

Thesis Dissertation

**ENCODING REVERSING PETRI NETS IN ANSWER SET  
PROGRAMMING LANGUAGE FOR SIMULATION- BASED  
REASONING**

**Eleftheria Kouppari**

**UNIVERSITY OF CYPRUS**



**DEPARTMENT OF COMPUTER SCIENCE**

**May 2019**

**UNIVERSITY OF CYPRUS**

**DEPARTMENT OF COMPUTER SCIENCE**

**ENCODING REVERSING PETRI NETS IN ANSWER SET  
PROGRAMMING LANGUAGE FOR SIMULATION- BASED REASONING**

**Eleftheria Kouppari**

Supervisors

Dr. Anna Philippou

Dr. Yannis Dimopoulos

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

May 2019

# Acknowledgments

I feel the need to thank the people that supported me and supplied me the assistance needed to complete my diploma thesis. In particular, I would like to express my gratitude to my supervisors Dr. Anna Philippou and Dr. Yannis Dimopoulos who guided and helped me during my research. Their contribution was vital for the success of my diploma thesis, as they offered me all the knowledge needed to overcome predicaments.

I can't dismiss the support provided to me by the Ph.D. student Kyriaki Psara. She was always available to offer further explanations and provide useful material.

I would also like to thank my friends and colleagues who helped me strengthen my self-confidence and made this process even more delightful.

Finally, I want to express my gratitude to my family. Without their help and support, this diploma thesis would not have finished.

# Abstract

The subject of this diploma thesis is the encoding of Reversible Petri Nets in Answer Set Programming language, for simulated-based reasoning. Petri net is a mathematical modelling language that find application in describing and analysing concurrent and distributed systems. It is a very useful tool as it can be applied in any area or system from both practitioners and theoreticians.

Reversible computation in Petri nets reverses the effects of an already executed transition. Reversible Petri nets are very useful for describing applications which naturally embed reversible behavior, like biological processes with bi-directional computation or system reliability by recovering from failures. Additionally, they can support analysis of the properties and problems held by concurrent systems and can be used to study reachability and coverability.

Answer Set Programming is oriented towards difficult search problems, and specifically NP-complete problems. The problem of deciding whether a specific state of the Petri net can be reached requires computing all possible paths that can be expanded from a reversible computation of the Petri Net up to a specific iteration, and it can be an NP-complete problem.

Therefore programs that implement Reversible Petri nets are created in ASP language as a part of this thesis diploma, along with some of the properties of RPNs. To facilitate the use of the programs for users that are not familiar with ASP, a tool is created, that automates some of the procedures.

# Contents

<b>Chapter 1.....</b>	<b>1</b>
<b>Introduction .....</b>	<b>1</b>
1.1 MOTIVATION .....	1
1.2 WORK PURPOSE .....	1
1.3 WORK METHODOLOGY .....	2
1.4 THESIS STRUCTURE .....	2
5.1 THE FINAL CHAPTER OF THIS THESIS,.....	3
<b>Chapter 2.....</b>	<b>4</b>
<b>Related Work.....</b>	<b>4</b>
2.1 REVERSIBLE COMPUTATION .....	4
2.2 REVERSIBLE MODELLING.....	6
2.2.1 Forms of Reversibility .....	7
2.3 PETRI NETS .....	9
2.3.1 Behavioral Properties of Petri nets .....	12
2.4 REVERSING PETRI NETS .....	14
2.5 SEMANTICS .....	17
2.5.1 Forward Execution.....	17
2.5.2 Backtracking .....	19
2.5.3 Causal Reversing .....	20
2.5.4 Out-of-causal-order Reversibility .....	21
2.6 ANSWER SET PROGRAMMING LANGUAGE.....	24
2.6.1 ASP syntax and Semantics .....	25
2.7 CLINGO .....	25

2.8 PETRI NETS IN ASP.....	25
<b>Chapter 3.....</b>	<b>29</b>
<b>Encoding Reversing Petri nets in ASP .....</b>	<b>29</b>
3.1 REVERSING PETRI NET STRUCTURE .....	29
3.2 FORWARD EXECUTION OF REVERSIBLE PETRI NETS .....	33
3.3 BACKTRACKING .....	38
3.4 CAUSAL REVERSING .....	44
3.5 OUT-OF-CAUSAL REVERSIBILITY: .....	45
3.6 PROPERTIES ON REVERSING PETRI NETS.....	49
3.7 SHORTEST PATH.....	52
<b>Tool.....</b>	<b>54</b>
4.1 PETRI NET OBJECT.....	54
4.2 CREATE NEW PETRI NET .....	55
4.3 EXECUTE IN CLINGO .....	56
4.4 INITIAL MARKING .....	56
4.5 FIND NEW STATES .....	57
4.6 PROPERTIES.....	58
<b>Chapter 5.....</b>	<b>60</b>
<b>Case study .....</b>	<b>60</b>
5.2 FORWARD EXECUTION.....	60
5.3 BACKTRACKING .....	63
5.4 CAUSAL REVERSING .....	66
5.5 OUT OF CAUSAL REVERSING .....	67

<b>Chapter 6.....</b>	<b>70</b>
<b>Conclusion.....</b>	<b>70</b>
6.1 SUMMARY .....	70
6.2 CHALLENGES .....	70
6.3 FUTURE WORK.....	71
<b>Appendices .....</b>	<b>76</b>
<b>User's Manual.....</b>	<b>76</b>
1. INITIAL WINDOW .....	76
2. CREATE A NEW PETRI NET .....	76
a. Edit Places.....	77
b. Edit Transitions.....	78
c. Edit Tokens .....	79
d. Edit Arcs .....	79
3. EXECUTE IN CLINGO.....	81
4. INITIAL MARKING.....	83
5. PROPERTIES .....	85
a. Reachability .....	85
b. Home State.....	86
c. Persistence .....	87
d. Transitions' Liveness .....	88
e. Shortest Path .....	89
6. RESULTS WINDOW.....	91
7. PROVE NEW STATES .....	92
<b>ASP Forward execution.....</b>	<b>94</b>
<b>Backtracking.....</b>	<b>99</b>

<b>Causal reversing .....</b>	<b>108</b>
<b>Non causal reversing .....</b>	<b>118</b>



## List of figures

Figure 2.1 Backtracking .....	8
Figure 2.2: Causal reversing .....	9
Figure 2.3: Out-of-Causal reversing .....	9
Figure 2.4: Petri Net .....	11
Figure 2.5 Petri net with transition T1 enabled.....	11
Figure 2.6 Petri Net after T1 fires .....	12
Figure 2.7 Not persistent Petri net.....	14
Figure 2.8 Catalysis example in Petri nets .....	15
Figure 2.9 Backward execution of Petri net occuring after transitions T1, T2, T3 fired with this order.....	20
Figure 2.10 Causal execution of the reversing Petri net occurring after transitions T1, T2, T3 fired with this order.....	21
Figure 2.11 Out-of-Causal reversing in a RPN occurring after t1, t2, t3 fired in this order. ....	23
Figure 2.12 ASP solving process. ....	24
Figure 3. 1 Initial Marking of a petri net.....	32
Figure 3. 2 Results of execution.....	37
Figure 3. 3 Backward execution.....	43
Figure 3. 4 Causal Execution .....	44
Figure 3. 5 Out of Causal Algorithm .....	45
Figure 3. 6 Initial Marking for Catalysis example .....	48
Figure 3. 7 Out-of-causal execution.....	49
Figure 4.1 Initial Window .....	55
Figure 4. 2 Create new RPN screen .....	55
Figure 4. 3 Execute Window.....	56

Figure 4. 4 Properties .....	57
Figure 5. 1 ERK Petri net.....	60
Figure 5. 3 Executing the ERK .....	61
Figure 5. 3 Results first solution, time instance 0.....	61
Figure 5. 4 Results first solution, time instance 1 .....	61
Figure 5. 5 Second solution, time instance 2 .....	62
Figure 5. 7 Second solution, time instance 0 .....	62
Figure 5. 7 Second solution, time instance 1 .....	62
Figure 5. 9 Backtracking execution .....	63
Figure 5. 9 Backtracking results window.....	63
Figure 5. 11 Check if transition c is reachable.....	64
Figure 5. 11 No solutions found when trying to prove transition c reachable. ....	64
Figure 5. 13 Check if transition b is reachable.....	64
Figure 5. 13 Proving that transition b is not reachable .....	64
Figure 5. 15 Solution Backtracking time instance 1. ....	65
Figure 5. 15 Solution of Backtracking time instane 0.....	65
Figure 5. 17 Backtracking solution time instance 2.....	66
Figure 5. 17 Backtracking solution time instance 3.....	66
Figure 5. 19 Causal Execution .....	67
Figure 5. 19 Causal Execution results.....	67
Figure 5. 20 Causal Execution .....	67
Figure 5. 22 Prove new states .....	68
Figure 5. 22 New state .....	68
Figure 5. 24 Check transition p3 for reachability .....	69
Figure 5. 24 Results of the execution.....	69
Figure 5. 25. We can see in Figure 5. 26 that the property is satisfied. ....	68

Figure I Initial window of the tool .....	76
Figure II Create new RPN screen .....	76
Figure III Screen when user will be able to insert a new arc. ....	77
Figure IV One place is added in the place's table.....	77
Figure V Message when trying to enter a place with a name that already exists. ....	77
Figure VII Adding two new places in the table and selecting line 2. ....	78
Figure VII Deleting line 2 from the table.....	78
Figure VIII Transitions Screen with two transitions t1 and t2. t1 is reversible while t2 is not.....	79
Figure IX Token's screen.....	79
Figure X Message when trying to insert an arc with empty tables. ....	79
Figure XI Arc's Screen after adding 6 arcs in the Petri Net. ....	80
Figure XII Label of arc filled with token a and bond b_c.....	80
Figure XIII Execute in Clingo screen .....	81
Figure XIV Open Figure File.....	81
Figure XV Message when trying to insert initial marking without selecting a figure first. ....	82
Figure XVI Initial Marking screen with option FILE ALREADY EXISTS, not selected.....	83
Figure XVII Error Message when trying to insert a token that is already defined in another marking. ....	83
Figure XIX Initial marking screen with option FILE ALREADY EXISTS, selected.....	84
Figure XIX Select a file from folder Initial screen .....	84
Figure XX Properties screen .....	85
Figure XXI Reachability screen.....	85
Figure XXIII Home State when FIND ALL HOME STATES unchecked .....	87

Figure XXIII Home State screen when FIND ALL HOME STATES checked. ....	87
Figure XXIV Persistence screen .....	87
Figure XXV Persistence screen .....	88
Figure XXVII Transition's Liveness screen.....	89
Figure XXVII l2-live checkbox selected .....	89
Figure XXVIII Shortest Path .....	89
Figure XXX Shortest Path screen with Marking selected. ....	90
Figure XXX Shortest Path screen when transition is selected. ....	90
Figure XXXI Results Window.....	91
Figure XXXII Prove new states window .....	92

# Chapter 1

## Introduction

## Contents

---

1.1 MOTIVATION.....	1
1.2 WORK PURPOSE.....	1
1.3 WORK METHODOLOGY.....	2
1.4 THESIS STRUCTURE.....	2

---

### 1.1 Motivation

Reversing Petri nets [8] are very useful for describing applications which naturally embed reversible behavior, like biological processes with bi-directional computation or system reliability by recovering from failures. Additionally, they can support analysis of the properties and problems held by concurrent systems, and can be used to study reachability and coverability.

The problem of deciding whether a specific state of the Petri net can be reached requires computing all possible paths that can be expanded from a reversible computation of the Petri net up to a specific iteration and it can be an NP-complete problem. Answer Set Programming is oriented towards such difficult search problems. Having in mind that it is naturally decidable and nondeterministic, we come to the conclusion that Answer Set Programming can help decide properties of Reversing Petri nets.

### 1.2 Work Purpose

The purpose of this diploma thesis is to implement the mechanisms of Reversing Petri nets in Answer Set Programming and specifically in Clingo. Additionally, some

properties of Reversible Petri Nets are decided. Finally, a tool is developed to facilitate the user experience with the programs implemented in Answer Set Programming.

### **1.3 Work Methodology**

The first step of the implementation of this thesis was to research and understand Reversing Petri nets along with their semantics and definitions. Initially, a study on Petri Nets and Reversible computation separately was necessary in order to fully understand the changes needed in order to make Petri nets reversible.

The second phase was to explore Answer Set Programming and understand the abilities and strengths offered by this computation paradigm. Papers that implemented simple and extended forms of Petri Nets in Answer Set Programming, were extensively studied.

When we had a clear understanding of both Reversing Petri nets and Answer Set Programming, we moved to the coding phase. Changes were made, and attributes were added on the existing forms of Petri nets, that were already implemented so that the programs would support reversibility. The three forms of reversible computation were implemented in the following order backtracking, causal reversibility, and out of causal reversibility.

To extend the research, we explored the properties of Reversible Petri Nets and tried to track the ones that could be useful and achieved by the implemented programs. The remaining time was definitive to decide what properties would be implemented in this phase and the ones that would be left behind for future work.

Finally, a tool was created to facilitate the use of the programs by users that are not familiarize with Answer Set Programming and to display the results in a more understandable and clear way.

### **1.4 Thesis Structure**

In Chapter 2, there is a historical background of Reversible computation, as well as the forms of executing reversible computation. Additionally, Petri nets are explained along with some properties, that will be of interest later on Reversible Petri nets. Consequently, the definition and semantics of Reversible Petri nets are given, and the 4 modes of execution are explained. The chapter ends with the description of Answer

Set Programming and the previous work done that implements the simple form of Petri nets in Answer Set Programming.

Chapter 3 explains the programs that implement Reversible Petri nets and all types of execution. Later on, there is an explanation for the programs that are made for each property. Finally, the chapter explains how the shortest path of reaching a certain goal state is found, using Answer Set Programming.

Chapter 4 is devoted to the tool created and all the options that offers to the user. The chapter is divided based on the main features of the tool, which are explained individually.

Chapter 5, contains a case study made on each mode of execution to prove its correctness and present the effectiveness of the tool.

The final chapter of this thesis, Chapter 6 concludes the work done on this thesis diploma, mentioning the problems faced during the implementation along with the limitations of the work. The chapter closes with a brief reference to the work that can be done in the future based on this thesis.

The thesis also includes a manual of the tool for the user and this can be found in Appendix A.

# Chapter 2

## Related Work

### Contents

---

2.1 REVERSIBLE COMPUTATION .....	4
2.2 REVERSIBLE MODELLING .....	6
2.2.1 Forms of Reversibility .....	7
2.3 PETRI NETS .....	9
2.3.1 Behavioral Properties of Petri nets .....	12
2.4 REVERSIBLE PETRI NETS .....	14
2.5 SEMANTICS .....	17
2.5.1 Forward Execution.....	17
2.5.2 Backtracking .....	19
2.5.3 Causal Reversing .....	20
2.5.4 Out-of-causal-order Reversibility .....	21
2.6 ANSWER SET PROGRAMMING LANGUAGE.....	24
2.6.1 ASP syntax and Semantics .....	25
2.7 CLINGO .....	25
2.8 PETRI NETS IN ASP.....	25

---

### 2.1 Reversible Computation



Reversible computation is an extended form of standard-forward computation, that allows to reverse the execution of an operation, or else, to undo that operation at any time. This allows the system to recover a previous state. Having in mind that all successful physical theories share the property of reversibility, future computing should also follow certain basic facts of fundamental physics, and be effectively used as an interface where logical reversibility implements physical reversibility[9].

Research for Reversible Computation started when Landauer observed that only irreversible operations generate heat by erasing bits of information. Those bits are not physically destroyed though. What actually happens is that they are pushed into the computer's thermal environment resulting in loss of energy. This is known as the von Neumann-Landauer [1] principle which states that the erasure of a bit's leads to at least  $kT \ln 2$  ( $k$  is the Boltzmann constant, and  $T$  is the temperature of the heat sink in kelvins) of physical energy dissipation.

The energy used in reversible operations can be recovered and reused for next executions. Therefore, if an operation is reversible, it will be carried out without bit erasures and thus, without energy loss. After this observation, the existence of thermodynamically reversible computers that dissipate less energy seems plausible.

According to Moore's law, computer power doubles every 18 months [24]. Further decreasing the size of transistors will increase their per-area leakage current and standby power, reduce signal energies, result in thermal fluctuations and progress within the traditional computer paradigm will finally come to an end [25]. Attempts to prevent and reduce these problems resulted in more expensive hardware, where computers using smaller transistors will no longer be cheaper, faster or more energy efficient than they already are and their price will become unreasonable. This is known as Landauer's limit and is threatening to end improvements in practical computer performance within the next few decades and to avoid this a solution could be to avoid losing track of logical information. In order to prevent this, the focus could be given on reversible computation.

Reversible computation is the only possible way to achieve energy and cost efficiency in computers. Lecerf was the first to express reversibility on Turing machines and invented the technique to uncompute histories [4]. Lecerf was oblivious to the Landauer's thermodynamic application and did not store the output of the machines.

Bennett remade Lecerf's machines [5] by taking into consideration Landauer's viewpoint. He created machines that saved any information that would otherwise be erased. Nonetheless, according to Landauer, Bennett's method only delays the unavoidable because the memory is not unlimited and thus, will at some point fill up. Bennett later on achieved to prove that fully reversible Turing machines can be constructed. This machine should erase any garbage information on its tape when it halts and therefore leaving behind only the desired output and input.

Toffoli and Fredkin were the first to construct a practical physical mechanism [6] for computation that was also physically reversible. They constructed reversible logic circuits, Toffoli [7] invented the Toffoli gate which is probably the most used reversible logic gate.

After these developments, reversible computation became a challenging field and it is extensively studied. It can be a solution to the problem of non-stop need for more energy by creating logic gates and circuit that support reversibility. However, there are some limitations that should be considered. An example is the number of computations of an execution. In a reversible computation, arbitrarily large computations executed in reverse would result in almost twice as many steps as an ordinary computation. As a result, a large amount of temporary storage may be needed. A balance between the efficiency of the reverse computation and the speed of computation is needed.

In order to accomplish Reversible Computation on a practical level, it may be necessary to construct new hardware and software to support reversibility. Nevertheless, Reversible Computation requires more attention and research so that it will be effectively implemented, and deal with the problems occurring of today's rapidly advancing technology.

## **2.2 Reversible Modelling**

Reversible computation raised questions that are not yet answered. It is necessary to define the main approaches, results, potential benefits and applications of reversible computation. In order to address these questions, it is useful to apply different notions of reversibility on suitable modeling languages and formulate the theoretical

foundations of what reversibility is, what it pursues and what advantages it provides to systems.

Specifically, exploring reversibility through formal languages will help understand the definitions and semantics of reversibility. It is convenient to apply reversibility on specific case studies and with different notions and strategies so that it is possible to define a unified theory for reversibility. This will additionally help in understanding how reversibility can help in specification, verification, and testing.

Reversible formalisms, once created, can be of use to experts outside Computer Science as well. It will prove a useful setting for studying and analyzing systems but it can also find application in biochemistry, mathematics, and material sciences since there are natural and artificial systems that embed or could use reversibility.

### **2.2.1 Forms of Reversibility**

Reversibility comes in many forms. It's crucial to understand each form of reversibility in order to interpret its role in natural systems and develop realistic formal systems. The two main categories of Reversibility are Uncontrolled and Rigid [9].

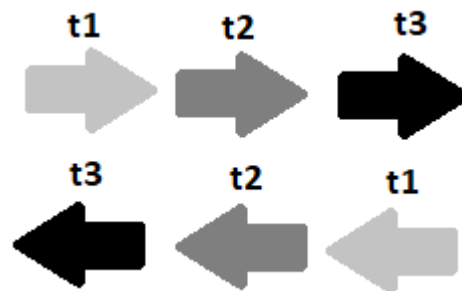
*Uncontrolled* category refers to the form of reversibility that does not specify when and whether reversible computation should be preferred instead of the forward computation. It defines the way an operation should be reversed and thus, helps to understand how reversible computation works.

*Rigid* category refers to the form of reversibility that when a forward step is executed and followed by the corresponding backward step, will result in the system to go back to the starting state, allowing the same computation to begin. This is not useful in cases of reliable systems, specifically when the error that we are trying to recover is permanent, rigid reversibility will result in infinite execution of the same states without overcoming the failure.

Forms of reversibility are categorized by the order in which operations are executed in forward or backward mode, resulting in three categories, backtracking, causal reversibility, and out-of-causal reversibility.

Backtracking is a process where the operations are uncomputed in the exact reverse order in which they were forward executed. In this form of reversibility, only one predecessor state is reachable at any state in the computation, and it ensures that initial computation will not be lost because it prohibits threads to freely backtrack. In concurrent systems, backtracking creates a fake causal dependency on concurrent operations, as they are forced to be undone in only one specific order.

In Figure 2.1 the backtracking execution of tasks t1, t2 and t3 is shown. The tasks were forward executed in the following order: t1, t2 and then t3. Therefore, the only way to reverse these actions is to firstly uncompute task t3, then task t2 and finally task t1 resulting in back to the initial state.



**Figure 2.1 Backtracking**

reversibility, operations can uncompute as soon as all its caused actions are undone. When concurrent operations exist in a system, more than one predecessor state can be reached with causal reversibility.

In the example shown in Figure 2.2: Causal reversing, task t1 and t2 are independent of one another, but t3 is depends on both t1 and t2. Thus, in a forward execution where t2 is executed right after t1 and t3 is consequently executed, 2 paths can be produced with causal reversing. As t1 and t2 are independent, they can be reversed in an arbitrary order, but t3 must always reverse first, because it is an effect of the other two tasks. The two paths produced are called causally equivalent paths.

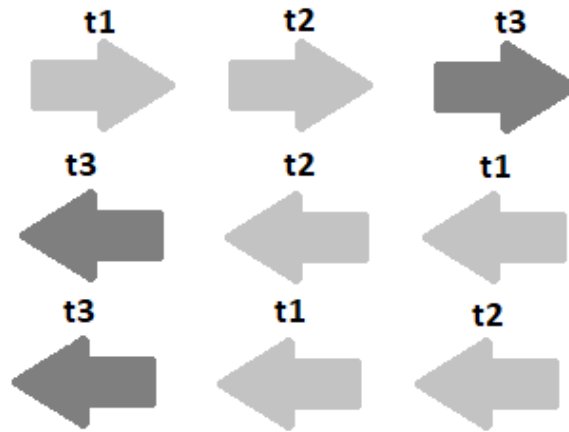


Figure 2.2: Causal reversing

The final form of reversibility is *out-of-causal reversing*. While in the 2 forms of reversibility mentioned above, the order of forward execution is taken into consideration, in out-of-causal reversing, operations can be undone in any order. This form of reversing is very important as it can be found in mechanisms driving long-running transactions and biochemical reactions. Out-of-causal reversibility can give access to unreachable states and it creates new alternatives of current states rather than just going forward and backward on previously visited states.

The Figure 2.3: Out-of-Causal reversing shows that the same execution of tasks t1, t2 and t3 can be undone in 6 paths, 5 more paths than the backtracking reversing and 4 more than the causal reversing.

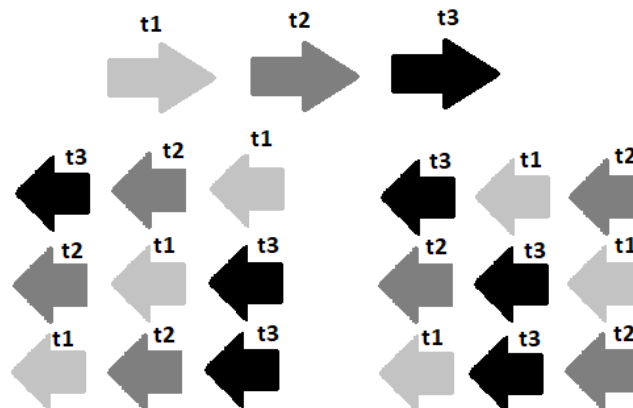


Figure 2.3: Out-of-Causal reversing

## 2.3 Petri nets

Petri nets, invented by Carl Adam Petri in his diploma thesis in 1962 [12], is a graphical and mathematical modelling tool. It is used mainly for describing and

analysing concurrent and distributed systems. It is a very useful tool as it can be applied in any area or system from both practitioners and theoreticians. Some of the areas of application are Business process modelling, concurrent programming, Data analysis, software design and many more.

A Petri net is a collection of nodes and arcs. A node can either be a place or a transition represented graphically by a circle or a rectangle respectively. Places may contain tokens. Tokens represent elements of the system that are subject to dynamic change.

A marking of a Petri net is the number of tokens that are stored in places at a specific state. The tokens that are initially stored in places represent the initial marking of the Petri net. Graphically, tokens are represented by black dots and are placed inside places.

Arcs are weighted and directed and connect a place to a transition or a transition to a place. Their weight signifies the number of tokens that will be moved through the arc, and it is graphically represented as a label above the arc. When the weight of an arc is equal to one the label is usually omitted.

The places of an arc that is directed from the place to transition, are called input places to that transition, while the places of an arc that is directed from the transition to the place, are called output places.

The formal definition of a Petri net is a 4-tuple. Specifically:

$$NP = \langle P, T, F, W, M_0 \rangle \text{ is a net}$$

Where

- $P$  is the finite set of places
- $T$  is the finite set of transitions
- $F$  is the set of directed arcs,  $F \subseteq (P \times T) \cup (T \times P)$
- $W: F \rightarrow \mathbb{N}^+$  a function that assigns weights to the arcs
- $M_0$  is the initial marking

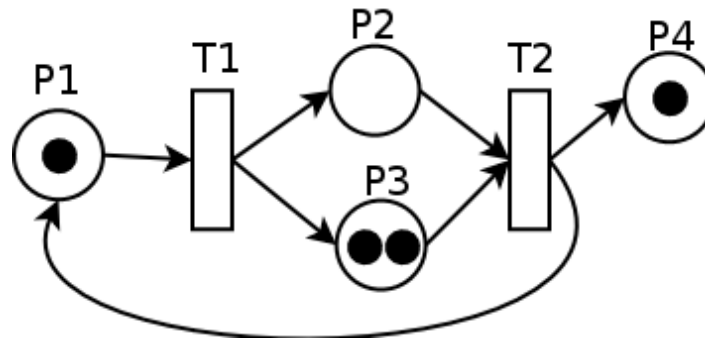


Figure 2.4: Petri Net

The Petri net in Figure 2.4: Petri Net contains four places, namely P1, P2, P3 and P4, and two transitions, namely T1 and T2. The arcs are not labelled, and so the weight is set by default to 1. In the initial marking,  $M_0$  of the Petri net, places P1 and P4 have 1 token, and P3 has 2 tokens. P1 is input place to T1, P2 and P3 are output places to T1 and input places of T2. T2's output place is P4.

A transition in a Petri net is called enabled when all its input places hold at least the amount of tokens specified in the arc's label. Let's consider the example in Figure 2.5 Petri net with transition T1 enabled. Transition T1 is enabled because all its input places contain the required amount of tokens. Place P1 contains one token, the label in arc  $P1 \rightarrow T1$  is absent, and thus only one token is required by the arc. Place P2 contains three tokens which is greater than the amount specified in the arc's  $P2 \rightarrow T1$ 's label.

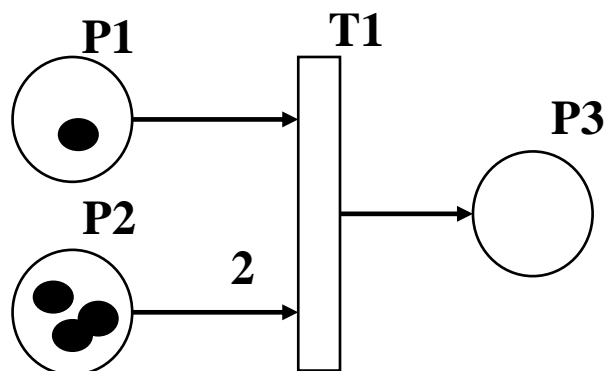


Figure 2.5 Petri net with transition T1 enabled

When a transition is enabled, it may or not fire. Firing of a transition will result in a change in the distribution of the tokens in the Petri net and thus a new marking. The input places of the firing transition will lose as many tokens as needed by the arc. The output places will gain the amount of tokens stated on the output arcs. This is

demonstrated in Figure 2.6 Petri Net after T1 fires, showing the new marking after T1 fires. P1 loses one token, P2 loses two tokens, and P3 gains one token. Thus in the new marking P1 is empty of tokens, while P2 and P3 contain 1 token each.

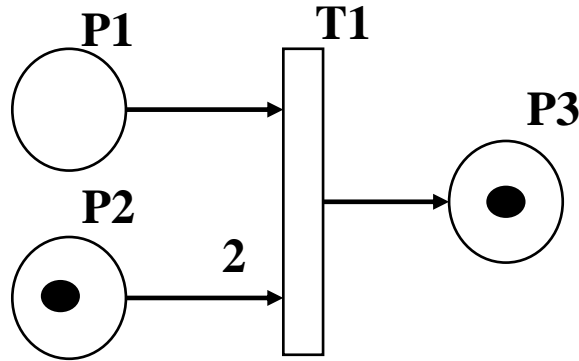


Figure 2.6 Petri Net after T1 fires

A transition without any input place is called source transition and it is unconditionally enabled. A transition without any output place is sink transition and it consumes tokens but does not produce any.

### 2.3.1 Behavioral Properties of Petri nets

Modeling systems with Petri Nets is useful because it helps in the analysis of properties of systems. There are two types of properties that can be found in a Petri net. Behavioral properties, which are depended on the initial marking of the Petri net and Structural Properties, on which initial marking is of no importance. In this diploma thesis emphasis is given on behavioral Properties of Properties. Below, some of the properties that can be decided of reversing Petri nets are defined.

#### ***I. Reachability***

Reachability is the problem of finding whether a marking  $M_n$  is reachable in a Petri net NP, with initial marking  $M_0$ ,  $\langle NP, M_0 \rangle$ . A marking  $M_n$  is said to be reachable in  $\langle NP, M_0 \rangle$  if there exists a sequence of firings,  $\sigma = M_0 T_1 M_1 T_2 M_2 T_3 \dots T_N M_n$  that transforms  $M_0$  to  $M_n$ . The set of all markings that are reachable by  $\langle NP, M_0 \rangle$ , is denoted by  $L(NP, M_0)$ . The problem of reachability is proved to be decidable [16][17] although it takes exponential space and time to verify[18].

#### ***II. Liveness***



The liveness concept refers to the complete absence of deadlocks in an operation system. In other words, when a Petri net is live, then there is no way it will get trapped in a state where no transition can fire. No matter what sequence of firing is selected, there is no path that will lead to trap. Although its an ideal property for many systems, it is very costly to verify. Thus, different levels of liveness are defined in order to relax the liveness condition.

