounds(60, 90, 500, 500);
        MENU.setOpaque(false);

        return MENU;
    }

    /**
    * The form to read from a file the RPN information.
    *
    * @return
    */
    private JPanel ReadfromFileForm() {
```

```java
    JPanel RFile = new JPanel();

    getContentPane().removeAll();
    setJMenuBar(menu());

    JLabel fn = new JLabel("RPN Filename (e.g. file.txt): ");
    fn.setFont(new Font("Consolas", Font.PLAIN, 14));
    fn.setHorizontalAlignment(SwingConstants.LEFT);
    JTextField filename = new JTextField(20);
    JButton readfile = new JButton("Read..");

    readfile.setFont(new Font("Consolas", Font.PLAIN, 14));
    readfile.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        try {
          String file = filename.getText();
          mainbody.intro(file);

          mainbody.M.add(new ArrayList<ArrayList<String>>());
          for (int m1 = 0; m1 < mainbody.Mo.get(0).size(); m1++) {
            mainbody.M.get(0).add(new ArrayList<String>());
            for (int m2 = 0; m2 < mainbody.Mo.get(0).get(m1).size(); m2++) {
              mainbody.M.get(0).get(m1).add(mainbody.Mo.get(0).get(m1).get(m2));
            }
          }
          for (int p = 0; p < mainbody.transitions.size(); p++) {
            mainbody.exectrans.add(0);
          }

          getContentPane().add(RPNFileForm());
          revalidate();
          repaint();
          pack();
        } catch (Exception er) {
          // Ignore the error and continues
        }
      }
    });

    revalidate();
    repaint();
    pack();

    RFile.add(fn);
    RFile.add(filename);
```

```java
    RFile.add(readfile);
    RFile.setBounds(60, 30, 500, 500);
    RFile.setOpaque(false);

    return RFile;
  }

  /**
  * The form to read from user the RPN information
  *
  * @return
  */
  private JPanel ReadfromUserForm() {
    JPanel RUser = new JPanel();

    getContentPane().removeAll();
    setJMenuBar(menu());

    JLabel tok = new JLabel("RPN Tokens (separated by comma):       ");
    tok.setFont(new Font("Consolas", Font.PLAIN, 14));
    tok.setHorizontalAlignment(SwingConstants.LEFT);
    JTextField tokw = new JTextField(30);

    JLabel pl = new JLabel("RPN Places (separated by comma):       ");
    pl.setFont(new Font("Consolas", Font.PLAIN, 14));
    pl.setHorizontalAlignment(SwingConstants.LEFT);
    JTextField plw = new JTextField(30);

    JLabel tr = new JLabel("RPN Transitions (separated by comma):      ");
    tr.setFont(new Font("Consolas", Font.PLAIN, 14));
    tr.setHorizontalAlignment(SwingConstants.LEFT);
    JTextField trw = new JTextField(30);

    JButton contin = new JButton("Continue");
    contin.setFont(new Font("Consolas", Font.PLAIN, 14));
    contin.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        try {
          String[] s = tokw.getText().split(",");
          for (int i = 0; i < s.length; i++) {
            mainbody.tokens.add(s[i]);
          }

          s = plw.getText().split(",");
          for (int i = 0; i < s.length; i++) {
            mainbody.places.add(s[i]);
          }
```

```java

          s = trw.getText().split(",");
          for (int i = 0; i < s.length; i++) {
            mainbody.transitions.add(s[i]);
          }
          getContentPane().removeAll();
          getContentPane().add(ReadfromUserForm2(tokw.getText(),
              plw.getText(), trw.getText()));
          revalidate();
          repaint();
          pack();
        } catch (Exception er) {
          // Ignore the error and continues
        }
      }
    });

    RUser.add(tok);
    RUser.add(tokw);
    RUser.add(pl);
    RUser.add(plw);
    RUser.add(tr);
    RUser.add(trw);
    RUser.add(contin);

    RUser.setBounds(60, 30, 500, 500);
    RUser.setOpaque(false);

    return RUser;
  }

  /**
  * The second form to read from user the RPN information about
     the directed
  * arcs and the initial marking
  *
  * @param s1
  *            the RPN tokens
  * @param s2
  *            the RPN places
  * @param s3
  *            the RPN transitions
  * @return
  */
  private JPanel ReadfromUserForm2(String s1, String s2, String
     s3) {
    JPanel RUser = new JPanel();
    getContentPane().removeAll();
    JLabel tok = new JLabel("RPN Tokens (separated by comma):
                ");
```

```
tok.setFont(new Font("Consolas", Font.PLAIN, 14));
tok.setHorizontalAlignment(SwingConstants.LEFT);
JTextField tokw = new JTextField(30);
tokw.setText(s1);

JLabel pl = new JLabel("RPN Places (separated by comma):
          ");
pl.setFont(new Font("Consolas", Font.PLAIN, 14));
pl.setHorizontalAlignment(SwingConstants.LEFT);
JTextField plw = new JTextField(30);
plw.setText(s2);

JLabel tr = new JLabel("RPN Transitions (separated by comma)
   :     ");
tr.setFont(new Font("Consolas", Font.PLAIN, 14));
tr.setHorizontalAlignment(SwingConstants.LEFT);
JTextField trw = new JTextField(30);
trw.setText(s3);

JLabel dr = new JLabel("        RPN Directed Arcs:
                      ");
dr.setFont(new Font("Consolas", Font.PLAIN, 14));
dr.setHorizontalAlignment(SwingConstants.LEFT);
JTextField drw = new JTextField(30);
drw.setEditable(false);

JLabel f = new JLabel("                    F(");
f.setFont(new Font("Consolas", Font.PLAIN, 14));
f.setHorizontalAlignment(SwingConstants.LEFT);

JComboBox<String> from = new JComboBox<String>();

for (int t = 0; t < mainbody.transitions.size(); t++) {
  from.addItem("" + mainbody.transitions.get(t));
}
for (int t = 0; t < mainbody.places.size(); t++) {
  from.addItem("" + mainbody.places.get(t));
}
from.setSelectedIndex(0);

JLabel c = new JLabel(",");
c.setFont(new Font("Consolas", Font.PLAIN, 14));
c.setHorizontalAlignment(SwingConstants.LEFT);

JComboBox<String> to = new JComboBox<String>();

for (int t = 0; t < mainbody.places.size(); t++) {
  to.addItem("" + mainbody.places.get(t));
}

```

```java
    for (int t = 0; t < mainbody.transitions.size(); t++) {
      to.addItem("" + mainbody.transitions.get(t));
    }

    to.setSelectedIndex(0);

    JLabel f2 = new JLabel(")= ");
    f2.setFont(new Font("Consolas", Font.PLAIN, 14));
    f2.setHorizontalAlignment(SwingConstants.LEFT);

