TPLP **13** (4-5): Online Supplement, July 2013.

Encoding Petri Nets in Answer Set Programming for Simulation Based Reasoning

SAADAT ANWAR, CHITTA BARAL
SCIDSE, Arizona State University, 699 S Mill Ave, Tempe, AZ 85281, USA

KATSUMI INOUE
Principles of Informatics Research Division, National Institute of Informatics, Japan

submitted 10 April 2013; accepted 23 May 2013

Abstract

One of our long term research goals is to develop systems to answer realistic questions (e.g., some mentioned in textbooks) about biological pathways that a biologist may ask. To answer such questions we need formalisms that can model pathways, simulate their execution, model intervention to those pathways, and compare simulations under different circumstances. We found Petri Nets to be the starting point of a suitable formalism for the modeling and simulation needs. However, we need to make extensions to the Petri Net model and also reason with multiple simulation runs and parallel state evolutions. Towards that end Answer Set Programming (ASP) implementation of Petri Nets would allow us to do both. In this paper we show how ASP can be used to encode basic Petri Nets in an intuitive manner. We then show how we can modify this encoding to model several Petri Net extensions by making small changes. We then highlight some of the reasoning capabilities that we will use to accomplish our ultimate research goal.

KEYWORDS: Petri Nets, ASP, Knowledge Representation and Reasoning, Modeling and Simulation

1 Introduction

The main motivation behind this paper [1] is to model biological pathways and answer questions of the kind that a biologist would ask. Examples of such questions include ones used in biology text books to test the understanding of students. We found Petri Nets (Petri 1962) to be the most suitable starting point for our needs as its graphical representation and semantics closely matches biological pathway diagrams.[2] However, answering those type of questions requires certain extensions to the Petri Net model and reasoning with multiple simulations and parallel evolutions. Although numerous Petri Net modeling, simulation and analysis systems exist (see (Anwar et al. 2013) for references), we did not find these systems to be a good match for all our needs. Some had limited adaptability outside their primary application domain, while others, though quite capable, did not offer easy extensibility. Most systems did not explore all possible state evolutions nor allowed different firing semantics.

To address the twin needs of easy extensibility and reasoning over multiple evolutions we

[1] For full version, see: (Anwar et al. 2013)
[2] Petri Nets are also used over a wide range of domains, such as, workflows, embedded systems and industrial control.

propose encoding Petri Nets using Answer Set Programming (ASP), which is a declarative programming language with competitive solvers and has been used in modeling domains such as spacecrafts, work flows, natural language processing and biological systems modeling. In our quest we found ASP to be preferable to process algebra, temporal logics, and mathematical equations applied to Petri Nets. Some of these techniques, like π-calculus is cumbersome even for small Nets, while others, like mathematical equations, impose restrictions on the classes of Petri Nets they can model (see (Anwar et al. 2013) for references).

Petri Net translation to ASP has been studied before (Heljanko and Niemelä 2000; Behrens and Dix 2007). However, these implementations are limited to specific classes of Petri Nets and have a different focus. For example, (Heljanko and Niemelä 2000) used ASP for the analysis of properties of *1-safe* Petri Nets such as reachability and deadlock detection. *1-safe* Petri Nets are very basic in nature as they can accommodate at most one token in a place and they don't allow source/sink transitions or inhibition arcs. As a result their ASP translation does not handle token aggregation, leading to simpler conflict resolution and weight-constraints than the general case. In (Behrens and Dix 2007), a new class of Petri Nets called Simple Logic Petri Nets is presented and translated to ASP code. This class of Petri Nets uses simple logic expressions for arc weights and positive ground literals as tokens. Their implementation does not carry the regular Petri Net notion of conflicting transitions, token consumption or aggregation. A token from one source place can be consumed by multiple transitions in a single firing step and the same token arriving at a place from two different transitions looses its identity and gets counted as one. Both of these implementations focus on analyzing properties of specific classes of Petri Nets. We design our implementation to simulate general Petri Nets. To our knowledge this has not been attempted in ASP before. However, some of our encoding scheme is similar to their work.

Thus the main contributions of this paper include showing how ASP allows intuitive declarative encoding of basic Petri Nets resulting in low specification-implementation gap; how our initial ASP encoding allows inclusion of additional Petri Net extensions by making small changes; how ASP encoding allows all possible state evolutions and reason with them; and how this work fits into our ultimate goal of answering questions. We also briefly touch on how some of Petri Net's structural and dynamic properties can be analyzed using our encoding. We now start with a brief background on ASP and Petri Nets.

2 Background on ASP and Petri Nets

Answer Set Programming (ASP) is a declarative logic programming language based on the Stable Model Semantics. For ASP syntax used in this paper, see (Gebser et al. 2011).

