A Syntactic Abstraction for Rule-Based Languages with Binding

Michael Pedersen[1,2]

Department of Plant Sciences
University of Cambridge
Cambridge, England

Abstract

Rule-based languages such as Kappa and BioNetGen excel in their support for handling combinatorial explosion of the number of protein complexes encoded by a signal transduction model. They do so by representing the transformation of complexes at the level of protein binding sites. However, this detailed level of representation can make models cumbersome to write and difficult to read. This paper introduces a syntactic abstraction of binding, away from specific sites, which in many cases results in shorter and more concise rules.

Keywords: Rule-based languages, Kappa, BioNetGen, protein complexes, binding

1 Introduction

The past decade has seen the emergence of many new formal languages and frameworks for modelling in biology. These include the so-called rule-based languages in which a rule generally describes one or more biological reactions in a compact manner. Two such language, Kappa [2] and the closely related BioNetGen [4], describe the transformation of protein complexes at the level of binding sites. For example, a rule for adding a binding between two proteins can take the following form in Kappa:

```
A{s}, B{t~p} -> A{s!1}, B{t~p!1}
```

[1] Email: mdp40@cam.ac.uk

[2] This work was supported by an EPSRC Postdoctoral Fellowship (EP/H027955/1).

Here `A` and `B` are proteins with binding sites `s` and `t`, respectively. The expression `t~p` indicates phosphorylation on `t`; other modification values, such as `u` for unphosphorylated, are also possible. The exclamation mark expressions `s!1` and `t!1` on the product side indicate a binding. The integer label identifies the bond, and different choices of integer labels are needed in the general case for writing complexes with more than one binding. When no binding expression is given, as in the reactants, this indicates that the sites are not bound.

The power of this approach arises from a "don't write don't care" policy: the above rule can be applied to any `A` and any phosphorylated `B` not already bound on the specified sites, but which may or may not be phosphorylated or bound to other proteins on other sites. An example application of the rule is shown in Figure 1 a). In the context of signal transduction pathways in particular, rules typically give rise to a combinatorially large number of concrete reactions. Traditional models based on concrete reactions, or on ordinary differential equations, are hence difficult both to write and to simulate, and may therefore be forced to rely on (possibly unfounded) assumptions for model reduction. In contrast, rule-based models are concise and can be simulated on-the-fly, without generating large sets of concrete reactions [3,9].

Despite the conceptual simplicity, large rules with many bindings can be cumbersome to write and difficult to read. Take for example the following Kappa rule from an EGFR model [2]:

```
EGFR{Y1068~p!1},Grb2{SH2!1,SH3!2},SoS{a!2,b},Ras{S1S2~gdp}
  -> EGFR{Y1068~p!1},Grb2{SH2!1,SH3!2},SoS{a!2,b!3},Ras{S1S2~gdp!3}
```

The rule essentially just expresses the binding of `SoS`, in a certain complex, to `Ras`. But this essence is not immediately apparent due to the elaborate specification of binding sites and bond labels. In this paper, we propose a syntactic abstraction of Kappa, away from specific binding sites, which we call $\mathcal{AL}$. This allows us to write the above rule in the following simpler form:

```
EGFR{Y1068~p}-Grb2-SoS + Ras{S1S2~gdp}
  -> EGFR{Y1068~p}-Grb2-SoS-Ras{S1S2~gdp}
```

The brackets now only specify modifications. A translation to Kappa then automatically deduces appropriate binding sites. The "don't write don't care" policy is preserved, so that the rule can be applied to proteins in any additional binding context.

The above simple abstraction of binding is adequate in many cases, but must be supplemented with further constructs in the general case. In Section 2 we outline these further constructs of $\mathcal{AL}$ informally, with examples from a case study EGFR signalling model adapted from [2]. In Sections 3 and 4 we formally define $\mathcal{AL}$ in terms of an abstract syntax and a translation to Kappa, respectively. The translation is implemented as part of the LBS tool [8]. We end in Section 5 with a brief discussion and possible future directions.

