Answer Set Programming Method for Network Completion for Time-Varying Genetic Networks modeled in Process Hitting

Emna Ben Abdallah[1], Tony Ribeiro[2], Morgan Magnin[1,2], and Katsumi Inoue[1]

[1] LUNAM Université, École Centrale de Nantes, IRCCyN UMR CNRS 6597
(Institut de Recherche en Communications et Cybernétique de Nantes),
1 rue de la Noë, 44321 Nantes, France.
[2] National Institute of Informatics,
2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan.

Abstract. The modeling of biologicals systems relies on background knowledge, deriving either from literature and/or the analysis of biological observations. But with the development of high-throughput data, there is a growing need for methods that are able to automatically revise the existing models with regard to additional observations obtained by biologists. Our research aims at providing a logical approach to revise biological regulatory networks thanks to time series data, *i.e.*, gene expression data depending on time. In this paper, we propose two revision methodologies for models expressed through a timed extension of the Process Hitting framework (which is a restriction, well suited for biological systems, of networks of synchronizing timed automata). The revision methods we introduce aim to make the minimum amount of modifications (addition/deletion of actions between biological components) to a given input model so that the resulting model is most consistent with the observed data. They are implemented in Answer Set Programming, which is a paradigm proven efficient to tackle knowledge representation problems. We illustrate the merits of our methods in a twofold way: (1) we exhibit the benefits of such automatic approaches on a case study consisting of a qualitative model of circadian clock; (2) we lead computational experiments on various benchmarks to show the performances of our algorithms.

Keywords: Answer Set Programming, Process Hitting, dynamic modeling, time-varying genetic networks, network revision

1 Introduction

Answer set programming (ASP) is a form of declarative programming that has been successively used in many knowledge representation and reasoning tasks [**?**,**?**,**?**]. In ASP, a problem is represented by a logic program where the answer sets correspond to the solutions of the problem. Solving the problem is then reduced to computing stable models using answer set solvers like *clasp* [**?**,**?**].

2 Answer Set Programming

In this section, we recapitulate the basic elements of ASP. An answer set program is a finite set of rules of the form

$$a_0 \text{ :- } a_1,\ \ldots,\ a_m,\ not\ a_{m+1},\ \ldots,\ not\ a_n \tag{1}$$

where $n \geq m \geq 0$, a_0 is a propositional atom or $\bot$, all $a_1, \ldots, a_n$ are propositional atoms and the symbol "*not*" denotes default negation. If $a_0 = \bot$, then Rule (1) is a constraint (in which case a_0 is usually omitted). The intuitive reading of a rule of form (1) is that whenever $a_1, \ldots, a_m$ are known to be true and there is no evidence for any of the default negated atoms $a_{m+1}, \ldots, a_n$ to be true, then a_0 has to be true as well. Note that $\bot$ can never become true.

In the ASP paradigm, the search of solutions consists in computing answer sets of answer set program. An answer set for a program is defined by Gelfond and Lifschitz [**?**] as follow. An interpretation I is a finite set of propositional atoms. An atom a is true under I if $a \in I$, and false otherwise. A rule r of form 1 is true under I if $\{a1,\ \ldots,\ a_m\} \subseteq I$ and $\{a_{m+1},\ \ldots,\ a_n\} \cap I = \emptyset$ implies $a_0 \in\ I$. An Interpretation I is a model of a program P if each rule $r \in P$ is true under I. Finally, I is an answer set of P if I is a subset-minimal model of P^I, where P^I is defined as the program that results from P by deleting all rules that contain a default negated atom from I, and deleting all default negated atoms from the remaining rules. Programs can yield no answer set, one answer set, or many answer sets. To compute answer sets of an answer set program, we run an ASP solver.

3 Process Hitting

Definition 1 introduces the Process Hitting(PH) [**?**] which allows to model a finite number of local levels, called *processes*, grouped into a finite set of components, called *sorts*. A process is noted a_i, where a is the sort's name, and i is the process identifier within sort a. At any time, exactly one process of each sort is *active*, and the set of active processes is called a *state*.