A Petri Net is a graph of a finite set of nodes and directed arcs, where nodes are split between places and transitions, and each arc either connects a place to a transition or a transition to a place. Each place has a number of tokens (called the its marking). Collective marking of all places in a Petri Net is called its *marking* (or *state*). Arc labels represent arc weights. When missing, arc-weight is assumed as one, and place marking is assumed as zero. A transition t is enabled when each of its input places p has at least the number of tokens equal to the arc weight from p to t. An enabled transition may fire consuming tokens from its input places equal to the arc weight and producing tokens to each of its output place equal to arc weight to the output place. Multiple transitions may fire, such that they do not consume more than available tokens, with the assumption that tokens cannot be shared. Fig. 1 shows a portion of the glycolysis path-

way (Reece et al. 2010). Places represent reactants and products, transitions represent reactions, and arc weights represent reactant / product quantity consumed / produced. We now introduce some concepts used in the paper.

A **Petri Net** is a tuple $PN = (P, T, E, W)$, where, $P = \{p_1, \ldots, p_n\}$ is a finite set of places; $T = \{t_1, \ldots, t_m\}$ is a finite set of transitions, $P \cap T = \emptyset$; $E^+ \subseteq T \times P$ is a set of arcs from transitions to places; $E^- \subseteq P \times T$ is a set of arcs from places to transitions; $E = E^+ \cup E^-$; and $W : E \rightarrow \mathbb{N} \setminus \{0\}$ is the arc-weight function. A **marking** $M = (M(p_1), \ldots, M(p_n))$ is the token assignment of each place node $p_i \in P$, where $M(p_i) \in \mathbb{N}$. Initial token assignment $M_0 : P \rightarrow \mathbb{N}$ is called the initial marking. Marking at step k is written as M_k. **Pre-set** / input-set of a transition t is $\bullet t = \{p \in P : (p,t) \in E^-\}$, while the **post-set** / 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 $\forall p \in \bullet t, W(p,t) \leq M(p)$. An enabled transition may fire. An **execution** is the simulation of change of marking from M_k to M_{k+1} due to firing of transition t. M_{k+1} is computed as follows: $\forall p_i \in \bullet t, M_{k+1}(p_i) = M_k(p_i) - W(p_i,t)$ and $\forall p_j \in t\bullet, M_{k+1}(p_j) = M_k(p_j) + W(t,p_j)$ A set of enabled transitions $T_e = \{t \in T : enabled_M(t)\}$ **conflict** if their simultaneous firing will consume more tokens than are available at an input place: $\exists p \in P : M(p) < \sum_{t \in T_e \wedge (p,t) \in E^-} W(p,t)$. A set of simultaneously firing, non-conflicting, enabled transitions $T_k = \{t_1, \ldots, t_m\} \subseteq T$ is called a **firing set**. Its execution w.r.t. marking M_k produces new marking M_{k+1} as follows: $\forall p \in P, M_{k+1}(p) = M_k(p) - \sum_{t \in T_k \wedge p \in \bullet t} W(p,t) + \sum_{t \in T_k \wedge p \in t\bullet} W(t,p)$. An **execution sequence** is the simulation of a firing sequence $\sigma = T_1, T_2, \ldots, T_k$. It is the transitive closure of executions, where subsequent markings become the initial marking for the next firing set. Thus, in the execution sequence $X = M_0, T_0, M_1, T_1, \ldots, M_k, T_k, M_{k+1}$, the firing of T_0 with respect to marking M_0 produces the marking M_1 which becomes the initial marking for T_1.

3 Translating Basic Petri Net Into ASP

In this section we present ASP encoding of simple Petri Nets. We describe, how a given Petri Net PN, and an initial marking M_0 are encoded into ASP for a simulation length k. Following sections will show how Petri Net extensions can be easily added to it. We encode a Petri Net, its initial marking (or initial state), and its simulation as follows:

