Programs as models:
Kappa language basics

Jean Krivine[1,4], Vincent Danos[2,4], Jerome Feret[1,3,4], Russ Harmer[3,4] and Walter Fontana[†1,4]

[1]Department of Systems Biology, Harvard Medical School, 200 Longwood Avenue, Boston MA 02115, USA
[2]School of Informatics, University of Edinburgh, Edinburgh, UK
[3]École Normale Supérieure, 45 rue d'Ulm, F-75230 Paris Cedex 05, France
[4]Plectix BioSystems, Inc., 20 Holland Street, Suite 400, Somerville, MA 02144, USA

Email: Jean Krivine - krivine@lix.polytechnique.fr; Vincent Danos - vincent.danos@gmail.com; Jerome Feret - jerome_feret@hms.harvard.edu; Russ Harmer - russ@pps.jussieu.fr; Walter Fontana[†]- walter@hms.harvard.edu;

[†]Corresponding author

Version 1.0 [October 24th, 2008]

Contents

Einen Satz verstehen, heißt eine Sprache verstehen.
Eine Sprache verstehen, heißt eine Technik beherrschen.

To understand a sentence is to understand a language.
To understand a language is to be master of a technique.

Ludwig Wittgenstein
Philosophische Untersuchungen, §199

1 Executive Summary

Kappa is a formal language for defining agents (typically meant to represent proteins) as sets of sites that constitute abstract resources for interaction. Sites can hold internal state, as generated through post-translational modifications, and engage in binding relations with sites of other agents. In this language, a possibly temporary association of proteins is a connected graph, called a complex (of agents). The nodes of the graph are agents, but the endpoints of edges are sites, which belong to agents. While an agent can bear many connections, a site can bear only one.

Kappa enables the expression of *rules* of protein-protein interaction at a simple yet meaningful level of *tunable resolution*. The idea of a rule is to specify only the molecular context required for an interaction, along with a rate constant. Therein lies the power of rules to control combinatorial complexity. The left hand side (lhs) of a rule defines a pattern consisting of agents for which some sites and/or states have been omitted. The right hand side (rhs) exhibits the modifications that occur when the lhs pattern is matched in a mixture of agents. The difference between rhs and lhs is called the *action* of the rule. Sites mentioned on the lhs are said to be *tested* by the rule. Sites that are tested but not modified are also referred to as the *context* of a rule. Rules expand by pattern matching into potentially numerous reaction instances (whose rate constants are related to the rate constant of the rule) that involve particular combinations of molecular species, each of which realizes the context required for the rule to apply.

Kappa-rules stand in analogy to reaction rules in organic chemistry, where aspects of molecules that are irrelevant to a chemical rearrangement are designated as “remainder” groups. In Kappa, they are simply not mentioned. While chemistry has a theoretical foundation for rationalizing rules of reaction, Kappa-rules only codify observations, not why these observations might make sense to a structural biologist or biochemist.

Kappa-rules are not only descriptive, they also are executable, and induce a stochastic dynamics on a mixture of agents (implemented by a suitably generalized Gillespie algorithm). A Kappa model of a biological system is a collection of rules (with rate constants) and an initial mixture of agents on which these rules act. This makes a Kappa model akin to a concurrent computer program whose instructions are rules that asynchronously change the state of a store represented by the reaction mixture. Programs are formal objects that can be analyzed statically. *Static analysis* is about discovering behavioral properties of a program without running it (hence the name), much like a system of differential equations can be studied without simulating it. This involves inspection of the causal dependencies among rules and approximation of the molecular states that are reachable from a given initial condition. It is the central role of static analysis that sets our framework apart from other rule-based approaches, whose goal is the automated assembly of kinetic differential equations by an iterative fixed-point construction of all reaction instances from a set of rules. The combinatorial explosion inherent in molecular signaling makes such goals impractical and often impossible. In a pilot study of EGF signaling, we collated 66 rules representing mechanistic observations of pertinent protein-protein interactions. These rules would produce 10^{19} molecular species. Our current EGF model has grown to about 350 rules. It thus appears more useful to forgo the expansion into an inscrutably large system of equations, and instead apply static analysis techniques directly to the rule collection. To complement the static analysis, rules can also drive a stochastic simulation.

A rule-based model affords the inclusion of a wealth of information about protein-protein interactions, without the need to prematurely hypothesize which molecular states are or are not relevant to the dynamics of a system of interest.

2 Kappa: A basic language for molecular biology

2.1 Taking a hint from chemistry: Rules, instances, and events

Although the foundations of chemistry are built on quantum mechanics, the vast body of chemical knowledge is not expressed in terms of wave functions. Rather, it is expressed in terms of a formal language representing the compositional structure of molecules, permitting the specification of exactly which parts are changing in a chemical reaction. Reaction schemes are usually codified in terms of patterns. For example, an aldol condensation (Figure 1A) is a scheme in which a carbonyl group reacts with a ketone or aldehyde to produce an α, β-unsaturated ketone or aldehyde. A reaction *rule* like the aldol condensation specifies a transformation involving particular moieties of the reactants, without spelling out the full context in which these moieties are embedded. This is typically expressed by "remainder groups", denoted by R. In contrast, Figure 1B exemplifies an *instance* of the aldol condensation rule, in which the Rs have been substituted with specific chemical parts. In this case, acetaldehyde reacts with acetone. Because the reaction instance of Figure 1B completely specifies the context of the involved functional groups, we shall refer to it as a flat or *fully contextualized reaction.* In contrast, the aldol condensation rule is a *decontextualized reaction.* A further level of detail consists in mentioning individuals, not just molecular species. At this level we specify which particular acetaldehyde and acetone molecules are involved in an actual encounter. We shall refer to this level as an *event.* An event is an important concept to which we shall return in detail later. A reaction scheme, such as in Figure 1A, may involve intermediate steps depending on temperature and pH. Some reactions might therefore be considered as "elementary" while others are composite.

To summarize, chemical knowledge is codified in terms of rules that specify how molecular moieties (such as functional groups) interact. A rule is a pattern that can be refined into many possible reaction instances by fully detailing the molecular parts that were left unspecified. An event is an actual occurrence of a reaction instance in a mixture, and thus involves individual molecules.

Our goal is to transfer the chemical rule-based perspective to molecular biology. However, molecular biology deals with high molecular weight objects that may have a large number of states and whose actions unfold in the context of numerous other kinds of molecular objects.

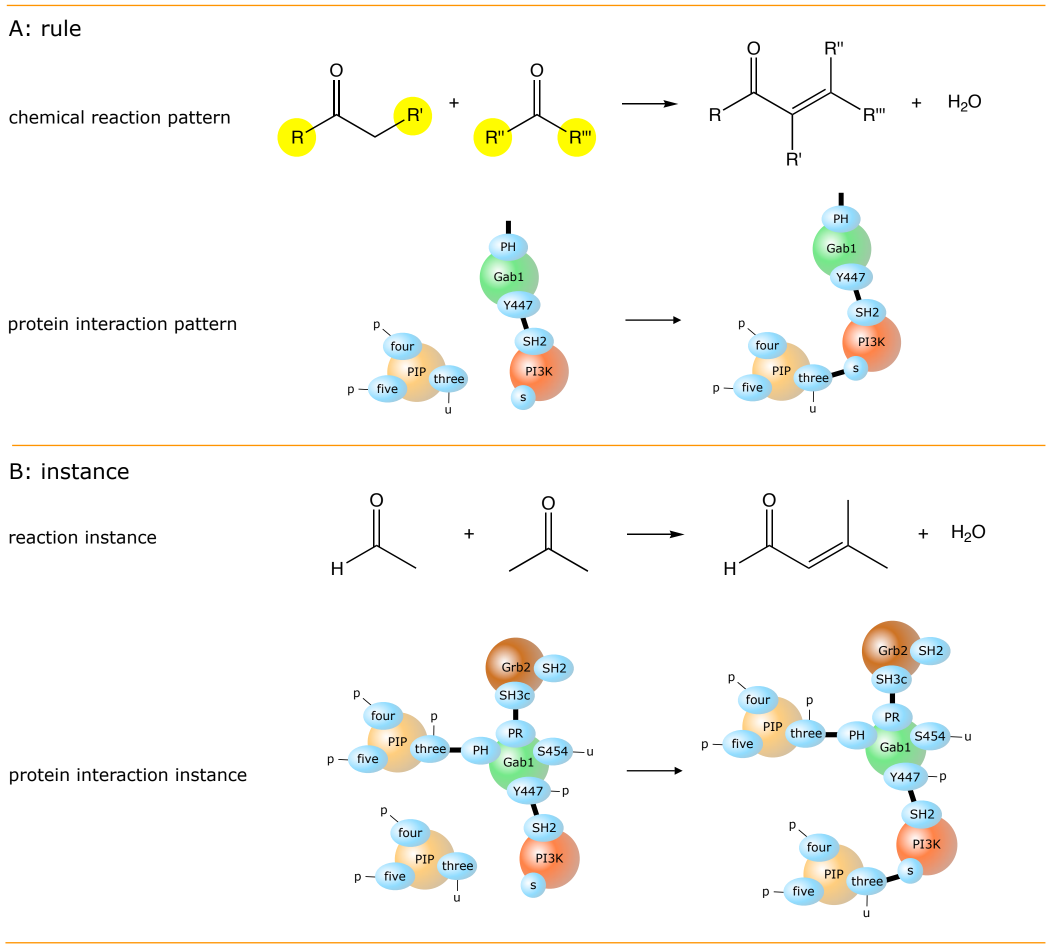


Figure 1: Specification of chemical reactions. An aldol condensation is used to illustrate the concept of a reaction rule (A), which details only those molecular parts that are relevant to a particular scheme of reaction. Because the transformation of these moieties is described independently of their contexts R, R', R", and R"' (highlighted), we refer to such a rule as a decontextualized reaction. Sometimes a generic scheme, such as (A), requires refinement into special classes defined by different types of contexts R-R"'. For example, we may wish to distinguish between an intra-molecular aldol condensation, in which R'≡R", and an inter-molecular case. B: Upon full specification of the contexts R, R', R", and R"', the rule (A) becomes an instance (B).

Most processes in signaling systems propagate information by state changes and non-covalent complex formation rather than substantial material reconfigurations as they occur with metabolic substrates. Classical chemical notation is at too detailed a level of granularity for expressing such processes in a useful way. We shall thus define a simple language that is more attuned to molecular systems biology.

2.2 Agents, complexes, and mixtures

The basic idea is to think of molecular objects as agents decorated with "sites" that may carry a modifiable state and/or bind other agents to form complexes. We shall give a formal syntax of the language, while making extensive use of an equivalent graphical rendering.

The grammatical rules below define well-formed expressions in Kappa. They should be read like instructions for building certain well-formed terms in the language that we call *agents*. We shall define the syntax of Kappa-rules in section 2.5. (The notion of "rule-based" models refers to rules expressing actions, not to the grammatical rules defining the terms of the language.) Throughout this section, let $\mathcal{A}$ be a set of agent names, $\mathcal{S}$ a set of site names (and let $\wp(\mathcal{S})$ denote the powerset of $\mathcal{S}$), $\mathbb{V}$ a set of internal states, and $\mathbb{N}$ a set of labels. Further, let $\psi : \mathcal{A} \mapsto \wp(\mathcal{S})$ be a map that associates an agent name to a set of sites, called the agent's interface.