The concurrent interactions between processes are defined by a set of *actions*. Each action is responsible for the replacement of one process by another of the same sort conditioned by the presence of at most one other process in the current state. An action is denoted by $a_i \rightarrow b_j \nearrow b_k$, which is read as a_i *hits* b_j to make it *bounce* to b_k, where a_i, b_j, b_k are processes of sorts a and b, called respectively *hitter*, *target* and *bounce* of the action. We also call a *self-hit* any action whose hitter and target sorts are the same, that is, of the form: $a_i \rightarrow a_i \nearrow a_k$.

The PH is therefore a restriction of synchronous automata, where each transition changes the local state of exactly one automaton, and is triggered by the local states of at most two distinct automata. This restriction in the form of the actions was chosen to permit the development of efficient static analysis methods based on abstract interpretation [**?**].

Definition 1 (Process Hitting). *A* Process Hitting *is a triple* $(\Sigma, \mathcal{L}, \mathcal{H})$ *where:*

- $\Sigma = \{a, b, \dots\}$ *is the finite set of* sorts*;*
- $\mathcal{L} = \prod_{a \in \Sigma} \mathcal{L}_a$ *is the set of* states *where* $\mathcal{L}_a = \{a_0, \dots, a_{l_a}\}$ *is the finite set of* processes *of sort* $a \in \Sigma$ *and* l_a *is a positive integer, with* $a \neq b \Rightarrow \mathcal{L}_a \cap \mathcal{L}_b = \emptyset$*;*
- $\mathcal{H} = \{a_i \rightarrow b_j \nearrow b_k \in \mathcal{L}_a \times \mathcal{L}_b^2 \mid (a, b) \in \Sigma^2 \wedge b_j \neq b_k \wedge a = b \Rightarrow a_i = b_j\}$ *is the finite set of* actions*.*

Example 1. The figure 1 represents a $\mathcal{PH}$ $(\Sigma, \mathcal{L}, \mathcal{H})$ with three sorts ($\Sigma = \{a, b, c\}$) and: $\mathcal{L}_a = \{a_0, a_1\}$, $\mathcal{L}_b = \{b_0, b_1\}$, $\mathcal{L}_z = \{z_0, z_1, z_2\}$.

A state of the networks is a set of active processes containing a single process of each sort. The active process of a given sort $a \in \Sigma$ in a state $s \in \mathcal{L}$ is noted $s[a]$. For any given process a_i we also note: $a_i \in s$ if and only if $s[a] = a_i$. The dynamic of the PH networks is satisfied thanks to the actions. Indeed, the transition from one state s_1 to its successor s_2 is done when there is a playable action (definition 2) at s_1. After each transition only one sort, or one component, changes its level from one process to another.

Definition 2 (Playable action). *Let* $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H})$ *be a Process Hitting and* $s \in \mathcal{L}$ *a state of PH. We say that the action* $h = a_i \rightarrow b_j \nearrow b_k \in \mathcal{H}$ *is* playable in state s *if and only if* $a_i \in s$ *and* $b_j \in s$ *(i.e.* $s[a] = a_i$ *and* $s[b] = b_j$*). The resulting state after playing* h *in* s *is called a* successor *of* s *and is denoted by* $(s \cdot h)$*, where* $(s \cdot h)[b] = b_k$ *and* $\forall c \in \Sigma, c \neq b \Rightarrow (s \cdot h)[c] = s[c]$*.*

We note that during these last years the Process Hitting framework was improved and we added new type of sorts like cooperative sorts and new actions like plural actions, actions with priority and actions with delay.

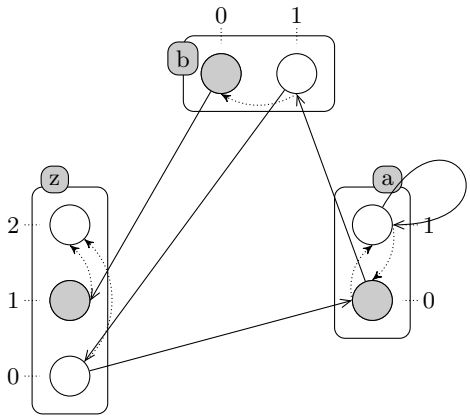

Fig. 1. A PH model example with three sorts: a, b and z (a is either at level 0 or 1, b at either level 0 or 1 and z at either level 0, 1 or 2). Boxes represent the *sorts* (network components), circles represent the *processes* (component levels), and the 5 *actions* that model the dynamic behavior are depicted by pairs of arrows in solid and dotted lines. The grayed processes stand for the possible initial state: $\langle a_1, b_0, z_1 \rangle$.