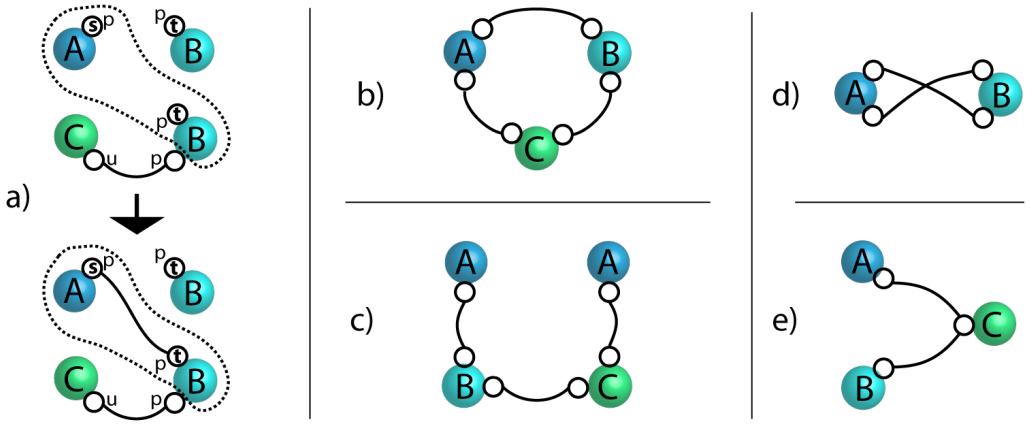

Figure 1. Graphical representations of protein complexes. The small circles are binding sites with names shown inside and modifications outside when relevant. a) One of two possible applications of an A-B binding rule. b) A circular complex. c) A complex with two occurrences of the same protein. c) A dimer with two bindings. d) Two proteins binding to the same site of a third.

Related work has addressed the problem of adding structure to Kappa models as a whole. Little b [6] and LBS [8,7] both introduce a general notion of modularity with support for writing Kappa models, and an extension of Kappa with a perturbation language is introduced in [1]. None of these, however, address the representation of binding itself.

The graph-based nature of complexes in Kappa naturally lends itself to graphical creation and visualisation of rules, supported by tools such as RuleBase[3]. In a graphical context, the abstraction of bindings proposed in this paper may be of less relevance. We stress however that textual representations remain important: they are preferred by some for efficient model development, and they are central to efforts to introduce further structure, such as modularity, into models.

2 An Informal Overview of $\mathcal{AL}$

Linearisation of Complexes. The complexes used in the above rules are linear in their binding structure, which naturally allows them to be written on the form `A-B-...-C`. This seems to be adequate in many cases, in particular since Kappa rules often only specify partial complexes. However, complexes in the general case have a graph structure, and $\mathcal{AL}$ represents this as a set of bindings, separated by the ' symbol. For example, the circular complex shown in Figure 1 b) can be represented by the expression `A-B'A-C'C-B`. Linear complexes are then just abbreviations for pairs of bindings, so e.g. `A-B-C` abbreviates `A-B'B-C`. The different occurrences of `A`, `B` and `C` in these expressions refer to the same protein. However, to cater for complexes with multiple occurrences of the same protein, proteins are labelled with integer *instance numbers.* For

[3] www.rulebase.org

example, the complex shown in Figure 1 c) can be written as `A_1-B'B-C'C-A_2`. When an instance number is omitted, a default of 1 is assumed. An example from our case study EGFR model is dissociation of bound receptor dimers:

```
EGF–EGFR–EGFR_2–EGF_2 -> EGF–EGFR + EGFR_2–EGF_2
```

Similar problems have been addressed for chemical compounds, see e.g. [10].

Differentiation and Combination of Sites. In the above examples we assume that two proteins `A` and `B` can bind on exactly one site, so that the binding `A-B` is uniquely defined. That is generally not the case: they could e.g. bind on two different sites as in Figure 1 d). Hence $\mathcal{AL}$ has a notion of *site numbers*, allowing two bindings to be written as `A^1-2^B` and `A^2-1^B`. When site numbers are omitted, a default of 1 is assumed. In our EGFR case study, the phosphatase `MKP3` can bind `ERK` on two distinct binding sites:

```
MKP3 + ERK{T185~p} -> MKP3–ERK{T185~p}
MKP3 + ERK{Y187~p} -> MKP3–2^ERK{Y187~p}
```

On the other hand, it may be that `A` and `B` bind a third protein `C` on the same site, as shown in Figure 1 e). $\mathcal{AL}$ addresses this with a notion of site renaming. The expressions `binding A{s}-B` and `binding A{s}-C` state that `B` and `C` both bind `A` on the same site, called `s`. This may also be used simply to spell out the names of sites, in place of those generated automatically by the translation to Kappa. An example from our EGFR case study is the binding of the kinase `MEK` and phosphatase `MKP3` to `ERK` on the same site:

```
binding ERK{T185}–MEK
binding ERK{T185}–MKP3
```

Subsequent `ERK-MEK` and `ERK-MKP3` complexes are bound on the same site, named `T185`. As before, a site number can optionally be included.