1. *Dead transition:* A transition is called dead if it cannot fire in any firing sequence of  $L\langle NP, M_0 \rangle$ .
2. *L1-live:* A transition is L1-live if it can fire at least once in some firing sequence in  $L\langle NP, M_0 \rangle$ .
3. *L2-live:* A transition Is L2-live if it can fire at least k times in a some firing sequence in  $L\langle NP, M_0 \rangle$ .
4. *L3-live:* A transition Is L3-live if it appears infinitely in some firing sequences in  $L\langle NP, M_0 \rangle$ .
5. *L4-live:* A transition Is L4-live if it is L1-live for every reachable marking in  $\langle NP, M_0 \rangle$ .

Finally, a Petri net is said to be Lk-live if all of its transitions are L4-live. It is clear that when a transition is L4-live then its also L3-live, L2-live for some integer k and L1-live.

### ***III. Reversibility and Home State***

A Petri net is said to be reversible if for every reachable marking M in  $\langle R, M_0 \rangle$ , it can go back to the initial marking  $M_0$ . Not all applications are interested in going back to the initial state and therefore Home State is defined. Home state is a marking that is reachable from any other marking in the Petri net.

### ***IV. Persistence***

A Petri net is said to be persistent, if for any couple of enabled transitions, the firing of one will not disable the other. In a persistent Petri net, once a transition is enabled it will stay enabled until it fires. The Petri net in Figure 2.7 Not persistent Petri netis not persistent because in the initial marking both transitions are enabled, but if one fires the

remaining tokens in place P2 will not be enough for the second to fire as well. The Petri net in Figure 2.5 Petri net with transition T1 enabled is persistent.

Studying the properties of a system can be very useful and this can be easily done with the use of Petri nets.

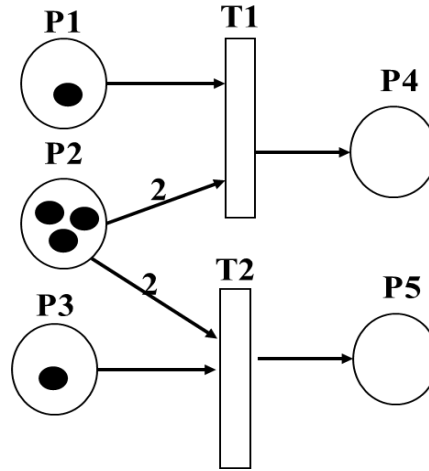


Figure 2.7 Not persistent Petri net

## 2.4 Reversing Petri nets

Reversible computation on Petri nets intends to reverse the effects of an already executed transition.

In 2016 articles [13][14] proposed to study reversible computation within Petri nets. The authors investigated the effect of adding reversed versions of selected transitions in a Petri net. These transitions were produced by reversing the direction of the arcs of the original transitions. Researches also explored Reachability and coverability. Another research done later on, covered the subject causality in Petri nets [20][21][22].

In order for Petri nets to support reversibility, some adaptations and restrictions are needed. Firstly, the tokens that are present in the Petri net need to be persistent. Secondly, the Reversing Petri nets considered in this diploma thesis are acyclic, although Petri nets with cycles are already proved to support reversibility.

In order for a forward-only Petri net to reverse a transition, it is necessary to insert a transition in the structure of the Petri net that undoes the effect of the transition. Consider the example in Figure 2.8 Catalysis example in Petri nets(a) where the process of catalysis is represented by a regular Petri net. Transition  $t_1$  is present just to undo the effect of the already executed transition  $t_1$ . The same process is also represented by

Figure 2.8 Catalysis example in Petri nets **Error! Reference source not found.** (b) by a reversible Petri net. Supporting reversibility by explicitly adding transitions in the Petri net's structure can result in large and more complex systems. Further than that, it does not embody reversible computation in the Petri net.

Yet, the Reversing Petri nets, supported by this dissertation thesis, are only valid when tokens are individual, although this restriction is currently lifted. That means, that only one token of a certain type can be present in a reversing Petri net. Another point where focus should be given is distinguishing the causal path of each transition. To achieve this, two new notions are introduced, namely *base* and *bonds*. A base is a persistent token that cannot be consumed. A base protects token's individuality through various transitions. A bond, is a connection between tokens created by transitions. The reverse of a transition will break the bond, and thus the effect of the transition.

An additional decision that was necessary is the option to a transition to take as input a subset of bonds and bases available. This will ensure that only the ones needed are selected and the rest are available for any other transition that may require them. Therefore, we introduce *negative* bonds/bases. When an arc is labeled with negative

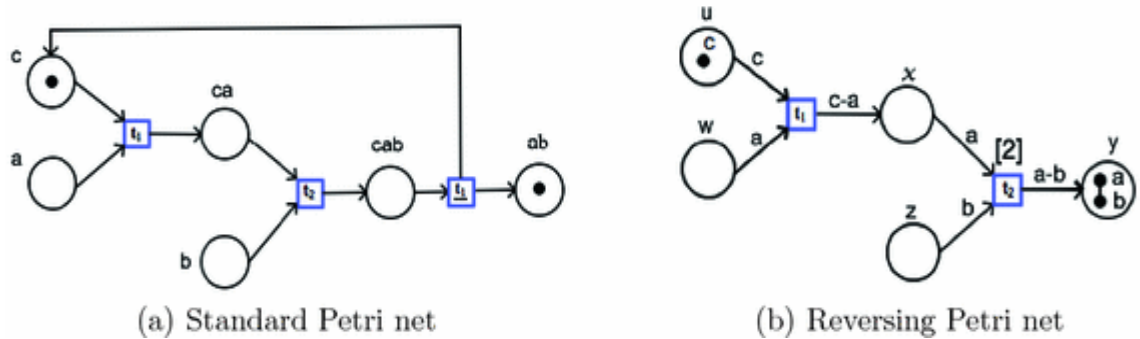


Figure 2.8 Catalysis example in Petri nets

token  $\overline{a}$  it expresses that token  $a$  should not be present in the incoming places of  $t$  for the transition to fire and similarly for bond  $\overline{b}$  bond/base.

The formal definition of a Reversing Petri net is a 5-tuple  $(A, P, B, T, F)$  where:

1.  $A$  is a finite set of bases or tokens ranged over by  $a, b, \dots$ .  $\overline{A} = \{ \overline{a} \mid a \in A \}$  where  $\overline{A}$  contains negative instance for each token and we write  $\mathcal{A} = A \cup \overline{A}$ .
2.  $P$  is a finite set of places.
3.  $B \subseteq A \times A$  is a set of bonds ranged over by  $\beta, \gamma, \dots$ . We use the notation  $a-b$  for a bond  $(a, b) \in B$ .  $\overline{B} = \{ \overline{b} \mid b \in B \}$  contains negative instances of the bonds. We write  $\mathcal{B} = B \cup \overline{B}$ .
4.  $T$  is a finite transitions.
5.  $F: (P \times T \cup T \times P) \rightarrow 2^{\mathcal{A} \cup \mathcal{B}}$  is a set of directed arcs.

Places and transitions are the same as in standard Petri nets. Tokens are distinguished from each other as they have a unique name. Bases may occur as stand-alone elements or they may merge together to form bonds. Arcs are, as in standard Petri nets, directed and connected places to transitions or transitions to places. They are labelled by a subset of  $\mathcal{A} \cup \mathcal{B}$ . Negative bases express the absence of the base. For a label  $\ell = F(x, t)$  or  $\ell = F(t, x)$ , any token  $a$  may appear in a label only once either as  $a$  or as  $\overline{a}$ , and if a bond  $(a, b) \in \ell$ , then  $a, b \in \ell$ . Additionally for  $\ell = F(t, x)$ , it must be that  $\ell \cap (A \cup B) = \emptyset$ , that is, negative tokens/bonds may only occur on arcs incoming to a transition.

Furthermore, we prohibit a  $\text{pre}(t) = \{p \in P \mid F(p, t) \neq \emptyset\}$  and  $\text{post}(t) = \{p \in P \mid F(t, p) \neq \emptyset\}$ . Finally, we write  $\text{pre}(t) = \bigcup_{p \in P} F(p, t)$  for the unions of all labels of the incoming arcs of transition  $t$  as well as  $\text{post}(t) = \bigcup_{p \in P} F(t, p)$  for the union of all labels of the outgoing arcs of transition  $t$ .

For a reversing Petri net to be well formed the following should hold for all transitions:

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

The above ascertains that no transition erases tokens (1), no transition destroys bonds (2), tokens and bonds are not cloned into more than one outgoing places(3).

A marking is the same as in standard Petri nets but with the addition of bonds.

History is needed in order to distinguish the cause of the state among other alternatives. History assigns a memory to each transition,  $H:T \rightarrow \epsilon \cup \mathbb{N}$ . An  $\epsilon$  history stands for a transition that has not yet taken place, while history  $n \in \mathbb{N}$  captures that the transition has taken place after  $n-1$  transitions and has not yet been reversed. The initial history for all transitions is set to  $\epsilon$ ,  $H_0 = \epsilon$ .

A pair of marking and history  $\langle M, H \rangle$  describe the state of a Petri net.

We define  $\text{con}(a, C)$  where  $a$  is a base and  $C$  a set of connections, to be the tokens connected to  $a$  via bonds as well as the bonds creating these connections according to set  $C$ . This is necessary because more than one connected components may arise in a place:

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

where  $\text{path}(a, w, C)$  if  $w = \langle \beta_1, \dots, \beta_n \rangle$ , 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  $\text{con}(S, C)$ , where  $S \subseteq A$ , for  $\bigcup_{a \in S} \text{con}(a, C)$ .

Graphically, tokens are represented by  $\bullet$ , bonds by a line between the tokens, places by circles and transitions by rectangles. Histories are represented as  $[m]$ , where  $m = H(t)$  for a transition  $t$ .

## 2.5 Semantics

Now that the definitions of the structure of reversible Petri nets are clear, we may define the executions within the reversing Petri nets.

### 2.5.1 Forward Execution

**Definition 1:** Given a reversing Petri net  $\text{RPN} (A, P, B, T, F)$ , a transition  $t \in T$ , and a state  $\langle M, H_i \rangle$ .

We say that  $t$  is forward enabled in  $\langle M, H_i \rangle$  if the following hold:

1. if  $\alpha \in F(x, t)$  for some  $x \in {}^\circ t$ , then  $\alpha \in M(x)$ , and if  $\overline{\alpha} \in F(x, t)$  then  $\alpha \notin M(x)$   
where  $\alpha$  is a token,

2. if  $\beta \in F(x, t)$  for some  $x \in {}^\circ t$ , then  $a \in M(x)$ , and if  $\overline{\beta} \in F(x, t)$  then  $\beta \notin M(x)$

where  $\beta$  is a bond,

3. if  $\alpha \in F(t, y_1)$ ,  $b \in F(t, y_2)$ ,  $y_1 \neq y_2$  then  $\beta \notin \text{con}(\alpha, M(x))$  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  $\beta \in F(y, t)$ .

From rule (1), (2), a transition is enabled if all tokens and bonds required from the incoming arcs of the transition are present in the incoming places, and all tokens and bonds that are required to be absent by the incoming arcs will not be present in the incoming places. Rule (3) states that no tokens are duplicated in forks transitions. Finally, rule (4), states that if a pre-existing bond appears in an outgoing arc of a transition, then it is also a precondition of the transition to fire, but if the bond appears in an outgoing arc of a transition but is not a requirement for the transition to fire, then the bond should not be present in an in-place of the transition.

The effect of a transition is defined as

$$\text{Eff}(t) = \text{post}(t) - \text{pre}(t)$$

and it is the set of new bonds created by the transition.

**Definition 2:** Given a reversing Petri net  $(A, P, B, T, F)$  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

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

And

$$H'(t) = \begin{cases} \text{Max}\{k | k = H(t''), t'' \in T\} + 1, & \text{if } t' = t \\ H'(t), & \text{otherwise} \end{cases}$$

which means, when a transition is executed, all tokens and bonds that were present in its incoming arcs are relocated from the input places to the output places along with their connected components. The history of the executed transition will get the next available integer key, stating that it was the last executed transition.

In [8] it is proved that transitions neither erase nor clone bases.

### 2.5.2 Backtracking

The first form of reversibility is considered in this section.

**Definition 3:** Consider a reversing Petri Net  $N = (A, P, B, T, F)$ , a state  $\langle M, Hi \rangle$  and a transition  $t \in T$ . We say that  $t$  is bt-enabled in  $\langle M, Hi \rangle$  if

$$H(t) = k \in \mathbb{N} \text{ with } k \geq k' \text{ for all } k' \in \mathbb{N}, k' = H(t') \text{ and } t' \in T.$$

Thus, a transition in Petri net is bt-enabled if it is the last transition to have been executed, as it will have the biggest history.

**Definition 4:** Again consider a reversing Petri net as state  $\langle M, H \rangle$  and a transition  $t$  bt-enabled in  $\langle M, H \rangle$ , we write  $\langle M, H \rangle \xrightarrow{u} \langle M', H' \rangle$  where,

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

and

$$H'(t) = \begin{cases} \varepsilon, & \text{if } t' = t \\ H(t), & \text{otherwise} \end{cases}$$

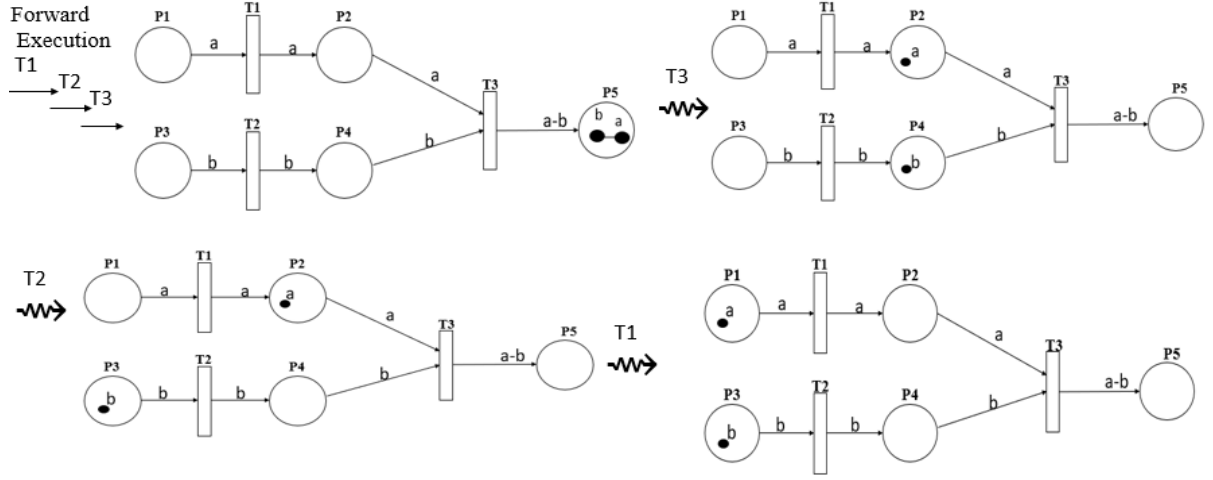
Thus, when a transition  $t$  is reversed in a backtracking mode all tokens and bonds in the postcondition of the transition along with their connected components, will be transferred to the incoming places of the transition and any newly created bonds will be broken. At the same time, the history is set to  $\varepsilon$  to indicate that the transition has been reversed.

Again, [8] proves that backtracking mode of reversibility, neither erases nor clones tokens, and exactly one instance of each base exists and transitions. Moreover, it proves that:

1. Tokens are preserved throughout the execution of the reversible Petri net,
2. Bonds can be created in the forward execution,
3. Destructed during the backward execution, and

#### 4. Preserved through actions that do not operate directly on the bond.

The example of Figure 2.9 Backward execution of Petri net occurring after transitions  $T_1$ ,  $T_2$ ,  $T_3$  fired with this order. demonstrates the Backward reversing of the Petri net that occurred after  $T_1$  fired followed by  $T_2$  and finally  $T_3$ . As shows, in the Petri net's marking, only  $P_3$  contains a base, and it is a bond between token  $a$  and  $b$ . When  $T_3$  is



**Figure 2.9** Backward execution of Petri net occurring after transitions  $T_1$ ,  $T_2$ ,  $T_3$  fired with this order.

reversed, the bond breaks, and  $a$ ,  $b$  tokens go back to places  $P_2$ ,  $P_4$ , respectively. Next,  $T_2$  will be reversed because it is the last transition to be executed and not reversed. Token  $b$  is transferred from place  $P_4$  back to its initial place,  $P_3$ .  $T_1$  is lastly reversed and token  $a$  is moved to its initial place.

### 2.5.3 Causal Reversing

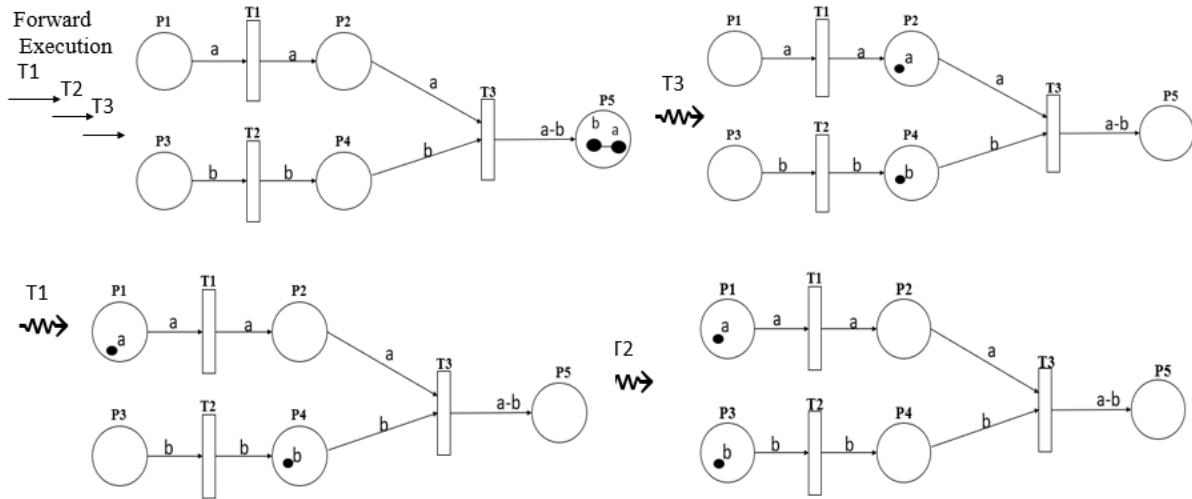
The next form of reversibility is Causal reversing. A transition  $t$  is considered co-enabled (causal enabled), when all transitions that are causally dependent on  $t$  are reversed or have not been executed yet.

**Definition 5:** Given a reversible Petri net  $(A, P, B, T, F)$ , a state  $\langle M, H \rangle$ , and a transition  $t \in T$ . Then  $t$  is co-enabled in  $\langle M, H \rangle$ , if  $H(t) \in \mathbb{N}$  and, for all  $a \in F(t, x)$ , if  $a \in M(y)$  for some  $y$  and  $\text{con}(a, M(y)) \cap \text{pre}(t') \neq \emptyset$  for some  $t'$  then either  $H(t') = \varepsilon$  or  $H(t') \leq H(t)$ .



The above definition, is equivalent [8]**Error! Reference source not found.** with the following:

Given a reversible Petri net  $(A, P, B, T, F)$ , a state  $\langle M, H \rangle$ , and a transition  $t \in T$ ,  $t$  is co-enabled in  $\langle M, H \rangle$ , if and only if  $H(t) \in \mathbb{N}$  and for all  $a, \beta \in F(t, x)$  we have  $a, \beta \in M(x)$ .



**Figure 2.10** Causal execution of the reversing Petri net occurring after transitions  $T_1$ ,  $T_2$ ,  $T_3$  fired with this order.

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

As show in Figure 2.10 Causal execution of the reversing Petri net occurring after transitions  $T_1$ ,  $T_2$ ,  $T_3$  fired with this order.in causal mode of reversing, the transitions  $T_1$  and  $T_2$  can be reversed in any order as they are not causally dependent on one another. In the example  $T_2$  is reversed after  $T_1$ , although  $T_1$  precedes in the forward execution.

#### 2.5.4 Out-of-causal-order Reversibility

The final form of reversibility is Out-of-Causal reversibility. In out-of-causal reversibility, executed transitions can be reversed at any time.

**Definition 6:** Given a reversing Petri net  $(A, P, B, T, F)$ , 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 effect of reversing a transition is that all bonds created by the transition are undone. This may result in tokens backtracking in the net. In particular, if the reversal of a transition causes the destruction of a bond which results in a coalition of bonds to be broken down into a set of subcomponents, then each of these coalitions should flow back, as far as possible, to the last transition in which this sub-coalition participated.

**Definition 7:** Given a reversing Petri net  $(A, P, B, T, F)$ , a history  $H$ , and a set of bases and bonds  $C$ :

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

The definition is used to capture the notion as far as possible. It states that if component  $C$  has been manipulated by some previously executed transition, then  $\text{last}(C, H)$  is the last executed such transition. Otherwise, if the component was not used by an unreversed transition, then  $\text{last}(C, H)$  is undefined.

Reversing a transition in an out-of-causal mode is defined below:

**Definition 8:** Given a reversing Petri net  $(A, P, B, T, F)$ , 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  $\langle M, H \rangle \xrightarrow{t} \langle M', H' \rangle$  where  $H'$  is defined as in Definition 3: and we have:

$$M'(x) = M(x) - \text{eff}(t) - \{ C_{a,x} \mid \exists a \in M(x), x \in t' \circ, t' \neq \text{last}(C_{a,x}, H') \}$$

$$\cup \{ C_{a,y} \mid \exists a, y, a \in M(x), \text{last}(C_{a,y}, H') = t', F(t', x) \cap C_{a,y} \neq \emptyset, \}$$

$$\cup \{ C_{a,y} \mid \exists a, y, a \in M(x), \text{last}(C_{a,y}, H') = \perp, C_{a,y} \subseteq M_0(x) \}$$

Where  $C_{b,z} = \text{con}(b, M(z) - \text{eff}(t))$  for  $b \in A, z \in P$ .

As shown by the definitions, when a transition  $t$  is reversed in an out-of-causal-order fashion, all bonds that are created by the transition are undone. The components that arise after the destruction of the bond, should be relocated back to the place where they were last used.

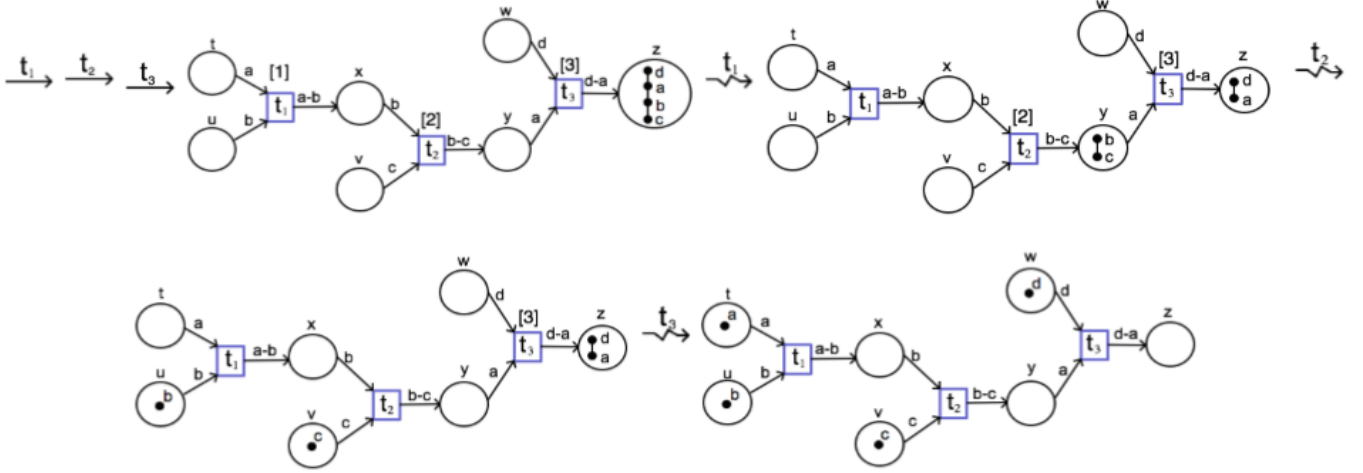


Figure 2.11 Out-of-Causal reversing in a RPN occurring after  $t_1$ ,  $t_2$ ,  $t_3$  fired in this order.

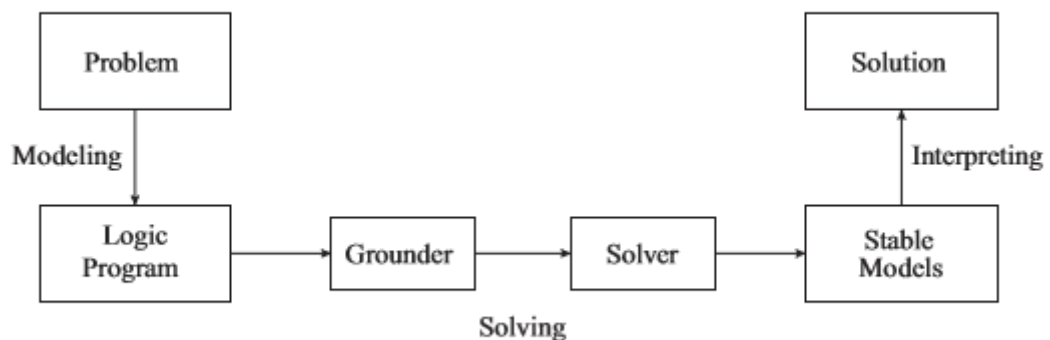
To fully capture out-of-causal reversible computation, consider the example in Figure 2.11 Out-of-Causal reversing in a RPN occurring after  $t_1$ ,  $t_2$ ,  $t_3$  fired in this order.. Transitions  $t_1$ ,  $t_2$ , and  $t_3$  were forward-executed and the marking of the Petri net consist of 3 bonds in place  $z$ . Token  $d$  is connected to  $a$ ,  $a$  is connected to  $b$ , and  $b$  is connected to  $c$ . As demonstrated,  $t_1$  is the first transition to reverse. Thus, the bond between  $a-b$  should break. This will result in components  $d-a$  and  $b-c$ . Bond  $d-a$  is last used by transition  $t_3$ , and thus should remain in place  $z$ . On the other hand, bond  $b-c$  is last used by transition  $t_2$ , and thus will be relocated to place  $y$ . Then,  $t_2$  is reversed. The effect of transition  $t_2$  is the creation of bond  $b-c$ , and thus, then the transition is reversed, the bond breaks resulting in the components  $b$ ,  $c$ . The last transition that used the components is  $t_1$ , but is already reversed and so, the components return to their initial marking.  $t_3$  is reversed last, and breaks bond  $a-d$ , causing them to return to their initial marking.

## 2.6 Answer Set Programming Language

Answer Set Programming (ASP) is a declarative logic modelling language. It is primarily oriented towards difficult search problems. The idea is instead of solving the problem, by programming the computer to do so, we simply describe the problem[27]. The first form of ASP is the planning method proposed in 1997 [23], which is based on the relation between plans and stable models. It is an outcome of years of research on knowledge representation, logic programming and constrain satisfaction. It is characterised for it is simplicity, ease of use and computational effectiveness.

Below the ASP process is described. The first step is to model the problem in the syntax of first logic programming. The next step, is that grounder generates a finite propositional representation of the input program, and consequently, a solver computes the stable models of the propositional program. The solution is constructed out of the stable models. This is illustrated in Figure 2.12 ASP solving process.

Next, we ompare ASP to Prolog, traditional logic programming language, so that we realize the differences. Prolog, is a top-down query evaluation, were variables are dealt with via unification and terms, which can be nested, are used as basic data structures. A solution is usually extracted from the instantiation of the variables in a successful query. On the other hand, solutions in ASP are captured by models, and instead computed in a bottom-up fashion. Variables are systematically replaced by using database techniques. Tuples and terms, which cannot be nested, are the preferred data structures. Last but not least, Unlike Prolog, ASP always terminates.



**Figure 2.12 ASP solving process.**

ASP can be used in Bioinformatics, Robotics, Music composition, Decision support systems, product configuration.

### 2.6.1 ASP syntax and Semantics

ASP consists of *atoms*, *literals* and *rules*. Atoms are elementary propositions that may be true or false. Literals are atoms and their negation. Negation in ASP does not stand as the standard negation operator, but instead it stands for the modality, “non-derivable”. Rules are expressions of the form:

$$\underbrace{a \leftarrow}_{\text{Head}} \underbrace{b_1, b_2, \dots, b_m, \text{ not } c_1, \dots c_n}_{\text{Body}}$$

$a, b_i, c_i$  are atoms. A rule is a justification to derive that the head is true, if all literals in the body are true.

Rules may have no body and are then called *facts*. For instance the rule:

$$a \leftarrow$$

In such rules, the head is unconditionally true, and the arrow is typically omitted.

Programs in ASP are a finite set of rules.

## 2.7 Clingo

Current answer set solvers work on variable-free programs, and thus, a grounder is needed. A grounder, given an input program with first-order variables, computes an equivalent variable-free program. An example of grounder created by the University of Potsdam is gringo [27]. Its output can be processed further with clasp, claspfolio, or clingcon.

Clingo combines both gringo and clasp into a monolithic system. This way it offers more control over the grounding and solving process than gringo and clasp can offer individually.

## 2.8 Petri nets in ASP

A simple form of Petri nets in ASP is developed in [15]. My implementation of reversing Petri nets is based on this work, so this section is devoted to explain and understand it.

The Petri net’s structure is defined with the following facts:

- f1: place( $p_i$ ), where  $p_i \in P$  is a place.
- f2: trans( $t_j$ ), where  $t_j \in T$  is a transition.
- f3: ptarc( $p_i, t_j, W(p_i, t_j)$ ), where  $p_i \in P$ ,  $t_j \in T$ ,  $W(p_i, t_j)$  is the weight of the arc from place  $p_i$  to transition  $t_j$ .
- f4: tparc( $t_i, p_j, W(t_i, p_j)$ ), where  $p_i \in P$ ,  $t_j \in T$ ,  $W(t_i, p_j)$  is the weight of the arc from transition  $t_i$  to place  $p_j$ .

Petri net execution simulation is executed in discrete time-steps. Therefore time is encoded by the following fact:

- f5: time( $t_{si}$ ) where  $0 \leq t_{si} \leq k$ .

The initial marking is represented as follows:

- i1: holds( $p_i, M0(p_i), 0$ ), for every place  $p_i \in P$  with initial marking  $M0(p_i)$ .

The 0 in the end of the holds triple, represents the time and is set to 0 to indicate that the fact is true for the initial instance.

ASP requires all variables in rule bodies to be domain restricted. Thus, we add the following facts to capture token quantities produced during the simulation.

- x1: num( $n$ ), where  $0 \leq n \leq \text{ntok}$

ntok is the maximum number of tokens that are allowed in the Petri net.

Lets now continue to the declaration of rules of enabled transitions.

- e1: notenabled( $T, TS$ ):- ptarc( $P, T, N$ ), holds( $P, Q, TS$ ),  $Q < N$ , place( $P$ ), trans( $T$ ), time( $TS$ ), num( $N$ ), num( $Q$ ).
- e2: enabled( $T, TS$ ) :- trans( $T$ ), time( $TS$ ), not notenabled( $T, TS$ ).

