Encoding Higher Level Extensions of Petri Nets in Answer Set Programming

Saadat Anwar[1], Chitta Baral[1], and Katsumi Inoue[2]

[1] SCIDSE, Arizona State University, 699 S Mill Ave, Tempe, AZ 85281, USA
[2] Principles of Informatics Research Divisions, National Institute of Informatics, Japan

Abstract. Answering realistic questions about biological systems and pathways similar to the ones used by text books to test understanding of students about biological systems is one of our long term research goals. Often these questions require simulation based reasoning. To answer such questions, we need formalisms to build pathway models, add extensions, simulate, and reason with them. We chose Petri Nets and Answer Set Programming (ASP) as suitable formalisms, since Petri Net models are similar to biological pathway diagrams; and ASP provides easy extension and strong reasoning abilities. We found that certain aspects of biological pathways, such as locations and substance types, cannot be represented succinctly using regular Petri Nets. As a result, we need higher level constructs like colored tokens. In this paper, we show how Petri Nets with colored tokens can be encoded in ASP in an intuitive manner, how additional Petri Net extensions can be added by making small code changes, and how this work furthers our long term research goals. Our approach can be adapted to other domains with similar modeling needs.

1 Introduction

One of our long term research objectives is to develop a system that can answer questions similar to the ones given in the biological texts, used to test the understanding of the students. In order to answer such questions, we have to model pathways, add interventions / extensions to them based on the question, simulate them, and reason with the simulation results. We found Petri Nets [1] to be a suitable formalism for modeling biological pathways, as their graphical representation is very close to the biological pathways, and they can be extended to add necessary assumptions and interventions relevant to the questions as shown in our prequel to this paper [2]. Looking through the pathways, we found that certain aspects of biological pathways, such as multiple locations and substance types (perhaps connected to these locations) cannot be represented by regular Petri Nets in a succinct manner.

Consider the simplified Petri Net model of the Electron Transport Chain [3] in Figure 1. The chain removes high energy electrons (e) from NADH ($nadh$) and delivers them to Oxygen ($o2$) by using electron carriers Coenzyme Q (q) and Cytochrome C ($cytc$). During the process, H+ (h) ions are transported

from the Mitochondrial Matrix (mm) to the Intermembrane Space (is). The cross-membrane H+ gradient thus produced drives ATP Synthase (not shown) to produce ATP. The transitions $t1-t4$ represent multi-protein complexes that form the chain. In order for $t1$ to fire, 2×NADH and 2×H+ ($nadh/2, h/2$) are required at the Mitochondrial Matrix (mm). It is clear that the location information embedded in this pathway is a vital part of its model. Regular Petri Nets that were a focus of our previous work [2] cannot capture this location information in a succinct way. In addition, when we use place nodes to represent locations (as in Figure 1), regular (uncolored) tokens do not provide sufficient fidelity to represent various token types needed as input to a transition. As a result, we have to use Petri Nets with colored tokens [4] to model such biological pathways[3]. In contrast to our previous work, the place nodes in this Petri Net model represent locations rather than substances. Even electron-carrier (substances) $q, cytc$ are locations for electrons (e) to be stored and shuttled. Colored tokens also provide a mechanism for differentiating between separate quantities of the same substance present in multiple locations (a common occurrence in biological systems).

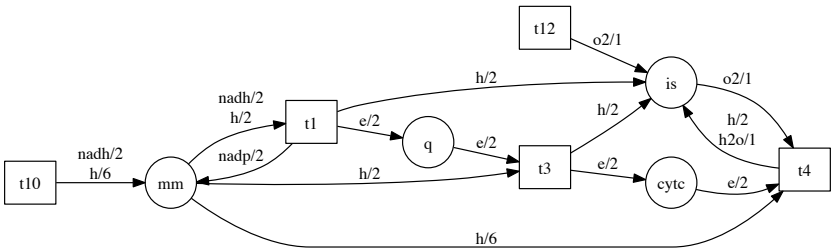

Fig. 1. Petri Net with tokens of colors $\{e, h, h2o, nadh, nadp, o2\}$. Circles represent places, and rectangles represent transitions. Arc weights such as "$nadh/2, h/2$", "$h/2, h2o/1$" specify the number of tokens consumed and produced during the execution of their respective transitions, where "$nadh/2, h/2$" means 2 tokens of color $nadh$ and 2 tokens of h. Similar notation is used to specify marking on places, when not present, the place is assumed to be empty of tokens.

Numerous Petri Net modeling and simulation systems exist [5,6,7,8], but we did not find them to be suitable for our application, either due to limited adaptability outside their intended application domain, limited extendibility, or ease of extendibility. In addition, most systems did not explore all possible state evolutions, allowed different firing semantics, or provided a way to guide the search through specification of partial state as *way-points*. We found the features, such as intuitive encoding, easy extendibility, and strong reasoning capability in Answer Set Programming (ASP), which is a declarative programming language with numerous competitive solvers [9]. It has been effectively used in various domains, such as spacecrafts, work flows, natural language processing, and biological systems [10]. The suitability of ASP to analyze Petri Nets is further reinforced over

[3] Though a Petri Net with colored tokens can be converted into a regular Petri Net without colored tokens, they are usually too large and cumbersome, hence inconvenient.

other techniques, such as process algebra, temporal logics, and mathematical equations when one considers the restrictions on Petri Nets imposed by mathematical techniques [11], the cumbersomeness of encoding in π-calculus even for small models [12], or the lack of applicability to higher level Petri Net extensions.

