Reaction Rules and Ecosystemic Hypergraph

Franck Pommereau[1] and Cédric Gaucherel[2]

[1] IBISC, Univ. Évry, Univ. Paris-Saclay, 91025 Évry, France
`franck.pommereau@univ-evry.fr`
[2] AMAP-INRA, CIRAD, CNRS, IRD, Univ. Montpellier, 34398 Montpellier, France
`cedric.gaucherel@cirad.fr`

Abstract. Ecologist use to reason on ecosystems through an *ecosystemic graph* (EG) whose nodes are the elements constituting the ecosystem, and whose arcs represent the influence of one element onto another. Recently, a formalism has been proposed to model the behaviour of ecosystems: the elements of an ecosystem are modelled with Boolean variables while "reaction rules" allow to describe how these variables may change. In this paper, we show how a reaction rule (RR) model can be translated to an EG, enlightening that crucial information is lost in this translation so the EG is intrinsically an ambiguous informal object. Then we propose to replace EG with *ecosystemic hypergraphs* (EH) that convey exactly the same information as RR systems. We believe that EH can support the same kind of visual reasoning as EG while being precise and analysable as RR. We then show how an EH can be seen as a Petri net extended with read-, inhibitor-, and reset-arcs, and we finally show that such a net can be translated to a regular Petri net that corresponds to the original Petri net semantics of RR systems.

1 Introduction

In order to describe an ecosystem, ecologists use to resort to an *ecosystemic graph* (EG) which is a graph like that on Figure 1 such that:

- its nodes are the physical elements that constitute the ecosystem (species, abiotic components, human artefacts, etc.);
- its arcs correspond to the processes that allow the elements to evolve, more precisely, an arc $A \to B$ describes the fact that element A has an influence on element B.

Figure 1 depicts such an EG that describes as simplified version of the termites ecosystem presented in [1]. This system includes the following elements:

`Rp`	the termites reproductives;
`Wk`	the termites workers;
`Sd`	the termites soldiers;
`Wd`	the wood used by the termites to build their mound and grow fungi;
`Fg`	the chambers in the mound dedicated to grow fungi;
`Te`	the fungi grown by the termites;

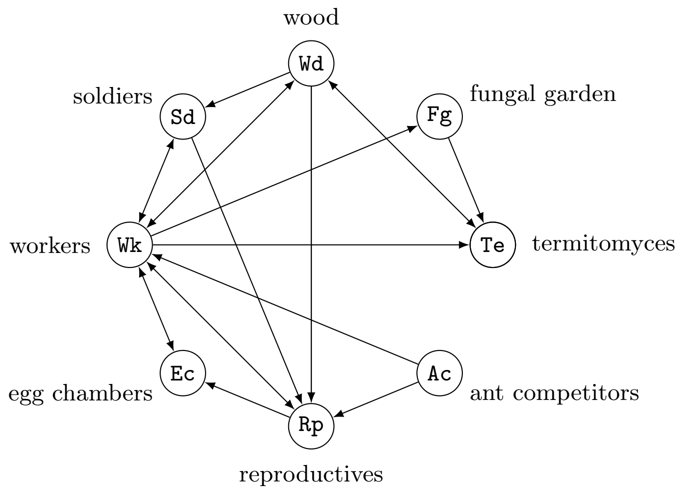


Fig. 1. Ecosystemic graph of the termites ecosystem, annotated with a short textual description of each element.

`Ec`	the chambers in the mound used as nursery;
`Ac`	the ant competitors that may attack the termites colony.

These elements evolve all together, depending one of the others. For instance, ant competitors may kill the workers and reproductives, but not the soldiers; so there in an arc in this EG from `Ac` towards `Wk` and `Rp`, but not towards `Sd`. In the more complete version of this system considered in [1], soldiers may kill the ants so there is also an arc from `Sd` towards `Ac`.

