Individual Diploma Thesis

CONCURRENT REVERSING PETRI NETS AN EXTENSION OF REVERSING PETRI NETS

Georgios Gregoriou

University Of Cyprus

Department of Computer Science

May 2025

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

Concurrent Reversing Petri nets an extension of Reversing Petri nets

Georgios Gregoriou

Supervisor

Anna Philippou

The Individual Diploma Thesis was submitted in partial fulfillment of the requirements for the acquisition of the Informatics degree from the Department of Computer Science of the University of Cyprus

May 2025

Abstract

This thesis introduces Concurrent Reversing Petri Nets (CRPNs), an extension of Reversing Petri Nets (RPNs) designed to more accurately model complex reversible and concurrent behaviors found in systems such as biochemical pathways. Although RPNs offer a strong basis for causal and out-of-causal-order reversibility, their capacity to manage numerous identical tokens, controlled execution, and coordinated actions is constrained. To address these gaps, CRPNs incorporate three key capabilities: (1) support for token multiplicity through the individual token interpretation; (2) conditional transition activation and reversal through logical preconditions and predetermined transition enablement permissions; and (3) the capacity to activate and reverse transitions involving shared token instances concurrently. These enhancements enable the user to have more control over states that occur within a modeled system. The framework is formalized and evaluated through two case studies: catalysis, illustrating simultaneous transition execution and reversal; and DNA mismatch repair, a complex biochemical process requiring out-of-causal-order reversibility and component reuse. We demonstrate that CRPNs offer increased modeling flexibility compared to traditional RPNs and Multi-Reversing Petri Nets (MRPNs), while preserving core principles like causal tracking and structural consistency.

Contents

Chapter 1

Introduction

1.1 Introduction

1.1 Motivation

Reversible computation has gained increasing prominence due to its applications across diverse domains including biological modeling, quantum computation, and low-energy computing. Central to this paradigm is the ability to execute transitions not only in the forward direction, advancing system state, but also in reverse, restoring previous configurations.

In a sequential context, reversibility typically means being able to undo previous actions in the exact reverse order they were performed, a technique known as backtracking. However, when dealing with concurrent systems, the concept becomes more complicated. In fact, multiple methods have been explored across different theoretical frameworks[1, 2, 3, 4, 5, 6]. A suitable and common approach to tackle a majority of such systems is causal-consistent reversibility, which follows the rule of order which states that only when the products of a transition have been undone may the transition be reversed [7]. The other well-studied approach is out-of-causal-order reversibility, which allows the reversal to be executed in an out-of-causal order [8, 9, 10], most notably found in biochemical systems.

Petri nets have become a potent formalism for representing causality [11, 12, 13] and concurrency in this context. Individual token histories and reversible transitions have been added to the classical model in recent attempts to support reversibility. However, the capacity of these improvements to convey complex behaviors present in biological and chemical systems is still limited, especially when those systems call for coordination between

simultaneous execution and reversal of transitions, multiple indistinguishable components, or control over execution.

1.2 Our extension

In this work, we present an extension to Reversing Petri Nets [5] (RPNs) that enables a richer class of behaviors, due to multiple indistinguishable tokens of the same type, but also by introducing two key features: controlled transition execution and reversal, and simultaneous activation and reversal of transitions. Even when several tokens of the same kind coexist, our model maintains a unique causal history for each token instance by using the individual token interpretation [19, 21]. In addition to enabling the simultaneous execution and undoing of paired transitions involving overlapping token instances, this feature permits transitions to be selectively fired or reversed based on propositional preconditions. For a variety of biologically inspired processes to be accurately captured, such expressive control over transition semantics is necessary.

To demonstrate the utility of our framework, we model two representative examples: catalysis, wherein a reaction is enabled or facilitated by the presence of a non-consumed agent(catalyst), and DNA mismatch repair [14], a complex multi-step biochemical pathway involving hierarchical dependencies, conditional reversibility, and component reuse. These case studies highlight the need for controlled and simultaneous transition mechanisms, and illustrate how our extension faithfully captures the behavior of systems that cannot be recreated by standard reversible Petri nets.

1.3 Methodology

The motivation for the development of this extension to the traditional RPNs was a fundamental limitation in the ability of current models to accurately and expressively represent complex biochemical systems, like DNA mismatch repair [20] (MMR). We started by investigating Reversing Petri Nets [5] (RPNs), which provide a potent formalism for using token histories to model causal and out-of-causal order reversibility. However, traditional RPNs treat each token as a unique entity, which becomes cumbersome and impractical when modeling systems with multiple indistinguishable instances of the same type. This limitation prompted our attention to the Multi-Reversing Petri Nets [19] (MRPNs) extension, which supports multiple tokens of the same type under the individual token interpretation, thereby preserving reversibility through local histories. While MRPNs addressed token multiplicity, they lacked mechanisms for controlled or conditional reversibility and did not support the simultaneous execution and reversal of transitions-features critically needed for accurately modeling biological phenomena like MMR. In the MMR helper proteins must be able to

initiate and reverse reactions out of strict causal order, sometimes simultaneously, as part of the complex temporal and causal interactions, as described in the Calculus of Covalent Bonding [20] (CCB) literature. Motivated by these realizations and difficulties, we developed CRPNs as a novel extension that combines token multiplicity, propositional logic-based transition control, and concurrent execution and reversal of transitions. As we will show in Chapters 3 and 4, this formulation offers the semantic flexibility required to more accurately represent realistic biological models, like catalysis and MMR. In order to close the gap between theoretical formalisms and the requirements of biochemical system modeling, CRPNs are a synthesis of important concepts from RPNs and MRPNs that have been revised and expanded.

1.4 Thesis organization

This thesis consists of five more chapters: Chapter 2 introduces the main components of Reversing Petri Nets (RPNs) and outlines the limitations which prompted our extension. Chapter 3 introduces Concurrent Reversing Petri Nets (CRPNs) and presents their definition and formalization, which focuses on supporting multiple indistinguishable tokens of the same type, controlling transition activation, and enabling simultaneous execution and reversal of transitions. The chapter also includes a modeling example based on catalysis in an RPN and chapter 3 in a CRPN. Chapter 4 presents the DNA mismatch repair process, as an example that cannot be modeled in a traditional RPN and then follows up with its modeling in a CRPN. Chapter 5 provides a detailed comparison between MRPNs and the newly proposed CRPNs, highlighting the improvements and the differences of the two. Finally, Chapter 6 summarizes the main contributions of the thesis and outlines potential directions for future research and enhancements to the proposed framework.

Chapter 2

Background Work

2.1 RPNs

Reversing Petri Nets (RPNs) have been proposed as a net-based formalism to capture both forward and reversible computation in systems which features individual tokens that can be connected via bonds [5]. This enables a causal-consistent form of reversibility, which has proven effective in modeling a range of phenomena including transaction systems and biochemical reactions. The definition of an RPN (A, P, B, T, F) as stated in [5] is the following:

1. A is a finite set of bases or tokens ranged over by a, b, ...
2. P is a finite set of places.
3. B $\subseteq$ {(a, b) | a, b ϵ A, a $\neq$ b} is a set of undirected bonds ranged over by β, γ, ...We use the notation a - b for a bond (a, b) $\in$ B and assume that a - b = b - a.
4. T is a finite set of transitions.
5. F : (P $\times$ T $\cup$ T $\times$ P) $\rightarrow 2^{A \cup B}$ defines a set of directed arcs each associated with a subset of A $\cup$ B where, whenever (a, b) ϵ F(x, y), then a, b ϵ F(x, y).

An RPN is represented as a bipartite directed graph consisting of two types of nodes: places found in P, drawn as circles, and transitions found in T, drawn as rectangles. Tokens found in A, depicted by black dots, are found inside places and represent resources in the modeled system, while bonds found in B, define which tokens are linked together. A set of directed arcs F is responsible for linking places to transitions and vice versa, and it also indicates the flow of tokens during execution. For instance, if a ϵ A, x ϵ P and t ϵ T then a ϵ F(x, t) implies that for transition t to fire a token a is required and F(t, x) implies that the transition t after its been fired will move token a to its outgoing place x.

A specific arrangement of tokens across the places is called a marking, with the initial marking describing the system's starting state.

A fundamental restriction of this definition is that tokens are unique: every instance is distinguished by its identity, even if it shares a base type with others. While this guarantees clarity in reversal semantics, it introduces inefficiencies when modeling systems with multiple indistinguishable components. For instance, to represent multiple molecules of the same kind or several parallel entities with identical roles, one must define distinct token types or duplicate transitions, leading to inflated models.

To relax this restriction, subsequent work [15] introduced token multiplicity giving each model the ability to contain any number of tokens that have the same type. This introduced the possibility of firing a transition more than once with new sets of tokens each time. This can introduce nondeterminism, backward conflict, when reversing transitions. Furthermore, to define reversible semantics in the instance of such backward conflicts, two methodologies were established, inspired by the individual token and the collective token interpretations [16, 17], defined to reason about causality in Petri nets.

2.2 MRPNs

A drawback of RPNs, as already mentioned, is their inability to manage multiple tokens of the same type, thus leading to an extension of RPNs, Multi-Reversing Petri Nets (MRPNs). Because each token in a RPN is distinct, the user is forced to duplicate transitions and labels when modeling multiple instances of the same component, which causes problems with readability and scalability. The individual token interpretation, which distinguishes tokens of the same type based on their causal history rather than their identity, allows MRPNs to be used in such scenarios. This enables token multiplicity while also preserving reversibility and causal consistency.

Formally, an MRPN [19] is a tuple (P, T, $\mathcal{A}$, $\mathcal{A}_v$, $\mathcal{B}$, F), where P and T denote places and transitions, respectively, $\mathcal{A}$ is a set of token types, $\mathcal{A}_v$ a set of token variables, $\mathcal{B} \subseteq \mathcal{A} \times \mathcal{A}$ a set of bond types, and F a function mapping arcs to sets of token or bond variables. Tokens are associated with causal histories in the form of sequences of transition applications, enabling both forward execution and causal-order reversal. Transitions consume tokens from their input places and produce tokens (and bonds) at output places, updating each token’s causal path.

MRPNs have also been modeled to have transition execution be conditioned by logical predicates by using guarded transitions [22, 23]. Each transition is associated with a forward condition (CF) and a reverse condition (CR). These are logical expressions over the data

values within tokens. A transition is allowed to fire or reverse only if its respective condition evaluates to TRUE under a given variable assignment. For example, only tokens with specific data (e.g., temperature ≥ 338°C) may enable or disable a transition.

Graphically, they are the same as the RPNs, with the exception of arcs labeled with token and bond variables as opposed to tokens. MRPNs, also, enable the same transition to be fired multiple times using different token instances of the same type, as long as the transition is enabled to fire.

This capability is critical in biological modeling. Consider catalysis:

In the catalysis example we have two inactive molecules a and b, which only with the intervention of a catalyst c are able to be bonded. The reaction is as follows: catalyst c binds itself to molecule a and because the energy barrier is lower, the reaction can proceed, and the two molecules a and b can be bonded. In the creation of the bond the catalyst is no longer needed and is separated from a, breaking its bond with it.