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 `ptarc(`$p_i, t_j, W(p_i,t_j)$`).` where $(p_i, t_j) \in E^-$ with weight $W(p_i, t_j)$.
f4: Facts `tparc(`$t_i, p_j, W(t_i,p_j)$`).` where $(t_i, p_j) \in E^+$ with weight $W(t_i, p_j)$.
i1: Facts `holds(`$p_i, M_0(p_i), 0$`)` for every place $p_i \in P$ with initial marking $M_0(p_i)$.
f5: Facts `time(`ts_i`).` where $0 \leq ts_i \leq k$ for each discrete execution time-step.
x1: Facts `num(`n`).`, where $0 \leq n \leq ntok$, where *ntok* is the max. token count at a place [3]

```
e1: notenabled(T,TS):-ptarc(P,T,N),holds(P,Q,TS),Q<N, place(P), trans(T),
  time(TS),num(N),num(Q).
e2: enabled(T,TS) :- trans(T), time(TS), not notenabled(T, TS).
a1: {fires(T,TS)} :- enabled(T,TS), trans(T), time(TS).
r1: add(P,Q,T,TS) :- fires(T,TS), tparc(T,P,Q), time(TS).
r2: del(P,Q,T,TS) :- fires(T,TS), ptarc(P,T,Q), time(TS).
r3: tot_incr(P,QQ,TS) :- QQ=#sum[ add(P,Q,T,TS)=Q : num(Q) : trans(T)],
  time(TS), num(QQ), place(P).
```

[3] Note that *ntok* can be arbitrarily chosen to be larger than the maximum expected token quantity produced during the simulation and hence is not used to enforce token count restriction. `num` atoms are there for proper grounding.

r4: `tot_decr(P,QQ,TS) :- QQ = #sum[ del(P,Q,T,TS)=Q : num(Q) : trans(T)], time(TS), num(QQ), place(P).`
r5: `holds(P,Q,TS+1) :-holds(P,Q1,TS),tot_incr(P,Q2,TS),tot_decr(P,Q3,TS), Q = Q1+Q2-Q3, place(P), num(Q;Q1;Q2;Q3), time(TS), time(TS+1).`
a2: `consumesmore(P,TS) :- holds(P,Q,TS), tot_decr(P,Q1,TS), Q1 > Q.`
a3: `consumesmore :- consumesmore(P,TS).`
a4: `:- consumesmore.`

The rule $e1$ encodes `notenabled(T,TS)` which captures the existence of an input place P of transition T that violates the minimum token requirement N at time-step TS. Where, the predicate `holds(P,Q,TS)` encodes the marking Q of place P at TS. Rule $e2$ encodes `enabled(T,TS)` which captures that transition T is enabled at TS since there is no input place P of transition T that violates the minimum input token requirement at TS. In biological context, $e2$ captures the conditions when a reaction (represented by T) is ready to proceed.

Rule $a1$ encodes `fires(T,TS)`, which captures the firing of transition T at TS. This rule has a choice atom as its head, which either picks the enabled transition T for firing at TS or not, effectively selecting a firing set as a subset of enabled transitions. Conflict checking is done by rules $a2, a3, a4$, which eliminate answer-sets which contain firing sets in conflict. In biological context, the selected transition-set models simultaneously occurring reactions and the conflict models limited reactant supply that cannot be shared. Such a conflict can lead to multiple choices in parallel reaction evolutions and different outcomes.

Rule pair $r1, r2$ encode `add(P,Q,T,TS),del(P,Q,T,TS)` and captures the addition or deletion of Q tokens to place P due to firing of transition T at time-step TS, respectively. Rule pair $r3, r4$ aggregates these `add`'s and `del`'s for each place P at time-step TS, respectively [4]. Rule $r5$ which encodes `holds(P,Q,TS+1)` uses the aggregated adds and deletes to update P's marking for the next time-step $TS+1$. In biological context, these rules capture the effect of a reaction on reactant and product quantities available in the next simulation step.

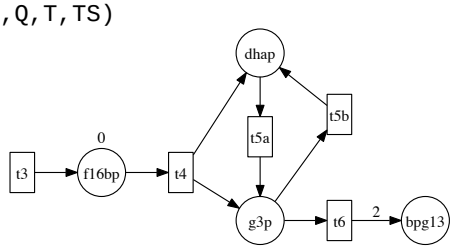

Fig. 1: Petri Net graph (of sub-section of glycolysis pathway) showing places as circles, transitions as boxes and arcs as directed arrows. Places have token count (or marking) written above them, assumed 0 when missing. Arcs labels represent arc-weights, assumed 1 when missing.

Rule $a2$ encodes `consumesmore(P,TS)` which captures overconsumption of tokens at input place P at time TS due to the firing set selected by $a1$. Overconsumption (and hence conflict) occurs when tokens $Q1$ consumed by the firing set are greater than the tokens Q available at P. Rule $a3$ generalizes this notion of overconsumption and constraint $a4$ eliminates answers where overconsumption occurs.

