Constraining rule-based dynamics with types

Vincent Danos
University of Edinburgh

Russ Harmer*
CNRS & Université Paris Diderot / Harvard Medical School

Glynn Winskel
University of Cambridge

October 17, 2011

Abstract

A generalized framework of site graphs is introduced in order to provide the first fully semantic definition of the side-effect-free core of the rule-based language Kappa. This formalization allows for the use of types either to confirm that a rule respects a certain invariant or to guide a restricted refinement process that allows us to constrain its run-time applicability.

1 Introduction

Rule-based modelling [2, 6] has been proposed as one possible solution to the problem of defining and investigating the highly combinatorial microscopic transition systems found in cellular signalling networks and various models in statistical physics. Specifically, *rules* define rewrites of partially-specified macroscopic *patterns* that can match various fully-specified microscopic entities so that different instances of a rule may induce differing kinds of microscopic transition.

In previous work, we have introduced a mathematical framework of *site graphs* and embeddings in order to analyze how patterns can be refined to more specific patterns that match fewer microscopic entities. This led to the theory of rule refinement [7] that explains how to split a rule into subcases which can then, if desired, be distinguished kinetically.

*Corresponding author: `russ.harmer@pps.jussieu.fr`

The present contribution extends this previous work in several ways. Firstly, by tweaking some of the basic definitions, we obtain a larger class of site graphs and homomorphisms that allows for a fully general expression of the binding, unbinding and state change actions fundamental to the rule-based framework. Specifically, we allow for a larger class of site graphs where sites can have internal states and where their binding state need not be given explicitly. This is accompanied by a broader class of homomorphisms within which we can recover our previous notion of embedding.

Secondly, unlike our previous treatment which uses a syntactic definition of *actions* as lists of rewrite instructions called 'action scripts', we introduce here a purely semantic definition of actions, as appropriate spans of monos, and an associated site graph *rewriting* in the double-push-out style [5]. This provides the first fully semantic definition of a rule-based language that, in particular, captures precisely the side-effect-free fragment of Kappa.

Finally, we formalize the idea of *typing* site graphs with homomorphisms into some fixed site graph, the *contact graph*, whose structure specifies all the admissible links and states. This is reminiscent of categories of bundles and, indeed, we find an adjunction associated with changing contact graph analogous to the familiar adjunction arising from a change of base.

This leads to our main conceptual point: types can be used to constrain the dynamics engendered by a collection of rules. Indeed, types can be used in both a static and a dynamic manner. On the one hand, a rule may or may not be compatible with a given contact graph so that a type can be used statically to enforce certain invariants. On the other hand, as we will see later, an action can be tagged with a type which will only be checked dynamically, via the change of contact graph adjunction, in order to reject potential events if they do not satisfy the properties demanded by the type.

2 Site graphs

First of all, we recall some basic preliminaries about the category of sets in order to fix notation. If A is a set, we write $A_\star$ for the disjoint union $A + \{\star\}$ that 'adds a point' to A; elements of disjoint unions are tagged with either inl or inr to identify their provenance so that an element $a \in A$ is written as $\mathsf{inl}(a)$ in $A_\star$.

Given a co-span $A \xrightarrow{f} C \xleftarrow{g} B$ in the category **Set** of sets and total functions, we fix $A \times_C B := \{\langle a, b\rangle \in A \times B \mid f(a) = g(b)\}$ together with the projections $\pi_A : A \times_C B \to A$ and $\pi_B : A \times_C B \to B$ as our canonical choice of pull-back in **Set**.

Given a span $\begin{tikzcd} A & C \arrow[l, "f"'] \arrow[r, "g"] & B \end{tikzcd}$, we fix $A +_C B$ to be the quotient of $A + B$ by (the reflexive, symmetric and transitive closure of) the relation $\mathsf{inl}(a) \simeq \mathsf{inr}(b)$ iff, for some $c \in C$, $a = f(c)$ and $b = g(c)$; together with the injections $\iota_A : A \to A +_C B$ and $\iota_B : B \to A +_C B$ defined respectively by $a \mapsto [\mathsf{inl}(a)]$ and $b \mapsto [\mathsf{inr}(b)]$, this is our canonical choice of push-out in **Set**.

2.1 The category of site graphs

In this section, we introduce the category of site graphs and homomorphisms and the important full subcategory of realizable site graphs.

2.1.1 Basic definitions

A *site graph* G is specified by a tuple $\langle \mathcal{A}_G, \mathcal{S}_G, \mathcal{E}_G, \sigma_G, \varepsilon_G, \lambda_G \rangle$ where

- $\mathcal{A}_G$, $\mathcal{S}_G$ and $\mathcal{E}_G$ are finite sets (of agents, sites and states);
- $\sigma_G : \mathcal{S}_G \to \mathcal{A}_G$ assigns sites to agents and $\varepsilon_G : \mathcal{E}_G \to \mathcal{S}_G$ assigns states to sites;
- λ_G is a symmetric relation on $((\mathcal{S}_G)_\star \times (\mathcal{S}_G)_\star) - \{\langle \star, \star \rangle\}$.

The relation λ_G encodes the *link*, or *edge*, structure of the graph:

- if $\langle s, s' \rangle \in \lambda_G$, where $s, s' \in \mathcal{S}_G$, then there is a *link* between them;
- if $\langle s, \star \rangle \in \lambda_G$, so that $s \in \mathcal{S}_G$, then s has a *stub*.

A site $s \in \mathcal{S}_G$ may have neither an incident edge nor a stub; we write this as $s \notin \lambda_G$. This means that its binding status is unspecified.

We use a Kappa-like syntax [2] to denote site graphs textually. Briefly, each node has an *interface* consisting of an agent name and a set of site names. The two ends of a link are represented by equal numerical superscripts to the sites in question; a stub is represented by a $\star$ superscript. States are represented as subscripts to their sites. So

$$A(s_{0,1}, t^1), B(s^{1,2,\star}), C(t^2_p, s^\star), D(s^\star)$$

represents a graph with four agents: A has a site s with unspecified binding status but two states 0 and 1, plus a site t bound to site s of B which has a stub and is also bound to site t of C, *et c.* Note that a site can have multiple links and/or a stub hence the need, compared with usual Kappa syntax, to represent stubs explicitly rather than as simply the absence of a link.