Previous work on Petri Net to ASP translation has been limited to specific classes of Petri Nets, such as regular Petri Nets [13] and Simple Logic Petri Nets (SLPN) [14], focusing on analyzing their properties. Neither used colored tokens. Please see our previous work [2] for more details. Though our focus in the current work is on biological questions, our approach is equally suited for hypothesis verification during drug design, drug interaction and biological systems model development. It can also be applied to other domains where Petri Nets are used for modeling and simulation, such as work flows, embedded systems, and industrial control.

Thus, the main contributions of this paper are as follows. In Section 3 we show how ASP allows intuitive declarative encoding of higher level Petri Net extension of colored tokens [4]. We then show how additional extensions can be incorporated in our encoding by making small changes. In this regard, we present changing the firing semantics (Section 3.2), priority transitions (Section 4), and timed transitions (Section 5). We show how Petri Nets and our encoding fit into our ultimate research goals of answering questions about biological pathways. We start with a brief background on ASP, multisets, and Petri Nets.

2 Fundamentals

Answer Set Programming (ASP) is a declarative logic programming language based on the Stable Model Semantics [15]. Code presented in this paper follows the Clingo [16] syntax. The reader is referred to [16,17] for the syntax and semantics of Answer Set Programs.

A **multiset** A over a domain set D is a pair $\langle D, m\rangle$, where $m : D \to \mathbb{N}$ is a function giving the multiplicity of $d \in D$ in A. Given two multsets $A = \langle D, m_A\rangle, B = \langle D, m_B\rangle$, $A \odot B$ if $\forall d \in D : m_A(d) \odot m_B(d)$, where $\odot \in \{<, >, \leq, \geq, =\}$, and $A \neq B$ if $\exists d \in D : m_A(d) \neq m_B(d)$. Multiset sum/difference is defined in the usual way. We use the short-hands $d \in A$ to represent $m_A(d) > 0$, $A = \emptyset$ to represent $\forall d \in D, m(d) = 0$, $A \otimes n$ to represent $\forall d \in D, m(d) \otimes n$, where $n \in \mathbb{N}$, $\otimes \in \{<, >, \leq, \geq, =, \neq\}$. We use the notation $d/n \in A$ to represent that d appears n-times in A; we drop A when clear from context. The reader is referred to [18] for details.

A basic **Petri Net** [1] is a bipartite graph of a finite set of place nodes $P = \{p_1, \ldots, p_n\}$, and transition nodes $T = \{t_1, \ldots, t_m\}$ connected through directed arcs $E = E^+ \cup E^-$. An arc goes from a place to a transition $E^- \subseteq P \times T$ or a transition to a place $E^+ \subseteq T \times P$. The state of a Petri Net is defined by the token allocation of all place nodes, collectively called its *marking* $M = (M(p_1), \ldots, M(p_n)), M(p_i) \in \mathbb{N}$. Arc weights $W : E \to \mathbb{N} \setminus \{0\}$ specify the number of tokens consumed or produced from place nodes at the head or tail of the arcs due to firing of a transition. Modeling capability of basic Petri

Nets is enhanced by adding reset, inhibit and read arcs. Reset arcs $R : T \to 2^P$ remove all tokens from their source places when fired. Inhibitor arcs $I : T \to 2^P$ prevent their transitions from firing until their source places are empty. Read arcs $Q \subseteq P \times T$ prevent their transitions from firing until their source places have at least the tokens specified by read arc weights $QW : Q \to \mathbb{N} \setminus \{0\}$.

Higher level Petri Nets extend the notion of tokens to typed (or colored) tokens. A **Petri Net with Colored Tokens** (with reset, inhibit and read arcs) is a tuple $PN^C = (P, T, E, C, W, R, I, Q, QW)$, where P, T, E, R, I, Q are the same as for basic Petri Nets, $C = \{c_1, \ldots, c_l\}$ is a finite set of colors (or types), and arc weights $W : E \to \langle C, m\rangle$, $QW : Q \to \langle C, m\rangle$ are specified as multi-sets of colored tokens over color set C. The state (or marking) of place nodes $M(p_i) = \langle C, m\rangle, p_i \in P$ is specified as a multiset of colored tokens over set C.

We will now define a number of concepts about Petri Nets used in this paper. The **initial marking** is the initial token assignment of place nodes and is represented by M_0. The marking at time-step k is written as M_k. The **pre-set** (or input-set) of a transition t is $\bullet t = \{p \in P | (p, t) \in E^-\}$, while the **post-set** (or output-set) is $t\bullet = \{p \in P | (t, p) \in E^+\}$. A transition t is **enabled** with respect to marking M, $enabled_M(t)$, if each of its input places p has at least the number of colored-tokens as the arc-weight $W(p, t)$[4], each of its inhibiting places $p_i \in I(t)$ have zero tokens and each of its read places $p_q : (p_q, t) \in Q$ have at least the number of colored-tokens as the read-arc-weight $QW(p_q, t)$, i.e. $(\forall p \in \bullet t, W(p, t) \leq M(p)) \wedge (\forall p \in I(t), M(p) = \emptyset) \wedge (\forall (p, t) \in Q, M(p) \geq QW(p, t))$ for a given t. Any number of enabled transitions may fire simultaneously as long as they don't conflict. The set $T_k = \{t_{k_1}, \ldots, t_{k_n}\} \subseteq T$ of such simultaneously firing transitions is called a **firing set**. Execution of a firing set T_k on a marking M_k computes a new marking M_{k+1} as: $\forall p \in P \setminus R(T_k), M_{k+1}(p) = M_k(p) - \sum_{t \in T_k \wedge p \in \bullet t} W(p, t) + \sum_{t \in T_k \wedge p \in t\bullet} W(t, p), \forall p \in R(T_k), M_{k+1}(p) = \sum_{t \in T_k \wedge p \in t\bullet} W(t, p)$, where $R(T_k) = \bigcup_{t \in T_k} R(t)$. A set of transitions $T_c \subseteq \{t : enabled_{M_k}(t)\}$ is in conflict **conflict** in PN^C with respect to M_k if firing them will consume more tokens than are available at one of their common input places, i.e., $\exists p \in P : M_k(p) < (\sum_{t \in T_c \wedge p \in \bullet t} W(p, t) + \sum_{t \in T_c \wedge p \in R(t)} M_k(p))$[5]. An **execution sequence** is the simulation of a firing sequence $\sigma = T_0, T_1, \ldots, T_k$, where each $T_i \subseteq T, 0 \leq i \leq k$ is a firing set. It is the transitive closure of executions, where subsequent markings become the initial marking for the next transition set. Thus in the execution sequence $X = M_0, T_0, M_1, T_1, \ldots, T_k, M_{k+1}$, the firing of T_0 at M_0 produces M_1, which becomes initial marking for T_1.