Definition 2.1 (Agents)**.**

(i)	agent	a	$::=$	$N(\sigma)$	
(ii)	agent name	N	$::=$	$A \in \mathcal{A}$	
(iii)	interface	σ	$::=$	$\varepsilon \mid s, \sigma$	
(iv)	site	s	$::=$	n_ι^λ	
(v)	site name	n	$::=$	$x \in \mathcal{S}$	
(vi)	internal state	ι	$::=$	ϵ	(*any state*)
				$\mid m \in \mathbb{V}$	
(vii)	binding state	λ	$::=$	ϵ	(*free*)
				$\mid\ -$	(*semi-link: "bound to something"*)
				$\mid\ ?$	(*unspecified: free or bound*)
				$\mid\ i \in \mathbb{N}$	(*bond label*)

The grammatical rule (i) defines the overall syntax of an agent as consisting of a name N, taken from the set $\mathcal{A}$ (rule ii), and an interface σ. For example, we may call an agent `ErbB1`. Rule (iii) defines the interface of an agent as a finite set $\sigma = \{s_1, s_2, \ldots, s_n\}$ of *sites*. The vertical bar (|) in (iii) indicates a choice in the recursive application of the grammar when constructing agents. The rule is recursive because σ appears on both sides of the definition: a set of sites consists of a site s and a set of sites. Each time we iterate over (iii), we instantiate a different site s. The s in (iii) refers to the syntactical category "site" defined in (iv). The construction of an interface terminates by choosing the empty interface ε. The sites of an agent control the interactions it participates in. These interactions are defined by Kappa-rules, section 2.5. As indicated in (v), a site s is referred to by an arbitrary name in $\mathcal{S}$, much like an agent. According to (iv), a site carries two types of information, notated as a superscript and subscript to the site name. The subscript ι (iota) of a site refers to its *internal state*, which may be an arbitrary value, as specified in (vi). In most biological interpretations, the value of an internal state indicates a post-translational modification, such as "phosphorylated", "unphosphorylated", "methylated". The superscript λ of a site refers to its *binding state*, defined in (vii) . Agents may be bound to other agents at sites that belong to them. To indicate that site `l` of agent `ErbB1` is bound to site `r` of agent `EGF`, we deploy the same superscript at both sites. For example, the expression $\texttt{ErbB1(l}^{2}\texttt{),EGF(r}^{2}\texttt{)}$ indicates an agent `ErbB1` that is bound to an agent `EGF` at the sites indicated. A superscript uniquely labels a bond between two agents, as laid out in rule (xi). The superscript ϵ means that the site is unbound (free), while a subscript ϵ indicates an unspecified state (like a wild card). We typically do not write the value ϵ. For example, $A(s^{\epsilon}_{\epsilon}) \equiv A(s)$.

According to Definition 2.1 the object consisting of agent `ErbB1` bound to agent `EGF` is not itself an agent, because an agent has only one name by virtue of (i). In fact, `ErbB1` bound to agent `EGF` is a *complex*, which is a different syntactical category. For reasons that will become clear in a moment, we shall define a syntactical category that is more general than a complex, called an *expression*.

Definition 2.2 (Expressions).

(viii)	expression	E	$::=$	$\varepsilon \mid a, E$

The symbol a refers to agents, as previously defined in (i)-(vii). Thus, an expression is simply a set of agents. The syntactical category of expression includes the notion of a complex. For example, the expression

$$\texttt{EGF(r}^1\texttt{),ErbB1(l}^1\texttt{,CR}^3\texttt{,Y1016}_p\texttt{),EGF(r}^2\texttt{),ErbB1(l}^2\texttt{,CR}^3\texttt{,Y1016}_u\texttt{)}$$

denotes a complex in which two `ErbB1` agents, each bound to an `EGF` agent, have dimerized on their sites named `CR`. An agent is an atomic entity, in the sense that it cannot be decomposed into further agents. A complex is a connected graph of agents. In chemistry, an agent would correspond to an atom and a complex to a molecule, see Figure 1. Within our domain of application, an agent is oftentimes a protein and a complex is a set of proteins connected by non-covalent bindings.

An expression is more general than a complex, as Definition 2.2 does not require any bindings between agents in an expression. Figure 2 illustrates an expression (and a graphical presentation) consisting of an agent `EGF(r)`, an agent $\texttt{ErbB1(l,CR,Y1016}_p\texttt{)}$, and a complex $\texttt{EGF(r}^1\texttt{),ErbB1(l}^1\texttt{,CR}^3\texttt{,Y1016}_p\texttt{), EGF(r}^2\texttt{),ErbB1(l}^2\texttt{,CR}^3\texttt{,Y1016}_u\texttt{)}$. In essence, an expression (viii) is a graph over agents whose connected components are complexes[1].

Definition 2.3 (Well-formedness).

(ix)	unique interface	each site name in the scope of an agent named A must be in $\psi(A)$
(x)	agent scope	a site name can occur only once in a given interface
(xi)	binary binding	a binding state $i \in \mathbb{N}$ occurs exactly twice, if it occurs at all

Definition 2.3 spells out conditions that make an expression "well-formed". An agent should be thought of as being associated with a unique interface (by virtue of the mapping ψ). As we shall see later, agents in an expression are oftentimes mentioned with only a subset of their sites. Thus, rule (ix) ensures that these sites are elements of the agent's interface. In addition, if a site is

[1]A word about typesetting. To facilitate the input and output of complexes in a session with a computational platform, we shall use an exclamation mark (!) followed by a number instead of the corresponding superscript and a wiggle (~) followed by a value (usually a literal) instead of the subscript. The keyboard notation will be confined to figures, as we prefer the more compact mathematical notation for typeset text. Generally, we shall use typewriter font when referring to an agent expression, while normal font when referring to the empirical molecular entity known under the same name. Thus, EGF is the empirical object known as the epidermal growth factor, while `EGF()` is a Kappa-agent.

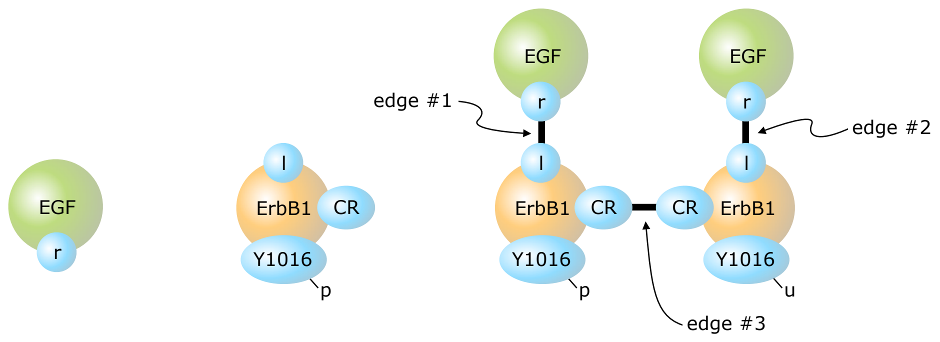
bound, its binding label occurs exactly twice (once at both sites connected by the edge so labeled).

We want an expression to represent the contents of a well-stirred mixture or chemical solution. To formalize this intent, we define structural equivalences between expressions. This is a standard procedure to offset the distortion in meaning arising from the constraints of linear text.

Definition 2.4 (Structural equivalence)**.**

(xii)	interface	$E, A(\sigma, s', s, \sigma), E'$	$\equiv$	$E, A(\sigma, s, s', \sigma), E'$	*(site permutation)*
(xiii)	mixture	E, a, a', E'	$\equiv$	E, a', a, E'	*(agent permutation)*
(xiv)	edge labels	$i, j \in \mathbb{N} \wedge i$ not in E	$\Rightarrow$	$E[i/j] \equiv E$	*(relabeling)*

The first two equivalences, (xii) and (xiii), erase any notion of space in the Kappa language. This is important to keep in mind, since textual (and graphical) renditions have a tendency to fool us. In particular, rule (xii) states that an interface is a set, not an ordered sequence of sites. Hence, the placement of sites in a graphical representation, such as Figure 2, has no significance. Rule (xiii) states that an expression has no spatial meaning. Every agent or complex is "equidistant" from any other, since all shuffles of an expression are equivalent. An expression therefore represents a well-mixed solution of molecular objects. Rule (xiv) states that we can relabel edges (bonds) as we please, provided the labels remain unique. Thus, if j is an edge label in an expression E and i is not, then we can substitute i for j in E (denoted by $E[i/j]$) without changing the meaning of E.



$\underline{\texttt{EGF(r)}}$, $\underline{\texttt{ErbB1(l,CR,Y1016}_p\texttt{)}}$,$\underline{\texttt{EGF(r}^1\texttt{),ErbB1(l}^1\texttt{,CR}^3\texttt{,Y1016}_p\texttt{),EGF(r}^2\texttt{),ErbB1(l}^2\texttt{,CR}^3\texttt{,Y1016}_u\texttt{)}}$

Figure 2: A Kappa expression. The textual representation of a small reaction mixture containing 6 agents that are divided into three complexes (underlined) is shown at the bottom. The two complexes on the left are simple agents, while the complex on the right is made of 4 agents hanging together as shown. An equivalent graphical rendition is depicted above the textual expression, exhibiting the complexes in the same order (left to right) as in the expression below.

In summary, an expression represents a mixture as a graph, whose nodes are agents and whose components (the connected subgraphs) are complexes. If there are 1859 EGF molecules in a mixture, the expression representing it will contain 1859 comma-separated `EGF()` agents. Some of these might occur in a complex with `ErbB1()` agents, others might be unbound.

2.2.1 Comma

Agents in an expression are separated by commas. The comma denotes coexistence as defined by the structural equivalence rule (xiii), Definition 2.4. At the same time, the comma emphasizes the agent-based style of our reasoning: the comma-separated units of an expression are agents, not complexes. It is thus important to keep in mind that comma-“separated” agents need not be disconnected in the graphical sense, as can be seen from the expression depicted in Figure 2. Likewise, to state that two agents are in the same complex requires specifying a path between them.

2.2.2 Sites

Sites carry state (which is modified by rules that will be defined in section 2.5). For example, a site might represent a particular phosphorylatable tyrosine residue, or an SH2 binding domain. Yet, sites are best thought of as logical resources or capabilities. For example, a protein might contain a transport signal sequence that is cut off upon crossing a membrane. Its presence or absence can be represented by the state of a site on the protein’s Kappa agent. The state of another site might represent the agent’s localization, such as endoplasmatic reticulum, cytoplasm, mitochondrion, or membrane. Naturally, the capabilities represented by sites are somehow physically implemented, but in Kappa we do not represent that implementation. Like any material resource, logical resources are tied up when in use: a site can be bound only to one other site. This makes complexes rather special kinds of graphs to which we turn next.

2.2.3 Site graphs

A complex is a connected graph with agent as nodes. However, unlike in a conventional graph, the edges (bonds) of a complex don’t just end up in agents, but in sites that belong to agents. We call such a graph a *site graph.* Because a site can sustain at most one edge, any path between two agents in a complex has a unique identification given by the sequence of (site,agent)-pairs traversed on that path. This is not the case in a conventional graph that contains several instances of the same node type (agent name). We shall return to the consequences of this property when defining matchings of patterns, after having discussed patterns.

2.3 Patterns

The agents in an expression representing the contents of a reaction mixture always appear with a fully specified interface. It is useful, however, to consider agents with a partially specified interface. Recall that chemical rules, such as the one in Figure 1A, refer to partially specified molecules for the purpose of compactly isolating a transformation that occurs across many different reaction instances involving completely sepcified molecules. In the context of Kappa, the term “molecule” refers to a completely specified complex (whose agents appear with full interface). We use the term “pattern” when referring to a partially specified molecule or agent.

A *pattern* is an expression of partially specified agents. Figure 3 depicts the basic types of patterns exemplified by an agent `A(x,y)` with two sites:

1. *Unspecified binding partner.* The expression $\mathtt{A(x_1,y_p^-)}$, Figure 3A, specifies an agent in

state 1 at site x and in state p at site y. In addition, site y is bound, but we don't specify to whom. We call this a *semi-link* and indicate it by a hyphen (−) instead of an edge label.

2. *Unspecified binding state.* The agent expression $\mathtt{A(x_1,y_p^?)}$, Figure 3B, is similar to the previous one, except that we do not care whether site y is bound. We indicate this by a question mark (?) in the bond superscript. Note that by *not* mentioning any binding state for site x we assert that this site is free (unbound).

3. *Unspecified internal state.* In $\mathtt{A(x,y_p)}$, Figure 3C, we do not care about the internal state of site x, because we omit its subscript. However, we do care that the site be free (as in all previous cases). Site y is in state p and free.