Rules e1 states that a transition is not enabled at time instance TS, when the input place of that transition has fewer tokens than the number of tokens stated at the arc between that place and that transition. If a transition is not set to be not enabled, then it means that it has at least the amount of tokens required and is thus set to enabled.

A subset of enabled transitions may fire and this is encoded by the rule:

a1: {fires(T, TS)} :- enabled(T, TS), trans(T), time(TS).

any transition that is enabled at a time instance TS can fire.

Consumption and production of tokens due to the firing of transitions are captured in rules:

r1: add(P, Q, T, TS) :- fires(T, TS), tparc(T, P, Q), time(TS).

r2: del(P, Q, T, TS) :- fires(T, TS), ptarc(P, T, Q), time(TS).

r3: tot incr(P, QQ, TS) :- QQ=#sum[add(P, Q, T, TS)=Q:num(Q):trans(T)],  
time(TS), num(QQ), place(P).

r4: tot decr(P, QQ, TS) :- QQ=#sum[del(P, Q, T, TS)=Q:num(Q):trans(T)],  
time(TS), num(QQ), place(P).

r5: holds(P, Q, TS+1) :-holds(P, Q1, TS), tot incr(P, Q2, TS), time(TS+1), tot  
decr(P, Q3, TS), Q=Q1+Q2-Q3, place(P), num(Q;Q1;Q2;Q3), time(TS)

Rule r1 encodes the addition of Q tokens to a place P due to the firing of the transition T at time instance TS. Respectively, r2 encodes the deduction of Q tokens at place P. Because more than one transitions can fire at an instance, rules r3, and r4, calculate the total amount of tokens that will be added to/removed from a place. If no transition that is connected to the place fires then no change will occur. In rule r5, the new marking of the Petri net is calculated.

To prevent overconsumption at a place due to the firing of two or more transitions that draw tokens from it, rules are extended with the following:

a2: consumesmore(P, TS) :- holds(P, Q, TS), tot decr(P, Q1, TS),  $Q1 > Q$ .  
a3: consumesmore :- consumesmore(P, TS).  
a4: :- consumesmore.

Rule a2 defines overconsumption at a specific time instance, while rule a3, states that if overconsumption is detected in an instance, then the whole execution has overconsumption. Rule r4 states, that if overconsumption is detected then the solution is deadlocked. Solutions with deadlocks are not desirable.

Additionally, two more rules are added to the program to ensure maximum activity:

a5: could not have(T, TS) :- enabled(T, TS), not fires(T, TS), ptarc(S, T, Q),  
holds(S, QQ, TS), tot decr(S, QQQ, TS),  $Q > QQ - QQQ$ .  
a6: :-not could not have(T, TS), enabled(T, TS), not fires(T, TS), trans(T),  
time(TS).

According to Rule a5 a transition that does not fire but is enabled, it is because it would result in overconsumption. Rule a6 states that any state where a transition is enabled and does not fire but does not result in overconsumption is deadlocked.



# Chapter 3

## Encoding Reversing Petri nets in ASP

### Contents

---

3.1 REVERSING PETRI NET STRUCTURE.....	29
3.2 FORWARD EXECUTION OF REVERSING PETRI NETS.....	33
3.3 BACKTRACKING.....	38
3.4 CAUSAL REVERSING.....	44
3.5 OUT-OF-CAUSAL REVERSIBILITY:.....	45
3.6 PROPERTIES ON REVERSING PETRI NETS.....	49
3.7 SHORTEST PATH.....	52

---

As mentioned in subchapter 2.4 Reversing Petri nets, Reversing Petri nets, have differences from the ordinary Petri nets implemented [8]. Therefore, the ASP rules, should be extended to support reversibility along with the 3 modes of reversible computations explained in Chapter 2.5 Semantics.

### 3.1 Reversing Petri Net Structure

We keep the same definition for places, transitions and time:

f1: place( $p_i$ ), where  $p_i \in P$  is a place.

f2: trans( $t_j$ ), where  $t_j \in T$  is a transition.

f3: time( $t_{si}$ ) where  $0 \leq t_{si} \leq k$ .

The first thing that had to change from the definition of ordinary Petri nets, was token representation. The implementation of tokens we saw in the previous chapter, refers to simple Petri net that can have quantity of the only one type of tokens. On reversing Petri nets though, tokens can be of different type but are unique, as only one token of a specific type can be defined.

f4: token( $q$ ), where  $q \in A$  is a token.

Therefore, any fact that used tokens, should now be altered as follows:

f5: ptarc( $p_i, t_j, q$ ), where  $(p_i, t_j) \in F$ ,  $\ell = F(p_i, t_j)$ ,  $q \in \ell$ .

f6: tparc( $t_i, p_j, q$ ), where  $(t_i, p_j) \in F$ ,  $\ell = F(t_i, p_j)$ ,  $q \in \ell$ .

i1: holds( $p_i, q, 0$ ), where  $p_i \in P$ ,  $q \in A$ , and  $q$  is initially at place  $p_i$ .

Facts f5 and f6 define the arcs of the Petri net. Specifically, they state the existences of an arc from place  $p_i$  to transition  $t_j$  or from transition  $t_i$  to the place  $p_j$ , with the label requesting the presence or producing the token  $q$ . Fact i1 is used to set the initial marking of the Petri net.

In order to fully define a reversing Petri net, negation and bonds need to be represented as well. We introduce the following facts for bonds:

f7: ptarcbond( $p_i, t_j, q, qq$ ), where  $(p_i, t_j) \in F$ ,  $\ell = F(p_i, t_j)$ ,  $\text{con}(q, qq) \in \ell$ .

f8: tparcbond( $t_i, p_j, q, qq$ ), where  $(t_i, p_j) \in F$ ,  $\ell = F(t_i, p_j)$ ,  $\text{con}(q, qq) \in \ell$ .

i2: holdsbonds( $p_i, q, qq, 0$ ), where  $p_i \in P$ ,  $q, qq \in A$ , and  $\text{con}(q, qq)$  is initially at place  $p_i$ .

And the following for negation:

f9:  $\text{ptarcabsence}(\text{pi}, \text{tj}, \text{q}), \text{ where } (\text{pi}, \text{tj}) \in \text{F}, \ell = \text{F}(\text{pi}, \text{tj}), \overline{\text{q}} \in \ell.$

f10:  $\text{tparcabsence}(\text{ti}, \text{pj}, \text{q}), \text{ where } (\text{ti}, \text{pj}) \in \text{F}, \ell = \text{F}(\text{ti}, \text{pj}), \overline{\text{q}} \in \ell.$

f11:  $\text{ptarcbondabsence}(\text{pi}, \text{tj}, \text{q}, \text{qq}), \text{ where } (\text{pi}, \text{tj}) \in \text{F}, \ell = \text{F}(\text{pi}, \text{tj}), \overline{\text{q-qq}} \in \ell.$

f12:  $\text{tparcbondabsence}(\text{ti}, \text{pj}, \text{q}, \text{qq}), \text{ where } (\text{ti}, \text{pj}) \in \text{F}, \ell = \text{F}(\text{ti}, \text{pj}), \overline{\text{q-qq}} \in \ell.$

Facts f7 and f8 define the arc that is labelled with a connection of two tokens, while i2 states that a bond between q and qq is initially set at place pi. Facts f9 and f10 state that the arc from place pi to transition tj, or from tj to pi, requires that token q is absent. f11 and f12, state the same but for bonds.

Reversing Petri nets also have a type of memory, namely history, that we should define:

h1:  $\text{History}(\text{h}), \text{ where } 0 \leq \text{tsi} \leq \text{k}.$

h2:  $\text{transHistory}(\text{tj}, 0, 0), \text{ where } \text{tj} \in \text{T}.$

Fact h1 is used, because ASP requires all variables to be domain restricted. The fact h2, is used to explain that every transition should be set to 0 to indicate that it has not yet fired forward initially. If instead the transition has been executed in a forward mode and we want to see the execution in a later on point, we could change the fact to the following:

h3:  $\text{transHistory}(\text{tj}, \text{h}, 0), \text{ where } \text{tj} \in \text{T}, 0 < \text{h} \leq \text{k}$

Below we can see a definition of the Petri net of Figure 2.9 it has returned to its initial marking.

Defining places:

f1:  $\text{place}(\text{p1}).$

f2:  $\text{place}(\text{p2}).$

f3:  $\text{place}(\text{p3}).$

f4: place(p4).

f5: place(p5).

Defining transitions:

f6: trans(t1).

f7: trans(t2).

f8: trans(t3).

Defining tokens:

f9: token(a).

f10: token(b).

Defining arcs:

f11: ptarc(p1, t1, a).

f12: tparc(t1, p2, a).

f13: ptarc(p2, t2, b).

f14: tparc(t2, p4, b).

f15: ptarc(p2, t3, a).

f16: ptarc(p4, t3, b).

f17: tparcbond(t3, p5, b, a).

Initial marking:

i1: holds(p1, a, 0).

i2: holds(p3, b, 0).

History and time:

f18: time(0..5).

f19: history(0..5).

f20: transHistory(t1, 0, 0).

f21: transHistory(t2, 0, 0).

f22: transHistory(t3, 0, 0).

**Figure 3. 1 Initial Marking of a petri net**

Now that the Petri net's structure is defined, we can move forward to implement the modes of operation.

### 3.2 Forward Execution of Reversible Petri Nets

Firstly, we need to redefine the rules of finding enabled transitions. We should also check for bonds and if any negative base is present in the place when it is unwanted:

- e1: `notenabled(T, TS):-ptarc(P, T, Q), not holds(P, Q, TS), time(TS).`
- e2: `notenabled(T, TS):-ptarcbond(P, T, Q1, Q2), not holdsbonds(P, Q1, Q2, TS), time(TS).`
- e3: `notenabled(T, TS):-ptarcabsence(P, T, Q), holds(P, Q, TS), time(TS).`
- e4: `notenabled(T, TS):-ptarcbondabsence(P, T, Q1, Q2), holdsbonds(P, Q1, Q2, TS), time(TS).`
- e5: `enabled(T, TS):-not notenabled(T, TS), trans(T), time(TS).`

Rules e1, e2 state that a transition is not enabled if it requires a token/bond and at current time instance, the place does not contain that token/bond. Rules e3, e4 state that a transition is not enabled if it requires the absence of a particular token/bond, and the transition does hold that token/bond. Finally, If none of the above holds, then the transition is indeed enabled.

Rule for firing of a transition remains the same :

- a1: `{fires(T, TS)}:-enabled(T, TS), time(TS).`

Rules for relocating tokens after the firing of a transition will be inserted:

- t1: `bonded(P, X, Y, TS):- connectedTokens(P, X, Y, TS).`
- t2: `bonded(P, X, Y, TS):- bonded(P, X, Z, TS), not same(Z, X), not same(X, Y), not same(Z, Y), connectedTokens(P, Z, Y, TS).`
- t3: `connectedTokens(P, X, Y, TS):-holdsbonds(P, X, Y, TS).`
- t4: `connectedTokens(P, X, Y, TS):-holdsbonds(P, Y, X, TS).`
- t5: `connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, Q1, _), tparcbond(T, TP, Q1, _).`

t6: connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, Q1, \_),  
tparcbond(T, TP, \_, Q1).

t7: connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, \_, Q1),  
tparcbond(T, TP, Q1, \_).

t8: connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, \_, Q1),  
tparcbond(T, TP, \_, Q1).

r1: add(P, Q, T, TS):- fires(T, TS), tparc(T, P, Q), time(TS).

r2: add(P, Q, T, TS):- fires(T, TS), tparcbond(T, P, Q, \_), time(TS).

r3: add(P, Q, T, TS):- fires(T, TS), tparcbond(T, P, \_, Q), time(TS).

r4: add(P, Q, T, TS):- fires(T, TS), ptarc(PT, T, Q1), tparc(T, P, Q1),  
bonded(PT, Q, Q1, TS), time(TS).

r5: add(P, Q, T, TS):-fires(T, TS), ptarc(PT, T, Q1), tparcbond(T, P, Q1, \_),  
bonded(PT, Q, Q1, TS), time(TS).

r6: add(P, Q, T, TS):-fires(T, TS), ptarc(PT, T, Q1), tparcbond(T, P, \_, Q1),  
bonded(PT, Q, Q1, TS), time(TS).

r7: add(P, Q, T, TS):-fires(T, TS), tparcbond(PT, T, Q1, \_), tparc(T, P, Q1),  
bonded(PT, Q, Q1, TS), time(TS).

r8: add(P, Q, T, TS):-fires(T, TS), tparcbond(PT, T, \_, Q1), tparc(T, P, Q1),  
bonded(PT, Q, Q1, TS), time(TS).

r9: add(TP, Q, T, TS):-fires(T, TS), connectedBondPlacesThrough(PT, T, TP,  
Q1), bonded(PT, Q, Q1, TS), time(TS).

Rules t1-t4 are rules to minimize the number of rules. t1-t4 finds all the bonded tokens, either directly or implicitly connected, while t5-t8 is all the possible combinations where two places can be connected by an arc labeled with a bond.

Rule r1 states that at place P token Q will be added because it is the output place of the firing transition T labelled by Q, r2 and r3 that the token Q will be moved to the place P due to the firing transition T which creates a bond between Q and another token (\_). Rules r4-r9 state that any token Q that is bonded with another one Q1 at the input place of the transition, will move along with its bonded token, Q1 to the output place of the transition.

Respectively the same rules are defined for the removal of tokens from a place.

```

r10:del(P, Q, T, TS):-fires(T, TS), ptarc(P, T, Q), time(TS).

r11:del(P, Q, T, TS):-fires(T, TS), ptarcbond(P, T, Q, _), time(TS).

r12:del(P, Q, T, TS):-fires(T, TS), ptarcbond(P, T, _, Q), time(TS).

r13:del(P, Q, T, TS):-fires(T, TS), ptarc(P, T, Q1), tparc(T, TP, Q1), bonded(P, Q,
    Q1, TS), time(TS).

r14:del(P, Q, T, TS):-fires(T, TS), ptarc(P, T, Q1), tparcbond(T, TP, Q1, _),
    bonded(P, Q, Q1, TS), time(TS).

r15:del(P, Q, T, TS):-fires(T, TS), ptarc(P, T, Q1), tparcbond(T, TP, _, Q1),
    bonded(P, Q, Q1, TS), time(TS).

r16:del(P, Q, T, TS):-fires(T, TS), ptarcbond(P, T, Q1, _), tparc(T, TP, Q1),
    bonded(P, Q, Q1, TS), time(TS).

r17:del(P, Q, T, TS):-fires(T, TS), ptarcbond(P, T, _, Q1), tparc(T, TP, Q1),
    bonded(P, Q, Q1, TS), time(TS).

r18:del(P, Q, T, TS):-fires(T, TS), connectedBondPlacesThrough(P, T, TP,
    Q1), bonded(P, Q, Q1, TS), time(TS).

```

We also need to include rules for creating and moving bonds:

```

r19:createBond(P, Q1, Q2, T, TS):-fires(T, TS), tparcbond(T, P, Q1, Q2),
    time(TS).

r20:moveBond(P, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(P, Q, Q1, TS),
    ptarc(P, T, Q2), tparc(T, TP, Q2), bonded(P, Q, Q2, TS).

r21:moveBond(P, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(P, Q, Q1, TS),
    ptarc(P, T, Q2), tparc(T, TP, Q2), bonded(P, Q1, Q2, TS).

r22:moveBond(P, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(P, Q, Q1, TS),
    ptarc(P, T, Q2), tparcbond(T, TP, Q2, _), bonded(P, Q, Q2, TS).

r23:moveBond(P, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(P, Q, Q1, TS),
    ptarc(P, T, Q2), tparcbond(T, TP, Q2, _), bonded(P, Q1, Q2, TS).

r24:moveBond(P, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(P, Q, Q1, TS),
    ptarc(P, T, Q2), tparcbond(T, TP, _, Q2), bonded(P, Q, Q2, TS).

r25:moveBond(P, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(P, Q, Q1, TS),
    ptarc(P, T, Q2), tparcbond(T, TP, _, Q2), bonded(P, Q1, Q2, TS).

```

```

r26:moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),
    ptarcbond(PT, T, Q2, _), tparc(T, TP, Q2), bonded(PT, Q, Q2, TS).

r27:moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),
    ptarcbond(PT, T, Q2, _), tparc(T, TP, Q2), bonded(PT, Q1, Q2, TS).

r28:moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),
    ptarcbond(PT, T, _, Q2), tparc(T, TP, Q2), bonded(PT, Q, Q2, TS).

r29:moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),
    ptarcbond(PT, T, _, Q2), tparc(T, TP, Q2), bonded(PT, Q1, Q2, TS).

r30:moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),
    connectedBondPlacesThrough(PT, T, TP, Q2), bonded(PT, Q, Q2, TS).

r31:moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),
    connectedBondPlacesThrough(PT, T, TP, Q2), bonded(PT, Q1, Q2, TS).

r32:break(PT, Q, Q1, TS):-moveBond(PT, TP, Q, Q1, TS).

```

A new bond is created in a place  $p$ , when  $p$  is the output place of a transition to place directed arc labelled with a bond. A bond is moved from a place to another, if one of the two tokens, need to be transferred to the new place, either because they are written on the label, or one of their connected tokens is. We need to clarify that r32, does not specify that a bond will break as to it will no longer connect the tokens, but that it will no longer be a bond in this place because it will move to another place. Forward execution cannot destroy bonds. Bonds are destroyed only on the reversible computation.

Below we can see the rules that define the new marking:

```

r33:holds(P, Q, TS+1):-add(P, Q, T, TS), fires(T, TS), time(TS).

r34:holds(P, Q, TS+1):-holds(P, Q, TS), fires(T, TS), not del(P, Q, T, TS),
    time(TS).

r35:holds(P, Q, TS+1):-holds(P, Q, TS), firing(F, TS), F=0.

r36:holdsbonds(P, Q1, Q2, TS+1):-createBond(P, Q1, Q2, T, TS), time(TS).

r37:holdsbonds(P, Q1, Q2, TS+1):-holdsbonds(P, Q1, Q2, TS), not break(P, Q1,
    Q2, TS),
    not break(P, Q2, Q1, TS), time(TS).

```



```

r38:holdsBonds(P, Q1, Q2, TS+1):-moveBond(P1, P, Q1, Q2, TS),

not holdsBonds(P, Q2, Q1, TS+1), time(TS).

r39: transHistory(T1, H1+1, TS+1):-H1=#max{H:transHistory(T, H, TS),
history(H), trans(T)}, trans(T1), fires(T1, TS), time(TS), history(H1),
time(TS+1), history(H1+1).

clingo version 5.3.0
Reading from figure2.lp ...
Solving...
Answer: 1
time(0) time(1) time(2) time(3) time(4) time(5) holds(p1,a,0)
holds(p3,b,0) enabled(t1,0) enabled(t2,0) enabled(t2,1)
enabled(t3,2) holds(p2,a,1) holds(p2,a,2) holds(p3,b,1)
holds(p4,b,2) fires(t1,0) fires(t2,1) fires(t3,2)
add(p2,a,t1,0) add(p4,b,t2,1) add(p5,b,t3,2) add(p5,a,t3,2)
createBond(p5,b,a,t3,2) holdsBonds(p5,b,a,3)
holdsBonds(p5,b,a,4) holdsBonds(p5,b,a,5) holdsBonds(p5,b,a,6)
holds(p5,b,3) holds(p5,b,4) holds(p5,b,5) holds(p5,b,6)
holds(p5,a,3) holds(p5,a,4) holds(p5,a,5) holds(p5,a,6)
del(p4,b,t3,2) del(p2,a,t3,2) del(p1,a,t1,0) del(p3,b,t2,1)
SATISFIABLE

Models          : 1+
Calls           : 1
Time            : 0.038s (Solving: 0.00s 1st Model: 0.00s Unsat:
0.00s)
CPU Time        : 0.031s
|

```

**Figure 3. 2 Results of execution**

```

transHistory(T,H,TS+1):-transHistory(T,H,TS),not fires(T,TS),not firesB(T,TS),
time(TS),history(H),time(TS+1).

```

As we can see from the rules, a place that is empty will have no predicate *holds* referring to it, in that specific instance. If no change arises, then the marking will be the same for the next instance. R36-r38 states the new connections that hold on each place. If a place has no connections then no predicate *holdsBonds* will refer to it.

We want to make sure that only one transition will fire at a time. To achieve this we define the following:

```

a2: firing(Q, TS):-Q=#count{T:fires(T, TS)}, time(TS).

a3: :-firing(Q, TS), Q>1, time(TS).

a4: :-firing(Q, TS), Q=0, enabled(T, TS), time(TS).

```

Thus, if the number of transitions that fire at a given time exceeds 1, then the execution leads to a deadlock.

In Figure 3. 2 above, we can see the results of the forward execution of the Petri net of Figure 3. 1. Time is the last value in all predicates. At time instance 0, the initial marking is presented, where place  $p1$  holds the token  $a$  and place  $p3$  token  $b$ . Transitions  $t1$  and  $t2$  are enabled, from the beginning but only one can fire. Transition  $t1$  is non deterministically selected to fire. Token  $a$  is added at place  $p2$ , and deleted from place  $p1$ . Nothing else happens at time instance 0, so we move on with time 1. Again, the marking is presented, where  $p2$  holds token  $a$ .  $p3$ , still holds  $b$  as nothing changed its marking. Transition  $t2$  is enabled and it fires causing token  $b$  to move from  $p3$  to  $p4$ . In time instance 2,  $t3$  is enabled and it fires. we can see that tokens  $a$  and  $b$  are added at final place  $p5$ , and deleted from  $p2$  and  $p4$  respectively. Additionally, a bond is created between  $a$  and  $b$ . In the next instance, we have token  $a$  and  $b$  in place  $p5$  and they are bonded. No transition can fire in the next instances, so the marking remains the same.

### 3.3 Backtracking

As mentioned in Chapter 2.4 Reversing Petri nets, Backtracking allows us to reverse only the last transition that had fired.

We can now demonstrate the next mode of reversibility, namely, backtracking.

Firstly, we again have some additional rules to minimize the number of rules used later on. We define them from the beginning as they are used in different parts of the program, and they will replace the ones used in forward execution only, as they make additional checks.

$t1: \text{same}(X, X):- \text{token}(X).$

$t2: \text{bonded}(P, X, Y, TS):- \text{connectedTokens}(P, X, Y, TS).$

$t3: \text{bonded}(P, X, Y, TS):- \text{bonded}(P, X, Z, TS), \text{ not same}(Z, X), \text{ not same}(X, Y),$   
 $\text{ not same}(Z, Y), \text{ connectedTokens}(P, Z, Y, TS).$

t4: connectedTokens(P, X, Y, TS):- holdsbonds(P, X, Y, TS), not breakBond(P, X, Y, TS), time(TS).

t5: connectedTokens(P, X, Y, TS):-holdsbonds(P, Y, X, TS) , not breakBond(P, Y, X, TS), time(TS).

t6: connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, Q1, \_), tparcbond(T, TP, Q1, \_).

t7: connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, Q1, \_), tparcbond(T, TP, \_, Q1).

t8: connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, \_, Q1), tparcbond(T, TP, Q1, \_).

t9: connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, \_, Q1), tparcbond(T, TP, \_, Q1).

Rule t1, state that to tokens are the same. Rule t2 and t3 find all connected tokens, either implicitly or directly, in a place at a specific time instance TS. Tokens that were connected but the firing transition will destroy the bond in this instance, are not considered bonded in this time instance. rules t6 to t7 again refer to all the combinations of arcs labelled with bonds that could set 2 places connected and using a token.

Rules for enabled transitions to backtrack:

eb1: notenabledB(T, TS):-HH=#max{ HHH:transHistory(T1, HHH, TS), history(HHH), trans(T1) }, history(HH), history(H), trans(T), transHistory(T, H, TS), HH>H, time(TS).

eb2: notenabledB(T, TS):-transHistory(T, H, TS), H=0.

eb3: notenabledB(T, t):-irreversible(T).

eb4: enabledB(T, TS):-trans(T), time(TS), not notenabledB(T, TS).

According to rule eb1, a transition is not *bt*-enabled, when another transition exists with bigger history. Additionally, based on eb2 a transition that has 0 history, and thus has not yet fired forward, cannot be reversed. If nothing of the above holds, for a transition t, then t is *bt*-enabled, by eb4.

We need to define the rule for firing backwards:

a1: {firesB(T, TS)}:-enabledB(T, TS), trans(T), time(TS).

any transition that is backward enabled can fire.

We include rules for moving the tokens and bonds back:

r1: addB(P, Q, T, TS):-firesB(T, TS), ptarc(P, T, Q), time(TS).

r2: addB(P, Q, T, TS):-firesB(T, TS), ptarcbond(P, T, Q, \_), time(TS).

r3: addB(P, Q, T, TS):-firesB(T, TS), ptarcbond(P, T, \_, Q), time(TS).

r4: addB(P, Q, T, TS):-firesB(T, TS), ptarc(P, T, Q1), tparc(T, TP, Q1),  
bonded(TP, Q1, Q, TS), time(TS).

r5: addB(P, Q, T, TS):-firesB(T, TS), ptarc(P, T, Q1), tparcbond(T, TP, Q1, \_),  
bonded(TP, Q1, Q, TS), time(TS).

r6: addB(P, Q, T, TS):-firesB(T, TS), ptarc(P, T, Q1), tparcbond(T, TP, \_, Q1),  
bonded(TP, Q1, Q, TS), time(TS).

r7: addB(P, Q, T, TS):-firesB(T, TS), ptarcbond(P, T, Q1, \_), tparc(T, TP, Q1),  
bonded(TP, Q1, Q, TS), time(TS).

r8: addB(P, Q, T, TS):-firesB(T, TS), ptarcbond(P, T, \_, Q1), tparc(T, TP, Q1),  
bonded(TP, Q1, Q, TS), time(TS).

r9: addB(P, Q, T, TS):-firesB(T, TS), connectedBondPlacesThrough(P, T, TP,  
Q1), bonded(TP, Q1, Q, TS), time(TS).

r10: delB(P, Q, T, TS):-firesB(T, TS), tparc(T, P, Q), time(TS).

r11: delB(P, Q, T, TS):-firesB(T, TS), tparcbond(T, P, Q, \_), time(TS).

r12: delB(P, Q, T, TS):-firesB(T, TS), tparcbond(T, P, \_, Q), time(TS).

r13: delB(P, Q, T, TS):-firesB(T, TS), tparc(T, P, Q1), bonded(P, Q, Q1, TS),  
time(TS).

r14: delB(P, Q, T, TS):-firesB(T, TS), tparcbond(T, P, Q1, \_), bonded(P, Q, Q1,  
TS), time(TS).

r15: delB(P, Q, T, TS):-firesB(T, TS), tparcbond(T, P, \_, Q1), bonded(P, Q, Q1,  
TS), time(TS).

Rules r1-r16 work in the opposite way of the forward execution. Any token that is written on the label will be moved from the output place back to the input place along with its bonded tokens, if that specific bond does not break in this instance. The token will be deleted from the output place and added to the input place.

Moving and breaking bonds:

```

r16:breakBond(P, Q1, Q2, TS):-firesB(T, TS), tparcbond(T, P, Q1, Q2), time(TS).
r17:breakB(P, Q1, Q2, TS):-moveBondB(P, PT, Q1, Q2, TS), time(TS).
r18:breakB(P, Q1, Q2, TS):-breakBond(P, Q1, Q2, TS).
r19:moveBondB(TP, PT, Q1, Q2, TS):-firesB(T, TS), holdsbonds(TP, Q1, Q2,
    TS), ptarc(PT, T, Q), tparc(T, TP, Q), bonded(TP, Q1, Q, TS).
r20:moveBondB(TP, PT, Q1, Q2, TS):-firesB(T, TS), holdsbonds(TP, Q1, Q2,
    TS), ptarc(PT, T, Q), tparc(T, TP, Q), bonded(TP, Q2, Q, TS).
r21:moveBondB(TP, PT, Q1, Q2, TS):-firesB(T, TS), holdsbonds(TP, Q1, Q2,
    TS), ptarc(PT, T, Q), tparcbond(T, TP, Q, _), bonded(TP, Q1, Q, TS).
r22:moveBondB(TP, PT, Q1, Q2, TS):-firesB(T, TS), holdsbonds(TP, Q1, Q2,
    TS), ptarc(PT, T, Q), tparcbond(T, TP, Q, _), bonded(TP, Q2, Q, TS).
r23:moveBondB(TP, PT, Q1, Q2, TS):-firesB(T, TS), holdsbonds(TP, Q1, Q2,
    TS), ptarc(PT, T, Q), tparcbond(T, TP, _, Q), bonded(TP, Q1, Q, TS).
r24:moveBondB(TP, PT, Q1, Q2, TS):-firesB(T, TS), holdsbonds(TP, Q1, Q2,
    TS), ptarc(PT, T, Q), tparcbond(T, TP, _, Q), bonded(TP, Q2, Q, TS).
r25:moveBondB(TP, PT, Q1, Q2, TS):-firesB(T, TS), holdsbonds(TP, Q1, Q2,
    TS), tparcbond(PT, T, Q, _), tparc(T, TP, Q), bonded(TP, Q1, Q, TS).
r26:moveBondB(TP, PT, Q1, Q2, TS):-firesB(T, TS), holdsbonds(TP, Q1, Q2,
    TS), tparcbond(PT, T, Q, _), tparc(T, TP, Q), bonded(TP, Q2, Q, TS).
r27:moveBondB(TP, PT, Q1, Q2, TS):-firesB(T, TS), holdsbonds(TP, Q1, Q2,
    TS), tparcbond(PT, T, _, Q), tparc(T, TP, Q), bonded(TP, Q1, Q, TS).
r28:moveBondB(TP, PT, Q1, Q2, TS):-firesB(T, TS), holdsbonds(TP, Q1, Q2,
    TS), tparcbond(PT, T, _, Q), tparc(T, TP, Q), bonded(TP, Q2, Q, TS).
r29:moveBondB(TP, PT, Q1, Q2, TS):-firesB(T, TS), holdsbonds(TP, Q1, Q2,
    TS), connectedBondPlacesThrough(PT, T, TP, Q), bonded(TP, Q1, Q, TS).

```

r30:moveBondB(TP, PT, Q1, Q2, TS):-firesB(T, TS), holdsbonds(TP, Q1, Q2, TS), connectedBondPlacesThrough(PT, T, TP, Q), bonded(TP, Q2, Q, TS).

Rule r16 refers to the break of a bond due to the reversing of a transition, while r17 refers to the break of a bond of a place because it is now moved to another. r8's aim is to combine the rules r16 and r17 in one. r18-r30 find the bonds that need to move from one place to another.