Each such interaction has to be described, which is generally made in natural language, as we did above. Such a description remains informal and is not suitable for a rigorous analysis. To this aim, a language has been proposed in [1] to formally describe ecosystems, we call it the language of *reaction rules* (RR language).

2 Reaction Rules

RR language allows to describe ecosystems as a set of Boolean variables representing its elements, plus a set of rules representing the processes involving these elements. Figure 2 shows a version of the termites ecosystem modelled using RR. In the left-hand column, all the elements are defined and listed under chosen categories that allow to classify them (inhabitants, structures, resources, and competitors). Each element is defined as a name (*e.g.*, `Rp`), an initial value (plus or minus; *e.g.*, `Rp` is initially on), and a short textual description. Then, the processes are defined as *constraints* or *rules*, collectively called *actions*. Both describe how some variables may be assigned depending on the value of other variables. To do so, the left-hand side of an action lists a set of variables values that acts

```
inhabitants:                        constraints:
  Rp+: reproductives                  Fg- >> Te-
  Wk-: workers                      rules:
  Sd-: soldiers                       Rp+ >> Ec+
  Te-: termitomyces                   Rp+, Ec+ >> Wk+
structures:                           Wk+ >> Wd+, Te+, Fg+, Ec+
  Ec-: egg chambers                   Wk+, Wd+ >> Sd+, Rp+
  Fg-: fungal gardens                 Wk+, Te+ >> Wd-
resources:                            Wd- >> Wk-, Te-
  Wd-: wood                           Wk- >> Fg-, Sd-
competitors:                          Wk-, Rp- >> Ec-
  Ac+: ant competitors                Ac+, Sd- >> Wk-, Rp-
```

Fig. 2. Reaction rules model of the termites ecosystem

as a condition for executing the action, while its right-hand side describes how some variables are assigned when the action is executed. For instance, rule `Ac+, Sd- >> Wk-, Rp-` models that, if ant competitors are present but soldiers are not, then workers and reproductives may be killed. The only difference between constraints and rules is that the former have the priority on the latter (*i.e.*, if a constraint may be executed, no rule is allowed to be executed). Constraints allows to ensure the ecological consistency of the system, for instance, constraint `Fg- >> Te-` states that whenever the fungal gardens disappear, no termitomyce can remain present in the system.

Figure 3 show a grammar in Backus-Naur form of the RR language. We can observe that a model may have no constraints at all, but it must have rules. Moreover, for a RR model to be correct, each action must be consistent in that it cannot have a variable that is listed as both on and off in any of its sides.

```
       model ::= declarations  constraints?  rules
declarations ::= NAME  ":"  ↲  ⇉  variable+  ⇇
    variable ::= value  ":"  TEXT  ↲
       value ::= NAME  sign
        sign ::= "+" | "-"
 constraints ::= "constraints"  ":"  ↲  ⇉  action+  ⇇
       rules ::= "rules"  ":"  ↲  ⇉  action+  ⇇
      action ::= side  ">>"  side  ↲
        side ::= value  (","  value)*
```

Fig. 3. BNF grammar of the RR formalism, where: ↲ stands for a new line; ⇉ stands for the beginning of an indented block; ⇇ stands for the end of an indented block; NAME is any string of alphanumerical characters, except "`constraints`" and "`rules`" that are reserved keywords; TEXT is an arbitrary string with no new line inside. Comments are allowed, they start with character "`#`" and end with the line.

2.1 Semantics

A *state* of a RR system is a valuation of its variables, which we represent by the set of its on variables. The *initial state* is defined by the variables declaration, for instance, the termites model has initial state $\{\texttt{Rp}, \texttt{Ac}\}$. An action is *enabled* at a state s if (1) all the on variables in its left-hand side are present in s, and (2) none of the off variables in its left-hand side are present in s. An enabled action may be executed, yielding a new state s' that is s from which (3) all the on variables in the right-hand side of the action have been added, and (4) all the off variables in the right-hand side of the action have been removed. However, if $s = s'$ then the action is not allowed to be executed and it is considered as not enabled at all.