[4] In the following text, for simplicity, we will use $W(p, t)$ to mean $W(\langle p, t\rangle)$. We use similar simpler notation for QW.

[5] The reset arc is involved here because we use a modified execution semantics of reset arcs compared to the standard definition [19]. Even though both capture similar operation, our definition allows us to model elimination of all quantity of a substance as soon as it is produced, even in a maximal firing set semantics. Our semantics considers reset arc's token consumption in contention with other arcs, while the standard definition does not.

If the Figure 1 Petri Net has the marking: $M_0(mm) = [nadh/2, h/6]$, $M_0(q) = [e/2]$, $M_0(cytc) = [e/2]$, $M_0(is) = [o2/1]$, then transitions $t1, t3, t4$ are enabled. However, either $\{t1, t3\}$ or $\{t4\}$ can fire simultaneously in a single firing at time 0 due to limited h tokens in mm. $t4$ is said to be in conflict with $t1, t3$.

3 Translating Petri Nets with Colored Tokens to ASP

In this section we present an ASP encoding of a Petri Net with Colored Tokens PN^C, with an initial marking M_0 and a simulation length k. This work extends our encoding of regular Petri Nets in [2]. The following sections will show how Petri Net extensions can be easily added to this initial encoding. We represent the Petri Net PN^C with initial marking M_0, and simulation time with the following facts and rules.

f1: Facts `place(`p_i`)` where $p_i \in P$ is a place.
f2: Facts `trans(`t_j`)` where $t_j \in T$ is a transition.
f3: Facts `col(`c_k`)` where $c_k \in C$ is a color.
f4: Rules `ptarc(`p_i, t_j, n_c, c, ts_k`) :- time(`ts_k`).` for each $(p_i, t_j) \in E^-$, $c \in C$, $n_c = m_{W(p_i,t_j)}(c) : n_c > 0$.[6]
f5: Rules `tparc(`t_i, p_j, n_c, c, ts_k`) :- time(`ts_k`).` for each $(t_i, p_j) \in E^+$, $c \in C$, $n_c = m_{W(t_i,p_j)}(c) : n_c > 0$.
f6: Rules `ptarc(`p_i, t_j, n_c, c, ts_k`) :- holds(`p_i, n_c, c, ts_k`), num(`n_c`), `n_c`>0, time(`ts_k`).` for each $(p_i, t_j) : p_i \in R(t_j)$, $c \in C$, $n_c = m_{M_k(p_i)}(c)$.
f7: Rules `iptarc(`$p_i, t_j, 1, c, ts_k$`) :- time(`ts_k`).` for each $(p_i, t_j) : p_i \in I(t_j)$, $c \in C$.
f8: Rules `tptarc(`p_i, t_j, n_c, c, ts_k`) :- time(`ts_k`).` for each $(p_i, t_j) \in Q$, $c \in C$, $n_c = m_{QW(p_i,t_j)}(c) : n_c > 0$.
i1: Facts `holds(`$p_i, n_c, c, 0$`).` for each place $p_i \in P, c \in C, n_c = m_{M_0(p_i)}(c)$.
f9: Facts `time(`ts_i`)` where $0 \leq ts_i \leq k$ are the discrete simulation time steps.
f10: Facts `num(`n`)` where $0 \leq n \leq ntok$ are token quantities[7]

Next, we encode Petri Net's **execution behavior**, which proceeds in discrete time steps. For a transition t_i to be enabled, it must satisfy the following conditions: (i) $\nexists p_j \in \bullet t_i : M(p_j) < E^-(p_j, t_i)$, (ii) $\nexists p_j \in I(t_i) : M(p_j) > 0$, and (iii) $\nexists (p_j, t_i) \in Q : M(p_j) < QW(p_j, t_i)$. These three conditions are encoded as $e1, e2, e3$, respectively and we encode the absence of any of these conditions for a transition as $e4$:

e1:
```
notenabled(T,TS) :- ptarc(P,T, N,C,TS), holds(P,Q,C,TS), place(P),
    trans(T), time(TS), num(N), num(Q), col(C), Q<N.
```
e2:
```
notenabled(T,TS) :- iptarc(P,T,N,C,TS), holds(P,Q,C,TS), place(P),
    trans(T), time(TS), num(N), num(Q), col(C), Q>=N.
```