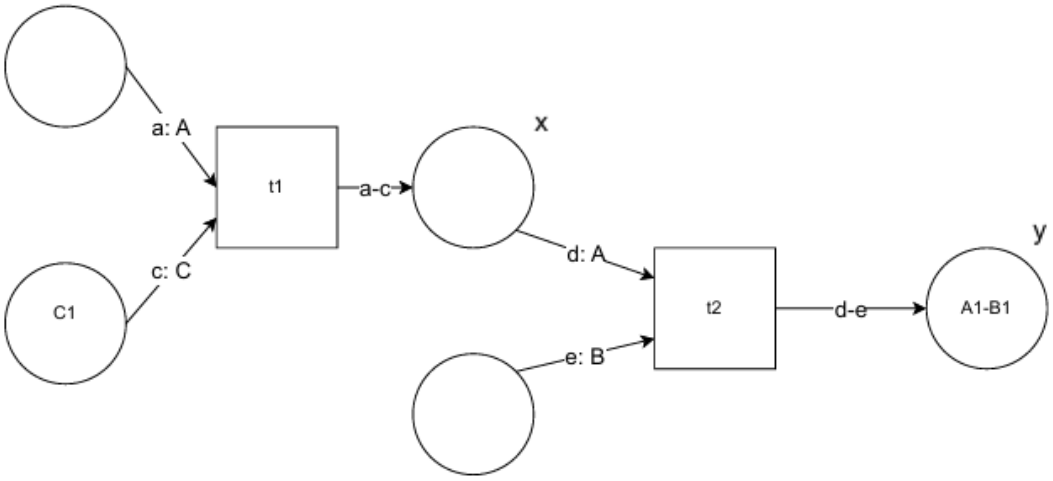


Figure 1. Catalysis in MRPNs

A MRPN model of this process is shown in Fig. 1, where transition t_1's execution bonds molecules A_1 and C_1 and moves them to place x, while the execution of t_2 bonds molecules A_1 and B_1 and moves them to place y. In Fig.1 we see the final state which arises after the firing of transitions t_1 and t_2 and the reversal of t_1. In this state, bond $A_1 – B_1$ has stayed in place y, while base C_1 has returned to its starting place.

These two actions, A_1 bonding to B_1 and A_1 breaking its bond with C_1, happen simultaneously and cannot be recreated with traditional reversing Petri nets because they require the simultaneous firing and reversal of transitions t_2 and t_1, respectively. By bonding the elements first and then reversing the bond with the catalyst, there exists an in-between stage, where a component that consists of the two molecules A_1 and B_1 and the catalyst C_1

appears which does not appear in catalysis normally. With this extension we aim to bypass this in-between stage by using simultaneous execution of paired transitions.

Chapter 3

Concurrent Reversing Petri Net

3.1 Definition of Concurrent Reversing Petri Nets

We present CRPNs, an extension of RPNs that allows specific transitions to be fired or reversed and allow simultaneous activation and reversal of two transitions following the individual token interpretation. Formally, they are defined as follows:

Definition 1. A Concurrent Reversing Petri net (CRPN) is a tuple (Pl, T, $\mathcal{A}$, $\mathcal{A}_v$, $\mathcal{B}$, F, D, C) where:

1. Pl is a finite set of places and T is a finite set of transitions.
2. $\mathcal{A}$ is a finite set of base or token types ranged over by A, B, ...
3. $\mathcal{A}_v$ is a finite set of token variables ranged over by a, b, … We write type(a) for the type of variable a and assume that type(a) $\in \mathcal{A}$ for all a $\in \mathcal{A}_v$.
4. $\mathcal{B} \subseteq \mathcal{A} \times \mathcal{A}$ is a finite set of undirected bond types ranged over by β, γ, … We assume $\mathcal{B}$ to be a symmetric relation and we consider the elements (A, B) and (B, A) to refer to the same bond type, which we denote by A-B. Furthermore, we write $\mathcal{B}_v \subseteq \mathcal{A}_v \times \mathcal{A}_v$, assuming that (a, b) and (b, a) represent the same bond, also denoted as a-b.
5. F :(Pl x T $\cup$ T x Pl) $\rightarrow$ P($\mathcal{A}_v \cup \mathcal{B}_v$) defines a set of directed labelled arcs each associated with a subset of $\mathcal{A}_v \cup \mathcal{B}_v$, where (a, b) $\in$ F(x, y) implies that a, b $\in$ F(x, y). Moreover, for all t $\in$ T, x, y $\in$ Pl, x $\neq$ y, F(x, t) $\cap$ F(y, t) = $\emptyset$.
6. D = $D_f \cup D_b \cup D_{fb}$, where $D_f \subseteq T$, $D_b \subseteq \underline{T}$ and $D_{fb} \subseteq ((T \times \underline{T}) \times (\mathcal{A}_v \times \mathcal{A}_v))$, where $\underline{T} = \{\underline{t} \mid t \in T\}$
7. C: T $\rightarrow$ Prop, where Prop is a simple propositional logic language s.t. Ψ:= True | $\neg\Psi$ | $\Psi \wedge \Psi$ | $\Psi \vee \Psi$ | p, where p are propositions relating to the existence of token and bond instances capturing preconditions for the firing of transitions, which will be explained in section 3.2

As standard in net-based frameworks, places and transitions are connected via labelled directed arcs. These labels are derived from $\mathcal{A}_v \cup \mathcal{B}_v$. They express the requirements, and the effects of transitions based on the type of tokens consumed. Thus, collections of tokens corresponding to the same types and connections as the variables on the labelled arc are able to participate in the transition. More precisely, if F(x, t)=X $\cup$ Y, where X $\subseteq \mathcal{A}_v$,Y $\subseteq \mathcal{B}_v$, the firing of t requires a distinct token instance of type type(a) for each a ϵ X, such that the overall selection of tokens are connected together satisfying the restrictions posed by Y. Similarly, if F(t, x) = X $\cup$ Y, where X $\subseteq \mathcal{A}_v$,Y $\subseteq \mathcal{B}_v$, this implies that during the forward execution of the transition for each a ϵ X a token instance of type type(a) will be transmitted to place x by the transition, in addition to the bonds specified by Y, some of which will be created as an effect of the transition. We make the assumption that if (a, b) ϵ Y then a, b ϵ X and the same variable cannot be used on two incoming arcs of a transition.

$\underline{T}$ defines the reversal of the transitions found in T, indicated by an underscore. Therefore, for each transition t ϵ T, $\underline{t}$ is the reversal of the transition.

D defines that for every transition t_f found in D_f, t_f can be fired on its own, for every transition $\underline{t_b}$ found in D_b, $\underline{t_b}$ can be reversed on its own and for every pair of transitions t_f, $\underline{t_b}$, and variables u, v used by transition t_f and t_b respectively, ((t_f, $\underline{t_b}$), (u, v)) found in D_{fb} defines that t_f can be fired while at the same time $\underline{t_b}$ is reversed as long as the token instances used for u and v are the same (Note that type(u) = type(v) must hold). Transitions or pair of transitions that are not found in D cannot be fired / reversed.

We introduce the following notations. We write $\bullet t = \{ x \epsilon Pl \mid F(x, t) \neq \emptyset \}$ and $t\bullet = \{ x \epsilon Pl \mid F(t, x) \neq \emptyset\}$ for the incoming and outgoing places of transition t, respectively. Furthermore, we write pre(t) = $\bigcup_{x \epsilon Pl} F(x, t)$ for the union of all labels on the incoming arcs of transition t, and post(t) = $\bigcup_{x \epsilon Pl} F(t, x)$ for the union of all labels on the outgoing arcs of transition t.

We restrict our attention to well-formed CRPNs, which satisfy the conservation property [18] in the sense that the number of tokens in a net remains constant during execution.

Definition 2. A CRPN (Pl, T, $\mathcal{A}$, $\mathcal{A}_v$, $\mathcal{B}$, F, D, C) is well-formed if for all t ϵ T:

1. $\mathcal{A}_v \cap$ pre(t) = $\mathcal{A}_v \cap$ post(t) and
2. F(t, x) $\cap$ F(t, y) = $\emptyset$ for all x, y ϵ Pl, x $\neq$ y

Thus, a well-formed CRPN satisfies (1) whenever a variable exists in the incoming arcs of a transition then it also exists on its outgoing arcs, and vice versa, which implies that transitions neither create nor erase tokens, and (2) tokens/bonds cannot be cloned into more than one outgoing place.

In the context of token multiplicity, a mechanism is needed in order to distinguish between token instances with respect to their causal path. To capture this, we distinguish between token instances, as follows:

Definition 3. Given a CRPN (Pl, T, $\mathcal{A}$, $\mathcal{A}_v$, $\mathcal{B}$, F, D, C) a token instance has the form (A, i, xs, sp) where xs is a (possibly empty) list of triples $[(k_1, t_1, v_1), \ldots, (k_n, t_n, v_n)]$ with $n \geq 0$, where $i \geq 1$, $A \in \mathcal{A}$, sp $\in$ Pl, and for all i, $k_i \in \mathbb{N}$, $t_i \in$ T, and $v_i \in \{*\} \cup \mathcal{A}_v$. We write $\mathcal{A}_I$ for the set of token instances ranged over by A_1, A_2, …, and we define the set of bond instances $\mathcal{B}_I$ by $\mathcal{B}_I = \mathcal{A}_I \times \mathcal{A}_I$. Furthermore, given $A_i = (A, i, [(k_1, t_1, v_1), \ldots, (k_n, t_n, v_n)], sp)$, we write

$startPl(A_i) = sp$: Starting place of token A_i
$type(A_i) = A$
$A_i{\downarrow} = (A, i)$
$history(A_i) = [(k_1, t_1, v_1), \ldots, (k_n, t_n, v_n)]$
$cpath\ A_i = [(t_1, v_1), \ldots, (t_n, v_n)]$
$last(A_i) = (k_n, t_n, v_n)$
$A_i + (k,t,v) = (A, i, [(k_1, t_1, v_1), \ldots, (k_n, t_n, v_n), (k,t,v)], sp)$
$Init(A_i) = (A, i, [(k_1, t_1, v_1), \ldots, (k_{n-1}, t_{n-1}, v_{n-1})], sp)$

The set of token instances $\mathcal{A}_I$ corresponds to the basic entities that occur in a system. In the initial state of a net, tokens have the form (A, i, [], sp) where i is a unique identifier for the specific token instance of type A. As computation proceeds the tokens evolve to capture their causal path. If a transition t is executed in the forward direction, with some token instance $(A, i, [(k_1, t_1, v_1), \ldots, (k_n, t_n, v_n)], sp)$ substituted for a variable v, then the token evolves to $(A, i, [(k_1, t_1, v_1), \ldots, (k_n, t_n, v_n), (k, t, v)], sp)$, where k is an integer that characterizes the executed transition, as we will formally define in the sequel. In the instance that the effect of a transition t is reverted for a specific execution k, then all token instances that have partaken in that execution will have the instance (k, t, _) removed from their histories.

As with RPNs the association of token/bond instances to places is called a marking such that $M: P \rightarrow 2^{\mathcal{A}_I \cup \mathcal{B}_I}$, where we assume that if $(A_i, B_i) \in M(x)$ then $A_i, B_i \in M(x)$. In addition, we employ the notion of a history, which assigns a memory to each transition $H: T \rightarrow 2^{\mathbb{N}}$. Intuitively, a history of $H(t) = \emptyset$ for some $t \in T$ captures that the transition has not taken place, or every execution of it has been reversed, and a history such that $k \in H(t)$, captures that the transition had a firing with identifier k that was not reversed. Note that $|H(t)| > 1$ may arise due to cycles but also due to the consecutive execution of the transition by different token instances. A pair of a marking and a history, $<M, H>$, describes a state of an MRPN with $<M_0, H_0>$ the initial state, where $H_0(t) = \emptyset$ for all $t \in T$ and if $A_i \in M_0(x)$, $x \in P$, then $A_i = (A, i, [], sp)$, and $A_i \in M_0(y)$ implies that $x = y$.

Finally, we define $con(A_i, W)$, where $A_i \in \mathcal{A}_I$ and $W \subseteq \mathcal{A}_I \cup \mathcal{B}_I$, to be the tokens connected to A_i as well as the bonds creating these connections according to set W:

$$con(A_i, W) = (\{A_i\} \cap W) \cup \{ x \mid \exists w \text{ s.t. } path(A_i, w, W), (B_i, C_i) \in w, x \in \{(B_i, C_i), B_i, C_i\}\}$$

where $path(A_i, w, W)$ if $w = <\beta_1, \ldots, \beta_n>$, and for all $1 \leq i \leq n$, $\beta_i=(x_{i-1}, x_i) \in W \cap \mathcal{B}_I$, $x_i \in W \cap \mathcal{A}_I$, and $x_0 = A_i$

Definition 4. Given a CRPN (Pl, T, $\mathcal{A}$, $\mathcal{A}_v$, $\mathcal{B}$, F, D, C), an initial marking M_0 and a set of bases and bonds $C \subseteq A_I \cup B_I$, we write:

$$lastT(C) = \begin{cases} t, \text{ if } \forall A_i \in C\ \exists k_i, v_i, \text{ such that, } last(A_i) = (k_i, t, v_i) \\ \perp, \text{ otherwise} \end{cases}$$

$$lastPl(C) = \begin{cases} x, \text{ if } t = lastT(C), \{x\} = \{y \mid \forall A_i \in C, last(A_i) = (k_i, t, v_i), v_i \neq *, v \in F(t, y)\} \\ sp, \text{ if } lastT(C) = \perp \text{ and } \forall A_i \in C \mid startPl(A_i) = sp \\ \perp, \text{ otherwise} \end{cases}$$

Thus, $last(C) \neq \perp$ represents the last transition that has been executed that has used component C in its input arcs. Otherwise, if $last(C) = \perp$ (undefined), there is no transition that has been fired using component C that has not been reversed. Similarly, $lastP(C) \neq \perp$ represents the place where component C is currently found in and is the outgoing place connected to lastT(C), assuming such a place is unique, or the starting place of component C in the initial marking if $lastT(C) = \perp$ and every token found in C has the same starting place, and undefined otherwise.

