**REPORT DOCUMENTATION PAGE**

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA, 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. **REPORT DATE (DD-MM-YYYY)**
   -  

2. **REPORT TYPE**
   - Technical Report

3. **DATES COVERED (From - To)**
   - -

4. **TITLE AND SUBTITLE**
   - Causal analysis of rule-based models of signalling pathways

5a. **CONTRACT NUMBER**
   - W911NF-14-1-0367

5b. **GRANT NUMBER**
   - 

5c. **PROGRAM ELEMENT NUMBER**
   - 

5d. **PROJECT NUMBER**
   - 

5e. **TASK NUMBER**
   - 

5f. **WORK UNIT NUMBER**
   - 

6. **AUTHORS**
   - Jonathan Laurent

7. **PERFORMING ORGANIZATION NAMES AND ADDRESSES**
   - Harvard Medical School
     - 25 Shattuck Street
     - Boston, MA 02115-6027

8. **PERFORMING ORGANIZATION REPORT NUMBER**
   - 

9. **SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS (ES)**
   - U.S. Army Research Office
     - P.O. Box 12211
     - Research Triangle Park, NC 27709-2211

10. **SPONSOR/MONITOR'S ACRONYM(S)**
    - ARO

11. **SPONSOR/MONITOR'S REPORT NUMBER(S)**
    - 66070-NS-DRP.1

12. **DISTRIBUTION AVAILABILITY STATEMENT**
    - Approved for public release; distribution is unlimited.

13. **SUPPLEMENTARY NOTES**
    - The views, opinions and/or findings contained in this report are those of the author(s) and should not contrued as an official Department of the Army position, policy or decision, unless so designated by other documentation.

14. **ABSTRACT**
    - Internship Report - no abstract

15. **SUBJECT TERMS**
    - causality, formal pathways, rule-based models

16. **SECURITY CLASSIFICATION OF:**
    - a. REPORT UU
    - b. ABSTRACT UU
    - c. THIS PAGE UU

17. **LIMITATION OF ABSTRACT**
    - UU

18. **NUMBER OF PAGES**
    - 19a. NAME OF RESPONSIBLE PERSON
      - Walter Fontana
    - 19b. TELEPHONE NUMBER
      - 617-432-5474

---

Standard Form 298 (Rev 8/98)
Prescribed by ANSI Std. Z39.18
Report Title
Causal analysis of rule-based models of signalling pathways

ABSTRACT
Internship Report - no abstract
Causal analysis of rule-based models of signalling pathways

INTERNSHIP REPORT

JONATHAN LAURENT
jonathan.laurent@ens.fr

Under the supervision of:

WALTER FONTANA
walter_fontana@hms.harvard.edu

JÉRÔME FERET
feret@ens.fr
Acknowledgment

This work was sponsored by the Defense Advanced Research Projects Agency (DARPA) and the U.S. Army Research Office under grant number W911NF-14-1-0367. The views, opinions, and/or findings contained in this report are those of the authors and should not be interpreted as representing the official views or policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the Department of Defense.
1 Introduction

When a cell receives a signal from its environment through a membrane receptor, a cascade of internal reactions is triggered which leads ultimately to its response, usually a change in its metabolism, its ability to divide, or in the expression level of a gene. Many diseases including some forms of cancer are related to some defects in these signalling pathways. Such biological systems usually feature thousands of proteins interacting in a highly concurrent way and their complexity makes modelling necessary to the development of better biocuration techniques.

1.1 The kappa modelling language

The kappa language was designed to model interactions between proteins with rewriting rules over site-graphs. We illustrate this with the example of phosphorylation reactions.

Many proteins can be turned on and off by attaching or removing phosphate groups to some of their residues. The action of attaching a phosphate group to a protein is called phosphorylation and an enzyme catalyzing a phosphorylation reaction is called a kinase. A kinase may need to be phosphorylated itself in order to be effective and so signalling pathways often feature phosphorylation cascades. In kappa, proteins are modelled by agents. An agent features some sites through which it can bind other agents. Moreover, some sites can hold an internal state, usually \( u \) when unphosphorylated and \( p \) when phosphorylated. The number and the nature of the sites featured by an agent depend on its type, each type of agent being described in the signature of the kappa model. In most of the examples of this report, signatures have to be inferred directly from the rules.

Let’s model a kinase by an agent of type \( K \) with one binding site \( d \). It acts on a substrate which is an agent of type \( S \) with a binding site \( d \) and two phosphorylation sites \( x \) and \( y \) which can hold the states \( u \) or \( p \). Below are the textual and graphical representations of a kinase with a free binding site and a substrate which is phosphorylated at site \( y \) but not at site \( x \).

\[
K(d) : \quad S(d,x_u,y_p) : \quad d \quad d \quad S
\]

The interaction between the kinase and the substrate is captured by the rules displayed Figure [1]. The first rule is bidirectional and states that a kinase and a substrate can bind (\( b \)) and unbind (\( u \)) to each other through their sites \( d \). The second rule states that if a kinase is bound to a substrate with an unphosphorylated site \( x \), the latter can get phosphorylated. The third does the same for
In the textual notation, a shared exponent denotes the existence of a bond between two sites. Sites without exponents are considered to be free.

An important remark is that a rule can feature underspecified agents. When it does, it can be triggered whatever the binding or internal states of the missing sites are. This characteristic of kappa makes it different from most modelling techniques traditionally used by biologists like differential equations systems or Petri nets. Indeed, the latter require that one variable is introduced for each fully specified species, which leads quickly to a combinatorial explosion.

In kappa, a reaction mixture is modelled by a large site-graph. When a pattern matching the left hand side of a rule \( r \) is recognized in it, it can be updated locally according to \( r \). The agents preserved by \( r \) are those of the longest prefix featuring agents of the same type and mentioning the same sites in both sides of \( r \). The sites they mention are updated according to their state in the right hand side of \( r \) and the others are left unchanged. Agent featured in the left hand side of \( r \) and not in its right hand side are removed and agents featured in the the right hand side of \( r \) and not in its left hand side are created.

For instance, the rule:

\[
K(d), S(x_p) \rightarrow K(d), K(d)
\]

deletes an instance of \( S \) and creates an instance of \( K \). For a rigorous definition of kappa’s semantics, see [3].

After having specified an initial mixture and a reaction rate for each rule, it is possible to run stochastic simulations of a model with the kappa simulator [2]. An example of source file for a small kappa model is provided Figure 2.

---

[1] For instance, if a protein has 10 distinct phosphorylation sites, it yields at least \( 1024 \) different species. The situation in real world signalling networks where many proteins can bind each other and form large complexes is even worst.
1.2 From rules to pathways

The aim of causal analysis is to understand better how pathways emerge from a multitude of potential low level protein-protein interactions. A great number of these interactions is known thanks to the progresses of experimental methods but the way they result in complex signalling behaviors remains to be understood. One difficulty in figuring this out comes from the density of the reaction network involved along with its apparent lack of structure. In particular, there are many crosstalks between signalling processes that seem completely unrelated. As a consequence, small signalling models that focus on a limited number of interactions in order to stay tractable for human reasoning usually bring a limited insight. Thus, the development of automated analysis techniques for large models of protein-protein interaction networks appears to be an important step towards a better understanding of signalling pathways.

In this report, we introduce a formal notion of story as an attempt to precise the fuzzy concept of pathway. Figure 3 features an example of a story generated from the model of Figure 2. It describes how a set of rule applications can be chained together to make an observable event happen along with the necessary temporal precedence relation between them. Here, the observable event is the formation of a SoS-Grb2-Shc complex, as mentioned in the last line of Figure 2. For clarity, the agents that are targeted by each rule application are not mentioned and only rule names are displayed.

1.3 Summary

In section 2, we introduce a generic framework for studying event systems and we define the notion of a story. An important class of such event systems for which efficient algorithms can be written is studied in section 3 and kappa models are described within this formalism in section 4. Then, we describe some techniques to produce stories from models, using simulation traces in section 5 and then statically in section 6.

2 A generic framework for event systems

2.1 Event systems

Let $Q$ a set of states, intuitively the set of all the possible states of the world. A subset of $Q$ is called a context. Moreover, an event $e$ is defined by:

- A label denoted $\text{label}(e)$
- A set of states $\text{pre}(e) \subseteq Q$ called precondition
- A map $\text{eff}(e) : \text{pre}(e) \to Q$ called effect
# Agent signatures

%agent: EGF(r)
%agent: EGFR(CR,C,N,L,Y1016~u~p,Y1092~u~p,Y1172~u~p)
%agent: SoS(PR,S~u~p)
%agent: Shc(Y~u~p,PTB~u~p)
%agent: Grb2(SH3c,SH3n,SH2~u~p)

# Constants

%var: 'Avogadro' 6.0221413E+23
%var: 'V' 4.2E-14
%var: 'k_on' 2.5E08 / ('Avogadro' * 'V') # molecule^-1 s^-1
%var: 'k_off' 2.5 # s^-1
%var: 'k_cat' 1 # s^-1

# Initial mixture

%init: 5000 EGF(r), EGFR(CR,C,N,L,Y1016~u,Y1092~u,Y1172~u)
%init: 10000 SoS(PR,S~u) Shc(Y~u,PTB~u)
%init: 5000 Grb2(SH3c,SH3n,SH2~u)

# Rules

'EGFR.EGFR' EGF(r!2), EGFR(L!2,CR,C,N,C), EGF(r!3), EGFR(L!3,CR,N,C) <-> EGF(r!2), EGFR(L!2,CR,C,N,C), EGF(r!3), EGFR(L!3,CR,N,C) @ 'k_on'/2, 'k_off'/2
'EGF.EGFR' EGF(r), EGFR(L,CR) <-> EGF(r!), EGFR(L!,CR) @ 'k_on', 'k_off'
'Shc.Gr2' Shc(Y!p), Grb2(SH2) -> Shc(Y!p!), Grb2(SH2!) @ 5*'k_on'
'Shc/Gr2' Shc(Y!p!), Grb2(SH2!) -> Shc(Y!p), Grb2(SH2) @ 'k_off'
'EGF.Grb2' EGF(Y1092~p), Grb2(SH2) -> EGF(Y1092~p!), Grb2(SH2!) @ 'k_on', 'k_off'
'EGF.Shc' EGF(Y1172~p), Shc(PB) -> EGF(Y1172~p!), Shc(PB!) @ 'k_on'
'EGF/Grb2' EGF(Y1172~p!), Shc(SP) -> EGF(Y1172~p), Shc(SP!) @ 'k_off'
'Grb2/SoS' Grb2(SH3n), SoS(PR,S~u) -> Grb2(SH3n), SoS(PR!,S~u) @ 'k_on'
'Grb2/SoS' Grb2(SH3n!), SoS(PR!) -> Grb2(SH3n), SoS(PR) @ 'k_off'
'EGF.int' EGF(CR!,N,C), EGF(CR!,N,C!) -> EGF(CR!,N,C), EGF(CR!,N,C!) @ 'k_on'
'EGF/int' EGF(CR!,N,C!), EGF(CR!,N,C!2) -> EGF(CR!,N,C!), EGF(CR!,N,C!2) @ 'k_on'
'pY1092EGF' EGF(N!,1), EGF(C11,Y1092~u) -> EGF(N!1), EGF(C11,Y1092~u) @ 'k_cat'
'pY1172EGF' EGF(N!,1), EGF(C11,Y1172~u) -> EGF(N!1), EGF(C11,Y1172~u) @ 'k_cat'
'uY1092EGF' EGF(N!,1), EGF(C11,Y1092~u) -> EGF(N!1), EGF(C11,Y1092~u) @ 'k_cat'
'uY1172EGF' EGF(N!,1), EGF(C11,Y1172~u) -> EGF(N!1), EGF(C11,Y1172~u) @ 'k_cat'
'pYShc' EGF(Y1172~p1), Shc(PB1!,Y~u) -> EGF(Y1172~p1), Shc(PB1!,Y~u) @ 'k_cat'
'uYShc' Shc(Y~u) -> Shc(Y~u) @ 'k_cat'

# The pattern we’re interested in

%obs: 'SoS.Grb2.Shc' Grb2(SH2!,SH3n!,2),SoS(PR!,2),Shc(Y!p!1)

Figure 2: A kappa model of SoS recruitment
Figure 3: A possible scenario for SoS recruitment
The precondition of an event is the set of states from which it can be triggered and its effect maps each of these states to a new one.