A *homomorphism* $f : G_1 \to G_2$ of site graphs is specified by a tuple of functions $\langle f_{\mathcal{A}} : \mathcal{A}_{G_1} \to \mathcal{A}_{G_2}, f_{\mathcal{S}} : \mathcal{S}_{G_1} \to \mathcal{S}_{G_2}, f_{\mathcal{E}} : \mathcal{E}_{G_1} \to \mathcal{E}_{G_2}\rangle$ such that

$$\begin{tikzcd}
\mathcal{A}_{G_1} \arrow[d, "f_{\mathcal{A}}"'] & \mathcal{S}_{G_1} \arrow[l, "\sigma_{G_1}"'] \arrow[d, "f_{\mathcal{S}}"] \\
\mathcal{A}_{G_2} & \mathcal{S}_{G_2} \arrow[l, "\sigma_{G_2}"]
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathcal{S}_{G_1} \arrow[d, "f_{\mathcal{S}}"'] & \mathcal{E}_{G_1} \arrow[l, "\varepsilon_{G_1}"'] \arrow[d, "f_{\mathcal{E}}"] \\
\mathcal{S}_{G_2} & \mathcal{E}_{G_2} \arrow[l, "\varepsilon_{G_2}"]
\end{tikzcd}$$

commute and the link structure of G_1 is preserved:

- if $\langle s, s'\rangle \in \lambda_{G_1}$, for $s, s' \in \mathcal{S}_{G_1}$, then $\langle f_{\mathcal{S}}(s), f_{\mathcal{S}}(s')\rangle \in \lambda_{G_2}$;
- if $\langle s, \star\rangle \in \lambda_{G_1}$, so $s \in \mathcal{S}_{G_1}$, then $\langle f_{\mathcal{S}}(s), \star\rangle \in \lambda_{G_2}$.

The existence of a homomorphism f from G_1 to G_2 corresponds, not to any standard notion of *embedding* but rather, to a notion of *matching*: a site with unspecified binding status may be mapped to a site with a stub and/or links. However, we use the term *matching* only when $f_{\mathcal{S}}$ is injective.

2.1.2 Basic categorical structure

We write **SGrph** for the category of site graphs and homomorphisms with composition defined in the obvious component-wise fashion. The empty site graph $\mathbf{0}$ where $\mathcal{A}_{\mathbf{0}} = \mathcal{S}_{\mathbf{0}} = \mathcal{E}_{\mathbf{0}} = \emptyset$ is an initial object; and the singleton site graph $\mathbf{1}$ with $\mathcal{A}_{\mathbf{1}} = \mathcal{S}_{\mathbf{1}} = \mathcal{E}_{\mathbf{1}} = \emptyset_{\star}$ and where the unique site has both a stub and a self-loop is a terminal object. An arrow f is a mono if, and only if, its three constituent functions—$f_{\mathcal{A}}$, $f_{\mathcal{S}}$ and $f_{\mathcal{E}}$—are all injective; and is an epi if, and only if, the three functions are all surjective.

Given a co-span $G_1 \xrightarrow{f_{13}} G_3 \xleftarrow{f_{23}} G_2$, we take pull-backs in **Set**

$$\begin{tikzcd}
\mathcal{A}_{G_0} \arrow[r, "f_{02\mathcal{A}}"] \arrow[d, "f_{01\mathcal{A}}"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{A}_{G_2} \arrow[d, "f_{23\mathcal{A}}"] \\
\mathcal{A}_{G_1} \arrow[r, "f_{13\mathcal{A}}"'] & \mathcal{A}_{G_3}
\end{tikzcd}
\quad
\begin{tikzcd}
\mathcal{S}_{G_0} \arrow[r, "f_{02\mathcal{S}}"] \arrow[d, "f_{01\mathcal{S}}"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{S}_{G_2} \arrow[d, "f_{23\mathcal{S}}"] \\
\mathcal{S}_{G_1} \arrow[r, "f_{13\mathcal{S}}"'] & \mathcal{S}_{G_3}
\end{tikzcd}
\quad
\begin{tikzcd}
\mathcal{E}_{G_0} \arrow[r, "f_{02\mathcal{E}}"] \arrow[d, "f_{01\mathcal{E}}"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{E}_{G_2} \arrow[d, "f_{23\mathcal{E}}"] \\
\mathcal{E}_{G_1} \arrow[r, "f_{13\mathcal{E}}"'] & \mathcal{E}_{G_3}
\end{tikzcd}$$

to define the span $G_1 \xleftarrow{f_{01}} G_0 \xrightarrow{f_{02}} G_2$, where the link structure of G_0 is defined by:

- $\langle s_1, s_2\rangle \ \lambda_{G_0} \ \langle s_1', s_2'\rangle$ iff $s_1 \ \lambda_{G_1} \ s_1'$ and $s_2 \ \lambda_{G_2} \ s_2'$;
- $\langle s_1, s_2\rangle \ \lambda_{G_0} \ \star$ iff $s_1 \ \lambda_{G_1} \ \star$ and $s_2 \ \lambda_{G_2} \ \star$.

The span $G_1 \xleftarrow{f_{01}} G_0 \xrightarrow{f_{02}} G_2$ is our specified choice of pull-back in **SGrph**; intuitively, G_0 is an 'intersection' of G_1 and G_2, *i.e.* the largest, and so the least general, site graph that can match any site graph that either G_1 or G_2 can match—although, of course, this only makes sense in the context of the co-span into G_3. One immediate consequence of this is that **SGrph** has finite products, obtained by taking pull-backs from the terminal object.

Dually, given $G_1 \xleftarrow{f_{01}} G_0 \xrightarrow{f_{02}} G_2$, define $G_1 \xrightarrow{f_{13}} G_3 \xleftarrow{f_{23}} G_2$ with the push-outs in **Set**, analogous to the above pull-backs, and defining the link structure of G_3 as:

- $[s]\ \lambda_{G_3}\ [s']$ iff either, for some $s_1, s'_1 \in \mathcal{S}_{G_1}$, $\mathsf{inl}(s_1) \in [s]$ and $\mathsf{inl}(s'_1) \in [s']$ and $s_1\ \lambda_{G_1}\ s'_1$; or, for some $s_2, s'_2 \in \mathcal{S}_{G_2}$, $\mathsf{inr}(s_2) \in [s]$ and $\mathsf{inr}(s'_2) \in [s']$ and $s_2\ \lambda_{G_2}\ s'_2$;
- $[s]\ \lambda_{G_3}\ \star$ iff either, for some $s_1 \in \mathcal{S}_{G_1}$, $\mathsf{inl}(s_1) \in [s]$ and $s_1\ \lambda_{G_1}\ \star$; or, for some $s_2 \in \mathcal{S}_{G_2}$, $\mathsf{inr}(s_2) \in [s]$ and $s_2\ \lambda_{G_2}\ \star$.

The co-span $G_1 \xrightarrow{f_{13}} G_3 \xleftarrow{f_{23}} G_2$ is our choice of push-out in **SGrph**; intuitively, it is a 'union' of G_1 and G_2, *i.e.* the smallest, and so most general, site graph that can match any site graph that both G_1 and G_2 can match—relative to the given span from G_0. This means that **SGrph** has finite co-products, obtained by taking push-outs from its initial object.

2.1.3 Realizable site graphs

The structure of a site graph can be interpreted in two different ways: either its stubs and edges specify *possibilities*, *i.e.* admissible bonds and free sites; or they describe an *actuality*, *i.e.* a real configuration of agents.

For our purposes, this latter interpretation only makes sense for site graphs whose link/stub and state structure is 'deterministic' in the following natural sense:

- for all $s \in \mathcal{S}_G$, if $\langle s, s_1\rangle \in \lambda_G$ and $\langle s, s_2\rangle \in \lambda_G$ then $s_1 = s_2$;
- the state map ε_G is injective.