In some cases it is necessary to represent a reaction of a set of components on one component. For example in the bio-chemical reactions :$X \xrightarrow{Y} Z$ or $X + Y \rightarrow Y + Z$, where X is a set of reactives, Y a set of catalysts and Z a set of products.The plural action permits to represent this kind of reactions in PH. The plural is made up of two sets of processes of different sorts, which represent all the hitters and the bonds.

Definition 3 (Plural action). *It is a reaction of this form:*
$A \rightarrowtail B \mid A, B \in Proc \setminus \emptyset \wedge \forall q \in B, \exists p \in A, (p \neq q \wedge \Sigma(p) = \Sigma(q))$

Example 2. We give a simple example to represent a plurial action by a cooperation between two biological components (x and y) in order to activate another component (z) and change its level from 0 to 1: $\{x_1, y_1, z_0\} \rightarrow \{x_1, y_1, z_1\}$.

In some dynamics it is crucial to have information about the delays between two events (two states in PH). The normal actions cannot show this information we just know that the state s_2 will be after s_1 in the next step but it is not possible to know how much time this transition takes time. We propose to add the delay in the action attributes which is responsable of the transition between the two states. That means that this action needs to be played during a specific time so that the system doesn't change the state (defintion 4).

Definition 4 (timed action).

Definition 5 (temporised sort).

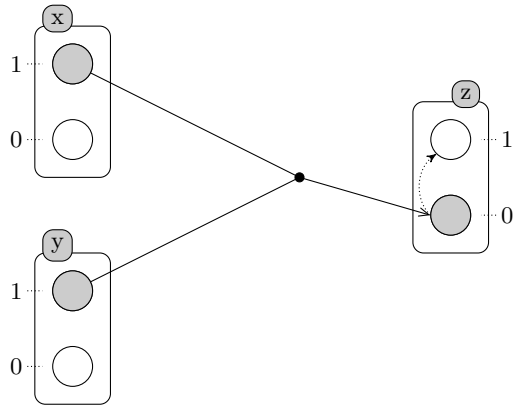

Fig. 2. Representation of a plural action in Process Hitting network: $\{x_1, y_1, z_0\} \rightarrow \{x_1, y_1, z_1\}$.

The PH was chosen for several reasons. First, it is a general framework that, although it was mainly used for biological networks, allows to represent any kind of dynamical model, and converters to several other representations are available (see sectionAlthough an efficient dynamical analysis already exists for this framework, based on an approximation of the dynamics, it is interesting to identify its limits and compare them to the approached we present later in this paper. Finally, the particular form of the actions in a PH model allow to easily represent them in ASP, with one fact per action, as described in the next section. Other representations may have required supplementary complexity; for instance, a labeling would be required if actions could be triggered by a variable number of processes.

The rest of the report focuses on the representation of the previous definitions through ASP than we give the example of Circadian Clock network. Later we propose an approch to resolve the completion problem of PH networks with the use of ASP.

4 Circadian clock

4.1 Definition

4.2 Circadian clock in PH

5 PH through ASP

5.1 Translation of PH networks to ASP

PH network is easy to be presented in ASP. Indeed we need only 3 predicates to define the whole network: `"sort"` to define sorts, `"process"` for the processes and `"action"` for the network actions. We will see in example **??** how a PH network is defined with these predicates.

Example 3 (Representation of a PH network in ASP). The representation of the PH network of figure 1 in ASP is the following:

```
1  process("a", 0..1). process("b", 0..1). process("z", 0..2).
2  sort(X) ←process(X,I)
3  action("a",0,"b",1,0). action("a",1,"a",1,0). action("b",1,"z",0,2).
4  action("b",0,"z",1,2). action("z",0,"a",0,1).
```

In line 1 we create the list of processes corresponding to each sort, for example the sort `"z"` has 3 processes numbered from `0` to `2`; this specific predicate will in fact expand into the three following predicates: `process("z", 0)`, `process("z", 1)`, `process("z", 2)`. Line 2 enumerates every sort of the network from the previous information. Finally, all the actions of the network are defined in lines 3 and 4; for example, the first predicate `action("a",0,"b",1,0)` represents the action $a_0 \to b_1 \Rsh b_0$.