An event system is a triple $(Q, E, C)$ with $E$ a set of events over $Q$ and $C \subseteq \mathcal{P}(Q)$ a set of contexts such that for all event $e \in E$, $\text{pre}(e) \in C$ and $\forall c \in C$, $\text{eff}(e)(c) \in C$. All the concepts introduced in section 2 are defined relatively to an event system.

2.2 Traces

A trace is defined as a finite sequence of events. Its precondition is defined by induction on its size by the following equations:

$$\text{pre}(e) = Q$$
$$\text{pre}(e, t) = \{ q \in \text{pre}(e) : \text{eff}(e)(q) \in \text{pre}(t) \}$$

with $e$ an event and $t$ a trace. A trace $t$ is said to be valid in the context $c$ if $c \subseteq \text{pre}(t)$, in which case we write $c \vdash t$. The effect and the postcondition of a trace in context $c$ are defined by:

$$\text{eff}_c(e_1, \ldots, e_n) = \text{eff}(e_n) \circ \ldots \circ \text{eff}(e_1)|_c \quad \text{post}_c(e) = (\text{eff}_c(t))(c)$$

They can be defined in a context-independent way too:

$$\text{eff}(t) = \text{eff}_{\text{pre}(t)}(t) \quad \text{post}(t) = \bigcup_{c \in \text{pre}(t)} \text{post}_c \quad \text{post}_{\text{pre}(t)}(t)$$

2.3 Concurrent events and equivalent traces

The order in which some events happen might not always matter. Let’s take the example of a kappa agent which can be phosphorylated on its both sites in parallel by two different enzymes. Let’s write $p_1$ the event of phosphorylating site 1 and $p_2$ the event of phosphorylating site 2. Then, we may want the traces $(p_1, p_2)$ and $(p_2, p_1)$ to be considered equivalent. Indeed, the definition of a trace forces us to decide on an order between $p_1$ and $p_2$ but this order is nor relevant neither necessary. Worst, in the real world, $p_1$ and $p_2$ could have happened in the same time. This draws our motivation to formalize a concept of concurrent events.

2.3.1 Concurrent events

Two events $e_1$ and $e_2$ are said to be concurrent in context $c$ if the three following assertions are equivalent:

$$c \vdash e_1, e_2 \quad c \vdash e_2, e_1 \quad c \vdash e_1 \land c \vdash e_2$$
and the following is true when they hold:

\[ \text{eff}_c(e_1, e_2) = \text{eff}_c(e_2, e_1) \]

In this case, we write \( e_1 \circ_c e_2 \). Moreover, \( e_1 \) and \( e_2 \) are said to be concurrent if they are in any context \( c \in C \), which we write \( e_1 \diamond e_2 \).

**Remark:** This definition allows events which can’t come close to each other in any context to be considered concurrent, that is events \( e_1 \) and \( e_2 \) such that:

\[ (\not\exists c, c \vdash e_1, e_2) \land (\not\exists c, c \vdash e_2, e_1) \land (\not\exists c, c \vdash e_1 \land c \vdash e_2) \]

This can be counter-intuitive and that’s why we define notions of *non-trivial concurrency*. Two events \( e_1 \) and \( e_2 \) are said to be *non-trivially* concurrent in context \( c \) if \( e_1 \circ_c e_2, c \vdash e_1 \) and \( c \vdash e_2 \). They are *non-trivially* concurrent if they are concurrent in any context, this being non-trivially in at least one of them. This last notion admits a nice algebraic characterization for a class of systems kappa models belongs to, as we see in section 3.3.

### 2.3.2 Equivalent traces

As said before, we want to consider two traces differing only by the permutation of concurrent events to be equivalent. Let \( c \) a context. We define the relation \( \simeq_c \) as the smallest equivalence relation on the set of valid traces in context \( c \) such that:

\[ e_1 \circ_{\text{post}_c(t)} e_2 \implies t \cdot e_1, e_2 \cdot t' \simeq_c t \cdot e_2, e_1 \cdot t' \]

where \( \cdot \) stands for the concatenation operator on traces. This relation is called the Mazurkiewicz equivalence in context \( c \).

We define \( \sim \) as a stronger equivalence relation on traces that is context-independent. It is the smallest relation on traces such that:

\[ e_1 \circ e_2 \implies t \cdot e_1, e_2 \cdot t' \sim t \cdot e_2, e_1 \cdot t' \]

Two equivalent traces for \( \sim \) are said to be *strongly similar*. Here, two elements can be permuted only if they are concurrent in any context and not only in the current one. Note that we have:

\[ t \sim t' \implies t \preceq t' \]

but the converse is not true\(^2\). However, we prove in section 3.4 that for an important subset of kappa models called *regular* models, the relations \( \sim \) and \( \simeq_c \) are identical in any context \( c \).

---

\(^2\) With \( Q = \{0, 1\} \), let’s consider \( t = (c, x, y) \) and \( t' = (c, y, x) \) such that \( \text{pre}(t) = Q, \text{post}(t) = \{0\} \) and \( x \circ_{\{0\}} y \) but not \( x \circ_{\{1\}} y \).
2.4 Configurations

Let $\mathcal{T}$ the set of all traces on $E$. An element of the quotient $(\mathcal{T}/\sim)$ is called a configuration and its traces are said to be its trajectories. For the reasons we explained in section 2.3, it seems more relevant to define our concept of a kappa story on configurations rather than on traces.

2.4.1 Representing configurations

Let $t = e_1, \ldots, e_n$ a trace and $\mathcal{E}_t = \{e_i\}$ the set of its events, each one of them being distinguished so that $|\mathcal{E}_t| = n$. The precedence relation $\preceq_t$ is defined as the smallest order relation over $\mathcal{E}_t$ such that:

$$(i < j \land \neg e_i \diamond e_j) \implies e_i \preceq_t e_j$$

Besides, a permutation $\sigma$ of $\{1, \ldots, n\}$ is said to preserve it when:

$$e_{\sigma(i)} \preceq_t e_{\sigma(j)} \implies \sigma(i) \leq \sigma(j)$$

Then, writing $\sigma(t) := e_{\sigma(1)}, \ldots, e_{\sigma(n)}$ the reordering of $t$ by $\sigma$, the following theorem holds:

**Theorem 1.** Let $t$ and $t'$ two traces. Then $t \sim t'$ if and only if $t' = \sigma(t)$ for $\sigma$ a permutation preserving the precedence relation on $t$.

**Proof.** It is easy to check that two traces differing only in the permutation of two concurrent events have the same precedence relation. By induction and using this fact, we can prove the first direction.

Conversely, let $t$ a trace, $\preceq_t$ its precedence relation and $\sigma$ a reordering preserving it. We will prove that $\sigma(t) \sim t$ by induction on the size of $t$. For this, we show that $e_{\sigma(1)}$ is concurrent with all the events of $t$ from $e_1$ to $e_{\sigma(1)-1}$. Indeed, let’s suppose $\neg (e_i \diamond e_{\sigma(1)})$ for $1 \leq i < \sigma(1)$. Then:

$$e_i \preceq_t e_{\sigma(1)} \quad \text{(by definition of the precedence relation)}$$

$$\therefore e_{\sigma(\sigma^{-1}(i))} \preceq_t e_{\sigma(1)} \quad \text{(because } \sigma(\sigma^{-1}(i)) = i)$$

$$\therefore \sigma^{-1}(i) \leq 1 \quad \text{(because } \sigma \text{ preserves } \preceq_t)$$

Therefore, $\sigma^{-1}(i) \leq 1$ and then $i = \sigma(1)$, which is a contradiction. As a consequence, it is possible to set $e_{\sigma(1)}$ in first position in $t$ by making it commute with all the events from $e_1$ to $e_{\sigma(1)-1}$. The trace obtained – call it $t'$ – is strongly similar to $t$. We conclude by applying the induction hypothesis on $t$ and $t'$ without their first element. \qed

8
As a consequence of this theorem, a configuration is represented by a tuple consisting in a set of events along with an order relation on them. Then, the canonical projection $C : T \rightarrow (T/\sim)$ maps $t$ to $(E_t, \preceq_t)$ as we have indeed:

$$t \sim t' \iff (E_t, \preceq_t) = (E_t', \preceq_t')$$

### 2.4.2 Some operations on configurations

All trajectories of a configuration $(E, \preceq)$ can be recovered by ordering the events of $E$ in a way that preserves $\preceq$. Moreover, as the operators $\text{eff}_c$, $\text{pre}$ and $\text{post}_c$ are preserved by strong similarity, that is:

$$\text{pre}(t) = \text{pre}(t') \quad \text{eff}_c(t) = \text{eff}_c(t') \quad \text{post}_c(t) = \text{post}_c(t')$$

for any context $c$ and $(t, t')$ two strongly similar traces, they can also be defined on a configuration from any of its trajectory. As a consequence, a context being given, we can talk about valid configurations too. Finally, we say that $(E, \preceq)$ is a sub-configuration of $(E', \preceq')$ when $E \subseteq E'$ and $(\preceq) \subseteq (\preceq')$.

### 2.5 Stories

Let $\Omega$ a subset of interest of $E$ called set of observable events and $c$ a context. Then, a story reaching $\Omega$ from $c$ is defined as a configuration containing an event of $\Omega$ which is valid in $c$. We write $S(\Omega, c)$ the set of all such stories.

For instance, in kappa, we would take $c = \{I\}$, where $I$ is the initial mixture and $\Omega$ would contain all the possible instanciations of the obs rule. Moreover, we would have $S(\Omega, c) \neq \emptyset$ if and only if obs is reachable from $I$. In this case, a story can be considered as a proof of reachability. However, although tempting, this way of considering stories as proofs is misleading as it takes us away from the spirit of biological pathways stories are aimed to formalize.