Definition 1. *Given a Petri Net PN and its encoding $\Pi(PN, M_0, k)$. We say that there is a 1-1 correspondence between the answer sets of $\Pi(PN, M_0, k)$ and the execution sequences of PN iff for each answer set A of $\Pi(PN, M_0, k)$, there is a corresponding execution sequence $X = M_0, T_0, \ldots, M_k, T_k$ of PN such that $\{fires(t, j) : t \in T_j, 0 \leq j \leq k\} = \{fires(t, ts) : fires(t, ts) \in A\}$ and $\{holds(p, q, j) : p \in P, q = M_j(p), 0 \leq j \leq k\} = \{holds(p, q, ts) : holds(p, q, ts) \in A\}$*

[4] $r3, r4$ use the `QQ=#sum[]` construct to sum the Q values into QQ.

Proposition 1. *There is a 1-1 correspondence between the answer sets of* $\Pi^0(PN, M_0, k)$ *and the execution sequences of* PN.

Next we look at an example. The Petri Net in Fig. 1 is encoded as follows:

```
num(0..60).time(0..2).place(f16bp;dhap;g3p;bpg13).trans(t3;t4;t5a;t5b;t6).
tparc(t3,f16bp,1).ptarc(f16bp,t4,1).tparc(t4,dhap,1).tparc(t4,g3p,1).
ptarc(dhap,t5a,1).tparc(t5a,g3p,1).ptarc(g3p,t5b,1).tparc(t5b,dhap,1).
ptarc(g3p,t6,1).tparc(t6,bpg13,2).holds(f16bp;dhap;g3p;bgp13,0,0).
```

its simulation results in tens of answer-sets, for example:

```
holds(bpg13;dhap;f16bp;g3p,0,0) fires(t3,0) holds(bpg13;dhap;g3p,0,1)
holds(f16bp,1,1) fires(t3;t4,1) holds(bpg13,0,2) holds(dhap;f16bp,1,2)
holds(g3p,1,2) fires(t3;t4;t5a;t5b,2)
```

4 Changing Firing Semantics

The ASP code above implements the *set firing* semantics. It can produce a large number of answer-sets, since any subset of a firing set will also be fired as a firing set. For our biological system modeling, it is often beneficial to simulate only the maximum activity at each time-step. We accomplish this by defining the *maximal firing set* semantics, such that a maximal subset of non-conflicting transitions fires at a time step [5]. We enforce it with the following additional rules:

a5:
```
could_not_have(T,TS) :- enabled(T,TS), not fires(T,TS), ptarc(S,T,Q),
  holds(S,QQ,TS), tot_decr(S,QQQ,TS), Q > QQ - QQQ.
```
a6:
```
:- not could_not_have(T,TS), enabled(T,TS), not fires(T,TS), trans(T),
  time(TS).
```

Rule *a*5 encodes `could_not_have(T,TS)` which means that an enabled transition T that did not fire at time TS, could not have fired as its firing would have resulted in overconsumption. Rule *a*6 eliminates any answer-sets in which an enabled transition did not fire, that would not have caused overconsumption, ensuring maximality of firing set. With this extension, the answer sets produced for Petri Net in Fig. 1 reduces to 2.

Proposition 2. *There is 1-1 correspondence between the answer sets of* $\Pi^1(PN, M_0, k)$ *and the execution sequences of* PN.

Other firing semantics can be encoded with similar ease[6]. We now look at Petri Net extensions and show how they can be easily encoded by making small changes to our ASP encoding.

5 Extension - Reset Arcs

A Reset Arc in a Petri Net PN^R is an arc from place p to transition t that consumes all tokens from its input place p on firing of t. Fig. 2 shows an extended version of the Petri Net in Fig. 1 with a reset arc from *dhap* to *tr* (shown with double arrowhead). In biological context it models the removal of all quantity of compound *dhap*.

[5] Such semantics reduces the reachable markings and its computational power has been analyzed by others. Our semantics is different from the firing multiplier (max. parallelism) approach, which requires an exponential time firing algorithm in the number of transitions. See (Anwar et al. 2013) for more.

[6] For example, if *interleaved* firing semantics is desired, replace rules *a*5, *a*6 with the following:

a5':
```
more_than_one_fires:-fires(T1;T2,TS),T1!=T2,trans(T1;T2),time(TS).
```
a6':
```
:- more_than_one_fires.
```

Definition 2 (Reset Arc)**.** *A Reset Petri Net is a tuple* $PN^R = (P, T, E, W, R)$ *where,* P, T, E, W *are the same as for PN; and* $R : T \rightarrow 2^P$ *defines reset arcs*