3.2 Forward Execution

When firing a transition in a CRPN (Pl, T, $\mathcal{A}$, $\mathcal{A}_v$, $\mathcal{B}$, F, D, C), a set of token and bond instances, determined by the transition's incoming arcs, is selected and transferred to the transition's outgoing places, potentially creating bonds in the process. Precisely, for a transition t we define eff(t) to represent the bonds that are created on its outgoing arcs but are not present on its incoming ones:

eff(t) = post(t) - pre(t)

Due to the possibility of multiple token instances that possess the same type, a transition can be fired with different token instances. To enable this behavior, we define the following:

Definition 5. An injective function $\mathcal{W}$: V $\rightarrow$ $\mathcal{A}_I$, where V $\subseteq$ $\mathcal{A}_v$, is called a type-respecting assignment if for all a ϵ V, if $\mathcal{W}$(a) = A_i then type(a) = type(A_i).

We extend the above notation and write $\mathcal{W}$(a, b) for ($\mathcal{W}$(a), $\mathcal{W}$(b)) and, given a set L $\subseteq$ $\mathcal{A}_v \cup \mathcal{B}_v$, we write $\mathcal{W}$(L) = { $\mathcal{W}$(x) | x ϵ L}.

Based on the above we define the following:

Definition 6. Given an CRPN (Pl, T, $\mathcal{A}$, $\mathcal{A}_v$, $\mathcal{B}$, F, D, C), a state <M, H> and a transition t ϵ D_f, we say that t is forward-enabled in <M, H> if there exists a type-respecting S: pre(t) $\cap$ $\mathcal{A}_v$ $\rightarrow$ $\mathcal{A}_I$ such that:

1. $\forall$ a ϵ $\bigcup_{x \epsilon Pl}$ F(x, t), S(a) ϵ M(x)
2. If a, b ϵ F(x, t) for some x ϵ •t and (a, b) ϵ eff(t), then S(a, b) $\notin$ M(x)
3. If a ϵ F(t, y_1) and b ϵ F(t, y_2), for some y_1, y_2 ϵ t•, $y_1 \neq y_2$, then con(S(a), comp(t ,S , M)) $\neq$ con(S(b), comp(t, S, M))
4. Sat(t, <M, H>, S)

where comp(t, S, M) = ($\bigcup_{x \epsilon \bullet t}$ M(x) $\cup$ S(eff(t)))

Thus, t forward-enabled in state <M, H> if there exists a type-respecting assignment S to the tokens of the outgoing arcs of the transition t_b, which we will refer to as a forward-enabling-assignment, such that (1) the token instances and bonds needed for t to be fired, based on the assignment S, are found in the incoming places of the arcs that they were assigned to, (2) no two already bonded token instances that are to be used to fire t can be

bonded again, and (3) connected token instances that are directed by transition t cannot be sent to different outgoing places. This helps maintain the original amount of token instances because otherwise components could be cloned through the firing of a transition, (4) preconditions of t must be met for it to be fired, where Sat is a function that takes as input a transition t, a state <M, H> and an assignment S and returns true only if the preconditions of t are met in state <M, H> for assignment S. Note that comp(t, S, M) = ($\bigcup_{x \in \bullet t} M(x) \cup S(eff(t))$) is composed of the token and bond instances that occur in the incoming places of t ($\bigcup_{x \in \bullet t} M(x)$, which includes the new bond instances created by t (S(eff(t)). Intuitively, comp(t, S, M) contains the components that are moved forward by the transition.

To execute the transition t according to an enabling assignment S, the selected token instances along with their connected components are relocated to the outgoing places of the transition t as specified by the outgoing arcs, with bonds created accordingly. An additional effect is the update of the affected token and bond instances to capture the executed transition in their causal path. To capture this update, we define the following, where k is an integer associated with the specific transition instance:

$$A_i \oplus (S, t, k) = \begin{cases} A_i + (k, t, a), \text{ if } S(a) = A_i \\ A_i + (k, t, *), \text{ if } S^{-1}(A_i) = \perp \end{cases}$$

Note that A_i may not belong to the range of S, i.e. $S^{-1}(A_i) = \perp$, if A_i was not specifically selected to instantiate a variable in pre(t) but, nonetheless, belonged to a connected component transferred by the transition. This is recorded in the causal path of the token instance via the triple (k, t, *). Moreover, we write $(A_i, B_j) \oplus (S, t, k)$ for $(A_i \oplus (S, t, k), B_j \oplus (S, t, k))$ and, given $L \subseteq A_I \cup B_I$, we write $L \oplus (S, t, k) = \{x \oplus (S, t, k) \mid x \in L\}$. Finally, the history of the executed transition is updated to include the next unused integer. Given the above we define:

Definition 7. Given a CRPN (Pl, T, $\mathcal{A}$, $\mathcal{A}_v$, $\mathcal{B}$, F, D, C), a state <M, H> and a transition $t \in D_f$, that is enabled in state <M, H>, and an enabling assignment S, we write $\langle M, H \rangle \xrightarrow{(t, S)} \langle M', H' \rangle$ where for all $x \in Pl$:

$$M'(x) = (M(x) - \bigcup_{a \in F(x, t)} con(S(a), M(x))) \cup \bigcup_{a \in F(t, x)} (con(S(a), comp(t, S, M)) \oplus (S, t, k))$$

where k = max({0} ∪ H(t)) +1 and

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

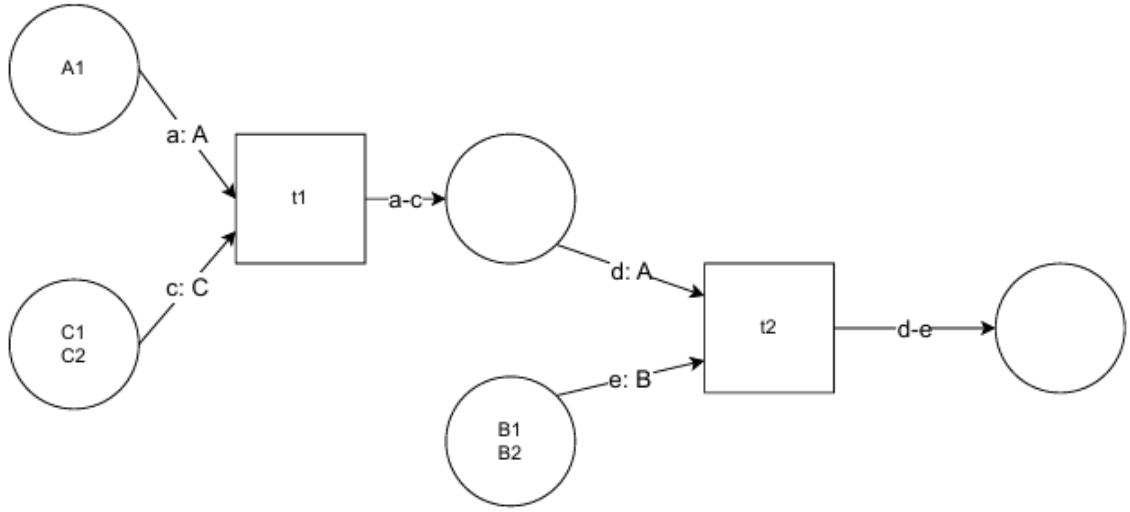


Figure 2. Catalysis in CRPNs initial state

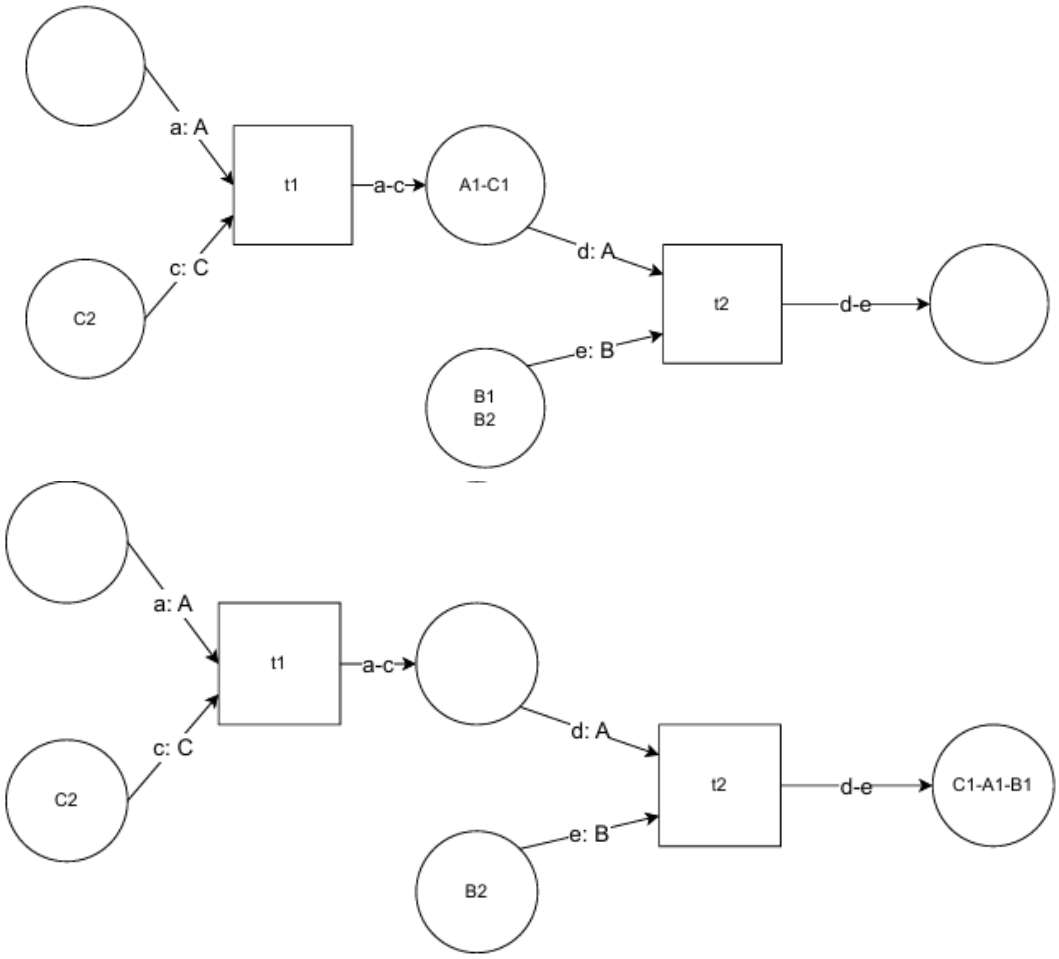


Figure 3. Example of forward execution

In Fig 2. we see the initial state of catalysis model in a CRPN and in Fig. 3 we show the result of the forward execution of t_1, thus a bond is created between A_1 and C_1 and then we show the forward execution of t_2 thus a bond is created between A_1 and B_1. $t_1 \in D_f$ is forward-enabled in Fig. 2 since 1. is met since both A_1 and C_1 are in the correct incoming places of t_1, 2. is met since A_1 and C_1 are not bonded token instances, 3. is met since no bonded token instances are sent to different places by transition t_1 and 4. is met since transition t_1 has no preconditions (similarly for $t_2 \in D_f$). Before this action, in Fig. 2, A_1 = (A, 1, [], spA_1), B_1 = (B, 1, [], spB_1), B_2 = (B, 2, [], spB_2), C_1 = (C, 1, [], spC_1), C_2 = (C, 2, [], spC_2) and after in first state of Fig. 3, (A, 1, [(1, t_1, a)], spA_1), B_1 = (B, 1,[], spB_1), B_2 = (B, 2, [], spB_2), C_1 = (C, 1, [(1, t_1, c)], spC_1), C_2 = (C, 2, [], spC_2).