4. *Omitted site.* In $\mathtt{A(y_p)}$, Figure 3D, we omit site x entirely, asserting that we don't care about its internal state nor its binding state. Site y is in state p and free.

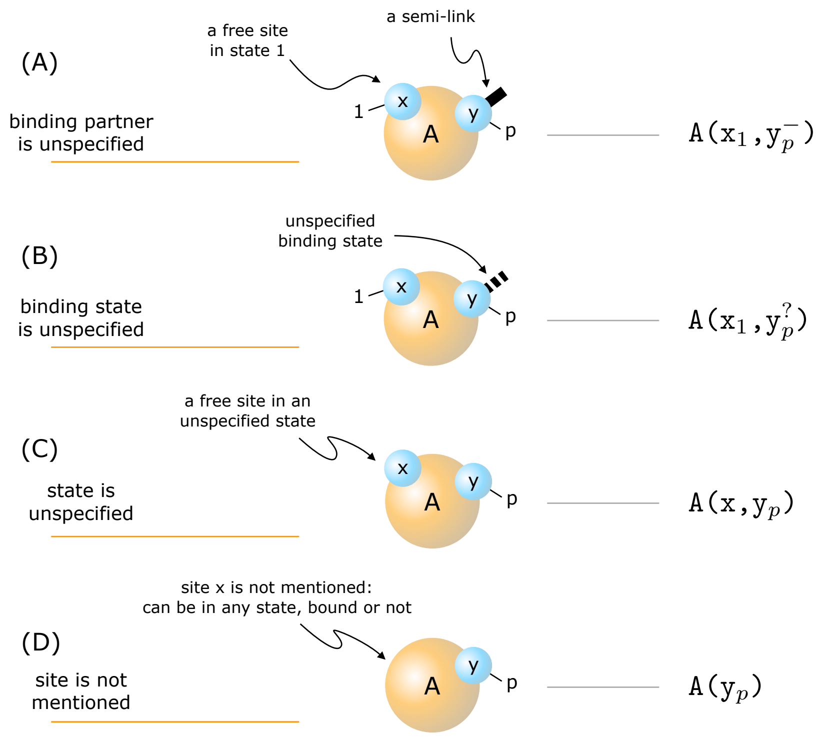

(A)
a free site in state 1
a semi-link
binding partner is unspecified
1
x
y
A
p
A(x1,y−p)
(B)
unspecified binding state
binding state is unspecified
A(x1,y?p)
(C)
a free site in an unspecified state
state is unspecified
A(x,yp)
(D)
site x is not mentioned: can be in any state, bound or not
site is not mentioned
A(yp)

Figure 3: Basic Kappa patterns. A pattern is a partially specified agent (or set of agents). Various ways of omitting information are shown. A: the binding partner of a site is left unspecified. B: the binding state of a site is left unspecified. C: the internal state of a site is unspecified. D: both internal and binding states of a site are left unspecified by not mentioning the site at all. See text for details.

2.4 Pattern matching

Matching is a process that establishes whether a more detailed expression E' conforms to a less detailed expression E. To gain some intuition, consider agents first. A specification A' of an agent *conforms to* (matches) a specification A, if

(i) A' and A coincide in agent name and all site names that A mentions, and

(ii) the state values ($\iota \in \mathbb{V} \mid \epsilon$) and binding values ($\lambda \in \mathbb{N} \mid - \mid ?$) of each site mentioned in A, are either equal or less specific than those mentioned in A'. With regard to binding state, '?' is less specific than ϵ or '$-$', and '$-$' is less specific than a label $i \in \mathbb{N}$. With regard to internal state, ϵ is less specific than a value $\iota \in \mathbb{V}$.

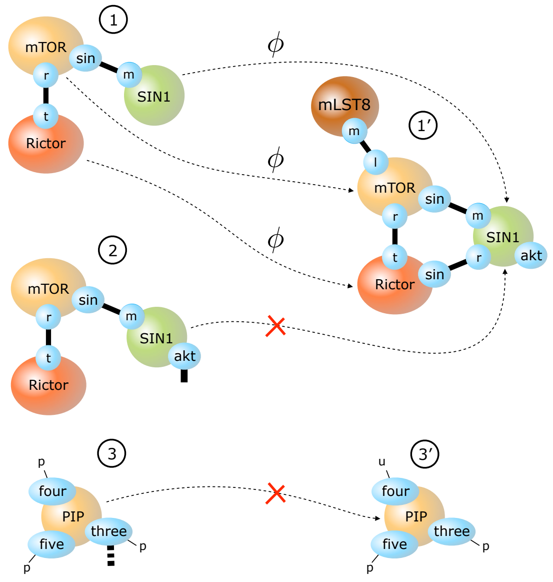


Figure 4: Graph embedding. The embedding (fitting) of less detailed graphs on the left into more detailed graphs on the right. Graph 1 on the left embeds into complex 1' (a signaling assembly known as mTORC2, consisting of mTOR, Rictor, SIN1, and mLST8). Graph 1 agrees in all the names and states it mentions with graph 1'. Since graph 1 omits the site `l` from agent `mTOR`, `l`'s state in graph 1' is irrelevant. The label ϕ indicates the injective mapping from agents in 1 into agents in 1'; we say graph 1 embeds into graph 1', ϕ, $G_1 \lhd_\phi G'_1$. The graphs G'_1 and G_1 can be rendered in terms of their respective string expressions E'_1 and E_1. Upon arranging the strings according to ϕ, the criteria in Definition A.1 establish that E'_1 conforms to E_1, $E'_1 \vDash E_1$. Graph 2, in contrast, does not fit complex 1', as the former demands that `SIN1` be bound to something at its site `akt`, but 1' specifies site `akt` to be free. In graph 3, agent `PIP` on the left does not match `PIP` in complex 3' on the right, as the latter has site `four` in an unphosphorylated state, while the former requests a phosphorylated state. There is no disagreement on site `three`, as graph 3 does not care about its binding state (the dotted line stands for a '?' in the textual representation, indicating "bound or unbound").

The concept of a match can be extended to expressions (mixtures) E' and E, by saying that E' conforms to E, written as $E' \vDash E$, if every agent in E' conforms to a distinct agent in E. In particular, anything conforms to an empty expression. Usually, E' is a reaction mixture, and E is the pattern on the left hand side of a rule (see section 2.5). Finding a match of E' to E may necessitate the inspection of several structural equivalences of E', generated by reordering agents, interfaces, and relabeling bonds using Definition 2.4. For the sake of precision, matching is formally defined in Appendix A.1. In essence, a match $E' \vDash E$ asserts that E is a subsequence of E' at the level of agents. (Our definition above also requires that the sites mentioned in an agent of E' have an internal state and binding state at least as specific as those of the sites in the corresponding agent of E).

As seen in Figure 2, an expression E' can be represented by a graph $G' = \llcorner E' \lrcorner$, in which nodes are agents identified by their sequential position in the expression. When expressions E' and E (with E' being more detailed than E) are translated into their corresponding graphs $G' = \llcorner E' \lrcorner$ and $G = \llcorner E \lrcorner$, the concept of matching $E' \vDash E$ becomes one of *embedding* G into G', denoted by $G \lhd_\phi G'$, where ϕ is (roughly - for a precise account, see Appendix A.1) a mapping from nodes of the smaller graph G into nodes of a larger graph G', while preserving agent identities.

Caution: we think of the less detailed graph G as being embedded ("fitted") into the more detailed graph G', while a more detailed expression E' matches a less detailed pattern E. Figure 4 provides a few graphical examples to fix the concepts. The formal definitions in Appendix A.1 appear a bit involved because agents have a type (e.g. an agent of type *EGF*) as well as an identity (e.g. the agent #1 of type *EGF*, as distinct from the agent #2 of type *EGF*, in an activated EGF-receptor dimer). Intuitively, an embedding of one graph G into another G' (the first not larger than the second) is a process whereby we move G over G', trying to overlay G on G' such that agent types match (as well as the states of the sites mentioned by the agents of G). Several overlays may be possible for a given pair of graphs G and G'.

2.4.1 Rigidity

Suppose we need to decide whether a graph P (the pattern) embeds into a graph C representing a specific complex. As pointed out in section 2.2.3, the graphs we are dealing with are site graphs. Within such a graph, any agent A is addressable from any agent R by a sequence of (site,agent)-pairs describing a path from R to A. As a consequence, an embedding of an agent of P into an agent of C can be extended in at most one way to an embedding of P into C . Identifying whether a graph P embeds into C has therefore a computational cost bounded by $|P|$, the number of agents in P. Furthermore, there can be at most $|E|$ embeddings of P in the reaction mixture E.

2.5 Rules

The main use of patterns is in the definition of Kappa rules. In analogy to chemical reaction rules, a rule is a pair of expressions that are typically patterns:

$$E_{\text{left}} \longrightarrow E_{\text{right}}.$$

The pattern E_{left} defines conditions on internal states and binding states of agents that have to be satisfied for the rule to apply. Rules are applied to a mixture, that is, an expression S representing the contents of a reaction system at a given time. The basic idea is illustrated in

Figure 5 for the rule

$$\begin{array}{c}\texttt{Gab1(PH}^{-}\texttt{,Y447}^{1}\texttt{),PI3K(SH2}^{1}\texttt{,s),PIP(three}_{u}\texttt{,four}_{p}\texttt{,five}_{p}\texttt{)} \\ \downarrow \\ \texttt{Gab1(PH}^{-}\texttt{,Y447}^{1}\texttt{),PI3K(SH2}^{1}\texttt{,s}^{2}\texttt{),PIP(three}_{u}^{2}\texttt{,four}_{p}\texttt{,five}_{p}\texttt{)}\end{array} \tag{1}$$

(which we write vertically for ease of placement on the page). Below the rule in Figure 5, we have sketched a hypothetical mixture. We want to identify *a* configuration of (fully specified) agents in the reaction mixture S that satisfies the pattern of reactants on the left hand side (lhs), E_{left}, of the rule. When such a configuration has been located, it is replaced by the configuration specified on the right hand side (rhs), E_{right}, of the rule. Replacement consists in updating the internal states and the binding states that are changed by the rule. The operational meaning of a match is explained in section 2.4 (and formalized in Appendix A.1; the notion of replacement is formalized in Appendix A.2.

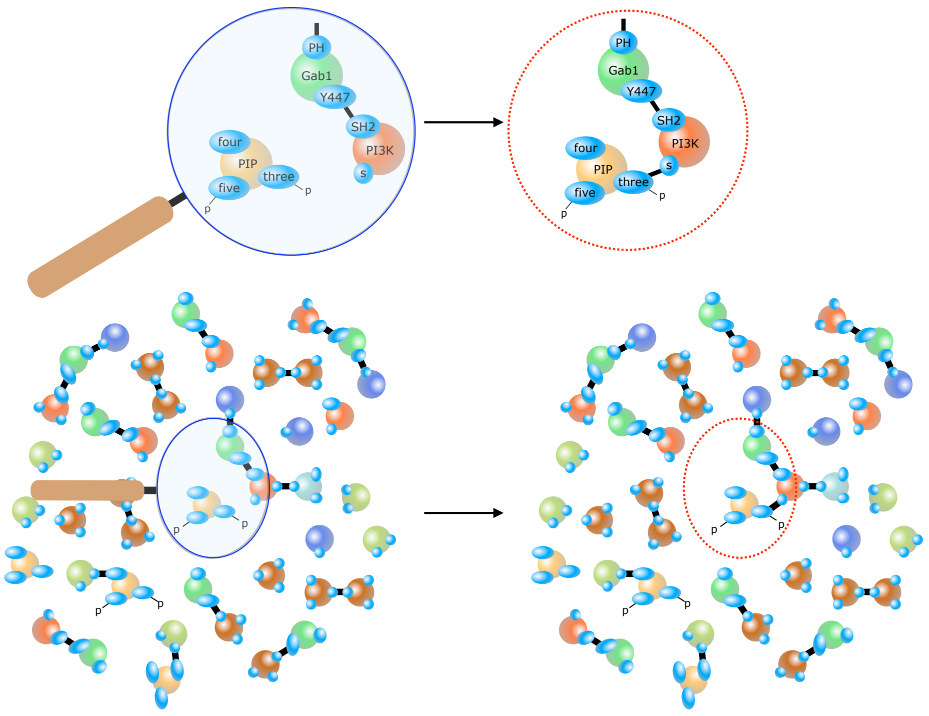


Figure 5: Rule application in Kappa. First, a match between the pattern on the left hand side of a rule (blue lens) and the mixture (bottom) is identified. The action specified by the rule is then applied to the matching configuration, resulting in a new configuration according to the rule's right hand side (red circle). Many matchings may be possible for any given rule and many different rules may be applicable at any given moment. Rules and matchings are chosen for execution in a way that generates probabilistically correct sequences of events, following a generalization of Gillespie's algorithm for stochastic chemical kinetics.