Petri Net execution semantics with reset arcs is modified for conflict detection and execution as follows: A set of enabled transitions **conflict in** PN^R w.r.t. M_k if firing them simultaneously will consume more tokens than are available at any one of their common input-places. $T_e = \{t \in T : enabled_{M_k}(t)\}$ conflict if $\exists p \in P : M_k(p) < (\sum_{t \in T_e \wedge (p,t) \in E^-} W(p,t) + \sum_{t \in T_e \wedge p \in R(t)} M_k(p))$. **Execution of a transition set** T_i in PN^R has the following effect: $\forall p \in P \setminus R(T_i), M_{k+1}(p) = M_k(p) - \sum_{t \in T_i \wedge p \in {}^\bullet t} W(p,t) + \sum_{t \in T_i \wedge p \in t^\bullet} W(t,p)$ and $\forall p \in R(T_i), M_{k+1}(p) = \sum_{t \in T_i \wedge p \in t^\bullet} W(t,p)$ where $R(T_i) = \bigcup_{t \in T_i} R(t)$ represents the places emptied by T_i due to reset arcs. Since a reset arc from p to t, $p \in R(t)$ consumes current marking dependent tokens, we extend `ptarc` to include time and replace $f3, f4, e1, r1, r2$ with $f6, f7, e3, r6, r7$, respectively in the Section 4 encoding and add rule $f8$ for each reset arc:

f6: Rules `ptarc(`$p_i, t_j, W(p_i,t_j), ts_k$`):-time(`$ts_k$`).` for each non-reset arc $(p_i, t_j) \in E^-$
f7: Rules `tparc(`$t_i, p_j, W(t_i,p_j), ts_k$`):-time(`$ts_k$`).` for each non-reset arc $(t_i, p_j) \in E^+$

```
e3: notenabled(T,TS) :- ptarc(P,T,N,TS), holds(P,Q,TS), Q < N, place(P),
  trans(T), time(TS), num(N), num(Q).
r6: add(P,Q,T,TS) :- fires(T,TS), tparc(T,P,Q,TS), time(TS).
r7: del(P,Q,T,TS) :- fires(T,TS), ptarc(P,T,Q,TS), time(TS).
```

f8: Rules `ptarc(`p_i, t_j, X, ts_k`) :- holds(`p_i, X, ts_k`), num(`X`), `$X > 0$`.` for each reset arc between p_i and t_j using $X = M_k(p_i)$ as arc-weight at time step ts_k.

Rule $f8$ encodes place-transition arc with marking dependent weight to capture the notion of a reset arc [7]. We chose our definition to allow modeling of biological process that removes all available quantity of a substance in a maximal firing set. Consider Fig. 2, if *dhap* has 1 or more tokens, our semantics would only permit either *t5a* or *tr* to fire in a single time-step, while the standard semantics can allow both *t5a* and *tr* to fire simultaneously, such that the reset arc removes left over tokens after (*dhap*, *t5a*) consumes one token. We could have, instead, encoded our reset arc semantics using self-modifying nets, but our current encoding provides a simpler solution. Standard semantics, however, can be easily added [8].

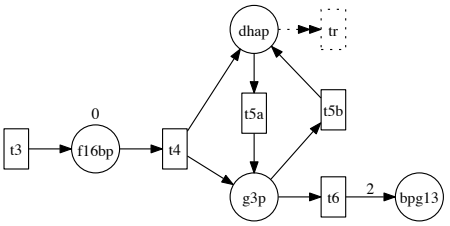

Fig. 2: Petri Net of Fig 1 extended with a reset arc from *dhap* to *tr* shown with double arrowhead.

Proposition 3. *There is 1-1 correspondence between the answer sets of* $\Pi^2(PN^R, M_0, k)$ *and the execution sequences of* PN^R.

6 Extension - Inhibitor Arcs

[7] We use a modified reset arc execution semantics. Contrary to the standard reset arcs, our reset arc consumes tokens from its source place in contention with other place-transition arcs from the same place, and not as a side effect.
[8] Standard reset semantics can be added to Section 4 encoding by splitting $r5$ into $r5a', r5b'$ and adding $f8', a7'$:

f8': `rptarc(`p_i, t_j`).` - for each reset arc between p_i and t_j

```
a7': reset(P,TS) :- rptarc(P,T), place(P), trans(T), fires(T,TS), time(TS).
r5a': holds(P,Q,TS+1):-holds(P,Q1,TS),tot_incr(P,Q2,TS),tot_decr(P,Q3,TS),Q=Q1+Q2-Q3,
  place(P), num(Q;Q1;Q2;Q3), time(TS;TS+1), not reset(P,TS).
r5b': holds(P,Q,TS+1):-tot_incr(P,Q,TS),place(P),num(Q),time(TS;TS+1),reset(P,TS).
```

An inhibitor arc is a place–transition arc that inhibits its transition from firing as long as the place has any tokens in it. An inhibitor arc does not consume any tokens from its input place. Fig. 3 shows a Petri Net with inhibition arc from atp to $gly1$ with a bulleted arrowhead. It models biological feedback regulation in simplistic terms, where excess atp downstream causes the upstream atp production by glycolysis gly to be inhibited until the excess quantity is consumed (Reece et al. 2010).