#### 2.5.1 Stories : proofs or explanations ?

Indeed, when biologists study pathways, the reachability of the observable is usually taken for granted and what really matters is the question of how this process is implemented. For instance, when designing drugs, it may be of particular interest to identify a step which is common to all the scenarios leading to the realization of a biological process.

This example is in opposition with the principle of proof-irrelevance most logicians work with which stipulates that all the proofs of a fact should be considered equivalent. Clearly, we don’t want to formalize pathways in this spirit. The question then is the following: if all elements of $S(\Omega, c)$ should not

---

3 That is, $E \subseteq E'$ and $\forall x, y \in E, x \preceq y \implies x \preceq y'$. 

9
be considered equivalent, should they be considered in contrast all different and equally relevant? It seems the answer here is still no. Indeed, let’s consider the following examples:

1. Some stories may contain some events which make no contribution in producing the observable. We might want to eliminate these.

2. In kappa, two stories might feature the phosphorylation of the same agent by two different but isomorphic enzymes. Should they be considered different?

This motivates us to introduce some compression techniques along with some equivalence relations on \( S(\Omega, c) \).

### 2.5.2 Sub-stories and weak compression

\( S(\Omega, c) \) can be equipped with a relation \(<_{\Omega, c}\) such that \( s <_{\Omega, c} s' \) if and only if \( s \) is a sub-configuration of \( s' \). In this case, we say that \( s \) is a sub-story of \( s' \). When there is no ambiguity about which observables and context are referred to, we sometimes just write \( s < s' \) but \(<\) should not be confused with the sub-configuration relation as there is the implicit requirement that its operands are in \( S(\Omega, c) \). Finally, a story is said to be minimal if it has no strict sub-story.

In order to remove from a story some events that are not essential in producing the observable, a natural idea is to consider instead one of its minimal sub-stories. Such a technique is implemented in \texttt{KaSim} and is called weak-compression. Details about this algorithm are given in section 5.2. However, a stronger notion of compression is sometimes needed, as we illustrate now with an example.

### 2.5.3 The example of double phosphorylation

Let’s consider the kappa model introduced in Figure 1 with the initial mixture \( K(d), K(d), S(x_u, y_u) \) and let’s take for an observable rule the rule \texttt{obs} which tests the pattern \( S(x_p, y_p) \) and does nothing. Then, here is an example of a story \( s_0 \) where the first enzyme binds the substrate, phosphorylates its two sites and then unbinds:

\[
\begin{align*}
&b_1 &\xleftarrow{px_1} &u_1 &\xrightarrow{\texttt{obs}} \\
& &\xleftarrow{py_1} &
\end{align*}
\]

Note that such a story is not unique in general.
It is useful to see a story as a graph whose nodes are events and whose edges correspond to the precedence relation over them. For clarity, all the edges are not represented here: only a transitive reduction of the graph is displayed.

Moreover, as explained in section 4, a kappa event consists in a rule along with an instanciation map giving the identifiers of the agents it targets. Here, an event is labelled $r_i$ where $r$ is the name of the rule applied, and $i \in \{1, 2\}$ gives the identifier of the enzyme involved. There is no need to specify which substrate is targeted as there is only one in the mixture. Finally $s_0$ illustrates the fact that $px_1$ and $py_1$ are concurrent. Here are two other stories about the same phenomenon:

$s_1 : b_1 \rightarrow px_1 \rightarrow u_1 \rightarrow b_1 \rightarrow py_1 \rightarrow u_1 \rightarrow obs$

$s_2 : b_1 \rightarrow px_1 \rightarrow u_1 \rightarrow b_2 \rightarrow py_2 \rightarrow u_2 \rightarrow obs$

In $s_1$, the first enzyme unbinds and rebinds the substrate after it has phosphorylated its first site and before it phosphorylates its second. Remark that if a story is seen as a graph whose edges correspond to the precedence relation and $s$ and $s'$ are two valid stories, we have $s \ll s'$ if and only if the graph of $s$ can be embedded in the graph of $s'$. Here, it can be noticed that $s_0 \ll s_1$. Indeed, we can get $s_0$ from $s_1$ by compressing the $(u_1, b_1)$ pattern in it.

In $s_2$, after the first enzyme has phosphorylated the site $x$ of the substrate, it unbinds and the second enzyme binds to phosphorylate $y$. Contrary to $s_1$, $s_2$ is minimal and cannot be compressed further. However, one may argue that whether kinase 1 or 2 is used to phosphorylate the substrate does not really matter. If we abstract away this information, stories $s_1$ and $s_2$ turn out to be the same and can be represented as follows, where nodes are not labelled by events anymore but by rules:

$$b \rightarrow px \rightarrow u \rightarrow b \rightarrow py \rightarrow u \rightarrow obs$$

This graph can be considered as an abstract story. Similarly, $s_0$ can be abstracted into:

$$b \quad px \quad u \quad obs$$

$$b \quad py \quad u \quad obs$$

---

5 A transitive reduction of a directed graph is a graph with as few edges as possible that has the same reachability relation as the given graph.

6 An embedding of a graph $G = (V, E)$ in another graph $G' = (V', E')$ is a injective graph-morphism $\varphi : G \rightarrow G'$, that is a map from $V$ to $V'$ such that $x$ and $\varphi(x)$ have the same label for all $x$ and $\forall x, y, (x, y) \in E \implies (\varphi(x), \varphi(y)) \in E'$. 

11
As the latter is embedded in the former, we say that $s_0$ is a strongly compressed form of both $s_1$ and $s_2$. In section 2.5.4, we formalize this notion of an abstract story and introduce strong compression, as implemented in KaSim.

### 2.5.4 Abstract stories and compression

An abstraction over $S(\Omega, c)$ consists in:

- An abstract domain $A$. In the last example, it would be the set of all directed graphs whose nodes are labelled by rules, which is roughly the choice made by the current version of the kappa simulator.

- An abstraction map $\alpha : S(\Omega, c) \to A$. In the last example, $\alpha$ abstracts away the distinction between the two kinases present in the mixture.

An abstraction $(A, \alpha)$ yields an equivalence relation $\sim_\alpha$ over stories defined by:

\[ s \sim_\alpha s' \iff \alpha(s) = \alpha(s') \]

In the last example, $s_1$ and $s_2$ would be equivalent in this sense. Moreover, a common pattern is to take $A$ as a set of labelled directed graphs. Then, it is possible to introduce a notion of compression modulo $\alpha$ as a relation $\rightarrow_\alpha$ over stories defined by:

\[ s \rightarrow_\alpha s' \iff \alpha(s') \text{ is embedded into } \alpha(s) \]

In this context, a story is said to be minimal if it can’t be compressed into a shorter one. In the last example, we would have $s_1 \rightarrow_\alpha s_0$ and $s_2 \rightarrow_\alpha s_0$, $s_0$ being minimal.

The notion of strong compression as implemented in the kappa simulator corresponds to compression modulo $\alpha$ where $\alpha$ abstracts away the identity of the agents targeted by an event. More precisely, $\alpha$ maps a story to a graph with one node for each of its events whose edges correspond to the precedence relation on them. However, the node associated to an event $e$ is not labelled with $e$ itself, in which case we would get weak compression, but by the rule it is an instantiation of.

Note that many other relevant notions of compression can be defined with this formalism. For instance, any equivalence relation being given over the set $E$ of possible events, we could label each node of an abstract story with the equivalence class of the event it corresponds to. In particular, we could define a notion of filtered compression where, compared to strong compression, only the identity of the agents of some type would be abstracted away.

Finally, it is possible to abstract away the precedence order over the events of a story and to use another relation to define the edges of its abstract counterpart. This will be very useful in Section 6 when generating stories statically.
3 Grid-based systems

In this section, we study an important class of event systems we call grid-based. Most concepts defined previously admit a nice algebraic characterization in this setting and efficient algorithms can be provided that do not necessarily exist in general. An event system \((Q, E, C)\) is said to be grid-based if:

- A state is given by the values of some state variables. That is, there exists a set of variables \(X\) and for each variable \(x \in X\) a set \(V_x\) of values it can take such that \(Q = \prod_{x \in X} V_x\).
- All contexts of \(C\) along with the sets \(\text{pre}(e)\) and \(\text{eff}(e)\) for \(e \in E\) can be encoded as partial valuations of \(X\).

Before we provide a rigorous definition, we have to discuss this concept of a partial valuation.

3.1 Partial valuations

A partial valuation of \(X\) is a set of traits over \(X\), a trait being an element of:

\[
\bigcup_{x \in X} \{x\} \times V_x
\]

A partial valuation \(\varphi\) can be interpreted both as a set of equality tests, a trait \((x, v)\) standing for the constraint \(x = v\), or as a set of assignments, \((x, v)\) standing for \(x := v\). In the first case, it denotes the subset of \(Q\):

\[
[\varphi] = \{q \in Q : \forall (x, v) \in \text{val}, q_x = v\}
\]

Finally, we provide the following operators for manipulating partial valuations:

- If \(\varphi\) is a partial valuation, we write \(\overline{\varphi}\) its support, that is the set \(\{x : (x, v) \in \varphi\}\) of the variables constrained in \(\varphi\).

- A partial valuation is said to be contradictory if it contains two contradictory constraints, that is both \((x, v)\) and \((x, v')\) for \(v \neq v'\).

- If \(\varphi_1 \supseteq \varphi_2\), we say that \(\varphi_1\) implies \(\varphi_2\). Beware that:

\[
\varphi_1 \supseteq \varphi_2 \iff [\varphi_1] \subseteq [\varphi_2]
\]

- \(\varphi_1 \cup \varphi_2\) is the concatenation of \(\varphi_1\) and \(\varphi_2\). Beware that:

\[
[\varphi_1 \cup \varphi_2] = [\varphi_1] \cap [\varphi_2]
\]

- \(\varphi_1\) and \(\varphi_2\) are said to be compatible if their concatenation is not contradictory. Then, we write \(\varphi_1 \uparrow \varphi_2\).
• They are said to be *incompatible* otherwise, in which case we write \( \varphi_1 \downarrow \varphi_2 \).

• If \( \psi \) is a set of variables, then \( \varphi \setminus \psi \) is defined as \( \{ (x, v) : x \notin \psi \} \). In particular, \( \varphi_1 \setminus \varphi_2 \) is the set of constraints in \( \varphi_1 \) which involve variables that are unconstrained in \( \varphi_2 \).

• Finally, we define the *update* of \( \varphi_1 \) by \( \varphi_2 \) as:

\[
\varphi_1 \uparrow \varphi_2 = \varphi_2 \cup (\varphi_1 \setminus \varphi_2)
\]

We write \( \text{pval}(\mathcal{X}) \) the set of all non-contradictory partial valuations of \( \mathcal{X} \). We’re now able to define grid-based systems rigorously.