    String swith = "";

    for (int t = 0; t < mainbody.tokens.size(); t++) {
      swith += mainbody.tokens.get(t) + " ";
    }
    for (int t = 0; t < mainbody.tokens.size(); t++) {
      for (int j = t + 1; j < mainbody.tokens.size(); j++) {
        swith += mainbody.tokens.get(t) + "-" + mainbody.tokens.
            get(j) + " ";
      }
    }

    String[] stwith = swith.split(" ");

    JList<String> with = new JList<String>(stwith);

    with.setVisibleRowCount(2);

    with.setSelectionModel(new DefaultListSelectionModel() {
      @Override
      public void setSelectionInterval(int index0, int index1) {
        if (super.isSelectedIndex(index0)) {
          super.removeSelectionInterval(index0, index1);
        } else {
          super.addSelectionInterval(index0, index1);
        }
      }
    });
    with.setSelectedIndex(0);

    JButton add = new JButton("Add");
    add.setFont(new Font("Consolas", Font.PLAIN, 14));
    add.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        List<String> list = with.getSelectedValuesList();
        String temp = "";
        for (int g = 0; g < list.size(); g++) {
          if (g != 0)
            temp += ",";
          temp += list.get(g);
```

```
        }

        String st = "" + "F(" + from.getSelectedItem() + "," + to.getSelectedItem() + ")={" + temp + "}";

        String s = drw.getText();
        if (s.equals("")) {
          drw.setText(st);
        } else {
          drw.setText(s + "," + st);
        }
        revalidate();
        repaint();
        pack();
        with.clearSelection();
      }
    });

    JButton edit = new JButton("Edit");
    edit.setFont(new Font("Consolas", Font.PLAIN, 14));
    edit.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        drw.setEditable(true);
        revalidate();
        repaint();
        pack();
      }
    });

    JTextPane rpnsem = new JTextPane();
    rpnsem.setPreferredSize(new Dimension(350, 200));

    String msg = "Insert the initial marking for each place (a place can contain tokens or/and bonds, or 0 if it's empty):\n";
    for (int m = 0; m < mainbody.places.size(); m++) {
      msg += "- " + mainbody.places.get(m) + ":\n";
    }

    rpnsem.setText(msg);
    rpnsem.setFont(new Font("Consolas", Font.PLAIN, 17));

    JScrollPane jsp = new JScrollPane(rpnsem);

    JButton contin = new JButton("Continue");
    contin.setFont(new Font("Consolas", Font.PLAIN, 14));
    contin.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        try {
          getContentPane().removeAll();
```

```
    String[] s = drw.getText().split(",F");
    for (int i = 0; i < s.length; i++) {
      if (i == 0) {
        s[i] = s[i].substring(1);
      }
      String[] e1 = s[i].split("=");
      String[] fin = e1[0].substring(1, e1[0].length() - 1).split(",");
      mainbody.arcs.add(new Arc(fin[0], fin[1], e1[1]));
    }

    String[] pt = rpnsem.getText().split(":|\\r?\\n");
    mainbody.Mo.add(new ArrayList<ArrayList<String>>());
    for (int i = 0; i < mainbody.places.size(); i++) {
      mainbody.Mo.get(0).add(new ArrayList<String>());

      String[] si = pt[2 * i + 3].split(",");
      for (int j = 0; j < si.length; j++) {
        if (!si[j].equals("0"))
          mainbody.Mo.get(0).get(i).add(si[j]);
      }
    }
    mainbody.M.add(new ArrayList<ArrayList<String>>());
    for (int m1 = 0; m1 < mainbody.Mo.get(0).size(); m1++) {
      mainbody.M.get(0).add(new ArrayList<String>());
      for (int m2 = 0; m2 < mainbody.Mo.get(0).get(m1).size(); m2++) {
        mainbody.M.get(0).get(m1).add(mainbody.Mo.get(0).get(m1).get(m2));
      }
    }
    for (int p = 0; p < mainbody.transitions.size(); p++) {
      mainbody.exectrans.add(0);
    }

    mainbody.calcDmatrices();
    for (int t = 0; t < mainbody.transitions.size(); t++) {
      mainbody.history.add(0);
    }
    mode = 0;
    getContentPane().add(whileForm("initial n", 0));
    revalidate();
    repaint();
    pack();
  } catch (Exception er) {
    // Ignore the error and continues
```

```java
            }
        }
    });

    JLabel space = new JLabel("                              ");
    JLabel space2 = new JLabel("              ");

    RUser.add(tok);
    RUser.add(tokw);
    RUser.add(pl);
    RUser.add(plw);
    RUser.add(tr);
    RUser.add(trw);
    RUser.add(dr);
    RUser.add(edit);
    RUser.add(drw);
    RUser.add(f);
    RUser.add(from);
    RUser.add(c);
    RUser.add(to);
    RUser.add(f2);
    RUser.add(new JScrollPane(with));
    RUser.add(space);
    RUser.add(add);
    RUser.add(jsp);
    RUser.add(space2);
    RUser.add(contin);

    RUser.setBounds(60, 30, 500, 500);
    RUser.setOpaque(false);

    return RUser;
  }

  /**
  * The form which displays the RPN information
  *
  * @return
  */
  private JPanel infoForm() {
    JPanel RPN = new JPanel();

    JTextPane rpnsem = new JTextPane();
    rpnsem.setEditable(false);
    rpnsem.setPreferredSize(new Dimension(500, 400));

    rpnsem.setText("Tokens: \n" + mainbody.tokens + "\n\nPlaces: \n" + mainbody.places + "\n\nTransitions: \n"
      + mainbody.transitions + "\n\nArcs: \n" + mainbody.arcs + "\n\nInitial Marking: \n"
```

```java
        + mainbody.Mo.get(0));
      rpnsem.setFont(new Font("Consolas", Font.PLAIN, 17));

      JScrollPane jsp = new JScrollPane(rpnsem);

      JButton contin = new JButton("Continue");
      contin.setFont(new Font("Consolas", Font.PLAIN, 14));
      contin.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
          try {
            getContentPane().removeAll();

            if (Mhistory.isEmpty()) {
              mainbody.calcDmatrices();
              for (int t = 0; t < mainbody.transitions.size(); t++) {
                mainbody.history.add(0);
              }
              mode = 0;
              getContentPane().add(whileForm("initial n", 0));
            } else {
              getContentPane().add(whileForm("k r", Mhistory.size() - 1));
            }
            revalidate();
            repaint();
            pack();
          } catch (Exception er) {
            // Ignore the error and continues
          }
        }
      });

      revalidate();
      repaint();
      pack();

      RPN.add(jsp);
      RPN.add(contin);

      RPN.setBounds(60, 100, 500, 500);
      RPN.setOpaque(false);
      return RPN;
    }

    private JPanel RPNFileForm() {
      JPanel RPN = new JPanel();

      JTextPane rpnsem = new JTextPane();
      rpnsem.setEditable(false);
```

```java
    rpnsem.setPreferredSize(new Dimension(500, 400));

    rpnsem.setText("Tokens: \n" + mainbody.tokens + "\n\nPlaces:
        \n" + mainbody.places + "\n\nTransitions: \n"
      + mainbody.transitions + "\n\nArcs: \n" + mainbody.arcs +
         "\n\nInitial Marking: \n"
      + mainbody.Mo.get(0));
    rpnsem.setFont(new Font("Consolas", Font.PLAIN, 17));