Update rules:

r31:holds(P, Q, TS+1):-addB(P, Q, T, TS), firesB(T, TS), time(TS).

r32:holds(P, Q, TS+1):-holds(P, Q, TS), firesB(T, TS), not delB(P, Q, T, TS), time(TS).

r33:holdsbonds(P, Q1, Q2, TS+1):-holdsbonds(P, Q1, Q2, TS), not break(P, Q1, Q2, TS), not break(P, Q2, Q1, TS), not breakB(P, Q1, Q2, TS), not breakB(P, Q2, Q1, TS), time(TS).

r34:holdsbonds(P, Q1, Q2, TS+1):-moveBondB(P1, P, Q1, Q2, TS), not holdsbonds(P, Q2, Q1, TS+1), time(TS).

r35:transHistory(T, H, TS+1):-transHistory(T, H, TS), not fires(T, TS), not firesB(T, TS), time(TS), history(H), time(TS+1).

r36:transHistory(T, 0, TS+1):-transHistory(T, H, TS), firesB(T, TS), history(H), time(TS), time(TS+1).

Rules r31 and r32 update the marking of the places by creating a predicate holds if tokens are added and deleting it if tokens are deleted. No predicate will exist for places that hold no tokens at all. r33 to r34 update the bonds between the tokens. r33 maintains the holdsbond predicate if the bond does not break or move to another place, while r34 creates a new predicate when the bond is moved to the new place. Rule r35 and r36 retain the history of the transition to the same value, if it neither fires nor reverses, and resets the history if it reverses.

We still have to make sure that only one transition will fire either forward or backward so we will replace the rules a2:-a4: of forward execution with the following:

```

a2: firing(Q, TS):-Q1=#count{T:fires(T, TS)}, Q2=#count{T1:firesB(T1, TS)},
    Q=Q1+Q2, time(TS).

a3: :-firing(Q, TS), Q>1, time(TS).

a4: :-firing(Q, TS), Q=0, enabled(T, TS), time(TS).

a5: :-firing(Q, TS), Q=0, enabledB(T, TS), time(TS).

```

Based on the above rules, we set an execution to be a deadlock, when more than one transition fires, in either direction, and if no transition fires in any direction but available transitions are present. We want at most and at least one transition to fire or reverse, if there are enabled.

```

clingo version 5.3.0
Reading from figure2.lp ...
Solving...
Answer: 1
time(0) time(1) time(2) time(3) time(4) time(5) holds(p1,a,0)
holds(p3,b,0) enabled(t1,0) enabled(t2,0) enabledB(t1,1)
enabledB(t1,5) enabledB(t2,2) enabledB(t2,4) enabledB(t3,3)
enabled(t2,1) enabled(t2,5) enabled(t3,2) enabled(t3,4)
fires(t1,0) fires(t2,1) fires(t2,5) fires(t3,2) firesB(t2,4)
firesB(t3,3) holds(p3,b,1) holds(p3,b,5) holds(p2,a,1)
holds(p2,a,2) holds(p2,a,4) holds(p2,a,5) holds(p4,b,2)
holds(p4,b,4) add(p2,a,t1,0) add(p4,b,t2,1) add(p4,b,t2,5)
add(p5,b,t3,2) add(p5,a,t3,2) createBond(p5,b,a,t3,2)
holdsBonds(p5,b,a,3) breakBond(p5,b,a,3) holds(p2,a,6)
holds(p4,b,6) holds(p5,b,3) holds(p5,a,3) addB(p3,b,t2,4)
addB(p2,a,t3,3) addB(p4,b,t3,3) del(p3,b,t2,5) del(p3,b,t2,1)
del(p4,b,t3,2) del(p2,a,t3,2) del(p1,a,t1,0)
SATISFIABLE

Models          : 1+
Calls           : 1
Time            : 0.069s (Solving: 0.00s 1st Model: 0.00s Unsat:
0.00s)
CPU Time        : 0.047s

```

**Figure 3. 3 Backward execution**

The Figure 3. 3 shows one solution that is found using the backtracking mode to execute the Figure 2.9 **Error! Reference source not found.** 48 solutions are found when executing the reversing Petri net in backward mode. Again, the initial marking is shown, and the first transition to fire is  $t1$ , although both,  $t1$  and  $t2$  are enabled. In another solution  $t2$  will be the one to fire. The marking changes as token  $a$  is moved from  $p1$  to  $p2$ . At the next instance  $t2$  is still forward enabled and  $t1$  is bt-enabled, as it is the last transition to fire.  $T2$  fires sequentially, and token  $b$  is moved from place  $p3$  to place  $p4$ , setting transition  $t3$  to be enabled.  $T2$  is now backward enabled.  $T3$  fires and

moves tokens  $a$  and  $b$  at place  $p5$  and creates a bond between them.  $T3$  is bt enabled. No other transitions are enabled to fire so  $t3$  will be the one to reverse. the bond created previously breaks, and tokens  $a$  and  $b$  go back to places  $p2$  and  $p4$  respectively.  $T2$  is bt-enabled and  $t3$  is forward enabled.  $T2$  reverses, and token  $b$  goes back to its initial marking,  $p3$ . Finally, we have  $t1$  backtrack enabled, this path would take us back to the initial marking of the petri net and  $t2$  forward enabled.  $t2$  is fired and token  $b$  is again transferred to place  $p4$ .

### 3.4 Causal reversing

The causal reversing mode relaxes the restrictions of backtracking, and allows us to reverse any transition that all its causal dependent transitions had not taken place or had already been reversed. This is equal to the following:

Any transition that has on the output place all the tokens and bonds mentioned in the label of the output arcs, it is causally enabled.

For the out of causal program, we keep all the rules from backtracking program, except enabled rules which will become:

```

eo1: notenabledC(T, TS):-tparc(T, P, Q), not holds(P, Q, TS), time(TS).

eo2: notenabledC(T, TS):-tparcbond(T, P, Q1, Q2), not holdsbonds(P, Q1, Q2, TS),
    time(TS).

eo3: notenabledC(T, TS):-irreversible(T).

eo4: enabledC(T, TS):-trans(T), not notenabledC(T, TS), time(TS).

```

Rule eo1 and eo2 specifies that if a transition exports from the output arc tokens or bonds but that token/bond is not present in the output place, the transition is not enabled. If the previous condition is not true, then the transition is causally enabled.

```

time(0) time(1) time(2) time(3) time(4) time(5) holds(p1,a,0)
holds(p3,b,0) enabled(t1,0) enabled(t2,0) enabledC(t1,2)
enabledC(t1,3) enabledC(t1,4) enabledC(t2,1) enabledC(t2,2)
enabledC(t2,4) enabledC(t2,5) firesC(t1,4) firesC(t2,2)
holds(p2,a,2) holds(p2,a,3) holds(p2,a,4) holds(p4,b,1)
holds(p4,b,2) holds(p4,b,4) holds(p4,b,5) enabled(t1,1)
enabled(t1,5) enabled(t2,3) enabled(t3,2) enabled(t3,4)
fires(t1,1) fires(t1,5) fires(t2,0) fires(t2,3) holds(p2,a,6)
holds(p4,b,6) holds(p1,a,1) holds(p1,a,5) holds(p3,b,3)|

```

Figure 3. 4 Causal Execution



We now execute the same example as before but now with causal mode of reversing. We get 82 solutions, almost double the number of the solutions found in the backtracking mode. In **Error! Reference source not found.**, there is one solution found. Of course, we have the same initial marking in the beginning, and the enabled transitions are t1 and t2. T2 fires first, moving token b from place p3 to p4. Then, t1 is forward enabled and t2 is causally enabled. T1 fires as we can see, moving token a from p1 to p2. At time instance 2, we have t1 and t2 enabled both causally, and t2 will reverse causing the marking to become: a in p2, and b in place p3. Although it was not the last transition to fire forward, it can still be reversed as no transition that is causally enabled on t2 has fired. Right after, it will fire again and t1 will be the one to reverse, only to go forward in the next instance.

### 3.5 Out-of-causal Reversibility:

In order to make reversibility in out-of-causal mode, we use the following algorithm mentioned in [28].

---

**Algorithm 1:** Out of Causal Algorithm

---

```

reset transition  $t$ ;
forall  $p$  in Places do
   $M_n[p] = M_n[p] - effect(t)$ ;
  forall  $a$  in Bases do
     $C = con(a, M_n[p])$ ;
     $t' = lasttransitioninpath(M_0, M_n)$ ;
    1 while  $(C \cap post(t') = \emptyset \text{ or } H_n(t' = 0)) \text{ and } t \neq \epsilon$  do
      |  $t' = onetransitionback$ ;
    end
    if  $t = \epsilon$  then
      forall  $p'$  in Places do
        if  $C \subseteq M_0[p'] \text{ and } p' \neq p$  then
          |  $p = p - C$ ;
          |  $p' = p' + C$ ;
        end
      end
    else
      forall  $p'$  in Places do
        if  $C \cap F(t', p') \neq \emptyset \text{ and } p \neq p'$  then
          |  $p = p - C$ ;
          |  $p' = p' + C$ ;
        end
      end
    end
  end
end
end
end

```

---

**Figure 3. 5** Out of Causal Algorithm

As we can see, the algorithm searches all places of the Reversing Petri net and removes the effect that the transition created from their marking. Then, for every base in that place it searches for the last transition that it used it and goes back in the petri net until it finds it or reaches a transition that is empty. In case it reached the null transition, it sets the base back to the initial place. Otherwise, it looks for the input place of the transition that last used the base, and the base is saved there.

We keep the same rules t1:-t8: and we continue to define the rules of oc-enabled transitions. According to the definition of out of causal reversibility, any transition that has fired forward can be reversed:

eoc1: notenabledOC(T, TS):-transHistory(T, H, TS), H<=0.

eoc2: notenabledB(T, t):-irreversible(T).

eoc3: enabledOC(T, TS):-not notenabledOC(T, TS), trans(T), time(TS).

Thus, only transitions with 0 history and irreversible transitions cannot be reversed.

Fire transition rule is defined in the same way as in the previous two modes :

r1: {firesOC(T, TS)}:-enabledOC(T, TS), trans(T), time(TS).

Further along, we have a rule defining effect of a transition:

r2: effect(T, Q1, Q2, TS):-firesOC(T, TS), tparcbond(T, P, Q1, Q2), time(TS).

The effect of a transition is to create a bond and reversing a transition leads to the break of that bond and backtracking associated tokens. If the transition has no effect, then no change will occur other than initializing the history to 0.

r3: breakBond(P, Q1, Q2, TS):-holdsbonds(P, Q1, Q2, TS), effect(T, Q1, Q2, TS), firesOC(T, TS).

r4: placeOfTrans(P, T, TS):-firesOC(T, TS), time(TS), place(P).

r5: usingToken(T, Q, P, TS):-tparc(T, \_, Q), place(P), time(TS).

r6: usingToken(T, Q, P, TS):-tparcbond(T, \_, Q, \_), place(P), time(TS).  
 r7: usingToken(T, Q, P, TS):-tparcbond(T, \_, \_, Q), place(P), time(TS).  
 r8: usingToken(T, Q, P, TS):-tparc(T, \_, Q1), bonded(P, Q, Q1, TS).  
 r9: usingToken(T, Q, P, TS):-tparcbond(T, \_, Q1, \_), bonded(P, Q, Q1, TS).  
 r10: usingToken(T, Q, P, TS):-tparcbond(T, \_, \_, Q1), bonded(P, Q, Q1, TS).  
 r11: maxTrans(T, Q, P, TS):-H1=#max{H:usingToken(T1, Q, P, TS) ,  
 transHistory(T1, H, TS)}, token(Q), transHistory(T, H1, TS), H1>0,  
 place(P).

Rule r3 sets any bond created by a transition that is reversing in this instance to break. Rule r4 is used to refer to all places in the reversing Petri net. Rules r5-r10 find all tokens that are used from the transition either implicitly or directly. Rule r11 finds the transition with the biggest history from all transitions that use the token. History must be positive in order to consider that the last transition which used the token.

r12: moveToken(From, To, Q, TS):-holds(From, Q, TS), placeOfTrans(From, T1, TS), T1!=T2, not bonded(From, Q, \_, TS), maxTrans(T2, Q, From, TS), tparc(T2, To, \_), transHistory(T2, H, TS).  
 r13: moveToken(From, To, Q, TS):-holds(From, Q, TS), placeOfTrans(From, T1, TS), T1!=T2, not bonded(From, Q, \_, TS), maxTrans(T2, Q, From, TS), tparcbond(T2, To, \_, \_), transHistory(T2, H, TS).  
 r14: moveToken(From, To, Q, TS):-holds(From, Q, TS), placeOfTrans(From, T1, TS), not bonded(From, Q, \_, TS), maxTrans(T1, Q, From, TS), trans(T1), holds(To, Q, 0).  
 r15: moveBondOC(From, To, Q1, Q2, TS):-From!=To, holdsbonds(From, Q1, Q2, TS), not breakBond(From, Q1, Q2, TS), placeOfTrans(From, T1, TS), T1!=T2, maxTrans(T2, Q1, From, TS), tparc(T2, To, \_).  
 r16: moveBondOC(From, To, Q1, Q2, TS):-From!=To, holdsbonds(From, Q1, Q2, TS), not breakBond(From, Q1, Q2, TS), placeOfTrans(From, T1, TS), T1!=T2, maxTrans(T2, Q2, From, TS), tparc(T2, To, \_).  
 r17: moveBondOC(From, To, Q1, Q2, TS):-From!=To, holdsbonds(From, Q1, Q2, TS), not breakBond(From, Q1, Q2, TS), placeOfTrans(From, T1, TS), T1!=T2, maxTrans(T2, Q1, From, TS), tparcbond(T2, To, \_, \_).

```

r18:moveBondOC(From, To, Q1, Q2, TS):-From!=To, holdsbonds(From, Q1, Q2,
    TS), not breakBond(From, Q1, Q2, TS), placeOfTrans(From, T1, TS),
    T1!=T2, maxTrans(T2, Q2, From, TS), tparcbond(T2, To, _, _).

r19:moveBondOC(From, To, Q1, Q2, TS):-From!=To, holdsbonds(From, Q1, Q2,
    TS), placeOfTrans(From, T1, TS), maxTrans(T1, Q1, From, TS),
    holdsbonds(To, Q1, Q2, 0).

r20:moveBondOC(From, To, Q1, Q2, TS):-From!=To, holdsbonds(From, Q1, Q2,
    TS), placeOfTrans(From, T1, TS), maxTrans(T1, Q2, From, TS),
    holdsbonds(To, Q1, Q2, 0).

```

Rules r12-r20 move the bases from the output place of the reversing transition to the output place of the transition that last used it.

```

r21:addOC(P, Q, T, TS):-moveToken(_, P, Q, TS), firesOC(T, TS).

r22:addOC(P, Q, T, TS):-moveBondOC(_, P, Q, _, TS), firesOC(T, TS).

r23:addOC(P, Q, T, TS):-moveBondOC(_, P, _, Q, TS), firesOC(T, TS).

r24:delOC(P, Q, T, TS):-moveToken(P, _, Q, TS), firesOC(T, TS).

r25:delOC(P, Q, T, TS):-moveBondOC(P, _, Q, _, TS), firesOC(T, TS).

r26:delOC(P, Q, T, TS):-moveBondOC(P, _, _, Q, TS), firesOC(T, TS).

r27:break(P, Q1, Q2, TS):-breakBond(P, Q1, Q2, TS).

r28:break(P, Q1, Q2, TS):-moveBondOC(P, _, Q1, Q2, TS).

```

Any newly bonds break when transition reverses, or if they are moved, r27 and r28.

The rest of the rules are the same as in the causal and backtracking mode.

We now execute the example of figure Figure 2.8 Catalysis example in Petri nets with initial marking:

```

holds(u, c, 0) .
holds(w, a, 0) .
holds(z, b, 0) .

```

**Figure 3. 6 Initial Marking for Catalysis example**

```

time(0) time(1) time(2) time(3) time(4) time(5) holds(u,c,0)
holds(w,a,0) holds(z,b,0) enabled(t1,0) enabled(t1,4)
enabled(t2,1) enabled(t2,5) fires(t1,0) fires(t1,4)
fires(t2,1) fires(t2,5) enabledOC(t1,1) enabledOC(t1,2)
enabledOC(t1,5) enabledOC(t2,2) enabledOC(t2,3) firesOC(t1,2)
firesOC(t2,3) createBond(x,c,a,t1,0) createBond(x,c,a,t1,4)
createBond(y,a,b,t2,1) createBond(y,a,b,t2,5)
holdsbonds(x,c,a,1) holdsbonds(x,c,a,5) holdsbonds(y,a,b,2)
holdsbonds(y,a,b,3) holdsbonds(y,a,b,6) breakBond(y,a,b,3)
holdsbonds(y,c,a,2) holdsbonds(y,c,a,6) breakBond(y,c,a,2)
holds(u,c,3) holds(u,c,4) holds(w,a,4) holds(x,a,1)
holds(x,a,5) holds(z,b,1) holds(z,b,4) holds(z,b,5)
add(x,c,t1,0) add(x,c,t1,4) add(y,a,t2,1) add(y,a,t2,5)
add(x,a,t1,0) add(x,a,t1,4) add(y,b,t2,1) add(y,b,t2,5)
add(y,c,t2,1) add(y,c,t2,5) holds(x,c,1) holds(x,c,5)
holds(y,a,2) holds(y,a,3) holds(y,a,6) holds(y,b,2)
holds(y,b,3) holds(y,b,6) holds(y,c,2) holds(y,c,6)
delOC(y,c,t1,2) delOC(y,b,t2,3) delOC(y,a,t2,3) del(x,a,t2,5)
del(x,a,t2,1) del(x,c,t2,5) del(x,c,t2,1) del(w,a,t1,0)
del(u,c,t1,0) del(z,b,t2,1) addOC(z,b,t2,3) addOC(w,a,t2,3)
addOC(u,c,t1,2) del(z,b,t2,5) del(w,a,t1,4) del(u,c,t1,4)

```

**Figure 3. 7 Out-of-causal execution**

By executing the example for 5 time instances we get 8 different solutions in total.

The first transition to fire in the solution above, is t1 as it is the only transition enabled. We can see that token a and c are moved to place x and a bond is created between them. The new marking sets t2 enabled, which fires. Token a is labelled on the arc and is moved along with token c because they are bonded to place y. Token b is moved as well at place y and token a and b are bonded. So, now we have at place y, token a, b and c all bonded. At time instance 2, t1 will reverse, although it is not the last transition that is executed and although it has causally dependent transitions that have not reversed yet. The result is to break the bond that was created by transition t1, namely bond a-c. This should take c back to the initial marking, (addOC(u, c, t1, 2)) which is exactly what happens, but should leave token a in y because it is bonded with token b. So now, the catalysis example is solved. We took the token c which is present just to help with bonding tokens a and b back to its initial place, and we are left with bond a-b.

### 3.6 Properties on Reversing Petri nets

In subchapter 2.3 Petri nets, there are some interesting behavioural properties that are useful to check on Petri nets. Some of those properties, could be applied and used in

Reversing Petri nets as well. We redefine them to better match Reversing Petri nets and implement programs in ASP to help us identify them in different Reversing Petri nets.

### *I. Reachability*

Reachability for a marking is the problem of finding whether a marking  $M_n$  is reachable in a Reversing Petri net  $P$ , with initial marking  $M_0$ ,  $\langle P, M_0 \rangle$ . A marking  $M_n$  is said to be reachable in  $\langle P, M_0 \rangle$  if there exists a sequence of firings and reverses of transitions that transforms  $M_0$  to  $M_n$ ,  $\sigma = M_0 T_1 M_1 T_2 M_2 T_3 \dots T_N M_n$ . The set of all markings that are reachable by  $\langle P, M_0 \rangle$ , is denoted by  $L(P, M_0)$ .

Reachability for a transition is the problem of finding whether a transition can fire in a Reversing Petri net  $P$ , with initial marking  $M_0$ . A transition  $t$  is said to be reachable in  $\langle P, M_0 \rangle$  if there exists a sequence of firing and reverses of transitions that lead to a state where  $t$  is enabled.

We can find if a state is reachable by restricting the program. If we want to reach a state where token  $t$  is present in place  $pt$  and token  $t2$  is present in place  $pt2$ . We will define the rule:

- r1: atAll:-not holds(pt, t, \_) , not holds(pt2, t2, \_).
- r2: sameTime:-holds(pt, t, T) , holds(pt2, t2, T), time(T).
- r3: :-not atAll, not sameTime.
- r4: :-atAll.

Rules r1&r4 state that we have a deadlock, if we cannot produce any of the two states we want. Rule r2 states we want them to hold at the same time instance, while r3, states, that we have a deadlock, if we can produce both of them but not at the same time instance. The rules should change to match the requirements we have. Execution will fail and will produce unsatisfiable, if no path is found to satisfy all rules.

When we want to see if a transition  $t$  can be reach we can just write the following rule:

- r5: :-not fires(t, \_).

Which means that we have a deadlock if  $t$  does not fire.

### *II. Liveness*

Liveness is defined the same as in forward-only Petri nets.

We can implement rules to explain Dead Transitions, L1-live and L2-live.

Dead Transitions:

d1: notdead(T):-enabled(T, TS).

d2: dead(T):-not notdead(T), trans(T).

A transition is not dead if it is enabled, it is not necessary that it fires. A transition is dead if we cannot prove that it is not dead.

L1-live:

d3:l1(T):- enabled(T, TS).

A transition that is enabled at any time, is said to be L1-live.

L2-live:

If we want a transition to be enabled at least 4 times:

l2(T):-enabled(T, TS), enabled(T, TS1), enabled(T, TS2), enabled(T, TS3),  
TS!=TS1, TS!=TS2, TS!=TS3, TS1!=TS2, TS1!=TS3, TS2!=TS3.

Which is explained as, the transition T is enabled at 4 different time instances.

The rest Liveness properties, could not be implemented in time and are left for future work.

### *III. Persistence*

Again the definition is not changed between the two types of Petri nets.

p1: useTheSame(T1, T2):-ptarc(P, T1, Q), ptarc(P, T2, Q).

p2: useTheSame(T1, T2):-ptarc(P, T1, Q), ptarcbond(P, T2, Q, \_).

p3: useTheSame(T1, T2):-ptarc(P, T1, Q), ptarcbond(P, T2, \_, Q).

p4: useTheSame(T1, T2):-ptarcbond(P, T1, Q, \_), ptarcbond(P, T2, Q, \_).

p5: useTheSame(T1, T2):-ptarcbond(P, T1, Q, \_), ptarcbond(P, T2, \_, Q).

p6: useTheSame(T1, T2):-ptarcbond(P, T1, Q, \_), ptarc(P, T2, Q).

p7: useTheSame(T1, T2):-ptarcbond(P, T1, \_, Q), ptarc(P, T2, Q).

p8: directly(T1, T2):-tparc(T1, P, \_), ptarc(P, T2, \_).

p9: directly(T1, T2):-tparc(T1, P, \_), ptarcbond(P, T2, \_, \_).

p10: directly(T1, T2):-tparcbond(T1, P, \_, \_), ptarc(P, T2, \_).

p11: directly(T1, T2):-tparcbond(T1, P, \_, \_), ptarcbond(P, T2, \_, \_).

p12: causally(T1, T2):-directly(T1, T2).

p13: causally(T1, T2):-directly(T1, T3), causally(T3, T2).

p14: persistence(T1, T2):-trans(T1), trans(T2), not useTheSame(T1, T2), not  
persistence(T2, T1), not causally(T1, T2), not causally(T2, T1).

So, two transitions are persistent if they are not causally dependent, and they do not use the same token from the same place.

#### IV. *Home State and Reversibility*

A home state is a state that can be reached from any other state of the Reversing Petri net. Therefore, we can define a home State as the state that is reachable, with all transitions that fire to produce the state are reversible. As in reachable property, if we want to reach a state where token  $t$  is present in place  $pt$  and token  $t2$  is present in place  $pt2$ . We will define the rules:

p15: atAll:-not holds(pt, t, \_) , not holds(pt2, t2, \_).

p16: sameTime:-holds(pt, t, T) , holds(pt2, t2, T), time(T).

p17: :-not atAll, not sameTime.

p18: :-atAll.

p19: :-fires(T,TS),irreversible(T).

### 3.7 Shortest Path

The property of reachability is often accompanied with the shortest path problem. This is actually the problem of finding the smallest route leading to the reachable state. Clingo supports Incremental mode[26], in which the search does not stop until it finally satisfies the goal state, or else the reachable state. If the goal state is not reachable the execution will continue infinitely. Therefore, it is necessary to ensure that the goal state is indeed reachable from the initial marking and type of execution.

Some changes were required in order for the incremental mode to successfully run the algorithms. A significant change, is that time is no longer bounded, as the program increments the search until it satisfies the goal.



In incremental mode the initial mode is defined in a different format. Here is an example of the new way to define the initial marking:

```
i1: init(rp, r).  
i2: init(fp, f).  
i3: init(pp, p).  
i4: init(mp, m).  
i5: init(ep, e).
```

Firstly, incremental mode syntax, demands that the program is divided into three sections:

```
#program base
```

Program base section includes all facts, like the Petri net's figure and the rules for the initial marking of the Petri net.

```
same(X, X):-token(X).  
holds(P, Q, 0):-init(P, Q).  
holdsbonds(P, Q1, Q2, 0):-initbonds(P, Q1, Q2).
```

The second section is:

```
#program step(t).
```

The program step includes all rules that are performed in every incremental step and is consisted of all the rules of the program.

The final section defines the goal:

```
#program check(t).  
:-query(t), goal_state_declaration.
```

In the final section, the user needs to define the goal state in the above format. e.g. if we are looking for the shortest state for transition a to fire we write the following:

```
#program check(t).  
:-query(t), not fires(a, t).
```

Which states that we don't want transition a not to fire.

This mode of execution in Clingo, will give us the shortest path to reach our goal or will continue infinitely if the goal cannot be reached at all.

# Chapter 4

## Tool

### Contents

---

4.1	PETRI NET OBJECT.....	54
4.2	CREATE NEW PETRI NET .....	55
4.3	EXECUTE IN CLINGO .....	56
4.4	INITIAL MARKING .....	56
4.5	PROVE NEW STATES .....	57
4.6	PROPERTIES.....	58
4.7	RESULTS .....	59

---

As we can see from the examples included above, the solutions we get from Clingo, are not so clear and easy to understand. Therefore, a tool is created to facilitate the user interact with the tool and better understand the output. The tool allows users to create a new petri net in the format explained in 3.1 Reversing Petri Net Structure, set an initial marking for the petri net, run the Petri net with all 4 modes, namely forward execution, Backtracking, Causal and non-Causal, for a duration they select, to check properties of the Reversing Petri net, and finally to see the results in a clearer way. The tool is created in Java Swing and we can see the initial window in the Figure 4.1 Initial Window.

#### 4.1 Petri net Object

There is a class implementing an object Petri net which contains three arrayLists of Strings for places, transitions and tokens. Every time a file is read containing a Petri net, a new Petri net object is created. This helps when information from the file is needed.

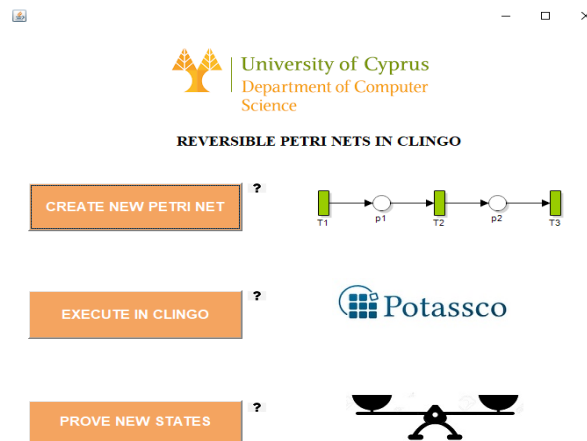


Figure 4.1 Initial Window

## 4.2 Create New Petri net

As seen in **Error! Reference source not found.**, a user can create a new Petri net from the tool, by inserting the specifications of the Petri net. It is necessary to define all the places, transitions and tokens of the Petri net. Later on, the user can define the arcs of the Petri net by selecting the from place/transition and to transition/place and labelling them with tokens or bonds of tokens. When the user has defined all necessary

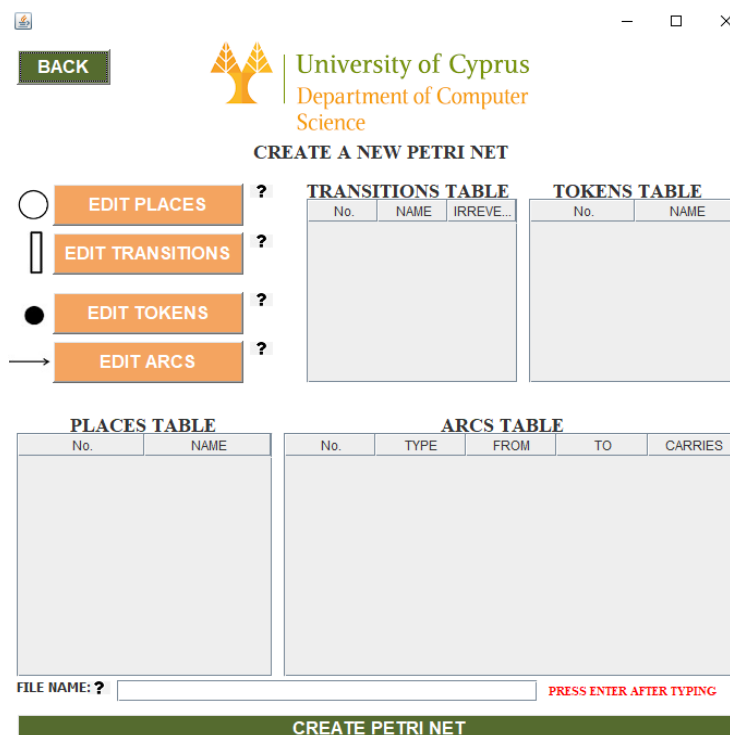
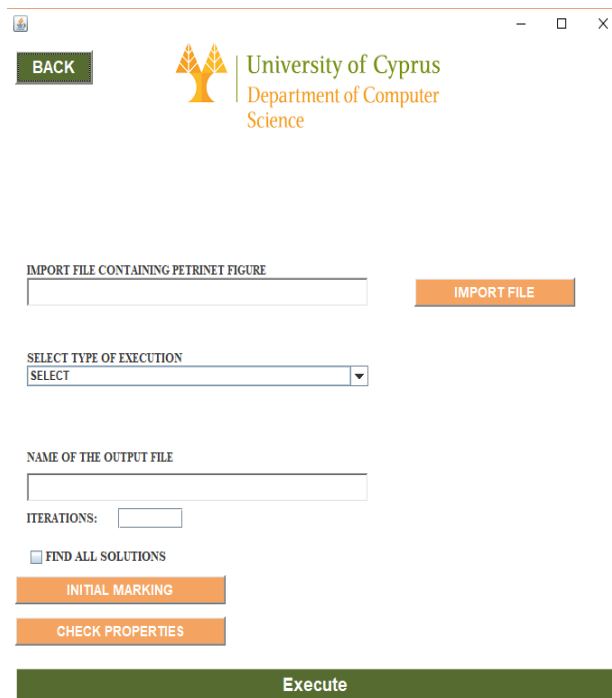


Figure 4. 2 Create new RPN screen

information, he/she can save the Petri net Structure in a file with extension *.lp* in the folder LPfiles.