The idea is that each site is a resource that, at any given time, can have at most one state and be dedicated to at most one task, *i.e.* it can be free (a stub) or bound (linked to another site) but not both and, if bound, only to one thing. We call such site graphs *realizable*.

If $f : G_1 \to G_2$ and G_2 is realizable then G_1 need not be realizable, *e.g.* there is an obvious homomorphism mapping $G_1 := A(s^{1,2}), B(t^1, t^2)$ to $G_2 := A(s^1), B(t^1)$. (We are implicitly defining the homomorphism with our choice of 'names' for nodes and sites: the node named A in G_1 is mapped to the node named A in G_2, the sites named t in G_1 are both mapped to the site named t in G_2, *et c.*) However, if $f_{\mathcal{S}}$ is injective, *i.e.* f is a matching, then G_1 must be realizable too since $f_{\mathcal{S}}$ being injective would force any link non-determinism in G_1 to be propagated to G_2.

In fact, given that G_2 is realizable, f being a matching is a sufficient, but not necessary, condition for G_1 to be realizable. However, the following weaker condition, that 'f is *V-preserving*', *is* equivalent to G_1 being realizable: if $s\ \lambda_{G_1}\ s_1$, $s\ \lambda_{G_1}\ s_2$ and $s_1 \neq s_2$ then $f_{\mathcal{S}}(s_1) \neq f_{\mathcal{S}}(s_2)$. This condition allows the obvious homomorphism from $A(s^1), B(t^1), A(s^2), B(t^2)$ to $A(s^1), B(t^1)$, which is non-injective on sites, while still disallowing the homomorphism from $A(s^{1,2}), B(t^1, t^2)$.

We write **rSGrph** for the full subcategory of **SGrph** whose objects are realizable site graphs. The category **rSGrph** inherits pull-backs from **SGrph**: given a co-span $G_1 \xrightarrow{f_{13}} G_3 \xleftarrow{f_{23}} G_2$, the pull-back of f_{13} and f_{23}, considered as arrows of **SGrph**, yields a span $G_1 \xleftarrow{f_{01}} G_0 \xrightarrow{f_{02}} G_2$ where f_{01} and f_{02} are injective on sites, since pull-backs preserve monos, so that G_0 is realizable. This span is also the pull-back in **rSGrph**.

The situation is more complicated with regard to push-outs. Firstly, push-outs need not exist since the construction in **SGrph** does not guarantee that G_3 is realizable, even if G_0, G_1 and G_2 all are, *e.g.* if a site has different states in G_1 and G_2. Secondly, even if a push-out exists, it may not be the same as in **SGrph**, *e.g.* if $G_0 = A(s)$ and $G_1 = G_2 = A(s^1), B(t^1)$ then $G_3 = A(s^1), B(t^1)$ in **rSGrph** but $G_3 = A(s^{1,2}), B(t^1), B(t^2)$ in **SGrph**.

2.2 Typing site graphs

In rule-based modelling, our main interest is in realizable site graphs as they represent actual configurations of agents and connected components. However, we can also use arbitrary, *i.e.* not necessarily realizable, site graphs to *type* (realizable) site graphs: a homomorphism from a (realizable) site graph G to an arbitrary site graph C guarantees that all edges and stubs in G also occur in C, so C could be taken as a specification of admissible stubs and edges that is *satisfied* by G, *i.e.* as a type. We formalize this intuition with the standard notion of a *slice* category over C before introducing the subcategories of interest to us here, $\mathbf{SGrph}_C$ and $\mathbf{rSGrph}_C$.

2.2.1 Categories over C

The slice category $\mathbf{SGrph}/C$ over a site graph C has, for objects, all arrows $h : G \to C$ of $\mathbf{SGrph}$ into C; we think of these as witnesses that 'G has type C'. We refer to C as the *contact graph* and $h : G \to C$ as a *contact map*. Note that many different Cs could act as contact graphs for G. Moreover, we require no particular properties of C; its status as a contact graph is bestowed by fiat.

An arrow $f : h_1 \to h_2$ between $h_1 : G_1 \to C$ and $h_2 : G_2 \to C$ is an arrow $f : G_1 \to G_2$ of $\mathbf{SGrph}$ making $h_1 = h_2 \circ f$. In other words, a homomorphism from G_1 to G_2 that preserves typing.

The category $\mathbf{SGrph}_C$ is obtained as a subcategory of $\mathbf{SGrph}/C$ by restricting the objects to be those contact maps $h : G \to C$ that are *locally injective* on sites: no two sites of the *same agent* of G can map to the same site of C. This means that each agent a of G has at most one copy of each site of its corresponding agent $h(a)$ of C, *i.e.* agents have sets, not multi-sets, of sites. As an immediate consequence, all arrows $f : h_1 \to h_2$ are locally injective on sites. If, moreover, f is injective on agents then it is injective on sites too; however, the converse is not true in general, *e.g.* there is a natural arrow from $G_1 := A(s), A(t)$ to $G_2 := A(s,t)$.

The basic categorical structure of $\mathbf{SGrph}$ carries over largely unchanged to $\mathbf{SGrph}_C$, the exception being that the terminal object is now the identity $1_C : C \to C$ on C. Given contact maps $h_i : G_i \to C$, the pull-back of $h_1 \xrightarrow{f_{13}} h_3 \xleftarrow{f_{23}} h_2$ is constructed by taking the pull-back in $\mathbf{SGrph}$

$$\begin{tikzcd}
G_0 \arrow[r, "f_{02}"] \arrow[d, "f_{01}"'] \arrow[dr, phantom, "\lrcorner", very near start] & G_2 \arrow[d, "f_{23}"] \\
G_1 \arrow[r, "f_{13}"'] & G_3
\end{tikzcd}$$

and defining $h_0 := h_1 \circ f_{01} = h_2 \circ f_{02}$. This is well-defined since, for any agent, site or state x of G_0, $f_{13}(f_{01}(x)) = f_{23}(f_{02}(x))$ and so $h_1(f_{01}(x)) = h_3(f_{13}(f_{01}(x))) = h_3(f_{23}(f_{02}(x))) = h_2(f_{02}(x))$. The push-outs of $\mathbf{SGrph}$ carry over to $\mathbf{SGrph}_C$ in analogous fashion.

2.2.2 The subcategory $\mathbf{rSGrph}_C$

The situation is rather more interesting in $\mathbf{rSGrph}_C$, the full subcategory of $\mathbf{SGrph}_C$ containing only (contact maps from) realizable site graphs.

An arrow of $\mathbf{rSGrph}_C$ is still a mono if, and only if, its three constituent maps are all injective. However, the characterization of epis requires some care. Specifically, an arrow $f : h_1 \to h_2$ is an epi if, and only if, every connected component of G_2 contains at least one agent in the image of f. This follows from the following *rigidity* lemma that depends on G_1 and G_2 being realizable and their contact maps h_1 and h_2 being locally injective.

