Encoding Reversing Petri Nets in Answer Set Programming

Yannis Dimopoulos, Eleftheria Kouppari, Anna Philippou^(✉),
and Kyriaki Psara

Department of Computer Science, University of Cyprus, Nicosia, Cyprus
{yannis, ekoupp02, annap, kpsara01}@cs.ucy.ac.cy

Abstract. Reversing Petri nets (RPNs) have been proposed as a reversible approach to Petri nets, which allows the transitions of a net to be reversed. This work presents an approach towards an implementation of RPNs to support their simulation and analysis. Specifically, we define how to model RPNs in Answer Set Programming (ASP), a declarative programming framework with competitive solvers. We highlight how the methodology can be used to reason about the behavior of RPN models.

1 Introduction

Reversibility is a phenomenon referring to the ability of a system to execute its actions in both the forward and the reverse directions. It occurs in a variety of systems (e.g., quantum computation and biochemical systems) and can be exploited in many others (e.g., robotics, manufacturing systems, distributed systems, and logical circuits). Recently, its study has been receiving increased attention. Among the developments, a variety of formal frameworks of modeling reversible systems have been defined, contributing towards an improved understanding of the basic principles of reversibility. A natural next step for this work is the development of techniques for the automatic analysis of reversible models.

In this paper we consider reversing Petri nets (RPNs) [8], a recently-proposed Petri-net framework that allows transitions to be carried out in both the forward and the reverse directions in or out of causal order. Specifically, we present work in progress towards a framework for simulation and analysis of RPN models via an encoding into Answer Set Programming (ASP) [5, 7] a declarative programming framework with competitive solvers that may be used to model a system as well as a query about the system via a logic program, such that models of the program provide the answers to the set query. ASP has proved to be a promising approach towards reasoning about Petri nets with encodings to ASP having been defined for a number of Petri net subclasses, e.g. [1–3, 6]. In this paper we provide a systematic way of modelling RPNs and their causal reversibility semantics in ASP. Our long term goal is the development of an ASP-based framework for reasoning about RPN models.

2 Reversing Petri Nets

In this section we briefly recall RPNs and we refer the reader to [8, 9] for the full exposition. Following [9], RPNs are cyclic structures defined as follows:

Definition 1. A *reversing Petri net* (RPN) is a tuple (A, P, B, T, F) where:

1. A is a finite set of *bases* or *tokens* ranged over by $a, b, \dots$. We write $\bar{A} = \{\bar{a} \mid a \in A\}$ and $\mathcal{A} = A \cup \bar{A}$.
2. P is a finite set of *places* and T a finite set of *transitions*.
3. $B \subseteq A \times A$ is a set of undirected *bonds* ranged over by $\beta, \gamma, \dots$. We write $a-b$ for a bond $(a, b) \in B$, $\bar{B} = \{\bar{\beta} \mid \beta \in B\}$, and $\mathcal{B} = B \cup \bar{B}$.
4. $F : (P \times T \cup T \times P) \rightarrow 2^{A \cup B}$ defines a set of directed *arcs*.

RPNs are built on the basis of a set of *bases* or *tokens* each having a unique name. Tokens may occur as stand-alone elements or merge together to form *bonds*. *Places* and *transitions* have the standard meaning. Directed arcs connect places to transitions and vice versa and are labelled by a subset of $\mathcal{A} \cup \mathcal{B}$ where $\bar{A}$ and $\bar{B}$ are the sets of “negative” tokens and bonds, expressing token and bond absence, respectively. For a label $\ell = F(x, t)$ or $\ell = F(t, x)$, we assume that each token a can appear in ℓ at most once, either as a or as $\bar{a}$, and that if $(a, b) \in \ell$ then $a, b \in \ell$. Furthermore, for $\ell = F(t, x)$ it must be that $\ell \cap (\bar{A} \cup \bar{B}) = \emptyset$. $F(x, y) = \emptyset$ implies that there is no arc between x and y . Finally, we assume that F is defined so that transitions (1) do not erase tokens, (2) do not destroy bonds, and (3) do not clone tokens/bonds into more than one outgoing place.