For instance, in the initial state $s_0 \overset{\text{df}}{=} \{\texttt{Rp}, \texttt{Ac}\}$ of the termites model:

- rule `Rp+ >> Ec+` is enabled because $\texttt{Rp} \in s_0$ and its execution yields a new state $\{\texttt{Rp}, \texttt{Ac}, \texttt{Ec}\} \neq s_0$;
- rule `Rp+, Ec+ >> Wk+` is not enabled because $\texttt{Ec} \notin s_0$;
- rule `Ac+, Sd- >> Wk-, Rp-` is enabled because $\texttt{Ac} \in s_0$ and $\texttt{Sd} \notin s_0$ and its execution yields state $\{\texttt{Ac}\} \neq s_0$;
- constraint `Fg- >> Te-` could be executed but this would not change the state so it is actually not enabled.

We note by $s \xrightarrow{a} s'$ the fact that an action a is enabled at state s and its execution yields state s'.

Given is state s, its *successors* are all the states s' such that:

- $s \xrightarrow{c} s'$ for any constraint c;
- if no constraint is enabled, $s \xrightarrow{r} s'$ for any rule r;
- otherwise, s has no successor and is called a *deadlock*.

The *state space* of a model is a *labelled transition system* $(S, L, \rightarrow)$ where S (the set of reachable states), L (the set of labels), and $\rightarrow$ (the set of labelled transitions) are the smallest sets such that:

- $s_0 \in S$;
- if $s \in S$ and s' is a successor of s such that $s \xrightarrow{a} s'$ for an action a, then $a \in L$, $s' \in S$, and $(s, a, s') \in \rightarrow$ (with is also noted as $s \xrightarrow{a} s'$).

2.2 Translation do ecosystemic graph

It is easy to see that a RR model carries all the information allowing to build an ecosystemic graph. Actually, we can proceed has follows:

1. The nodes of the EG are exactly the variables of the RR model.
2. For each action such that `A` appears in the left-hand side and `B` appears in the right-hand side (with $\texttt{A} \neq \texttt{B}$), there is an arc $\texttt{A} \rightarrow \texttt{B}$ in the EG.

However, the RR carries strictly more information than the EG, in particular:

- the EG does not provide an initial state;
- an arc `A` $\rightarrow$ `B` in the EG does not indicates which value of `A` will cause which change on `B`;
- one arc in the EG may come from different actions in the RR;
- there is no way to know which arcs have to be gathered together into an action.

The first point may be solved by having two kind of nodes in the EG (*i.e.*, those for initially on, resp. off, elements). The second point may be solved by having decorations on the starting and ending of each arc indicating the expected values of the variables. It would also be necessary to allow multiple arcs between two nodes in the case where multiple actions allow distinct conditions on a variables to cause distinct assignment on another. The two last points cannot be solved using a graph because an action may actually relate several variables (in its left-hand side) to several others (in its right-hand side), while an arc relates only one variable to one other.

To solve this, we propose to use *ecosystemic hypergraphs* (EH) instead as a non ambiguous version of EG. As show below, it turns out that EH can be seen as a graphical version of RR and both are equivalent (*i.e.*, we show how to bijectively translate from one to the other).

3 Ecosystemic Hypergraph

An EH is an hypergraph whose nodes are the variables of an ecosystem, just like in RR and EG, and whose hyperarcs correspond to the actions in RR. In order to annotate these hyperarcs, we will render them as square-shaped nodes connected to the circle-shaped nodes representing the variables. The connections will be annotated with white or black dots representing the sign of the variable they connect to.