Lemma [rigidity] Let $h_1 : G_1 \to C$ and $h_2 : G_2 \to C$ be objects of $\mathbf{rSGrph}_C$. If G_1 is connected then the least partial function $f_{\mathcal{A}} : \mathcal{A}_{G_1} \rightharpoonup \mathcal{A}_{G_2}$ sending a_1 to a_2 extends to at most one arrow $f : h_1 \to h_2$ of $\mathbf{rSGrph}_C$.

Proof. The proof is iterative; for the base case, we need $a_2 \in h_2^{-1}(h_1(a_1))$. There is then, by local injectivity of h_1 and h_2, at most one way to define $f_{\mathcal{S}}$ on a_1's sites; and, by injectivity of $\mathcal{E}_{G_2}$, at most one way to define $f_{\mathcal{E}}$ on a_1's states. If all this succeeds and all stubs of a_1 are also preserved, we have a 'partial homomorphism' f defined on a_1. We now iterate, assuming such a partial f that preserves all links between the agents of $\mathsf{dom}(f_{\mathcal{A}})$. Let us consider any $a_1 \in \mathcal{A}_{G_1} - \mathsf{dom}(f_{\mathcal{A}})$ having links to a non-empty set S of sites in $\mathsf{dom}(f_{\mathcal{S}})$. Since G_2 is realizable, there can be at most one $a_2 \in h_2^{-1}(h_1(a_1))$ (possibly *already* in the image of f) that has links to all the $s \in f_{\mathcal{S}}(S)$. If this a_2 exists then, by local injectivity of h_1 and h_2 and by injectivity of $\mathcal{E}_{G_2}$, there is at most one way to extend $f_{\mathcal{S}}$ and $f_{\mathcal{E}}$. If all stubs of a_1 are preserved in a_2, we continue; otherwise f does not extend to any homomorphism.

The point of this lemma is that, given a 'seed', it extends in at most one way to a homomorphism. This is a synergistic consequence of realizability of the G_is and local injectivity of the contact maps; dropping either of these constraints immediately invalidates rigidity. An important consequence of rigidity is that, for any arrow $f_2 : h_2 \to h_3$ of $\mathbf{rSGrph}_C$, if we know how just one agent of each connected component of G_2 maps into G_3, we know the whole mapping f_2. Therefore, $f_1 : h_1 \to h_2$ is an epi if, and only if, at least one agent of each connected component of G_2 is in its image.

Another useful consequence of rigidity is that any arrow $f : h_1 \to h_2$ of $\mathbf{rSGrph}_C$ decomposes uniquely (up to automorphisms) into an epi $f' : h_1 \twoheadrightarrow h_2'$, where $h_2' : G_2' \to C$, and the unique arrow $!_{G_2''} : \mathbf{0} \to G_2''$ from the initial object:

$$\begin{tikzcd}
G_1 \arrow[r, "\cong"] \arrow[d, "f"'] & G_1 + \mathbf{0} \arrow[d, "f' + !_{G_2''}"] \\
G_2 \arrow[r, "\cong"] & G_2' + G_2''
\end{tikzcd}$$

There is a special class of objects $h : G \to C$ in $\mathbf{rSGrph}_C$, which we call *mixtures*, characterized as those h that are

- *locally surjective*: every agent a of G displays the same sites as its counterpart $h_{\mathcal{A}}(a)$ in C;
- *definite*: if $s \notin \lambda_G$ then $h_{\mathcal{S}}(s) \notin \lambda_C$; and if $\varepsilon_G^{-1}(s) = \emptyset$ then $\varepsilon_C^{-1}(h_{\mathcal{S}}(s)) = \emptyset$.

In words, a mixture is a site graph where every agent displays every site it possibly can, if a site has a state in C then it must also in G and if a site has a stub and/or incident edge in C then it must have one or the other in G too. In effect, a mixture is a *fully-specified* site graph with respect to C.

2.2.3 Change of contact graph

Given a homomorphism $h : C \to C'$, we can define functors between the slice categories $\mathbf{SGrph}/C$ and $\mathbf{SGrph}/C'$. The mapping from $\mathbf{SGrph}/C$ to $\mathbf{SGrph}/C'$ is immediate:

- an object $h_1 : G_1 \to C$ becomes $h_*(h_1) := h \circ h_1 : G'_1 \to C'$;
- an arrow $f : h_1 \to h_2$ becomes $h_*(f) := f : h \circ h_1 \to h \circ h_2$.

The reverse mapping relies on the existence of pull-backs in $\mathbf{SGrph}$:

- an object $h'_1 : G'_1 \to C'$ becomes $h^*(h'_1) := h_1 : G_1 \to C$ as defined by the pull-back

$$\begin{tikzcd}
G_1 \arrow[r, "h'"] \arrow[d, "h_1"'] \arrow[dr, phantom, "\lrcorner", very near start] & G'_1 \arrow[d, "h'_1"] \\
C \arrow[r, "h"'] & C'
\end{tikzcd}$$

- an arrow $f' : h'_1 \to h'_2$ becomes $h^*(f') := f : h_1 \to h_2$

$$\begin{tikzcd}
G_1 \arrow[dr, dashed, "f"] \arrow[drr, bend left, "f' \circ h'"] \arrow[ddr, bend right, "h_1"'] & & \\
& G_2 \arrow[r, "h''"] \arrow[d, "h_2"] \arrow[dr, phantom, "\lrcorner", very near start] & G'_2 \arrow[d, "h'_2"] \\
& C \arrow[r, "h"'] & C'
\end{tikzcd}$$

by applying universality of the pull-back to the outer commuting square.

It is then straightforward to show that $h_* : \mathbf{SGrph}/C \to \mathbf{SGrph}/C'$ is left adjoint to $h^* : \mathbf{SGrph}/C' \to \mathbf{SGrph}/C$. The right adjoint h^* can be used to re-visualize a site graph according to the contact graph C rather than C'. As we will see later, this can be exploited to verify dynamically whether or not an instance of a rule respects its declared type.

It is important to note that the h^* functor does *not* in general restrict to a functor from $\mathbf{rSGrph}/C'$ to $\mathbf{rSGrph}/C$: the pull-back of h and h'_1—even from realizable G'_1—need not yield realizable G_1. However, if h is injective on sites then, since pull-backs preserve monos, h' and h'' are both injective on sites, so G_1 and G_2 are realizable. Moreover, if f' is an arrow of $\mathbf{rSGrph}$ then $f' \circ h'$ is injective on sites, so f is also injective on sites and thus an arrow of $\mathbf{rSGrph}$. Finally, if we also have that h'_1 and h'_2 are locally injective on sites, then clearly h_1 and h_2 are also locally injective on sites, *i.e.* they are objects of $\mathbf{rSGrph}_C$. In summary, provided h is injective on sites, h^* does restrict to a functor from $\mathbf{rSGrph}/C'$ to $\mathbf{rSGrph}/C$ and, if additionally h'_1 and h'_2 are locally injective on sites, then h^* further restricts to a functor from $\mathbf{rSGrph}_{C'}$ to $\mathbf{rSGrph}_C$.