### 4.3 Execute in Clingo

In order to execute a Petri net in the programs of Clingo in any of the forms referred in Chapter 3, the user has to select a Reversing Petri net. Furthermore, there 4 types of executions, namely Forward, Backtracking, Causal and Out of Causal execution. The user also has to enter a duration for the execution and an initial marking.



**Figure 4. 3 Execute Window**

There is also the option to find all solutions produced from the execution. This is done by running the clingo command with a 0 in the end. There is also the choice for checking properties in the Petri net. This is further explained in following subchapter.

### 4.4 Initial Marking

Once user have selected the figure of the Petri net, the next step is to import initial marking either by importing a file that contains it or create a new one. Again the new marking will be created based on the Petri net's figure. Further than the initial marking defined by the user, Petri net's transitions are set to have zero history, unless specified by user to another value. Initial marking screen is demonstrated by

BACK

University of Cyprus  
Department of Computer Science

**PROPERTIES**

REACHABILITY

HOME STATE

PERSISTENCE

TRANSITIONS' LIVENESS

SHORTEST PATH

DELETE

SUBMIT

Figure 4. 4 Properties

## 4.5 Find New States

The tool offers the user, the opportunity to compare non-causal-execution with the

BACK

University of Cyprus  
Department of Computer Science

**INITIAL MARKING**

☐ FILE ALREADY EXISTS

NEW MARKING

PLACE:

TOKEN:

BOND:

ADD

ADD

DELETE

DONE

other three forms and see if any new states can be produced. When a user selects the figure file and the mode of execution to compare, the program runs the Clingo command producing all possible outputs with that mode of execution. When the command is finished, the program runs the Clingo command again but this time with non-causal-execution. Later, each state present in non-causal output file, is searched in the other output file. The program stops searching as soon as it finds one state that cannot be produced. Finding all states would be very difficult because non-causal execution produces a large output file, and processing that file is time-consuming. The initial marking and time of execution given by the user may produce executions that don't produce new states in non-causal execution. For example, if the program is only allowed to run for 1 time instance. The user should be aware of this when selecting the initial marking and duration of execution.

## 4.6 Properties

Properties supported by the tool are:

*Reachability:* Where user defines the transition, or state that wants to check. The tool, runs the form of reversibility selected in the *Executing Window* by restricting it to only include solutions that satisfy the reachability state/transition. If the execution does not produce the goal state/transition it returns Unsatisfiable .

*Home State:* The user can select the property Home State, and either enter a state to check if it can be reached or find all States that are Home States. Any state is considered Home State, if it is produced by a sequence of transitions that no irreversible firings occur.

*Persistence:* The user selects the transitions to check for persistent. Persistence is a property that depends only on the structure of the figure and thus, independent of the type of execution. The first step completed by the system is to run a command in Clingo to verify or decline the existence of persistence between the two transitions. Right after, the command for the type of execution selected in the *Execute Window* will run. The user also has the chance to check all transitions for persistence.

*Transitions' Liveness:* Tool offers users the chance to check the figure for dead, 11-live and 12-live transitions. The files containing the rules of Liveness will be added to

the command, and thus all transitions that have this property will be presented in the results window.

*Shortest Path:* as mentioned above, we often need to know the shortest path that we can follow to reach a certain state/transition. The tool allows the user to search for the shortest path of a state or transition. There is no need to check manually if the state can be reached from the initial marking and type of execution of the Petri net, because the tool checks before running the shortest Path algorithm. When a user searches the shortest path to reach a state, the tool creates the new initial file based on the old one, and then run a command for reachability of the goal state. If the execution terminates successfully then the shortest path algorithm is run next. Otherwise, if the result of the execution is unsatisfiable, then shortest path algorithm is not run, and thus we have prevented the system from getting in an infinite loop.

## Results

The most important feature of the tool is that it presents the solution to user in a more understandable way. Having a great algorithm but not being able to make sense of the results is useless. Thus, we create a class, that reads the output file of Clingo, and splits the results in time steps. It presents one time-step and user can move forward and backward in time watching the state of the Petri net change while transitions fire forward and backward. This is done by separating the output in lists of categories, collecting all holds together, all fires together, etc. Then, for each time step, only the predicates relative to this time instance are presented. All properties selected by user will be presented here as well, along with the result “Satisfied” or “Unsatisfied”.

# Chapter 5

## Case study

### Contents

---

5.2	FORWARD EXECUTION.....	60
5.3	BACKTRACKING .....	63
5.4	CAUSAL REVERSING .....	66
5.5	OUT OF CAUSAL REVERSING .....	67

---

This chapter is devoted to check and prove the correctness of the algorithms created, as well as to demonstrate their effectiveness. Therefore, we will execute the example of the Petri net in Figure 5. 1 ERK Petri net in all forms of execution of Reversing Petri nets and compare the results.

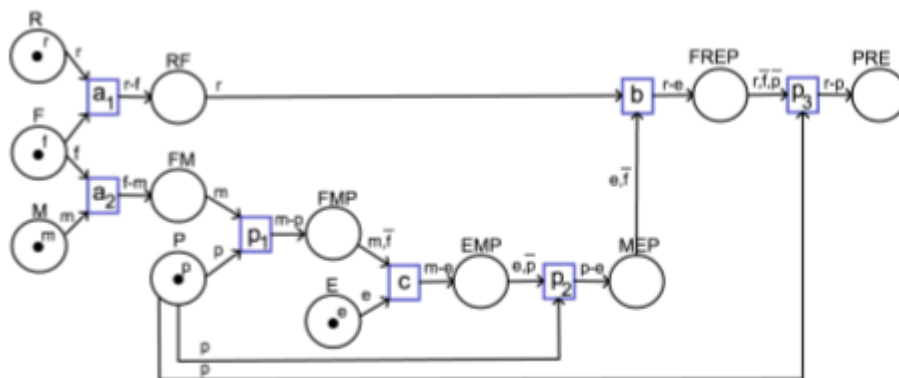


Figure 5. 1 ERK Petri net

### 5.1 Forward execution

Firstly, we execute the ERK reversing Petri net in the forward mode of execution. Initially, we provide all places that have no input will tokens. Therefore, at the initial



marking, place R holds the token r, place M holds the token m, place F holds token f, place P holds token p and place E holds token e. We ask the tool to provide all possible solutions and execute the reversing Petri net for 12 time instances, as shown Figure 5. 3.

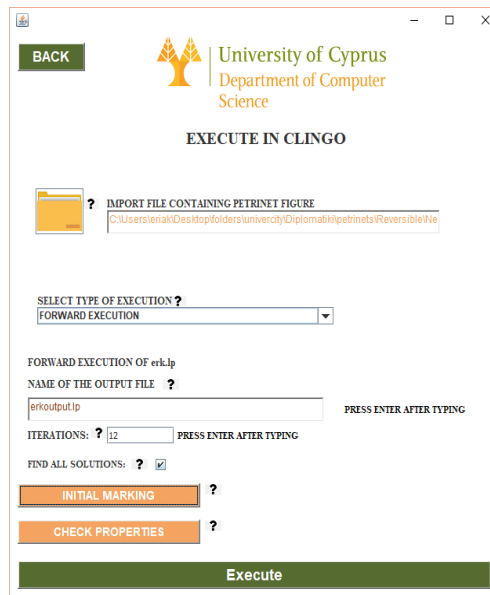


Figure 5. 3 Executing the ERK

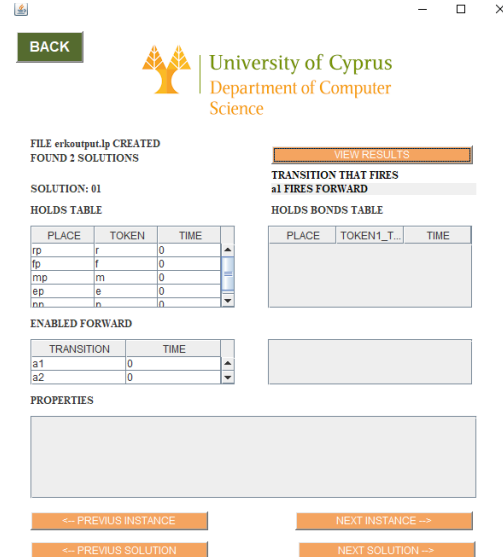


Figure 5. 3 Results first solution, time instance 0

We can see at the upper left of the screen in Figure 5. 3 that only two solutions are found. This is because at the first instance, we have two enabled transitions. The transitions are a1 and a2. Therefore, we have a solution that starts by firing transition a1, and a solution that starts by firing a2.

The first solution of the results can be seen in Figure 5. 3 and Figure 5. 4. The transition a1 is the first to fire, causing the creation of the bond r-f at place rfp. Place rp

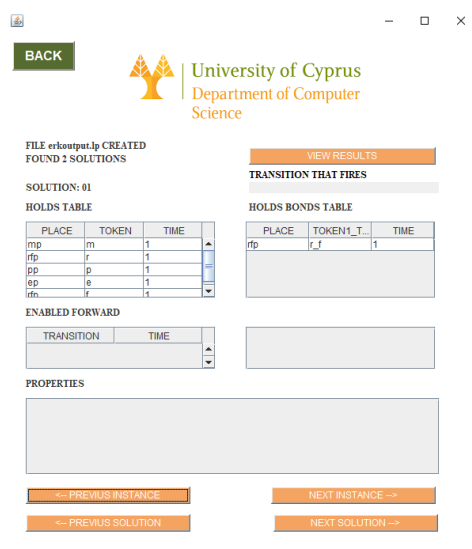


Figure 5. 4 Results first solution, time instance 1

looses the token r and place fp loses token f. Both tokens will be transferred to place rfp where the bond is created. The rest of the marking remains the same. There are no more enabled transitions, because a2 requires the token f, which is currently at place rfp, while transition b requires token e from mep, that is not present.

In order for the execution to continue, the token f should be transferred back to the initial marking so that transition a2 can fire. Forward

execution though, doesnot allow reversing the transition and thus, the marking will remain the same until the end of the simulation.

The second solution, moves just a little further, allowing two transitions to fire.

The results can be seen in Figure 5. 7 and Figure 5. 7. We again have the initial marking and the enabled transitions, a1 and a2. The first transition that fires is transition a2, as expected. Transition a2, causes the tokens f and m, to move from place fp and mp

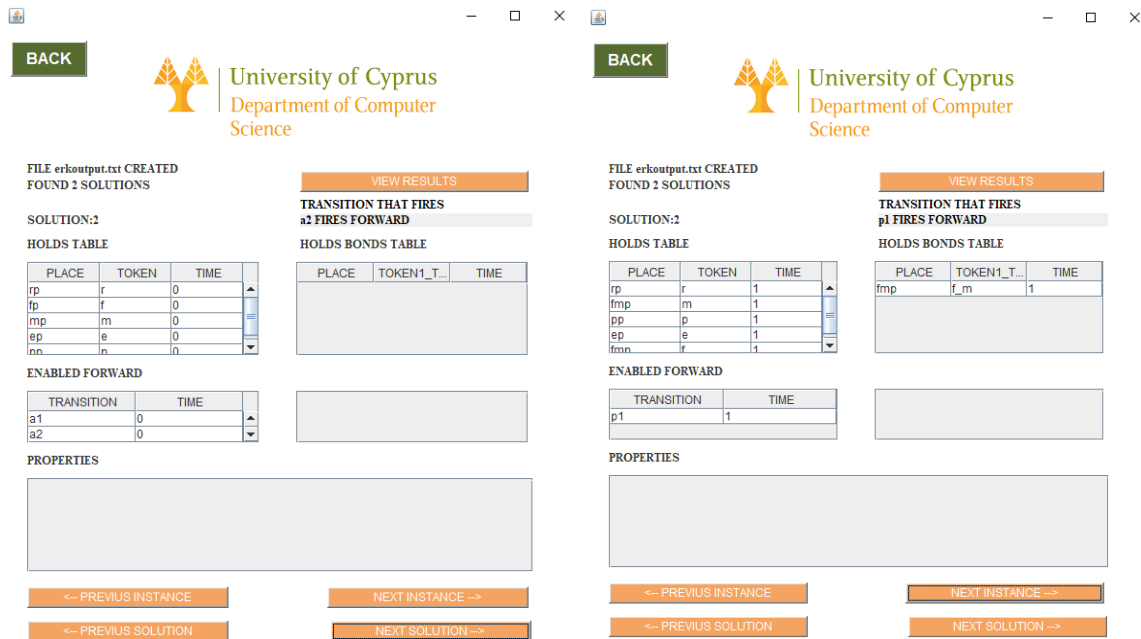


Figure 5. 7 Second solution, time instance 0

Figure 5. 7 Second solution, time instance 1

respectively, to place fmp and bond. This is demonstrated in screen in Figure 5. 7 in table holds and holds bonds. The new marking causes transition p1 to be enabled, as both input places contain all necessary tokens/bonds. Transition p1 fires at time instance 1 and the marking is changed as demonstrated in Figure 5. 5. Place pp loses token p and fmp loses tokens f and m. All three tokens are transferred to place fmpp, where token m is bonded with p. The bond that exists between m and f will be transferred aswell. In the new marking no transition is enabled, because transition c, which is the next in sequence after transition p1, requires the

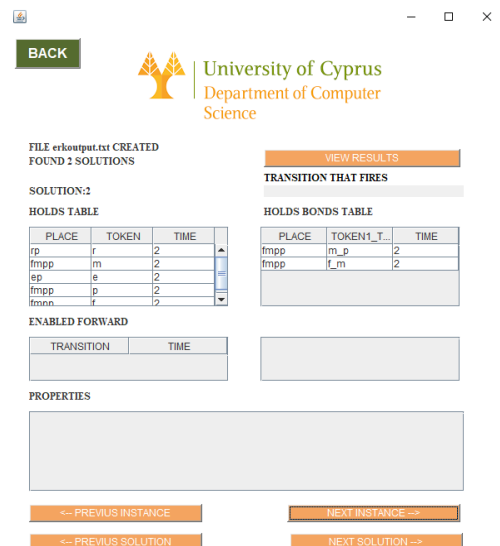


Figure 5. 5 Second solution, time instance 2

removal of the token f. The transition a1 requires token f that is currently in place p1. Forward execution does not allow the reverse of transitions, and therefore, token f cannot be moved from p1 to a1 to enable them. Thus, the marking remains the same for the rest of the simulation.

We now look at the time the clingo tool required to calculate the results. In total, the program run for 0.30 seconds. Solving the problem required 0.04 seconds while the first solution was found in 0.02 seconds. The tool also reports the time spent on models that were unsatisfiable. The tool is quick when executing in forward mode. A simulator created a programming language, other than logic, would require more time to find all paths.

## 5.2 Backtracking

In Figure 5. 9, we execute the same reversing Petri net as in forward execution, for the same time duration and with the same initial marking and asking again the system to find all possible paths.

**EXECUTE IN CLINGO**

BACK

University of Cyprus  
Department of Computer Science

EXECUTE IN CLINGO

IMPORT FILE CONTAINING PETRINET FIGURE  
C:\Users\erik\Desktop\Folders\university\Diplotaki\petrinets\ReversibleNe

SELECT TYPE OF EXECUTION ?  
BACKTRACKING

BACKTRACKING OF erk1.p  
NAME OF THE OUTPUT FILE ?  
erkoutput.txt

ITERATIONS: ? 12 PRESS ENTER AFTER TYPING

FIND ALL SOLUTIONS: ? ☒

INITIAL MARKING ?

CHECK PROPERTIES ?

Execute

**Backtracking results window**

FILE erkoutput.txt CREATED  
FOUND 610 SOLUTIONS

SOLUTION: 01

HOLDS TABLE

PLACE	TOKEN	TIME
rp	r	0
fp	f	0
mp	m	0
sp	e	0
in	n	0

ENABLED FORWARD

TRANSITION	TIME
a1	0
a2	0

PROPERTIES

VIEW RESULTS

TRANSITION THAT FIRES  
a1 FIRES FORWARD

HOLDS BONDS TABLE

ENABLED BACKWARD REVERSIBLE

<< PREVIOUS INSTANCE

NEXT INSTANCE >>

<< PREVIOUS SOLUTION

NEXT SOLUTION >>

Figure 5. 9 Backtracking execution

Figure 5. 9 Backtracking results window

We now get 610 solutions, because we can move forward and backward through the same states of the reversing Petri nets. The solutions are calculated in 1.01 seconds and the first model was calculated from the first 0.01 second.

We move forward to prove that the states that are created are the same as in forward mode, although the transitions are executed in different sequences, creating different solutions. The amount of solutions is big, so we will not search each one of them to see if any new states are created. We will try to prove that no new states are created. We will do this by proving that both transitions b and c are not reachable, and therefore,

**PROPERTIES**

PROPERTY TYPE	PROPERTY
Reachable	Trans c

REACHABILITY ?  
HOME STATE ?  
PERSISTENCE ?  
TRANSITIONS' LIVENESS ?  
SHORTEST PATH ?

All the properties entered in properties table, all need to hold in order to get results. If any of the properties does not hold, then the result of the execution will be unsatisfiable, and there is no way to know which property was not satisfied. Therefore, it is best for the user to check one property at each execution, unless he/she wants to check the compination of the properties.

DELETE

SUBMIT

Figure 5. 11 Check if transition c is reachable

FILE erkouput.txt CREATED  
NO SOLUTIONS FOUND

VIEW RESULTS

TRANSITION THAT FIRES

HOLDS TABLE

HOLDS BONDS TABLE

ENABLED FORWARD

ENABLED BACKWARD REVERSIBLE

PROPERTIES

← PREVIOUS INSTANCE      NEXT INSTANCE →  
← PREVIOUS SOLUTION      NEXT SOLUTION →

Figure 5. 11 No solutions found when trying to prove transition c reachable.

only a1,a2 and p1 fires, causing the same states as in forward mode.

We can see in Figure 5. 11, the window with the property of reachability for trans c, and in Figure 5. 11 that the execution fails. Therefore, the tool cannot find any path that

**PROPERTIES**

PROPERTY TYPE	PROPERTY
Reachable	Trans b

REACHABILITY ?  
HOME STATE ?  
PERSISTENCE ?  
TRANSITIONS' LIVENESS ?  
SHORTEST PATH ?

All the properties entered in properties table, all need to hold in order to get results. If any of the properties does not hold, then the result of the execution will be unsatisfiable, and there is no way to know which property was not satisfied. Therefore, it is best for the user to check one property at each execution, unless he/she wants to check the compination of the properties.

DELETE

SUBMIT

Figure 5. 13 Check if transition b is reachable

FILE erkouput.txt CREATED  
NO SOLUTIONS FOUND

VIEW RESULTS

TRANSITION THAT FIRES

HOLDS TABLE

HOLDS BONDS TABLE

ENABLED FORWARD

ENABLED BACKWARD REVERSIBLE

PROPERTIES

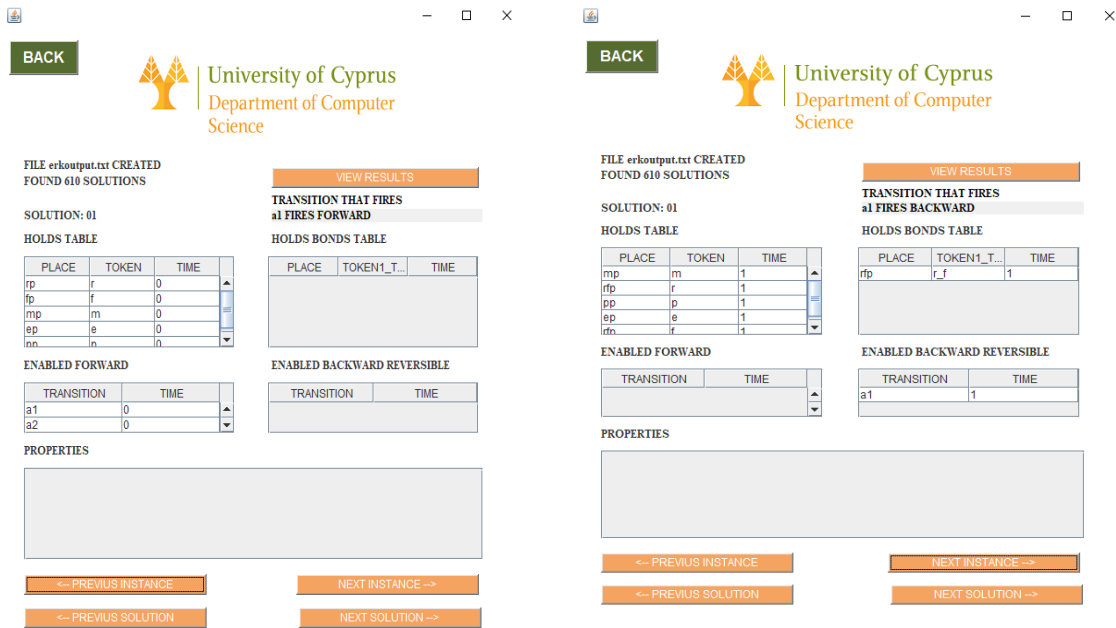
← PREVIOUS INSTANCE      NEXT INSTANCE →  
← PREVIOUS SOLUTION      NEXT SOLUTION →

Figure 5. 13 Proving that transition b is not reachable

allows the execution of the transition c. The same happens with transition b as demonstrated in Figure 5. 13 and Figure 5. 13.

Below, we show why transitions b and c are not reachable, although Backtracking reverses transitions. We do this by explaining one of the solutions produced from Backtracking. The solution found is demonstrated in Figure 5. 15.

We can see the initial marking, and the enabled transitions a1 and a2. In the first instance, a1 fires in a forward mode. The new marking is presented in Figure 5. 15.



**Figure 5. 15 Solution of Backtracking  
time instane 0**

**Figure 5. 15 Solution Backtracking time  
instance 1.**

We can see that the table of enabled transitions in time instance 1, is empty, signifying that no transition is enabled to fire forward. There is although a transition that is enabled to backtrack, namely transition a1, which is the last transition to be executed forward. The only way for the execution to continue, is to reverse transition a1. No other transition can be reversed, because backtracking allows the reverse of only the last transtioin that is executed in a forward mode. Thus, a1 is reversed. This will lead us in the same state with the initial stat, having enabled transitions a1 and a2 and marking the same as the initial marking. Time instance two can be seen in Figure 5. 17. The tool now decides to execute transition a2, moving tokens f and m to place fmp and bonding them. The new marking enables transition p1 and disables transition a1. Transition p1 will transfer bonded tokens m and f from place fmp to place fmpp, and

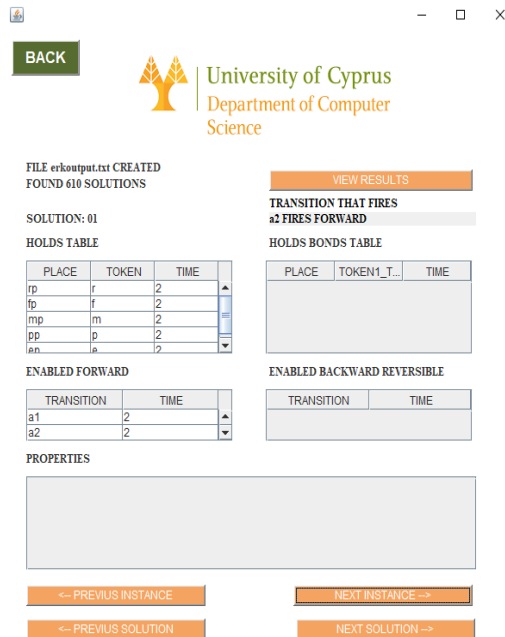


Figure 5. 17 Backtracking solution time  
instance 2

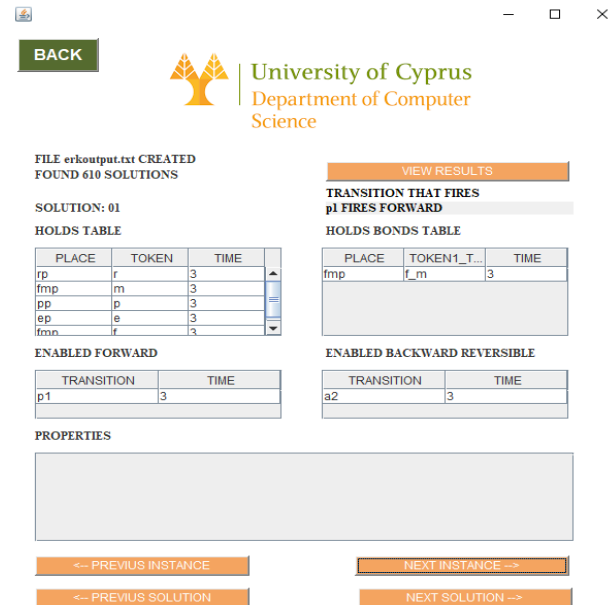


Figure 5. 17 Backtracking solution time  
instance 3

token p from place pp to place fmp, and bond token m with token p like in **Error! Reference source not found.** The marking now will be as follows:

place fmp contains tokens f, m and p, where m will be bonded with f and p will be bonded with m,

place rp will contain token r, and

place ep will contain token e.

From this state, we have not transition enabled to fire forward because we need token f to move back to place fp. This requires the reverse of the transition a2, which bonded tokens m and f, but transition a2 is not enabled to backtrack because it is not the last transition to have fired forward. As we can see, only p1 is allowed to backtrack, which will take us back to the previous state.

By looking at the solution, we can understand that there is no way to reach transition c and b, if we do not allow the reversing of transitions in out of causal mode.

### 5.3 Causal reversing

We execute the same reversing Petir net in a causal mode Figure 5. 19, and we get 610 solutions Figure 5. 19. The number of solutions is the same as in backtracking mode, because we don't have persistent transitions that can be executed in the same time. Therefore, causal reversing will produce the same solutions as in backtracking, and thus no new states are produced.

Figure 5. 19 Causal Execution

Figure 5. 19 Causal Execution results

## 5.4 Out of Causal reversing

We can move to executing the example in an out of causal order, as seen in Figure 5. 20. This time, we do not ask the system to calculate all possible solutions. Non causal execution produces a large output file with thousands of solutions. The file requires time to be produce. Running manually the file in the clingo tool, produced 13039 solutions and needed 40.843 seconds to calculate them.

Figure 5. 20 Causal Execution

In order to check if all these solutions contain the same solutions as in forward mode, we execute the command that finds new states, from window Prove New States

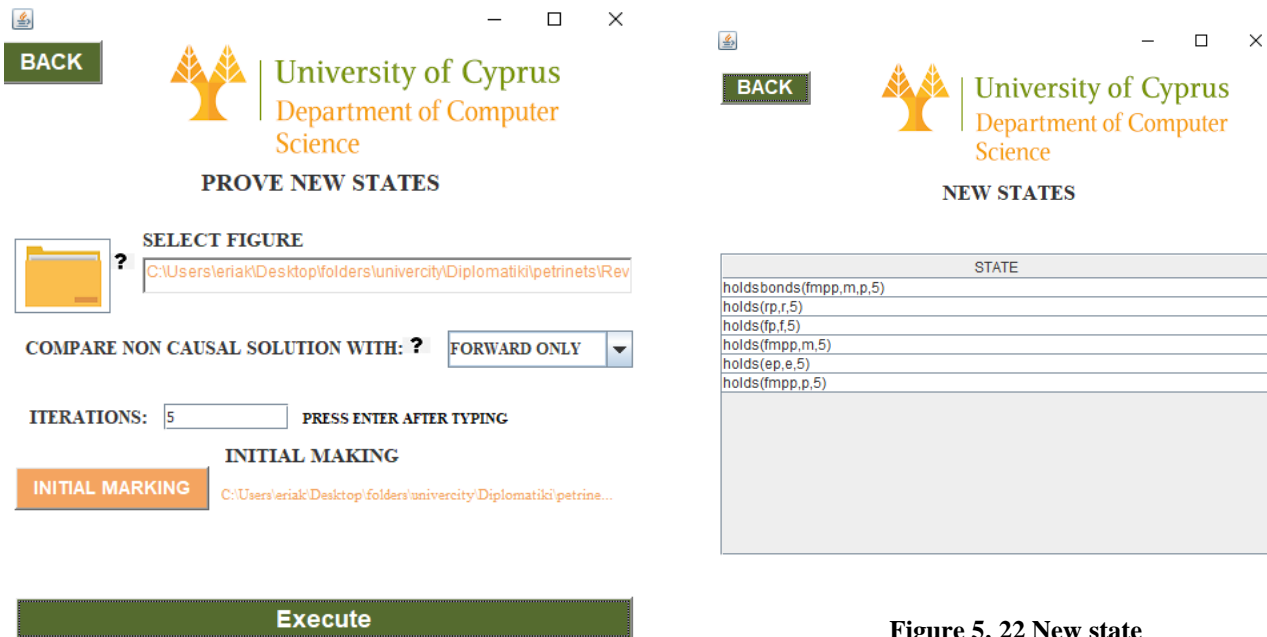


Figure 5. 22 Prove new states

Figure 5. 22 New state

as seen in Figure 5. 22. The result is seen in Figure 5. 22. The table contains the state where fmpp place, contains tokens m and token p, and token f is moved back to place f, which is exactly the states that occurs after transition a2 fires out of causal and c fires forward.

We can also see in Figure 5. 24 that we can check the property of reachability of transition p3. As we see from the results window, the property is satisfiable, and we can also see the sequence of transitions that lead to this specific state.



University of Cyprus  
Department of Computer Science

BACK

PROPERTIES

REACHABILITY ?

HOME STATE ?

PERSISTENCE ?

TRANSITIONS' LIVENESS ?

SHORTEST PATH ?

All the properties entered in properties table, all need to hold in order to get results. If any of the properties does not hold, then the result of the execution will be unsatisfiable, and there is no way to know which property was not satisfied. Therefore, it is best for the user to check one property at each execution, unless he/she wants to check the compination of the properties.

DELETE

SUBMIT

PROPERTY TYPE	PROPERTY
Reachable	Trans p3

University of Cyprus  
Department of Computer Science

BACK

FILE erkoutput.txt CREATED  
FOUND 1 SOLUTION

SOLUTION: 01

HOLDS TABLE

PLACE	TOKEN	TIME
rp	r	0
fp	f	0
mp	m	0
ep	e	0
nn	n	0

ENABLED FORWARD

TRANSITION	TIME
a1	0
a2	0

PROPERTIES

PROPERTY	SATISFIED/UNSATISFIED
ReachableTrans p3	SATISFIABLE

<- PREVIOUS INSTANCE

NEXT INSTANCE ->

VIEW RESULTS

TRANSITION THAT FIRES

a1 FIRES FORWARD

HOLDS BONDS TABLE

NON CAUSALY REVERSIBLE

Figure 5. 24 Results of the execution

Figure 5. 24 Check transition p3 for reachability

# Chapter 6

## Conclusion

---

6.1	SUMMARY .....	70
6.2	CHALLENGES .....	70
6.3	FUTURE WORK.....	71

---

### 6.1 Summary

This diploma thesis is concerned with the programming of RPNs in ASP. Algorithms for *forward*, *backtracking*, *causal reversing* and *non-causal reversing* are developed. When studying the executions of Petri nets, we discovered that ASP is helpful for properties of RPNs, like *reachability*, *home State*, *persistence* and *shortest Path*. Additional programs to implement the properties mentioned were developed. These algorithms can be run from the command line entering all files necessary, and produce the results, which the user can process for better understanding.