Example 4 (Representation of PH network with plural-timed actions in ASP). We can take the example at the figure 2 and consider that the action $\{x_1, y_1, z_0\} \xrightarrow{D} \{x_1, y_1, z_1\}$ has a delay "D" and in PH the action becomes: $x_1 \wedge y_1 \to z_0 \Rsh z_1$. So that the PH network in ASP is:

```
5  process("x", 0..1). process("y", 0..1). process("z", 0..1).
6  action("x",1,"y",1,"z",0,1, D).
```

The number of indegree in this action $i = 2$, there is only 2 hitters. It is possible to have an indegree greater than 2. We will show later that the number of the maximum indegree should be an input for in our algorithm.

The predicat `action` represents the ordinary actions as well as the plural actions. Indeed an ordinary action is a plural action with an indegree 1. Moreover all the ordinary actions are a timed action with a dealy equal to 1 unit of time. Each action need at least one step to be played For example the action `action("a",0,"b",1,0)` is equivalent to `action("a",0,"b",1,0,1)` So the plural-timed action is a generalized way to represent the actions in a Process Hitting network. Thus in the following part we will consider only plural-timed actions.

5.2 Circadian clock in ASP

6 Completion of PH networks

6.1 Asynchronous and non-deterministic networks

6.2 Formalization

Definition 6 (Applicability). *Let C be a chronogram and a be an action of a Process Hitting PH such that $a = action(S_1, P_1, \ldots, S_n, P_n, G, P, P', D)$. Let t be a time step of C, the action a is applicable at t, iff $\forall t', 0 \leq t - D < t' < t$, the active process of each sort S_i is P_i and the active process of the sort G is P.*

Definition 7 (Consistency). *Let C be a chronogram and a be an action of a Process Hitting PH such that $a = action(S_1, P_1, \ldots, S_n, P_n, G, P, P', D)$. The action a is consistent with C, if there is a gene change at each time step t of C where a can be applied.*

6.3 Algorithm

In this section we propose two algorithms to complete Process Hitting networks. Both algorithms takes as input a Process Hitting and a chronogram of genes evolutions of this network. Knowing the genes influences (or assuming all possible influences), these algorithms will complete the input network by adding the delayed actions that could have realized the changes observed in the chronogram. The first algorithm we propose is a complete approach: assuming that all input observations are perfect it will correctly revise the input Process hitting by adding new actions and removing existing ones. Algorithm **??** show the pseudo code of our completion algorithm. It first consider the gene changes, to generates all possible actions that can realize each change. At this step, new actions can be added to the input network to perform the changes observed that no current action in the network could perform. Then, it consider the time steps where there is no changes to remove the actions that would made one at this moment. Here, both the actions generated and existing ones that are not consistent with the observation are removed from the network.

Theorem 1 (Completeness and Correctness). *Let PH be a Process Hitting, C be a chronogram of the genes of PH and A be the set of actions of PH whose realized the chronogram C. Let PH' be a Process Hitting and A' be the set of actions of PH' such that $A' \subseteq A$. Given PH' and C as input, Algorithm 1 is complete and correct: it will output a process hitting PH'', with A'' the set of actions of PH'' such that $A \subseteq A''$ (A'' can realize C) and all action $a \in A''$ are consistent with C.*

Proof. Lets suppose that the algorithm is not complete, then there is an action $a \in A$ that realized C and $a \notin A''$. After step 1.2, PH'' contains all actions that can realized each gene change. Here there is no action $a \in A$ that realized C which is not generated by the algorithm, so $a \in A''$. Then it implies that at step 2, the action a is removed from A''. It means that there is a time step t where

Algorithm 1 PH-Completion($PH, Chronogram, Influences, indegree$)

- INPUT: a Process Hitting PH', a chronogram C of the genes evolution of a Process Hitting PH, the genes influences and a maximal action indegree i.
- Let $PH'' := PH'$
- Step 1: For each time where a gene G changes its Pue from P to P' in C:
  - 1.1: Let D be the delay since the last time a gene has changed.
  - 1.2: Generate all actions with delay D which involve all subsets of size i of the genes $S_1, \ldots, S_n$ having an influence on G:

$$A := action(S_1, P_1, \ldots, S_n, P_n, G, P, P', D)$$

  - Add all action A in PH''
- Step 2: For each time step t where there is no gene change
  - Remove from PH'' all actions that are applicable at t
- OUTPUT: PH'' a completed Process Hitting that realizes the chronogram.

a is applicable, but in the chronogram C there is no gene change at t. But since $a \in A$, a change should have occurs at t, this is a contradiction. So there is no $a \in A$ that could be remove from A'' at step 2, the algorithm is complete. □

The algorithm is also correct and this conclusion is trivial from the operation of step 2. If it was not correct, then there is an action $a \in A''$ that is not consistent with C. But all such action are removed at step 2. So that the algorithm output is correct.