[6] The time parameter ts_k allows us to capture reset arcs, which consume tokens equal to the current (time-step based) marking of their source nodes.
[7] The token count predicate `num`'s limit can be arbitrarily selected to be higher than the expected token count. It is there for efficient ASP grounding and not to impose a limit on the token quantity.

e3: `notenabled(T,TS) :- tptarc(P,T,N,C,TS), holds(P,Q,C,TS), place(P), trans(T), time(TS), num(N), num(Q), col(C), Q<N.`
e4: `enabled(T,TS) :- trans(T), time(TS), not notenabled(T,TS).`

Rule $e1$ captures the existence of an input place P with insufficient number of tokens for transition T to fire. Rule $e2$ captures existence of a non-empty source place P of an inhibitor arc to T preventing T from firing. Rule $e3$ captures existence of a source place P with less than arc-weight tokens required by the read arc to transition T for T to be enabled. The, `holds(P,Q,C,TS)` predicate captures the marking of place P at time TS as Q tokens of color C. Rule $e4$ captures enabling of transition T when no reason for it to be not enabled is determined by $e1, e2, e3$. In a biological context, this enabling is equivalent to a reaction's pre-conditions being satisfied. A reaction can proceed when its input substances are available in the required quantities, it is not inhibited, and any required activation quantity of activating substances is available.

Any subset of enabled transitions can fire simultaneously at a given time-step. We select a subset of fireable transitions using a choice rule:

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

The choice rule $a1$ either picks an enabled transition T for firing at time TS or not. The combined effect over all transitions is to pick a subset of enabled transitions to fire. Whether these transitions are in conflict are checked by later rules $a2, a3, a4$. In a biological context, the multiple firing models parallel processes occurring simultaneously. The marking is updated according to the firing set using the following rules:

r1: `add(P,Q,T,C,TS) :- fires(T,TS), tparc(T,P,Q,C,TS), time(TS).`
r2: `del(P,Q,T,C,TS) :- fires(T,TS), ptarc(P,T,Q,C,TS), time(TS).`
r3: `tot_incr(P,QQ,C,TS) :- col(C), QQ = #sum[add(P,Q,T,C,TS) = Q : num(Q) : trans(T)], time(TS), num(QQ), place(P).`
r4: `tot_decr(P,QQ,C,TS) :- col(C), QQ = #sum[del(P,Q,T,C,TS) = Q : num(Q) : trans(T)], time(TS), num(QQ), place(P).`
r5: `holds(P,Q,C,TS+1):-place(P),num(Q;Q1;Q2;Q3),time(TS),time(TS+1),col(C), holds(P,Q1,C,TS), tot_incr(P,Q2,C,TS), tot_decr(P,Q3,C,TS), Q=Q1+Q2-Q3.`

Rules $r1$ and $r2$ capture that Q tokens of color C will be added or removed to/from place P due to firing of transition T at the respective time-step TS. Rules $r3$ and $r4$ aggregate these tokens for each C for each place P (using aggregate assignment `QQ = #sum[...]`) at the respective time-step TS. Rule $r5$ uses the aggregates to compute the next marking of P for color C at the time-step $(TS + 1)$ by subtracting removed tokens and adding added tokens to the current marking. In a biological context, this captures the effect of a process / reaction, which consumes its inputs and produces outputs for the downstream processes. We capture token overconsumption using the following rules:

a2: `consumesmore(P,TS) :- holds(P,Q,C,TS), tot_decr(P,Q1,C,TS), Q1 > Q.`
a3: `consumesmore :- consumesmore(P,TS).`

a4: `:- consumesmore.`

Rule $a2$ determines whether firing set selected by $a1$ will cause overconsumption of tokens at P at time TS by comparing available tokens to aggregate tokens removed as determined by $r4$. Rule $a3$ generalizes the notion of overconsumption, while rule $a4$ eliminates answer with such overconsumption. In a biological context, conflict (through overconsumption) models the limitation of input substances, which dictate which downstream processes can occur simultaneously.

Proposition 1. *Let PN^C be a Petri Net with colored tokens, reset, inhibit, and read arcs and M_0 be an initial marking and let $\Pi^3(PN^C, M_0, k)$ be the ASP encoding of PN^C and M_0 over a simulation of length k as defined in Section 3. Then $X^3 = M_0, T_0, M_1, \ldots, T_k$ is an execution sequence of PN^C (with respect to M_0) iff there is an answer-set A of $\Pi^3(PN^C, M_0, k)$ such that: $\{fires(t, j) : t \in T_j, 0 \leq j \leq k\} = \{fires(t, ts) : fires(t, ts) \in A\}$ and $\{holds(p, q, c, j) : p \in P, c/q \in M_j(p), 0 \leq j \leq k\} = \{holds(p, q, c, ts) : holds(p, q, c, ts) \in A\}$*

3.1 Example Execution

Given the above translation rules, the following facts and rules encode the Petri Net in Figure 1 with an initial marking of zero tokens[8]:

```
time(0..5). num(0..30). place(mm;is;q;cytc). trans(t1;t3;t4;t10;t12).
col(nadh;h;e;nadp;h2o;o2). holds(mm,0,nadh,0). holds(mm,0,h,0).
tparc(t12,is,1,o2,TS):-time(TS). tparc(t10,mm,6,h,TS):-time(TS).
tparc(t10,mm,2,nadh,TS):-time(TS). ptarc(mm,t1,2,nadh,TS):-time(TS).
```