ASP is very strict in structure and semantics and mistakes can be easily made. Typing mistakes can lead to unexpected results and are often difficult to find. To minimize the number of user's error, a tool is created. Most of the files are done automatically from the tool, so misspelling a predicate is no longer a problem. Nevertheless, the user should still be careful to enter the correct data in the tool. Finally, the results are processed by the tool and presented in a clear format to the user.

### 6.2 Challenges

Throughout the process of implementing this diploma thesis, I came across several difficulties. Some I managed to overcome, while others, due to the lack of time, no solution is found.

The first months, were devoted to understand PNs, RPNs and ASP, as all subjects were unknown to me before. Related articles and books were studied extensively, to capture the changes of PNs, that where necessary to take place in order for them to become reversible. ASP was studied later. In order to understand the structure, capabilities and boundaries of the ASP, I created small programs. Again, books and especially Clingo's guide where helpful. Next, I tried to understand the forward execution of PNs in ASP that was developed by [15].

A particular difficulty I came across with, is developing the forward mode of execution of RPNs. Due to the absence of lists, or any other similar data structure in ASP, representing the tokens owned by a place, can no longer be represented by only one holds predicate, like in [15]. Multiple predicates are necessary to do this. Adapting the program to this, required many changes from the original program. In addition, history, bonds and negative token/bond is completely absent in the ordinary PNs, and thus, had to be added.

Furthermore, ASP can produce a massive output file with out-of-causal reversibility. This can be restricted with the use of properties like reachability or deadlock. Solutions that are executing in a cycle should be restricted to minimize the number of output data. The most difficult part of this procedure, was tracking deadlocks as a property of RPNs. One way to achieve this was by implementing a program to locate traps and sythons. Again, due to the fact that lists' are not supported by ASP, no way to refer to the set of predecessors and successors was found. Therefore, deadlock is not an implemented property of this diploma thesis.

### **6.3 Future Work**

As shown by all the work done, ASP is a very useful tool. It can help us save time when studying properties, as well as finding all paths produced by an execution. This is impossible to do with programming languages. As we saw in the example above, some executions may produce thousands of solutions, and calculating each solution individually by a simulator will either take time or can not be done.

There are many improvements that can be done in the tool we created, so that it will become more useful. The first thing is to adapt the programs created to support reversing Petri nets with cycles, as well as tokens that are repeated. An improvement

that could be of great use, is the implementation of a program that can track deadlocks. This will minimise the amount of solutions produced, as many of these solutions contain deadlocks, or the system moves forward and backward in states already visited. Additionally, L3 and L4- liveness on transitions would be useful to implement. Also, the tool could be expanded, with algorithms that further process the output of the clingo tool or even compare different solutions from clingo, to check if a state is reachable from each and every path. Finally, the tool could be expanded in order to be used for model checking.

## References

- [1] J. Timler and C. S. Lent, Maxwell's demon and quantum-dot cellular automata, vol. 94, (2), (JOURNAL OF APPLIED PHYSICS), 15 JULY 2003, pp. 1050-1060, 2003.
- [2] M. Gebser et al, Answer Set Solving in Practice. Morgan & Claypool Publishers, 2012.
- [3] M. P. Frank, "Back to the Future: The case of Reversible Computing," 2018.
- [4] Y. Lecerf, Logique mathématique : I. machines de turing réversibles. récursive insolubilité en  $n$  [epsilon]  $N$  de l'équation  $u=0n$   $u$ , ou  $0$  est un isomorphisme de codes ; II. récursive insolubilité de l'équation générale de diagonalisation de deux monomorphismes de monoïdes libres  $qx=[psi]$   $x$ , EURATOM, Bruxelles, Tech. Rep. EUR 518.f, 1964.
- [5] C. H. Bennet, Logical Reversibility of Computation, IBM Journal of Research and Development, vol. 17, (6), pp. 525-532, Nov. 1973, 1973.
- [6] E. F. Fredkin and T. Toffoli, Design principles for achieving high-performance submicron digital technologies, in Collision-Based Computing, A. Andrew, Ed. London: Springer, pp. 27-46, 2002.
- [7] T. Toffoli, Reversible computing, in In Proceedings of International Colloquium on Automata, Languages, and Programming, Heidelberg, pp. 632-644, 1980.
- [8] K. Psara and A. Philippou, Reversible computation in petri nets, in International Conference on Reversible Computation, Leicester, pp. 84-101, 2018.
- [9] K. Psara, Reversible Computation in Formal Models of Concurrency, unpublished.
- [10] M. Gebser et al, Potasco User Guide. (2nd ed.) University of Potsdam, 2017
- [11] K. Morita, Reversible computing and cellular automata - A survey, vol. 395, (1), , 1 April 2008, pp. 101-131, 2008.

- [12] Carl A. Petri, Communication with Automata, PhD thesis, Technische Hochschule Darmstadt, Germany, Published in *Schriftendes Instituts für Instrumentelle Mathematik* 3, pp. 1-128, University of Bonn, Germany, 1962.
- [13] K. Barylska, M. Koutny, L. Mikulski, and M. Piatkowski. Reversible computation vs. reversibility in Petri nets. In *Proceedings of RC 2016, LNCS 9720*, pp 105– 118. Springer, 2016.
- [14] K. Barylska, L. Mikulski, M. Piatkowski, M. Koutny, and E. Erofeev. Reversing transitions in bounded Petri nets. In *Proceedings of CS&P 2016*, volume 1698 of *CEUR Workshop Proceedings*, pages 74–85. CEUR-WS.org, 2016.
- [15] S. Anwar, C. Baral, and K. Inoue. Encoding Petri Nets in Answer Set Programming for Simulation Based Reasoning, volume 2. *Abs/1306.3542*, 2013.
- [16] S. R. Kosaraju. Decidability of reachability in vector addition systems. In *Proceedings of 74th Annual ACM Symp. Theory Computing*, San Francisco, May 5-7, 1982, pages 267-281, 1982.
- [17] E. W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM*, vol. 13 no.3, pages 441- 460, 1984.
- [18] R. J. Lipton. The reachability problem requires exponential space. New Haven, CT, Yale University, Department of Computer Science, Res. Rep. 62, 1976.
- [19] T. Murata, Petri nets: Properties, analysis and applications, in *Proceedings of the IEEE*, pp. 541-580, 1989.
- [20] D. G. Stork and R. J. van Glabbeek. Token-controlled place refinement in hierarchical Petri nets with application oactive document workflow. In *Proceedings of ICATPN2002, LNCS2360*, pp. 394–413. Springer, 2002.
- [21] R. J. van Glabbeek. The individual and collective token interpretations of Petri nets. In *Proceedings of CONCUR 2005, LNCS 3653*, pp. 323–337. Springer, 2005.
- [22] R. J. van Glabbeek, U. Goltz, and J. Schicke. On causal semantics of Petrinets. In *Proceedings of CONCUR 2011, LNCS 6901*, pp. 43–59. Springer, 2011.
- [23] Y. Dimopoulos, B. Nebel and J. Köhler. Encoding planning problems in non-monotonic logic programs. *Proceedings of 4th European Conference on Planning, Recent Advances in AI Planning*, France, September 24–26, 1997, Springer. pp. 273–285, 1997.

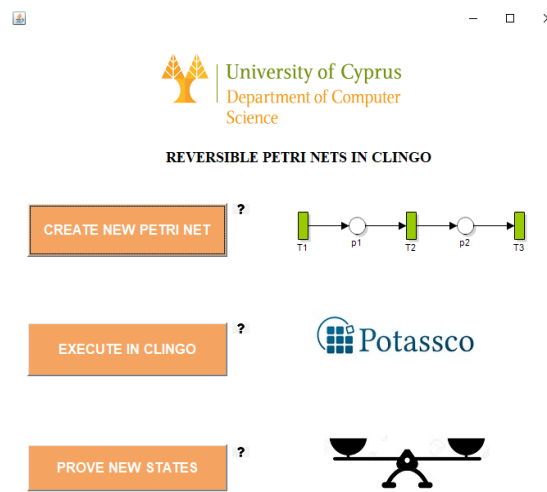
- [24] H. Buhrman, J. Tromp and P. M. Vitányi, Time and space bounds for reversible simulation, in Proceedings of the 28th International Colloquium on Automata, Languages and Programming, Springer-Verlag London, pp. 1017-1027, 2001.
- [25] M. P. Frank, Back to the Future: The case of Reversible Computing, 2018.
- [26] G. Brewka, T. Eiter and M. Truszczynski, Answer Set Programming at a Glance, vol. 54, (12), pp. 93-103, 2011.
- [27] clingo and gringo, Available: <https://potassco.org/clingo/2018>.
- [28] A. Philippou, K. Psara, Out-of-causal Order, unpublished.

# Appendix A

## User's Manual

### 1. Initial Window

The Figure I Initial window *of the tool*. User has three options. *Create a new Petri*



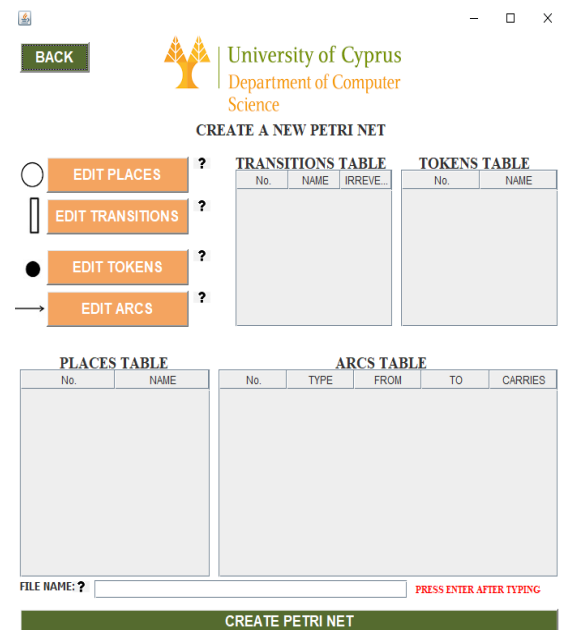
**Figure I** Initial window of the tool

*Net, execute a figure in Clingo, and prove that new states are created when running a non-causal execution of the figure.*

### 2. Create a new Petri net

When selecting the first option, *Create a new Petri net*, the screen in Figure II Create new RPN screen will appear.

By editing places, transitions, tokens, arcs, the user has the option to add places, transitions tokens and arcs in the figure. In order for the user to insert arcs, he/she should first insert places, transitions and tokens, as no arc exists without input place/transition or output transition/place or label.



**Figure II** Create new RPN screen



When the user will be able to insert arcs in the structure of the Petri net, the screen will have green tick right next to the buttons of places transitions and tokens, like shown in figure Figure III Screen when user will be able to insert a new arc..

The user can see the current state of the Petri nets from the tables that show all

necessary information.

*Create Petri Net:* User can save the file either by the default name which is output.lp or with a name that will be give in the text box next to FILE NAME label. The file name must end with the extenstion .lp. Then user should press create Petri Net The file will be created and saved in the folder LPFiles that contains all the figures the user can use.

**Figure III** Screen when user will be able to insert a new arc.

#### a. Edit Places

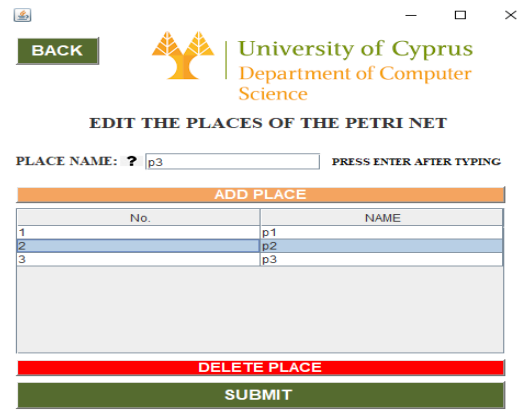
Adding places is demonstrated in Figure IV**Error!** Reference source not found. the user can enter the name of the place and then *ADD* it in the place's table. User should be aware to press *enter* after finishing typing the name of the place. The label next to the text box will be red until user presses *enter* and then will turn black.

**Figure IV** One place is added in the place's table.

**Figure V** Message when trying to enter a place with a name that already exists.

The user will get the message appeared in Figure V when he/she tries to add a place that has the same name with another place, transition or a token. A similar message will appear when trying to add a place with the same name as a transition or a token.

When selecting a certain line and then pressing *DELETE* the line will be deleted from the table as demonstrated in Figure VII. When user has added in the table all the



BACK

University of Cyprus  
Department of Computer Science

EDIT THE PLACES OF THE PETRI NET

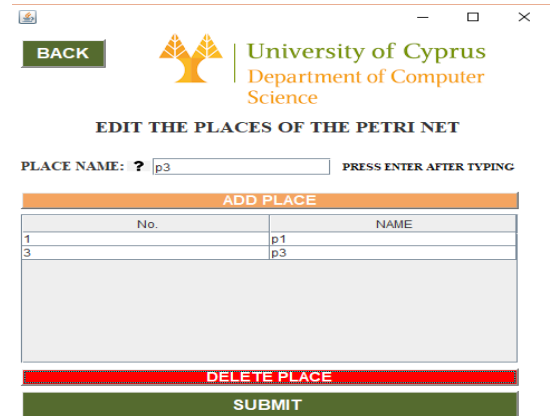
PLACE NAME: ? p3 PRESS ENTER AFTER TYPING

ADD PLACE	
No.	NAME
1	p1
2	p2
3	p3

DELETE PLACE

SUBMIT

Figure VII Adding two new places in the table and selecting line 2.



BACK

University of Cyprus  
Department of Computer Science

EDIT THE PLACES OF THE PETRI NET

PLACE NAME: ? p3 PRESS ENTER AFTER TYPING

ADD PLACE	
No.	NAME
1	p1
3	p3

DELETE PLACE

SUBMIT

Figure VII Deleting line 2 from the table

places he/she can press *SUBMIT* to import the places in the places structure. If cancel is pressed instead no change will happen in the places of the Petri net.

### b. Edit Transitions

*Transitions screen* has one more field, named *irreversible*. User should check this box if he/she wants to create a transition that is irreversible. Figure VIII demonstrates the addition of two transitions, one is irreversible and one that is not.

Again, error messages will appear if user tries to insert a transition with a name that already exists either as a transition, place or token.

User can delete a certain entry of transitions by selecting it and then pressing the *delete*. button.

User should again be aware to press *SUBMIT* to save the changer. Otherwise all changes will be lost.

BACK

University of Cyprus  
Department of Computer  
Science

EDIT THE TOKENS OF THE PETRI NET

Token name: ?  PRESS ENTER AFTER TYPING

ADD TOKEN

No.	NAME
1	q

DELETE TOKEN

SUBMIT

**Figure IX Token's screen**

### c. Edit Tokens

Tokens screen in Figure IX is exactly the same as places screen.

### d. Edit Arcs

Arc screen will only be available once user filled tables of places transitions and

BACK

University of Cyprus  
Department of Computer  
Science

CREATE A NEW PETRI NET

☐ EDIT PLACES ?

☐ EDIT TRANSITIONS ?

☐ EDIT ARCS ?

TRANSITIONS TABLE

No.	NAME	IRREVE.
-----	------	---------

TOKENS TABLE

No.	NAME
-----	------

PLEASE INSERT AT LEAST ONE ENTRY IN PLACE'S TABLE, TRANSITION'S TABLE AND TOKEN'S TABLE.

OK

**Figure X Message when trying to insert an arc with empty tables.**

University of Cyprus  
Department of Computer Science

EDIT THE ARCS OF THE PETRI NET

TYPE: ? ☒ tparc ☐ ptarc FROM: t2 TO: p4

ARC'S LABEL ?   
 TOKEN: a LABEL

ADD

BOND: ? a a

ADD DELETE LABEL

ADD ARC

No.	TYPE	FROM	TO	CARRIES
1	ptarc	p1	t1	a,
2	ptarc	p2	t1	b,
3	tparc	t1	p3	a_b,
4	ptarc	p3	t2	a,
5	ptarc	p4	t2	c,
6	tparc	t2	p4	a_c,

DELETE ARC

SUBMIT

Figure XI Arc's Screen after adding 6 arcs in the Petri Net.

tokens with at least one entry. If the previous condition does not hold, the message in Figure X will appear.

In Figure XI the Arc screen is shown after 6 arcs are added in the Petri net.

The *type* field shown in the up left of the screen is used to define the type of the arc that is going to be added. If the arc has as input a transition and output a place then the first option, *tparc* should be selected. If the arc has as input a place and output a transition then the option *ptarc* should be selected.

As soon as the user selects the appropriate type of the arc, the lists *From* and *To* will be filled with places and transitions depending on the type of the arc. If the arc is set of type *tparc* the *From* list will contain only transitions and *To* list will contain places, while if the arc is set to be *ptarc* then *From* list will be filled with places and *To* with transitions.

Then user should fill the label of the arc. The label can contain tokens or bonds. User can add as many tokens or bonds as he/she wants but only one of a kind can be added. If a token is set on the label in a bond, it cannot appear in the same label individually and the other way around. This is due to the definition of RPNs that states that only one token of a certain type can exist in a RPN. When user wants to add a token in the label, he/she can select the token from the *TOKEN* list and then press add.

User will notice that the token will be added in the list below *Label* as in Figure XII. The same holds with bonds. User should selected the two bonded tokens from the lists and add them to the label. In the example of the figure, user had selected in the first list

University of Cyprus  
Department of Computer Science

EDIT THE ARCS OF THE PETRI NET

TYPE: ? ☒ tparc ☐ ptarc FROM: t1 TO: p1

ARC'S LABEL ?   
 TOKEN: a LABEL

ADD

BOND: ? a a

ADD DELETE LABEL

ADD ARC

No.	TYPE	FROM	TO	CARRIES
1	ptarc	p1	t1	a,
2	ptarc	p2	t1	b,
3	tparc	t1	p3	a_b,
4	ptarc	p3	t2	a,
5	ptarc	p4	t2	c,
6	tparc	t2	p4	a_c,

DELETE ARC

SUBMIT

Figure XII Label of arc filled with token a and bond b\_c.

of the bond token *b* and the second list the token *c* and thus, when pressed added, bond *b\_c* is created and added in the list of the label.

When user has finished adding in the label he/she can press the button *ADD ARC*, to add the arc in arc's table.

User should notice an extra field when adding arcs that are of type *ptarc*. This field, namely negative, is placed right below tokens and bonds and should be used when the output transition of the arc, requires the absence of a token or a bond in order to fire.

In order for the changes to be submitted, user should press the *SUBMIT* button. If *BACK* button is pressed instead, all data will be lost.

### 3. Execute In Clingo

When user selects choice Execute in clingo, the screen in Figure XIII will appear.

*IMPORT FILE*: The user has to select a file that contains the figure of the petri net in ASP language. This is the type of files created in the *55Create a new Petri Net* option. The user can select the file by clicking on the file icon on the left upper page. This will open a new frame with the files that are saved in folder LPfiles like the one in Figure XIV Open Figure File.

*SELECT TYPE OF EXECUTION*: The user also has to select the type of the execution from the combo box below *select type of execution* label. By pressing on the combo box user can see 4 options:

*FORWARD EXECUTION*: This will execute the forward mode of executing RPNs.

*BACKWARD EXECUTION*: This will allow the forward execution to backtrack at any time of the execution.

Figure XIII Execute in Clingo screen

Figure XIV Open Figure File

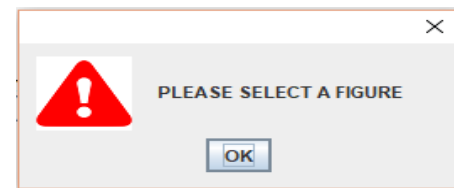
*CAUSAL EXECUTION*: This will allow the forward execution to causal reverse at any time during the execution.

*NON CAUSAL EXECUTION*: This will allow the execution to non causally reverse at any time in during the execution.

*NAME OF THE OUTPUT FILE*: User can save the output file of the execution with a different name to save the results. This can be done by filling the text box below *NAME OF THE OUPUT FILE* with the name of the file, that must end with the extension .txt and then pressing enter. If the box is left empty, then the results will be saved in the default output file *output.txt* and will be overridden when another execution runs.

*FIND ALL SOLUTIONS*: The screen also contains a check box with the label *FIND ALL SOLUTIONS*. User should select this box if he/she wants the tool to find all possible solutions that can be produced given the figure, the type of execution, the iterations, initial marking and any selected properties. This option may delay the execution, especially with the non-causal type of execution and will produce a large output file. User should accompany this option with properties to minimize the amount of solutions.

*INITIAL MARKING*: User must import or create an initial marking for the figure. Before pressing the button, user should select the figure. Otherwise, message in Figure XV Message when trying to insert initial marking without selecting a figure first. will appear.



**Figure XV Message when trying to insert initial marking without selecting a figure first.**

*CHECK PROPERTIES*: User can add propertied to check. Before pressing the button, user should select both a type of execution and a petri net structure.

*EXECUTE IN CLINGO*: User can execute the command with all the files selected and properties by pressing the button *EXECUTE IN CLINGO*. If any of the necessary information is missing, user will get an explanatory message.

## 4. Initial Marking

This screen allows user to import an initial marking for the Petri net or create a new state based on the figure given. The file can also be saved for later use. Once user selects the option for *initial marking*, the screen in appears.

**FILE ALREADY EXISTS** : User should select the field if the file with the initial marking exists already. Then the screen will change as in Figure XVI. **Error! Reference source not found..** If the file does not exist, user should live the field unchecked and continue to define the new initial marking.

**SCREEN WHEN FILE DOES NOT EXISTS:**

**PLACE:** Beside the PLACE label, there is a combo box, which is filled with all the places defined in the Petri net the user selected in window *EXECUTE IN CLINGO*. User should select a place, and then select the tokens and bonds the places holds initially.

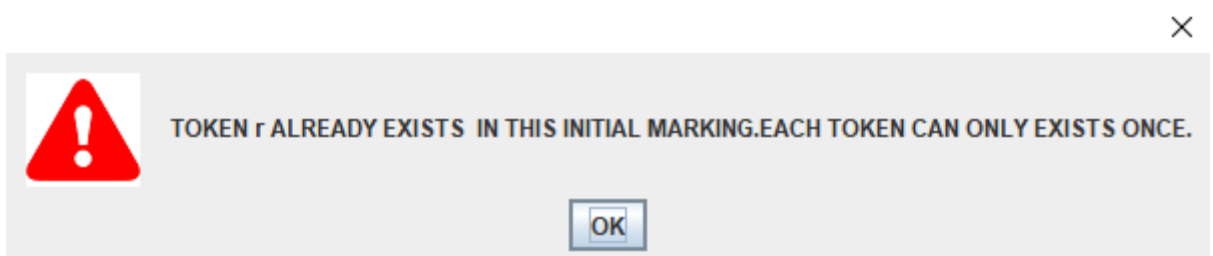
**TOKEN:** This contains the list of tokens that are available in the Petri net. User should select the token that initially exists in the place specified by *PLACES* combobox. If the token is already defined in another place user will get an error message, like the one in the figure Figure XVII.

**BOND:** To create a bond, there exists two lists that contain all tokens that are available in the Petri net. When user selects a token from one list and a token from the other and pressed add, a bond between the two selected tokens is created and added in the place selected in *PLACES* combobox. If any of the tokens is already defined in another place user will get an error message, like the one in the figure Figure XVII.

**TABLE:** The table is used to present the user the initial marking defined.

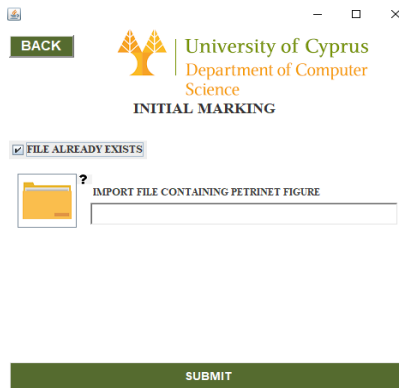
**DELETE:** User can delete a line in the table, by selecting the line and pressing the *DELETE* button.

**Figure XVI Initial Marking screen with option FILE ALREADY EXISTS, not selected.**



**Figure XVII Error Message when trying to insert a token that is already defined in another marking.**

*SAVE THE FILE AS:* User has the option to change the name of the output file with



**Figure XIX Initial marking screen with option *FILE ALREADY EXISTS*, selected**

the initial marking that will be created from the tool, and save the file for future use.

The file should end with the extension *.lp*, and will be saved in the *.* If user leaves the field empty, the file will be saved with the default name. The default name for the output file is *initial.lp* and this file is overridden with every execution. The file is saved in folder *Initial* that exists in the same folder with the Java files.

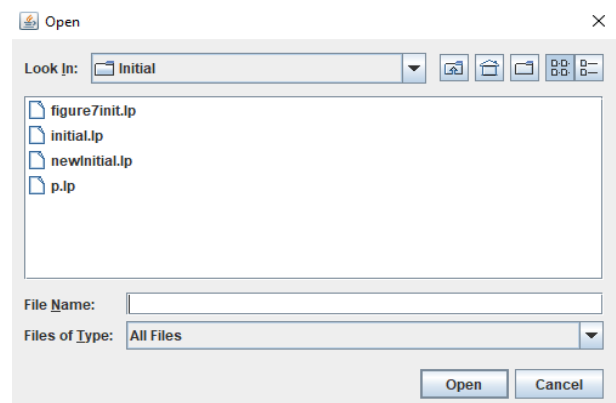
*SUBMIT:* User should press submit as soon as he/she has entered all necessary information for the initial marking of the Petri net.

*FILE ALREADY EXISTS screen:*

*FILE ICON:* User should press the file icon when he/she wants to import a file with initial marking. The button will open a new frame like the one in **Error! Reference source not found.** directed in the folder *Initial*. The path of the file will be presented in the textbox beside the file icon.

*SUBMIT:* User should press submit as soon as he/she has selected the correct file with the initial marking.

In order for the changes to be submitted, user should press the *SUBMIT* button. If *BACK* button is pressed instead, all data will be lost.



**Figure XIX Select a file from folder *Initial* screen**



## 5. Properties

The properties screen in Figure XX, will only has 5 options, namely *REACHABILITY*, *HOME STATE*, *PERSISTENCE*, *TRANSITIONS' LIVENESS*, *SHORTEST PATH*.

When user selects properties to check by pressing on the buttons which will open new screen. All the properties added will be visible from the *table* next to the buttons. Any entry of the table can be deleted by selecting the entry and then pressing the button *DELETE*.

User should be aware, that all the properties entered in properties table, all need to hold to get results. If any of the properties does not hold, then the result of the execution will be unsatisfiable, and there is no way to know which property was not satisfied. Therefore, it is best for the user to check one property at each execution, unless he/she wants to check the combination of the properties.

When the user is satisfied with the properties added, he/she can press the button *SUBMIT* to submit the changes.

In order for the changes to be submitted, user should press the *SUBMIT* button. If *BACK* button is pressed instead, all data will be lost.

### a. Reachability

Reachability is the problem of deciding whether a certain state is reached with a sequence of firings. In the tool there two types of reachability transitions' reachability and marking reachability.

*Transitions' reachability*: This property will check if there exists a sequence of forward or backward firings that sets the selected transition to fire, and if so present the solutions where this hold. If the transition cannot fire, and thus the reachability of the transition is unsatisfied,

Figure XXI Reachability screen

*UNSATISFIABLE* will be presented in the results window.

*Reversibly Reached:* User also has the option to check if the transition fires backwards, depending on the type of execution selected in the execute window. This field is not visible when the type of execution selected is *Forward execution*.

When the user selects the transition to check, he/she can press *ADD* button to insert the property in the properties table place in the right of the page. Then the user can proceed with adding more transitions to check, or submit the properties, by pressing *SUBMIT* button.

*Markings' reachability:* This property will check if there exists a sequence of forward or backward firings that causes the state of the Petri net to contain the marking selected. Again, if the markings reachability is cannot be satisfied, then the results window will present *UNSATISFIABLE*. All the markings inserted from the user will need to be satisfied in the same time, or else the marking is set to be unsatisfied.

When the user selects the marking to check, he/she can press *ADD* button to insert the property in the properties table place in the right of the page, and then continue with the next marking or submit the properties by pressing the *SUBMIT* button.

In order for the changes to be submitted, user should press the *SUBMIT* button. If *BACK* button is pressed instead, all data will be lost.

#### **b. Home State**

The property of Home State is defined as the condition, where there exists a state *S*, and the any other state of the system, can return to this state *S*.

The screen of Home State property as presented in Figure XXIII contains a check box with the label *FIND ALL HOME STATES*. User should check this field if he/she wants the tool to find all home states, and it will lead to the screen shown in Figure XXIII, else user should leave the box unchecked.

**Figure XXIII Home State when FIND ALL HOME STATES unchecked**

In screen of Figure XXIII user can see add a marking to check if this marking is home state. User should first select the place from the *list* right next to the *PLACE NAME* label, and then select the *token* or *bond* that this place should hold. In order for the user to add a token in the marking of the *place*, he/she should first select the *token* from the *list*, and then press the *ADD* button right below the list. Similarly, if the user wants to add a bond, he should select the first token of the bond from the first list next to the *BOND* label, and the second token from the second list, and then press the button *ADD*.

Once the user is satisfied with the state entered in the table, he/she should press *SUBMIT* to submit the property.

In order for the changes to be submitted, user should press the *SUBMIT* button. If *BACK* button is pressed instead, all data will be lost.

### c. Persistence

Tool allows the user to check if any two transitions are persistent, which means, that the firing of one does not set the other to be not enabled. The screen is demonstrated in demonstrated in Figure XXIV. As shown before, user can select to

**Figure XXIV Persistence screen**

check the label FIND ALL PERSISTENCE TRANSITIONS, and the tool will find all the pairs of the transitions that are persistent. This will lead to the change of the screen to the one in Figure XXV.

*FIRST TRANSITION*: Right next to the label, there is a list that contains all the transitions of the Petri net. This is used to define the first transition of the pair the user wants to check for persistence.

*SECOND TRANSITION*: Right next to the label there exists a list that contains all the transitions of the Petri net. This is used to define the second transition of the pair the user wants to check for persistence.

Once the user has selected both transitions he can press the button *ADD* to add to the table the two transitions.

By pressing the *DELETE RECORD*, user can delete the line of the table that is selected. When he/she has added all the transitions he/she wants, he/she can press the button *SUBMIT* to submit the properties.

In order for the changes to be submitted, user should press the *SUBMIT* button. If *BACK* button is pressed instead, all data will be lost.

#### d. Transitions' Liveness

The tool also offers the user the chance to check for the liveness of the transition. Specifically, the tool can check for dead, 11-live and l2-live transitions as seen in Figure XXVII. Dead Transition is a transition that cannot fire in the Petri net, 11-live transition is the transition that can fire at least once and l2-live is any transition that can fire at least K time, where K is an integer greater than zero.