We write $ot = \{x \in P \mid F(x, t) \neq \emptyset\}$ and $to = \{x \in P \mid F(t, x) \neq \emptyset\}$, $\text{pre}(t) = \bigcup_{x \in P} F(x, t)$ and $\text{post}(t) = \bigcup_{x \in P} F(t, x)$, and define the *effect* of a transition as $\text{eff}(t) = \text{post}(t) - \text{pre}(t)$. Furthermore, we employ the notion of a *marking* defined as a distribution of tokens and bonds across places, $M : P \rightarrow 2^{A \cup B}$. In addition, a *history* assigns a memory to each transition, $H : T \rightarrow 2^{\mathbb{N}}$, where $H(t) = \emptyset$ captures that transition t has not taken place, or it has been reversed, and $H(t) = \{k_1, \dots, k_n\}$ captures that t was executed and not reversed n times where the k_i indicate the order of the execution instances. A pair $\langle M, H \rangle$ describes a *state* of a RPN. Finally, we write $\text{con}(a, C)$, where $a \in C$ and $C \subseteq A \cup B$, for the connected component of token a in the graph described by C .

We proceed to define forward execution for RPNs.

Definition 2. Consider a RPN (A, P, B, T, F) , a transition $t \in T$, and a state $\langle M, H \rangle$. We say that t is *forward enabled* in $\langle M, H \rangle$ if:

1. if $a \in F(x, t)$ (resp. $\beta \in F(x, t)$), for some $x \in ot$, then $a \in M(x)$ (resp. $\beta \in M(x)$), and if $\bar{a} \in F(x, t)$ (resp. $\bar{\beta} \in F(x, t)$) for some $x \in ot$, then $a \notin M(x)$ (resp. $\beta \notin M(x)$),
2. if $a \in F(t, y_1)$, $b \in F(t, y_2)$, $y_1 \neq y_2$, then $b \notin \text{con}(a, M(x))$ for all $x \in ot$,
3. if $\beta \in F(t, x)$ for some $x \in to$ and $\beta \in M(y)$ for some $y \in ot$, then $\beta \in F(y, t)$.

Definition 3. Given a RPN (A, P, B, T, F) , a state $\langle M, H \rangle$, and a transition t forward enabled in $\langle M, H \rangle$, we write $\langle M, H \rangle \xrightarrow{t} \langle M', H' \rangle$ where:

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

$$H'(t') = \begin{cases} H(t') \cup \{\max(\{0\} \cup \{k \mid k \in H(t''), t'' \in T\}) + 1\}, & \text{if } t' = t \\ H(t'), & \text{otherwise} \end{cases}$$

Thus, when a transition is executed all tokens and bonds on its incoming arcs are relocated from its input to its output places with their connected components.

According to causal-order reversibility, a transition may be reversed only if all transitions causally dependent on it have either been reversed or not executed:

Definition 4. Consider a RPN (A, P, B, T, F) , a state $\langle M, H \rangle$, and a transition $t \in T$. Then t is *co-enabled* in $\langle M, H \rangle$ if $H(t) \neq \emptyset$ and, for all $a \in F(t, x)$, if $a \in M(y)$ for some y and $\text{con}(a, M(y)) \cap \text{pre}(t') \neq \emptyset$ for some t' then either $H(t') = \emptyset$ or there is $k \in H(t)$ such that $k > k'$ for all $k' \in H(t')$.

When a transition is reversed in a causal fashion, all tokens and bonds in the postcondition of the transition and their connected components are transferred to the incoming places of the transition and any created bonds are broken.

Definition 5. Given a RPN (A, P, B, T, F) , a state $\langle M, H \rangle$, and a transition t co-enabled in $\langle M, H \rangle$, we write $\langle M, H \rangle \xrightarrow{t}_c \langle M', H' \rangle$ where

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

$$H'(t') = \begin{cases} H(t') - \{k\}, & \text{if } t' = t, k = \max(H(t)) \\ \{k' \mid k' \in H(t'), k' < k\} \cup \{k' - 1 \mid k' \in H(t'), k' > k\}, & \text{otherwise} \end{cases}$$

An example of a reversing Petri net can be seen in Fig. 1 simulating the assembly of a three-component product. A principal process during the re-manufacturing of worn-out or malfunctioning products is disassembly that enables the dumping, cleaning, repair or replacement of components as desired. Therefore, reversible computation can be used as means of modelling the disassembly process while considering the product topology, mating relations, and precedence relations. Here we have three tokens a, b , and c representing the three components of an assembly line. Token d represents the machine required in order to assemble the components into the final product. The system consists of two independent transitions, t_2 and t_3 causally following transition t_1 , and transition t_4 causally following t_2 . This structure gives the ability to the machine to directly bond component a with c by executing t_3 or to indirectly bond a with c through component b . The ability to reverse every transition in

Fig. 1. Causal-order example

the context of manufacturing task planning, can be used in order to recover from failures during the assembly process. In the figure, we observe the execution of transition sequence $t_1; t_3; \bar{t}_3; t_2; t_4$, demonstrating a causally-ordered reversal of transition t_3 (denoted by $\bar{t}_3$) in order to accommodate the manufacturing of a product consisting of $a-b-c$. Note that a non-empty history of transitions is presented as a list over the transitions.