We get thousands of answer-sets, for example[9]:

```
fires(t10;t12,0) holds(is,1,o2,1) holds(mm,6,h,1) holds(mm,2,nadh,1)
fires(t1;t10;t12,1) holds(is,2,h,2) holds(is,2,o2,2) holds(mm,10,h,2)
   holds(mm,2,nadh,2) holds(mm,2,nadp,2) holds(q,2,e,2)
fires(t1;t3;t10;t12,2) holds(cytc,2,e,3) holds(is,6,h,3) holds(is,3,o2,3)
   holds(mm,12,h,3) holds(mm,2,nadh,3) holds(mm,4,nadp,3) holds(q,2,e,3)
fires(t1;t3;t4;t10;t12,3) holds(cytc,2,e,4) holds(is,12,h,4)
   holds(is,1,h2o,4) holds(is,3,o2,4) holds(mm,8,h,4) holds(mm,2,nadh,4)
   holds(mm,6,nadp,4) holds(q,2,e,4)
fires(t3;t4;t10;t12,4) holds(cytc,2,e,5) holds(is,16,h,5)
   holds(is,2,h2o,5) holds(is,3,o2,5) holds(mm,6,h,5) holds(mm,4,nadh,5)
   holds(mm,6,nadp,5) fires(t1;t10;t12,5)
```

3.2 Changing Firing Semantics

The above code implements a *set firing* semantics, which can produce a large number of answer-sets[10]. In biological domain, it is often preferable to simulate

[8] We show a few of the `tparc/5, ptarc/5, holds/4` to illustrate the approach, the rest of them can be encoded in a similar fashion.

[9] We are only showing colored tokens with non-zero quantity. Also, we are representing `fires(t1,ts),...,fires(tm,ts)` as `fires(t1;...;tm,ts)` to conserve space.

[10] A subset of a firing set can also be fired as a firing set by itself.

the maximum parallel activity at each time step. We accomplish this by enforcing the **maximal firing set** semantics by extending its encoding for regular Petri Nets in [2] to colored tokens as follows:

a5: `could_not_have(T,TS):-enabled(T,TS),not fires(T,TS), ptarc(S,T,Q,C,TS), holds(S,QQ,C,TS), tot_decr(S,QQQ,C,TS), Q > QQ - QQQ.`

a6: `:- not could_not_have(T,TS), time(TS), enabled(T,TS), not fires(T,TS), trans(T).`

Rule $a5$ captures the fact that transition T, though enabled, could not have fired at TS, as its firing would have caused overconsumption. Rule $a6$ eliminates any answers where an enabled transition could have fired without causing overconsumption but did not. This modification reduces the number of answers produced for the Petri Net in Figure 1 to 4. We can encode other firing semantics with similar ease[11]. We now look at how additional extensions can be easily encoded by making small code changes.

4 Extension - Priority Transitions

Priority transitions enable ordering of Petri Net transitions, favoring high priority transitions over lower priority ones [20]. In a biological context, this is used to model primary (or dominant) vs. secondary pathways / processes in a biological system. This prioritization may be due to an intervention (such as prioritizing elimination of a metabolite over recycling it).

A **Priority Colored Petri Net** with reset, inhibit, and read arcs is a tuple $PN^{pri} = (P, T, E, C, W, R, I, Q, QW, Z)$, where: $P, T, E, C, W, R, I, Q, QW$ are the same as for PN^C, and $Z : T \to \mathbb{N}$ is a priority function that assigns priorities to transitions. Lower number signifies higher priority. A **transition t_i is enabled in** PN^{pri} if it would be enabled in PN^C (with respect to M) and there isn't another transition t_j that would be enabled in PN^C (with respect to M) s.t. $Z(t_j) < Z(t_i)$. We add the following facts and rules to encode transition priority and enabled priority transitions:

f11: Facts `transpr(`t_i`,`pr_i`)` where pr_i is $t_i's$ priority.

a7: `notprenabled(T,TS) :- enabled(T,TS), transpr(T,P), enabled(TT,TS), transpr(TT,PP), PP < P.`

a8: `prenabled(T,TS) :- enabled(T,TS), not notprenabled(T,TS).`

Rule $a7$ captures that an enabled transition T is not priority-enabled, if there is another enabled transition with higher priority at TS. Rule $a8$ captures that transition T is priority-enabled at TS since there is no enabled transition with higher priority. We replace rules $a1, a5, a6$ with $a9, a10, a11$ respectively to propagate priority as follows:

[11] For example, if *interleaved* semantics is desired, rules $a5, a6$ can changed to capture and eliminate answer-sets in which more than one transition fires in a firing set as:

a5': `more_than_one_fires :- fires(T1,TS),fires(T2,TS),T1!=T2,time(TS).`

a6': `:-more_than_one_fires.`

a9: `{fires(T,TS)} :- prenabled(T,TS), trans(T), time(TS).`
a10: `could_not_have(T,TS) :- prenabled(TS,TS), not fires(T,TS),`
`ptarc(S,T,Q,C,TS), holds(S,QQ,C,TS), tot_decr(S,QQQ,C,TS), Q > QQ - QQQ.`
a11: `:- prenabled(tr,TS), not fires(tr,TS), time(TS).`

Rules $a9, a10, a11$ perform the same function as $a1, a5, a6$, except that they consider only priority-enabled transitions as compared all enabled transitions.