The histories of the two transitions will both have the instance 1 in them indicating their one activation. If t_1 was to be fired once again then its history would equal to {1,2} and so on.

3.3 Out-of-Causal-Order Reversing

Out-of-causal-order reversing arises in systems where the reversal of a transition does not strictly require that all its subsequent dependent transitions be reversed first. This phenomenon is especially relevant in biological contexts as in [14]. This relaxation allows for greater flexibility in modeling reversible processes that do not strictly require Causal-Order reversibility. Given the above we define:

Definition 8. Given a CRPN (Pl, T, $\mathcal{A}$, $\mathcal{A}_v$, $\mathcal{B}$, F, D, C), a state <M, H> and $\underline{t} \in D_b$, we say that transition $\underline{t}$ is oco-enabled in <M, H> if there exists a type-respecting S: post(t) $\cap$ $\mathcal{A}_v \rightarrow \mathcal{A}_I$ such that:

$\exists(k_b, t_b, \_)$ where $\forall$ a $\in \bigcup_{x \in Pl} F(t_b, x)$, $(k_b, t_b, \_) \in$ history(S(a))

Thus, t_b is oco-enabled in state <M, H> if there exists a set of instances identified by a type-respecting assignment S to the tokens of the outgoing arcs of transition t, such that all have been used in the same firing of transition t.

To reverse t according to an enabling assignment S, the bonds created by the transition are reversed and every new connected component C is moved to place X, where lastPl(C) = X. An additional effect is the update of the affected token and bond instances to capture the reversal of transitions in their causal path. To capture this update, we define the following, where k is an integer associated with the specific transition instance being reversed:

$$A_i \ominus (S, t, k) = \begin{cases} A_i - (k, t, a), \text{ if } S(a) = A_i \\ A_i - (k, t, *), \text{ if } S^{-1}(A_i) = \perp \end{cases}$$

Note that A_i may not belong to the range of S, i.e. $S^{-1}(A_i) = \perp$, if A_i was not specifically selected to instantiate a variable in pre(t) but, nonetheless, belonged to a connected component transferred by the transition. This is recorded in the causal path of the token instance via the triple (k, t, *). Moreover, we write $(A_i, B_j) \ominus (S, t_b, k)$ for $(A_i \ominus (S, t, k), B_j \ominus (S, t, k))$ and, given $L \subseteq A_I \cup B_I$, we write $L \ominus (S, t, k) = \{x \ominus (S, t, k) \mid x \in L\}$. Finally, the history of the executed transition is updated to not include integer k. Given the above we define:

Definition 9. Given a CRPN (Pl, T, $\mathcal{A}$, $\mathcal{A}_v$, $\mathcal{B}$, F, D, C), a state <M, H> and a transition $\underline{t} \in D_b$, that is oco-enabled in state <M, H>, and an enabling assignment S, we write $\langle M, H\rangle \xrightarrow{(\underline{t}, S)} \langle M', H'\rangle$ where for all $x \in Pl$:

$$M'(x) = (M(x) \cup \bigcup_{y \in Pl,\, a \in post(t_b),\, S(a) \in M(y),\, \varphi,\, y \neq x} ([[con(S(a), M(y) - S(eff(t_b))) \ominus (S, t_b, k_b)] \ominus All^*])$$
$$- (\cup S(eff(t)) \cup_{a \in post(t_b),\, S(a) \in M(x)} (con(S(a), M(x) - S(eff(t_b))) \mid \neg\xi))$$

where φ: $lastPl([\,[con(S(a), M(y) - S(eff(t_b)))]\ominus (S, t_b, k_b)] \ominus All^*) = x$

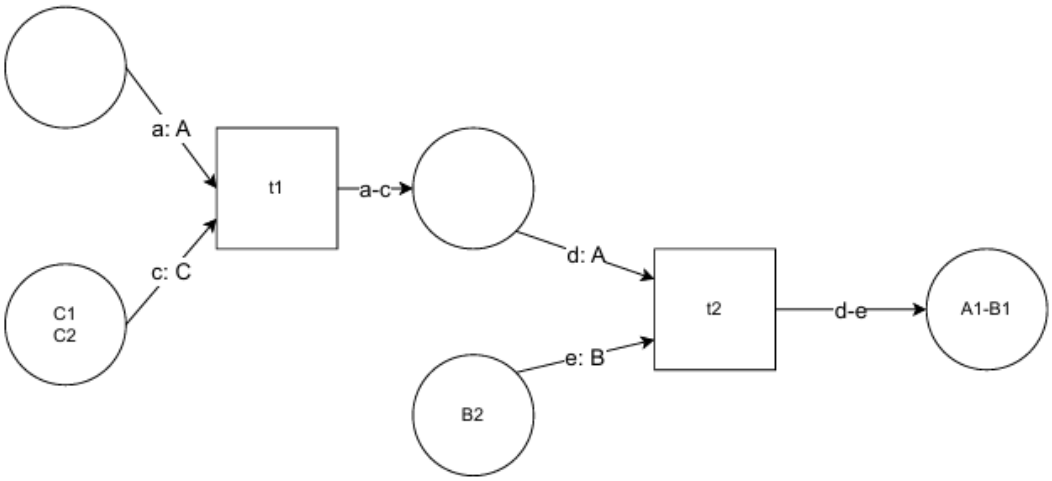
ξ: $lastPl([\,[con(S(a), M(x) - S(eff(t_b)))]\ominus (S, t_b, k_b)] \ominus All^*) = x$

where $k = \max(\bigcup_{k' \in H(t)} k' \mid \underline{t}$ is oco-enabled for its k'^{th} execution) and

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

where All* refers to every (k, t, v) that in the history of any token instance in a component appears with v=* (if after the reversal of a transition, a certain combination of k and t always

appears with *, i.e. (k, t, *), in every token instance's history, in a specific component, that it appears in, then (k, t, *) is removed from all the token instance's histories)



4. Catalysis in CRPNs final state

In Fig. 4 we show the result of the reversal of the first execution of t_1, after t_1 and t_2 have been fired as shown in the second CRPN of Fig. 3. $t_1 \in D_b$ is oco-enabled in the second state of Fig. 3 since 1. is met since for $k_b = 1$ both token instances A_1 and C_1 contain set $(1, t_1, \_)$ in their histories. Before this action, in the second state of Fig. 3, $A_1 = (A, 1, [(1, t_1, a), (1, t_2, d)], spA_1)$, $B_1 = (B, 1, [(1, t_2, e)], spB_1)$, $B_2 = (B, 2, [], spB_2)$, $C_1 = (C, 1, [(1, t_1, c), (1, t_2, *)], spC_1)$, $C_2 = (C, 2, [], spC_2)$ and after in first state of Fig. 4, $(A, 1, [(1, t_2, d)], spA_1)$, $B_1 = (B, 1,[(1, t_2, e)], spB_1)$, $B_2 = (B, 2, [], spB_2)$, $C_1 = (C, 1, [], spC_1)$, $C_2 = (C, 2, [], spC_2)$.The history of transition t_1 will no longer have element 1 in it.

3.4 Concurrent Execution

Definition 10. Given a CRPN (Pl, T, $\mathcal{A}$, $\mathcal{A}_v$, $\mathcal{B}$, F, D, C), a state <M, H> and $((t_f, \underline{t}_b), (u, v)) \in D$, we say that $((t_f, \underline{t}_b), (u, v))$ is concurrently-enabled in <M, H> if there exists a type-respecting S: $(pre(t_f) \cup post(t_b)) \cap \mathcal{A}_v \rightarrow \mathcal{A}_I$ such that:

1. $S(u) = S(v)$
2. $\forall a \in \bigcup_{x \in Pl} F(x, t_f), S(a) \in M(x)$
3. $\exists (k_b, t_b, \_)$ where $\forall a \in \bigcup_{x \in Pl} F(t_b, x), (k_b, t_b, \_) \in history(S(a))$
4. If $a, b \in F(x, t_f)$ for some $x \in \bullet t_f$ and $(a, b) \in eff(t_f)$, then $S(a, b) \notin M(x)$, unless if $\exists (c, d) \in eff(t_b)$, with $(S(c), S(d)) = (S(a), S(b))$
5. If $a \in F(t_f, y_1)$ and $b \in F(t_f, y_2)$, for some $y_1, y_2 \in t_f\bullet$, $y_1 \neq y_2$, then $con(S(a), comp_f(t_f, t_b, S, M)) \neq con(S(b), comp_f(t_f, t_b, S, M))$
6. $Sat(t_f, <M, H>, S)$

Where $comp_f(t_f, t_b, S, M) = ((\bigcup_{x \in \bullet t_f} M(x) - S(eff(t_b))) \cup S(eff(t_f)))$

Thus, simultaneous firing and reversing of transitions t_f and t_b respectively is concurrently-enabled in state <M, H> if there exists a type-respecting assignment S to the tokens of the outgoing arcs of transition t_b and to the tokens of the incoming arcs of transition t_f, which we will refer to as a concurrent-enabling assignment of (t_f, t_b), such that (1) the token instances used for the variables u and v on the incoming arcs of t_f and t_b ,respectively, are the same, (2) the token instances and bonds needed for t_f to be fired, based on the assignment S, are found in the incoming places of the arcs that they were assigned to, (3) the token instances and bonds needed for the reversal of t_b all have been used in the same firing of transition t_b, (4) no two already bonded token instances that are to be used to fire t_f can be bonded again, unless their bond was a result of the execution of t_b that will be reversed, (5) connected token instances that are directed by transition t_f cannot be sent to different outgoing places. This helps maintain the original amount of token instances because otherwise components could be cloned through the firing of a transition, (6) Preconditions of t_f must be met for it to be fired, where Sat is a function that takes as input a transition t, a state <M, H> and an assignment S and returns true only if the preconditions of t are met in state <M, H> for assignment S. Note that $comp_f(t_f, t_b, S, M) = (\bigcup_{x \in \bullet t_f} M(x) \cup S(eff(t_f)))- S(eff(t_b))$ is composed of the token and bond instances that occur in the incoming places of t_f ($\bigcup_{x \in \bullet t_f} M(x)$, which includes the new bond instances created by t_f ($S(eff(t_f))$, and excludes the bonds that were created due to the activation of t_b (- $S(eff(t_b))$). Intuitively, $comp_f(t_f, t_b, S, M)$ contains the components that arise after cancelling the effect of t_b and then implementing the effect of t_f.