Whether two proteins bind each other on multiple sites, and whether two proteins bind a third on the same site, is important information which is explicit in $\mathcal{AL}$ models. In contrast, this is implicit in Kappa models and is often revealed by a so-called *contact map* obtained through analysis.

Binding Exclusion. Sometimes we need to explicitly *exclude* a binding. We do this in $\mathcal{AL}$ using the `\` operator:

```
EGFR\EGFR ->
```

This specifies that an EGFR receptor can degrade when not already bound to another receptor, although it may or may not be bound to e.g. a ligand. A site number can be given for the exclusion, so e.g. `A\(B,2)` specifies that A is not bound on its second site to B.

There are cases where we *implicitly* assume proteins not to be bound, namely in complex formation rules such as the following:

```
EGFR{Y1148~p} + Shc -> EGFR{Y1148~p}–Shc
```

Because EGFR and Shc are bound on the product side and no binding is specified on the reactant side, we consider this an abbreviation for writing out the exclusion of bindings as follows:

```
EGFR{Y1148~p}\Shc + Shc\EGFR -> EGFR{Y1148~p}-Shc
```

A similar convention applies to rules with dissociation.

Reactant-Product Pairing. We finally turn our attention to the mechanism by which proteins on the reactant and product sides of rules are paired up. The difficulty lies in rules with more than one occurrence of a particular protein. Take for example the following contrived rule:

```
A + B + B -> A-B + B
```

The rule binds protein A to one of two B proteins, but which one? The choice could be based on the order of occurrence in rules, but that would give rise to a non-commutative + operator which can result in subtle errors; we discuss this in the context of Kappa in Section 4. We therefore adopt an alternative approach of pairing based on protein instance numbers, which as previously discussed are already used for distinguishing different occurrences of the same protein within complexes. We simply extend the scope of instance numbers to entire rules. The above rule could then be disambiguated as follows:

```
A + B_1 + B_2 -> A-B_1 + B_2
```

Furthermore, when a rule involves creation or deletion of proteins, we explicitly label the proteins with the *creation marker*, +, or the *deletion marker*, -, respectively. A contrived rule for deleting an agent B, creating an agent C, and phosphorylating an agent A, can then be written as follows:

```
(B_-) + A{s~u} -> (C_+) + A{s~p}
```

3 The Abstract Syntax of $\mathcal{AL}$

We start by giving an abstract syntax of Kappa based on the presentation in [5], but adapted for our purposes. We adopt the terminology of Kappa, and use the term *agents* as the abstraction for proteins in the language. For the sake of brevity, we omit initial conditions from both Kappa and $\mathcal{AL}$.

We let $\mathbb{N}$ denote the set of natural numbers and let $h, i, j, k \in \mathbb{N}$. We let $\mathbb{R}$ denote the set of real numbers and let $q \in \mathbb{R}$. We write $\{x_i\}_I$ for finite indexed sets and omit both the index and the set when these are unimportant. As a special case, we write $\{x_i \mapsto y_i\}$ for finite partial functions f with $f(x_i) = y_i$. The domain of definition and the image of a function f are denoted by $\mathrm{dom}(f)$ and $\mathrm{im}(f)$, respectively. We write $\underline{x}$ for a list with the ith element denoted by x_i, or simply x when the particular index is unimportant. We assume given a countably infinite set of agent names ranged over by N, a countably infinite

p_κ	$::= \{r_\kappa\}$	Program
r_κ	$::= \underline{a_\kappa} \to^q \underline{a'_\kappa}$	Rule
a_κ	$::= N(\sigma)$	Agent
σ	$::= \underline{n \mapsto (\iota, \lambda)}$	Sites
ι	$::= \epsilon_\iota \mid m$	Internal state
λ	$::= \epsilon_\lambda \mid ? \mid i$	Binding state

Table 1
The abstract syntax of Kappa.

set of site names ranged over by n, and a set of internal states ranged over by m. The abstract syntax for Kappa is then given in Table 1.