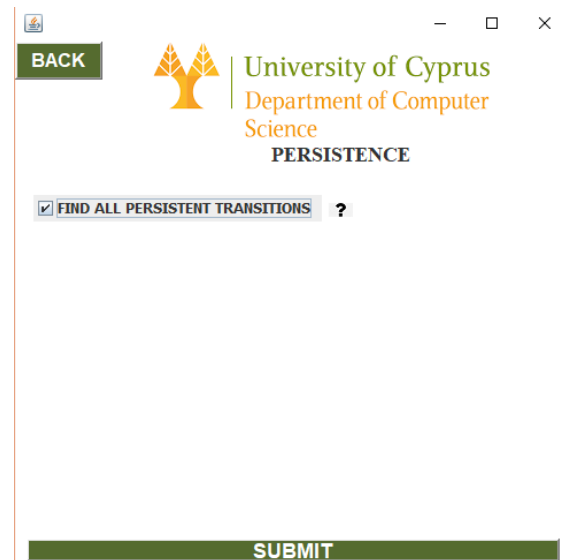


Figure XXV Persistence screen

Once user has selected the l2-live transition, the screen will become like in Figure XXVII, where a new field with the label *ENTER K* appears. User can now enter the *k* value, and then press enter, to let the textbox know it has finished entering the number. As soon as user has selected everything he/she wants and entered all necessary

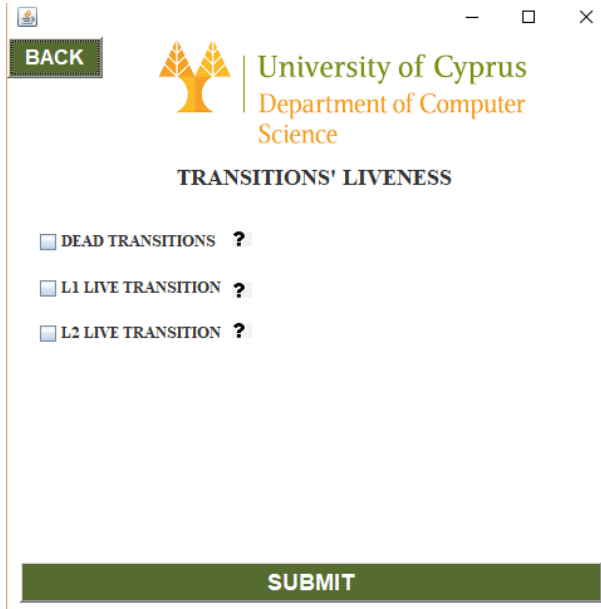


Figure XXVII Transition's Liveness screen

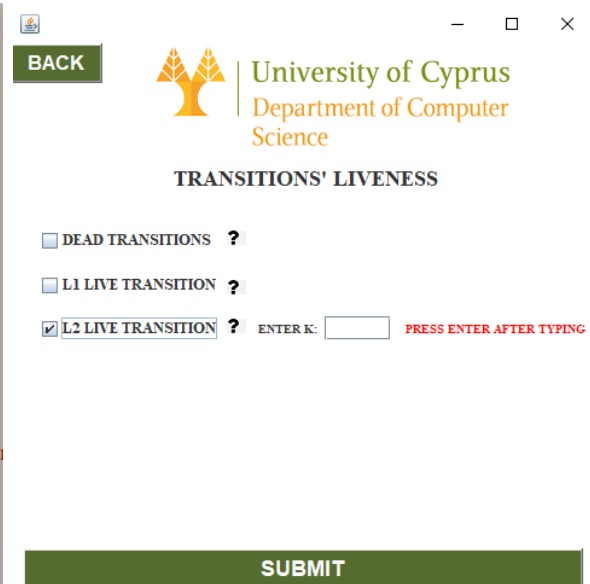


Figure XXVII l2-live checkbox selected

information, he/she can press *SUBMIT* button.

In order for the changes to be submitted, user should press the *SUBMIT* button. If *BACK* button is pressed instead, all data will be lost.

### e. Shortest Path

The tool offers the user the chance to find the shortest path to satisfy the reachability of a transition or a marking. The initial screen of the Shortest path looks like in Figure XXVIII. User has to select if he/she wants to check the shortest path for a marking or a transition and the screen will change like in Figure XXX, or Figure XXX respectively.

*Marking selected screen:*

In the screen where the marking is selected, user can select the place, and add a token or bond to it. If the user wants to add a token, he/she should select the token, and leave the other 2 lists of the bond to their default values, and press *ADD*, whereas if user wants to insert a bond to the place's marking, he/she should select one token from the first list

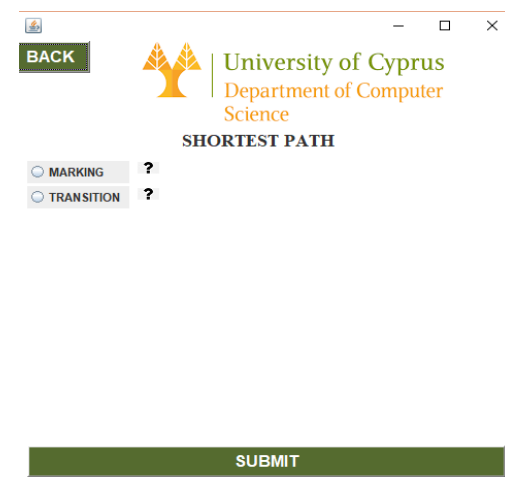


Figure XXVIII Shortest Path

beside *BOND* label and the second token from the second list, set the selected value of the list right next to the *TOKEN* label to the first value "--", and then press *ADD*. The values added will be visible from the table placed right next to the Marking panel.

User can also delete an entry from the marking's table by selecting the line and then pressing *DELETE* button.

Once user has inserted all information for the marking, he/she can press the *SUBMIT* button to submit the property.

#### *Transition selected screen:*

User can select the transition he/she wants to find the shortest bath to from the list right next to the *TRANSITION* label. *Reversibly reached* field should be checked when the user wants to find the shortest path to reverse a transition and will not be available if user selected *Forward* mode of execution in the *EXECUTE IN CLINGO* window. User can press *ADD* button to set this transition to be checked. User can press the *SUBMIT* button to submit the transition, or reset the selected transition to empty by just pressing *CLEAR*.

In order for the changes to be submitted, user should press the *SUBMIT* button. If *BACK* button is pressed instead, all data will be lost.

The screenshot shows a web application window titled "University of Cyprus Department of Computer Science". The main heading is "SHORTEST PATH". Below it, there are two radio buttons: "MARKING" (selected) and "TRANSITION". To the right of each radio button is a question mark icon. Below the radio buttons, there is a section titled "MARKING". It contains three input fields: "PLACE NAME:" with a dropdown menu showing "--", "TOKEN:" with a dropdown menu showing "--", and "BOND:" with two dropdown menus, each showing "--". To the right of these fields is a large empty table. At the bottom of the "MARKING" section, there are two buttons: "ADD" (green) and "DELETE" (red). At the very bottom of the window, there is a large green "SUBMIT" button.

**Figure XXX Shortest Path screen with Marking selected.**

The screenshot shows the same web application window as Figure XXX, but with the "TRANSITION" radio button selected. The "MARKING" radio button is now unselected. Below the radio buttons, there is a section titled "TRANSITION". It contains two input fields: "TRANSITION:" with a dropdown menu showing "--" and "SHORTEST PATH TO TRANSITION:" with a dropdown menu showing "--". Below these fields, there is a checkbox labeled "REVERSIBLY REACHED:". At the bottom of the "TRANSITION" section, there are two buttons: "ADD" (green) and "CLEAR" (red). At the very bottom of the window, there is a large green "SUBMIT" button.

**Figure XXX Shortest Path screen when transition is selected.**

## 6. Results Window

In Figure XXXI the Results window is demonstrated.

In order for the user to see any of the results, he/she should press the *VIEW RESULTS* button.

The first label on the left of the screen states the name of the *output file* that is created. In the example is *output.txt*, because no other value was given for the output file in the Execute window.

Right below, there is the label that states the number of solutions found. This is useful when user has selected to find all possible solution that satisfy a property.

On the screen user can also see the number of the solution that is explained, and user can move backward and forward the solutions from the 2 buttons at the right and left on the bottom of the screen. This buttons will not be visible, if user only wants to see one solution.

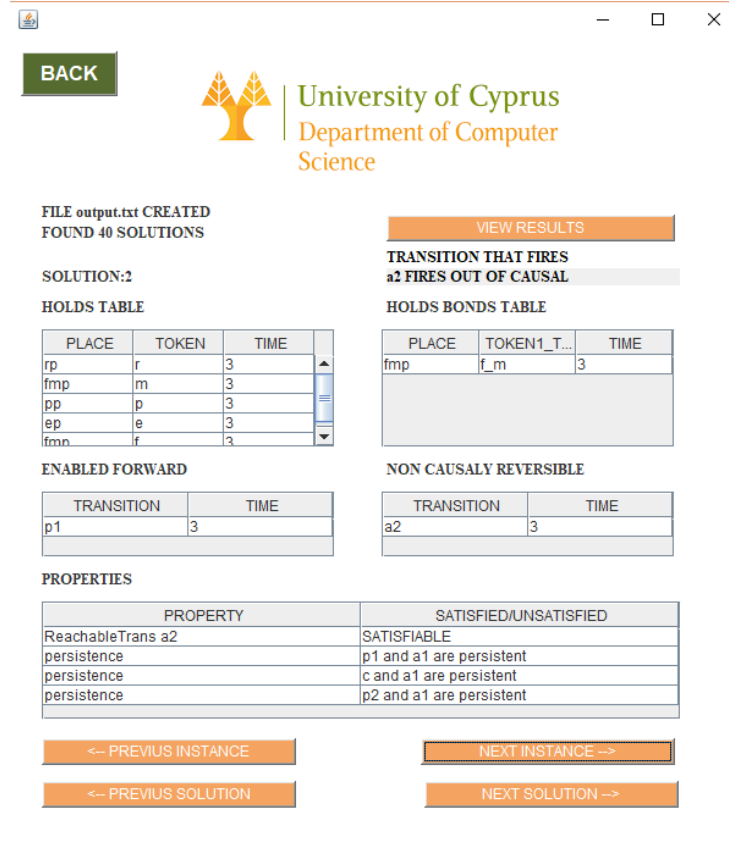


Figure XXXI Results Window

The first table on the upper left of the screen is the holds table, that states what tokens exist in each place at the current time instance, and the table at the upper right is the holdsbonds table, that states which tokens are bonded together in the place at the current time instance. In the example of the figure, place *fmp* holds tokens *f* and *m* at time instance *3* and they are *bonded*.

*Enabled Forward* table, that is placed in the middle left of the screen, shows all the transitions that are enabled to fire forward at the current time instance.

*Reversibly Enabled* table, is placed on the right of the *Enabled Forward* table, and changes name depending the type of execution selected in Executing in Clingo window. As we can see in the example, the selected type is non causal reversing.

On the upper right, below the VIEW RESULTS button, there is a label stating which *transition will fire* at this time instance and if it will fire forward or backward. In the figure above, a2 will fire in an out of causal mode.

User can move forward and backward in time by pressing the buttons PREVIOUS INSTANCE and NEXT INSTANCE respectively.

The final table, at the bottom of the page, is *Properties* table. Any property selected by the user will be presented in this table, along with some details. In the figure, reachability of transition a2 was checked and is satisfied, and the user also wanted all possible persistent pairs of transitions. As we can see 3 pairs are found.

User can go back to execute Something else, or just exit the tool by pressing the x button on the right upper corner of the frame.

## 7. Prove New States

The final option is PROVE NEW STATES, and is demonstrated in Figure XXXI. The purpose of this screen is to prove that the NON CAUSAL mode of execution, provides new states to the Petri net, that are not accessible with the other modes.

The screen has again the *file icon* for the user to enter the figure file that contains the Petri net.

Moreover, there is a list, with the label *COMPARE NON CAUSAL SOLUTION WITH* that contains the following mode of executing RPNs: *Forward Only*, *Backtracking*, *Causal Reversing*. The user can select one of the above, to compare the solution with non-causal reversibility.

Finally, there is a textbox with the label *ITERATIONS* that is used for the user to specify the amount of time he/she wants the programs to run, and the button INITIAL MARKING, so that the user can define the initial state of the Petri Net.

User should be aware, that both *iterations* and *initial marking* can be definitive. This is because a short duration, may not allow the

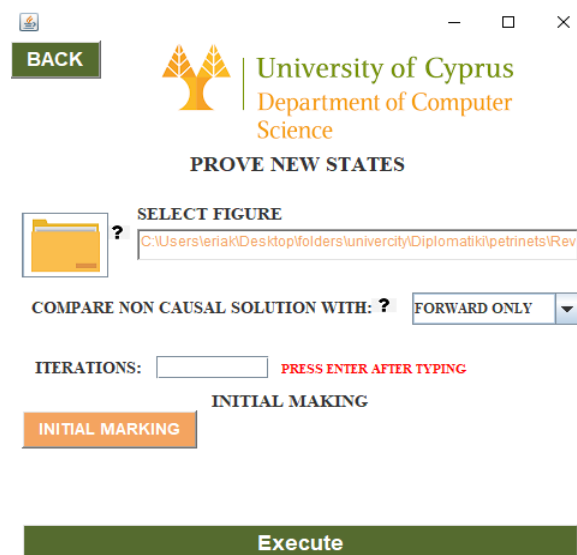


Figure XXXII Prove new states window



sequence to expand to the point where the two type of execution may differ, or the initial marking may not allow the firing of a transition that would fire in an *out of causal execution* but would not in any other form.

## Appendix B

### ASP Forward execution

same(X, X):-token(X).

bonded(P, X, Y, TS):-connectedTokens(P, X, Y, TS).

bonded(P, X, Y, TS):-bonded(P, X, Z, TS), not same(Z, X), not same(X, Y), not same(Z, Y), connectedTokens(P, Z, Y, TS).

connectedTokens(P, X, Y, TS):-holdsbonds(P, X, Y, TS).

connectedTokens(P, X, Y, TS):-holdsbonds(P, Y, X, TS).

connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, Q1, \_),  
tparcbond(T, TP, Q1, \_).

connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, Q1, \_),  
tparcbond(T, TP, \_, Q1).

connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, \_, Q1),  
tparcbond(T, TP, Q1, \_).

connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, \_, Q1),  
tparcbond(T, TP, \_, Q1).

% ENABLED TRANSITIONS

notenabled(T, TS):-ptarc(P, T, Q), not holds(P, Q, TS), time(TS), token(Q).

notenabled(T, TS):-ptarcbond(P, T, Q1, Q2), not holdsbonds(P, Q1, Q2, TS),  
time(TS), token(Q1), token(Q2).

notenabled(T, TS):-ptarcabsence(P, T, Q), holds(P, Q, TS), time(TS), token(Q).

notenabled(T, TS):-ptarcbondabsence(P, T, Q1, Q2), holdsbonds(P, Q1, Q2, TS),  
token(Q1), token(Q2), time(TS).

enabled(T, TS):-not notenabled(T, TS), trans(T), time(TS).

{fires(T, TS)}:-enabled(T, TS), trans(T), time(TS).

add(P, Q, T, TS):- fires(T, TS), tparc(T, P, Q), time(TS).

add(P, Q, T, TS):- fires(T, TS), tparcbond(T, P, Q, \_), time(TS).

add(P, Q, T, TS):- fires(T, TS), tparcbond(T, P, \_, Q), time(TS).

add(P, Q, T, TS):- fires(T, TS), ptarc(PT, T, Q1), tparc(T, P, Q1), bonded(PT, Q, Q1, TS), time(TS).

add(P, Q, T, TS):-fires(T, TS), ptarc(PT, T, Q1), tparcbond(T, P, Q1, \_), bonded(PT, Q, Q1, TS), time(TS).

add(P, Q, T, TS):-fires(T, TS), ptarc(PT, T, Q1), tparcbond(T, P, \_, Q1), bonded(PT, Q, Q1, TS), time(TS).

add(P, Q, T, TS):-fires(T, TS), tparcbond(PT, T, Q1, \_), tparc(T, P, Q1), bonded(PT, Q, Q1, TS), time(TS).

add(P, Q, T, TS):-fires(T, TS), tparcbond(PT, T, \_, Q1), tparc(T, P, Q1), bonded(PT, Q, Q1, TS), time(TS).

add(TP, Q, T, TS):-fires(T, TS), connectedBondPlacesThrough(PT, T, TP, Q1), bonded(PT, Q, Q1, TS), time(TS).

del(P, Q, T, TS):-fires(T, TS), ptarc(P, T, Q), time(TS).

del(P, Q, T, TS):-fires(T, TS), tparcbond(P, T, Q, \_), time(TS).

del(P, Q, T, TS):-fires(T, TS), tparcbond(P, T, \_, Q), time(TS).

del(P, Q, T, TS):-fires(T, TS), ptarc(P, T, Q1), tparc(T, TP, Q1), bonded(P, Q, Q1, TS), time(TS).

del(P, Q, T, TS):-fires(T, TS), ptarc(P, T, Q1), tparcbond(T, TP, Q1, \_), bonded(P, Q, Q1, TS), time(TS).

del(P, Q, T, TS):-fires(T, TS), ptarc(P, T, Q1), tparcbond(T, TP, \_, Q1), bonded(P, Q, Q1, TS), time(TS).

del(P, Q, T, TS):-fires(T, Ts), ptarcbond(P, T, Q1, \_), tparc(T, TP, Q1), bonded(P, Q, Q1, TS), time(TS).

del(P, Q, T, TS):-fires(T, Ts), ptarcbond(P, T, \_, Q1), tparc(T, TP, Q1), bonded(P, Q, Q1, TS), time(TS).

del(PT, Q, T, TS):-fires(T, TS), connectedBondPlacesThrough(PT, T, TP, Q1), bonded(PT, Q, Q1, TS), time(TS).

createBond(P, Q1, Q2, T, TS):-fires(T, TS), tparcbond(T, P, Q1, Q2), time(TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT, T, Q2), tparc(T, TP, Q2), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT, T, Q2), tparc(T, TP, Q2), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT, T, Q2), tparcbond(T, TP, Q2, \_), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT, T, Q2), tparcbond(T, TP, Q2, \_), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT, T, Q2), tparcbond(T, TP, \_, Q2), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT, T, Q2), tparcbond(T, TP, \_, Q2), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), tparcbond(PT, T, Q2, \_), tparc(T, TP, Q2), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),  
ptarcbond(PT, T, Q2, \_), tparc(T, TP, Q2), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),  
ptarcbond(PT, T, \_, Q2), tparc(T, TP, Q2), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),  
ptarcbond(PT, T, \_, Q2), tparc(T, TP, Q2), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),  
connectedBondPlacesThrough(PT, T, TP, Q2), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),  
connectedBondPlacesThrough(PT, T, TP, Q2), bonded(PT, Q1, Q2, TS).

holds(P, Q, TS+1):-add(P, Q, T, TS), fires(T, TS), time(TS).

holds(P, Q, TS+1):-holds(P, Q, TS), fires(T, TS), not del(P, Q, T, TS), time(TS).

holds(P, Q, TS+1):-holds(P, Q, TS), firing(F, TS), F=0.

break(PT, Q, Q1, TS):-moveBond(PT, TP, Q, Q1, TS).

holdsbonds(P, Q1, Q2, TS+1):-createBond(P, Q1, Q2, T, TS), time(TS).

holdsbonds(P, Q1, Q2, TS+1):-holdsbonds(P, Q1, Q2, TS), not break(P, Q1, Q2,  
TS), not break(P, Q2, Q1, TS), time(TS).

holdsbonds(P, Q1, Q2, TS+1):-moveBond(P1, P, Q1, Q2, TS), not holdsbonds(P,  
Q2, Q1, TS+1), time(TS).

%CONSUME MORE

consumesmore(P, TS):- not holds(P, Q, TS), del(P, T, Q, TS), place(P), trans(T),  
ptarc(P, Q, TS).

consumesmore:-consumesmore(P, TS).

:-consumesmore.

firing(Q, TS):-Q=#count{T:fires(T, TS)}, time(TS).

:-firing(Q, TS), Q>1, time(TS).

:-firing(Q, TS), Q=0, enabled(T, TS), time(TS).

#show time/1.

#show enabled/2.

#show fires/2.

#show holds/3.

#show holdsbonds/4.

#show add/4.

#show del/4.

#show createBond/5.

# Appendix C

## Backtracking

same(X, X):-token(X).

bonded(P, X, Y, TS):-connectedTokens(P, X, Y, TS).

bonded(P, X, Y, TS):-bonded(P, X, Z, TS), not same(Z, X), not same(X, Y), not  
same(Z, Y), connectedTokens(P, Z, Y, TS).

connectedTokens(P, X, Y, TS):-holdsbonds(P, X, Y, TS), not breakBond(P, X, Y,  
TS), time(TS).

connectedTokens(P, X, Y, TS):-holdsbonds(P, Y, X, TS), not breakBond(P, Y, X,  
TS), time(TS).

connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, Q1, \_),  
tparcbond(T, TP, Q1, \_).

connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, Q1, \_),  
tparcbond(T, TP, \_, Q1).

connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, \_, Q1),  
tparcbond(T, TP, Q1, \_).

connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, \_, Q1),  
tparcbond(T, TP, \_, Q1).

% FORWARD RULES

%ENABLED TRANSITIONS

notenabled(T, TS):-ptarc(P, T, Q), not holds(P, Q, TS), time(TS).

notenabled(T, TS):-ptarcbond(P, T, Q1, Q2), not holdsbonds(P, Q1, Q2, TS),  
time(TS).

notenabled(T, TS):-ptarcabsence(P, T, Q), holds(P, Q, TS), time(TS).

notenabled(T, TS):-ptarcbondabsence(P, T, Q1, Q2), holdsbonds(P, Q1, Q2, TS),  
time(TS).

enabled(T, TS):-not notenabled(T, TS), trans(T), time(TS).

{fires(T, TS)}:-enabled(T, TS), trans(T), time(TS).

add(P, Q, T, TS):-fires(T, TS), tparc(T, P, Q), time(TS).

add(P, Q, T, TS):-fires(T, TS), tparcbond(T, P, Q, \_), time(TS).

add(P, Q, T, TS):-fires(T, TS), tparcbond(T, P, \_, Q), time(TS).

add(P, Q, T, TS):-fires(T, TS), ptarc(PT, T, Q1), tparc(T, P, Q1), bonded(PT, Q,  
Q1, TS), time(TS).

add(P, Q, T, TS):-fires(T, TS), ptarc(PT, T, Q1), tparcbond(T, P, Q1, \_),  
bonded(PT, Q, Q1, TS), time(TS).

add(P, Q, T, TS):-fires(T, TS), ptarc(PT, T, Q1), tparcbond(T, P, \_, Q1),  
bonded(PT, Q, Q1, TS), time(TS).

add(P, Q, T, TS):-fires(T, TS), tparcbond(PT, T, Q1, \_), tparc(T, P, Q1),  
bonded(PT, Q, Q1, TS), time(TS).

add(P, Q, T, TS):-fires(T, TS), tparcbond(PT, T, \_, Q1), tparc(T, P, Q1),  
bonded(PT, Q, Q1, TS), time(TS).

add(TP, Q, T, TS):-fires(T, TS), connectedBondPlacesThrough(PT, T, TP, Q1),  
bonded(PT, Q, Q1, TS), time(TS).



del(P, Q, T, TS):-fires(T, TS), ptarc(P, T, Q), time(TS).

del(P, Q, T, TS):-fires(T, TS), ptarcbond(P, T, Q, \_), time(TS).

del(P, Q, T, TS):-fires(T, TS), ptarcbond(P, T, \_, Q), time(TS).

del(P, Q, T, TS):-fires(T, TS), ptarc(P, T, Q1), bonded(P, Q, Q1, TS), time(TS).

del(P, Q, T, TS):-fires(T, Ts), ptarcbond(P, T, Q1, \_), bonded(P, Q, Q1, TS),  
time(TS).

del(P, Q, T, TS):-fires(T, Ts), ptarcbond(P, T, \_, Q1), bonded(P, Q, Q1, TS),  
time(TS).

del(PT, Q, T, TS):-fires(T, TS), connectedBondPlacesThrough(PT, T, TP, Q1),  
bonded(PT, Q, Q1, TS), time(TS).

createBond(P, Q1, Q2, T, TS):-fires(T, TS), tparcbond(T, P, Q1, Q2), time(TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT,  
T, Q2), tparc(T, TP, Q2), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT,  
T, Q2), tparc(T, TP, Q2), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT,  
T, Q2), tparcbond(T, TP, Q2, \_), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT,  
T, Q2), tparcbond(T, TP, Q2, \_), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT,  
T, Q2), tparcbond(T, TP, \_, Q2), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT, T, Q2), tparcbond(T, TP, \_, Q2), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarcbond(PT, T, Q2, \_), tparc(T, TP, Q2), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarcbond(PT, T, Q2, \_), tparc(T, TP, Q2), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarcbond(PT, T, \_, Q2), tparc(T, TP, Q2), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarcbond(PT, T, \_, Q2), tparc(T, TP, Q2), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), connectedBondPlacesThrough(PT, T, TP, Q2), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), connectedBondPlacesThrough(PT, T, TP, Q2), bonded(PT, Q1, Q2, TS).

break(PT, Q, Q1, TS):- moveBond(PT, TP, Q, Q1, TS).

%BACKWORD RULES

%ENABLED TRANSITIONS

%notenabledB(TRANSITION, TIME).

notenabledB(T, TS):-HH=#max{HHH:transHistory(T1, HHH, TS), history(HHH), trans(T1)}, history(HH), history(H), trans(T), transHistory(T, H, TS), HH>H, time(TS).

notenabledB(T, TS):-transHistory(T, H, TS), H=0.

notenabledB(T, t):-irreversible(T).

enabledB(T, TS):-trans(T), time(TS), not notenabledB(T, TS).

%BACKTRACKING FIRE

{firesB(T, TS)}:-enabledB(T, TS), trans(T), time(TS).

addB(P, Q, T, TS):-firesB(T, TS), ptarc(P, T, Q), time(TS).

addB(P, Q, T, TS):-firesB(T, TS), ptarcbond(P, T, Q, \_), time(TS).

addB(P, Q, T, TS):-firesB(T, TS), ptarcbond(P, T, \_, Q), time(TS).

addB(P, Q, T, TS):-firesB(T, TS), ptarc(P, T, Q1), tparc(T, TP, Q1), bonded(TP, Q1, Q, TS), time(TS).

addB(P, Q, T, TS):-firesB(T, TS), ptarc(P, T, Q1), tparcbond(T, TP, Q1, \_), bonded(TP, Q1, Q, TS), time(TS).

addB(P, Q, T, TS):-firesB(T, TS), ptarc(P, T, Q1), tparcbond(T, TP, \_, Q1), bonded(TP, Q1, Q, TS), time(TS).

addB(P, Q, T, TS):-firesB(T, TS), ptarcbond(P, T, Q1, \_), tparc(T, TP, Q1), bonded(TP, Q1, Q, TS), time(TS).

addB(P, Q, T, TS):-firesB(T, TS), ptarcbond(P, T, \_, Q1), tparc(T, TP, Q1), bonded(TP, Q1, Q, TS), time(TS).

addB(P, Q, T, TS):-firesB(T, TS), connectedBondPlacesThrough(P, T, TP, Q1), bonded(TP, Q1, Q, TS), time(TS).

delB(P, Q, T, TS):- firesB(T, TS), tparc(T, P, Q), time(TS).

delB(P, Q, T, TS):- firesB(T, TS), tparcbond(T, P, Q, \_), time(TS).

delB(P, Q, T, TS):- firesB(T, TS), tparcbond(T, P, \_, Q), time(TS).

delB(P, Q, T, TS):- firesB(T, TS), tparc(T, P, Q1), bonded(P, Q, Q1, TS),  
time(TS).

delB(P, Q, T, TS):- firesB(T, TS), tparcbond(T, P, Q1, \_), bonded(P, Q, Q1, TS),  
time(TS).

delB(P, Q, T, TS):- firesB(T, TS), tparcbond(T, P, \_, Q1), bonded(P, Q, Q1, TS),  
time(TS).

breakBond(P, Q1, Q2, TS):- firesB(T, TS), tparcbond(T, P, Q1, Q2), time(TS).

breakB(P, Q1, Q2, TS):- moveBondB(P, PT, Q1, Q2, TS), time(TS).

breakB(P, Q1, Q2, TS):- breakBond(P, Q1, Q2, TS).

moveBondB(TP, PT, Q1, Q2, TS):- firesB(T, TS), holdsbonds(TP, Q1, Q2, TS),  
ptarc(PT, T, Q), tparc(T, TP, Q), bonded(TP, Q1, Q, TS).

moveBondB(TP, PT, Q1, Q2, TS):- firesB(T, TS), holdsbonds(TP, Q1, Q2, TS),  
ptarc(PT, T, Q), tparc(T, TP, Q), bonded(TP, Q2, Q, TS).

moveBondB(TP, PT, Q1, Q2, TS):-firesB(T, TS), holdsbonds(TP, Q1, Q2, TS),  
ptarc(PT, T, Q), tparcbond(T, TP, Q, \_), bonded(TP, Q1, Q, TS).

moveBondB(TP, PT, Q1, Q2, TS):-firesB(T, TS), holdsbonds(TP, Q1, Q2, TS),  
ptarc(PT, T, Q), tparcbond(T, TP, Q, \_), bonded(TP, Q2, Q, TS).

moveBondB(TP, PT, Q1, Q2, TS):-firesB(T, TS), holdsbonds(TP, Q1, Q2, TS),  
ptarc(PT, T, Q), tparcbond(T, TP, \_, Q), bonded(TP, Q1, Q, TS).

moveBondB(TP, PT, Q1, Q2, TS):-firesB(T, TS), holdsbonds(TP, Q1, Q2, TS),  
ptarc(PT, T, Q), tparcbond(T, TP, \_, Q), bonded(TP, Q2, Q, TS).

moveBondB(TP, PT, Q1, Q2, TS):-firesB(T, TS), holdsbonds(TP, Q1, Q2, TS),  
ptarcbond(PT, T, Q, \_), tparc(T, TP, Q), bonded(TP, Q1, Q, TS).

moveBondB(TP, PT, Q1, Q2, TS):- firesB(T, TS), holdsbonds(TP, Q1, Q2, TS),  
ptarcbond(PT, T, Q, \_), tparc(T, TP, Q), bonded(TP, Q2, Q, TS).

moveBondB(TP, PT, Q1, Q2, TS):-firesB(T, TS), holdsbonds(TP, Q1, Q2, TS),  
ptarcbond(PT, T, \_, Q), tparc(T, TP, Q), bonded(TP, Q1, Q, TS).

moveBondB(TP, PT, Q1, Q2, TS):-firesB(T, TS), holdsbonds(TP, Q1, Q2, TS),  
ptarcbond(PT, T, \_, Q), tparc(T, TP, Q), bonded(TP, Q2, Q, TS).

moveBondB(TP, PT, Q1, Q2, TS):-firesB(T, TS), holdsbonds(TP, Q1, Q2, TS),  
connectedBondPlacesThrough(PT, T, TP, Q), bonded(TP, Q1, Q, TS).

moveBondB(TP, PT, Q1, Q2, TS):-firesB(T, TS), holdsbonds(TP, Q1, Q2, TS),  
connectedBondPlacesThrough(PT, T, TP, Q), bonded(TP, Q2, Q, TS).