An action ((t_f, $\underline{t}_b$), (u, v)) where $t_f \in T$, $\underline{t}_b \in \underline{T}$, $u \in \mathcal{A} \cap post(t_f)$, $v \in \mathcal{A} \cap post(t_b)$ defines the simultaneous activation of transition t_f and reversal of transition t_b where u is a variable found in the outgoing arcs of t_f, v is a variable found in the outgoing arcs of t_b and both have been assigned the same token instance. Given the above we define:

Definition 11. Given a CRPN (Pl, T, $\mathcal{A}$, $\mathcal{A}_v$, $\mathcal{B}$, F, D, C), a state <M, H>, an action a = ((t_f, $\underline{t}_b$), (u, v)) $t_f \in T$, $\underline{t}_b \in \underline{T}$, $u \in \mathcal{A} \cap post(t_f)$ $v \in \mathcal{A} \cap post(t_b)$ that is concurrently-enabled, with assignment S we write <M, H> $\xrightarrow{(a, S)}$ <M', H'> where for all $x \in Pl$:

$M'(x) = (M(x)$

$\cup \bigcup_{a \in F(t_f, x)} ([[con(S(a), comp_f(t_f, t_b, S, M)) \ominus (S, t_b, k_b)] \ominus All^*] \oplus (S, t_f, k_f))$

$\cup \bigcup_{y \in Pl, a \in post(t_b), S(a) \in M(y), \varphi, y \neq x} ([[con(S(a), M(y) - S(eff(t_b))) \ominus (S, t_b, k_b)] \ominus All^*] | \bigcup_{c \in F(y, t_f)} (c \mid S(c) = S(a)) = \varnothing))$

$- ($

$\bigcup_{a \in F(x, t_f)} con(S(a), M(x) - S(eff(t_b)))$

$\cup S(eff(t_b))$

$\bigcup_{a \in post(t_b), S(a) \in M(x)} (con(S(a), M(x) - S(eff(t_b))) \mid \neg\xi, \bigcup_{c \in F(y, t_f)} (c \mid S(c) = S(a)) = \varnothing))$

where $k_f = max(\{0\} \cup H(t)) + 1$ and

$k_b = max(\bigcup_{k' \in H(t)} k' \mid \underline{t_b}$ is oco-enabled for its k'^{th} execution) and

$$H'(t') = \begin{cases} H(t') - \{k_b\}, \text{ if } t' = t_b \\ H(t') \cup \{k_f\}, \text{ if } t' = t_f \\ H(t'), \text{ otherwise} \end{cases} \quad \text{and}$$

φ: $lastPl([[con(S(a), M(y) - S(eff(t_b)))] \ominus (S, t_b, k_b)] \ominus All^*) = x$ and

ξ: $lastPl([[con(S(a), M(x) - S(eff(t_b)))] \ominus (S, t_b, k_b)] \ominus All^*) = x$

With the addition of concurrent execution, we can now move from the first state as shown in Fig. 3 directly to the state as shown in Fig. 4 bypassing the intermediate second state as shown in Fig. 3. We will set into effect $((t_2, \underline{t_1}),(d, a)) \in D_{fb}$, where d is the variable on the arc between the incoming place of t_2 and t_2 and DP is the variable in the arc between the incoming place of t_1 and t_1. Therefore $((t_2, \underline{t_1}),(d, a))$ is concurrently-enabled. 1. is met since the token instance used for both of those variables is A_1 (the same), 2. is met since both A_1 and B_1 are in the correct incoming places of t_2, 3. is met since for $k_b = 1$ both token instances A_1 and C_1 contain set $(1, t_1, \_)$ in their histories, 4. is met since A_1 and B_1 are not bonded token instances, 5. is met since no bonded token instances are sent to different places by transition t_2 and 6. is met since transition t_2 has no preconditions. Before this action, in the first state of Fig. 3, $A_1 = (A, 1, [(1, t_1, a)], spA_1)$, $B_1 = (B, 1, [], spB_1)$, $B_2 = (B, 2, [], spB_2)$, $C_1 = (C, 1, [(1, t_1, c)], spC_1)$, $C_2 = (C, 2, [], spC_2)$ and after in Fig. 4, $(A, 1, [(1, t_2, d)], spA_1)$, $B_1 = (B, 1, [(1, t_2, e)], spB_1)$, $B_2 = (B, 2, [], spB_2)$, $C_1 = (C, 1, [], spC_1)$, $C_2 = (C, 2, [], spC_2)$.

And thus, with this extension to the RPNs the operator has more control over the transitions that can be fired or reversed at a given moment.

Chapter 4

DNA misatch repair

4.1 Description of DNA mismatch repair

The DNA mismatch repair [14, 20] is initiated when a DNA strand contains a mismatched pair of bases (adenine (A), cytosine (C), guanine (G) or thymine (T)), that is, two bases that cannot match namely A-C, A-G, T-C and T-G (intended bonds are between A-T and C-G). The process after identifying the mismatch will continue with the methylation of the DNA strand, as we will explain and model later on, and by using the proteins MutS, MutL, MutH, and UvrD, the methylated strand can be identified and therefore the mismatch can be targeted. The process then continues by removing the component which contains the mismatched base from the DNA strand. Specifically, MutS recruits MutL and MutH after first binding to the mismatch. MutL can then create a loop in the DNA and recognize the methylated strand. In this loop, the old strand is covered by MutL and itself, while the newer strand is now outside. The unmethylated strand is then cleaved by MutH. Because of the size of the proteins and the DNA loop, this occurs some distance away from the mismatch. After detecting the cleavage, UvrD can proceed along the strand in the direction of the mismatch. As it progresses, the outer strand is removed. The DNA loop vanishes concurrently with the release of the MutL, MutH, and MutS proteins. Ultimately, UvrD removes the incorrect base along with its neighbors by moving to a position just beyond the mismatch. After that, UvrD removes itself from the DNA, leaving the original, proper strand to be finished.

In the DNA mismatch repair process, there are instances where bonds are created and destroyed that involve the same token instance of a component, simultaneously, which prompted us to create this extension to the traditional RPNs. Therefore, now the components of token instances that appear in the net at any given time follow the anatomy of the component as it is in the real DNA mismatch repair process without extra in-between stages.

In the following section we model the CRPNs for the DNA mismatch repair. We note that for the sake of clarity some transitions create multiple bonds between multiple token

instances so the diagrams can remain as small as possible. In this instance we assume that each transition that creates multiple bonds is composed of a number of transitions equal to the number of bonds it creates and each one of these sub-transitions can be reversed.

4.2 Modeling DNA mismatch repair

Now we will model the DNA repair process, by first creating a DNA strand with a pair of mismatched bases and then following the sequence of actions as found in [20] for the repair process, in a CRPN (Pl, T, $\mathcal{A}$, $\mathcal{A}_v$, $\mathcal{B}$, F, D, C)

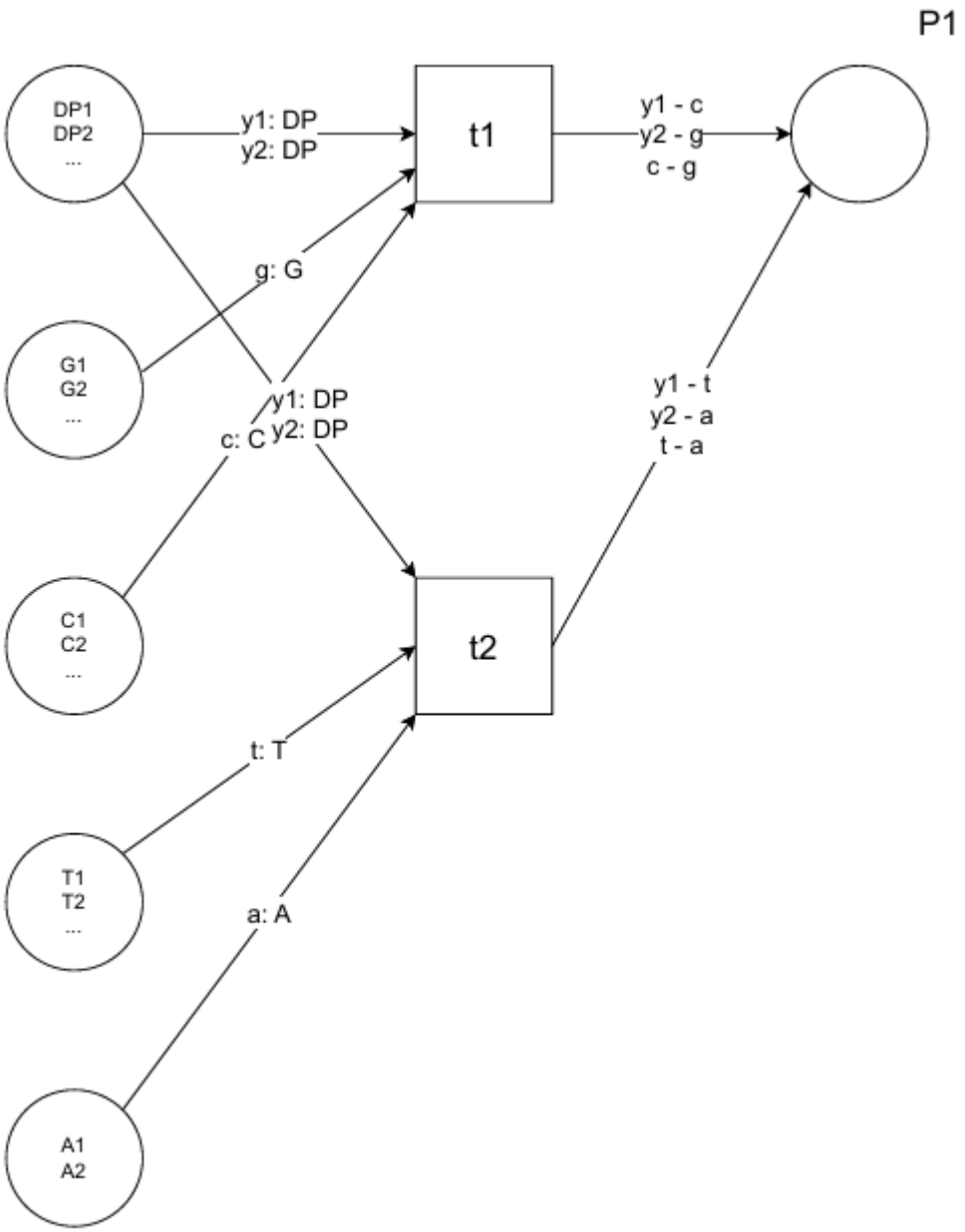


Figure 5. Creation of the first 4 elements of the DNA chain

The CRPN in Fig. 5 implements the **creation of the first 4 elements in the DNA strand** (2 bases adenine (A), cytosine (C), guanine (G) or thymine (T) and 2 deoxyribose/phosphate groups (DP)). The component created and moved to the outgoing place P_1 of the transitions will either be the component DP-C-G-DP if t_1 is executed or DP-T-A-DP if t_2 is executed (components DP-A-T-DP and DP-T-A-DP are considered the same).