More generally, if h is V-preserving then so are h' and h'' whereupon G_1 and G_2 are both realizable. Moreover, if h'_1 and h'_2 are locally injective on sites then h_1 and h_2 also are, *i.e.* they are objects of $\mathbf{rSGrph}_C$. However, $f' \circ h'$ need not be injective on sites so, in general, neither is f.

2.3 Rewriting site graphs

In this section, we introduce *actions*, the semantic analogue of action scripts, and use them to define rewriting of site graphs in the double-push-out (DPO) style. This is largely a straightforward exercise but does require a little care to identify an appropriate class of actions for which DPO rewriting works correctly.

2.3.1 Actions as spans

The most general possible definition of action is as a span in $\mathbf{SGrph}$

$$\begin{tikzcd}
& G_0 \arrow[dl, hook'] \arrow[dr] & \\
G_\ell & & G_r
\end{tikzcd}$$

with intuitive reading that G_ℓ is rewritten into G_r while the common part of G_ℓ and G_r matched by G_0 remains unchanged. Indeed, this is an instance of the general approach of building (bi-)categories of partial maps out of categories of total maps [8].

It would be possible to develop a theory of 'untyped' site graph rewriting by restricting actions to a smaller class. For the purposes of this paper, however, we prefer to work in the typed setting of $\mathbf{rSGrph}_C$ from the outset as this has the advantage of coming with an intrinsic notion of mixture. Moreover, a span $h_\ell \xleftarrow{\alpha_\ell} h_0 \xrightarrow{\alpha_r} h_r$ in $\mathbf{rSGrph}_C$ comes with the additional guarantee that $h_\ell \circ \alpha_\ell = h_0 = h_r \circ \alpha_r$ meaning that the contact information of everything in G_0 is the same on both sides of the span.

It makes sense to additionally ask for α_r to be a mono, as we do not wish to express actions that 'merge' agents, and for the span to be *extremal*: if there is an epi $\alpha_0 : h_0 \twoheadrightarrow h'_0$ and arrows $\alpha'_\ell : h'_0 \to h_\ell$ and $\alpha'_r : h'_0 \to h_r$ making

$$\begin{tikzcd}
 & h_0 \arrow[dl, bend right, "\alpha_\ell"'] \arrow[d, two heads, "\alpha_0"] \arrow[dr, bend left, "\alpha_r"] & \\
 & h'_0 \arrow[dl, "\alpha'_\ell"] \arrow[dr, "\alpha'_r"'] & \\
h_\ell & & h_r
\end{tikzcd}$$

commute then α_0 is an isomorphism. Intuitively, this means that nothing more can be added to h_0, *i.e.* h_ℓ and h_r consist of an exact copy of h_0 plus (possibly) additional connected components. Note that any span from the empty site graph $\mathbf{0}$ is extremal.

2.3.2 Double push-out rewriting

Let us now investigate the class of actions that we can use in order to express site graph rewriting as an instance of the general double-push-out (DPO) approach. The essential difficulty of DPO rewriting comes in the first step where, given the left leg of the span and a matching into some mixture M, we must "complete the push-out" via an intermediate M_0:

$$\begin{tikzcd}
 & G_0 \arrow[dl, "\alpha_\ell"'] & & & & G_0 \arrow[dl, "\alpha_\ell"'] \arrow[d] \\
G_\ell \arrow[d, "m"'] & & \arrow[r, squiggly] & & G_\ell \arrow[d, "m"'] & M_0 \arrow[dl] \\
M & & & & M &
\end{tikzcd}$$

(To lighten notation, we write just G_0, *et c.*, rather than the more accurate $h_0 : G_0 \to C$.)

Let us first consider the special case where G_0 is the empty site graph $\mathbf{0}$ which corresponds to a situation where G_ℓ is to be completely excised from M and replaced by G_r—since G_0 is everything of G_ℓ to be preserved by the action. In this case, since a push-out from an initial object is always a co-product, we know that, if M_0 exists, then $M \cong G_\ell + M_0$. In effect, this means that G_ℓ does not merely match M but that it is literally contained in M; so an action that removes and/or adds agents can only remove and/or add entire connected components of/to M, *i.e.* mixtures with respect to the ambient contact graph C:

$$M \;\cong\; G_\ell + M_0 \;\rightsquigarrow\; G_r + M_0 \;\cong\; M'$$

This restriction enforces a 'no side-effects' condition that guarantees that all things in M that are modified by the action are explicitly mentioned by the action, just as is the case for multi-set rewriting.

In the general case, where G_0 may be non-empty, we first decompose α_ℓ uniquely (up to isomorphism) into an epi $\alpha'_\ell : G_0 \twoheadrightarrow G'_\ell$ and $!_{G''_\ell} : \mathbf{0} \to G''_\ell$, so that $G_\ell \cong G'_\ell + G''_\ell$. It is then sufficient to be able to complete the push-out for α'_ℓ and the restriction $m' := m \circ \iota_{G'_\ell}$ of m to G'_ℓ; the missing mixture G''_ℓ can be dealt with afterwards as per the above special case.

We construct an M_0 by setting $\mathcal{A}_{M_0} := \mathcal{A}_M$, $\mathcal{S}_{M_0} := \mathcal{S}_M$ and defining the interesting part of its structure according to the standard DPO-style prescription of 'add to G_0 everything that is in M but not in G'_ℓ':

- $\mathcal{E}_{M_0} := \mathcal{E}_M - (\mathcal{E}_{G'_\ell} - \mathcal{E}_{G_0})$;
- $\lambda_{M_0} := \lambda_M - (\lambda_{G'_\ell} - \lambda_{G_0})$.

We cannot define $\mathcal{A}_{M_0}$ or $\mathcal{S}_{M_0}$ in this more subtle way as this could lead to 'orphaned' sites, states and links, *e.g.* a site might not be attached to any agent. It is easy to see that this always gives rise to a commuting square with the inclusions from G_0 to M_0 and M_0 to M. However, unless α'_ℓ is *surjective on agents and sites*, this square cannot be a push-out in general since any agent or site of G'_ℓ not also in G_0 will be duplicated by the push-out.

Intuitively, this condition enforces the idea that G_0 contains everything that is preserved by the action; in particular, the existence of the agents and sites of G'_ℓ is not in question. This condition plays an analogous role to the *dangling* condition in DPO rewriting of normal graphs although, rather than using it as a means to reject certain matchings, we instead use it to constrain our notion of valid action.

Technically speaking, now that this push-out has been constructed, it is always possible to complete the DPO rewrite:

$$\begin{tikzcd}
 & G_0 \arrow[dl, tail, "\alpha'_\ell"'] \arrow[d] & & & & G_0 \arrow[dl, tail, "\alpha'_\ell"'] \arrow[d] \arrow[dr, tail] & \\
G'_\ell \arrow[d, "m'"'] & M_0 \arrow[dl] & \arrow[r, squiggly] & {} & G'_\ell \arrow[d, "m'"'] & M_0 \arrow[dl] \arrow[dr] & G'_r \arrow[d] \\
M_\ell & & & & M_\ell & & M_r
\end{tikzcd}$$

However, in order for M_r to be a mixture, we need some additional properties of α_r. Firstly, that it is also surjective on agents and sites so that there is a bijection between the agents and sites of G'_ℓ and G'_r. Secondly, for all sites s of G'_ℓ, $s \notin \lambda_{G_\ell}$ if, and only if, $s \notin \lambda_{G_r}$ (where we have slightly abusively reused s to denote the counterpart of s in G'_r). In other words, unspecified binding statuses must be preserved by actions.