We can think of a rule as an *action* that is applied to a configuration in the mixture. The action is the difference between the right hand side (rhs) and lhs of a rule. The differences may be many, such as changing several internal states and binding states at once, but they all boil down to a handful of *elementary actions* (Figure 6) that cannot be further decomposed within the present definition of Kappa: binding, unbinding, and the change of an internal state. Kappa also allows for the creation and the removal of an agent.

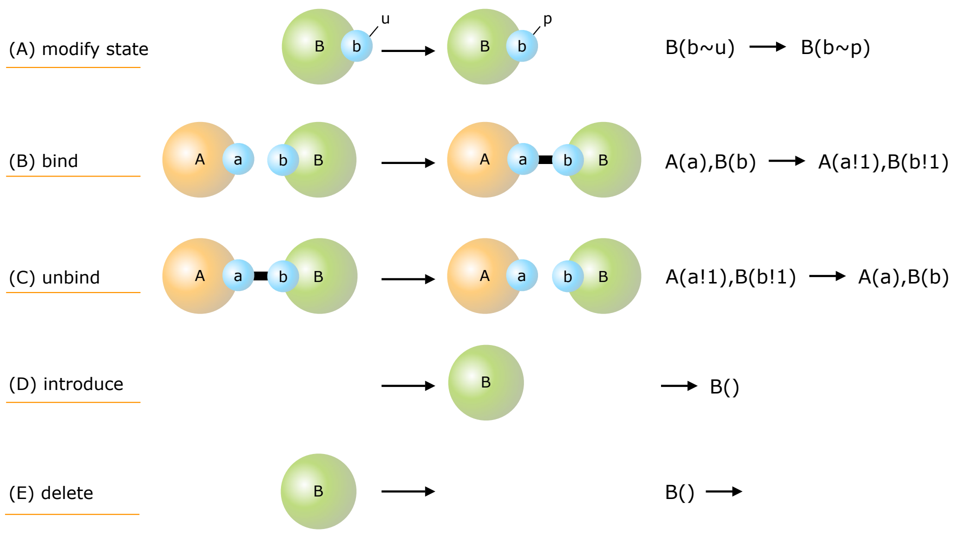


Figure 6: Elementary actions in Kappa. The figure depicts the five atomic differences between the right and left hand sides of a rule. (A): the modification of a site's internal state; (B) and (C): the modification of a site's binding state affects two agents; (D) and (E): the introduction or deletion of an agent. The rules shown are the simplest cases exhibiting atomicity. For example, if the modification in (A) were to specify another agent that remains unchanged, it would still be an atomic rule (even if we might not consider the mere presence of a catalyst as a sufficiently mechanistic picture).

Rules must obey certain constraints to be sound. Obviously, expressions E_{left} and E_{right} must be well-formed, that is, in compliance with Definitions 2.1, 2.2, and 2.3. The interpretation of a rule, however, requires a mapping of agent identities across the arrow. We must know which agents on the right of a (textual) rule correspond to which agents on its left. There are several ways of defining such a mapping. We opted for a simple convention: both sides of a rule, E_{left} and E_{right}, are compared with one another proceeding from the left of each expression. The comparison only checks agent names and interfaces, but is blind to the states of the sites mentioned. It ends at the first difference. This procedure identifies a longest left-anchored substring – a prefix – common to both expressions. (It may be empty.) The prefix now establishes a sequential correspondence between agents on the left and right hand sides of a rule. Anything after the common prefix is interpreted in terms of deletions and introductions of agents, depending on whether an agent is missing on the right or left hand side, respectively. Subtleties of the mapping rise to the user's attention only when using textual input. A graphical interface hides them completely. See Appendix B for more details.

In Kappa, virtually any rule is acceptable by the parser and executable by the interpreter. There is indeed no reason to be more restrictive at the level of the pure language (which is a nifty formal object in its own right). In practice, however, there is plenty of reason to be disciplined. We explain the subtleties of how Kappa rules are interpreted in Appendix B. For most practical applications, researchers using Kappa will probably do so through a graphical interface, like the one we provide. A graphical front end for building rules is not only of cognitive help, but also a very effective way of enforcing discipline without becoming fully aware of the loss of total freedom, see Appendix C.

For a Kappa rule to be *sensible* in most modeling practices, it should comply with a few guidelines.

Guideline 1 *A site whose internal state or binding state is modified by a rule should be mentioned on the left hand side in a defined internal state or binding state, respectively.*

This is the most important guideline to follow. Thus, $A() \rightarrow A(x_m)$ is not a sensible rule. It is a legal rule, but it has a meaning that is most likely not intended, see Appendix B.2. The rule $A(x^?), B(y) \rightarrow A(x^1), B(y^1)$ is not advisable either, as the site x of agent A is not in a defined binding state. Similarly, the rule $A(x) \rightarrow A(x_m)$ has site x in a definite binding state (free), but not in a defined internal state (it mentions none, which means ϵ). We discourage using such a rule because it generates a kinetics that is probably not intended, as discussed in Appendix B.2.3.

Guideline 2 *A rule should be atomic whenever possible.*

Naturally, compliance with this guideline depends on one's knowledge. Atomic rules make the causal analysis of a system much more insightful. (We detail causal analysis later.) For example, we may wish to ask for the events that are necessary to activate a specific target. A mechanism of action or pathway is obviously more informative if it is not *a priori* clear how events in a distributed system conspire in generating a particular observable. The rule $a\_lot\_of\_damage() \rightarrow apoptosis()$ is perfectly legal, but will not contribute much to the causal understanding of how or why a system triggers apoptosis.

Guideline 3 *A rule that introduces an agent should specify the full interface of the agent and assign definite states to all its sites.*

The Kappa interpreter does not rely on a separate declaration or definition of an agent's interface, other than what it can infer from the rules. A rule that creates an agent, will create it with exactly the interface defined in that rule.

Compliance with certain guidelines can be enforced by suitable user interfaces. Typically the user will interact with the Kappa interpreter in the context of a modeling environment that provides means for managing "agent cards" that define an agent's interface. The creation of an agent may then refer to its card and generate the appropriate interface automatically, thereby enforcing Guideline 3. See Appendix B.2.1 for more detail. Likewise, Guideline 1 can be enforced painlessly by a graphical rule designer, as exemplified in Appendix C.

The semantics of agent removal is defined as prying the agent from the complex it is a member of, if any. The binding states of all affected agents in the complex are updated. (An alternative semantics would be to zap the whole complex to which the removed agent belongs. We felt our current definition is more conservative.) The affected complex does not have to be mentioned. It is identified at execution time by inspecting the bindings of the agent slated for deletion. Such a rule is called *contextual.* For example, a match to the rule shown in Figure 6E might pick a B in the reaction mixture that is bound to several agents whose binding states must be updated upon execution of the rule.

For convenience we also allow a semi-link on the left hand side of a rule. In particular, `A(a⁻)` $\rightarrow$ `A(a)` is a valid rule. Here too, the implementation has to identify at run time to whom the match to A is bound. The rule is compliant with Guideline 1, since we do care that site a is bound, although we don't care to specify to whom.

2.5.1 Executable knowledge

A rule is a formalized (hence unambiguous and computer-actionable) assertion about a molecular interaction at the level of granularity set by the language. The empirical foundation of such an assertion can vary along a wide spectrum - from outright absent (e.g. a counterfactual) to gold standard. In the context of a knowledge environment, rules may (and should) be linked to information detailing provenance, experimental conditions, biological scope (such as cell type or localization of the event), and structural information about agents and sites. Rules have great appeal as concise specifications of "facts", whether conjectures or pebbles of knowledge. Most importantly, however, a rule is an executable specification. Executable means that the actions described by rules are precise operations used to drive the dynamics of an expression representing a mixture of agents. The coalescence of description (specification) and execution (implementation) is crucial for making the static analysis of models (described elsewhere) powerful.

2.5.2 Don't care, don't write

Patterns give rise to a significant economy of expression by omitting context that is irrelevant to the action specified by a rule. For example, a particular mechanism for the binding of a kinase to a substrate with 10 phosphorylatable tyrosine residues, $K(s), S(k) \rightarrow K(s^1), S(k^1)$, might be independent of the phosphorylation state of each site. In such a case, 1024 fully contextualized reactions are simply expressed by a single rule that omits mentioning the 10 sites and thus matches any of the 1024 possible configurations. This is the "don't care, don't write" principle. Thus, the two rules depicted in Figure 7 specify the same dimerization, but the first is more specific than the second, as it insists on site r being unbound.

2.5.3 Modifying rules

There is a compelling reason for using rules as model elements: the option of tuning (and updating) the amount of context in the specification of a rule. This achieves an economy of expression that is needed for handling combinatorial systems. It also permits an easy update of rules to reflect changes in the observations or hypotheses they represent. This is illustrated with rule refinement, discussed in section 2.5.5 and Figure 8 below. Moreover, rules can be easily added or deleted, as they are self-contained elements of interaction. In a differential equation setting, a rule is spread across a network of reaction instances, which makes updating a simple interaction that is embedded in a combinatorial context a nightmare.

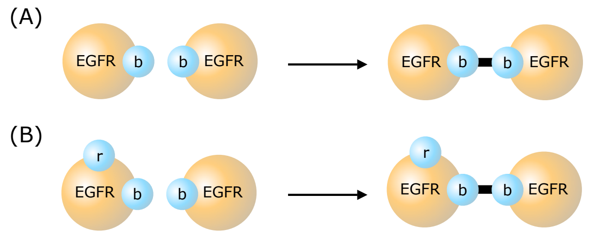


Figure 7: Don't care, don't write. The states of sites mentioned on the left hand side of a rule are meant to be *tested* as preconditions for application. The testing is a component of the matching process defined in section 2.4 and Appendix A.1. Omitted sites are not tested and their state has therefore no bearing on the applicability of the rule. Both rules in the Figure specify the same action, the binding of two `EGFR` agents to one another at their respective sites b. The second rule, however, requires (for one agent) site r to be in a definite binding state - free - while placing no constraints on its internal state. The macthing process filters those configurations in a mixture to which the action specified by a rule is applied. The second rule is more specific than the first. We say the second rule "refines" the first.

2.5.4 Rate constants

We focused on the action part of a rule, but a rule is not complete without an associated rate constant. In the Kappa framework, the rate constant retains its biophysical meaning and role in deterministic mass action kinetics. In the stochastic simulation scheme described later, the deterministic rate constant is related to the parameter of an exponential reaction probability density by a factor $1/V^{(n-1)}$, where n is the reaction order. (In Kappa, reactions are, at least for now, treated as elementary with regard to kinetics, even when they are not elementary with regard to Kappa-actions. Thus the reaction order is really the molecularity of the reaction.) Keep in mind that a rule typically covers many reaction instances in which the reactants differ with regard to states *not* mentioned in the rule. All these reactions inherit the same rate constant associated with the rule. Indeed a major reason for refining a rule by adding more context is to account for situations that demand different rate constants. We turn to the concept of refinement next.