Proposition 2. *Let PN^{pri} be a Petri Net with colored tokens, reset, inhibit, read arcs and priority based transitions and M_0 be an initial marking and let $\Pi^5(PN^{pri}, M_0, k)$ be the ASP encoding of PN^{pri} and M_0 over a simulation of length k as defined in Section 4. Then $X^5 = M_0, T_0, M_1, \ldots, T_k$ is an execution sequence of PN^{pri} (with respect to M_0) iff there is an answer-set A of $\Pi^5(PN^{pri}, M_0, k)$ such that: $\{fires(t,j) : t \in T_j, 0 \leq j \leq k\} = \{fires(t,ts) : fires(t,ts) \in A\}$, and $\{holds(p,q,c,j) : p \in P, c/q \in M_j(p), 0 \leq j \leq k\}$ $= \{holds(p,q,c,ts) : holds(p,q,c,ts) \in A\}$*

5 Extension - Timed Transitions

Biological processes vary in time required for them to complete. Timed transitions [21] model this variation of duration. The timed transitions can be reentrant or non-reentrant[12]. We extend our encoding to allow reentrant timed transitions.

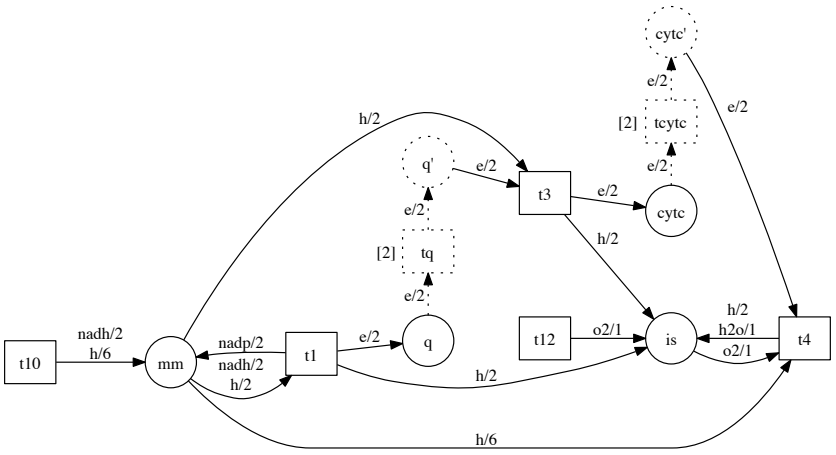

Fig. 2. An extended version of the Petri Net model from Fig. 1. The new transitions $tq, tcytc$ have a duration of 2 each (shown in square brackets ("[ ]") next to the transition). When missing, transition duration is assumed to be 1.

[12] A **reentrant** transition is like a vehicle assembly line, which accepts new parts while working on multiple vehicles at various stages of completion; whereas a **non-reentrant** transition only accepts new input when the current processing is finished.

A **Priority Colored Petri Net with Timed Transitions**, reset, inhibit, and query arcs is a tuple $PN^D = (P, T, E, C, W, R, I, Q, QW, Z, D)$, where $P, T, E, C, W, R, I, Q, QW, Z$ are the same as for PN^{pri}, and $D : T \rightarrow \mathbb{N} \setminus \{0\}$ is a duration function that assigns positive integer durations to transitions.

Figure 2 shows an extended version of Petri Net model of the Electron Transport Chain [3] shown in Figure 1. The new transitions *tq* and *tcytc* (shown in dotted outline) are timed transitions modeling the speed of the small carrier molecules, Coenzyme Q (q) and Cytochrome C ($cytc$) as an effect of membrane fluidity. Higher numbers for transition duration represent slower movement of the carrier molecules due to lower fluidity. **Execution in** PN^D changes, since the token update from M_k to M_{k+1} can involve transitions that started at some time l before time k, but finish at $k+1$. Thus, the new marking is computed as follows: $\forall p \in P \setminus R(T_k), M_{k+1}(p) = M_k(p) - \sum_{t \in T_k \wedge p \in \bullet t} W(p,t) + \sum_{t \in T_l \wedge p \in t\bullet : l \leq k, l+D(t)=k+1} W(t,p)$, and $\forall p \in R(T_k)$,
$M_{k+1}(p) = \sum_{t \in T_l \wedge p \in t\bullet : l \leq k, l+D(t)=k+1} W(t,p)$, where $R(T_i) = \cup_{t \in T_i} R(t)$.

A timed transition t produces its output $D(t)$ time units after being fired. We replace $f5$ with $f12$ adding transition duration and replace rule $r1$ with $r6$ that produces tokens at the end of transition duration [13]:

f12: Rules `tparc(`$t_i, p_j, n_c, c, ts_k, D(t_i)$`):-time(`$ts_k$`).` for each $(t_i, p_j) \in E^+$, $c \in C$, $n_c = m_{W(t_i,p_j)}(c) : n_c > 0$.

r6:
```
add(P,Q,T,C,TSS):-fires(T,TS),time(TS;TSS), tparc(T,P,Q,C,TS,D),
TSS=TS+D-1.
```

Proposition 3. *Let PN^D be a Petri Net with colored tokens, reset, inhibit, read arcs and priority based timed transitions and M_0 be an initial marking and let $\Pi^6(PN^D, M_0, k)$ be the ASP encoding of PN^D and M_0 over a simulation of length k as defined in Section 5. Then $X^6 = M_0, T_0, M_1, \ldots, T_k$ is an execution sequence of PN^D (with respect to M_0) iff there is an answer-set A of $\Pi^6(PN^D, M_0, k)$ such that: $\{fires(t,j) : t \in T_j, 0 \leq j \leq k\} = \{fires(t,ts) : fires(t,ts) \in A\}$, and $\{holds(p,q,c,j) : p \in P, c/q \in M_j(p), 0 \leq j \leq k\}$
$= \{holds(p,q,c,ts) : holds(p,q,c,ts) \in A\}$*

6 Example Use of Our Encoding and Reasoning Abilities

We illustrate the usefulness of our encoding by applying it to the following simulation based reasoning question[14] from [3]: "Membranes must be fluid to function properly. How would decreased fluidity of the membrane affect the efficiency of the electron transport chain?"