A Kappa program p_κ is a set of rules. A rule r_κ consists of lists of reactant and product agents together with a rate constant q. An agent a_κ consists of an agent name N and a list σ of sites; a site in turn consists of a site name n, an internal state ι and a binding state λ. In the concrete syntax, we write a site in σ on the form $n \sim \iota!\lambda$. The internal state can either be a wild-card, ϵ_ι, meaning "don't care", or a given state m. In the concrete syntax, the wild-card is assumed when no internal state is given, e.g. as in $n!\lambda$. The binding state can be "unbound", written ϵ_λ, which is assumed in the concrete syntax when no binding state is given, e.g. as in $n \sim \iota$. The binding state can also be the wild-card, ?, meaning "bound or unbound", or it can be a specific bond label i.

The abstract syntax for $\mathcal{AL}$ is given in Table 2. An $\mathcal{AL}$ program p has a set of rules like Kappa programs, but it also has a site renaming ϕ. The site renaming is a finite partial function from triples (N, i, N'), representing the ith binding in agent N to agent N', to site names n. In the concrete syntax, the site renaming function is specified by `binding` expressions. A rule r consists of a set of reactant complexes, a rate constant, and a set of product complexes. Contrast this with the lists of atomic agents in Kappa. In the concrete syntax, complexes in the reactant and product sets are separated by the + symbol. A complex c can either be a single agent, or it can be a set of bindings each consisting of two agents together with their respective site numbers. In the concrete syntax, bindings are separated by the ' symbol. An agent a can be an agent name labelled with an instance number and a change marker, together with a modification; it can also be an agent with a binding exclusion for a given agent name and site number. The change marker δ can be a $+$, indicating creation; it can be a $-$, indicating deletion; or it can be a $\bullet$, indicating no change. In the concrete syntax, $\bullet$ is assumed when the change marker is omitted. Finally, a modification is a finite partial function from site names to internal states. This is in contrast to Kappa, where modifications

p	$::= (\phi, \{r\})$	Program
ϕ	$::= \{(N, i, N') \mapsto n\}$	Site renaming
r	$::= \{c\} \to^q \{c'\}$	Rule
c	$::= a \mid \{a^{\,i}\!-^{j}\, a'\}$	Complex
a	$::= N_{(i,\delta)}(\rho) \mid a\backslash(N, j)$	Agent
δ	$::= + \mid - \mid \bullet$	Change marker
ρ	$::= \{n \mapsto m\}$	Modification

Table 2
The abstract syntax of $\mathcal{AL}$.

are represented by lists and also include bindings.

4 The Semantics of $\mathcal{AL}$

We define the semantics of $\mathcal{AL}$ in terms of a translation from $\mathcal{AL}$ programs to Kappa programs. We refer to [5] for a formal definition of the semantics of Kappa. One important point, however, is that Kappa adopts a "longest common prefix" policy for pairing reactant and product agents. Consider for example the following:

```
A{s~u!1}, B{t!1,r!2}, A{s~u!2}, C
 -> A{s~u!1}, B{t!1,r!2}, A{s~p!2}, D
```

The longest common prefix for this rule is `A{s}, B{t,r}, A{s}`. It consists of both agent names and site names. Reactant and product agents which are in the longest common prefix are paired by position and updated accordingly; agents after the prefix on the reactant side (here `C`) are deleted from any context; and agents after the prefix on the product side (here `D`) are created. This can give rise to subtle errors, for example if the order of the sites `t` and `r` were switched on one side of the rule. This problem is addressed in $\mathcal{AL}$ through protein instance numbers, creation markers and deletion markers. The translation from $\mathcal{AL}$ to Kappa orders both agents and the sites within agents appropriately.

We proceed with some preliminary definitions. For any given linearly ordered set $(X, \leq)$ and $x \subseteq X$, we let $\mathrm{sort}_{\leq}(x)$ be the list representation of x with elements ordered according to $\leq$. We write $x \stackrel{\Delta}{\simeq} y$ for definitions where x equals y if y is defined, and where x is undefined otherwise.