2.5.5 Refinements

Refinement is an important concept in any rule-based approach. The left hand pattern of a rule defines a set of matching instances. This set can be narrowed by adding more context to the pattern. We call R_2 a refinement of R_1, when its matching instances are a subset of those of R_1. For example, the rules depicted in Figure 8 are statements about the dissociation of mTOR from Rictor, two components of the mTORC2 signaling complex. The topmost rule 1 places no conditions on the dissociation of mTOR from Rictor, other than their mutual association to begin with. However, experimental evidence suggests that the stability of the mTOR/Rictor association depends on a third protein, SIN1, bound to both and stabilizing their association. To express this finding, we might first add a $SIN1$ binding site (labelled sin) to $mTOR$. We then could say that if $mTOR$ is free at its site sin, the complex should dissociate more readily. This is shown as case 2 in Figure 8, which is a refinement of case 1, as it specifies an additional condition for dissociation. (For the sake of less clutter, we only show the left hand sides of rules. The

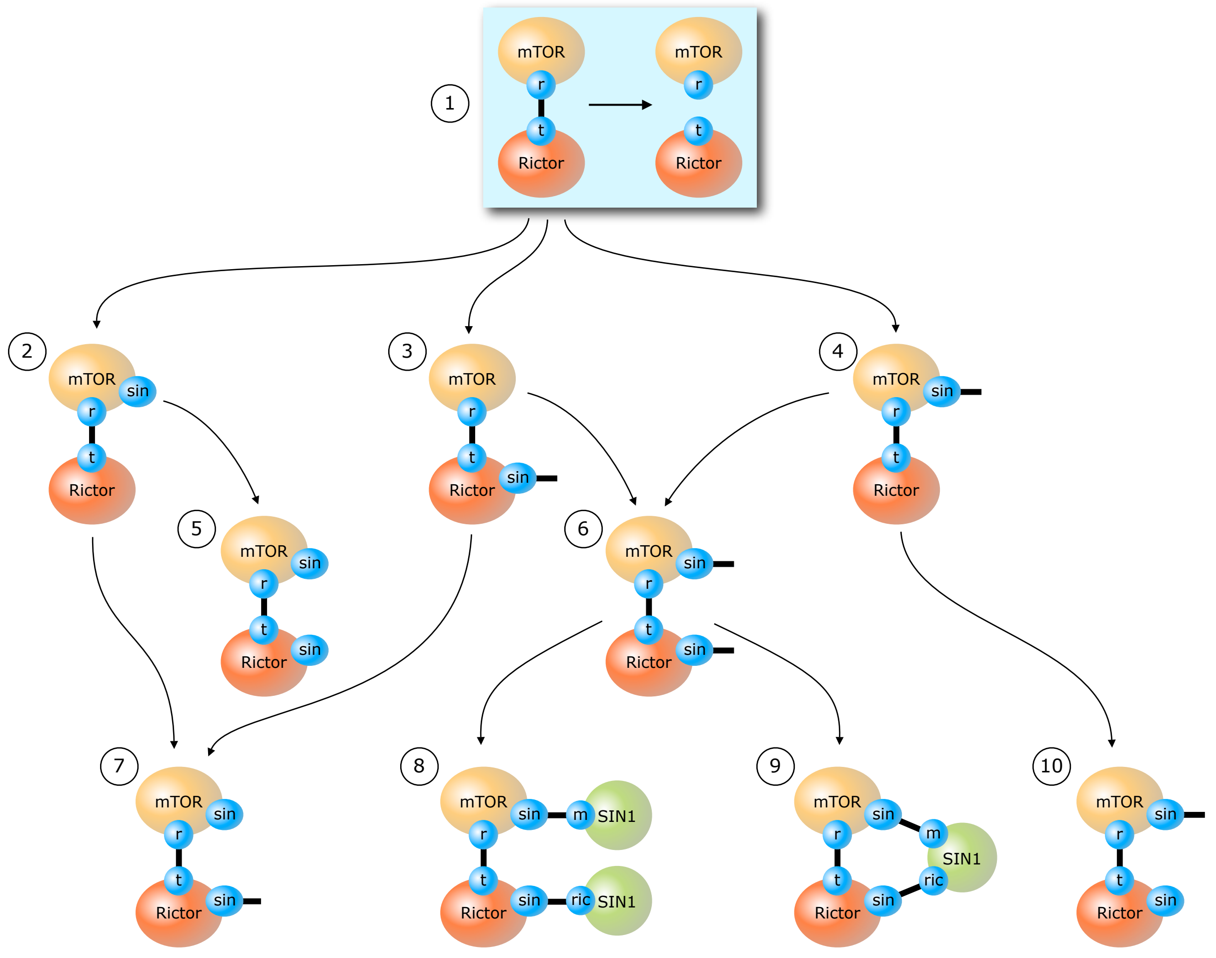


Figure 8: Rule refinement. A rule is refined by adding more context to it. Refinement does not change the action of a rule (that is, what a rule modifies); it is only more specific about the conditions that have to be satisfied for the rule to apply. The Figure shows some refinements of the dissociation action on an $mTOR/Rictor$ complex. Except for case 1, only the left hand sides of rules are shown; the corresponding right hand sides are identical to the left hand side, but with the $mTOR/Rictor$ binding removed. Curved arrows indicate the refinement relation, which is a subset relation. The instances matched by the rule at the tip of an arrow form a subset of the instances matched by the rule at its shaft. For the sake of less clutter, only the reduced relation is shown here. (The reduced relation is the unique minimal relation whose transitive closure yields the original. In symbols: R is reduced, if $(x, y) \in R \Rightarrow (x, y) \notin (R \setminus (x, y))^*$, where the $*$ means transitive closure.) The topmost rule is the one with the least possible constraints and thus represents the action *per se*. It can be thought of as "typifying" the class of rules that are its refinements. See text for details.

corresponding right hand sides remove the link between $mTOR$ at site r and $Rictor$ at site t.) We would also like to say that if either only $Rictor$ or $mTOR$ are bound to $SIN1$, dissociation should be faster (since then $SIN1$ is not bound to both like a scaffold). This is expressed in rules 7 and 10. Still, we aren't saying all there is to say, since rule 7 doesn't cover all instances of rule 2, and neither does rule 10 with regard to rule 4. Both rule 2 and rule 4 do not mention the *sin* binding state of *Rictor*. Moreover, we wish to distinguish the case in which both *Rictor* and *mTOR* are bound to a distinct $SIN1$ from the case in which both are bound to the same $SIN1$ (which warrants a lower dissociation rate constant than all other cases). This is achieved by further refinements into rules 8 and 9. In the end, rules 5, 7, 8, 9, and 10 (the endpoints of a directed acyclic refinement graph) constitute a partition of the matching instances of rule 1. By replacing rule 1 and assigning higher rate constants of dissociation to rules 5, 7, 8, and 10, as compared to rule 9, we can express the observation that the simultaneous binding of SIN1 to both Rictor and mTOR stabilizes the Rictor/mTOR association.

Figure 8 summarizes the refinement relations between different rule versions. An arrow, $R_1 \rightarrow R_2$, means that R_2 is a refinement of R_1. The refinement relation defines a directed acyclic graph, or DAG. One property to note is that following arrows backwards from more to less contextualized (refined) rules gets us to a “root rule” that expresses exactly the action common to rules in the same DAG. In this case, it is the dissociation of mTOR from Rictor. This is very useful, since it provides a natural means for organizing a large collection of rules by grouping rules according to their action(s). All rules in a group are then refinements that differ solely in the conditions they demand for the action(s) to occur. If we were to refine all the way down to fully contextualized rules, we would end up with reaction instances, which is the level of detail required by differential equations. An expansion into fully contextualized instances is either unnecessary, because it provides too much irrelevant information, or unfeasible, because the refinements are too numerous or even infinite. The latter is the case when polymerization occurs, which is far more frequent than one might think at first. We shall turn to polymerization in the next section.

The process of refinement can be greatly assisted by a modeling environment that provides services for checking whether a particular rule refines another (or how the two rules overlap with regard to their matching instances) as well as services that propose refinement choices to the user. There is nothing wrong with refinements that overlap in their matching instances, as long as the modeler is aware that instances matched by more than one rule have correspondingly expedited kinetics, as the rate constants sum up in the simulation.

2.6 Polymerization, global constraints, and context free languages

In section 2.2.1, when discussing the meaning of the comma (’,’) in Kappa expressions, we emphasized that the comma does say nothing about whether two agents are or are not in the same complex. This is particularly important to appreciate in the context of rules. Consider, again, the mTORC2 case, which we used in the previous section to introduce the concept of rule refinement. Here we shall further our understanding of the Kappa language by focusing on the assembly of mTORC2. At its core, mTORC2 seems to be a cyclic complex, in which mTOR, Rictor, and SIN1 bind to each other. To express these binding capabilities, we might formulate the three rules 1-3 in Figure 9A. However, taken together, these rules result in an unwanted situation. In Figure 9B and C we show two situations in which reactants are matched by the pattern of rule 1. In case B, the molecular species $\texttt{mTOR(lst}^1\texttt{,sin}^2\texttt{,r),mLST8(m}^1\texttt{),SIN1(m}^2\texttt{,ric}^3\texttt{),Rictor(sin}^3\texttt{,t)}$ (a complex with fully specified interface - the kind of things that float in our virtual mixture) is matched by the left hand side of rule 1, giving rise to an intra-molecular binding event that completes the assembly of mTORC2. In case C, rule 1 matches two reactants and gives rise to an inter-molecular binding event. However, one of the reactants already contains `Rictor`, and the product now contains two `Rictor` agents. This is the beginning of a polymerization process. Indeed, the execution of rules 1-3 with a starting mixture of many `mTOR`, `Rictor`, and `SIN1` agents would lead to very little cyclical complex and to a variety of oligomers of the linear mTOR/SIN1/Rictor "monomer", since, *ceteris paribus*, bimolecular reactions tend to be favored over monomolecular reactions by mass action kinetics. In other words, rules 1-3 by themselves do not guarantee that when an intra-molecular binding is possible, it gets favored. How does nature do it? Possibly through geometric constraints that might bring Rictor into proper juxtaposition with mTOR, or by an effective increase in the local likelihood of an mTOR/Rictor encounter when the two are connected via SIN1 (an entropic constraint). Kappa, as defined, cannot represent geometric constraints, because it does not know about space.

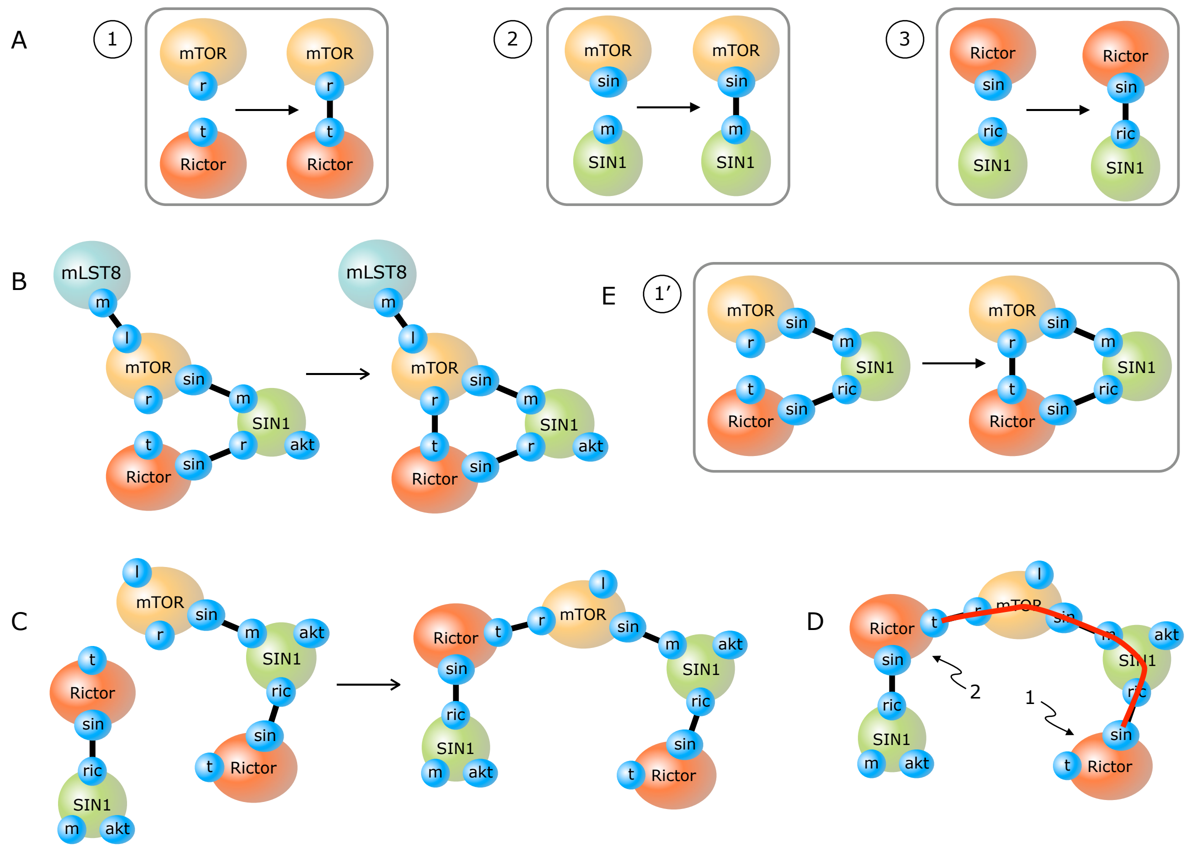


Figure 9: Rules and polymerization. If a system of rules creates both intra- and inter-molecular configurations to which a rule R is applicable, R will generate polymers.