[13] We can easily make these timed transitions non-reentrant by adding rule $e5$ that disallows a transition from being enabled if it is already in progress:

e5:
```
notenabled(T,TS1):-fires(T,TS0), num(N), TS1>TS0,
tparc(T,P,N,C,TS0,D), col(C), time(TS0), time(TS1), TS1<(TS0+D).
```

[14] As it appeared in `https://sites.google.com/site/2nddeepkrchallenge/`

To answer this question, first we build a Petri Net model of the Electron Transport Chain, including its interplay with the membrane. Our model is shown in Figure 1. We base our model on [3, Figure 9.15], which shows multiple substances flowing through the protein complexes that form this chain. For example, the first complex ($t1$), removes electrons from NADH ($nadh$) arriving at the Mitochondrial Matrix (mm) and delivers them to the mobile carrier Coenzyme Q (q), converting NADH to NAD+ ($nadp$). As a side effect, $t1$ also moves H+ (h) ions from mm to the Intermembrane Space (is). The speed at which q (which lives in the membrane) shuttles electrons to next complex depends upon the membrane fluidity.

To answer the question, we need to model change in fluidity and its impact on the mobile carriers. Background knowledge tells us that lower fluidity leads to slower movement of mobile carriers, leading to longer transit times. We model this using an intervention to the Petri Net model of Figure 1, extending it with additional delay transitions in the path of q and Cytochrome C ($cytc$), as shown with dotted outline in Figure 2. We encode both models in ASP using encodings from Sections 3 and 5, respectively, and simulate them for a fixed number of time-steps ts using the *maximal firing set* semantics from Section 3.2. A plot of H+ produced over time is shown in Figure 3. We compute the efficiency of the electron transport chain as the quantity of H+ ("h") ions moved (from "mm") to "is" over the simulation duration "ts", i.e. "$\frac{h}{ts}$". We found that this value decreased from 4.5 to 3 with decrease of membrane fluidity (modeled as timed transitions of duration 2). Thus, our results show that decreased fluidity of the membrane results in lowering the efficiency of the electron transport chain.

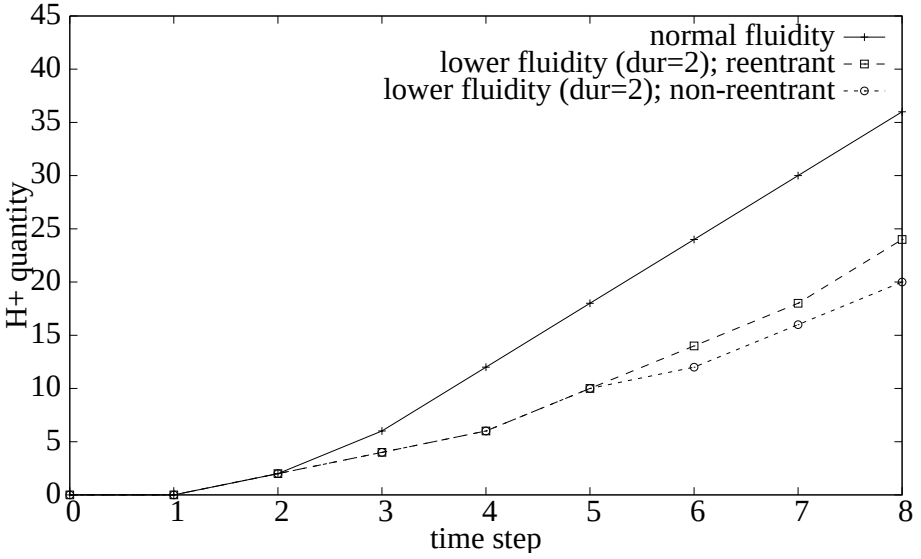

Fig. 3. H+ production in the intermembrane space over time for the normal fluidity, lower fluidity (reentrant), and lower fluidity (non-reentrant transitions).

If we permit additional background knowledge about the mobile carriers, we can refine our ASP encoding, modeling the mobile carriers with non-reentrant timed transitions (Section 5 Footnote 13) [15]. Repeating our simulation with non-reentrant timed transitions results in an efficiency value of 2.5, which is a larger reduction in the efficiency of the electron chain due to decreased fluidity.

ASP's enumeration of the entire simulation evolution allows us to perform additional reasoning not directly possible with Petri Nets. For example, partial state or firing sequence can be encoded (as ASP constraints) as *way-points* to guide the simulation. A simple use case is to enumerate answer-sets where a transition t fires when one of its upstream source products S is found to be depleted. These answer-sets are used to identify another upstream substance responsible for t's firing. Our encoding allows various Petri Net dynamic and structural properties to be easily analyzed, as described in our previous work [2].

7 Related Work and Conclusion

Now we look at some of the existing Petri Net systems that support higher level Petri Net constructs[16] (focusing on the ones used for biological modeling). We also look at some ways existing Petri Net tools are used for biological analysis and put them in context of our research.

CPN Tools [5], Renew [8], Snoopy [6] all support Colored Petri Nets. All but CPN Tools directly support inhibit and reset arcs. All but Snoopy are limited to one particular firing semantics, while Snoopy allows three distinct firing semantics. Neither pursues more than one simulation and all break ties arbitrarily. Cell Illustrator [7] does not support colored tokens, but does provide a rich graphical environment with pathway representation similar to standard biological pathways. It also only supports one possible evolution.