The translation of complexes gives rise to "intermediate" modifications τ with either internal states, bindings, or both:

$$\tau \;::=\; \{n \mapsto s\} \qquad\qquad s \;::=\; \iota \mid \lambda \mid (\iota, \lambda)$$

We take the notational liberty of using the same symbols for terms with intermediate modifications τ as for those with standard $\mathcal{AL}$ modifications σ. A partial operator $\circ$ of the form $\tau \circ \tau' = \tau''$ for combining intermediate modifications is needed in the semantics of complexes. It is defined as follows:

$$(\tau \circ \tau')(n) \quad \triangleq \quad \begin{cases} \tau(n) \circ \tau'(n) & \text{if } n \in \mathrm{dom}(\tau) \cap \mathrm{dom}(\tau') \\ \tau(n) & \text{if } n \in \mathrm{dom}(\tau) \backslash \mathrm{dom}(\tau') \\ \tau'(n) & \text{if } n \in \mathrm{dom}(\tau') \backslash \mathrm{dom}(\tau) \end{cases}$$

$$\text{and } s \circ s' \quad \triangleq \quad \begin{cases} \iota & \text{if } \{s, s'\} = \{\iota\} \\ (\iota, \lambda) & \text{if } \{s, s'\} = \{\iota, \lambda\} \vee \{s, s'\} = \{\iota, (\iota, \lambda)\} \end{cases}$$

The operator is only defined when modifications agree: any internal state combines with any binding in the natural manner, but two internal states can only be combined when they are identical; bindings can never be combined, i.e. a particular binding can be specified at most once in a complex. Note that $\circ$ is associative and commutative. We can hence extend it to sets of intermediate modifications in the natural manner.

A second operator $\triangleleft$ of the form $\tau \triangleleft \tau' = \tau''$ on intermediate modifications is needed to ensure that agents which are bound on one side in a rule are either bound or unbound on the other side, as discussed in Section 2:

$$(\tau \triangleleft \tau')(n) \quad \triangleq \quad \begin{cases} \tau(n) \triangleleft \tau'(n) & \text{if } n \in \mathrm{dom}(\tau) \cap \mathrm{dom}(\tau') \\ \tau(n) & \text{if } n \in \mathrm{dom}(\tau) \backslash \mathrm{dom}(\tau') \end{cases}$$

$$\text{and } s \triangleleft s' \quad \triangleq \quad \begin{cases} (\iota, \epsilon_\lambda) & \text{if } s = \iota \wedge \neg \exists \iota'.s' = \iota' \\ s & \text{otherwise} \end{cases}$$

We extend this operator to the form $\{a_i\} \triangleleft \{a_j\} = \{a_k\}$ on sets of agents:

$$\begin{aligned} A \triangleleft A' \quad \triangleq \quad & A'' \cup A''' \text{ where} \\ & A'' \triangleq \{N_{(i,\bullet)}(\tau \triangleleft \tau') \mid N_{(i,\bullet)}(\tau) \in A \wedge N_{(i,\bullet)}(\tau') \in A'\} \\ & A''' \triangleq \{N_{(i,\delta)}(\tau) \in A \mid \delta \in \{+, -\}\} \end{aligned}$$

We define a total function of the form $\kappa(\tau) = \rho$ from intermediate modifications to Kappa modifications as follows:

$$\kappa(\iota) \triangleq (\iota, ?) \qquad \kappa(\lambda) \triangleq (\epsilon_\iota, \lambda) \qquad \kappa(\iota, \lambda) \triangleq (\iota, \lambda)$$

Modifications with only an internal state are extended with the wild card binding, and modifications with only a binding are extended with wild card internal state. The κ function can be extended to a total function of the form $\kappa(a) = a_\kappa$ for translating $\mathcal{AL}$ agents to Kappa agents; we here assume a given linear ordering $\leq_{\mathrm{n}}$ of the form $n \leq_{\mathrm{n}} n'$ on site names:

$$\kappa(N_{(i,\delta)}(\tau)) \quad \stackrel{\Delta}{\simeq} \quad N(\kappa(\tau)) \text{ where } \kappa(\{n \mapsto s\}) \stackrel{\Delta}{\simeq} \text{sort}_{\leq_\text{n}}\{n \mapsto \kappa(s)\}$$

The instance number and change marker are discarded, and the modifications are sorted; we here assume an extension of the sort function to sets of pairs, sorted according to the first element of pairs.

We also assume a given linear ordering $\leq_\text{N}$ of the form $N \leq_\text{N} N'$ on agent names. This is needed to order the agents in reactants and products in a meaningful way with respect to the "longest common prefix policy" of Kappa. We define an extension to the form $N_{(i,\delta)}(\tau) \leq_\text{N} N'_{(i',\delta')}(\tau')$ on agents as the smallest linear ordering satisfying the following two conditions:

$$\begin{aligned} N_{(i,\delta)}(\tau) &\leq_\text{N} N'_{(j,\delta)}(\tau') \quad \text{if } N <_\text{N} N' \vee (N = N' \wedge i \leq j) \\ N_{(i,\delta)}(\tau) &\leq_\text{N} N'_{(j,\delta')}(\tau') \quad \text{if } \delta \neq \delta' \wedge \delta < \delta' \text{ where } \bullet < + < - \end{aligned}$$

That is, agents with the same change marker δ are ordered according to their names (first) and instance numbers (second), and agents with addition/deletion markers are ordered after those without.