Figure 4 (left) shows a very simple EH that corresponds to the termites ecosystem reduced to its last rule and the corresponding variables. It can be read as follows:

- each circle-shaped node is a variable-node, those drawn with double lines are initially on, the others are initially off;

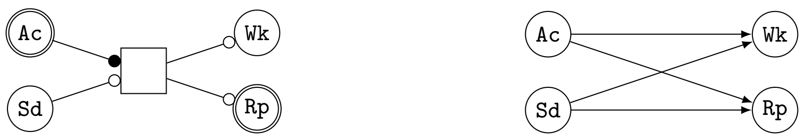


Fig. 4. Left: the EH corresponding to rule `Ac+, Sd- >> Wk-, Rp-` (*i.e.*, a RR system that would contain only this rule and the variables it involves). Right: the EG corresponding to the same system.

- the square-shaped node is an action-node that corresponds to the rule, it represents an hyperarc connecting nodes `Ac` and `Sd` (*i.e.*, the left-hand side of the rule) to nodes `Wk` and `Rp` (*i.e.*, the right-hand side of the rule);
- action-nodes read and write the values of variable-nodes:
  - a black dot at the square-end of an arc means that the action expects the variable to be on; a white dot means that the variable is expected to be off;
  - a black dot at the circle-end of an arc means that the action sets the variable to on; a white dot that it sets it to off;
  - an action may both read and write a variable in which case there will be a dot at each end off the arc connecting the action-node to the variable-node.

This is also summarised in Figure 5 where the two first columns list all the way for a variable to appear in each side of an action, and the third column shows the corresponding EH connection. (The two other columns will be described in the next section.)

Finally, an EH has two kind of action nodes: those representing rules are depicted using a single-line border while those representing constraints are depicted using a double-line border.

It is easy to check that there is a one-to-one correspondence between an EH and a RR, except for the textual description of the variables that actually plays no role in the RR. In particular, both notations precisely define the set of variables and their initial state; as well and the set of constraints and rules, each side of which being precisely encoded as a list of variables values.

As an example, Figure 6 show the EH for our termites RR model.

It turns out that an EH can also be seen as a Petri net, as discussed now.

4 Petri nets semantics

A *multiset* m over a domain D is a function $m : D \to \mathbb{N}$ (natural numbers), where, for $d \in D$, $m(d)$ is the number of occurrences of d in the multiset m. We note bu $D^\star$ the set of all multisets over D. We shall note multisets as sets with repetitions, for instance $m_1 \stackrel{\text{df}}{=} \{1, 1, 2\}$ is a multiset such that $m_1(1) = 2$, $m_1(2) = 1$, and $m_1(x) = 0$ for all $x \notin \{1, 2\}$. A multiset $m \in D^\star$ may be naturally extended to any domain $D' \supset D$ by defining $m(d) \stackrel{\text{df}}{=} 0$ for all $d \in D' \setminus D$. Let $m_1, m_2 \in D^\star$ and $\Delta \subseteq D$, we define:

- $m_1 \leq m_2$ iff $m_1(d) \leq m_2(d)$ for all $d \in D$;
- $(m_1 + m_2) \in D^\star$ is defined by $(m_1 + m_2)(d) \stackrel{\text{df}}{=} m_1(d) + m_2(d)$ for all $d \in D$;
- $(m_1 - m_2) \in D^\star$ is defined by $(m_1 - m_2)(d) \stackrel{\text{df}}{=} \max(0, m_1(d) - m_2(d))$ for all $d \in D$;
- $(m_1 \setminus \Delta) \in D^\star$ is defined by $(m_1 \setminus \Delta)(d) \stackrel{\text{df}}{=} m_1(d)$ for all $d \in D \setminus \Delta$, and $(m_1 \setminus \Delta)(d) \stackrel{\text{df}}{=} 0$ for all $d \in \Delta$;
- $(m_1|_\Delta) \in D^\star$ is defined by $(m_1|_\Delta)(d) \stackrel{\text{df}}{=} m_1(d)$ for all $d \in \Delta$, and $(m_1|_\Delta)(d) \stackrel{\text{df}}{=} 0$ for all $d \in D \setminus \Delta$;
- for $d \in D$, we note by $d \in m_1$ the fact that $m_1(d) > 0$.