3 Translating RPNs into ASP

Answer Set Programming (ASP) [5, 7] is an extension of logic programming with negation as failure under the *stable model* or *answer set* semantics. An Answer

Set Program is a set of rules of the form

$$A_0 \leftarrow A_1, A_2, \dots, A_m, \text{not } A_{m+1}, \text{not } A_{m+2}, \dots, \text{not } A_n$$

with the intuitive reading that if all atoms $A_1, A_2, \dots, A_m$ are true and none of $A_{m+1}, A_{m+2}, \dots, A_n$ is true, then A_0 must be true. Most ASP systems, such as **clingo** (<https://potassco.org/clingo/>) that we use for our encoding, extend the basic language with additional constructs such as *choice rules* of the form $\{A_0^1, \dots, A_0^k\} \leftarrow A_1, \dots, A_m, \text{not } A_{m+1}, \dots, \text{not } A_n$, with the meaning that if the right hand side of the rule holds, then some subset of the atoms $\{A_0^1, \dots, A_0^k\}$ must hold true, which introduces a form of non-determinism that is useful when modeling combinatorial problems. ASP systems, such as **clingo**, couple a highly expressive language that provides constructs from various fields, such as database systems and constraint programming, with powerful solvers. Therefore, these systems can be successfully applied to a large number of application domains and they have shown the potential of solving problems with thousands of variables and hundreds of thousands of rules in seconds or minutes.

In the following we discuss a translation of reversing Petri nets under the causal reversibility semantics into ASP for a simulation length that is encoded by the last argument **TS** of the predicates of our model. The basic predicates that represent the input network are **trans**(**T**), **token**(**Q**), and **place**(**P**), that correspond to the transitions, tokens and places, respectively, whereas predicates **ptarc**(**P**,**T**,**Q**) declare that $Q \in F(P, T)$ (similarly for **tparc**(**T**,**P**,**Q**)) and **ptarcbond**(**P**,**T**,**Q1**,**Q2**) refer to bonds, i.e. $Q1-Q2 \in F(P, T)$. The markings of a RPN are captured by predicates **holds** for tokens and **holdsbond**s for bonds. That is, **holdsbond**s(**P**,**Q1**,**Q2**,**TS**) means $Q1-Q2 \in M_{TS}(P)$, where M_{TS} refers to the marking at step **TS** of the simulation.

In the following we discuss some excerpts of the RPN to ASP translation that has been implemented in the **clingo** system.

Forward-enableness of Definition 2 is captured via **notenabled**(**T**,**TS**), which assumes the value true if transition **T** cannot be enabled at step **TS**. Some of the rules for this predicate are depicted in Listing 3.1. The first rule refers to the case $Q \in F(P, T)$ and $Q \notin M_{TS}(P)$, and the second to the similar case for bonds. The rule in lines 5–6 corresponds to the second item of Definition 2, i.e. $Q1 \in F(P1, T)$, $Q2 \in F(P2, T)$ and $Q2 \in \text{con}(Q1, M_{TS}(P))$, where predicate **connected** defines the set $\text{con}(a, C)$. The third case of Definition 2 corresponds to the rule in lines 7–9.

If none of the conditions that disable a transition holds, the transition is enabled, as encoded by the rule in line 11. The *choice rule* of line 12, allows **clingo** to assign any of the values true or false to atom **fires**(**T**,**TS**), i.e. choose whether transition **T** executes or not.

Lines 15–22 of Listing 3.1 are the rules for bond addition and deletion that result from forward transition execution. The rule in line 15 says that $Q1-Q2 \in M(TP_{TS+1})$ for $Q1-Q2 \in F(T, TP)$ if transition **T** fires at time **TS**. The rule in lines 16–18 enforces that $Q1-Q2 \in M(TP_{TS+1})$ for $Q1-Q2 \in M(PT_{TS})$, $Q1-Q2 \in \text{con}(Q, M(PT_{TS}))$, $Q \in F(PT, T) \cap F(T, TP)$, and transition **T** that fires at time **TS**. Lines 20–22 implement bond deletion in a similar way. The encoding also contains rules for token addition and deletion.