    JScrollPane jsp = new JScrollPane(rpnsem);
    JButton contin = new JButton("Continue");
    contin.setFont(new Font("Consolas", Font.PLAIN, 14));
    contin.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        try {
          getContentPane().removeAll();
          mainbody.calcDmatrices();
          for (int t = 0; t < mainbody.transitions.size(); t++)
              {
            mainbody.history.add(0);
          }
          getContentPane().add(whileForm("initial n", 0));
          revalidate();
          repaint();
          pack();
        } catch (Exception er) {
          // Ignore the error and continues
        }
      }
    });
    revalidate();
    repaint();
    pack();

    RPN.add(jsp);
    RPN.add(contin);
    RPN.setBounds(60, 100, 500, 500);
    RPN.setOpaque(false);

    return RPN;
  }

  /**
  * The form which execute the main operation of the simulator.
     In this form
  * the user can choose which transition to execute and see the
     new marking
  * of the model after the execution.
  *
  * @param st
```

```java
 *            the name of transition and the form of execution we will use
 *            (b for backtracking, c for causal, o for out-of-causal)
 * @param pos
 *            the marking that we want to display on screen
 * @return
 */
private JPanel whileForm(String st, int pos) {
  JPanel RFile = new JPanel();
  JLabel space = new JLabel("                    ");
  JLabel spacep = new JLabel(
  "                                                                    ");

  getContentPane().removeAll();

  String[] s = st.split(" ");
  String type = "";

  if (s[1].equals("f")) {
    type = "forward";
  } else if (s[1].equals("b")) {
    type = "backtracking";
  } else if (s[1].equals("c")) {
    type = "causal";
  } else if (s[1].equals("o")) {
    type = "out-of-causal";
  } else {
    type = "marking";
  }

  if (!s[1].equals("n")) {
    for (int t = 0; t < mainbody.transitions.size(); t++) {
      if (t == mainbody.transitions.indexOf(s[0])) {
        if (mainbody.exectrans.get(t).equals(0))
          mainbody.exectrans.set(t, 1);
      } else
        mainbody.exectrans.set(t, 0);
    }

    ArrayList<ArrayList<ArrayList<String>>> tempDP = mainbody.mulMatrix(mainbody.exectrans, mainbody.Dplus);
    ArrayList<ArrayList<ArrayList<String>>> tempDM = mainbody.mulMatrix(mainbody.exectrans, mainbody.Dmin);
    mainbody.effect = new ArrayList<String>();
    for (int i = 0; i < tempDP.size(); i++) {
      for (int j = 0; j < tempDP.get(0).size(); j++)
        mainbody.effect.addAll(tempDP.get(i).get(j));
```

```
    }
    for (int i = 0; i < tempDM.size(); i++) {
      for (int j = 0; j < tempDM.get(0).size(); j++)
        mainbody.effect.removeAll(tempDM.get(i).get(j));
    }

    if (s[1].equals("f")) {
      mainbody.ForwardExec();
    } else if (s[1].equals("b")) {
      mainbody.Backtracking();
    } else if (s[1].equals("c")) {
      mainbody.Backtracking();
    } else if (s[1].equals("o")) {
      mainbody.OutOfCausalExec();
    }
  }

  ArrayList<String> fenabled = new ArrayList<String>();
  fenabled.addAll(mainbody.fenabled());

  ArrayList<String> benabled = new ArrayList<String>();
  if (mode == 1 || mode == 0)
    benabled.addAll(mainbody.benabled());

  ArrayList<String> coenabled = new ArrayList<String>();
  if (mode == 2 || mode == 0)
    coenabled.addAll(mainbody.coenabled());

  ArrayList<String> oenabled = new ArrayList<String>();
  if (mode == 3 || mode == 0)
    oenabled.addAll(mainbody.oenabled());

  if (fenabled.isEmpty() && benabled.isEmpty() && coenabled.isEmpty() && oenabled.isEmpty()) {
    JLabel fn = new JLabel("There are no enabled transitions!");
    fn.setFont(new Font("Consolas", Font.PLAIN, 14));
    RFile.add(fn);
  } else {
    JLabel fn = new JLabel("<html>forward enabled transitions:<br/></html>", SwingConstants.LEFT);
    fn.setFont(new Font("Consolas", Font.PLAIN, 14));
    RFile.add(fn);

    JComboBox<String> forw = new JComboBox<String>();
    if (fenabled.isEmpty()) {
      forw.addItem("      ---      ");
    } else {
      forw.addItem(" - Select - ");
    }
```

```java

    for (int f = 0; f < fenabled.size(); f++) {
      forw.addItem(fenabled.get(f));
    }
    forw.setSelectedIndex(0);
    forw.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        String s = (String) forw.getSelectedItem();

        if (!s.equals(" - Select - ") && !s.equals("      ---      ")) {
          getContentPane().add(whileForm(s + " f", Mhistory.size()));
          revalidate();
          repaint();
          pack();
        }
      }
    });
    RFile.add(forw);

    JLabel bn = new JLabel("<html>backtracking enabled transitions: <br/></html>", SwingConstants.LEFT);
    bn.setFont(new Font("Consolas", Font.PLAIN, 14));
    bn.setHorizontalAlignment(SwingConstants.LEFT);
    RFile.add(bn);

    JComboBox<String> bac = new JComboBox<String>();
    if (benabled.isEmpty()) {
      bac.addItem("      ---      ");
    } else {
      bac.addItem(" - Select - ");
    }

    for (int f = 0; f < benabled.size(); f++) {
      bac.addItem(benabled.get(f));
    }
    bac.setSelectedIndex(0);
    bac.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        String s = (String) bac.getSelectedItem();

        if (!s.equals(" - Select - ") && !s.equals("      ---      ")) {
          mode = 1;
          getContentPane().add(whileForm(s + " b", Mhistory.size()));
          revalidate();
          repaint();
          pack();
```

```
    }
  }
});
RFile.add(bac);

JLabel cn = new JLabel("<html>causal enabled transitions: <br/></html>", SwingConstants.LEFT);
cn.setFont(new Font("Consolas", Font.PLAIN, 14));
cn.setHorizontalAlignment(SwingConstants.LEFT);
RFile.add(cn);

JComboBox<String> cau = new JComboBox<String>();
if (coenabled.isEmpty()) {
  cau.addItem("      ---      ");
} else {
  cau.addItem(" - Select - ");
}

for (int f = 0; f < coenabled.size(); f++) {
  cau.addItem(coenabled.get(f));
}
cau.setSelectedIndex(0);

cau.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    String s = (String) cau.getSelectedItem();

    if (!s.equals(" - Select - ") && !s.equals("      ---      ")) {
      mode = 2;
      getContentPane().add(whileForm(s + " c", Mhistory.size()));
      revalidate();
      repaint();
      pack();
    }
  }
});
RFile.add(cau);