Partial site renaming functions ϕ are extended to total functions in the translation. For this we rely on a function of the form $\text{ext}(\phi, X) = \{\theta\}$ where X is a finite set of site names to be excluded in the extensions and $\{\theta\}$ is a set of total site renaming functions θ of the form $\theta(N, i, N') = n$. The definition is as follows, where the operator $\setminus$ denotes set difference:

$$\text{ext}(\phi, X) \quad \stackrel{\Delta}{\simeq} \quad \{\theta \mid \phi \subset \theta \wedge \text{im}(\theta \setminus \phi) \cap (\text{im}(\phi) \cup X) = \emptyset \wedge (\theta \setminus \phi) \text{ is injective}\}$$

The definition ensures that extensions agree with the given ϕ where defined; that additional site names used in the extensions are not in ϕ or in the given set X; and that the extensions are injective outside of the domain of ϕ. The set X should be thought of as the set of site names n in a program p, denoted by $\text{sites}(p)$ in the following and defined in the expected manner.

Finally, a means of generating integer binding labels is needed. For notational convenience, we simply assume a given bijective function of the form $\text{bind}(N, h, i, N', h', j) = k$ where h/h' are instance numbers and i/j are site numbers. The function could e.g. be defined based on a Gödel numbering. The semantics of $\mathcal{AL}$ is then given by a partial denotational function of the form $[\![p]\!]_\text{p} = p_\kappa$ which, together with auxiliary functions for the other categories of the abstract syntax, is defined in Table 3.

The denotational function for rules first applies the denotational function for complexes to each complex on the reactant and product sides and then takes the union of the result, thus obtaining sets A and A' of agents. This union reflects that Kappa does not explicitly aggregate agents into complexes.