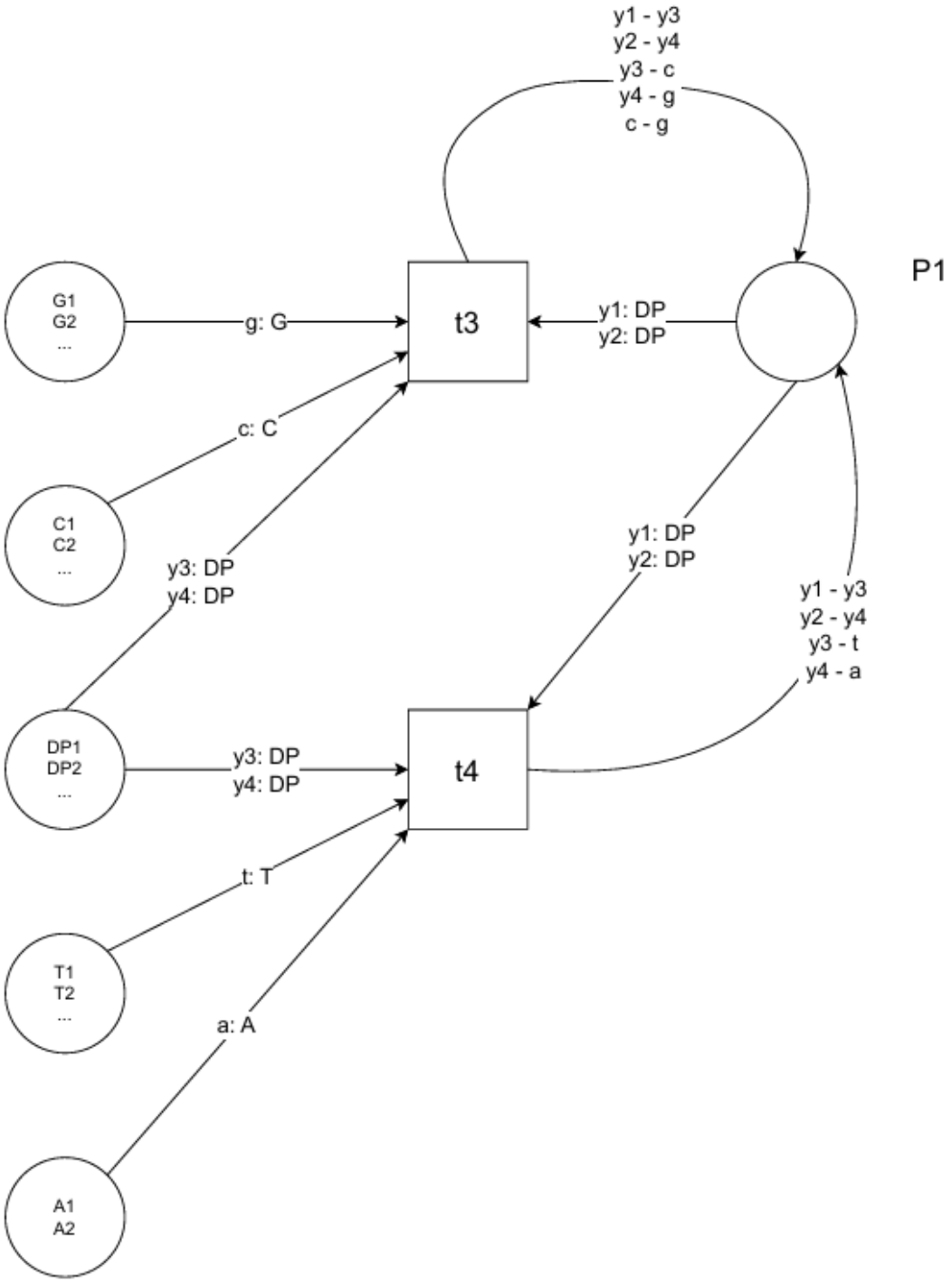


Figure 6. Expanding the DNA chain

Now we continue by **expanding the DNA strand**, as shown in Fig. 6. Transitions t_1 and t_2 are responsible for adding new components DP-C-G-DP if t_1 is executed or DP-T-A-DP if t_2 is executed to the DNA chain created in Fig. 5 in place P_1, linking the DPs of the new components with the DPs on the edge of the DNA chain.

The preconditions for transitions t_1 and t_2 to fire are:

1. $|y_1|, |y_2| \leq 2$, where $|y|$ refers to the number of bonds that instance y has connected to it
2. There exists path P of length 3 where the $S(y_1)$ and $S(y_2)$ are the starting and end points of the path respectively and for every other token instance x in the path type(x) = C || G || T || A, where if type(x) = C then x is a cytosine base, if type(x) = G then x is a guanine base, if type(x) = T then x is a thymine base and if type(x) = A then x is an adenine base

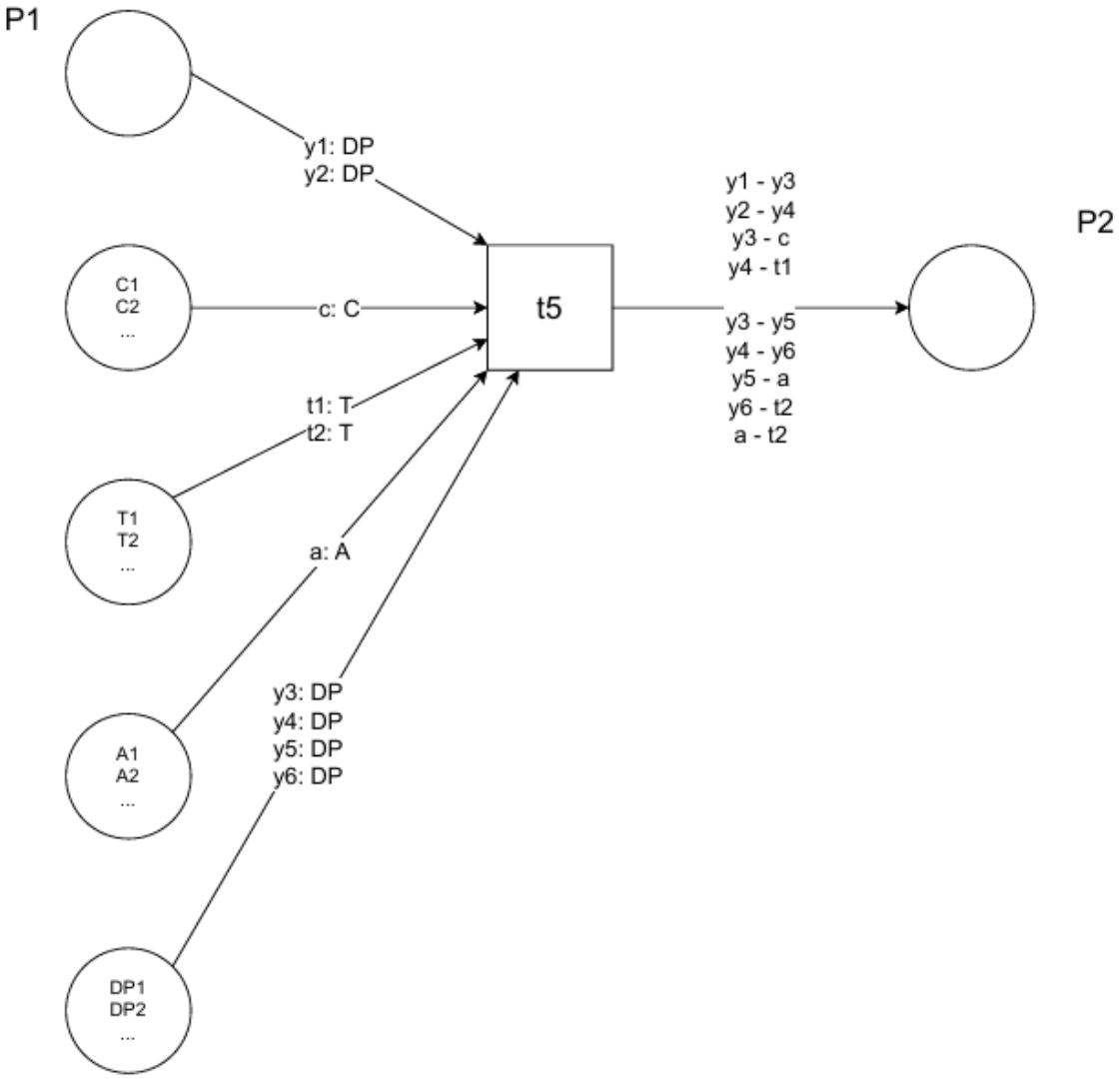


Figure 7. Mismatch in the DNA chain

We now proceed with the **creation of a mismatch in the DNA strand followed by a correct set of 4 elements**. In the example, as shown in Fig. 7, transition t_1 receives two token instances of types A and G and creates an A-G type mismatch on the DNA strand found in place P_1 and adds a C-G match right after as transition t_3 does in Fig. 6, moving the whole component to place P_2. Similarly, the rest of the mismatches (T-G, T-C, A-C) can be implemented in the same way. The preconditions for the transition to fire are:

1. $|y_1|, |y_2| \leq 2$, where |y| refers to the number of bonds that instance y has connected to it
2. There exists path P of length 3 where the $S(y_1)$ and $S(y_2)$ are the starting and end points of the path respectively and for every other token instance x in the path type(x) = C || G || T || A

After the creation of a mismatch, we can continue using the second CRPN to continue the DNA chain. It is important to mention here that we will focus on examples with one DNA mismatch in a particular DNA strand.

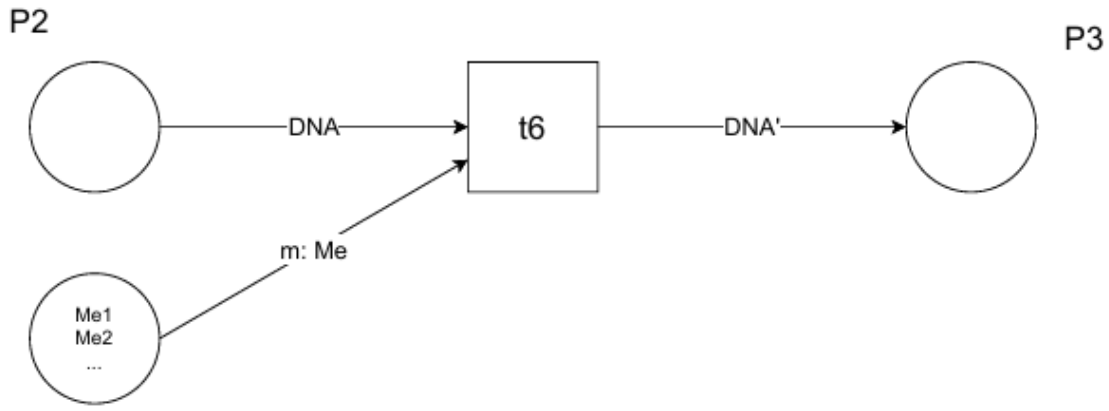


Figure 8. Methylation of the DNA chain

The CRPN in Fig. 8 implements the first step in DNA mismatch repair process. We will follow a repair mechanism in bacteria called **Methyl Mismatch Repair (MMR)** which focuses on the methylation of the DNA strand after it has been created. Methylation refers to the **bond creation with a methyl group**, in our example transition t_6 bonds the methyl group (Me) with the adenine base A_1 and moves the component to place P_3. Where DNA is a label for the exact component used in [20], which is found in place P_2, and DNA' = DNA ∪ {(A_1, Me)}.

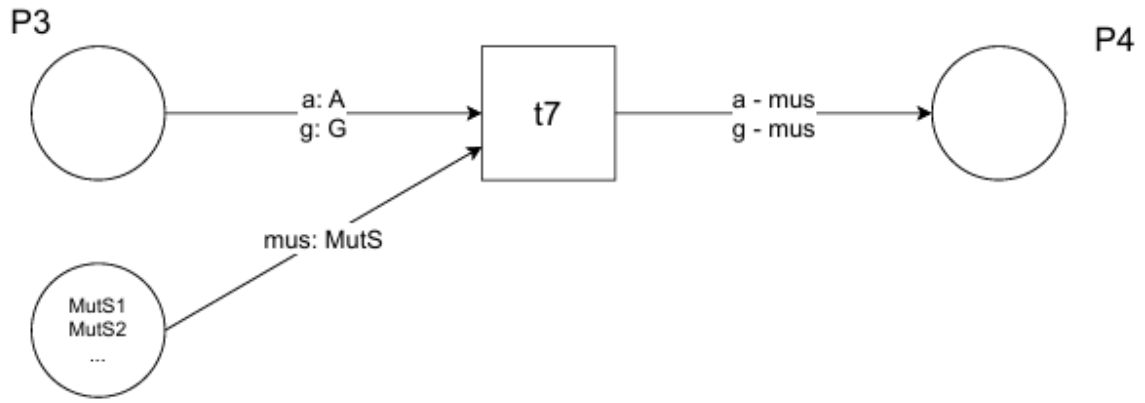


Figure 9. MutS binds to the mismatch

After the methylation process, we can continue with the main reactions in the MMR system, which will show the importance of the extension we have established to the RPN model. As shown in Fig. 9, **MutS attaches to the mismatched bases** of the DNA strand found in place P_3 and the new DNA strand is moved to place P_4 (bonds are created by transition t_7 between A_2 and MutS and between G_3 and MutS). The preconditions for the transition to fire are:

1. $|a|, |g| = 1$
2. $con(S(a), M(x)) = con(S(g),M(x))$, where x the place S(a) and S(g) are found

Precondition 1. targets the two bases that are mismatched (they are the only token instances in the component that are bonded with only one token instance).
Precondition 2. ensures that token instances that are used for a and g are token instances that are part of the same component.