JLabel oocn = new JLabel("<html>out-of-causal enabled transitions: <br/></html>", SwingConstants.LEFT);
oocn.setFont(new Font("Consolas", Font.PLAIN, 14));
oocn.setHorizontalAlignment(SwingConstants.LEFT);
RFile.add(oocn);

JComboBox<String> ooca = new JComboBox<String>();
if (oenabled.isEmpty()) {
  ooca.addItem("      ---      ");
} else {
```

```java
      ooca.addItem(" - Select - ");
    }

    for (int f = 0; f < oenabled.size(); f++) {
      ooca.addItem(oenabled.get(f));
    }
    ooca.setSelectedIndex(0);
    ooca.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        String s = (String) ooca.getSelectedItem();

        if (!s.equals(" - Select - ") && !s.equals("      ---")) {
          mode = 3;
          getContentPane().add(whileForm(s + " o", Mhistory.size()));
          revalidate();
          repaint();
          pack();
        }
      }
    });
    RFile.add(ooca);
  }

  JTextPane rpnsem = new JTextPane();
  rpnsem.setEditable(false);
  rpnsem.setPreferredSize(new Dimension(370, 200));
  String MH = "";
  if (s[1].equals("r")) {
    MH = Mhistory.get(pos);
  } else {
    MH = s[0] + " " + type + "\n\n" + "M = " + mainbody.M.get(0);
    Mhistory.add(MH);
  }
  rpnsem.setText(MH);
  rpnsem.setFont(new Font("Consolas", Font.PLAIN, 17));

  JScrollPane jsp = new JScrollPane(rpnsem);
  JButton previous = new JButton("Prev");
  previous.setFont(new Font("Consolas", Font.PLAIN, 14));
  previous.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      getContentPane().removeAll();
      if (pos == 0) {
        getContentPane().add(whileForm("k r", pos));
      } else {
        getContentPane().add(whileForm("k r", pos - 1));
      }
```

```
                revalidate();
                repaint();
                pack();
            }
        });

        JButton next = new JButton("Next");
        next.setFont(new Font("Consolas", Font.PLAIN, 14));
        next.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                getContentPane().removeAll();
                if (pos == Mhistory.size() - 1) {
                    getContentPane().add(whileForm("k r", pos));
                } else {
                    getContentPane().add(whileForm("k r", pos + 1));
                }
                revalidate();
                repaint();
                pack();
            }
        });

        revalidate();
        repaint();
        pack();

        RFile.add(spacep);
        RFile.add(jsp);
        RFile.add(space);
        RFile.add(previous);
        RFile.add(next);
        RFile.setBounds(60, 30, 500, 500);
        RFile.setOpaque(false);

        return RFile;
    }

    /**
    * The function calculates the date and time of the program.
    *
    * @return
    */
    private String DateNTime() {
        String str = "";
        Calendar rightNow = Calendar.getInstance();
        date = "" + rightNow.getTime();
        String sptab[] = date.split(" ");
        switch (sptab[0]) {
            case "Mon":
                day = "Monday";
```

```
            break;
          case "Tue":
            day = "Tuesday";
            break;
          case "Wed":
            day = "Wednesday";
            break;
          case "Thu":
            day = "Thursday";
            break;
          case "Fri":
            day = "Friday";
            break;
          case "Sat":
            day = "Saturday";
            break;
          case "Sun":
            day = "Sunday";
            break;
        }
        switch (sptab[1]) {
          case "Jan":
            month = "01";
            break;
          case "Feb":
            month = "02";
            break;
          case "Mar":
            month = "03";
            break;
          case "Apr":
            month = "04";
            break;
          case "May":
            month = "05";
            break;
          case "Jun":
            month = "06";
            break;
          case "Jul":
            month = "07";
            break;
          case "Aug":
            month = "08";
            break;
          case "Sep":
            month = "09";
            break;
          case "Oct":
            month = "10";
```

```
        break;
      case "Nov":
        month = "11";
        break;
      case "Dec":
        month = "12";
        break;
    }
    dat = sptab[2];
    clc = sptab[3];

    str += day + ", " + dat + "/" + month + ", " + clc;

    return str;
  }

  /**
  * This function insert to the simulator the menu bar.
  *
  * @return
  */
  private JMenuBar menu() {
    JMenuBar menuBar = new JMenuBar();
    JMenu menu;
    JMenuItem menuItem;

    menu = new JMenu("File");
    menu.setFont(new Font("Arial", Font.PLAIN, 14));
    menuItem = new JMenuItem("Close");
    menuItem.setFont(new Font("Arial", Font.PLAIN, 14));
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Restart");
    menuItem.setFont(new Font("Arial", Font.PLAIN, 14));
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuBar.add(menu);

    menu = new JMenu("RPN info");
    menu.setFont(new Font("Arial", Font.PLAIN, 14));
    menuItem = new JMenuItem("RPN information");
    menuItem.setFont(new Font("Arial", Font.PLAIN, 14));
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuBar.add(menu);

    String str = DateNTime();
    menu = new JMenu(str);
    menu.setFont(new Font("Arial", Font.PLAIN, 14));
    menuBar.add(Box.createHorizontalGlue());
```

```java
    menuBar.add(menu);

    return menuBar;
  }

  public static void main(String[] args) {
    mainbody.history = new ArrayList<Integer>();
    mainbody.tokens = new ArrayList<String>();
    mainbody.places = new ArrayList<String>();
    mainbody.transitions = new ArrayList<String>();
    mainbody.arcs = new ArrayList<Arc>();
    mainbody.Mo = new ArrayList<ArrayList<ArrayList<String>>>();
    mainbody.M = new ArrayList<ArrayList<ArrayList<String>>>();
    mainbody.Dplus = new ArrayList<ArrayList<ArrayList<String
        >>>();
    mainbody.Dmin = new ArrayList<ArrayList<ArrayList<String
        >>>();

    mainbody.exectrans = new ArrayList<Integer>();