$$
\begin{array}{rcl}
[\![(\phi, \{r\})]\!]_{\mathrm{p}} & \stackrel{\Delta}{\simeq} & \{[\![r]\!]_r\theta\} \text{ where } \theta \in \mathrm{ext}(\phi, \mathrm{sites}(\{\mathrm{r}\}) \\
[\![\{c\} \to^l \{c'\}]\!]_{\mathrm{r}}\theta & \stackrel{\Delta}{\simeq} & \kappa(\mathrm{sort}_{\leq_{\mathrm{N}}}(A \triangleleft A')) \to^q \kappa(\mathrm{sort}_{\leq_{\mathrm{N}}}(A' \triangleleft A)) \text{ where} \\
 & & A \stackrel{\Delta}{\simeq} \bigcup\{[\![c]\!]_{\mathrm{c}}\theta\} \text{ and } A' \stackrel{\Delta}{\simeq} \bigcup\{[\![c']\!]_{\mathrm{c}}\theta\} \text{ if:} \\
 & & 1.\ \forall N_{(i,\bullet)}(\tau) \in A\, \exists \tau'.\, N_{(i,\bullet)}(\tau') \in A' \wedge \mathrm{dom}(\tau) = \mathrm{dom}(\tau') \\
 & & \quad \forall {N'}_{(i,\bullet)}(\tau') \in A'\, \exists \tau.\, {N'}_{(i,\bullet)}(\tau) \in A \wedge \mathrm{dom}(\tau') = \mathrm{dom}(\tau) \\
 & & 2.\ \neg\exists N_{(i,+)}(\tau) \in A \text{ and } \neg\exists N_{(i,-)}(\tau) \in A' \\
 & & 3.\ |A{\downarrow}| = \sum\{|[\![c]\!]_{\mathrm{c}}\theta{\downarrow}|\} \text{ and } |A'{\downarrow}| = \sum\{|[\![c']\!]_{\mathrm{c}}\theta{\downarrow}|\} \\
 & & \quad \text{where } N_{i,\delta}(\tau){\downarrow} \stackrel{\Delta}{\simeq} N_i \\
 & & 4.\ \forall N_{(i,-)}(\tau) \in A\, \neg\exists \tau', j.\, N_{(j,+)}(\tau') \in A' \text{ and} \\
 & & \quad \forall {N'}_{(j,+)}(\tau') \in A'\, \neg\exists \tau, i.\, {N'}_{(i,-)}(\tau) \in A \\
[\![N_{(i,\delta)}(\rho)]\!]_{\mathrm{a}}\theta & \stackrel{\Delta}{\simeq} & N_{(i,\delta)}(\rho) \\
[\![a\backslash(N, j)]\!]_{\mathrm{a}}\theta & \stackrel{\Delta}{\simeq} & N'_{(i,\delta)}(\tau \circ \{\theta(N', j, N) \mapsto \epsilon_\lambda\}) \text{ where } N'_{(i,\delta)}(\tau) \stackrel{\Delta}{\simeq} [\![a]\!]_{\mathrm{a}}\theta \\
[\![a]\!]_{\mathrm{c}}\theta & \stackrel{\Delta}{\simeq} & \{[\![a]\!]_{\mathrm{a}}\} \\
[\![a\,{}^i\!-\!{}^j\,a']\!]_{\mathrm{c}}\theta & \stackrel{\Delta}{\simeq} & \{N_{(h,\delta)}(\tau), N'_{(h',\delta')}(\tau')\} \text{ where} \\
 & & \tau \stackrel{\Delta}{\simeq} \tau'' \circ \{\theta(N, i, N') \mapsto k\} \\
 & & \tau' \stackrel{\Delta}{\simeq} \tau''' \circ \{\theta(N', j, N) \mapsto k\} \\
 & & N_{(h,\delta)}(\tau'') \stackrel{\Delta}{\simeq} [\![a]\!]_{\mathrm{a}}\theta \text{ and } N'_{(h',\delta')}(\tau''') \stackrel{\Delta}{\simeq} [\![a']\!]_{\mathrm{a}}\theta \\
 & & k \stackrel{\Delta}{\simeq} \mathrm{bind}(N, h, i, N', h', j) \\
[\![\{a\,{}^i\!-\!{}^j\,a'\}]\!]_{\mathrm{c}}\theta & \stackrel{\Delta}{\simeq} & \{N_{(i,\delta)}(\circ T) \mid T = \{\tau \mid N_{(i,\delta)}(\tau) \in S\} \wedge T \neq \emptyset\} \\
 & & \text{where } S \stackrel{\Delta}{\simeq} \bigcup\{[\![a\,{}^i\!-\!{}^j\,a']\!]_{\mathrm{c}}\theta\}
\end{array}
$$

Table 3
The definition of the denotational function from $\mathcal{AL}$ programs to Kappa programs.

The $\triangleleft$ operator is then applied to the resulting sets; the sets are sorted to obtain lists of agents; and finally the κ function is applied (it is assumed extended to lists in the natural way). In addition to implicit conditions for well-definedness, determined by whether the functions applied are defined, rules have four explicit conditions. The first condition states that any agents which are not marked for addition or deletion must be present on both the reactant and product sides, and with the same modification sites; the second states that only agents on the product side (reactant side) can be marked for addition (deletion); the third states that the same instance of an agent

cannot occur in different elements of a sum; and the fourth condition states that an agent cannot be both deleted and added by a rule. The latter avoids the possibility that agents which are marked as added or deleted are included in the "longest common prefix", which would result in an undesired semantics.

The denotational functions for agents and complexes use the given site renaming function for extending modification functions with appropriate binding sites. The case of a single binding gives rise to a set consisting of the two agents with their modifications composed with a new binding. Finally, in the case of a complex, i.e. a set of bindings, the denotational function is applied to each binding; the union of the result is obtained to get a set of agents; and the composition function is used to combine all the modification functions for the same agent.

5 Discussion

It is clear that some Kappa rules, such as the one shown in the introduction, can be simplified significantly using $\mathcal{AL}$. For other rules, in particular those with binding exclusion, the $\mathcal{AL}$ representation can be more complicated. However, from our (limited) experience with the EGFR case study, we observe that these more advanced language constructs are not needed for the majority of rules. In quantifiable terms, the rules in the $\mathcal{AL}$ EGFR model are on average 20% shorter than the rules in the original Kappa model.

One may argue that the explicit use of binding sites in Kappa closely matches available experimental data, and that $\mathcal{AL}$ is therefore not well suited for writing models based directly on experimental data. In these cases it could be of interest to translate Kappa models to $\mathcal{AL}$ models for improved readability. Such a translation should be straightforward to define. This could form the basis of a model development tool supporting two views, in Kappa and in $\mathcal{AL}$, for editing the same model.