There are several possibilities for avoiding polymer formation. We could refine rule 1 in Figure 9A to become rule 1' in Figure 9E. The new rule set, however, forces a course of events in that the mTOR/Rictor binding is the last one to occur in the formation of the cyclical mTORC2 complex. We do not have empirical evidence that this is the case. It now becomes a matter of modeling style. Do we want to be faithful to empirical observations as much as possible or do we want to use shortcuts to generate desired complexes because we “know” (do we?) that nothing much rides on how mTORC2 is assembled? Kappa does not prevent a modeler from encoding outcomes he or she firmly believes in. Yet Kappa also supports a modeling style in which a course of events can be discovered from as faithful a representation of empirical observations as possible.

We cannot simply replace rules 1-3 with rules analogous to rule 1’ (to which we refer as rules 2’ and 3’), since we must allow for some unconditional binding events or no complex would ever get started. We could, however, *add* rules 1’-3’ to rules 1-3, and assign to the former high rate constants. In this way we use rule refinement to encode what otherwise would result from geometric constraints. The new rule set still generate polymers (because of rules 1-3). In addition, the relative lower rate constants of rules 1-3 compared to rules 1’-3’ slow down complex initiation, which may not correspond to observations.

In a different approach to the problem, we inspect the reactants matched by the left hand side of rule 1 (say) and decide whether the reaction leads to unwanted products, in which case the (attempted) reaction is rejected. (The consequences of rejecting reaction for correct stochastic simulation will be detailed in a later section.) To disallow reactions (not rules!) requires knowing who the reactants are and that can only be known at “execution time” by the simulator (described later). The decision whether a reaction instance leads to unwanted products hinges on a characterization of the set of molecular species to be avoided. In the mTORC2 case, a decision criterion is illustrated in Figure 9D for rejecting reaction C. If the product contains a path between two agents of the same type, say $\texttt{Rictor}_1$ and $\texttt{Rictor}_2$, where the site at $\texttt{Rictor}_1$ where the path starts differs from the site at which the path ends at $\texttt{Rictor}_2$, then the product can polymerize further and the reaction should therefore be rejected. This is illustrated by the red path in Figure 9D: the segment from (and excluding) $\texttt{Rictor}_2$ to (and including) $\texttt{Rictor}_1$ can be repeated on the side of $\texttt{Rictor}_1$ - and the rules to do so are certainly available in the rule set, or the reactants could not have been produced to begin with. (We could equally well have chosen the two `SIN1` instances.) Here is another way of saying the same thing. Imagine a rule that creates a binding between two patterns, P_1 and P_2, as in rule 1 of Figure 9A. Suppose further the binding is between agent X in P_1 and agent Y in P_2. If embedding the left hand side of the rule in the mixture results in two complexes that contain an agent of the same type Z and the path from X to Z in P_1 ends at a different site of Z than the path from Y to Z in P_2, the attempted binding reaction should be rejected. Suppose, for example, that embedding rule 1 of Figure 9A in the mixture picks the two complexes depicted in Figure 9C. The attempted binding reaction is then rejcted with $Z = SIN1$ (and also $Z = Rictor$).

This approach eliminates polymers as characterized by a constraint. Such characterization, however, can hardly be general. For example, one might be dealing with cyclical complexes that contain two agents of the same type and local assembly reactions that can generate polymers. In this case, we need to amend the previous constraint to allow complexes that contain two agents of the same type. While this approach eliminates unwanted (and possibly infinite) sets of configurations, it does so at the cost of attempting reactions which then are rejected. It is very effective to additionally generate all Kappa rules that express the final cyclization of a complex. Referring back to Figure 9, rules 1’-3’ could be added with high rate constants, as discussed previously, but the rates for rules 1-3 need no longer be kept low, because of the no-polymerization constraints. To the extent that rules 1’-3’ heighten the likelihood of complex closure, they will diminish the likelihood of rejected reactions by decreasing the number of reactants prone to polymerization. As in many other circumstances encountered before, a modeling environment will be useful that automatically generates cylce closure rules and provides a constraint language for the user to characterize unwanted products.

2.6.1 Global constraints and context free languages

Why not define the Kappa language to express directly constraints of the kind discussed in the previous section. For example, to express the no-polymerization condition, we would need to assert that a component on the left hand side of a rule does contain an agent of a certain type (such as `Rictor`). However, such an assertion is *global* (non-local). An assertion is *local*, if the (computational) cost of checking its validity is bounded by a constant that does not depend on the size of the system, but only on the size of the left hand side of the rules that define a given system. This constant can therefore be computed *a priori*. To test whether an expression matched by the pattern `A()` on the left hand side of a rule does or does not contain an agent of type `B` requires exploring the whole graph (complex) to which the matched agent belongs. In the worst case this exploration extends to the entire mixture, because the complex might comprise the whole mixture (as in the case of a percolating polymer). The worst-case cost of the exploration then scales with the size of the system. Likewise, the assertion that agent `A` is bound to agent `B`, without specifying an actual path between `A` and `B`, is non-local, since we may have to scan through the whole mixture to check whether there is (or is not) a path from `A` to `B`. In Kappa, an assertion that `A` and `B` are in the same complex must specify a path of bindings from `A` to `B`. A test whether an expression satisfies that path is then bounded by the size of the path we specified on the left hand side of the rule. A language for which a matching condition can be verified with a worst-case cost independent of the size of the reaction mixture is called "context-free". In computer science, a language is called context-free if recognizing a pattern expressed in that language does *not* scale with the size of the context surrounding the pattern. This has nothing to do with the expressive power of the language. FORTRAN is a context-free language capable of expressing any computable function. Kappa is all about managing and reasoning over contextual dependencies in biomolecular interactions. On the other hand, a language like BNGL that allows statements such as "`A` is bound to `B`" (notated as $A.B$) or "`A` is not bound to `B`" (notated as $A + B$) is not context-free, i.e. non-local.

There are several reasons to prefer a context-free language for expressing biomolecular interactions.

1. Kappa provides a clean separation of local from non-local constraints, in which Kappa is used to express local constraints on interaction, while non-local constraints are handled by separate directives to the simulation engine. Since BNGL allows non-local constraints to be stated directly in the language, a user would have to be disciplined to stay in the local fragment of the language. A translation from Kappa into BNGL is trivial: We only need to make explicit that the comma (',') in Kappa expressions is noncommittal with regard to connectedness. For every comma-separated complex (connected subgraph) on the left hand side of a Kappa rule, we generate two BNGL rules; one translating the comma into a dot (asserting connectedness), the other translating it into a plus (asserting disconnectedness).

For example:

$$A(b^1), B(a^1), C(d^2), D(c^2) \to \ldots \qquad \begin{cases} A(b^1).B(a^1).C(d^2).D(c^2) \to \ldots \\ A(b^1).B(a^1) + C(d^2).D(c^2) \to \ldots \end{cases} \tag{2}$$

A translation from BNGL to Kappa, however, is not trivial. An expression $A.B$ would have to be expanded into as many expressions as there are possibilities for agents `A` and `B` to be in the same complex. This, in turn, requires computing all possible reachable chemical species, given a rule set and an initial mixture. This is easier said than done. Even assuming this could be done in practice, the number of generated Kappa rules will, in general, scale with system size, reflecting the translation of a global assertion into local statements.

2. The conceptual hygiene that results from separating local and global constraints has a major payoff in the notion of a rule set as a program, a perspective that will be taken up later. Suffice it to say here that programs written in a context-free language can be analyzed in ways that capitalize on concepts and techniques developed over decades of research into programming languages and compilers. Virtually all of the static analysis of Kappa rule systems are possible because of the context-free nature of Kappa.

3. Local interactions provide a more useful substrate for the mechanistic interpretation of processes than global constraints. In Part II we shall take up "local views" as a fundamental technique in reasoning about programs as models. The local view of an agent expresses the internal states of its sites plus the type of bindings it is engaged in. A binding type consists of the name of the agent at the other end of a bond and the name of the site accepting the link, but not its internal state. Local views underlie much of the program analysis tools we describe in the follow-up paper II. There is a fragment of Kappa in which locality is pushed to the agent level and interaction is local in the strong sense of not depending on any prior coordination between the interacting agents, but solely on their disposition to engage in a binding based on their local views. We shall discuss an evolutionary interpretation of strict locality in Part II.

2.7 Summary

In this section we defined and discussed a language, Kappa, for the representation of biomolecular interactions. The language attempts to do for molecular biology what chemical notation did for chemistry.

- Agents represent indivisible units that contain sites with internal state and binding state. Sites represent interaction capabilities defined by rules (see below). The totality of sites constitutes the interface of an agent. Binding between agents leads to the formation of connected sets of agents, which we call complexes.

- Rules rewrite sets of agents. The left hand side of a rule is a pattern defined in terms of partially specified agents describing the context required for the rule to apply. The right hand side of a rule shows the pattern resulting from its application. The use of patterns represents a "don't care, don't write" assumption.

- The actions of a rule are the differences between its right and left hand side. The primitive action are the modification of an internal state, the modification of a binding state, the introduction and the deletion of an agent.
- Kappa is a context-free language that separates local conditions from global constraints. The cost for identifying an instance matched by the left hand side of a rule only depends on the size of the left hand side of a rule, but not on the size of the expression to which the rule is applied.
- The utility of Kappa consists in formalizing observations about protein-protein interaction mechanisms, enabling easy updating, and providing an operational semantics that makes such formalizations executable.

Appendix

A Matching and replacing a pattern

A.1 Matching a pattern

We formalize the notion of "being conformant" (matching) as a satisfaction relation $\models$. Symbols refer to the corresponding syntactical categories as in the agent Definition 2.1. The specificity ranking of binding states is such that '?' (unknown) subsumes 'ϵ' (free) and '$-$' (bound), and '$-$' (bound) subsumes a binding label indicating a specific bond to an agent identified in the expression. Likewise, the specificity ranking of internal states is such that 'ϵ' (unspecified) subsumes any specified state. In symbols:

binding state:
$$\begin{tikzcd} & \epsilon & \\ ? \arrow[ur] \arrow[dr] & & \\ & - \arrow[r] & \lambda \in \mathbb{N}, \end{tikzcd}$$
internal state:
$$\begin{tikzcd} \epsilon \arrow[r] & \iota \in \mathbb{V} \end{tikzcd}$$

where the arrow means "is a superset of" (or, equivalently "is less specific than"): $x \rightarrow y \equiv x \supseteq y$. Equality applies between two λs that are identical in value. Of course, we have $\epsilon = \epsilon$ and $? = ?$ (question marks). In the following, a fraction denotes an inference from the precondition (in the numerator) to the postcondition (in the denominator), i.e., $\frac{A}{B}$ means "if A then B".