    Interface frame = new Interface();
    frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

    frame.contentPane.setLayout(null);

    frame.setLocationByPlatform(true);
    frame.setContentPane(frame.contentPane);
    frame.getContentPane().add(frame.MENUForm());
    frame.pack();
    frame.setVisible(true);
    System.out.println("WELCOME TO RPN Simulator ! \n\n");
  }
}
```

Appendix C

Simulator Operation Functions

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

import org.omg.Messaging.SyncScopeHelper;

public class mainbody {
  static Scanner reader = new Scanner(System.in);

  static ArrayList<Integer> history; // a list with the history
     values of each
  // transition
  static ArrayList<String> transitions; // a list with all the
     transitions of
  // RPN
  static ArrayList<String> places; // a list with all the places
      of RPN
  static ArrayList<String> tokens; // a list with all the tokens
      of RPN
  static ArrayList<Arc> arcs; // a list with all the directed
     arcs of RPN
  static ArrayList<ArrayList<ArrayList<String>>> Mo; // the
     initial marking
  // matrix of RPN
  static ArrayList<ArrayList<ArrayList<String>>> M; // the
     current marking
  // matrix of RPN

  static ArrayList<ArrayList<ArrayList<String>>> Dplus; // the
     matrix with the
  // outgoing arcs of
  // RPN
  static ArrayList<ArrayList<ArrayList<String>>> Dmin; // the
     matrix with the
```

```
// incoming arcs of
// RPN
static ArrayList<ArrayList<ArrayList<String>>> Lplus; // the matrix which
// contains the
// tokens to be
// added to the
// marking
static ArrayList<ArrayList<ArrayList<String>>> Lmin; // the matrix which
// contains the
// tokens to be
// removed from the
// marking

static ArrayList<String> effect; // the effect of the transition we are
// executing

static ArrayList<Integer> exectrans; // a list which contains 1 in the
// position of the transition we are
// executing

/**
* This method is aimed at finding the connected components of a given token
* a in a specific place b.
*
* @param a
*            the name of an element for which we want to find its connected
*            tokens
* @param place
*            a list with the current marking in the place
* @param calcon
*            a list which includes the tokens that have already calculated
* @return a list which contains all the bases and bonds that are directly
*         and indirectly connected to the given token
*/
public static ArrayList<String> con(String a, ArrayList<String> place, ArrayList<String> calcon) {
  String s;
  String[] matrix;
  ArrayList<String> list = new ArrayList<String>();
  matrix = a.split("-");
  if (matrix.length == 1) { // the given element is a base
    for (int i = 0; i < place.size(); i++) {
```

```
        s = place.get(i); // takes the elements of the given place
        // one-by-one
        matrix = s.split("-");
        if (matrix.length == 2) { // if the specific element is a bond
          // if the bond is between the given element and another
          // token then we are calling recursively the connected
          // function on the other token
          if (matrix[0].equals(a)) {
            calcon.add(a);
            if (!calcon.contains(matrix[1])) {
              list.addAll(con(matrix[1], place, calcon));
            }
            if (!list.contains(s)) {
              list.add(s);
            }
          } else if (matrix[1].equals(a)) {
            calcon.add(a);
            if (!calcon.contains(matrix[0])) {
              list.addAll(con(matrix[0], place, calcon));
            }
            if (!list.contains(s)) {
              list.add(s);
            }
          }
        // if the specific element is a base then it directly added
        // in the list
        } else if (s.equals(a) && !list.contains(s)) {
          list.add(s);
          calcon.add(s);
        }
      }
    }
    return list;
  }

  /**
  * This function finds the position of a specific element inside a list.
  *
  * @param x
  *            the element we want to find
  * @param list
  *            the list where we want to search for the specific element
  * @return the number in the list where the given element exist
  */
```

```java
  public static int positionOf(String x, ArrayList<String> list) {
    for (int i = 0; i < list.size(); i++) {
      if (list.get(i).equals(x))
        return i;
    }
    return -1;
  }

  /**
  * This function calculates all the forward enabled transitions of the
  * model.
  *
  * @return a list with all the forward enabled transitions
  */
  public static ArrayList<String> fenabled() {
    boolean bool = true;
    ArrayList<String> fenable = new ArrayList<String>();
    String[] s;

    for (int i = 0; i < transitions.size(); i++) {
    ArrayList<Integer> to = new ArrayList<Integer>();
    ArrayList<Integer> ot = new ArrayList<Integer>();
    bool = true;
    for (int j = 0; j < arcs.size(); j++) {
      if (arcs.get(j).to.equals(transitions.get(i))) {
        ot.add(positionOf(arcs.get(j).from, places));
        for (int t = 0; t < arcs.get(j).with.size(); t++) {
          s = arcs.get(j).with.get(t).split("|");
          if (s[0].equals("!")) {
            if (M.get(0).get(positionOf(arcs.get(j).from, places))
            .contains(arcs.get(j).with.get(t).substring(1))
            || M.get(0).get(positionOf(arcs.get(j).from, places))
            .contains(rev(arcs.get(j).with.get(t).substring(1)))) {
              bool = false;
              break;
            }
          } else {
            if (!M.get(0).get(positionOf(arcs.get(j).from, places)).contains(arcs.get(j).with.get(t))
           && !M.get(0).get(positionOf(arcs.get(j).from, places))
            .contains(rev(arcs.get(j).with.get(t)))) {
              bool = false;
              break;
            }
```

```
            }
          }
        }
        if (arcs.get(j).from.equals(transitions.get(i))) {
          if (!to.contains(j))
            to.add(j);
        }
      }
      if (to.size() > 1)
        for (int m = 0; m < to.size(); m++)
          for (int n = m + 1; n < to.size(); n++)
            for (int k = 0; k < ot.size(); k++) {
              for (int lm = 0; lm < arcs.get(to.get(m)).with.size(); lm++) {
                ArrayList<String> calcon = new ArrayList<String>();
                ArrayList<String> temp = con(arcs.get(to.get(m)).with.get(lm), M.get(0).get(ot.get(k)),
                calcon);
                for (int ln = 0; ln < arcs.get(to.get(n)).with.size(); ln++) {
                  if (temp.contains(arcs.get(to.get(n)).with.get(ln))
                    || temp.contains(rev(arcs.get(to.get(n)).with.get(ln)))) {
                    bool = false;
                    break;
                  }
                }
              }
            }

      for (int n = 0; n < to.size(); n++)
        for (int k = 0; k < ot.size(); k++) {
          for (int w = 0; w < arcs.get(to.get(n)).with.size(); w++) {
            Arc check = new Arc(places.get(ot.get(k)), transitions.get(i), arcs.get(to.get(n)).with.get(w));

            if (M.get(0).get(ot.get(k)).contains(arcs.get(to.get(n)).with.get(w))
              || M.get(0).get(ot.get(k)).contains(rev(arcs.get(to.get(n)).with.get(w))))
              if (!check.includedIn(arcs)) {
                bool = false;
                break;
              }
          }
        }