### 3.2 Definition of a grid-based system

An event system \((Q, E, C)\) is said to be *grid-based* if there exists a set of variables \( \mathcal{X} \) and for each \( x \in V_x \) a set of values \( V_x \) such that:

\[
Q = \prod_{x \in \mathcal{X}} V_x \quad C = \{ [c] : c \in \text{pval}(\mathcal{X}) \}
\]

Moreover, for each event \( e \in E \), there has to exist \( \theta, \mu \in \text{pval}(\mathcal{X}) \) such that:

- \( \text{pre}(e) = [\theta] \)
- \( \text{post}_{[c]}(e) = [c \uparrow \mu] \) for any partial valuation \( c \)

In the case there is no ambiguity, \( \theta \) and \( \mu \) are written respectively \( \text{pre}(e) \) and \( \text{eff}(e) \), overloading the previous notation. Then, we write:

\[
\text{post}(e) = \text{pre}(e) \uparrow \text{eff}(e)
\]

Moreover, contexts are identified with partial valuations. Finally, the following convention has to be followed so the forthcoming theorems are true:

**Convention 1.** For any event \( e \), \( \text{pre}(e) \cap \text{eff}(e) = \emptyset \).

This means that if an event has the test \( x = v \) in its precondition, the assignment \( x := v \) can’t belong to \( \text{eff}(e) \). It is quite natural as such an assignment, if present, would be useless.

### 3.3 An algebraic characterization of concurrency

Using the formalism of partial valuations, we can state and prove an algebraic characterization of concurrency.
Theorem 2. Let $e_1$ and $e_2$ two events in a grid-based system. Then, $e_1$ and $e_2$ are non-trivially concurrent if and only if the following conditions hold:

- $\text{pre}(e_1) \cap \text{eff}(e_2) = \emptyset$
- $\text{pre}(e_2) \cap \text{eff}(e_1) = \emptyset$
- $\text{pre}(e_1) \uparrow \text{pre}(e_2)$
- $\text{eff}(e_1) \uparrow \text{eff}(e_2)$

Proof. Let’s suppose the conditions $a.$ to $d.$ hold and take a context $c$.

- Let’s prove that:
  
  \[(c \vdash e_1 \land c \vdash e_2) \implies (c \vdash e_1, e_2 \land c \vdash e_2, e_1 \land \text{eff}_c(e_1) = \text{eff}_c(e_2))\]

  Suppose that $c \vdash e_1$ and $c \vdash e_2$. In particular, we have $c \vdash e_2$ and then $c \supseteq \text{pre}(e_2)$. Combining this with $b.$ it yields $c \not\vdash \text{eff}(e_1) \supseteq \text{pre}(e_2)$ which implies $c \vdash e_1, e_2$. We can show $c \vdash e_2, e_1$ the same way using $a$. Finally, the fact that $\text{eff}_c(e_1) = \text{eff}_c(e_2)$ is a direct consequence of $d$.

- Then, let’s prove that:
  
  \[(c \vdash e_1, e_2) \implies (c \vdash e_1 \land c \vdash e_2)\]

  Suppose $c \vdash e_1, e_2$. Then, $c \vdash e_1$ holds obviously. Moreover, as $c \not\vdash \text{eff}(e_1) \supseteq \text{pre}(e_2)$, using $b.$ we get $c \supseteq \text{pre}(e_2)$ and so $c \vdash e_2$.

- Symmetrically, we can show that $(c \vdash e_2, e_1) \implies (c \vdash e_1 \land c \vdash e_2)$.

Therefore, $e_1$ and $e_2$ are concurrent. They are non-trivially concurrent in the context $\text{pre}(e_1) \cup \text{pre}(e_2)$, which is coherent thanks to $c$.

Conversely, let’s suppose that $e_1$ and $e_2$ are non-trivially concurrent. Then:

- Property $c.$ is true thanks to the non-triviality hypothesis.

- Property $d.$ is true because in a context $c$ where $c \vdash e_1$ and $c \vdash e_2$, whose existence is guaranteed by the previous point, $\text{eff}_c(e_1) = \text{eff}_c(e_2)$.

- Property $a.$ is true. Indeed, Let’s suppose $\overline{\text{pre}(e_1) \cap \text{eff}(e_2)} \neq \emptyset$. Then, there are two possible cases:

  - $(x, v) \in \text{pre}(e_1)$ and $(x, v) \in \text{eff}(e_2)$. In this case, let’s take a context $c$ such that $c \vdash e_2, e_1$. Such a context exists thanks to the non-triviality hypothesis. Then, let’s consider $c' = c \setminus \{x\} \cup \{(x, v')\}$ where $v' = \hat{v}$ if $(x, \hat{v}) \in \text{pre}(e_2)$ or any value different from $v$. Otherwise, then, $c' \vdash e_2, e_1$ but $c' \not\vdash e_1$, which is a contradiction.\(^7\)

\(^7\) Here, we use the facts that $(x, v) \not\in \text{pre}(e_2)$ by convention\(^1\) and that each variable can take at least two different values, so we can choose $v' \neq v$. 

15
This terminates the proof. \(\square\)

**Proposition 1.** Two events \(e_1\) and \(e_2\) are trivially concurrent if and only if the following properties hold:

\[
\begin{align*}
\text{pre}(e_1) & \downarrow \text{pre}(e_2) & \text{post}(e_1) & \downarrow \text{pre}(e_2) & \text{post}(e_2) & \downarrow \text{pre}(e_1)
\end{align*}
\]

**Proof.** The most difficult thing to prove here is that:

\[
\exists c, \ c \vdash e_1, e_2 \iff \text{post}(e_1) \uparrow \text{pre}(e_2)
\]

Suppose that \(\text{post}(e_1) \uparrow \text{pre}(e_2)\). Then:

\[
\begin{align*}
(\text{eff}(e_1) \cup \text{pre}(e_1) \setminus \text{eff}(e_1)) & \uparrow \text{pre}(e_2) \\
\therefore (\text{pre}(e_1) \setminus \text{eff}(e_1)) & \uparrow \text{pre}(e_2) \\
\therefore \text{pre}(e_1) & \uparrow (\text{pre}(e_2) \setminus \text{eff}(e_1))
\end{align*}
\]

and the context:

\[
c = \text{pre}(e_1) \cup (\text{pre}(e_2) \setminus \text{eff}(e_1))
\]

is coherent. In this context, \(c \vdash e_1, e_2\). The converse is easy. \(\square\)

### 3.4 Regular systems

A system is said to be regular if \(\text{eff}(e) \subseteq \text{pre}(e)\) for any event \(e\). That is, every variable which is modified by an event has to be tested first. As we will see, kappa programs are not regular in general but it can be argued that models which are biologically relevant are\(^8\) For instance, KaSim emits a warning when encountering a rule like \(A(x) \rightarrow A(x_p)\) although it’s perfectly valid kappa. A remarkable property of regular systems is the following:

**Theorem 3.** In a regular system, the strong similarity relation \(\sim\) is identical to the Mazurkieviescz equivalence relation \(\simeq\) in any context \(c\).

**Proof.** All comes down to the fact that for all context \(c\), if \(e_1\) and \(e_2\) are non-trivially concurrent in context \(c\), then they are non-trivially concurrent. Indeed, let \(c\) such a context.

- \(\text{pre}(e_1) \uparrow \text{pre}(e_2)\) as \(c \vdash e_1\) and \(c \vdash e_2\)

---

\(^8\) One reason for this is that the kinetics of a chemical reaction modifying a residue should depend on its previous state. This is especially true in the particular case where the residue is already in the state the rule would set it to.
• \( \text{eff}(e_1) \uparrow \text{eff}(e_2) \) as \( \text{eff}_c(e_1, e_2) = \text{eff}_c(e_2, e_1) \)

• \( \text{pre}(e_1) \cap \text{eff}(e_2) = \emptyset \). Indeed, if this is wrong, two cases are possible:
  
  – \((x, v) \in \text{pre}(e_1) \) and \((x, v) \in \text{eff}(e_2)\). As we are working in a regular system, \(x \in \text{pre}(e_2)\). Moreover, \((x, v) \notin \text{pre}(e_2)\) by convention \[ \] Therefore, \((x, v') \in \text{pre}(e_2)\) for \(v \neq v'\), which contradicts the fact that \(c \vdash e_1 \land c \vdash e_2\).
  
  – \((x, v) \in \text{pre}(e_1) \) and \((x, v') \in \text{eff}(e_2)\) for \(v \neq v'\). This contradicts the fact that \(c \vdash e_2, e_1\).

\( \text{pre}(e_1) \cap \text{eff}(e_2) = \emptyset \) can be shown symmetrically.

By Theorem 2, this terminates the proof. \[\square\]

4 The case of kappa

In this section, we consider a kappa model and show how it is possible to build a grid-based event system \((Q, E, C)\) from it.

4.1 States

Before we define events, we have to decide upon a set of states of the world \(Q\). It is natural to think about \(Q\) as the set of all mixtures whose agents respect their type signature and are given unique identifiers. However, in order to fit the formalism of grid-based systems, such mixtures have to be encoded in a series of state variables.

We introduce a set \(I\) of agent identifiers. For practical reasons, we want to be able to extract the type of an agent from its identifier through a map:

\[ \tau : I \rightarrow T \]

where \(T\) stands for the set of agent types. Therefore, a natural choice for \(I\) is \(T \times \mathbb{N}\) but other encodings are possible. We call \text{port} a tuple \((u, x)\) where \(u\) is an agent identifier and \(x\) the name of a site which is present in the signature of \(\tau(u)\). Finally, the state of the agents of a mixture is described by the following set of variables:

• For each agent identifier \(u\), a variable \(u_{\exists}\) is introduced which is \text{true} if \(u\) is in the mixture, \text{false} if it was once but was deleted, and \text{null} otherwise. It is important to distinguish between \text{false} and \text{null} so that an event can’t give an identifier that had been used in the past to a newly created agent.
• For each port $x$, a variable $x_\iota$ is introduced, which describes its internal state. For instance, it might be $p$ for phosphorylated or $u$ for unphosphorylated. It is $\text{null}$ when the agent $x$ belongs to is not in the mixture.

• For each port $x$, a variable $x_\lambda$ is introduced which describes its linking state. The latter can take three types of values: $\text{free}$ if $x$ is not bound, $\text{bound}(y)$ where $y$ is a port, or $\text{null}$ if the agent $x$ belongs to is not in the mixture.

For $J \subseteq I$, we write $\mathcal{X}_J$ the set of all these variables for the agent identifiers of $J$. Then, we define $Q$ as the set of valuations over $\mathcal{X}_I$. Despite the fact that $I$ and so $\mathcal{X}_I$ are typically infinite, any valuation describing a kappa mixture has only a finite number of non-null values.

Note that this definition of $Q$ is restrictive as it implies that the precondition of any event can only be based on the state of the current mixture. We could have chosen to encode in $Q$ some informations about past events. For instance, in such a theory, it would have been possible to define events in such a way that any valid story does not feature a given pattern. On the other hand, we wouldn’t have had a grid-based system anymore and writing algorithms for it would have been much harder.

### 4.2 Rules and events

It is easier to define the notion of an event and to understand its relation with the notion of a kappa rule for models without side effects. Intuitively, a kappa rule is said to have a side effect when it can modify agents it does not mention explicitly. There are two ways of introducing side effects in kappa:

1. **By removing a semi-link.** Indeed, it is possible to write rules like:

   $$ r_1 : A(x^\neg) \rightarrow A(x) $$

   Such a rule tests the existence of a link between the site $x$ of an agent of type $A$ and any other agent and then removes it. Therefore, it affects the agent $A$ is bound to although not mentioned explicitly.