These conditions rule out non-deterministic rules, *e.g.* sending $s \notin \lambda_{G_\ell}$ to $s \ \lambda_{G_r} \star$, and guarantee that actions are always *reversible*, *i.e.* the span

$$\begin{tikzcd}
 & G_0 \arrow[dl, tail, "\alpha'_r"'] \arrow[dr, tail, "\alpha'_\ell"] & \\
G'_r & & G'_\ell
\end{tikzcd}$$

is also an action—which is in fact just another way of saying that actions cannot have side-effects. Indeed, in full Kappa, there are only two kinds of rule that violate reversibility—deletion of only part of a connected component; and undoing a wild-card-binding—both of which also induce side-effects (due to the implicit deletion of links not mentioned by the action). Our restriction on actions rules out the former and the latter cannot even be expressed with the present formulation of site graphs; so our formalism allows us to express only side-effect-free Kappa.

In summary, a *valid action*, or *rule*, is an action where α'_ℓ and α'_r are bijections on agents and sites, the eliminated G''_ℓ and introduced G''_r are mixtures and where sites with unspecified binding status must be preserved by the rule and its reverse. We should stress that these are sufficient, but not necessary, conditions on actions for them to be compatible with DPO-style rewriting; more general conditions could undoubtedly be formulated but, for our present purposes, they strike a convenient balance that is sufficient for side-effect-free Kappa since all actions expressible therein are valid.

2.3.3 Static constraints on actions

The explicit typing of site graphs with a contact graph allows us to express certain invariants as static constraints on rules. For example, the rule

$$r_1 := A(s^1), B(s^1) \longrightarrow A(s^\bullet), B(s^\bullet)$$

cannot be expressed as a span over the contact graph

$$T_1 := A(s^{1,\bullet}), B(s^{1,2}), C(s^{2,\bullet})$$

since its RHS does not respect the typing constraints of T_1. Specifically, the lack of a stub on B's site s means that s must always be bound, preventing a rule such as r_1 that destroys this invariant. A careful choice of contact graph can therefore enforce fairly subtle constraints, *e.g.* the 'bond displacement' rule

$$r_2 := A(s^1), B(s^1), C(s^\bullet) \longrightarrow A(s^\bullet), B(s^1), C(s^1)$$

can be expressed as a span over T_1 since it does preserve the invariant.

Note that this use of contact graphs only determines the validity, or otherwise, of a rule with respect to a type: the rule is either accepted or rejected. In the next section, we show how to use types to constrain the *dynamics* engendered by rules.

3 Rule-based dynamics

In this section, we assume a collection of rules $\mathcal{R}$ valid with respect to a contact graph C. Each rule $r \in \mathcal{R}$ has a real-valued *rate constant* k_r and may also have its own personal type C_r accompanied by a homomorphism $h_r : C_r \to C$ which we will use, in conjunction with the change of contact graph adjunction, to reject certain instances of r dynamically. However, before getting to that, we first briefly recall the standard notion of continuous-time Markov chain (CTMC) which underlies the stochastic semantics of $\mathcal{R}$.

A CTMC has a set $\mathcal{S}$ of *states* and a set $\mathcal{T}$ of *transitions* between distinct states, *i.e.* no self-loops. Each transition τ from s is assigned a real number $\mathcal{A}_\tau(s)$ known as its *activity*; each state s acquires a total activity, defined as the sum $\mathcal{A}(s) := \sum_\tau \mathcal{A}_\tau(s)$ of the activities of all its outgoing transitions.

The dynamics of a CTMC depends on the waiting time in, and transition probabilities from, each state s. The former is described by the exponential random variable $p(\delta t) := \mathcal{A}(s) \cdot e^{-\mathcal{A}(s)\delta t}$, the minimum of the family $p_\tau(\delta t) := \mathcal{A}_\tau(s) \cdot e^{-\mathcal{A}_\tau(s)\delta t}$ of independent random variables, *i.e.* the time until the first transition; and, for the latter, each transition τ from s naturally acquires the probability $\mathcal{A}_\tau(s)/\mathcal{A}(s)$ of being chosen.

3.1 The stochastic semantics of $\mathcal{R}$

In order to instantiate this general scheme to the specific case of our system $\mathcal{R}$ of rules, we must define the states, transitions and activities thereof that $\mathcal{R}$ induces. It turns out that the states are straightforward to define—they are just mixtures with respect to the contact graph C—but the definition of transitions is more subtle.

To see why, let us first consider the world of multi-set rewriting, *i.e.* reactions operating on a multi-set of named, structureless molecular species. In this setting, the notion of *event*, or transition, is fairly straightforward: the choice of a multi-set of reactants that are to be replaced by the multi-set of products. The only subtle point comes about when dealing with reactions having symmetries between reactants, *e.g.* $A + A \longrightarrow A_2$; is an event an ordered or an unordered pair of As?

The answer, conceptually, depends on whether the reaction is intended to represent the formation of an asymmetric or a symmetric dimer, the former proceeding twice as fast as the latter. However, the relative paucity of the formal language of reactions makes it impossible to express this distinction syntactically. The difference between these two reactions must therefore be encoded in the rate constant—given a semantic convention fixing whether one considers an event as an ordered or an unordered pair by default.

In the case of rules, this question acquires new potency since we are now working in a far richer syntactic medium: not only do 'molecular species', or *complexes*, have internal structure—agents, sites, links, *et c.*—but the rules need only partially specify those complexes. In particular, we can now express the distinction between *s*ymmetric and *a*symmetric homodimerization:

$$\begin{aligned} r_s &:= A(s^\bullet), A(s^\bullet) \longrightarrow A(s^1), A(s^1) \\ r_a &:= A(\ell^\bullet, r^\bullet), A(\ell^\bullet, r^\bullet) \longrightarrow A(\ell^\bullet, r^1), A(\ell^1, r^\bullet) \end{aligned}$$

They each have a non-trivial automorphism of their LHS, suggesting perhaps that our notion of event should be that of an unordered pair. However, what happens if either rule is matched to a pair of non-isomorphic complexes? Or, indeed, if an asymmetric rule is applied in a symmetric context?

Ultimately, the question being posed is: when should two matchings of the LHS of a rule into a mixture be considered indistinguishable *from the rule's point of view*? Any non-trivial automorphism of that LHS *may* give rise to such indistinguishability—but only if it survives the action of the rule. If an automorphism is destroyed by the action, the two matchings can be distinguished *post hoc* and correspond to two distinct 'reaction centres'.

In the case of r_s and r_a above, r_s preserves the non-trivial symmetry and, as such, defines a *symmetric* binding mechanism which cannot distinguish between two complexes that match it, even if they are actually *different*. On the other hand, r_a breaks the symmetry and, as such, defines an *asymmetric* binding mechanism that induces two distinct reaction centres and can even distinguish *identical* complexes that match it.