This paper gives a *syntactic* abstraction of binding. A natural question arises of whether a *semantic* abstraction is possible: is there a mathematical structure for representing complexes that is simpler than a graph with sites, while still allowing for a "don't write don't care" policy for handling combinatorial explosion? At one extreme, bindings could be discarded altogether, giving rise to multisets. A dimerisation rule such as `A + B -> A-B` could then be interpreted as "`A` in any context (i.e. as an element of any multiset) can bind to `B` in any context", see Figure 2 a). The difficulty arises for dissociation, where it is unclear how to split the multiset containing `A` and `B`. It therefore seems that bindings must be represented at some level in the semantic representation of complexes. Taking this idea further, one could attempt to maintain bindings between sub-components of a complex, see Figure 2 b). But then a problem arises of how to dissociate sub-components when applying e.g. the

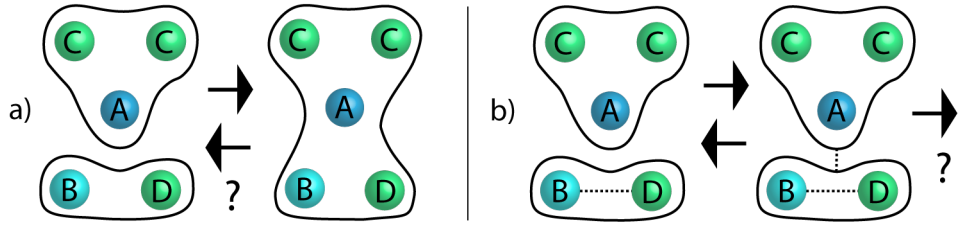

Figure 2. Alternative representations of complexes. a) An application of a rule `A + B -> A-B` under a representation with no bindings, illustrating the problem of dividing the context for dissociation. b) An application of the same rule under a representation with partial bindings (first two arrows), illustrating the problem of applying an additional rule `B-D -> B + D` (third arrow) to a sub-component of a complex.

rule `B-D -> B + D`. Although very preliminary, this reasoning suggests that the binding-level semantic representation of complexes employed by Kappa may be the most natural one when seeking to handle combinatorial explosion in the number of protein complexes.

References

[1] Vincent Danos, Jérôme Feret, Walter Fontana, Russ Harmer, and Jean Krivine. Rule-based modelling and model perturbation. *Trans. on Comput. Syst. Biol.*, 5750(11):116–137, 2009.

[2] Vincent Danos, Jérôme Feret, Walter Fontana, Russell Harmer, and Jean Krivine. Rule-based modelling of cellular signalling. In *CONCUR*, volume 4703 of *LNCS*, pages 17–41. Springer, 2007. Tutorial paper.

[3] Vincent Danos, Jérôme Feret, Walter Fontana, and Jean Krivine. Scalable simulation of cellular signaling networks. In *APLAS*, volume 4807 of *Lecture Notes in Computer Science*, pages 139–157. Springer, 2007.

[4] J. R. Faeder, Michael L. Blinov, and William S. Hlavacek. Graphical rule-based representation of signal-transduction networks. In L. M. Liebrock, editor, *Proc. 2005 ACM Symp. Appl. Computing*, pages 133–140. ACM Press, 2005.

[5] Jérôme Feret, Vincent Danos, Jean Krivine, Russ Harmer, and Walter Fontana. Internal coarse-graining of molecular systems. *Proceedings of the National Academy of Sciences*, 106(16):6453–6458, April 2009.

[6] A. Mallavarapu, M. Thomson, B. Ullian, and J. Gunawardena. Programming with models: modularity and abstraction provide powerful capabilities for systems biology. *J. R. Soc. Interface*, 2008.

[7] Michael Pedersen. *Modular languages for systems and synthetic biology.* PhD thesis, School of Informatics, University of Edinburgh, 2010.

[8] Michael Pedersen and Gordon Plotkin. A Language for Biochemical Systems: Design and Formal Specification. In *Trans. on Comput. Syst. Biol.*, volume 5945, pages 77–145. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[9] Michael W. Sneddon, James R. Faeder, and Thierry Emonet. Efficient modeling, simulation and coarse-graining of biological complexity with NFsim. *Nature Methods*, 8(2):177–183, February 2011.

[10] David Weininger. SMILES, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 28(1):31–36, 1988.