Petri Nets have been previously used to analyze biological pathways [22,23,24], but most of this analysis has been limited to dynamic and structural properties of the Petri Net model. [25] took a different approach, where they surveyed the Petri Net implementations and came up with questions answerable by each. Contrary to the previous work, we focus on real world biological questions as they appear in college level biological text books; we model these questions as Petri Net extensions; and leverage ASP as a rich reasoning environment.

Conclusion: In this paper we presented the suitability of using Petri Nets with colored tokens for modeling biological pathways. We showed how such Petri Nets can be intuitively encoded in ASP, simulated, and reasoned with, in order to answer real world questions posed in the biological texts. We showed how our initial encoding can be easily extended to include additional extensions, such as maximal firing semantics, priority transitions, and timed transitions. Our encoding has a low specification-implementation gap, it allows enumeration of

[15] Similar modeling is also possible by using inhibitor arcs from mobile carriers to the transitions preceding them.

[16] The Petri Net Tools Database web-site summarizes a large slice of existing tools `http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html`

all possible state evolutions, the ability to guide the search by specifying waypoints (such as partial state), and a strong reasoning ability. Our focus in this work is more on encoding flexibility, exploring all possible state evolutions, and reasoning capabilities; and less on performance. We showcased the usefulness of our encoding by an example. We briefly compared our work to other Petri Net systems and their use in biological modeling and analysis. In follow on papers, we will extend our work to include the High-level Petri Net standard.

References

1. Petri, C.A.: Kommunikation mit Automaten. Technical report, Institut für Instrumentelle Mathematik, Bonn, Germany (1962)
2. Anwar, S., Baral, C., Inoue, K.: Encoding Petri nets in Answer Set Programming for simulation based reasoning. `http://arxiv.org/abs/1306.3542` (2013)
3. Reece, J., Cain, M., Urry, L., Minorsky, P., Wasserman, S.: Campbell Biology. Pearson Benjamin Cummings (2010)
4. Peterson, J., et al.: A note on colored Petri nets. Information Processing Letters **11**(1) (1980) 40–43
5. Jensen, K., Kristensen, L., Wells, L.: Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. International Journal on Software Tools for Technology Transfer (STTT) **9**(3) (2007) 213–254
6. Heiner, M., Herajy, M., Liu, F., Rohr, C., Schwarick, M.: Snoopy - a unifying Petri net tool. In: Application and Theory of Petri Nets. Volume 7347 of Lecture Notes in Computer Science. (2012) 398–407
7. Nagasaki, M., Saito, A., Jeong, E., Li, C., Kojima, K., Ikeda, E., Miyano, S.: Cell illustrator 4.0: A computational platform for systems biology. In Silico Biology **10**(1) (2010) 5–26
8. Kummer, O., Wienberg, F., Duvigneau, M.: Renew–the reference net workshop. Petri Net Newsletter **56** (1999) 12–16
9. Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., Truszczyński, M.: The first answer set programming system competition. In: Logic Programming and Nonmonotonic Reasoning. Springer Berlin Heidelberg (2007) 3–17
10. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Communications of the ACM **54**(12) (2011) 92–103
11. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4) (1989) 541–580
12. van der Aalst, W.: Pi calculus versus Petri nets: Let us eat "humble pie" rather than further inflate the "pi hype". BPTrends **3**(5) (2005) 1–11
13. Heljanko, K., Niemelä, I.: Bounded LTL model checking with stable models. In: Logic Programming and Nonmotonic Reasoning. Springer (2001) 200–212
14. Behrens, T.M., Dix, J.: Model Checking with Logic Based Petri Nets. Technical report ifi-07-02, Insitut für Informatik, Technische Universität Clausthal Julius-Albert Str. 4, 38678 Clausthal-Zellerfeld, Germany, Clausthal University of Technology, Dept of Computer Science (May 2007)
15. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. Logic Programming: Proceedings of the Fifth International Conference and Symposium (1988) 1070–1080

16. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Schneider, M.: Potassco: The Potsdam answer set solving collection. aicom **24**(2) (2011) 105–124
17. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
18. Syropoulos, A.: Mathematics of multisets. Multiset Processing (2001) 347–358
19. Araki, T., Kasami, T.: Some decision problems related to the reachability problem for Petri nets. Theoretical Computer Science **3**(1) (1976) 85–104
20. Best, E., Koutny, M.: Petri net semantics of priority systems. Theoretical Computer Science **96**(1) (1992) 175–215
21. Ramchandani, C.: Analysis of asynchronous concurrent systems by Petri nets. Technical report, DTIC Document (1974)
22. Sackmann, A., Heiner, M., Koch, I.: Application of Petri net based analysis techniques to signal transduction pathways. BMC Bioinformatics **2**(7) (November 2006) 482
23. Hofestädt, R., Thelen, S.: Qualitative modeling of biochemical networks. In Silico Biology **1** (1998) 39–53
24. Li, C., Suzuki, S., Ge, Q.W., Nakata, M., Matsuno, H., Miyano, S.: Structural modeling and analysis of signaling pathways based on Petri nets. Journal of bioinformatics and computational biology **4**(05) (2006) 1119–1140
25. Peleg, M., Rubin, D., Altman, R.B.: Using Petri net tools to study properties and dynamics of biological systems. Journal of the American Medical Informatics Association **12**(2) (2005) 181–199