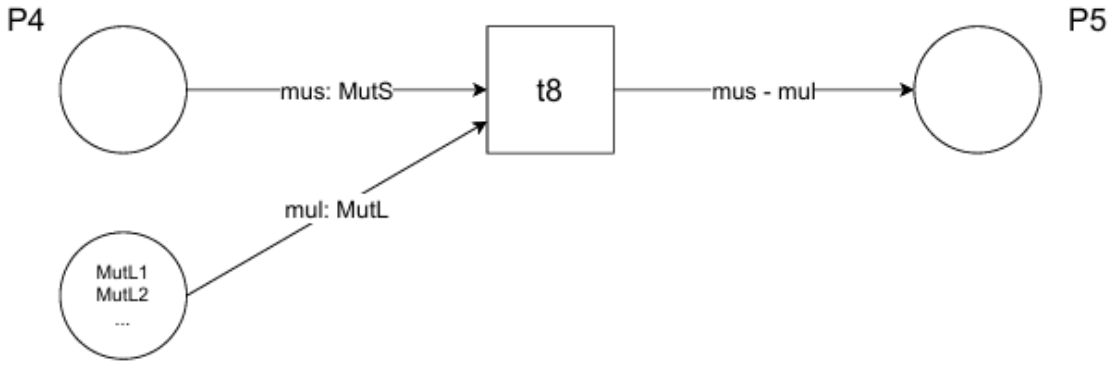


Figure 10. MutL bonds to MutL

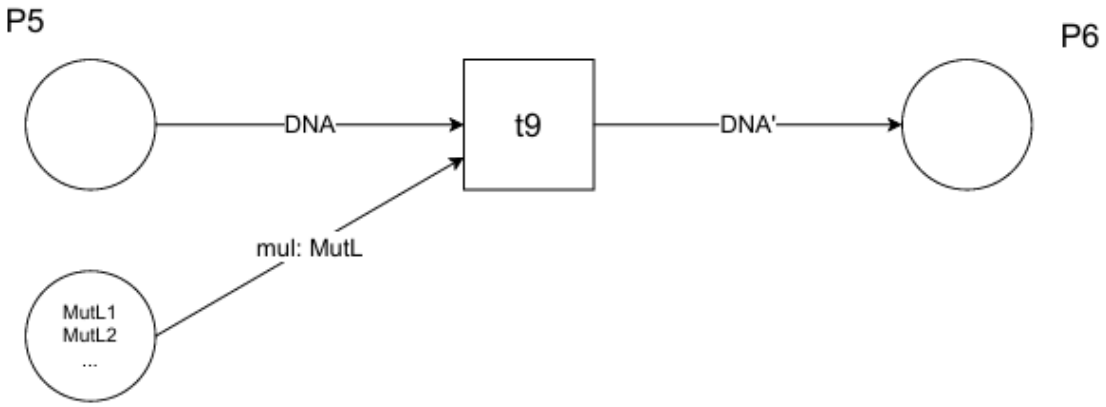


Figure 11. MutL bonds to the DNA strand (on Me)

Now the new proteins **MutL and MutS** are permitted to **bond together** using transition t_8 as shown in Fig. 10, thus the DNA chain from place P_4 is moved to place P_5, followed by the creation of a **bond between MutL and Me** from the DNA strand as shown in Fig 11, where DNA is a label for the exact component used in [20], which is taken from place P_5 to P_6 by adding the bond (MutL, Me), and DNA' = DNA ∪ {(MutL, Me)}.

Then **MutH bonds with MutL**. This is similar to the bonding between MutL and Me as shown in Fig. 11 (after the creation of the bond the DNA chain is moved to place P_7).

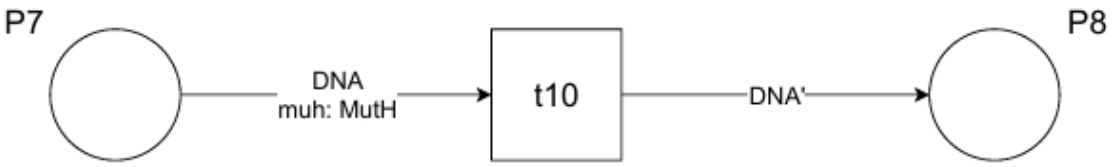


Figure 12. MutH bonds to the DNA strand (on DP_8)

Then **MutH will create a bond with DP_8** through transition t_{10}, as shown in Fig. 12, thus the DNA chain from place P_7 is moved to place P_8, where DNA is a label for the exact component used in [20], and DNA' = DNA ∪ {(MutH, DP_8)}, and due to the creation of this bond, the bond between DP_8 and DP_9 is destroyed. These two actions happen at the same time thus we will use a pair of transitions to implement it.

The transition t_4 (transition t_4 found in Fig. 6, we note that in this case assume that t_4 was only responsible for the creation of the bond between DP_8 and DP_9 and its reversal will not destroy other bonds created while DP_8 and DP_9 were being bonded) is the transition

responsible for the creation of the bond between DP_8 and DP_9 and t_{10} is responsible for the creation of the bond between MutH and DP_8. Thus, we will set into effect ((t_{10}, $\underline{t_4}$),(DP_8, DP_1)) ϵ D_{fb}, where DP_8 is the variable on the arc between the incoming place of t_{10} and t_{10} and DP_1 is the variable in the arc between the incoming place of t_4 and t_4. Therefore, ((t_{10}, $\underline{t_4}$),(DP_8, DP_1)) is concurrently-enabled. 1. is met since the token instance used for both of those variables is DP_8 (the same), 2. is met since in this stage both MutH and DP_8 will be in the correct incoming places of the transition shown in Fig. 12, 3. is met since both token instances DP_8 and DP_9 contain set (k_b, t_4, _) in their histories, since they have been used to fire t_4 for a specific execution k_b, 4. is met since MutH and DP_8 are not bonded token instances, 5. is met since no bonded token instances are sent to different places by transition t_{10} and 6. is met since the following precondition of transition t_{10} to fire is met.

The precondition for the transition to fire is:

1. con(S(c), M(x)) = con(S(muh), M(x)) , where x the place S(c) and S(muh) are found

Precondition 1. ensures that token instances that are used for c and muh are token instances that are part of the same component.

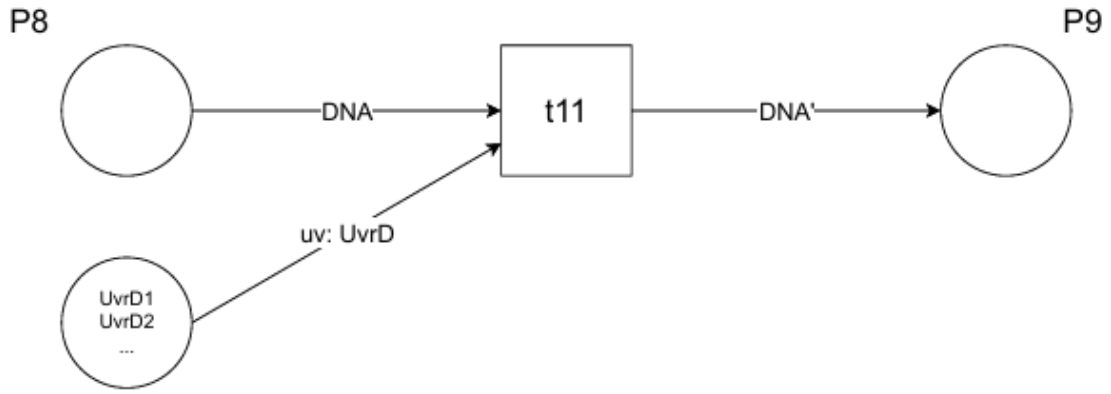


Figure 13. UvrD bonds to the DNA strand (on DP_{10})

Then **UvrD creates a bond with DP_{10}** through transition t_{11}, as shown in Fig. 13, thus the DNA chain from place P_8 is moved to place P_9, where DNA is a label for the exact component used in [20], and DNA' = DNA $\cup$ {(UvrD, DP_{10})}, and due to the creation of this bond, the bond between DP_{10} and DP_9 is destroyed. These two actions happen at the same time thus a pair of transitions is needed to implement it as shown previously.

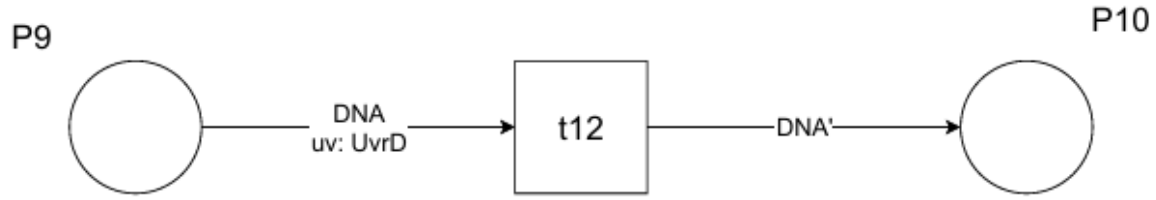


Figure 14. UvrD bonds to the DNA strand (on T_2)

Then **UvrD creates a bond with T_2** through transition t_{12}, as shown in Fig. 14, thus the DNA chain from place P_9 is moved to place P_{10}, where DNA is a label for the exact component used in [20], and DNA' = DNA ∪ {(UvrD, T_2)}, and due to the creation of this bond, the bond between T_2 and A_1 is destroyed. These two actions happen at the same time thus a pair of transitions is needed to implement it as shown previously.

The precondition for the transition to fire is:

1. con(S(c), M(x)) = con(S(uv), M(x)) , where x the place S(c) and S(uv) are found

Precondition 1. ensures that token instances that are used for c and uv are token instances that are part of the same component.

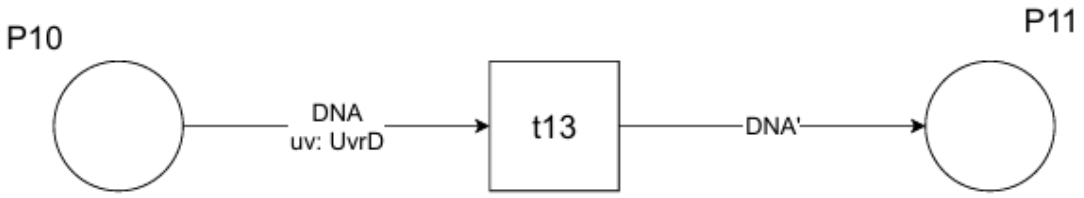


Figure 15. UvrD bonds to the DNA strand (on DP_{11})

Then **UvrD creates a bond with D_{11}** through transition t_{13}, as shown in Fig. 15, thus the DNA chain from place P_{10} is moved to place P_{11}, where DNA is a label for the exact component used in [20], and DNA' = DNA ∪ {(UvrD, DP_{11})}, and due to the creation of this bond, the bonds between UvrD and DP_{10} and between DP_{10} and DP_{11} are destroyed. We will follow the creation and destruction of bonds as they are found in paper [20] therefore we will use a pair of transitions for the creation of the bond between UvrD and DP_{11} and reversal of the bond between UvrD and DP_{10} and then follow up with the reversal of the bond between DP_{10} and DP_{11}. For the first part:

The precondition for the transition to fire is:

1. con(S(c), M(x)) = con(S(uv), M(x)) , where x the place S(c) and S(uv) are found

Precondition 1. ensures that token instances that are used for c and uv are token instances that are part of the same component.

Then **the bond between DP_{10} and DP_{11} is reversed**.

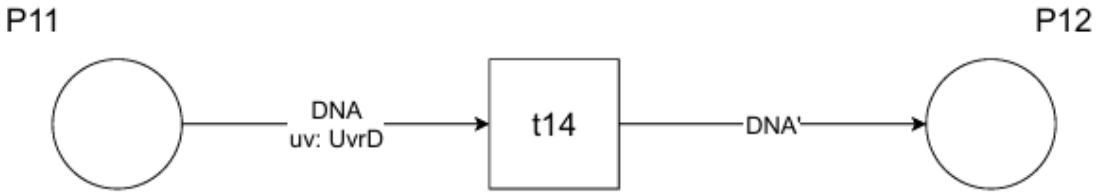


Figure 16. UvrD bonds to the DNA strand (on C_3)

Then **UvrD creates a bond with C_3** through transition t_{14}, as shown in Fig. 16, thus the DNA chain from place P_{11} is moved to place P_{12}, where DNA is a label for the exact component used in [20], and DNA' = DNA ∪ {(UvrD, C_3)}, and due to the creation of this bond, the bond between T_2 and UvrD is destroyed. These two actions happen at the same time thus a pair of transitions is needed to implement it as shown previously.
The precondition for the transition to fire is:

1. con(S(c), M(x)) = con(S(uv), M(x)) , where x the place S(c) and S(uv) are found

Precondition 1. ensures that token instances that are used for c and muh are token instances that are part of the same component.