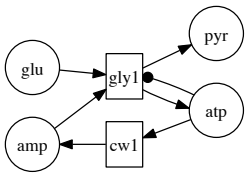

Fig. 3: Petri Net showing feedback inhibition arc from atp to $gly1$ with a bullet arrowhead. Inhibitor arc weight is assumed 1 when not specified.

Definition 3 (Inhibitor Arc)**.** *A Petri Net with reset and inhibitor arcs is a tuple* $PN^{RI} = (P, T, E, W, R, I)$*, where,* P, T, E, W, R *are the same as for* PN^{R}*; and* $I : T \rightarrow 2^{P}$ *defines inhibitor arcs.*

Petri Net execution semantics with inhibit arcs is modified for determining enabled transitions as follows: A transition t is **enabled in** PN^{RI} with respect to marking M, $enabled_M(t)$, if all its input places p have at least the number of tokens as the arc-weight $W(p,t)$ and all $p \in I(t)$ have zero tokens, i.e. $(\forall p \in \bullet t : W(p,t) \leq M(p)) \wedge (\forall p \in I(t) : M(p) = 0)$. We add inhibitor arcs to our encoding in Section 5 as follows:

f9: Rules `iptarc(`$p_i, t_j, 1, ts_k$`):-time(`ts_k`).` for each inhibitor arc between $p_i \in I(t_j)$ and t_j.

```
e4: notenabled(T,TS) :- iptarc(P,T,N,TS), holds(P,Q,TS), place(P),
  trans(T), time(TS), num(N), num(Q), Q >= N.
```

The new rule $e4$ makes a transition T not-enabled when a source place P of an inhibitor arc from P to T has more than the arc-weight tokens N in it, where N is always 1 per rule $f9$.

Proposition 4. *There is 1-1 correspondence between the answer sets of* $\Pi^{3}(PN^{RI}, M_0, k)$ *and the execution sequences of* PN.

7 Extension - Read Arcs

A read arc (a test arc or a query arc) is an arc from place to transition, which enables its transition only when its source place has at least the number of tokens as its arc weight. It does not consume any tokens from its input place. Fig. 4 shows a Petri Net with read arc from h_is to syn shown with arrowhead on both ends. It models the ATP synthase syn activation requiring a higher concentration of $H+$ ions h_is in the intermembrane space [9]. The reaction itself consumes a lower quantity of $H+$ ions represented by the regular place-transition arc (Reece et al. 2010).

Definition 4 (Read Arc)**.** *A Petri Net with reset, inhibitor and read arcs is a tuple* $PN^{RIQ} = (P, T, W, R, I, Q, QW)$*, where,* P, T, E, W, R, I *are the same as for* PN^{RI}*;* $Q \subseteq P \times T$ *defines read arcs; and* $QW : Q \rightarrow \mathbb{N} \setminus \{0\}$ *defines read arc weight.*

Petri Net execution semantics with read arcs is modified for determining enabled transitions as follows: A transition t is **enabled in** PN^{RIQ} with respect to marking M, $enabled_M(t)$, if all its input places p have at least the number of tokens as the arc-weight $W(p,t)$, all $p_i \in I(t)$ have zero tokens and all $p_q : (p_q, t) \in Q$ have at least the number of tokens as the arc-weight $W(p,t)$, i.e. $(\forall p \in \bullet t : W(p,t) \leq M(p)) \wedge (\forall p \in I(t) : M(p) = 0) \wedge (\forall (p,t) \in Q : M(p) \geq QW(p,t))$. We add read arcs to our encoding of Section 6 as follows:

[9] Our model of ATP synthase syn activation is oversimplified, since the actual model requires an $H+$ concentration differential across membrane.

f10: Rules `tptarc(`$p_i, t_j, QW(p_i,t_j), ts_k$`):-time(`$ts_k$`).` for each read arc $(p_i, t_j) \in Q$.
e5: `notenabled(T,TS):-tptarc(P,T,N,TS),holds(P,Q,TS), place(P),trans(T), time(TS), num(N), num(Q), Q < N.`

The new rule $e5$ make a transition T not-enabled when a source place P of a read arc to T has less than the arc weight N tokens.

Proposition 5. *There is a 1-1 correspondence between the answer sets of* $\Pi^4(PN^{RIQ}, M_0, k)$ *and the execution sequences of* PN^{RIQ}.