```

```
    if (bool)
      fenable.add(transitions.get(i));
    }
    return fenable;
  }

  /**
  * This function calculates all the backtracking enabled
     transitions of the
  * model.
  *
  * @return a list with all the backtracking enabled transitions
  */
  public static ArrayList<String> benabled() {
    ArrayList<String> benabled = new ArrayList<String>();

    int max = maxHistory();

    for (int i = 0; i < transitions.size(); i++) {
      if (!history.get(i).equals(0) && (max == i)) {
        benabled.add(transitions.get(i));
      }
    }

    return benabled;
  }

  /**
  * This function calculates all the causal enabled transitions
     of the model.
  *
  * @return a list with all the causal enabled transitions
  */
  public static ArrayList<String> coenabled() {
    ArrayList<String> coenabled = new ArrayList<String>();
    boolean enabled = true;
    for (int i = 0; i < transitions.size(); i++) {
      enabled = true;
      if (!history.get(i).equals(0)) {
        for (int f = 0; f < arcs.size(); f++) {
          if (arcs.get(f).from.equals(transitions.get(i))
            && !M.get(0).get(positionOf(arcs.get(f).to, places))
                .containsAll(arcs.get(f).with)) {
            enabled = false;
          }
        }
        if (enabled)
          coenabled.add(transitions.get(i));
      }
    }
```

```

    return coenabled;
  }

  /**
  * This function reverse the tokens of a bond (e.g. if we have the bond a-b
  * it returns the bond b-a)
  *
  * @param s
  *            the bond we want to reverse
  * @return the reversing bond
  */
  public static String rev(String s) {
    String newS = s;
    String[] sp = s.split("-");
    if (sp.length > 1) {
      newS = sp[1] + "-" + sp[0];
    }
    return newS;
  }

  /**
  * This function calculates all the out-of-causal-order enabled transitions
  * of the model.
  *
  * @return a list with all the out-of-causal-order enabled transitions
  */
  public static ArrayList<String> oenabled() {
    ArrayList<String> oenable = new ArrayList<String>();
    for (int i = 0; i < transitions.size(); i++) {
      if (!history.get(i).equals(0)) {
        oenable.add(transitions.get(i));
      }
    }
    return oenable;
  }

  /**
  * This function reads the RPN information given by a user or from a file.
  *
  * @param filename
  * @throws IOException
  */
  public static void intro(String filename) throws IOException {
    BufferedReader filereader = null;
    boolean usereader = false;
```

```
    if (filename.equals("")) {
      usereader = true;
    } else {
      filereader = new BufferedReader(new FileReader(filename));
    }

    String[] s, e, fin;

    if (usereader) {
      System.out.println("- Enter Petri Net's tokens (separated by a comma) :");
      s = reader.next().split(",");
    } else
      s = filereader.readLine().split(",");

    for (int i = 0; i < s.length; i++) {
      tokens.add(s[i]);
    }

    if (usereader) {
      System.out.println("- Enter Petri Net's places (separated by a comma) :");
      s = reader.next().split(",");
    } else
      s = filereader.readLine().split(",");

    for (int i = 0; i < s.length; i++) {
      places.add(s[i]);
    }

    if (usereader) {
      System.out.println("- Enter Petri Net's transitions (separated by a comma) :");
      s = reader.next().split(",");
    } else
      s = filereader.readLine().split(",");

    for (int i = 0; i < s.length; i++) {
      transitions.add(s[i]);
    }

    if (usereader) {
      System.out.println("- Enter Petri Net's directed arcs (separated by comma) - ex. F(p,t)=a-b :");
      s = reader.next().split(",F");
    } else
      s = filereader.readLine().split(",F");

    for (int i = 0; i < s.length; i++) {
      if (i == 0) {
```

```
        s[i] = s[i].substring(1);
      }
      e = s[i].split("=");
      fin = e[0].substring(1, e[0].length() - 1).split(",");
      arcs.add(new Arc(fin[0], fin[1], e[1]));
    }

    if (usereader)
      System.out.println("- Enter Petri Net's initial marking :");

    Mo.add(new ArrayList<ArrayList<String>>());
    for (int i = 0; i < places.size(); i++) {
      Mo.get(0).add(new ArrayList<String>());

      if (usereader) {
        System.out.print("  --> Tokens at place " + places.get(i) + " : ");
        s = reader.next().split(",");
      } else
        s = filereader.readLine().split(",");

      for (int j = 0; j < s.length; j++) {
        if (!s[j].equals("0"))
          Mo.get(0).get(i).add(s[j]);
      }
    }
    if (!usereader)
      filereader.close();
  }

  /**
  * This method calculates the new table created by the addition of the two
  * matrices
  *
  * @param A
  *            the first matrix which contains sets of tokens and bonds in
  *            each position
  * @param B
  *            the second matrix which contains sets of tokens and bonds in
  *            each position
  * @return the matrix which contains the addition of the two matrices
  */
  public static ArrayList<ArrayList<ArrayList<String>>> addMatrix(ArrayList<ArrayList<ArrayList<String>>> A,
    ArrayList<ArrayList<ArrayList<String>>> B) {
```

```java
    ArrayList<ArrayList<ArrayList<String>>> C = new ArrayList<ArrayList<ArrayList<String>>>();

    C.addAll(A);
    for (int i = 0; i < B.size(); i++) {
      for (int j = 0; j < B.get(i).size(); j++) {
        for (int k = 0; k < B.get(i).get(j).size(); k++) {
          if (!C.get(i).get(j).contains(B.get(i).get(j).get(k)))
            C.get(i).get(j).add(B.get(i).get(j).get(k));
        }
      }
    }
    return C;
  }

  /**
  * This method calculates the new table created by the subtraction of the
  * two matrices
  *
  * @param A
  *            the first matrix which contains sets of tokens and bonds in
  *            each position
  * @param B
  *            the second matrix which contains sets of tokens and bonds in
  *            each position
  * @return the matrix which contains the subtraction of the two matrices
  */
  public static ArrayList<ArrayList<ArrayList<String>>> subMatrix(ArrayList<ArrayList<ArrayList<String>>> A,
    ArrayList<ArrayList<ArrayList<String>>> B) {
    ArrayList<ArrayList<ArrayList<String>>> C = new ArrayList<ArrayList<ArrayList<String>>>();

    C.addAll(A);
    for (int i = 0; i < B.size(); i++) {
      for (int j = 0; j < B.get(i).size(); j++) {
        for (int k = 0; k < B.get(i).get(j).size(); k++) {
          C.get(i).get(j).remove(B.get(i).get(j).get(k));
        }
      }
    }
    return C;
  }

  /**
  * This function finds the place of the RPN that contains the
```

```
       specific
     * bases.
     *
     * @param s
     *            the name of the base we want to check for
     * @return the place on the marking which contain the given
       base
     */
    public static int placeof(String s) {
      for (int i = 0; i < M.get(0).size(); i++) {
        if (M.get(0).get(i).contains(s))
          return i;
        }
      return -1;
    }

    /**
     * This method calculates the new table created by the