Then **the bond between C_3 and G_1 is reversed**.

The MMR process continues with similar processes to the ones shown above until the mismatch and the proteins used in the MMR process are not bonded to the DNA strand. Therefore, we have established the importance of the extension we have proposed to the RPNs.

Chapter 5

Comparison with Multi-Reversing Petri Nets

5.1 MRPNs

In this section, we compare the expressive capabilities of our extended model with those of Multi-Reversing Petri Nets (MRPNs) introduced in prior work [19] and in chapter 2. MRPNs extend Reversing Petri Nets by supporting multiple tokens of the same type under the individual token interpretation, enabling causal-order reversing semantics. Our model functions similarly while also introducing additional mechanisms to increase control and expressiveness, particularly in contexts that require more precise regulation of transition behavior.

5.2 Token Multiplicity and Causality

Both MRPNs and our model allow multiple indistinguishable tokens of the same base type to coexist. In both frameworks, these tokens are distinguished solely by their causal paths, which are updated with every transition in which a token participates. This shared foundation ensures that the semantics of token evolution and the structure of bond formation remain consistent between the two models.

5.3 Transition Control

A key point of departure from MRPNs lies in our model's explicit control over transition execution and reversal. In MRPNs, reversibility is implicitly determined by the state of the

tokens and the causal dependencies in the system. In contrast, our model introduces a control structure $D = D_f \cup D_b \cup D_{fb}$, which statically determines whether a transition may fire, be reversed, or participate in a simultaneous action. This allows the user to have more control over the behavior of the system, allowing certain transitions to be designated as forward only or reversible only under specific assignments, as the user pleases.

This distinction proves particularly useful in biological modeling, where irreversible actions (e.g., cleavage, strand excision) must be explicitly constrained, and optional reversal is allowed only under certain preconditions (e.g., the absence of a repair complex). While MRPNs are expressive enough to encode these behaviors indirectly through token constraints, our model provides a more direct and declarative mechanism.

5.4 Simultaneous Transition Execution

Another notable enhancement in our model is the ability to fire and reverse transitions simultaneously. This feature allows transitions t_1 and t_2 to execute in opposite directions in a single step, provided they share a token instance for their variables u and v respectively, and $((t_1,t_2),(u,v)) \in D_{fb}$ holds. MRPNs do not support simultaneous execution natively; instead, such behavior must be modeled by sequences of intermediate steps, potentially introducing auxiliary places or tokens which may be unwanted in some systems.

Simultaneous actions are essential in contexts where coordinated changes must occur, such as enzyme binding coupled with product release, or concurrent structural shifts in macromolecular complexes. By allowing paired transitions to operate in tandem, our model reflects the atomicity of such reactions more faithfully and reduces the modeling overhead required to express them.

5.5 Expressiveness and Encoding

The work on MRPNs shows that for every net with multiple tokens of the same type, there exists a corresponding net with only single-token instances and an isomorphic labeled transition system (LTS), establishing expressiveness equivalence. While our model preserves this baseline result under the individual token interpretation, the addition of control directives and simultaneous execution introduces behaviors that cannot be captured by traditional MRPNs without significant restructuring.

In particular, enforcing irreversible transitions or modeling atomic bidirectional transitions would require encoding control logic externally in MRPNs, potentially obscuring the core net structure. Our model, by contrast, offers native support for such features, allowing them to be expressed directly in the net definition.

5.6 Destroying bonds by the execution of transitions

Another key semantic difference lies in how bond deletion is handled. In MRPNs, the breaking of bonds can occur as part of the forward execution of a transition, allowing transitions to consume bonded components and emit them unconnected, even if the bond was not created by that same transition. This offers flexibility in modeling systems where bond dissolution occurs as part of a transformation or dissociation. In contrast, our model enforces a conservative bond discipline, where the only permissible way to undo a bond is by reversing the specific transition that created it. This restriction ensures a tighter coupling between causal history and bond structure, preserving a form of structural determinism. While this may limit expressiveness in some artificial scenarios, it aligns well with biological intuition, where bond formation and breakage are typically treated as reversible actions with defined energetic or enzymatic triggers. It also supports cleaner out-of-causal-order semantics by preventing unintended structural rewrites through forward transitions alone.

Chapter 6

Conclusion

6.1 Summary

In this paper, we have presented an extension of Reversing Petri Nets that allows for the simultaneous activation of transitions, explicit control over the execution and reversal of transitions, and multiple indistinguishable tokens of the same type. The model can capture out-of-causal-order reversibility and follows the individual token interpretation, in which each token instance is identified by its causal history. More expressive modeling of intricate, concurrent systems where exact coordination and selective reversibility are crucial is made possible by these additions.

The addition of a control structure that statically chooses which transitions may be fired or reversed, or take part in compound actions is a unique aspect of our model. This makes it possible to depict both coordinated behavior through simultaneous execution and irreversible transitions, like those found in biochemical pathways. These capabilities go beyond what is natively supported in Multi-Reversing Petri Nets (MRPNs), offering a more direct and structured means of expressing biological dependencies and process-level control.

We have used two case studies to illustrate the applicability of our approach: DNA mismatch repair (MMR) and catalysis. While the latter emphasizes the necessity of controlled, multi-step activity including binding and coordinated reversal/firing, the former demonstrates how catalysis can be modeled without in between steps which are not true to real life. Using a combination of coordinated and concerted transitions, we were able to accurately mimic the activity of helper proteins and DNA components in MMR. These illustrations demonstrate how well our model captures true biochemical processes that require both structural adaptability and semantic accuracy.

The extension is particularly well-suited to biological systems where reversibility is governed not just by causality but also by biochemical constraints and conditional interactions. In the context of the MMR pathway, our model supports complex bonding and unbonding

sequences and enables abstraction over lower-level steps without sacrificing correctness. In conclusion, the proposed framework is a strong and adaptable extension of reversible Petri nets. It expands the use of net-based models in computational biology and reversible computation by allowing the modeling of highly organized, concurrent, and biologically faithful systems through the incorporation of token multiplicity, transition control, and simultaneous actions.

6.2 Future research

While we abstracted from spatial aspects and reaction rates in the current framework, these remain promising directions for future work.

Our next step is looking into tool support for simulation and verification using the extended model, which includes incorporating spatial limitations and stochastic behaviors. Furthermore, extending the theoretical foundations to encompass richer forms of synchronization and interaction (e.g., multi-party bonds or temporal guards) would further enhance the model's expressiveness and applicability to a broader class of systems. Some weaknesses of this model which we would like to improve is the need to break transitions into sub-transitions to undo a specific effect and the ability to activate/reverse more than two transitions simultaneously.

Bibliography

[1] Vincent Danos & Jean Krivine (2005): *Transactions in RCCS*. In: *Proceedings of CONCUR* 2005, LNCS 3653, Springer, pp. 398–412, doi:10.1007/11539452_31.

[2] Iain Phillips & Irek Ulidowski (2007): *Reversing algebraic process calculi. Journal of Logic and Algebraic Programming* 73(1-2), pp. 70–96, doi:10.1016/j.jlap.2006.11.002.

[3] Ivan Lanese, Claudio Antares Mezzina & Jean-Bernard Stefani (2016): *Reversibility in the higher-order-calculus. Theoretical Computer Science* 625, pp. 25–84, doi:10.1016/j.tcs.2016.02.019.

[4] Irek Ulidowski, Iain Phillips & Shoji Yuen (2014): *Concurrency and Reversibility*. In: *Proceedings of RC* 2014, LNCS 8507, Springer, pp. 1–14, doi:10.1007/978-3-319-08494-7_1.

[5] Anna Philippou & Kyriaki Psara (2022): *Reversible computation in nets with bonds. Journal of Logical and Algebraic Methods in Programming* 124, p. 100718, doi:10.1016/j.jlamp.2021.100718.

[6] Hernán C. Melgratti, Claudio Antares Mezzina & Irek Ulidowski (2020): *Reversing Place Transition Nets. Logical Methods in Computer Science* 16(4). Available at https://lmcs.episciences.org/6843.

[7] Vincent Danos & Jean Krivine (2004): *Reversible Communicating Systems*. In: *Proceedings of CONCUR* 2004, LNCS 3170, Springer, pp. 292–307, doi:10.1007/978-3-540-28644-8_19.

[8] Iain Phillips, Irek Ulidowski & Shoji Yuen (2012): *A Reversible Process Calculus and the Modelling of the ERKSignalling Pathway*. In: *Proceedings of RC* 2012, LNCS 7581, Springer, pp. 218–232, doi:10.1007/978-3-642-36315-3_18.

[9] Iain Phillips & Irek Ulidowski (2015): *Reversibility and asymmetric conflict in event structures. Journal of Logical and Algebraic Methods in Programming* 84(6), pp. 781–805, doi:10.1016/j.jlamp.2015.07.004.

[10] Stefan Kuhn & Irek Ulidowski (2018): *Local reversibility in a Calculus of Covalent Bonding. Science of Computer Programming* 151(Supplement C), pp. 18–47, doi:10.1016/j.scico.2017.09.008.

[11] Rob J. van Glabbeek, Ursula Goltz & Jens-Wolfhard Schicke (2021): *On Causal Semantics of Petri Nets. CoRR* abs/2103.00729. Available at https://arxiv.org/abs/2103.00729.

[12] Jetty Kleijn & Maciej Koutny (2002): *Causality semantics of Petri nets with weighted inhibitor arcs*. In: *Proceedings of CONCUR* 2002, LNCS 2421, Springer, pp. 531–546, doi:10.1007/3-540-45694-5_35.

[13] Grzegorz Rozenberg & Joost Engelfriet (1996): *Elementary net systems*. In: *Advanced Course on Petri Nets*, LNCS1491, Springer, pp. 12–121, doi:10.1007/3-540-65306-6_14.

[14] Angew Chem Int Ed Engl. (2016): *Mechanisms in E. coli and Human Mismatch Repair* July 18; 55(30): 8490–8501. doi:10.1002/anie.201601412.

[15] Anna Philippou & Kyriaki Psara (2022): *A collective interpretation semantics for reversing Petri nets. Theoretical Computer Science* 924, pp. 148–170, doi:10.1016/j.tcs.2022.05.016.

[16] Rob J. van Glabbeek (2005*): The Individual and Collective Token Interpretations of Petri Nets. In: Proceedings of CONCUR* 2005, LNCS 3653, Springer, pp. 323–337, doi:10.1007/11539452_26.

[17] Rob J. van Glabbeek & Gordon D. Plotkin (1995): *Configuration Structures*. In: *Proceedings of LICS* 1995, IEEE Computer Society, pp. 199–209, doi:10.1109/LICS.1995.523257.

[18] Y. Edmund Lien (1976): *A note on transition systems. Information Sciences* 10(4), pp. 347–362, doi:10.1016/0020-0255(76)90054-2.

[19] Anna Philippou & Kyriaki Psara (2022): *Token Multiplicity in Reversing Petri Nets Under the Individual Token Interpretation.* EXPRESS/SOS 2022 EPTCS 368, 2022, pp. 131–150, doi:10.4204/EPTCS.368.8.

[20] Stefan Kuhn & Irek Ulidowski (2022): *Modelling of DNA mismatch repair with a reversible process calculus. Theoretical Computer Science* 925 pp. 68–86, doi.org/10.1016/j.tcs.2022.06.009.

[21] Adel Benamira (2020): *Causal Reversibility in Individual Token Interpretation of Petri Nets. The Computer Science Journal* 21(4), doi:10.7494/csci.2020.21.4.3728.

[22] Philippou, A., Psara, K., Siljak, H. (2019). *Controlling Reversibility in Reversing Petri Nets with Application to Wireless Communications*. In: *Thomsen, M., Soeken, M. (eds) Reversible Computation*. RC 2019. Lecture Notes in Computer Science(), vol 11497. Springer, Cham. https://doi.org/10.1007/978-3-030-21500-2_15.

[23] Anna Philippou & Kyriaki Psara (2020): *Reversible Computation in Petri Nets.* Available at *https://arxiv.org/pdf/2101.07066.*