8 Example Use Case of Our Encoding and Additional Reasoning Abilities

We illustrate the usefulness of our encoding by applying it to the following simulation based reasoning question from (Reece et al. 2010)[10]:

Question 1. *"At one point in the process of glycolysis, both dihydroxyacetone phosphate (DHAP) and glyceraldehyde 3-phosphate (G3P) are produced. Isomerase catalyzes the reversible conversion between these two isomers. The conversion of DHAP to G3P never reaches equilibrium and G3P is used in the next step of glycolysis. What would happen to the rate of glycolysis if DHAP were removed from the process of glycolysis as quickly as it was produced?"*

In order to answer this question, we create a Petri Net model of sub-portion of the normal glycolysis pathway relevant to the question (Fig. 1) and encode it in ASP using the encoding in Section 3. We then extend this pathway by adding a reset arc to it for modeling the immediate removal of *dhap* (Fig. 2) and encode it in ASP using the encoding in Section 5. We simulate both models for the same number of time steps using the maximal firing set semantics from Section 4, since we are interested in the maximum change (in $bpg13$ production) between the two scenarios. Fig. 5 shows the average quantity of $bpg13$ produced by the normal and the extended Petri Net models for a 15-step simulation and the spread of unique $bpg13$ quantities produced during these simulations. We compute the rate of glycolysis as the ratio "$bpg13/ts$" at the end of the simulation. We post process these results to determine the average rates as well as the spread among the answer-sets. Our results show a lower rate of $bpg13$ production and hence glycolysis in the extended pathway, with a spread of $bpg13$ quantity ranging from zero to the same amount as the normal pathway. Although we are using average rates to answer this question, the ASP encoding produces all possible state evolutions, which may be further analyzed by assigning different probabilities to these state evolutions.

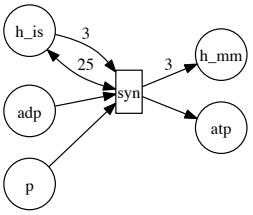

Fig. 4: Petri Net with read arc from *h_is* to *syn* shown with arrowhead on both ends. The transition *syn* will not fire unless there are at least 25 tokens in *h_is*, but when it executes, it only consumes 3 tokens.

ASP allows us to perform additional reasoning not directly supported by various Petri Net formalisms examined by us [11]. For example, if we are given partial state information, we can use it in ASP as way-points to guide the simulation. Consider that we want to determine the

[10] As it appeared in `https://sites.google.com/site/2nddeepkrchallenge/`
[11] We examined a significant number of Petri Net formalisms

cause of a substance S's quantity recovery after it is depleted. Using ASP, we can enumerate only those answer-sets where a substance S exhausts completely and then recovers by adding constraints. The answer sets produced can then be passed on to an additional ASP reasoning step that determines the general cause leading to the recovery of S after its depletion. This type of analysis is possible, since the answer-sets enumerate the entire simulation evolution and all possible state evolutions.

Although not a focus of this work, various Petri Net properties can be easily analyzed using our encoding. For example dynamic properties can be analyzed as follows: one can test reachability of a state (or marking) S by adding a constraint that removes all answer-sets that do not contain S; boundedness by adding a constraint on token count; basic liveness by extending the Petri Net model with source transitions connected to all places, switching to the interleaved firing semantics, and removing answers where the subject transition to be tested is not fired. Intuitively, structural properties can be analyzed as follows: T-invariants can be extracted by enumerating transitions $(t_i, \ldots, t_{k-1})$ whenever marking $M_k = M_i, k > i$ using an interleaved firing semantics. P-invariants can be extracted by using interleaved firing semantics to visit all possible state (or marking) evolutions, selecting a subset of places $P_i \subseteq P$, and eliminating all answer-sets where $\sum_{p \in P_i} M_{k+1}(p) \neq \sum_{p \in P_i} M_k(p), k \geq 0$.

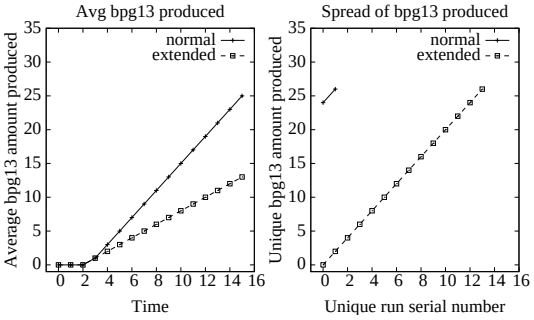

Fig. 5: Plots showing the average quantity of $bpg13$ produced by the normal and extended Petri Net models in Fig. 1 and Fig. 2 for a simulation length of 15 and the spread of these quantities across answer sets.