All action of the outputted Process Hitting PH' are consistent with the input chronogram C.

The second algorithm we propose is an approximated approach: assuming that input observations can be noisy it will revise the input Process hitting by adding action and changing the delay of existing ones. Algorithm **??** show the pseudo code of our approximated algorithm. Like in Algorithm **??**, it generates all possible actions that can realize each change observed. But then, since observation are not perfect, we cannot safely removed non-consistent actions. Here, we just merge actions by replacing the one that only differ by there delay by one action where its delay is the average. The intuition is that, in practice, if there is enougth observation, the delay of those actions should tend to the real Pue.

Theorem 2 (Complexity). *Let PH be a Process Hitting, S be the number of sorts of PH and P be the maximal number of processes of a sort of PH. Let C be a chronogram of the genes of PH over T units of time, such that c is the number of gene change of C. The complexity of completing PH by generating actions from the observations of C with Algorithm 1 belongs to $O(c * i^S + (T - c) * P * i^S)$ that is bound by $O(T^2 * P^{S+2})$. The complexity of completing PH by generating actions from the observations of C with Algorithm 2 belongs to $O(c * i^S + (T * P * i^S)^2)$ that is bound by $O(T * P^S + T^2 * P^{2S+2})$.*

Proof. Let i be the maximal indegree of an action in PH, $0 \leq i \leq P$. Let p be a process of PH and n be the number of sorts that can influence p. There

Algorithm 2 PH-Completion($PH, Chronogram, Influences, indegree$)

- INPUT: a Process Hitting PH, a chronogram C of the genes evolution of PH, the genes influences and a maximal action indegree i.
- Let $PH' := PH$
- Step 1: For each time where a gene G changes its Pue from P to P' in C:
  - 1.1: Let D be the delay since the last time a gene has changed.
  - 1.2: Generate all actions with delay D which involve all subsets of size i of the genes $S_1, \ldots, S_n$ having an influence on G:

$$A := action(S_1, P_1, \ldots, S_n, P_n, G, P, P', D)$$

  - Add all action A in PH'
- Step 2: Merge each action with the same hitters, $S_1, P_1, \ldots, S_n, P_n$ and the same target, G, P, P', into one action where the delay is the average.
- OUTPUT: a completed Process Hitting that realize the chronogram.

is i^S possible combinations of those process that can hit p, each of those can form an action. There is P process and atmost T possibles delay, so that there are $T * P * i^S$ possibles actions, thus the memory of our algorithm is bound by $O(T * P * i^S)$, which belongs to $O(T * P^{S+1})$ since $0 \leq i \leq P$.

Atmost i^S actions can be produced for each gene change, the complexity of step 1 is then $O(c * i^S)$. In step 2 of Algorithm 1 at each time step where there is no gene change, it checks all actions generated so far. Since there is $T * P * i^S$ possibles actions, this operation complexity is $O((T - c) * T * P * i^S)$. Then the complexity of Algorithm 1 is $O(c * i^S + (T - c) * T * P * i^S)$. Since $0 \leq i \leq P$ and $0 \leq c \leq T$ the complexity belongs to $O(T * P^S + T^2 * P^{S+1})$ and is bound by $O(T^2 * P^{S+2})$.

In Algorithm 2, merging the actions at step 2 is polynomial in the number of actions and the complexity of this operation belongs to $O((T * P * i^S)^2)$. The complexity of this algorithm belongs to $O(c*i^S+(T*P*i^S)^2)$. Since $0 \leq i \leq P$ and $0 \leq c \leq T$ the complexity of Algorithm 2 is bound by $O(T*P^S+(T*P*P^S)^2) = (T * P^S + T^2 * P^{2S+2})$. □

7 Conclusion and perspectives