In general, for a rule r, we therefore define an *r-event* to be a matching of the LHS of r up to automorphisms of the LHS preserved by the action of r, *cf.* [1]. We write $\mathcal{E}_r(M)$ for the set of all r-events in the mixture M; this is always a finite set. Note that any matchings identified by this quotient necessarily provoke *exactly the same* rewrite of the mixture; moreover, any other matching would perform a different microscopic rewrite, so our notion of event is the coarsest possible quotienting of matchings that ensures that an event induces a unique microscopic transition.

We can finally complete our description of the CTMC defined by $\mathcal{R}$. The set of transitions from a mixture M is the union $\mathcal{T} := \bigcup_r \mathcal{E}_r(M)$ of all events in M. The activity of each $r \in \mathcal{R}$ in M is defined as $\mathcal{A}_r(M) := k_r \cdot |\Phi_r(M)|$ so that the activity of any given r-event is simply k_r. This convention for activity is known as the *mass-action rate law*.

3.2 Dynamic rejection of events

We now turn to the question of rejecting events dynamically. We write r_L (resp. r_R, r_P) for the LHS (resp. RHS, preserved zone) of $r \in \mathcal{R}$; $h_L : r_L \to C$, $h_P : r_P \to C$ and $h_R : r_R \to C$ for the static typing of r by the contact graph C; and $\alpha_L : h_P \rightarrowtail h_L$ and $\alpha_R : h_P \rightarrowtail h_R$ for the action of r.

We also assume that each $r \in \mathcal{R}$ is accompanied by a finite collection of refined types $C_{r,i}$, each with a mono $h_{r,i} : C_{r,i} \rightarrowtail C$ to the overall contact graph C. We require that, for each $h_{r,i}$, there exist arrows $h_{L,i} : r_L \to C_{r,i}$, $h_{P,i} : r_P \to C_{r,i}$ and $h_{R,i} : r_R \to C_{r,i}$ making the obvious triangles commute:

$$\begin{tikzcd} & r_L \arrow[dl, "h_{L,i}"'] \arrow[dr, "h_L"] & \\ C_{r,i} \arrow[rr, "h_{r,i}"'] & & C \end{tikzcd} \qquad \begin{tikzcd} & r_P \arrow[dl, "h_{P,i}"'] \arrow[dr, "h_P"] & \\ C_{r,i} \arrow[rr, "h_{r,i}"'] & & C \end{tikzcd} \qquad \begin{tikzcd} & r_R \arrow[dl, "h_{R,i}"'] \arrow[dr, "h_R"] & \\ C_{r,i} \arrow[rr, "h_{r,i}"'] & & C \end{tikzcd}$$

This ensures that $C_{r,i}$ is still a valid static type for r. Each $h_{r,i}$ thus specifies a (potentially) more stringent requirement for r to be applied. The idea is that an r-event will be accepted if, and only if, it respects *at least one* of the $C_{r,i}$s. Note that this is a property of an *instance* of r, not of r itself; it is verified with the change of contact graph adjunction.

In order to define precisely what we mean by an instance of the rule r, recall that a matching $f : r_L \to M$ of r into a mixture M decomposes into an epi $f_M : r_L \twoheadrightarrow M_{r_L}$ and an arrow from the initial object $\mathbf{0}$ to the rest of M (which we ignore); for the sake of readability, we are writing r_L rather than the more correct h_L. We call the graph M_{r_L} a *ground refinement* of r_L. We obtain the ground refinements, M_{r_P} and M_{r_R}, of r_P and r_R by the DPO-rewriting of M_{r_L} according to the action of r. We obtain a new action defined by $\alpha_{L,f} : M_{r_P} \rightarrowtail M_{r_L}$ and $\alpha_{R,f} : M_{r_P} \rightarrowtail M_{r_R}$. In effect, we slide the rule r along the matching f to obtain a specific, fully-instantiated instance of r in the mixture M.

For each refined type $C_{r,i}$, we now use $h^*_{r,i}$ to change contact graph from C to $C_{r,i}$:

$$\begin{tikzcd}
 & & & M_{P,i} \arrow[dll, dashed, "\alpha_{L,i}"'] \arrow[drr, dashed, "\alpha_{R,i}"] & & & \\
M_{r_L} \arrow[d] & M_{L,i} \arrow[l] \arrow[d] \arrow[dl, phantom, "\llcorner", very near start] & & & & M_{R,i} \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & M_{r_R} \arrow[d] \\
C & C_{r,i} \arrow[l, "h_{r,i}"] & & & & C_{r,i} \arrow[r, "h_{r,i}"'] & C
\end{tikzcd}$$

Since $h_{r,i}$ is a mono, $M_{L,i}$, $M_{R,i}$ and $M_{P,i}$ are realizable. The resulting span $M_{L,i} \longleftarrow M_{P,i} \longrightarrow M_{R,i}$ defines an action that *refines* the original rule r, *i.e.* it describes the same rewrite as r but with a (potentially) more stringent test which matches fewer mixtures with respect to C. However, in general, one or other (or both) of $M_{L,i}$ and $M_{R,i}$ need not be a mixture with respect to $C_{r,i}$; such a refinement does not respect the typing constraint $C_{r,i}$. If, for some i, $M_{L,i}$ and $M_{R,i}$ are *both* mixtures with respect to $C_{r,i}$ then we say that the (original) matching *satisfies* the refined type $C_{r,i}$; otherwise it is a *null event*.

For example, consider the rule r

$$r := A(d^\bullet), D(a^\bullet) \longrightarrow A(d^1), D(a^1)$$

with contact graph C, further constrained by contact graphs C_1 and C_2:

$$\begin{array}{rcl}
C & := & A(d^{1,\bullet}, b^{2,3,\bullet}), B_1(a^{2,\bullet}), D(a^{1,\bullet}), B_2(a^{3,\bullet}) \\
C_1 & := & A(d^\bullet, b^1), B_1(a^{1,\bullet}), D(a^\bullet) \\
C_2 & := & A(d^{2,\bullet}, b^{3,\bullet}), D(a^{2,\bullet}), B_2(a^{3,\bullet}).
\end{array}$$

The ground refinement

$$M_r := A(d^\bullet, b^1), B_1(a^1), D(a^\bullet) \longrightarrow A(d^2, b^1), B_1(a^1), D(a^2)$$

is a null event as neither $M_{L,2}$ nor $M_{R,2}$ respects C_2 (because C_2 does not contain B_1) and, although $M_{L,1}$ respects $C_{r,1}$, in $M_{R,1}$ as defined by the pull-back

$$\begin{tikzcd} A(d, b^1), B_1(a^1), D(a) \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & M_{r_R} \arrow[d] \\ C_{r,1} \arrow[r] & C \end{tikzcd}$$

A's site d and D's site a have opposing binding statuses in C_1 and M_{r_R} and, as such, acquire unspecified binding status in the pull-back; so $M_{R,1}$ is not a mixture with respect to $C_{r,1}$. Intuitively, $C_{r,1}$ and $C_{r,2}$ express the logical constraint that 'B_1, but not B_2, blocks D's access to A', a phenomenon known to biophysicists as *steric occlusion* which arises when (i) one protein physically blocks the access of a second to its desired binding site; and (ii) the second protein could have bound if the first were not there.