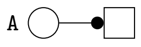
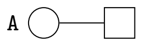
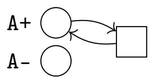
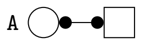
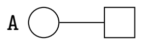
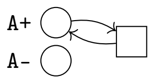
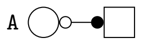
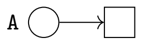
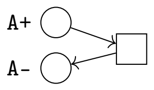
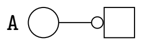
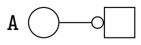
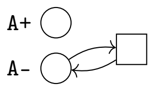
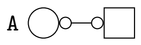
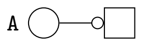
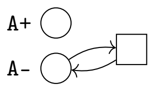
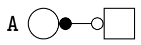
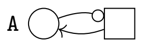
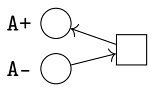
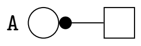
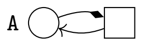
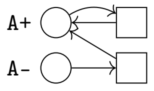
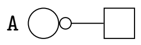
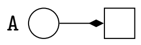
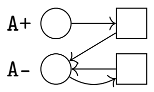
left-hand side	right-hand side	ecosystemic hypergraph	extended Petri net	regular Petri net
A+	(no A)			
A+	A+			
A+	A-			
A-	(no A)			
A-	A-			
A-	A+			
(no A)	A+			
(no A)	A-			

Fig. 5. Columns 1—3: translation between reaction rules and ecosystemic hypergraphs, and vice-versa. Columns 4—5: translation to extended ans regular Petri nets, see Section 4.

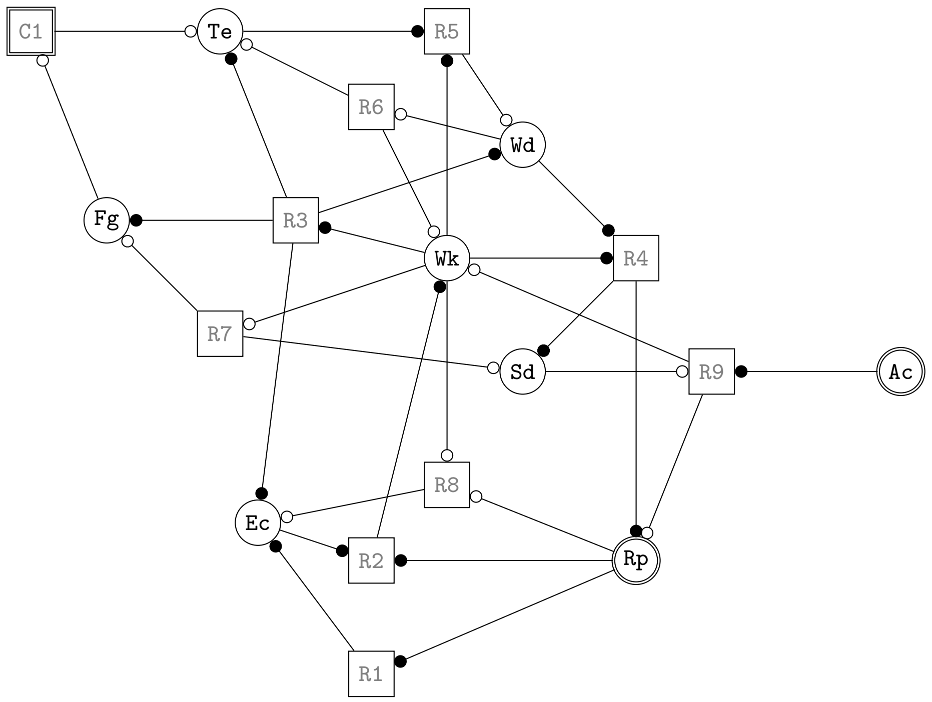


Fig. 6. Translation of the RR model from Figure 2 into the equivalent EH. Action-nodes have been labelled with the name of the corresponding actions (`C1` for the constraint, `R1` to `R9` for the rules, in the order they have been defined.)