       multiplication of the
     * two matrices
     *
     * @param A
     *            the first matrix which contains 0s and 1s in each
        position
     * @param B
     *            the second matrix which contains sets of tokens
       and bonds in
     *            each position
     * @return the matrix which contains the multiplication of the
       two matrices
     */
    public static ArrayList<ArrayList<ArrayList<String>>>
        mulMatrix(ArrayList<Integer> A,
      ArrayList<ArrayList<ArrayList<String>>> B) {
      ArrayList<ArrayList<ArrayList<String>>> C = new ArrayList<
          ArrayList<ArrayList<String>>>();

      C.add(new ArrayList<ArrayList<String>>());
      for (int k = 0; k < B.get(0).size(); k++) {
        C.get(0).add(new ArrayList<String>());
        for (int j = 0; j < A.size(); j++) {
          if (A.get(j).equals(1)) {
            C.get(0).get(k).addAll(B.get(j).get(k));
          }
        }
      }
      return C;
    }

    /**
```

```java
* This function calculates the matrices which contains the information of
* the incoming and outgoing arcs of the RPN model
*/
public static void calcDmatrices() {
  for (int i = 0; i < transitions.size(); i++) {
    Dplus.add(new ArrayList<ArrayList<String>>());
    Dmin.add(new ArrayList<ArrayList<String>>());
    for (int j = 0; j < places.size(); j++) {
      Dplus.get(i).add(new ArrayList<String>());
      Dmin.get(i).add(new ArrayList<String>());
    }
  }

  for (int a = 0; a < arcs.size(); a++) {
    if (transitions.contains(arcs.get(a).from)
      && places.contains(arcs.get(a).to))
      Dplus.get(transitions.indexOf(arcs.get(a).from)).get(places.indexOf(arcs.get(a).to))
      .addAll(arcs.get(a).with);
    else if (places.contains(arcs.get(a).from)
      && transitions.contains(arcs.get(a).to))
      Dmin.get(transitions.indexOf(arcs.get(a).to)).get(places.indexOf(arcs.get(a).from))
      .addAll(arcs.get(a).with);
  }
}

/**
* This function is aimed at finding the connected components of each token
* that exist in a given matrix.
*
* @param matrix
*            the list that will be used for the process of function
* @param eff
*            an integer which takes values -1 or 1 based on if we want to
*            remove the effect of the executed transition or not
* @return
*/
public static ArrayList<ArrayList<ArrayList<String>>> conMatrix(ArrayList<ArrayList<ArrayList<String>>> matrix,
int eff) {

  ArrayList<ArrayList<ArrayList<String>>> cmatrix = new ArrayList<ArrayList<ArrayList<String>>>();
  ArrayList<String> markingplace = new ArrayList<String>();
```

```
    int p = -1;
    for (int i = 0; i < matrix.size(); i++) {
      cmatrix.add(new ArrayList<ArrayList<String>>());
      for (int j = 0; j < matrix.get(0).size(); j++) {
        cmatrix.get(i).add(new ArrayList<String>());
        for (int k = 0; k < matrix.get(i).get(j).size(); k++) {
          if (!cmatrix.get(i).get(j).contains(matrix.get(i).get(j).get(k))
            && tokens.contains(matrix.get(i).get(j).get(k))) {
            p = placeof(matrix.get(i).get(j).get(k));
            markingplace.clear();
            markingplace.addAll(M.get(0).get(p));

            if (eff == -1) {
              markingplace.removeAll(effect);
            }
            cmatrix.get(i).get(j)
            .addAll(con(matrix.get(i).get(j).get(k), markingplace, new ArrayList<String>()));
          }
        }
      }
    }
    return cmatrix;
  }

  /**
  * This function was created for the forward execution of a transition of a
  * RPN model. The function proceeds based on matrix equations.
  */
  public static void ForwardExec() {
    ArrayList<ArrayList<ArrayList<String>>> tdp = mulMatrix(exectrans, Dplus);
    ArrayList<ArrayList<ArrayList<String>>> tdm = mulMatrix(exectrans, Dmin);
    ArrayList<ArrayList<ArrayList<String>>> cdp = conMatrix(tdp, 1);
    ArrayList<ArrayList<ArrayList<String>>> cdm = conMatrix(tdm, 1);
    ArrayList<ArrayList<ArrayList<String>>> mcp = addMatrix(M, cdp);
    ArrayList<ArrayList<ArrayList<String>>> mp = addMatrix(mcp, tdp);
    M = subMatrix(mp, cdm);

    int max = maxHistory();
    for (int h = 0; h < history.size(); h++) {
      history.set(h, (history.get(h) + (history.get(max) + 1) * exectrans.get(h)));
```

```java
    }
  }

  /**
  * This function was created for the backtracking execution of
     a transition
  * of a RPN model. The function proceeds based on matrix
     equations.
  */
  public static void Backtracking() {
    ArrayList<ArrayList<ArrayList<String>>> tdp = mulMatrix(
        exectrans, Dplus);
    ArrayList<ArrayList<ArrayList<String>>> tdm = mulMatrix(
        exectrans, Dmin);
    ArrayList<ArrayList<ArrayList<String>>> cdp = conMatrix(tdp,
        1);
    ArrayList<ArrayList<ArrayList<String>>> cdm = conMatrix(tdm,
        -1);
    ArrayList<ArrayList<ArrayList<String>>> mm = addMatrix(M,
        cdm);
   M = subMatrix(mm, cdp);

    int r = exectrans.indexOf(1);
    for (int h = 0; h < history.size(); h++) {
      history.set(h, (history.get(h) - history.get(r) *
          exectrans.get(h)));
    }
  }

  /**
  * This function finds and returns the maximum value in the
     history matrix
  *
  * @return maximum value in history
  */
  public static int maxHistory() {
    int max = -1;
    int vmax = -1;
    for (int i = 0; i < history.size(); i++) {
      if (history.get(i) > vmax) {
        max = i;
        vmax = history.get(i);
      }
    }
    return max;
  }

  /**
  * This function aims to find the last transition of the model
     that has been
```

```java
* executed, and contains the given set of tokens C on its
    outgoing arc.
*
* @param C
*            a set of tokens
* @return the position of the last transition of the set
*/
public static int last(ArrayList<String> C) {
  int last = -1;
  int hlast = -1;
  for (int t = 0; t < transitions.size(); t++) {
    for (int a = 0; a < arcs.size(); a++) {
      if (!history.get(t).equals(0) && arcs.get(a).from.equals
          (transitions.get(t))) {
        for (int c = 0; c < C.size(); c++) {
          if (arcs.get(a).with.contains(C.get(c)) && (history.
              get(t) > hlast)) {
            last = t;
            hlast = history.get(t);
          }
        }
      }
    }
  }
  return last;
}

/**
* This function puts values in the public lists of the program
    , Lplus and
* Lmin.
*/
public static void calcLmatrices() {
  ArrayList<String> conn = new ArrayList<String>();
  ArrayList<String> markingplace = new ArrayList<String>();

  int pt = -1;
  int lastt;
  Lplus.add(new ArrayList<ArrayList<String>>());
  Lmin.add(new ArrayList<ArrayList<String>>());
  for (int p = 0; p < places.size(); p++) {
    Lplus.get(0).add(new ArrayList<String>());
    Lmin.get(0).add(new ArrayList<String>());
    for (int t = 0; t < tokens.size(); t++) {
      pt = placeof(tokens.get(t));

      markingplace.clear();
      markingplace.addAll(M.get(0).get(pt));
      markingplace.removeAll(effect);