2. **By removing an agent which is not fully specified.** Consider the rule:

   $$ r_2 : A() \rightarrow $$

   Applied to the mixture $A(x^1), B(x^1)$, it results into the mixture $B(x)$, where $B$ has been modified although it is not mentioned explicitly in $r_2$.

Many algorithms written for kappa including the simulation algorithm and the algorithm introduced in section\[\[\] to generate stories statically have problems dealing with side effects. Hopefully, every kappa model admits a finite equivalent form without any of these.
4.2.1 Removing side effects

Let’s consider the following example where \( p \) has a side effect:

\[
\begin{align*}
\text{init} &: \quad \to S(d_u), K(x, y) \\
\text{bx} &: \quad S(d), K(x) \to S(d^1), K(x^1) \\
\text{by} &: \quad S(d), K(y) \to S(d^1), K(y^1) \\
p &: \quad S(d_{\neg u}) \to S(d_p)
\end{align*}
\]

By noticing that the site \( d \) of \( S \) can only be bound to the sites \( x \) or \( y \) of a \( K \) agent, it is possible to replace \( p \) by the following two rules:

\[
\begin{align*}
p_1 &: \quad S(d^1_u), K(x^1) \to S(d_p), K(x) \\
p_2 &: \quad S(d^1_u), K(y^1) \to S(d_p), K(y)
\end{align*}
\]

This leads to an equivalent model freed of any side effect. More generally, in order to remove the side effects of a model, we use its contact map to refine all the rules featuring semi-links or underspecified deleted agents. The contact map of a model is a graph whose nodes are port types (a port type is a tuple \( \langle A, x \rangle \) where \( A \) is an agent type and \( x \) a site name), an edge connecting two port types if and only if a bond appears between them in the right hand side of a rule. The way to deal with rate constants when refining rules is detailed in [4].

If \( m \) is the maximum arity of the contact map, a rule whose left hand side features \( k \) semi-links can be refined into at most \( m^k B_k \) rules without any semi-links where \( B_k \) is the \( k^{th} \) Bell number, that is the number of ways of partitioning a set of \( k \) elements. A reasonable bound for this is:

\[ m^k B_k \leq (mk)^k \]

Similarly, a rule deleting an agent with \( k \) unspecified site can be refined into less than \( (m + 1)^k B_k \) rules. However, this bounds are rarely reached in practice and removing the side effects of a model is tractable in most cases.

4.2.2 Instanciating rules into events

From now on, we consider only kappa models without side effects. Then, a rule \( r \) is given by:

- A set of local agent identifiers \( \text{ag}(r) \subseteq I \)
- Two partial valuations \( \text{pre}(r) \) and \( \text{eff}(r) \) of \( X_{\text{ag}(r)} \)
\[ K(d_1), S(d_1, x_u) \rightarrow K(d_1), S(d_1, x_p) \]

**Precondition**

- \( k \exists = \text{true} \)
- \( s \exists = \text{true} \)
- \( \langle k, d \rangle_\lambda = \text{bound}(s, d) \)
- \( \langle s, d \rangle_\lambda = \text{bound}(k, d) \)
- \( \langle s, x \rangle_\lambda = \text{free} \)
- \( \langle s, x \rangle_\iota = u \)

**Effect**

- \( \langle s, x \rangle_\iota := p \)

An example of rule encoding is given Figure 4. An event is defined as a tuple \( \langle r, \varphi \rangle \) where \( r \) is a rule and \( \varphi : \text{ag} r \rightarrow I \) an instantiation map. Note that \( \varphi \) has to be injective and to preserve the type of agents:

\[
\forall a \in \text{ag}(r), \tau(\varphi(a)) = \tau(a)
\]

If \( v \) is a partial valuation of \( \text{ag}(r) \), we write \( \varphi(v) \) the partial valuation of \( I \) we get after renaming each agent identifier in it with \( \varphi \). Then, it is possible to define the effect and the precondition of an event as:

\[
\text{pre}(\langle r, \varphi \rangle) = \varphi(\text{pre}(r)) \quad \text{eff}(\langle r, \varphi \rangle) = \varphi(\text{eff}(r))
\]

Finally, we discuss a feature of kappa that makes it possible to encode mechanisms through kinetic rate constants and that is especially tricky to handle in a satisfactory way for most static analysis techniques.

### 4.2.3 Differential reaction rates

The kappa language allows the user to specify two reaction rates for a single rule in order to handle in a different way the unimolecular and the bimolecular case. For instance, if \( S \) is a substrate and \( K \) a kinase, it is possible to model a phosphorylation rule as:

\[ K(\ ), S(x_u) \rightarrow K(\ ), S(x_p) @ 0 (\gamma) \]
As the bimolecular reaction rate is set to zero, $K$ and $S$ have to be in the same complex so the reaction can take place. This kind of rule can be handled in three different ways:

1. We can ignore the differential reaction rate and handle it as if the bimolecular reaction was possible. However, it will likely make causal analysis irrelevant as the steps of binding and unbinding between the kinase and the substrate will be seen as unnecessary and not appear in the compressed stories.

2. Events instanciating the rule $r$ test the existence of one particular path between $K$ and $S$.

3. Events instanciating the rule $r$ test the existence of any path between $K$ and $S$.

Let’s try to understand the differences between 2. and 3. on an example. Suppose we have to describe as an event $e$ a simulation transition where $r$ is applied to the mixture $K(d^1), S(d^1, x_u)$. Then, we have two possible preconditions for $e$:

<table>
<thead>
<tr>
<th>$e_2$</th>
<th>$e_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precondition</td>
<td>Precondition</td>
</tr>
<tr>
<td>$(s, x)_ι = u$ and $(k, d)$ is bound to $(k, d)$</td>
<td>$(s, x)_ι = u$ and there exists a path from $k$ to $s$</td>
</tr>
</tbody>
</table>

Let’s suppose we choose $e_2$ over $e_3$. It has the advantage that its precondition can be encoded as a partial valuation of $X$ and so our system stays grid-based with no added theoretical complexity. That being said, this solution is far from being perfect. Indeed:

- Suppose we want to trigger our event on the different mixture:
  
  $K(d^1), C(k^1, s^2), S(d^2, x_u)$

  where $C$ could be a scaffold. Here, $e_2$ cannot be triggered anymore but $e_3$ still can. As a consequence, a story featuring $e_3$ may be more compressible than a story featuring $e_2$.

- Suppose we have to translate into an event the simulation transition which triggers rule $r$ on a mixture with an enzyme, a kinase and a scaffold, each one of them being bound to the others. Here, there exist two different paths from the enzyme to the substrate: one which is direct and the other which goes through the scaffold. Which one should we encode in the event? In general, there is no canonical choice. We could pick one randomly or use some heuristics but this raises some elegance issues.

---

9 That is, there has to exist a path a links connecting $K$ and $S$. 

21
Finally, $e_3$ is not expressible in the setting of a grid-based system and it could lead to strong algorithmic overheads. Therefore, we will ignore differential reaction rates in the general theory and handle it case by case when it appears to be important in a particular algorithm.

4.3 Some conventions

We have seen how a kappa model can be translated into a grid-based system. If the latter is regular, we say the same of the model it comes from. For convenience, we take the initial conditions of a model into account by introducing an \texttt{init} rule which creates the initial mixture from scratch. Then, a trace or a configuration is said to be \textit{valid} if it is in the context $\{\epsilon\}$ with $\epsilon$ the empty mixture where all variables are set to null. As a consequence, traces usually begin with an event instanciating \texttt{init}. Finally, a trait $(x, v)$ is said to be \textit{non-null} if and only if $v \neq \text{null}$.

Now, we’re ready to study two very different strategies to generate stories, that is \textit{dynamically} in section 5 and then \textit{statically} in section 6. The algorithms we introduce are designed to work on regular models but they can be applied to non-regular ones at the price of losing some guarantees on their output. Thus, all properties requiring the regularity hypothesis will be mentioned as such.

5 Generating stories dynamically

In this section, we give a brief overview of the techniques currently in use in the kappa simulator to generate stories from simulation traces.

For each instance of the observable in the simulator output, we could turn the trace leading to it into a story by computing its precedence relation using Theorem 2 and then use some compression algorithms. However, simulation traces are typically very long, most of their events being likely to have absolutely nothing to do with the observable, and compression is expensive. Therefore, the simulation trace is filtered in such a way only the \textit{contributive lineage} of the observable remains before it is compressed and turned into a story. Note that the compression algorithms are implemented at the level of traces for convenience and efficiency but they are guaranteed to preserve the $\sim$ relation in the sense that:

$$t \sim t' \implies \text{compress}(t) \sim \text{compress}(t')$$

for regular kappa models. In sections 5.1 and 5.2 we define the concept of contributive lineage rigorously and show how the problem of weak-compression can be reduced to \textsc{minsat}.
5.1 Contributive lineage

Let $t = e_0, \ldots, e_n$ a valid trace. For $i < j$, we say that $e_i$ directly contributes to $e_j$ if $e_j$ tests a variable which is lastly modified by $e_i$, that is:

$$
\left( \text{eff}(e_i) \cap \text{pre}(e_j) \right) \setminus \bigcup_{i<k<j} \text{eff}(e_k) \neq \emptyset
$$

In this case, we write $e_i \rightarrow_t e_j$. The reflexive and transitive closure of $\rightarrow_t$ is written $\rightarrow_t^+$. If $e \rightarrow_t^+ e'$, we say that $e$ contributes to $e'$. The contributive lineage of an event $e$ in $t$ is defined as the set $\{e_i : e_i \rightarrow_t^+ e\}$. Then, the following proposition holds:

**Proposition 2.** If $t$ is a valid trace, the subtrace of $t$ whose events are those of the contributive lineage of one of its events is still valid.

For regular models, the $\rightarrow_t$ relation is preserved by the $\sim$ equivalence on traces and it can be defined on the events of a configuration. Then, it is interesting in its own right as it can be interpreted as a weak notion of causality. The following property gives some additional legitimacy to this point of view:

**Proposition 3.** Let $(Q, E, C)$ a grid-based system. Then, there exists $E' \supseteq E$ such that $(Q, E', C)$ is a grid-based system and the following three properties are equivalent for $e_1$ and $e_2$ two events:

1. There exists a trace $t$ over $E'$ such that $e_1 \rightarrow_t e_2$
2. There exists a context $c \in C$ and a trace $t$ over $E'$ such that:
   $$
c \vdash e_1 \cdot t \cdot e_2 \land c \vdash t \land c \not\vdash t \cdot e_2
   $$
3. $\text{eff}(e_1) \cap \text{pre}(e_2) \neq \emptyset$

If these hold, we say that $e_1$ is a possible cause of $e_2$.

5.2 The weak compression algorithm

Let $t = e_0, \ldots, e_n$ a valid trace, $e_0$ being an instance of the init rule and $e_n$ an instance of the observable. We want to find a minimal subtrace of $t$ which contains $e_n$ and is still valid. This problem is reducible to the MINSAT problem of finding a valuation satisfying a boolean formula with a minimal number of positive variables.