9 Related Work and Conclusion

Here we will elaborate a bit on some of the existing Petri Net systems[12], (especially ones used in biological modeling) and put them in context of our research. We follow that with some ways existing Petri Net tools are used for biological analysis and put them in the context of our research.

CPN Tools is a graphical tool for simulating Colored Petri Nets with discrete tokens. It supports guards, timed transitions, and hierarchical transitions but currently does not have direct support for inhibit or reset arcs. It pursues one simulation evolution, breaking transition choice ties randomly. Cell Illustrator is a closed source Java based graphical tool for simulating biological systems using Hybrid Functional Petri Nets (HPFNe). HPFNs combine features from Continuous as well as Discrete Petri Nets. Cell Illustrator only supports uncolored tokens and pursues one simulation evolution, breaking transition choice ties randomly. Snoopy is a graphical tool written in C++ for analyzing biological models. It supports simulating Colored, Stochastic, Hybrid, and Continuous Petri Nets with read, reset, and inhibit arcs under a few firing semantics, including maximal firing. However, it does not explore all possible evolutions and it is unclear if simulation results can be exported for further reasoning. Renew is an open source simulation tool written in Java. It supports Colored tokens, (reference semantics of) Object tokens, guard conditions, inhibit arcs, reset arcs, timed tokens, arc delays, and token reservation. Token nets can

[12] 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`

be created on the fly. Its use requires some knowledge of Java. Some features like arc pre-delays when combined with transition delays can lead to complicated semantics. It is unclear how ties are broken to resolve transition conflicts and whether all possible state evolutions are explored.

(Sackmann et al. 2006) and others have previously used Petri Nets to analyze biological pathways, but their analysis is mostly limited to dynamic and structural properties. (Peleg et al. 2005) surveyed various Petri Net implementations to study the properties and dynamics of biological systems. They defined a series of question that can be answered by Petri Nets using simulation. Contrary to their approach, we picked our questions from college level text books to capture real world questions. Our approach is also different in that we model the questions as Petri Net extensions and we can perform further reasoning on simulation runs.

Conclusion: In this paper we discussed the appropriateness of Petri Nets–especially ASP implementation of Petri Nets, for modeling biological pathways and answering realistic questions (the ones found in biology textbooks) with respect to such pathways. We presented how simple Petri Nets can be encoded in ASP and showed that ASP provides an elaboration tolerant way to easily realize various Petri Net extensions and firing semantics. The extensions include changing of the firing semantics, and allowing reset arcs, inhibitor arcs and read arcs. Our encoding has a low specification-implementation gap. It allows enumeration of all possible evolutions of a Petri Net simulation as well as the ability to carry out additional reasoning about these simulations. We also presented an example use case of our encoding scheme. Finally, we briefly discussed other Petri Net systems and their use in biological modeling and analysis and compared them with our work. Our focus in this paper has been less on performance and more on ease of encoding, extensibility, exploring all possible state evolutions, and strong reasoning abilities. In a follow on enhanced version of this paper, we will carry out detailed performance analysis. In a sequel of this paper, we will present extension of this work to other Petri Net extensions, such as colored tokens, priority transitions, and durative transitions.

References

ANWAR, S., BARAL, C., AND INOUE, K. 2013. Encoding petri nets in answer set programming for simulation based reasoning. `http://arxiv.org/abs/1306.3542`.

BEHRENS, T. M. AND DIX, J. 2007. 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.

GEBSER, M., KAMINSKI, R., KAUFMANN, B., OSTROWSKI, M., SCHAUB, T., AND SCHNEIDER, M. 2011. Potassco: The Potsdam answer set solving collection. *aicom 24*, 2, 105–124.

HELJANKO, K. AND NIEMELÄ, I. 2000. Petri net analysis and nonmonotomic reasoning. *Leksa Notes in Computer Science*, 7–19.

PELEG, M., RUBIN, D., AND ALTMAN, R. B. 2005. Using petri net tools to study properties and dynamics of biological systems. *Journal of the American Medical Informatics Association 12*, 2, 181–199.

PETRI, C. A. 1962. Kommunikation mit automaten. Tech. rep., Technical Report 2 (Schriften des IIM), Institut für Instrumentelle Mathematik, Bonn, Germany.

REECE, J., CAIN, M., URRY, L., MINORSKY, P., AND WASSERMAN, S. 2010. *Campbell Biology*. Pearson Benjamin Cummings.

SACKMANN, A., HEINER, M., AND KOCH, I. 2006. Application of petri net based analysis techniques to signal transduction pathways. *BMC Bioinformatics 2*, 7 (November), 482.