Listing 3.1. Forward enabledness and execution

```

1 notenabled(T,TS):-ptarc(P,T,Q),not holds(P,Q,TS).
2 notenabled(T,TS):-ptarcbond(P,T,Q1,Q2),
3     not holdsbonds(P,Q1,Q2,TS).
4
5 notenabled(T,TS):-tparc(T,P1,Q1),tparc(T,P2,Q2),P1!=P2,
6     connected(P,Q1,Q2,TS),ptarc(P,T,_).
7 notenabled(T,TS):-tparcbond(T,TP,Q1,Q2),ptarc(P,T,_),
8     holdsbonds(P,T,Q1,Q2,TS),
9     not ptarcbond(P,T,Q1,Q2).
10
11 enabled(T,TS):-not notenabled(T,TS).
12 {fires(T,TS)}:-enabled(T,TS).
13
14
15 addBond(TP,Q1,Q2,TS+1):-fires(T,TS),tparcbond(T,TP,Q1,Q2).
16 addBond(TP,Q1,Q2,TS+1):-fires(T,TS),tparc(T,TP,Q),
17     ptarc(P,T,Q),connected(P,T,Q,Q1,TS),
18     holdsbonds(P,T,Q1,Q2,TS).
19
20 delBond(P,T,Q1,Q2,TS+1):-fires(T,TS),ptarcbond(P,T,Q1,Q2).
21 delBond(P,T,Q1,Q2,TS+1):-fires(T,TS),ptarc(P,T,Q),
22     connected(P,T,Q,Q1,TS),holdsbonds(P,T,Q1,Q2,TS).

```

Listing 3.2. Co-enabledness

```

1 dependent(T2,T1,TS):-tparc(T1,_,Q),ptarc(_,T2,Q),
2     H2=#max{H:transHistory(T2,H,TS),history(H)},
3     H1=#max{H:transHistory(T1,H,TS),history(H)},
4     H2>H1,H1>0.
5 dependent(T2,T1,TS):-tparc(T1,_,Q),ptarc(_,T2,Q1),
6     connected(_,Q,Q1,TS),
7     H2=#max{H:transHistory(T2,H,TS),history(H)},
8     H1=#max{H:transHistory(T1,H,TS),history(H)},
9     H2>H1,H1>0.
10
11 notenabledC(T,TS):-dependent(T1,T,TS).
12 enabledC(T,TS):-not notenabledC(T,TS),transHistory(T,H,TS),H>0.
13 {reversesC(T,TS)}:-enabledC(T,TS).

```

Lines 1–9 of Listing 3.2 define predicate **dependent**(T2,T1,TS), which is true if the execution of transition T2 depends on the execution of transition T1. The rule in lines 1–4 requires that this is the case if $\max(H(T2)) > \max(H(T1))$ and there is Q such that $Q \in F(T1, x)$ and $Q \in F(y, T2)$. The rule in lines 5–9 covers the similar case where $Q \in F(T1, x)$, $Q1 \in F(y, T2)$, and $Q1 \in \text{con}(Q, M_{TS}(P))$. Lines 11 and 12 encode coenableness, whereas the rule in line 13 encodes the choice of reversing a transition or not.

The ASP encoding can be used to tackle complex reasoning tasks about RPNs. Consider for instance the network of Fig. 1 and its ASP representation in Listing 3.3. Lines 1–6 define the places, transitions and tokens of the network, whereas lines 8–24 list all arcs (both incoming and outgoing) associated with each transition. For instance, lines 11–14 define all arcs related to transition **t2**. Finally, line 26 represents the initial marking.

Listing 3.3. ASP representation of network of Fig. 1

```

1 place(r). place(s). place(t). place(u). place(v).
2 place(x). place(y). place(z).
3
4 trans(t1). trans(t2). trans(t3). trans(t4).
5
6 token(a). token(b). token(c). token(d).
7
8 ptarc(r,t1,d). ptarc(s,t1,a).
9 tparc(t1,u,a). tparc(t1,u,d).
10
11 ptarc(t,t2,b). ptarc(u,t2,a).
12 ptarc(u,t2,d).
13 tparc(t2,x,a). tparc(t2,x,b). tparc(t2,x,d).
14 tparcbond(t2,x,a,b).
15
16 ptarc(v,t3,c). ptarc(u,t3,a). ptarc(u,t3,d).
17 tparc(t3,y,a). tparc(t3,y,c). tparc(t3,y,d).
18 tparcbond(t3,y,a,c).
19
20 ptarc(v,t4,c). ptarc(x,t4,a). ptarc(x,t4,b).
21 ptarc(x,t4,d). ptarcbond(x,t4,a,b).
22 tparc(t4,z,a). tparc(t4,z,b). tparc(t4,z,c).
23 tparc(t4,z,d).
24 tparcbond(t4,z,a,b). tparcbond(t4,z,b,c).
25
26 holds(r,d,0). holds(s,a,0). holds(t,b,0). holds(v,c,0).