Indeed, let’s consider some boolean variables $k_0, \ldots, k_n$, each $k_i$ being interpreted as "we keep $e_i$ in the trace". For each boolean $k_i$ and variable $x$ such that $(x, v) \in \text{pre}(e_i)$ and $v \neq \text{null}$, we introduce the clause $C_{i,x,v}$ defined as follows:

$$
k_i \iff \bigvee_{j \in A} (k_j \land \bigwedge_{t \in B_j} \neg k_t)
$$

where:
• \( A = \{ j : i < j \land (x,v) \in \text{eff}(e_j) \} \)

• \( B_j = \{ l : i < l < j \land (x,v') \in \text{eff}(e_l), v' \neq v \} \) for each \( j \in A \)

Each valuation of the \( k_i \) defines a subtrace of \( t \). This trace is valid and contains \( e_n \) if and only if:

\[
(k_0 \land k_n) \land \bigwedge_i \bigwedge_{(x,v) \in \text{pre}(e_i)} C_{i,x,v}
\]

Therefore, compressing \( t \) comes down to finding a valuation of this formula with a minimal number of positive variables, which is an instance of \textsc{minsat}. An example is provided Figure 5. Conversely, weak compression is a \( \text{NP} \)-hard problem.

6 Generating stories statically

We introduce in this section an algorithm to compute stories statically, that is without running any simulation. Like most static analysis techniques, it abstracts away the kinetics properties of a model and focuses on its mechanisms. As a consequence, many stories it produces are very unlikely to appear in a simulation trace.

This behaviour is interesting in some situations where biocurators care about sleeping secondary pathways that become more active when a perturbation is introduced in the model. Producing unlikely stories might help spotting such perturbations. In other situations where only frequent stories are relevant, it is still possible to run simulations in order to annotate statically generated stories with frequency informations.

Besides, abstracting away kinetics gives kappa much smoother semantics properties that make reasoning over underspecified models easier. For instance, adding a rule to a model cannot discard a story although it can critically decrease its frequency by introducing competition effects. More generally, purely mechanistic properties are more robust regarding model composition. Therefore, static analysis techniques are especially relevant in the context of assembling models from nuggets of knowledge mined from litterature, which is a critical step towards making rule-based modelling widely used.

Finally, the techniques we introduce in this section can be easily adapted to the problem of finding stories respecting a user-defined property and they are best suited for interactive reasoning, as there are many ways a human agent can assist the computer in its exploration of the search space.

\textsuperscript{10} Indeed, it is very hard to take the rate constants of a model into account in a nontrivial way without running a stochastic simulation.
Let’s consider a model made of a single agent $S$ with two phosphorylation sites $x$ and $y$ and three rules acting on it. It is possible to represent the four possible states of the system along with the transition corresponding to each rule on a square diagram:

\[
\begin{align*}
px &: S(x_u) \rightarrow S(x_p) & (u, p) \rightarrow (p, p) \\
py &: S(x_u, y_u) \rightarrow S(x_u, y_p) \\
uy &: S(x_p, y_p) \rightarrow S(x_p, y_u)
\end{align*}
\]

Suppose the initial state is $S(x_u, y_u)$ and the observable state is $S(x_p, y_u)$. A trace leading to the observable is given by the dotted path (init, $py$, $px$, $uy$, $obs$). It can be compressed into the solid path (init, $px$, $obs$).

|   | $(s, x)_i$ | $(s, y)_i$ | $C_{4,x}$ | $(s, x)_i$ | $(s, y)_i$ | $C_{3,x}$ | $(s, x)_i$ | $(s, y)_i$ | $C_{2,x}$ | $(s, x)_i$ | $(s, y)_i$ | $C_{1,x}$ | $(s, x)_i$ | $(s, y)_i$ | $C_{4,y}$ | $(s, x)_i$ | $(s, y)_i$ | $C_{3,y}$ | $(s, x)_i$ | $(s, y)_i$ | $C_{1,y}$ |
| 0 | init | $\times$ | $u$ | $\times$ | $u$ | $C_{4,x}$ | $(x_4 \Rightarrow x_2)$ | $C_{3,x}$ | $(x_3 \Rightarrow x_2)$ | $C_{2,x}$ | $(x_2 \Rightarrow x_0)$ | $C_{1,x}$ | $(x_1 \Rightarrow x_0)$ | $C_{4,y}$ | $(x_4 \Rightarrow (x_3 \vee \neg x_1 \land x_0))$ | $C_{3,y}$ | $(x_3 \Rightarrow x_1)$ | $C_{1,y}$ | $(x_1 \Rightarrow x_0)$ |
| 1 | $py$ | $u$ | $u$ | $p$ | $C_{4,y}$ | $(x_4 \Rightarrow (x_3 \vee \neg x_1 \land x_0))$ | $C_{3,y}$ | $(x_3 \Rightarrow x_1)$ | $C_{1,y}$ | $(x_1 \Rightarrow x_0)$ |
| 2 | $px$ | $u$ | $p$ | $C_{4,y}$ | $(x_4 \Rightarrow (x_3 \vee \neg x_1 \land x_0))$ | $C_{3,y}$ | $(x_3 \Rightarrow x_1)$ | $C_{1,y}$ | $(x_1 \Rightarrow x_0)$ |
| 3 | $uy$ | $p$ | $p$ | $u$ | $C_{4,y}$ | $(x_4 \Rightarrow (x_3 \vee \neg x_1 \land x_0))$ | $C_{3,y}$ | $(x_3 \Rightarrow x_1)$ | $C_{1,y}$ | $(x_1 \Rightarrow x_0)$ |
| 4 | $obs$ | $p$ | $u$ | $C_{4,y}$ | $(x_4 \Rightarrow (x_3 \vee \neg x_1 \land x_0))$ | $C_{3,y}$ | $(x_3 \Rightarrow x_1)$ | $C_{1,y}$ | $(x_1 \Rightarrow x_0)$ |

In the table above, each column corresponds to a variable ($s_3$ is omitted) and is split into two parts: on the left are its tested values and on the right are its assigned values. The character $\times$ corresponds to the null value. Next to this table are the clauses corresponding to the two variables $(s, x)_i$ and $(s, y)_i$, abbreviated in $x$ and $y$.

Figure 5: An example of weak compression
6.1 Problem statement

Most models admit an infinite set of stories. Although an algorithm might be able to generate a finite representation of such a set, we see no structure in it that could enable such a representation to exist and we’re not especially optimistic that there is one.

A natural way to circumvent this is to generate only the set of all minimal stories for some compression relation. Using weak-compression, the set of minimal stories may still be infinite but it cannot in the case of strong-compression, as a consequence of Higman’s theorem on well-quasi-orderings. Unfortunately, the problem of computing the set of all strongly compressed stories of a model is undecidable. Indeed, it is more difficult than the reachability problem which consists in deciding whether or not a given rule can belong to a valid trace. This last problem is undecidable itself as a consequence of the Turing-completeness of kappa.

In the following presentation of our algorithm, we get rid of termination problems by limiting the size of the generated stories.

6.2 Overview

Our algorithm doesn’t generate stories directly but rather generates some abstract stories we call local stories. Then, it is easy to get concrete stories from them through a concretization operator. This abstract domain of local stories is especially designed to make the search process fast and non-redundant. Moreover, it admits a nice characterization of valid abstract stories, that is abstract stories \( s \) such that \( \alpha^{-1}(\{s\}) \neq \emptyset \), and this enables us to cut early the search branches that contain none of them.

Let \( A \) the set of all abstract stories and \( A^* \subseteq A \) the set of valid ones. The search process features a set of disjoint sets \( S = \{S_i \subseteq A\}_i \) sorted in a priority queue such that the following invariant holds:

\[
A^* \subseteq \bigcup_i S_i \cup F
\]

where \( F \subseteq A^* \) is the set of the valid abstract stories that have already been found. It starts with \( S = \{A\} \) and \( F = \emptyset \) and terminates eventually when \( S = \emptyset \) in which case \( F = A^* \). At each step of the algorithm, an element of lowest cost is picked in \( S \) and removed from it, call it \( S_i \). This element passes through a dual closure operator \( p \) respecting the following properties for all \( X \subseteq A \):

\[
p(X) \subseteq X \quad p(X) \cap A^* = X \cap A^* \quad (p \circ p)(X) = p(X)
\]

Then, three situations can occur:

1. We have \( p(S_i) = \{a\} \) with \( a \in A^* \) in which case \( a \) is added to \( F \).
2. We can prove $p(S_i) \cap A^* = \emptyset$, in which case nothing is done.

3. Otherwise, $S := S \cup b(p(S_i))$, with $b$ a branching function partitioning $p(S_i)$ into a set of sets of strictly lower cardinal.

In the case $A$ is finite, the algorithm is guaranteed to terminate. Indeed, let’s assume $A$ is of cardinal $n$. Then the vector $(v_1, \ldots, v_n)$ defined by:

$$v_i = |\{k : |S_k| = i\}|$$

decreases strictly for the lexicographic order at each iteration. The OCaml code of this search algorithm is given Figure 6. Before we give more details, we need to introduce rigorously the notion of a local story.

```ocaml
let visit (env, ss) s =  
  let s = propagate env s in  
  if is_bottom env s then (env, ss)  
else if is_solution env s then (add_solution s env, ss)  
else  
  let branches, env = branch env s in  
  (env, List.fold_right Spq.insert branches ss)

let rec search (env, ss) =  
  try  
    let s, ss = Spq.extract_min ss in  
    search (visit (env, ss) s)  
  with Leftistheap.Empty -> env
```

Figure 6: The OCaml code of the search algorithm

### 6.3 Local stories

#### 6.3.1 Definition

A local story is a tuple $(N, A)$ where:

- $N$ is a set of nodes, each node $i$ being labelled by a rule $r_i$.
- $A$ is a set of arrows, each arrow connecting an assignment of a node to a test of another one. More formally, an arrow $a$ is defined by:
  - Its source $\text{src}(a) = \langle i, \mu \rangle$ with $i$ a node and $\mu \in \text{eff}(r_i)$. 

– Its destination \( \text{dst}(a) = (j, \theta) \) with \( j \) a node and \( \theta \in \text{pre}(r_j) \).

The source node \( i \) of \( a \) is written \( \text{src}(a) \) and its destination node \( j \) is written \( \text{dst}(a) \). Finally, the variable modified by \( \mu \) is written \( \text{srcv}(a) \) and the variable tested by \( \theta \) is written \( \text{dstv}(a) \).

Contrary to a story whose nodes are labelled by events, the nodes of a local story are labelled by rules. However, for each test of each node, we keep track of the exact assignment that is responsible for its success.

### 6.3.2 Local stories as abstract stories

It is possible to map a valid trace \( t = e_0, \ldots, e_n \) into the local story \( l(t) \) defined as follows:

- For each event \( e_i = (r_i, \varphi_i) \) in \( t \), we introduce a node \( i \) labelled by \( r_i \).
- For each \( i < j \) and each trait \( (x, v) \in \text{eff}(e_i) \cap \text{pre}(e_j) \) such that \( x \) is not reassigned by any event between \( e_i \) and \( e_j \), we introduce an arrow \( a \) with:

\[
\text{src}(a) = (i, \varphi_i^{-1}(x, v)) \quad \text{dst}(a) = (j, \varphi_j^{-1}(x, v))
\]

where \( \varphi^{-1}(t) \) replaces any global agent identifier appearing in \( t \) by its local counterpart using \( \varphi^{-1} \).