Definition A.1 (Conforming, Matching)**.** To establish whether E' conforms to (matches) E, $E' \models E$, apply the following criteria:

(i)	site match	$n_{\iota'}^{\lambda'} \models n_{\iota}^{\lambda}$, if $\lambda' \subseteq \lambda$ and $\iota' \subseteq \iota$
(ii)	empty interface	$\sigma' \models \emptyset$
(iii)	interface	$\dfrac{s' \models s \quad \sigma' \models \sigma}{s', \sigma' \models s, \sigma}$
(iv)	agent name	$\dfrac{\sigma' \models \sigma}{N(\sigma') \models N(\sigma)}$
(v)	empty expression	$E' \models \varepsilon$
(vi)	expression	$\dfrac{a' \models a \quad E_l \models E_r}{a', E_l \models a, E_r}$

Definition A.1 is understood as a relation between literal expressions (strings of text), established by stepping through the strings E' and E from left to right. However, finding a match of E' to E may necessitate the inspection of several structural equivalences of E', generated by reordering agents, interfaces, and relabeling bonds using Definition 2.4. It is not part of Definition A.1 to produce such a reordering; rather, the reordering is an implicit input through the literal form of E'.

Embedding a graph into another graph. An expression E' can be represented by a graph

$G' = \llcorner E' \lrcorner$, in which nodes are agents identified by their sequential position in the expression. The structural equivalence between two expressions E_1 and E', $E_1 \equiv E'$ (Definition 2.4) then induces an isomorphism *iso* between the corresponding graphs $G_1 = \llcorner E_1 \lrcorner$ and G'. A match $E' \models E$ asserts that E is a subsequence of E' at the level of agents (Definition A.1 (i) also requires that the sites mentioned in an agent of E' have an internal state and binding state at least as specific as those of the sites in the corresponding agent of E). We can think of a match $E' \models E$ as a projection *proj* that removes agents in $G' = \llcorner E' \lrcorner$. Thus, $G_1 \xrightarrow{iso} G' \xrightarrow{proj} G$ defines an embedding of G into G_1. There may be several isomorphisms *iso* that generate distinct matches (and corresponding graph embeddings).

We go from graphs to expressions by uniquely labeling agent nodes and bonds with natural numbers. A labeled graph G then represents an expression $E = \ulcorner G \urcorner$, in which agents are written out in the order of their labels. An embedding between two graphs is given by two functions, one (f_1) for agents, the other (f_2) for bonds. f_1 is the composition of an isomorphism *iso* and an injection *inj* that maps nodes of a smaller graph into nodes of a larger graph, while preserving agent identities: $G_2 \xrightarrow{iso} G \xrightarrow{inj} G'$. There may be several isomorphisms *iso*, each of which generates an embedding. The pair (iso, f_2) constitutes a proof of structural equivalence between $E_2 = \ulcorner G_2 \urcorner$ and E, while $G \xrightarrow{inj} G'$ corresponds to the fact that $E' \models E$. We denote a particular embedding $\phi = iso \circ inj$ of G_2 into G' by $G_2 \lhd_\phi G'$.

A.2 Replacing a pattern

The execution of a rule $E_{\text{left}} \rightarrow E_{\text{right}}$ consists in testing whether an expression S conforms to E_{left}, $S \models E_{\text{left}}$, as defined in Appendix A.1, and then overwriting (updating) the matching region in S with E_{right}. Typically, the expression S represents the contents of a reaction mixture. Here we formalize what it means to overwrite an expression E_l with another expression E_r, $E_l[E_r]$. The definition of replacement below makes use of a "null"-agent $\emptyset$ for the purpose of describing agent deletion and addition. However, we have not defined a null-agent in Definition 2.1. Instead, we shall use the following convention. Let *prefix* be the longest common left-anchored substring between the lhs and the rhs in the rule *lhs* → *rhs*, as described in section 2.5. Let L (R) be the remainder of *lhs* (*rhs*) after the *prefix*, thus, *lhs* = *prefix,L* and *rhs* = *prefix,R*. For the replacement rules to add the agents in R and delete those in L, we pad the rule with appropriately placed null-agents, $|R|$ null agents on the left and $|L|$ on the right:

$$\mathit{prefix}, L, \underbrace{\emptyset, \ldots, \emptyset}_{|R| \text{ times}} \longrightarrow \mathit{prefix}, \underbrace{\emptyset, \ldots, \emptyset}_{|L| \text{ times}}, R.$$

Proper execution of replacement must avoid capturing (i.e. duplicating) bond labels that exist elsewhere in E_l. Our implementations automatically avoid capture by relabeling using Definition

2.4.

Definition A.2 (Replacement).

(i)	overwrite state	$n^{\lambda_l}_{\iota_l}[n^{\lambda_r}_{\iota_r}] = n^{\lambda_r}_{\iota_r}$
(ii)	internal state unchanged	$n^{\lambda_l}_{\iota_l}[n^{\lambda_r}] = n^{\lambda_r}_{\iota_l}$
(iii)	interface unchanged	$\sigma[\emptyset] = \sigma$
(iv)	overwrite interface	$(s, \sigma)[s_r, \sigma_r] = s[s_r], \sigma[\sigma_r]$
(v)	overwrite agent	$N(\sigma)[N(\sigma_r)] = N(\sigma[\sigma_r])$
(vi)	agent deletion	$N(\sigma)[\emptyset] = \emptyset$
(vii)	agent introduction	$\emptyset[N(\sigma_r)] = N(\sigma_r)$
(viii)	expression unchanged	$E[\varepsilon] = E$
(ix)	overwrite expression	$(a, E)[a_r, E_r] = a[a_r], E[E_r]$

When interpreting the action of rules, the reader is urged to fully understand the disambiguation of rules, see appendix B.

A.2.1 Contextual rules

Recall from section 2.5 that we allow two contextual rules: deletion of an agent and (for convenience) the release of an unspecified bond. When deleting an agent $N(\sigma_r)$, we need to identify a complex (if any) that matches $N(\sigma_r)$. Once the complex is identified, all bonds from agents in the complex to N are broken, and the instance N is deleted from the reaction mixture. We contextualize a deletion rule *in situ* - that is, we replace it with an instance that explicitly mentions the complex matching $N(\sigma_r)$. The right hand side of that rule instance then replaces the match of the left hand side as per Definition A.2. The same holds for releasing an unspecified bond (semi-link). Consider, for example, $A(x^-) \to A(x)$. The bond at x of the agent matching $A(x^-)$ is followed in the reaction mixture to identify the actual complex that this agent is a member of. That complex replaces the $A(x^-)$ to create a temporary contextualized unbinding rule. The action of this rule is executed as described in definition A.2.

B Interpreting a rule

B.1 Disambiguating the meaning of a textual rule

The interpretation of textual (line-oriented) rules seems obvious. Until it isn't. For example, Occam would read the reaction rule

$$A(x_u), B(x) \to B(x), A(x_p) \tag{3}$$

as meaning that the state at site x of agent A has changed from u (unphosphorylated, say) to p (phosphorylated, say). Moreover, the change of state requires the presence of $B(x)$, which appears to be a catalyst, since it occurs on both sides. But how does Occam know? The rule might means something quite different: (1) delete agent $A(x_u)$ and (2) delete agent $B(x)$ and (3) add a new agent $B(x)$ and (4) add a new agent $A(x_p)$. Occam will say that this isn't the simplest interpretation. To which we reply that we need to know which agents on the right of a textual rule correspond to which agents on its left, and simplicity is not enough to establish such a correpsondence. For example, identifying a maximal match between the right and the left hand side of a rule may not be unique. Even in the case of chemical reaction rules it is occasionally unclear where a particular carbon atom in a reactant ended up on the product side. If we wish to use the rewrite notation $E_{\text{left}} \to E_{\text{right}}$, we must supply a mapping. For example, we could assign to each agent an index, such as in

$$A_1(x_u), B_2(x) \to B_2(x), A_1(x_p).$$

This rule now says what Occam would like it to mean: in the presence of $B(x)$, change the state of A at x from u to p. Indexed agents with no match on the left or the right hand side then correspond to additions or deletions, respectively. However, writing an explicit correspondence between agents on both sides soon becomes tedious, unless supplied automatically. Indeed, our graphical rule representation (Appendix C) uses automatic indices. To generate an index assignment in textual input mode, we opted for a simple convention: both sides of a rule, E_{left} and E_{right}, are compared with one another proceeding from the left of each expression. The comparison only checks agent names and interfaces, but is blind to the states of the sites mentioned. It ends at the first difference. This procedure identifies a longest left-anchored substring – a prefix – common to both expressions. (This prefix may be empty.) The prefix now establishes a sequential correspondence between agents on the left and right hand sides of a rule. Anything after the common prefix is interpreted in terms of deletions and introductions of agents, depending on whether an agent is missing on the right or left hand side, respectively. The longest common prefix strategy avoids additional notation for explicit indexing but requires some care in arranging the agents. These issues rise to the user's attention only when using textual input. A graphical interface (Appendix C) hides them completely.

An example may help.

$$A(x^1), B(x^1, y_u) \to A(x^1), B(x^1, y_p) \qquad \{\text{change state of } B \tag{4}$$

The common prefix in rule (4) establishes a correspondence between the agents mentioned on the left and the right. This rule states that if agent A is bound at site x to B at site x and B is unphosphorylated at site y (more precisely, "site y is in state u"), B will be phosphorylated at y – a common situation in signaling.

Let us now replace $A(x^1), B(x^1, y_p)$ with $B(x^1, y_p), A(x^1)$. By themselves, these expressions denote the same graph or complex. However, in the context of a rule, where a correspondence between agents on both sides has to be established to represent a set of actions, the structural equivalence Definition 2.4 is suspended. The left and the right hand side of the rule have no common prefix, which triggers the addition and deletion actions:

$$A(x^1), B(x^1, y_u) \to B(x^1, y_p), A(x^1) \qquad \begin{cases} \text{delete the } A \text{ referenced on the left} \\ \text{delete the } B \text{ referenced on the left} \\ \text{add a } B(x, y_p) \\ \text{add an } A(x) \\ \text{bind } B(x, y_p) \text{ at } x \text{ to } A(x) \text{ at } x \end{cases} \tag{5}$$

Notice that the *local* outcome of rules (4) and (5) is exactly the same: a binding between the sites x of A and B, with B phosphorylated at y. However, any particular reaction *instance* arises from matching a reaction mixture to the left hand side of a rule – see appendix A.1 and Figure 5. As a consequence, a rule's actions can reach well beyond its local context. For example, the $A(x^1), B(x^1, y_u)$ on the left of rule (5) might match the molecular species $A(x^1, z^2), B(x^1, y_u), C(t^2)$ in which A is also bound to C. Upon execution, rule (5) will effectively replace $A(x^1, z^2), B(x^1, y_u), C(t^2)$ with $B(x^1, y_p), A(x^1)$ (or $A(x^1), B(x^1, y_p)$, which is the same), leaving an unbound C. Rule (4), in contrast, will only change the state of B at y without disconnecting C.

On their own, the patterns on the left and the right hand sides of rules are normal Kappa expressions, whose grammar is laid out in definitions 2.1, 2.2, 2.3, and 2.4. However, our choice of forgoing explicit indexing to keep track of all agent instances in an expression, creates the need for mapping agent identities across the arrow in order to specify the action(s) associated with a given rule. This suspends part (xiii) of the structural equivalence, definition 2.4, when patterns appear in a rule.

One might use an alternative notation, in which a rule is a pattern together with a set of actions on the pattern. While this does not eliminate the mapping issue (we still need to know which agent an element of the action refers to when the pattern contains multiple agents of the same type), it does have the advantage of making actions explicit. In contrast, the arrow notation of a rule requires an inference of the action by comparing left and right hand sides. We retain the arrow notation because of its ubiquity throughout chemistry and molecular biology.