```

Then, query

```

goal:- connected(P,a,c,T),place(P),time(T).
:- not goal.

```

asks for a reachable state where there is a place P s.t. $a - c \in M(P)$. The ASP encoding returns the answer `fires(t1,0) fires(t3,1)`. For query

```

goal:-C>1,C=#count{K2:connected(P,K1,K2,T),token(K2)},holds(P,K1,T).
:- not goal.

```

where a state is sought where some place holds a bond with at least three tokens, `clingo` finds the solution `fires(t1,0) fires(t2,1) fires(t4,2)`. We can arbitrarily increase the complexity of the analysis, by combining aggregates such as `#count` in conjunctive or disjunctive queries such as

```

goal1(T):- C>1,C=#count{K2:connected(P,K1,K2,T),token(K2)},
           holds(P,K1,T).
goal2(T):- connected(P,a,c,T), not connected(P,a,b,T),time(T).
goal:- goal1(T1),goal2(T2),T2>T1,time(T1),time(T2).
:- not goal.

```

where we search for a sequence of transitions that first create a bond with at least three tokens, and then a bond with a and c but without b . The answer computed now is `fires(t1,0) fires(t2,1) fires(t4,2) reversesC(t4,3) reversesC(t2,4) fires(t3,5)`.

4 Conclusions

We have presented work in progress towards a methodology for analysing reversible systems modeled as RPNs based on ASP. We argue that ASP allows an expressive and flexible methodology for defining models and their properties, which can handle difficult queries on complex models efficiently. As future work, we plan to extend our translation to out-of-causal reversibility, to capture a variety of RPN properties, and to apply the framework on realistic systems. We remark that a complementary approach which we are also exploring is the possibility to exploit existing model-checking Petri net tools (CPN tools [10]) for analysing RPN models through a translation of RPNs to coloured Petri nets [4]. CPN tools is a graphical tool for the simulation of Reversing Petri nets. However, it does not pursue more than one simulation and it breaks transition choice ties randomly. Its effectiveness is still to be verified as the approach is characterized by a blow-up in the state space during the translation to coloured Petri nets.

References

1. Anwar, S., Baral, C., Inoue, K.: Encoding higher level extensions of petri nets in answer set programming. In: Cabalar, P., Son, T.C. (eds.) LPNMR 2013. LNCS (LNAI), vol. 8148, pp. 116–121. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40564-8_12
2. Anwar, S., Baral, C., Inoue, K.: Encoding Petri nets in answer set programming for simulation based reasoning. TPLP, 13(4-5-Online-Supplement) (2013)
3. Anwar, S., Baral, C., Inoue, K.: Simulation-based reasoning about biological pathways using Petri nets and ASP. Logical Modeling of Biological Systems, pp. 207–243 (2014)
4. Barylska, K., Gogolinska, A., Mikulski, L., Philippou, A., Piatkowski, M., Psara, K.: Reversing computations modelled by coloured Petri nets. In: Proceedings of ATAED 2018, CEUR Workshop, vol. 2115, pp. 91–111 (2018)
5. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Morgan Claypool Publishers, San Rafael (2012)
6. Heljanko, K., Niemelä, I.: Bounded LTL model checking with stable models. TPLP 3(4–5), 519–550 (2003)
7. Lifschitz, V.: Answer Set Programming. Springer, Berlin (2019)
8. Philippou, A., Psara, K.: Reversible computation in petri nets. In: Kari, J., Ulidowski, I. (eds.) RC 2018. LNCS, vol. 11106, pp. 84–101. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99498-7_6
9. Philippou, A., Psara, K., Siljak, H.: Controlling reversibility in reversing petri nets with application to wireless communications. In: Thomsen, M.K., Soeken, M. (eds.) RC 2019. LNCS, vol. 11497, pp. 238–245. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21500-2_15
10. Ratzer, A.V., et al.: CPN tools for editing, simulating, and analysing coloured petri nets. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 450–462. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-44919-1_28