In the case of regular models, \( l \) is preserved by the strong similarity relation on traces, that is:

\( t \sim t' \implies l(t) = l(t') \)

Therefore, it is possible to define an abstraction \((A, \alpha)\) with \( A \) the abstract domain of local stories and, for all story \( s \), \( \alpha(s) = l(t) \) with \( t \) any one of its trajectories. In the next section, we introduce a necessary and sufficient condition for an abstract story \( a \) to be valid along with a characterization of \( \alpha^{-1}(\{a\}) \).

### 6.3.3 A characterization of validity

A first necessary condition for an abstract story to be valid is that there is exactly one incomming arrow towards each non-null test of each node. In this case, it is said to be complete. Moreover, the arrows have to be well-typed in the sense that they have to connect compatible variables. For instance, a binding assignment can’t be connected to a phosphorylation test. More formally, for each arrow \( a \) with \( \text{src}(a) = (i, \mu) \) and \( \text{dst}(a) = (j, \theta) \), there has to exist two instantiation maps \( \varphi_1 \) and \( \varphi_2 \) such that \( \varphi_1(\mu) = \varphi_2(\theta) \). This is true if and only if \( \tau(\mu) = \tau(\theta) \) where \( \tau(t) \) is defined as the type of a trait \( t \) by replacing each agent identifier \( i \) by \( \tau(i) \) in \( t \). Moreover, in the specific case of kappa, we have
to ensure that for each arrow that explains a test of the form $\langle x \rangle_\lambda = \text{bound}(y)$, a symmetric arrow explains $\langle y \rangle_\lambda = \text{bound}(x)$ too.

In an abstract story $s$, agents are local to each node. Therefore, in order to build traces from $s$, it is necessary to map each local agent of each node into a global entity so each rule can be instanciated into an event. Thus, suppose we have a map:

$$\pi : N \times I_L \to I_G$$

with $N$ the set of nodes of $s$, $I_L$ its set of local agent identifiers and $I_G$ a set of global agent identifiers. Then, let’s consider the set of events:

$$E_\pi(s) = \{ \langle r_i, \pi(i, \cdot) \rangle : i \in N \}$$

Two necessary conditions on $\pi$ for $E_\pi(s)$ to be the set of events of a trace $t$ such that $(\alpha \circ C)(t) = s$ are the following:

1. Two different agents of a same node can’t be mapped into the same global entity, that is:

$$\forall i, x, y, \pi(\langle i, x \rangle) = \pi(\langle i, y \rangle) \implies x = y$$

2. For each arrow $a$, the local agent featured by the source of $a$ and the one featured by its destination have to be mapped to the same global entity:

$$\pi(\langle \text{srcn}(a), (\text{ag} \circ \text{srcv})(a) \rangle) = \pi(\langle \text{dstn}(a), (\text{ag} \circ \text{dstv})(a) \rangle)$$

where $\text{ag}(v)$ maps a variable to the agent it describes for any variable $v$.

Now, suppose that these two conditions are met. By hypothesis, we have:

$$\pi(\langle \text{srcn}(a), \cdot \rangle(\text{srcv}(a))) = \pi(\langle \text{dstn}(a), \cdot \rangle(\text{dstv}(a)))$$

for each arrow $a$ and we write both sides $\text{var}_\pi(a)$. We call $\text{var}_\pi(a)$ the global variable featured by $a$. Let $\preceq$ a total order on $E_\pi(s)$ and $t$ the unique trace whose events are in $E_\pi(s)$ and are ordered by $\preceq$. Two necessary conditions on $\preceq$ so that $(\alpha \circ C)(t) = s$ are the following:

1. For each arrow $a$, $\text{srcn}(a) \prec \text{dstn}(a)$.

2. For each triple of distinct nodes $(x, y, z)$ such that there exists an arrow $a$ such that:

$$x = \text{srcn}(a) \land y = \text{dstn}(a) \land \text{var}_\pi(a) \in \pi(\langle z, \cdot \rangle(\text{eff}(r_z)))$$

we have $z \prec x \lor z \succ y$. 

29
When these two conditions hold, we say that $\preceq$ is a valid $\pi$-scheduling of $s$. An example of an abstract story which admits no valid $\pi$-scheduling is provided Figure 7. This leads us to our main theorem.

**Theorem 4.** Let $s$ an abstract story with well-typed arrows, $N$ its set of nodes and $I_L$ its set of local agent identifiers. Let $\equiv$ the smallest equivalence relation on $N \times I_L$ such that for every arrow $a$ of $s$:

$$\langle \text{src}(a), (\text{ag} \circ \text{srcv})(a) \rangle \equiv \langle \text{dst}(a), (\text{ag} \circ \text{dstv})(a) \rangle$$

Let $I_G = (N \times I_L / \equiv)$ and $\pi$ the canonical projection from $N \times I_L$ into $I_G$. Then $s$ is valid if and only if:

1. $s$ is complete
2. $\pi$ maps different agents of a same node of $s$ into different global entities:
   $$\forall n \in N, \forall x, y \in X_{I_L}, \pi(\langle n, x \rangle) = \pi(\langle n, y \rangle) \implies x = y$$
3. $s$ admits a valid $\pi$-scheduling $\preceq$

Moreover, if $s$ is valid, then $\alpha^{-1} \{s\}$ contains only one story (modulo renaming of agents) that is equal to $C(t)$ for $t$ any trace whose events are in $E_\pi(s)$ and are ordered by $\preceq$.

Note that the last part of the theorem only holds for regular models, in which case it suggests a straightforward concretization operator.

### 6.4 Partial abstract stories

In the high-level description of the search algorithm of section 6.2 we manipulated sets of abstract stories. These sets are encoded by partial abstract stories.

#### 6.4.1 Definition

Let $s$ and $s'$ two abstract stories whose sets of nodes are $N$ and $N'$. An injective map $f : N \to N'$ is said to be an embedding of $s$ into $s'$ if it preserves both the nodes labels and the arrows of $s$.

A rooted abstract story is a tuple $(s, \omega)$ where $s$ is a connex abstract story and $\omega$ one of its nodes, usually the observable. We say that $(s, \omega)$ is embedded into $(s', \omega')$ if there exists an embedding $f$ of $s$ into $s'$ such that $f(\omega) = f(\omega')$. Note that as $s$ and $s'$ are connex, such an embedding is unique if it exists.

A partial abstract story is a triple $(s, \omega, c)$ such that $(s, \omega)$ is a rooted abstract story and $c$ is a set of alternatives. An alternative $a$ is given by:

- A destination $\text{dst}(a) = \langle n, \theta \rangle$ where $i$ is a node of $s$ and $\theta \in \text{pre}(r_i)$, like in the definition of the destination of an arrow.
Let’s consider three rules $px$, $py$ and $obs$ and an abstract story $s$ featuring a node for each one of them – among other things that are not shown here:

$$px : A(x_u, y_p) \rightarrow A(x_p, y_u)$$
$$py : A(x_p, y_u) \rightarrow A(x_u, y_p)$$
$$obs : A(x_p, y_p) \rightarrow A(x_p, y_p)$$

Let’s suppose that the phosphorylation of the site $x$ of the unique agent featured by $obs$ is due to $px$ and that the phosphorylation of the site $y$ of the same agent is due to $py$, which gives us two arrows. These arrows make all the three agents featured by each node equivalent for $\equiv$, and so they are all mapped into the same global instance by $\pi$, let’s write it $a$.

On the diagram above, we write $x$ the global variable $\langle a, x \rangle$, and $y$ the global variable $\langle a, y \rangle$. We write next to each node the set of global variables it modifies and each arrow $a$ is labelled by $\text{var}_a(a)$. We show below that $s$ does not admit a valid scheduling, which implies that $s$ is not valid.

There is an arrow from $py$ to $obs$:

$$py \prec obs$$

The node $py$ modifies a variable that is featured by an arrow going from $px$ to $py$:

$$(py \prec px) \vee (py \succ obs)$$

Combining these two clauses:

$$py \prec px$$

With a symmetric reasoning, we can show that:

$$px \prec py$$

which is a contradiction. As a consequence, $s$ is not valid.

---

Figure 7: An example of impossible scheduling
• A set of possible sources $\text{src}^\star(a)$, a possible source being a tuple $\langle \hat{n}, \mu \rangle$ where $\hat{n}$ is either:
  
  – an existing node of $s$ labelled by a rule $r$
  – the special value $\text{new}(r)$ with $r$ a rule

and $\mu \in \text{eff}(r)$.

Intuitively, an alternative describes a set of possible arrows with the same destination but different sources. These possible sources can feature nodes that do not exist yet in $s$ but whose associated rule is known.

An alternative $a$ is said to be empty if $\text{src}^\star(a) = \emptyset$ and it is said to be resolved if $\text{src}^\star(a)$ is a singleton. Finally, if $p = (s, \omega, c)$ is a partial abstract story, we write $p = s$ its underlying abstract story and $\text{alt}(p) = c$ its set of alternatives.

6.4.2 The extension relation

We say that a rooted abstract story $(s', \omega')$ extends a partial abstract story $(s, \omega, c)$ when the following conditions hold:

• There exists an embedding $f$ from $(s, \omega)$ into $(s', \omega')$.

• For each $a \in c$ with $\text{dst}(a) = \langle m, \theta \rangle$, there exists $\langle \hat{n}, \mu \rangle \in \text{src}^\star(a)$ such that there is an arrow in $s'$ whose destination is $\langle f(m), \theta \rangle$ and whose source is:

  – $\langle n, \mu \rangle$ for $n$ a node of $s'$ labelled with $r$ which is not in $f(N)$ if $\hat{n} = \text{new}(r)$, with $N$ the set of nodes of $s$
  – $\langle f(\hat{n}), \mu \rangle$ otherwise

This can be used to represent sets of rooted abstract stories with partial abstract stories. Indeed, for $s_p$ a partial abstract story, we define $[s_p]$ as the set of complete rooted abstract stories with well-typed arrows extending $s_p$.

6.5 The search algorithm

Now that we have introduced the notion of a partial abstract story, we can complete our description of the search algorithm introduced in section 6.2. All abstract stories are implicitly rooted in their unique observable node and we use partial abstract stories to represent sets of them. In particular, the set $S$ of section 6.2 is now regarded as a set of partial abstract stories.

6.5.1 Manipulating partial abstract stories

The following invariant is preserved during the search process:
**Invariant 1.** For all $s \in S$, each non-null test of each node of $s$ is the destination of an arrow or of an alternative (exclusively).