We could alternatively use the $C_{r,i}$s as a *growth policy* [7] to generate statically the collection of rules that refine r and respect the $C_{r,i}$s. We prefer not to do this since, in general, it leads to an explosion in the number of rules and, as we will see in the next section, we can easily obtain the same effect through dynamic rejection of events. However, the principal advantage of enforcing constraints dynamically, rather than refining rules explicitly is that it allows us to make a clean separation of the *essential* mechanism from more incidental problems such as steric occlusion: it is not the fault of the binding mechanism if it fails *only* because some other agent is 'in the way'. Of course, rules *can* always be massaged to enforce such constraints but such rules are fragile and difficult to modify and/or incorporate into larger rule sets that may not be aware of its 'built in' assumptions. It seems more prudent, and scalable, to keep the rule simple and document its steric demands, *et c.*, separately.

3.3 Overestimating activity

In the previous section, we have seen how certain transitions of our CTMC built out of $\mathcal{R}$ may be rejected during simulation. When this happens, the state s of the CTMC does not change—it is as if there were a self-loop—so, in particular, the activity $\mathcal{A}_r(s)$ of each rule r and the total activity $\mathcal{A}(s)$ remain unchanged.

This requires a modification of the dynamics of the CTMC since the existence of a self-loop tends to increase the waiting time in that state. Specifically, the waiting time in state s becomes the time until a real event occurs. It is well-known that the time for $n \geq 1$ events to occur in a Poisson process is described by the Gamma random variable

$$\Gamma_{n,\mathcal{A}(s)}(\delta t) = (\mathcal{A}(s)^n \cdot \delta t^{n-1}/(n-1)!) \cdot e^{-\mathcal{A}(s)\delta t}$$

so, if the probability of a null event occurring in state s is $q(s)$, the total waiting time in state s is distributed as

$$p(\delta t) = \sum_{n=0}^{\infty} q(s)^n (1 - q(s)) \cdot \Gamma_{n+1,\mathcal{A}(s)}(\delta t).$$

In the case of our CTMC derived from $\mathcal{R}$, the probability q_M for each state M is not known statically; however, whenever a transition t from M is chosen, we can then detect whether or not it is a null event. Given that the chosen event is *some* r-event, the fact that r-events are chosen uniformly at random means that the probability $q_r(M)$ that 'the chosen r-event is a null event' is just the number of null r-events divided by the total number $|\mathcal{E}_r(M)|$ of r-events. The overall probabilility that 'the chosen event is a real r-event' is therefore $\mathcal{A}'_r(M)/\mathcal{A}(M)$. In effect, M has a *true* activity $\mathcal{A}'(M) = \sum_r (1 - q_r(M)) \cdot \mathcal{A}_r(M)$ underestimating its usual activity.

It follows that the probability that 'the chosen event is null' is $q(M) = \sum_r q_r(M) \cdot \mathcal{A}_r(M)/\mathcal{A}(M) = (\mathcal{A}(M) - \mathcal{A}'(M))/\mathcal{A}(M)$ and the probability that 'the next real event is an r-event' is $(\mathcal{A}'_r(M)/\mathcal{A}(M))/(\mathcal{A}'(M)/\mathcal{A}(M) = \mathcal{A}'_r(M)/\mathcal{A}'(M)$. Finally, we instantiate the above equation to obtain

$$\begin{aligned} p(\delta t) &= \mathcal{A}'(M) \cdot e^{-\mathcal{A}(M)\delta t} \cdot \sum_{n=0}^{\infty} (\mathcal{A}(M) - \mathcal{A}'(M))^n \cdot \delta t^n / n! \\ &= \mathcal{A}'(M) \cdot e^{-\mathcal{A}'(M)\delta t}. \end{aligned}$$

In summary, the waiting time in, and transition probabilities from, state M depend only on the (unknown!) *true* activities $\mathcal{A}'_r(M)$ and so the modified CTMC behaves exactly as if it were actually the true CTMC with no self-loops obtained by statically generating all rules satisfying the growth policy C_r. This argument is a more general case of the technique for dealing with null events arising for reasons of implementation efficiency [3, 9].

4 Conclusions

In this paper, we have given the first fully semantic definition of a rule-based language which encompasses a large fragment of Kappa, the only restriction being the forbidding of side-effects. This has clarified the connection between the notion of site graph rewriting, as expressed by rules in Kappa, and more traditional graph rewriting that has long used the double-push-out technique to formalize rewriting.

We have also investigated for the first time the possibility of typing rules to express invariants hidden in a rule or enforce constraints on the dynamics engendered by a rule set. This approach should usefully complement the analyses made with abstract interpretation [4], some of which address similar issues. However, this initial investigation remains purely mathematical; the important question remains as to how a powerful and pragmatically useful type system for rules should be defined and implemented.

Acknowledgements. We would like to thank Eric Deeds, Jérôme Feret, Walter Fontana and Jean Krivine for many discussions on topics related to the subject of this paper.

References

[1] M. L. Blinov, J. Yang, J. R. Faeder, and W. S. Hlavacek. Graph theory for rule-based modeling of biochemical networks. *Lect. Notes Comput. Sci.*, 4230:89–106, 2006.

[2] V. Danos, J. Feret, W. Fontana, R. Harmer, and J. Krivine. Rule-based Modelling of Cellular Signalling. *Lecture Notes in Computer Science*, 4703:17–41, 2007.

[3] V. Danos, J. Feret, W. Fontana, and J. Krivine. Scalable Simulation of Cellular Signaling Networks. *Lecture Notes in Computer Science*, 4807:139–157, 2007.

[4] V. Danos, J. Feret, W. Fontana, and J. Krivine. Abstract Interpretation of Cellular Signalling Networks. *Lecture Notes in Computer Science*, 4905:83–97, 2008.

[5] H. Ehrig, M. Pfender, and H. Schneider. Graph-grammars: an algebraic approach. In *14th Annual Symposium on Switching and Automata Theory*, pages 167–180. IEEE, 1973.

[6] W. Hlavacek, J. Faeder, M. Blinov, R. Posner, M. Hucka, and W. Fontana. Rules for Modeling Signal-Transduction Systems. *Science's STKE*, 2006(344), 2006.

[7] E. Murphy, V. Danos, J. Feret, R. Harmer, and J. Krivine. Rule-based modelling and model refinement. *Elements of Computational Systems Biology. Wiley Book Series on Bioinformatics*, 2009.

[8] E. Robinson and G. Rosolini. Categories of partial maps. *Information and computation*, 79(2):95–130, 1988.

[9] J. Yang, M. Monine, J. Faeder, and W. Hlavacek. Kinetic Monte Carlo method for rule-based modeling of biochemical networks. *Physical Review E*, 78(3):31910, 2008.