B.2 Subtleties

B.2.1 Patterns and agent cards

The syntax of patterns is the same as that of Kappa expressions. Patterns often make use of the semi-link and "don't-care" notation for binding states, but other than that a pattern looks like an expression representing a mixture. How can one tell, then, whether a Kappa expression is a pattern or a mixture? One cannot on the basis of the expression alone. There is no way to know which information a pattern omits unless we compare it to some definition of that agent. Such definition consists in the specification of all sites of an agent and serves as a reference. We call it an "agent card". However, the definition of an agent is typically only temporary, since it can only document what we currently know (or hypothesize) about that agent and not what we might know (or hypothesize) tomorrow. If, for example, Prof. Kappa's laboratory discovers that mTOR has a binding site for PI3K, the agent card of `mTOR(...)` should be updated to include that binding site, which she might name `pi3k`: `mTOR(pi3k, ...)`. It is important to understand that the agent card is *not* a concept known to Kappa, the language, but rather a useful concept in the *practice* of Kappa, as defined, for example, by a modeling environment. The update of the agent card turns all previously existing expressions that mention agent `mTOR` into patterns as far as site `pi3k` is concerned, since no previously existing rule could have possibly mentioned that site. Likewise, it seems meaningless to specify for each site all its possible binding states or internal states, as they depend on the set of rules that modify that site. It makes more sense to compute the *accessible* states of a site in the context of a specific set of rules and initial mixture (i.e., a model) rather than for an isolated agent card. The fact that the meaning of Kappa expressions is relative to the current interface of an agent, as defined in its card, is a natural reflection of the temporary nature of knowledge. The fact that it is so easy to update an agent without necessarily touching all the prior rules, is a strength of rule-based representations.

B.2.2 Agent creation

It is important to keep in mind the distinction between a formal language and a practice of utilizing that language for a particular purpose. In pure Kappa, it is perfectly legal to say:

$$A(x) \to A(x, y) \tag{6}$$

Notice, however, that the left and right hand side of this rule have *no* common prefix, because the agent A on the left has a different interface than the agent A mentioned on the right. The rule thus means (1) the deletion of A from whatever complex was matched by the left hand side. (The matching instance of A in the mixture might be bound to a B at A's unmentioned site z – recall: "don't care, don't write"), and (2) the creation of a new agent $A(x, y)$ with exactly the interface declared in the rule. The so-created agent will not be able to bind anything at site z (like the instance of A that has just been deleted), since it has no site z! The interpreter of Kappa does not – and should not – know about agent cards. As emphasized previously (B.2.1), agent cards are concepts that belong to the practice of Kappa, not the language definition. It is therefore sound practice not to take short-cuts when declaring the creation of an agent, and to specify the full interface with the sate of all its sites. Be aware, however, that when agent cards change, what once was a "full" specification, may no longer be one. In any practical setting, the Kappa language must be augmented with some knowledge management system.

B.2.3 Modifying unspecified states

Care is needed when modifying the internal state of an agent at a site whose state is left unspecified on the left. For example,

$$A(x) \to A(x_p) \tag{7}$$

is a perfectly legal rule. It asserts that regardless of the internal state at x, that site becomes phosphorylated. While this rules expresses the chemistry we intend, it does not behave with the kinetics we intend! The simulation of a rule-based system follows an analogous continuous time Monte-Carlo scheme as the one proposed for networks of fully specified reaction instances by Doob and also by Gillespie. (Our generalization for rules is sketched in a later section.) To understand why the kinetics of rule (7) differs from

$$A(x_u) \to A(x_p), \tag{8}$$

which has the proper kinetics, consider a mixture of $A(x_u)$ and $A(x_p)$. The $A(x)$ in (7) instantiates to both $A(x_u)$ and $A(x_p)$, while rule (8) only instantiates to $A(x_u)$. Thus, rule (7) effectively consists of two rules,

$$A(x_u) \to A(x_p) \qquad \text{and} \qquad A(x_p) \to A(x_p). \tag{9}$$

While the second rule does not change the mixture, it does formally constitute a reaction that will advance the simulated laboratory time. Compared to a system operating only under rule (8), which has the intended kinetics, rule (7) will appear dramatically slowed down, particularly when only a few $A(x_u)$ are left in the mixture.

B.2.4 What we cannot speak about, we must pass over in silence

Certain actions cannot be expressed with the current rule syntax of Kappa. For example, try to say that an `A(x)` is deleted and a new `A(x)` is created at the same time.

C The importance of graphical interfaces

Local rules tame the combinatorial explosion by specifying only those aspects of agents that are pertinent to an interaction. Therein lies the power of rules, and therein lie the subtleties of their interpretation, which are intrinsic to their textual representation. We rarely have difficulties in mapping agents between the left and the right hand side of chemical reaction rules. (Although sometimes it isn't immediately obvious where certain carbon atoms of the reactants ended up in the products.) The nuisance of rigorously applying a scheme for writing rules (the prefix scheme above) leads to errors. Computer scientists have long understood that graphical user interfaces are a very powerful means for avoiding such errors by relieving the user from having to be aware of the tricky nature of linear syntactical constructs. The tools built around the Kappa language therefore come with an interface for building rules. The rule builder allows the user to express the content of a rule graphically. By its very nature, a graphical expression provides cognitive aid for writing syntactically correct rules. For example, by drawing an edge between two sites, we instinctively understand that these sites are now "busy" and cannot accept another bond. In a textual representation we would have to be careful not to use the same label for two edges. Although mistakes of this sort will be detected by the parser, the parser cannot know whether what we say is what we mean, as long as what we say is grammatically correct. This is where a graphical rule builder becomes indispensable by forcing us to be explicit, and then putting out textual rules that mean what we say graphically. Figure 10 provides an example in conjunction with rule (4) above. A row is a rule, as it is being designed step by step. The two panels on each row stand for the left and the right hand side of the rule. We begin building rule (4) in Figure 10A by first adding agent $A(x)$ to the left hand side. The rule builder will automatically copy agent A to the right, making the assumption that *this* A will still exist after the reaction. If that is not what we want, we can go the right hand side and "delete" A. In this case, the rule builder

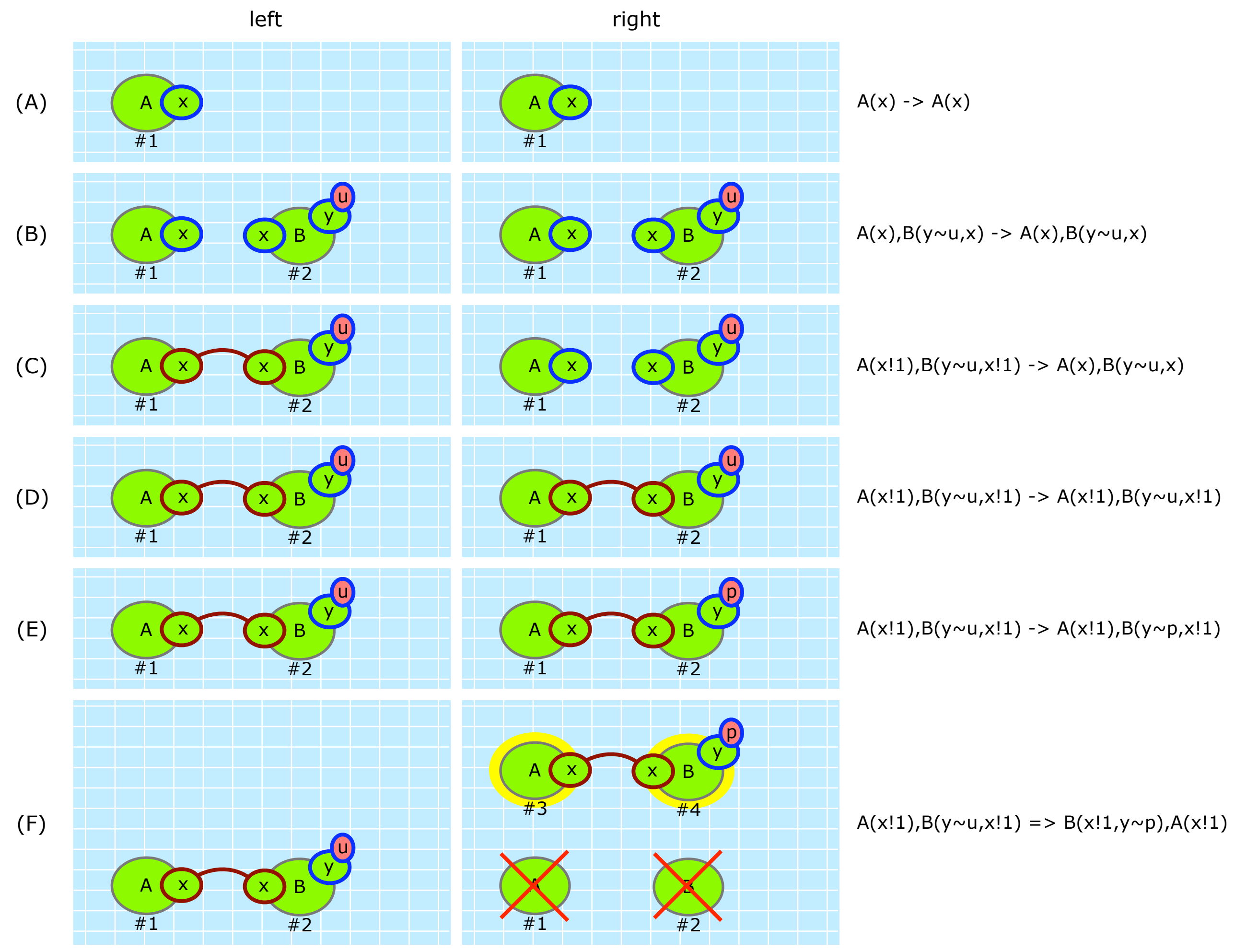

left
right
(A)
(B)
(C)
(D)
(E)
(F)
A
x
B
y
u
p
#1
#2
#3
#4
A(x) -> A(x)
A(x),B(y~u,x) -> A(x),B(y~u,x)
A(x!1),B(y~u,x!1) -> A(x),B(y~u,x)
A(x!1),B(y~u,x!1) -> A(x!1),B(y~u,x!1)
A(x!1),B(y~u,x!1) -> A(x!1),B(y~p,x!1)
A(x!1),B(y~u,x!1) => B(x!1,y~p),A(x!1)

Figure 10: Graphical interface. The figure depicts the building of rule (4) using the interface of the KappaFactory. See text for details.

will strike out A (as in Figure 10F) indicating that *this* A will be deleted. This is different from deleting A on the left, which would cause the rule builder to erase A from the rule alltogether. For transparency, the rule builder also associates agents with numbers (immediately below each agent in Figure 10) to indicate the mappings between agents on the left and the right made on our behalf. Next, Figure 10B, we add agent configuration $B(x, y_u)$ to the left. Again, the agent is automatically copied to the right. We then connect A and B at their respective sites x, to build a complex on the left hand side (Figure 10C). With this move we are only changing the state of existing agents on the left. The rule builder will not parrot this change on the right, because that would be assuming too much. After all, we might want to express the dissociation of A and B. (Intending to change the state of agents is much more likely than intending to delete or add them.) In the next step, Figure 10D, we therefore add the edge between A and B on the right. Finally, we edit the state at site y of B on the right from u to p, Figure 10E. At each step, the rule builder also displays the corresponding textual rule. Indeed, the rule designed in Figure 10E is output as rule (4). It was virtually impossible to make a mistake. If, on the other hand, our intent was the set of actions associated with rule (5), the final graphical rendering would look unmistakeably different, as shown in Figure 10F, and would have required a correspondingly different set of design steps. Notice that rather than swapping the order of the agents on the right, the rule builder used a different label for the edge between A and B on the right, thus preventing a common prefix and causing the interpretation as in (5). Also, the rule builder will indicate with a `=>` when a rule is not elementary.

The graphical front end is particularly useful when connected to an annotated database of currently defined signaling agents and their sites. This enables additional checks to alert the user upon reference to sites and/or agent names unknown to the database.