As we see later, it guarantees that only complete abstract stories are generated. Moreover, no valid abstract story is forgotten as guaranteed by the invariant:

**Invariant 2.**

$$A^* \subseteq \bigcup_{s \in S} [s] \cup F$$

Suppose that we have a partial abstract story $s$ with a resolved alternative, that is an alternative $a$ such that $\text{src}^*(a)$ is a singleton. Then, it is natural to remove this alternative and add the unique arrow it suggests in $s$. This is indeed a part of what the propagation operator actually does. In the case where $\text{src}^*(a) = \{\langle \text{new}(r), \mu \rangle\}$, adding such an arrow requires adding a new node $n$ labelled by $r$. Then, a new alternative has to be added to $s$ for each non-null test of $r$ in order to preserve Invariant 1. Moreover, this whole operation should leave $[s] \cap A^*$ invariant so that Invariant 2 is preserved. For this to hold, these alternatives have to be kind of exhaustive but that’s not sufficient. Indeed, if $\langle \text{new}(r), \mu' \rangle \in \text{src}^*(a')$ for another alternative $a'$, then $\langle n, \mu' \rangle$ has to be added to $\text{src}^*(a')$ too.

In order to make node creation easier to handle, we don’t encode $\text{alt}(s)$ as a set of alternatives. Instead, we define for each abstract story $x$ a set $A(x)$ such that:

$$\forall s, [s] \cap A^* \neq \emptyset \implies \text{alt}(s) \subseteq A(s)$$

and we encode instead its complementary set of disqualified possibilities $\text{disq}(s)$ such that $\text{alt}(a)$ is obtained from $A(s)$ by:

- Removing in it all the alternatives $a$ such that $\text{dst}(a)$ is the destination of an arrow in $s$.
- For all remaining alternative $a$, removing in $\text{src}^*(a)$ the elements $t$ such that $(a, t) \in \text{disq}(s)$.

It remains to define $A$. For this, let’s consider an abstract story $s$. If $n$ is a node of $s$ and $t$ a trait, the *kind* of the tuple $(n, t)$ is defined by:

$$\kappa_s((n, t)) = (r_n, t)$$

with $r_n$ the rule labelling $n$ in $s$. For $r$ a rule and $\theta \in \text{pre}(r)$, we define $\text{src-kinds}((r, \theta))$ as the set of all tuples $\langle r', \mu \rangle$ for $r'$ a rule and $\mu \in \text{eff}(r')$ such that $\tau(\mu) = \tau(\theta)$. Then, for any valid abstract story $s \in A^*$ and $a$ an arrow of $s$, we have:

$$(\kappa_s \circ \text{src})(a) \in (\text{src-kinds} \circ \kappa_s \circ \text{dst})(a)$$
As a consequence, $\text{src-kinds}(k)$ can be regarded as a superset of all the possible kinds of sources for an arrow whose destination has kind $k$ in a valid abstract story. Then, for $s$ an abstract story and $N$ its set of nodes, $A(s)$ is defined as the set of alternatives $a$ with $\text{dst}(a) = \langle j, \theta \rangle$ and:

$$
\text{src}^*(a) = \{ \langle \text{new}(r), \mu \rangle : \langle r, \mu \rangle \in (\text{src-kinds} \circ \text{dst})(a) \} \\
\cup \{ \langle i, \mu \rangle : i \in N, \langle r_i, \mu \rangle \in (\text{src-kinds} \circ \text{dst})(a) \}
$$

for $j \in N$ and $\theta \in \text{pre}(r_j)$.

### 6.5.2 Initialization of the algorithm

The algorithm is initialized with $S = \{s_0\}$ where $s_0$ is a partial abstract story with one node labelled by $\text{obs}$ and no arrow and such that $\text{disq}(s_0) = \emptyset$. We have indeed:

$$A^* \subseteq [s_0]$$

and Invariant $\square$ is respected by definition of $\text{disq}$.

### 6.5.3 Consistency checks

The search algorithm has to estimate at each iteration whether or not a partial story $s$ is inconsistent, that is:

$$[s] \cap A^* = \emptyset$$

Of course, such a test is undecidable in general as the reachability problem can be reduced to it. Hopefully, a wide range of inconsistent partial stories can be recognized using Theorem $4$. Indeed, if $s$ is a partial abstract story such that either condition 2 or 3 of Theorem $4$ does not hold for $s$, then it is inconsistent.

In order to check whether or not we are in this case, we first compute the equivalence relation $\equiv$ for $s$ with a union-find datastructure, whose $\text{find}$ operation corresponds to the $\pi$ map. Then, it is possible to check in quasi-linear time that no two agents of a same node are equivalent for $\equiv$. After this, we check that $s$ admits a valid scheduling for $\pi$, which is a problem of satisfiability modulo the theory of partial orders. More specifically, there is one variable for each node of $s$ and clauses are of the form:

$$a \prec b \quad \text{and} \quad (x \prec a) \lor (x \succ b)$$

The number of clauses is cubic in the number of nodes of $s$ in the worst case, but it is closer to be linear in practice. Moreover, although the satisfiability test is exponential in the worst case, it is often linear as the high proportion of clauses with only one literal makes constraint propagation very efficient. Finally, when a size limit on generated stories is given, partial abstract stories whose number
of nodes exceeds this limit are considered inconsistent. Moreover, we may want to limit to one the number of nodes labelled by $\text{init}$.

An other test the search algorithm has to perform is whether or not:

$$[s] = \{a\}, \ a \in A^*$$

for $s$ a partial abstract story. This one is especially easy as it holds if and only if $s$ passes the consistency check described above and has an empty set of alternatives. Indeed, if $s$ has an empty set of alternative, it is complete thanks to Invariant 1. We conclude using Theorem 4.

6.5.4 The $p$ propagation operator

Let $s$ a partial abstract story, $a \in \text{alt}(s)$ and $t \in \text{src}^*(a)$. We write $\text{choose}_{(a,t)}(s)$ the partial abstract story built from $s$ by adding the arrow whose source is given by $t$ and whose destination is given by $\text{dst}(a)$ to it, creating a new node if needed. Moreover, each time an arrow explaining a test of the form $\langle x \rangle_\lambda = \text{bound}(y)$ is added, the appropriate symmetric arrow explaining $\langle y \rangle_\lambda = \text{bound}(x)$ has to be added too.

The value of $\text{disq}$ does not need to be updated in $\text{choose}_{(a,t)}(s)$, which is one reason this encoding of the set of alternatives is relevant.

The $p$ operator takes a partial abstract story $s$ and does two things to it:

1. For every alternative $a$ of $s$, if $a$ is empty, we mark $s$ as inconsistent so it is further eliminated. If $a$ is resolved, that is $\text{src}^*(a)$ is a singleton $\{t\}$, then $s$ is updated into $\text{choose}_{(a,t)}(s)$.

2. For each alternative $a$ of $s$ and $t \in \text{src}^*(a)$, if $\text{choose}_{(a,t)}(s)$ is provably inconsistent, then we add $(a,t)$ to $\text{disq}(s)$. Here, we don’t use the expensive consistency check introduced in section 6.5.3 but a constant-time version of it where scheduling constraints are generated only for the neighborhood of the new arrow.

These two operations are repeated until they leave $s$ unchanged.

6.5.5 The $b$ branching operator

Let $s$ a partial abstract story, $a \in \text{alt}(s)$ and $t \in \text{src}^*(a)$ two well-chosen values. Then, $b(s) = \{s_1, s_2\}$ where:

- $s_1 = \text{choose}_{(a,t)}(s)$

- $s_2$ is built from $s$ by adding $(a,t)$ to $\text{disq}(s)$

Indeed, $\{[s_1], [s_2]\}$ is a partition of $[s]$. The choice of $(a,t)$ is governed by a heuristic we discuss in section 6.5.6.
6.5.6 Cost of a partial story

On many real world models, the search space becomes so large that this algorithm is unlikely to generate the whole set of valid stories in a reasonable amount of time. However, when guided in its exploration by a good heuristic function, it may find the most interesting ones very quickly.

Heuristics play an important role in two parts of the algorithm: when picking an element of $S$ and when branching. Indeed, each story of $S$ is associated a cost and an element of lowest cost is picked at each iteration. Moreover, branching is usually done in a way that minimizes the cost of the first branch $s_1$. Many strategies are possible to assign costs to partial abstract stories but here are some basic principles:

- Penalizing node creation is usually a good greedy strategy to generate minimal stories first.
- Two arrows $a$ and $a'$ such that $\var_{\pi}(a) = \var_{\pi}(a')$ should share the same source as often as possible and partial abstract stories that are parcimonious in this sense should be assigned lower costs.
- If the user is confident about the branching strategy and he wants to get to the first story very quickly, a depth-first search is a good bet.
- Partial abstract stories that are similar to stories that have already been found can be penalized so that the first stories to be generated are more representative of the whole set of them.

6.6 Experimental results and prospects

A generic OCaml implementation of this algorithm is available at [https://github.com/jonathan-laurent/kappa-stories](https://github.com/jonathan-laurent/kappa-stories). It was tested on the Kasim test suite provided at [https://github.com/Kappa-Dev/KaSim/tree/master/models/test_suite/cflows](https://github.com/Kappa-Dev/KaSim/tree/master/models/test_suite/cflows) and asked to generate a single compressed story for each test case: each run took less than one second on average. However, in the current stage of its development, it is not well-suited for generating exhaustive sets of strongly compressed stories, even for small models and with a bound on the size of relevant stories.

Indeed, although the number of strongly compressed stories might be reasonably small for most models of interest, generating all of them would require to cut very early huge parts of the search space containing only uncompressed stories. No solution to this problem is currently implemented but we expect some progresses in a near future. Indeed, we are working on defining a notion of compressible pattern on partial abstract stories such that the following property holds:
Proposition 4. If $s$ is a partial abstract story such that $\text{choose}_{(a,t)}(s)$ features a compressible pattern for any choice of $(a,t)$, then $\alpha^{-1}([s])$ does not contain any compressed stories.

Finally, the heuristic which is currently implemented is very naive – a depth-first search strategy penalizing the creation of nodes – but we’re confident that some minor improvements in it will make our implementation scalable to much larger models.

7 Conclusion and future work

In this report, we introduced stories as formal counterparts to biological pathways and provided some generic techniques to generate them from rule-based models of protein-protein interaction networks.

When discussing causality, people usually distinguish necessary causes from sufficient ones. In medicine or in biology, both notions are often too demanding to be met in practice and a notion of contributive causality is usually preferred, where an event $e_1$ is said to be a cause of an event $e_2$ if it is an insufficient but non-redundant part of a condition which is itself unnecessary but sufficient for the occurrence of $e_2$. Minimal stories are defined in this exact spirit, as any event $e_1$ of a minimal story explaining $e_2$ is a cause of $e_2$ in this sense.

Finally, a lot of work remains to be done on causal analysis, in addition to what was mentioned in section 6.6. For instance, it would be interesting to break stories into logical blocks that would be larger than atomic events. Such blocks may consist in recurrent biological patterns corresponding to specific computational or assembling primitives. They could be defined a priori from well-known motifs or generated a posteriori by learning algorithms. Moreover, although stories may be considered useful per se for the intuitive insight they provide into the structure of a model when displayed as graphs, more subtle analysis techniques may be developed to extract useful informations from them. In particular, we could use causal analysis to highlight the behavioural differences between two variations of a same model.
References