%UPDATE RULES%

holds(P, Q, TS+1):-add(P, Q, T, TS), fires(T, TS), time(TS).

holds(P, Q, TS+1):-addB(P, Q, T, TS), firesB(T, TS), time(TS).

holds(P, Q, TS+1):-holds(P, Q, TS), fires(T, TS), not del(P, Q, T, TS), time(TS).

holds(P, Q, TS+1):-holds(P, Q, TS), firesB(T, TS), not delB(P, Q, T, TS),  
time(TS).

holds(P, Q, TS+1):-holds(P, Q, TS), firing(F, TS), F=0.

holdsbonds(P, Q1, Q2, TS+1):-createBond(P, Q1, Q2, T, TS), fires(T, TS),  
time(TS).

holdsbonds(P, Q1, Q2, TS+1):-holdsbonds(P, Q1, Q2, TS), not break(P, Q1, Q2, TS), not break(P, Q2, Q1, TS), not breakB(P, Q1, Q2, TS), not breakB(P, Q2, Q1, TS), time(TS).

holdsbonds(P, Q1, Q2, TS+1):-moveBond(P1, P, Q1, Q2, TS), not holdsbonds(P, Q2, Q1, TS+1), time(TS).

holdsbonds(P, Q1, Q2, TS+1):-moveBondB(P1, P, Q1, Q2, TS), not holdsbonds(P, Q2, Q1, TS+1), time(TS).

transHistory(T1, H1+1, TS+1):-H1=#max{H:transHistory(T, H, TS), history(H), trans(T)}, trans(T1), fires(T1, TS), time(TS), history(H1), time(TS+1), history(H1+1).

transHistory(T, H, TS+1):-transHistory(T, H, TS), not fires(T, TS), not firesB(T, TS), time(TS), history(H), time(TS+1).

transHistory(T, 0, TS+1):-transHistory(T, H, TS), firesB(T, TS), history(H), time(TS), time(TS+1).

%DEADLOCK RULES%

%CONSUME MORE

consumesmore(P, TS):- not holds(P, Q, TS), del(P, T, Q, TS), place(P), trans(T), ptarc(P, Q, TS).

consumesmore:-consumesmore(P, TS).

:-consumesmore.

firing(Q, TS):-Q1=#count{T:fires(T, TS)}, Q2=#count{T1:firesB(T1, TS)}, Q=Q1+Q2, time(TS).

:-firing(Q, TS), Q>1, time(TS).

:-firing(Q, TS), Q=0, not consumesmore(T, TS), enabled(T, TS), time(TS).

:-firing(Q, TS), Q=0, enabledB(T, TS), time(TS).

%PRINT RULES

#show time/1.

#show enabled/2.

#show enabledB/2.

#show fires/2.

#show firesB/2.

#show holds/3.

#show holdsbonds/4.

## Appendix D

### Causal reversing

same(X, X):-token(X).

bonded(P, X, Y, TS):-connectedTokens(P, X, Y, TS).

bonded(P, X, Y, TS):-bonded(P, X, Z, TS), not same(Z, X), not same(X, Y), not same(Z, Y), connectedTokens(P, Z, Y, TS).

connectedTokens(P, X, Y, TS):-holdsbonds(P, X, Y, TS), not breakBond(P, X, Y, TS), time(TS).

connectedTokens(P, X, Y, TS):-holdsbonds(P, Y, X, TS), not breakBond(P, Y, X, TS), time(TS).

connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, Q1, \_),  
tparcbond(T, TP, Q1, \_).

connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, Q1, \_),  
tparcbond(T, TP, \_, Q1).

connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, \_, Q1),  
tparcbond(T, TP, Q1, \_).

connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, \_, Q1),  
tparcbond(T, TP, \_, Q1).

%FORWARD RULES

%ENABLED TRANSITIONS

notenabled(T, TS):-ptarc(P, T, Q), not holds(P, Q, TS), time(TS).

notenabled(T, TS):-ptarcbond(P, T, Q1, Q2), not holdsbonds(P, Q1, Q2, TS),  
time(TS).



notenabled(T, TS):-ptarcabsence(P, T, Q), holds(P, Q, TS), time(TS).

notenabled(T, TS):-ptarcbondabsence(P, T, Q1, Q2), holdsbonds(P, Q1, Q2, TS),  
time(TS).

enabled(T, TS):-not notenabled(T, TS), trans(T), time(TS).

{fires(T, TS)}:-enabled(T, TS), trans(T), time(TS).

add(P, Q, T, TS):-fires(T, TS), tparc(T, P, Q), time(TS).

add(P, Q, T, TS):-fires(T, TS), tparcbond(T, P, Q, \_), time(TS).

add(P, Q, T, TS):-fires(T, TS), tparcbond(T, P, \_, Q), time(TS).

add(P, Q, T, TS):-fires(T, TS), ptarc(PT, T, Q1), tparc(T, P, Q1), bonded(PT, Q,  
Q1, TS), time(TS).

add(P, Q, T, TS):-fires(T, TS), ptarc(PT, T, Q1), tparcbond(T, P, Q1, \_),  
bonded(PT, Q, Q1, TS), time(TS).

add(P, Q, T, TS):-fires(T, TS), ptarc(PT, T, Q1), tparcbond(T, P, \_, Q1),  
bonded(PT, Q, Q1, TS), time(TS).

add(P, Q, T, TS):-fires(T, TS), ptarcbond(PT, T, Q1, \_), tparc(T, P, Q1),  
bonded(PT, Q, Q1, TS), time(TS).

add(P, Q, T, TS):-fires(T, TS), ptarcbond(PT, T, \_, Q1), tparc(T, P, Q1),  
bonded(PT, Q, Q1, TS), time(TS).

add(TP, Q, T, TS):-fires(T, TS), connectedBondPlacesThrough(PT, T, TP, Q1),  
bonded(PT, Q, Q1, TS), time(TS).

del(P, Q, T, TS):-fires(T, TS), ptarc(P, T, Q), time(TS).

del(P, Q, T, TS):-fires(T, TS), ptarcbond(P, T, Q, \_), time(TS).

del(P, Q, T, TS):-fires(T, TS), ptarcbond(P, T, \_, Q), time(TS).

del(P, Q, T, TS):-fires(T, TS), ptarc(P, T, Q1), bonded(P, Q, Q1, TS), time(TS).

del(P, Q, T, TS):-fires(T, Ts), ptarcbond(P, T, Q1, \_), bonded(P, Q, Q1, TS),  
time(TS).

del(P, Q, T, TS):-fires(T, Ts), ptarcbond(P, T, \_, Q1), bonded(P, Q, Q1, TS),  
time(TS).

del(PT, Q, T, TS):-fires(T, TS), connectedBondPlacesThrough(PT, T, TP, Q1),  
bonded(PT, Q, Q1, TS), time(TS).

createBond(P, Q1, Q2, T, TS):-fires(T, TS), tparcbond(T, P, Q1, Q2), time(TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT,  
T, Q2), tparc(T, TP, Q2), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT,  
T, Q2), tparc(T, TP, Q2), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT,  
T, Q2), tparcbond(T, TP, Q2, \_), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT,  
T, Q2), tparcbond(T, TP, Q2, \_), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT,  
T, Q2), tparcbond(T, TP, \_, Q2), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT,  
T, Q2), tparcbond(T, TP, \_, Q2), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),  
ptarcbond(PT, T, Q2, \_), tparc(T, TP, Q2), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),  
ptarcbond(PT, T, Q2, \_), tparc(T, TP, Q2), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),  
ptarcbond(PT, T, \_, Q2), tparc(T, TP, Q2), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),  
ptarcbond(PT, T, \_, Q2), tparc(T, TP, Q2), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),  
connectedBondPlacesThrough(PT, T, TP, Q2), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),  
connectedBondPlacesThrough(PT, T, TP, Q2), bonded(PT, Q1, Q2, TS).

break(PT, Q, Q1, TS):-moveBond(PT, TP, Q, Q1, TS).

%%  
%%

%

%

%

## CAUSAL RULES

%

%

%

%%  
%%

%ENABLED TRANSITIONS

%notenabledC(TRANSITION, TIME).

notenabledC(T, TS):-tparc(T, P, Q), not holds(P, Q, TS), time(TS).

notenabledC(T, TS):-tparcbond(T, P, Q1, Q2), not holdsbonds(P, Q1, Q2, TS),  
time(TS).

notenabledC(T, t):-irreversible(T).

enabledC(T, TS):-trans(T), not notenabledC(T, TS), time(TS).

{firesC(T, TS)}:-enabledC(T, TS), trans(T), time(TS).

addC(P, Q, T, TS):-firesC(T, TS), ptarc(P, T, Q), time(TS).

addC(P, Q, T, TS):-firesC(T, TS), ptarcbond(P, T, Q, \_), time(TS).

addC(P, Q, T, TS):-firesC(T, TS), ptarcbond(P, T, \_, Q), time(TS).

addC(P, Q, T, TS):-firesC(T, TS), ptarc(P, T, Q1), tparc(T, TP, Q1), bonded(TP, Q1,  
Q, TS), time(TS).

addC(P, Q, T, TS):-firesC(T, TS), ptarc(P, T, Q1), tparcbond(T, TP, Q1, \_),  
bonded(TP, Q1, Q, TS), time(TS).

addC(P, Q, T, TS):-firesC(T, TS), ptarc(P, T, Q1), tparcbond(T, TP, \_, Q1),  
bonded(TP, Q1, Q, TS), time(TS).

addC(P, Q, T, TS):-firesC(T, TS), ptarcbond(P, T, Q1, \_), tparc(T, TP, Q1),  
bonded(TP, Q1, Q, TS), time(TS).

addC(P, Q, T, TS):-firesC(T, TS), ptarcbond(P, T, \_, Q1), tparc(T, TP, Q1),  
bonded(TP, Q1, Q, TS), time(TS).

addC(P, Q, T, TS):-firesC(T, TS), connectedBondPlacesThrough(P, T, TP, Q1),  
bonded(TP, Q1, Q, TS), time(TS).

delC(P, Q, T, TS):-firesC(T, TS), tparc(T, P, Q), time(TS).

delC(P, Q, T, TS):-firesC(T, TS), tparcbond(T, P, Q, \_), time(TS).

delC(P, Q, T, TS):-firesC(T, TS), tparcbond(T, P, \_, Q), time(TS).

delC(P, Q, T, TS):-firesC(T, TS), tparc(T, P, Q1), bonded(P, Q, Q1, TS), time(TS).

delC(P, Q, T, TS):-firesC(T, TS), tparcbond(T, P, Q1, \_), bonded(P, Q, Q1, TS),  
time(TS).

delC(P, Q, T, TS):-firesC(T, TS), tparcbond(T, P, \_, Q1), bonded(P, Q, Q1, TS),  
time(TS).

moveBondC(TP, PT, Q1, Q2, TS):-firesC(T, TS), holdsbonds(TP, Q1, Q2, TS),  
ptarc(PT, T, Q), tparc(T, TP, Q), bonded(TP, Q1, Q, TS).

moveBondC(TP, PT, Q1, Q2, TS):-firesC(T, TS), holdsbonds(TP, Q1, Q2, TS),  
ptarc(PT, T, Q), tparc(T, TP, Q), bonded(TP, Q2, Q, TS).

moveBondC(TP, PT, Q1, Q2, TS):-firesC(T, TS), holdsbonds(TP, Q1, Q2, TS),  
ptarc(PT, T, Q), tparcbond(T, TP, Q, \_), bonded(TP, Q1, Q, TS).

moveBondC(TP, PT, Q1, Q2, TS):-firesC(T, TS), holdsbonds(TP, Q1, Q2, TS),  
ptarc(PT, T, Q), tparcbond(T, TP, Q, \_), bonded(TP, Q2, Q, TS).

moveBondC(TP, PT, Q1, Q2, TS):-firesC(T, TS), holdsbonds(TP, Q1, Q2, TS),  
ptarc(PT, T, Q), tparcbond(T, TP, \_, Q), bonded(TP, Q1, Q, TS).

moveBondC(TP, PT, Q1, Q2, TS):-firesC(T, TS), holdsbonds(TP, Q1, Q2, TS),  
ptarc(PT, T, Q), tparcbond(T, TP, \_, Q), bonded(TP, Q2, Q, TS).

moveBondC(TP, PT, Q1, Q2, TS):-firesC(T, TS), holdsbonds(TP, Q1, Q2, TS),  
ptarcbond(PT, T, Q, \_), tparc(T, TP, Q), bonded(TP, Q1, Q, TS).

moveBondC(TP, PT, Q1, Q2, TS):-firesC(T, TS), holdsbonds(TP, Q1, Q2, TS),  
ptarcbond(PT, T, Q, \_), tparc(T, TP, Q), bonded(TP, Q2, Q, TS).

moveBondC(TP, PT, Q1, Q2, TS):-firesC(T, TS), holdsbonds(TP, Q1, Q2, TS),  
ptarcbond(PT, T, \_, Q), tparc(T, TP, Q), bonded(TP, Q1, Q, TS).

moveBondC(TP, PT, Q1, Q2, TS):-firesC(T, TS), holdsbonds(TP, Q1, Q2, TS),  
ptarcbond(PT, T, \_, Q), tparc(T, TP, Q), bonded(TP, Q2, Q, TS).

moveBondC(TP, PT, Q1, Q2, TS):-firesC(T, TS), holdsbonds(TP, Q1, Q2, TS),  
connectedBondPlacesThrough(PT, T, TP, Q), bonded(TP, Q1, Q, TS).

moveBondC(TP, PT, Q1, Q2, TS):-firesC(T, TS), holdsbonds(TP, Q1, Q2, TS),  
connectedBondPlacesThrough(PT, T, TP, Q), bonded(TP, Q2, Q, TS).

breakBondC(P, Q1, Q2, TS):-firesC(T, TS), tparcbond(T, P, Q1, Q2), time(TS).

breakC(P, Q1, Q2, TS):-moveBondC(P, PT, Q1, Q2, TS), time(TS).

breakC(P, Q1, Q2, TS):-breakBondC(P, Q1, Q2, TS), time(TS).

%%  
%%

%

%

%

UPDATE RULES

%

%

%

%%  
 %%

holds(P, Q, TS+1):-add(P, Q, T, TS), fires(T, TS), time(TS).

holds(P, Q, TS+1):-addC(P, Q, T, TS), firesC(T, TS), time(TS).

holds(P, Q, TS+1):-holds(P, Q, TS), fires(T, TS), not del(P, Q, T, TS), time(TS).

holds(P, Q, TS+1):-holds(P, Q, TS), firesC(T, TS), not delC(P, Q, T, TS), time(TS).

holds(P, Q, TS+1):-holds(P, Q, TS), firing(F, TS), F=0.

breakBond(P, Q1, Q2, TS):-breakBondC(P, Q1, Q2, TS).

holdsbonds(P, Q1, Q2, TS+1):-createBond(P, Q1, Q2, T, TS), fires(T, TS), time(TS).

holdsbonds(P, Q1, Q2, TS+1):-holdsbonds(P, Q1, Q2, TS), not break(P, Q1, Q2,  
 TS), not break(P, Q2, Q1, TS), not breakC(P, Q2, Q1, TS), not breakC(P, Q1, Q2, TS),  
 time(TS).

holdsbonds(P, Q1, Q2, TS+1):-moveBond(P1, P, Q1, Q2, TS), not holdsbonds(P,  
 Q2, Q1, TS+1), time(TS).

holdsbonds(P, Q1, Q2, TS+1):-moveBondC(P1, P, Q1, Q2, TS), not holdsbonds(P,  
 Q2, Q1, TS+1), time(TS).

transHistory(T1, H1+1, TS+1):-H1=#max{H:transHistory(T, H, TS), history(H),  
 trans(T)}, trans(T1), fires(T1, TS), time(TS), history(H1), time(TS+1), history(H1+1).

transHistory(T, H, TS+1):-transHistory(T, H, TS), not fires(T, TS), time(TS),  
 history(H), time(TS+1).

transHistory(T, 0, TS+1):-transHistory(T, H, TS), firesC(T, TS), history(H),  
 time(TS), time(TS+1).

%%  
%%

%

%

%

## DEADLOCK RULES

%

%

%

%%  
%%

%CONSUME MORE

consumesmore(P, TS):- not holds(P, Q, TS), del(P, T, Q, TS), place(P), trans(T),  
ptarc(P, Q, TS).

consumesmore:-consumesmore(P, TS).

:-consumesmore.

firing(Q, TS):-Q1=#count{T:fires(T, TS)}, Q2=#count{T2:firesC(T2, TS)},  
Q=Q1+Q2, time(TS).

:-firing(Q, TS), Q>1, time(TS).

:-firing(Q, TS), Q=0, not consumesmore(T, TS), enabled(T, TS), time(TS).

:-firing(Q, TS), Q=0, enabledC(T, TS), time(TS).

%%  
%%

%

%



%

PRINT RULES

%

%

%

%%%%%%%%%%%%%%Non-  
causal%%%%%%%%%

#show time/1.

#show enabled/2.

#show enabledC/2.

#show fires/2.

#show firesC/2.

#show holds/3.

#show holdsbonds/4.

# Appendix E

## Non causal reversing

same(X, X):-token(X).

bonded(P, X, Y, TS):-connectedTokens(P, X, Y, TS).

bonded(P, X, Y, TS):-bonded(P, X, Z, TS), not same(Z, X), not same(X, Y), not  
same(Z, Y), connectedTokens(P, Z, Y, TS).

connectedTokens(P, X, Y, TS):-holdsbonds(P, X, Y, TS), not breakBond(P, X, Y,  
TS), time(TS).

connectedTokens(P, X, Y, TS):-holdsbonds(P, Y, X, TS), not breakBond(P, Y, X,  
TS), time(TS).

%connectedTokens(P, X, Y, TS):-holdsbonds(P, X, Y, TS).

%connectedTokens(P, X, Y, TS):-holdsbonds(P, Y, X, TS).

connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, Q1, \_),  
tparcbond(T, TP, Q1, \_).

connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, Q1, \_),  
tparcbond(T, TP, \_, Q1).

connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, \_, Q1),  
tparcbond(T, TP, Q1, \_).

connectedBondPlacesThrough(PT, T, TP, Q1):-ptarcbond(PT, T, \_, Q1),  
tparcbond(T, TP, \_, Q1).

%%  
%%



add(P, Q, T, TS):-fires(T, TS), ptarc(PT, T, Q1), tparc(T, P, Q1), bonded(PT, Q, Q1, TS), time(TS).

add(P, Q, T, TS):-fires(T, TS), ptarc(PT, T, Q1), tparcbond(T, P, Q1, \_), bonded(PT, Q, Q1, TS), time(TS).

add(P, Q, T, TS):-fires(T, TS), ptarc(PT, T, Q1), tparcbond(T, P, \_, Q1), bonded(PT, Q, Q1, TS), time(TS).

add(P, Q, T, TS):-fires(T, TS), ptarcbond(PT, T, Q1, \_), tparc(T, P, Q1), bonded(PT, Q, Q1, TS), time(TS).

add(P, Q, T, TS):-fires(T, TS), ptarcbond(PT, T, \_, Q1), tparc(T, P, Q1), bonded(PT, Q, Q1, TS), time(TS).

add(TP, Q, T, TS):-fires(T, TS), connectedBondPlacesThrough(PT, T, TP, Q1), bonded(PT, Q, Q1, TS), time(TS).

del(P, Q, T, TS):-fires(T, TS), ptarc(P, T, Q), time(TS).

del(P, Q, T, TS):-fires(T, TS), ptarcbond(P, T, Q, \_), time(TS).

del(P, Q, T, TS):-fires(T, TS), ptarcbond(P, T, \_, Q), time(TS).

del(P, Q, T, TS):-fires(T, TS), ptarc(P, T, Q1), bonded(P, Q, Q1, TS), time(TS).

del(P, Q, T, TS):-fires(T, Ts), ptarcbond(P, T, Q1, \_), bonded(P, Q, Q1, TS), time(TS).

del(P, Q, T, TS):-fires(T, Ts), ptarcbond(P, T, \_, Q1), bonded(P, Q, Q1, TS), time(TS).

del(PT, Q, T, TS):-fires(T, TS), connectedBondPlacesThrough(PT, T, TP, Q1), bonded(PT, Q, Q1, TS), time(TS).

createBond(P, Q1, Q2, T, TS):-fires(T, TS), tparcbond(T, P, Q1, Q2), time(TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT, T, Q2), tparc(T, TP, Q2), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT, T, Q2), tparc(T, TP, Q2), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT, T, Q2), tparcbond(T, TP, Q2, \_), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT, T, Q2), tparcbond(T, TP, Q2, \_), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT, T, Q2), tparcbond(T, TP, \_, Q2), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarc(PT, T, Q2), tparcbond(T, TP, \_, Q2), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarcbond(PT, T, Q2, \_), tparc(T, TP, Q2), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarcbond(PT, T, Q2, \_), tparc(T, TP, Q2), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarcbond(PT, T, \_, Q2), tparc(T, TP, Q2), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarcbond(PT, T, \_, Q2), tparc(T, TP, Q2), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS), ptarcbond(PT, T, Q2, \_), tparcbond(T, TP, Q2, \_), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),  
ptarcbond(PT, T, Q2, \_), tparcbond(T, TP, Q2, \_), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),  
ptarcbond(PT, T, Q2, \_), tparcbond(T, TP, \_, Q2), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),  
ptarcbond(PT, T, Q2, \_), tparcbond(T, TP, \_, Q2), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),  
ptarcbond(PT, T, \_, Q2), tparcbond(T, TP, Q2, \_), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),  
ptarcbond(PT, T, \_, Q2), tparcbond(T, TP, Q2, \_), bonded(PT, Q1, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),  
ptarcbond(PT, T, \_, Q2), tparcbond(T, TP, \_, Q2), bonded(PT, Q, Q2, TS).

moveBond(PT, TP, Q, Q1, TS):-fires(T, TS), holdsbonds(PT, Q, Q1, TS),  
ptarcbond(PT, T, \_, Q2), tparcbond(T, TP, \_, Q2), bonded(PT, Q1, Q2, TS).

break(PT, Q, Q1, TS):-moveBond(PT, TP, Q, Q1, TS).

%%  
%%

%

%

%

OUT OF CAUSAL RULES

%

%

%

%%  
%%

%ENABLED TRANSITIONS

%notenabledB(TRANSITION, TIME).

notenabledOC(T, TS):-transHistory(T, H, TS), H<=0.

notenabledOC(T, t):-irreversible(T).

enabledOC(T, TS):-not notenabledOC(T, TS), trans(T), time(TS).

{firesOC(T, TS)}:-enabledOC(T, TS), trans(T), time(TS).

effect(T, Q1, Q2, TS):-firesOC(T, TS), tparcbond(T, P, Q1, Q2), time(TS).

breakBond(P, Q1, Q2, TS):-holdsbonds(P, Q1, Q2, TS), effect(T, Q1, Q2, TS),  
firesOC(T, TS).

placeOfTrans(P, T, TS):-firesOC(T, TS), time(TS), place(P).

usingToken(T, Q, P, TS):-tparc(T, \_, Q), place(P), time(TS).

usingToken(T, Q, P, TS):-tparcbond(T, \_, Q, \_), place(P), time(TS).

usingToken(T, Q, P, TS):-tparcbond(T, \_, \_, Q), place(P), time(TS).

usingToken(T, Q, P, TS):-tparc(T, \_, Q1), bonded(P, Q, Q1, TS).

usingToken(T, Q, P, TS):-tparcbond(T, \_, Q1, \_), bonded(P, Q, Q1, TS).

usingToken(T, Q, P, TS):-tparcbond(T, \_, \_, Q1), bonded(P, Q, Q1, TS).

maxTrans(T, Q, P, TS):-H1=#max{H:usingToken(T1, Q, P, TS), transHistory(T1, H, TS)}, token(Q), transHistory(T, H1, TS), H1>0, place(P).

moveToken(From, To, Q, TS):-holds(From, Q, TS), placeOfTrans(From, T1, TS), T1!=T2, not bonded(From, Q, \_, TS), maxTrans(T2, Q, From, TS), tparc(T2, To, \_), transHistory(T2, H, TS).

moveToken(From, To, Q, TS):-holds(From, Q, TS), placeOfTrans(From, T1, TS), T1!=T2, not bonded(From, Q, \_, TS), maxTrans(T2, Q, From, TS), tparcbond(T2, To, \_, \_), transHistory(T2, H, TS).

moveToken(From, To, Q, TS):-holds(From, Q, TS), placeOfTrans(From, T1, TS), not bonded(From, Q, \_, TS), maxTrans(T1, Q, From, TS), trans(T1), holds(To, Q, 0).

moveBondOC1(From, To, Q1, Q2, TS):-From!=To, holdsbonds(From, Q1, Q2, TS), not breakBond(From, Q1, Q2, TS), placeOfTrans(From, T1, TS), T1!=T2, maxTrans(T2, Q1, From, TS), tparc(T2, To, \_).

moveBondOC2(From, To, Q1, Q2, TS):-From!=To, holdsbonds(From, Q1, Q2, TS), not breakBond(From, Q1, Q2, TS), placeOfTrans(From, T1, TS), T1!=T2, maxTrans(T2, Q2, From, TS), tparc(T2, To, \_).

moveBondOC3(From, To, Q1, Q2, TS):-From!=To, holdsbonds(From, Q1, Q2, TS), not breakBond(From, Q1, Q2, TS), placeOfTrans(From, T1, TS), T1!=T2, maxTrans(T2, Q1, From, TS), tparcbond(T2, To, \_, \_).

moveBondOC4(From, To, Q1, Q2, TS):-From!=To, holdsbonds(From, Q1, Q2, TS), not breakBond(From, Q1, Q2, TS), placeOfTrans(From, T1, TS), T1!=T2, maxTrans(T2, Q2, From, TS), tparcbond(T2, To, \_, \_).

moveBondOC5(From, To, Q1, Q2, TS):-From!=To, holdsbonds(From, Q1, Q2, TS), placeOfTrans(From, T1, TS), maxTrans(T1, Q1, From, TS), holdsbonds(To, Q1, Q2, 0).



moveBondOC6(From, To, Q1, Q2, TS):-From!=To, holdsbonds(From, Q1, Q2, TS),  
placeOfTrans(From, T1, TS), maxTrans(T1, Q2, From, TS), holdsbonds(To, Q1, Q2,  
0).

moveBondOC(From, To, Q1, Q2, TS):-moveBondOC1(From, To, Q1, Q2, TS).

moveBondOC(From, To, Q1, Q2, TS):-moveBondOC2(From, To, Q1, Q2, TS).

moveBondOC(From, To, Q1, Q2, TS):-moveBondOC3(From, To, Q1, Q2, TS).

moveBondOC(From, To, Q1, Q2, TS):-moveBondOC4(From, To, Q1, Q2, TS).

moveBondOC(From, To, Q1, Q2, TS):-moveBondOC5(From, To, Q1, Q2, TS).

moveBondOC(From, To, Q1, Q2, TS):-moveBondOC6(From, To, Q1, Q2, TS).

addOC(P, Q, T, TS):-moveToken(\_, P, Q, TS), firesOC(T, TS).

addOC(P, Q, T, TS):-moveBondOC(\_, P, Q, \_, TS), firesOC(T, TS).

addOC(P, Q, T, TS):-moveBondOC(\_, P, \_, Q, TS), firesOC(T, TS).

delOC(P, Q, T, TS):-moveToken(P, \_, Q, TS), firesOC(T, TS).

delOC(P, Q, T, TS):-moveBondOC(P, \_, Q, \_, TS), firesOC(T, TS).

delOC(P, Q, T, TS):-moveBondOC(P, \_, \_, Q, TS), firesOC(T, TS).

break(P, Q1, Q2, TS):-breakBond(P, Q1, Q2, TS).

break(P, Q1, Q2, TS):-moveBondOC(P, \_, Q1, Q2, TS).

%%%%%%%%  
%%%%%%%%

%

%

%

## UPDATE RULES

%

%

%

%%  
%%

holds(P, Q, TS+1):-add(P, Q, T, TS), fires(T, TS), time(TS).

holds(P, Q, TS+1):-addOC(P, Q, T, TS), firesOC(T, TS), time(TS).

holds(P, Q, TS+1):-holds(P, Q, TS), fires(T, TS), not del(P, Q, T, TS), time(TS).

holds(P, Q, TS+1):-holds(P, Q, TS), firesOC(T, TS), not delOC(P, Q, T, TS),  
time(TS).

holds(P, Q, TS+1):-holds(P, Q, TS), firing(F, TS), F=0.

holdsbonds(P, Q1, Q2, TS+1):-createBond(P, Q1, Q2, T, TS), fires(T, TS), time(TS).

holdsbonds(P, Q1, Q2, TS+1):-holdsbonds(P, Q1, Q2, TS), not break(P, Q1, Q2,  
TS), not break(P, Q2, Q1, TS), time(TS).

holdsbonds(P, Q1, Q2, TS+1):-moveBond(P1, P, Q1, Q2, TS), not holdsbonds(P,  
Q2, Q1, TS+1), time(TS).

holdsbonds(P, Q1, Q2, TS+1):-moveBondOC(P1, P, Q1, Q2, TS), not holdsbonds(P,  
Q2, Q1, TS+1), time(TS).

transHistory(T1, H1+1, TS+1):-H1=#max{H:transHistory(T, H, TS), history(H),  
trans(T)}, trans(T1), fires(T1, TS), time(TS), history(H1), time(TS+1), history(H1+1).

transHistory(T, H, TS+1):-transHistory(T, H, TS), not fires(T, TS), not firesOC(T,  
TS), time(TS), history(H), time(TS+1).

```

transHistory(T, 0, TS+1):-transHistory(T, H, TS), firesOC(T, TS), history(H),
time(TS), time(TS+1).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

%

```

```

%

```

```

%

```

## DEADLOCK RULES

```

%

```

```

%

```

```

%

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

%CONSUME MORE

```

```

consumesmore(P, TS):- not holds(P, Q, TS), del(P, T, Q, TS), place(P), trans(T),
ptarc(P, Q, TS).

```

```

consumesmore:-consumesmore(P, TS).

```

```

:-consumesmore.

```

```

firing(Q, TS):-Q1=#count{T:fires(T, TS)}, Q2=#count{T1:firesOC(T1, TS)},
Q=Q1+Q2, time(TS).

```

```

:-firing(Q, TS), Q>1, time(TS).

```

```

:-firing(Q, TS), Q=0, not consumesmore(T, TS), enabled(T, TS), time(TS).

```

```

:-firing(Q, TS), Q=0, enabledOC(T, TS), time(TS).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

%

%

%

PRINT RULES

%

%

%

%%%%%%%%  
%%%%%%%%

#show time/1.

#show enabled/2.

#show enabledOC/2.

#show fires/2.

#show firesOC/2.

#show holds/3.

#show holdsbonds/4.