4.1 Regular Petri nets

A (regular) *Petri net* (RPN) is a tuple (S, T, W, M) where:

- S is the set of *places*, depicted as circle-shaped nodes;
- T is the set of *transitions*, depicted as square-shaped nodes;
- $W \in ((S \times T) \cup (T \times S))^\star$ is the weight of *arcs*;
- $M \in S^\star$ is the *initial* marking, depicted as black tokens $\bullet$ within places.

Given $x \in S \cup T$, we define ${}^\bullet x \overset{\text{df}}{=} W|_{(S \cup T) \times \{x\}}$ that is the preset of x, and $x^\bullet \overset{\text{df}}{=} W|_{\{x\} \times (S \cup T)}$ that is the postset of x.

A transition $t \in T$ is enabled at a marking M iff ${}^\bullet t \leq M$, which is noted by $M[t\rangle$. In such a case, t may fire, leading to the marking $M' \overset{\text{df}}{=} M - {}^\bullet t + t^\bullet$, which is noted by $M[t\rangle M'$.

Petri nets may be equipped with transitions priorities; the usual way to do so is to provide a binary relation $\prec \subset T^2$ such that $u \prec v$ means that u cannot fire if v is enabled (*i.e.*, v has the priority on u). This shall be used to render the priority of constraints over rules, in which case we may simply define that $u \prec v$ iff u is a transition for a rule and v a transition for a constraint.

4.2 Extended Petri nets

The regular Petri nets as defined above may be extended with:

- *read-arcs* (depicted as bare edges) that allow to test for the presence of tokens without consuming them;
- *inhibitor-arcs* (depicted with a white dot on the transition side) that allow to test for the absence of tokens in a place;
- *reset-arcs* (depicted with a black diamond on the transition side) that allow to consume all the tokens from a place, if any.

An *extended Petri net* (EPN) is thus a tuple (S, T, W, Z, I, R, M) where (S, T, W, M) is a RPN called the *underlying* RPN and:

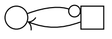
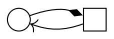
- $Z \in (P \times T)^\star$ defines the read-arcs and for $t \in T$ we define ${}^*t \overset{\text{df}}{=} Z|_{P\times\{t\}}$;
- $I \in (P \times T)^\star$ defines the inhibitor-arcs and for $t \in T$ we define ${}^\circ t \overset{\text{df}}{=} I|_{P\times\{t\}}$;
- $R \subseteq (P \times T)$ defines the reset-arcs and for $t \in T$ we define ${}^\diamond t \overset{\text{df}}{=} R \cap (P \times \{t\})$.

A transition $t \in T$ of an EPN is enabled iff $M[t\rangle$ in the underlying RPN and we have ${}^*t \leq M$ and $M \leq {}^\circ t$. In such a case, t may fire yielding a new marking $M' \overset{\text{df}}{=} (M|_{{}^\diamond t}) - {}^\bullet t + t^\bullet$. EPN may be equipped with transition priorities just like RPN.

4.3 Translating EH to EPN, then to RPN

An EH may be seen as an EPN by considering its variable-nodes as places (marked by one token iff initially on), and its action-nodes as transitions (with adequate priorities). Then, the third and fourth columns of Figure 5 show how each EH arc has to be rendered in the EPN. From this perspective, EH is just a simpler and consistent notation for a subclass of EPN. In particular, we can prove that the EPN we obtain through this translation are always safe, *i.e.*, that no reachable marking can have more than one token in a place.