```

```java
          conn = con(tokens.get(t), markingplace, new ArrayList<String>());
          lastt = last(conn);

          if (lastt > -1) {
            for (int a = 0; a < arcs.size(); a++) {
              if (arcs.get(a).from.equals(transitions.get(lastt)) && arcs.get(a).to.equals(places.get(p))) {
                for (int c = 0; c < conn.size(); c++) {
                  if (arcs.get(a).with.contains(conn.get(c))
                    && !Lplus.get(0).get(p).contains(tokens.get(t))) {
                    Lplus.get(0).get(p).add(tokens.get(t));
                  }
                }
              }
            }
          } else if (lastt == -1) {
            if (Mo.get(0).get(p).containsAll(conn)) {
              Lplus.get(0).get(p).add(tokens.get(t));
            }
          }
          if (pt == p) {
            for (int a = 0; a < arcs.size(); a++) {
              if (!Lmin.get(0).get(p).contains(tokens.get(t)) && arcs.get(a).to.equals(places.get(p))
                && (transitions.indexOf(arcs.get(a).from) != lastt)) {
                Lmin.get(0).get(p).add(tokens.get(t));
              }
            }
          }
        }
      }
    }

    /**
    * This function was created for the out-of-causal-order execution of a
    * transition of a RPN model. The function proceeds based on matrix
    * equations.
    */
    public static void OutOfCausalExec() {
      // change history
      int r = exectrans.indexOf(1);
      for (int h = 0; h < history.size(); h++) {
        history.set(h, (history.get(h) - history.get(r) * exectrans.get(h)));
      }
```

```
    Lplus = new ArrayList<ArrayList<ArrayList<String>>>();
    Lmin = new ArrayList<ArrayList<ArrayList<String>>>();
    calcLmatrices();

    // Marking
    ArrayList<ArrayList<ArrayList<String>>> clp = conMatrix(
        Lplus, -1);
    ArrayList<ArrayList<ArrayList<String>>> clm = conMatrix(Lmin
        , -1);

    ArrayList<ArrayList<ArrayList<String>>> E = new ArrayList<
        ArrayList<ArrayList<String>>>();
    E.addAll(M);
    for (int e = 0; e < places.size(); e++) {
      E.get(0).get(e).removeAll(effect);
    }
    ArrayList<ArrayList<ArrayList<String>>> lm = subMatrix(E,
        clm);
    M = addMatrix(lm, clp);
  }

  public static void main(String[] args) throws IOException {
  // TODO Auto-generated method stub

    history = new ArrayList<Integer>();
    tokens = new ArrayList<String>();
    places = new ArrayList<String>();
    transitions = new ArrayList<String>();
    arcs = new ArrayList<Arc>();
    Mo = new ArrayList<ArrayList<ArrayList<String>>>();
    M = new ArrayList<ArrayList<ArrayList<String>>>();
    Dplus = new ArrayList<ArrayList<ArrayList<String>>>();
    Dmin = new ArrayList<ArrayList<ArrayList<String>>>();

    exectrans = new ArrayList<Integer>();

    String file = "";

    System.out.println("Please choose :\nA. Read from User\nB.
        Read from file");
    String r = reader.next();
    if (r.equals("B")) {
      System.out.print("Please give the name of the file : ");
      file = reader.next();
    }
    intro(file);

    M.add(new ArrayList<ArrayList<String>>());
    for (int m1 = 0; m1 < Mo.get(0).size(); m1++) {
```

```
      M.get(0).add(new ArrayList<String>());
      for (int m2 = 0; m2 < Mo.get(0).get(m1).size(); m2++) {
        M.get(0).get(m1).add(Mo.get(0).get(m1).get(m2));
      }
    }

    // initialise executed transition matrix
    for (int p = 0; p < transitions.size(); p++) {
      exectrans.add(0);
    }

    System.out.println("Tokens = " + tokens);
    System.out.println("Places = " + places);
    System.out.println("Transitions = " + transitions);
    System.out.println("Arcs = " + arcs);
    System.out.println("Initial Marking = " + Mo);
    System.out.println();

    calcDmatrices();

    /**********************************
    * REPEATED PART STARTS HERE
    ****************************************/

    // initialise the history matrix
    for (int t = 0; t < transitions.size(); t++) {
      history.add(0);
    }

    while (true) {

      ArrayList<String> fenabled = new ArrayList<String>();
      fenabled.addAll(fenabled());
      ArrayList<String> benabled = new ArrayList<String>();
      benabled.addAll(benabled());
      ArrayList<String> coenabled = new ArrayList<String>();
      coenabled.addAll(coenabled());
      ArrayList<String> oenabled = new ArrayList<String>();
      oenabled.addAll(oenabled());

      if (fenabled.isEmpty() && benabled.isEmpty() && coenabled.
          isEmpty() && oenabled.isEmpty()) {
        System.out.println("No more enable transitions!");
        break;
      }

      String s = "";
      String tran = "";

      do {
```

```
      System.out.println("\nChoose which transition you want to execute from the lists below :");
      System.out.println("- forward enabled : " + fenabled);
      System.out.println("- backtracking enabled : " + benabled);
      System.out.println("- causal enabled : " + coenabled);
      System.out.println("- out-of-causal enabled : " + oenabled);
      System.out.print("\n-> ");
      s = reader.next();
      tran = reader.next();
    } while (!transitions.contains(s) && !fenabled.contains(s) && !benabled.contains(s)
      && !coenabled.contains(s) && !oenabled.contains(s));

    for (int t = 0; t < transitions.size(); t++) {
    if (t == transitions.indexOf(s)) {
      if (exectrans.get(t).equals(0))
        exectrans.set(t, 1);
    } else
      exectrans.set(t, 0);
    }

    // calculate the effect of transition s (assume that only one
    // transition is executed)
    ArrayList<ArrayList<ArrayList<String>>> tempDP = mulMatrix(exectrans, Dplus);
    ArrayList<ArrayList<ArrayList<String>>> tempDM = mulMatrix(exectrans, Dmin);
    effect = new ArrayList<String>();
    for (int i = 0; i < tempDP.size(); i++) {
      for (int j = 0; j < tempDP.get(0).size(); j++)
        effect.addAll(tempDP.get(i).get(j));
    }
    for (int i = 0; i < tempDM.size(); i++) {
      for (int j = 0; j < tempDM.get(0).size(); j++)
        effect.removeAll(tempDM.get(i).get(j));
    }

    if (fenabled.contains(s) && tran.equals("f")) {
      ForwardExec();
    } else if (benabled.contains(s) && tran.equals("b")) {
      Backtracking();
    } else if (coenabled.contains(s) && tran.equals("c")) {
      Backtracking();
    } else if (oenabled.contains(s) && tran.equals("o")) {
      OutOfCausalExec();
    }

```

```
        System.out.println("H = " + history);

        System.out.println("The new marking of the Petri Net is :");
        System.out.println(M.get(0));

        System.out.println("Continue? (y or n)");
        System.out.print("-> ");
        s = reader.next();
        if (s.equals("n"))
          break;
      }
      System.out.println("The End");
    }
}
```