Proposition 1. *The* EPN *obtained from any* EH *is always safe.*

Proof. The initial marking is safe by definition. Then, it is enough to observe that there exists only two ways to produce a new token in such a EPN. (1)  The inhibitor arc ensures that a token is added to the place only if it is empty. (2)  The reset arc ensures that the place is emptied before a token is added to it. □

Thanks to this property, we can translate our EPNs to RPNs using complementary places: every place `A` is translated to a pair of places `A+` and `A-` whose marking represents respectively the presence of a token in `A` or its absence. See also the last column in Figure 5. Then, a read-arc on `A` can be replaced with a pair of opposite arcs on `A+` (*i.e.*, a side-loop); an inhibitor-arc on `A` can be replaced with a pair of opposite arcs on `A-`. Reset-arcs are more complicated to remove because each such arc represent two situations: either there is no token to delete, or there is one token to delete. So, each reset-arc will give raise to two transitions, and thus, if a transition is connected to n reset arcs, it will be translated into 2^n transitions. Note that this growth in the number of transitions is exactly equivalent to the growth in the number of rules during the *normalisation* step described in the original Petri nets semantics of RR [1]. It is also easy to check that the RPN we obtain from an EH corresponds exactly to that we would have obtained from this original semantics. (Provided that we do not use constraints, which was not defined in [1].)

An important difference between RR and its EPN/RPN equivalent is that Petri nets do not forbid a transition to fire without changing the marking. This can be enforced in a RPN by removing all the transitions t such that ${}^\bullet t = t^\bullet$. But there is no simple way for such a trick in EPNs because of reset arcs that correspond to several situations. So, if EPNs are to be used, we have to explicitly forbid (or remove in a post-processing pass) every firing such as $M[t\rangle M$.

There is also an important difference between an EH and its EPN/RPN translation: the former describes only the initial state and has no defined dynamics while the later may fire transitions and change its marking. It would be straightforward to define an execution semantics of EHs, but we believe that an EH is rather a tool for static reasoning and thus we probably do not need such a semantics.

4.4 EH unfoldings

On the EH from Figure 6, it can be observed that constraint `C1` depends only on `Fg` to be off, which is only obtained by executing rule `R7`. This, in the following, we merge constraint `C1` into rule `R7` by adding `Te-` in its right-hand-side (or, equivalently, we drop node `C1` in the EH and add an arc $\boxed{\text{R7}}\!-\!\!\circ\,\texttt{Te}$). Doing so, we can translate the EH to a RPN without priorities which can be unfolded using one of the numerous tools to do so. However, it may be more interesting to use read-arcs instead of the side-loops because it allows more concurrency in the resulting net. When such a net is unfolded, *e.g.*, using CUNF [2], the result can be depicted borrowing some notations from EH for consistency, see also Figure 7:

- a condition `A+` (resp. `A-`) is depicted using double-lines (resp. single-lines);
- a read-arc is kept as such but draw in dotted-gray to put emphasis on what is changed;
- a regular arc from a condition `A+` (resp. `A-`) is drawn with a black (resp. white) dot at the end because the event consumes the condition;
- a regular arc towards a condition `A+` (resp. `A-`) is drawn using a black (resp. white) dot at the end because the event produces the new condition as described just above.

5 Perspectives

1. Make visible where the unfolding has been cut, and which conditions are regenerated.

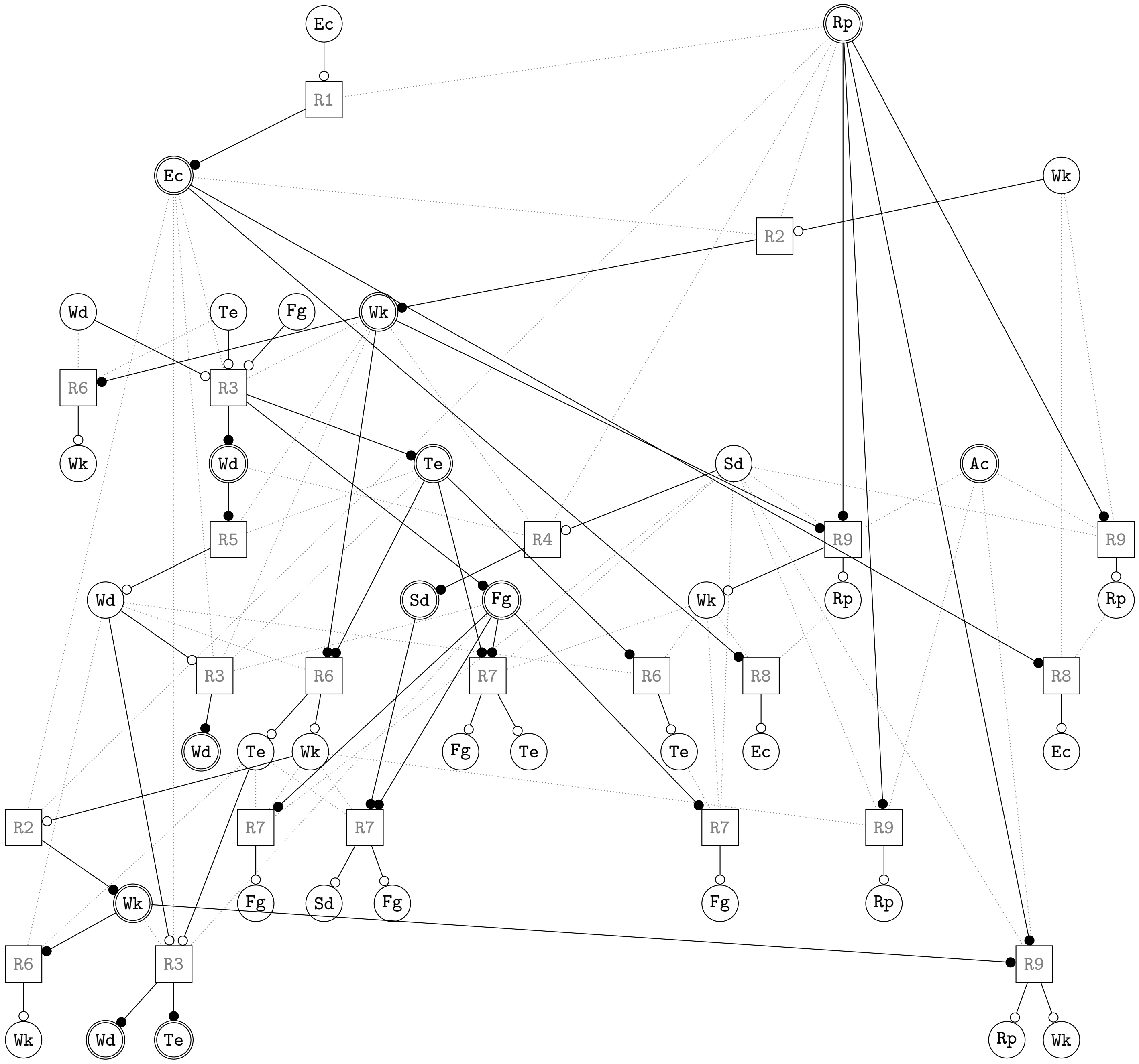


Fig. 7. Unfolding of the EH from Figure 6 when its constraint C1 has been merged into rule R7.

2. Unfold with priorities.
3. Can we expect a better unfolding?
4. Analyse these unfoldings.

References

1. Cédric Gaucherel and Franck Pommereau. Using discrete systems to exhaustively characterize the dynamics of an integrated ecosystem. *Methods in Ecology and Evolution*, 10(9), 2019.
2. César Rodríguez and Stefan Schwoon. Cunf: A tool for unfolding and verifying Petri nets with read arcs. In *ATVA*, volume 8172 of *LNCS*. Springer, 2013.

ptnet. → format pep

py3 → ecco → rr → int (petri-unfold)

ORIGINECODE

↑

/anaconda → 3.7.x

→ $\boxed{\text{networkx}}$ insuffisant (cout) dot
→ LaTeX

line 1019

iPython

```
%run -m ecco model/terminite_simples.rr
%model.spec()
%model.petri()
%model.unfold()
```

} class Model

ecco + modes | branche py3

conda || pip

$\boxed{\text{SKYPE 14h MARDI}}$