

When rule-based models need to count

Pierre Boutillier, Ioana Cristescu

▶ **To cite this version:**

Pierre Boutillier, Ioana Cristescu. When rule-based models need to count. [Research Report] Harvard Medical School. 2017. hal-01572177

HAL Id: hal-01572177
https://hal.science/hal-01572177

Submitted on 4 Aug 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

When rule-based models need to count

Pierre Boutillier[1] Ioana Cristescu[2]

Fontana Lab
Harvard Medical School
Boston, USA

Abstract

Rule-based modelers dislike direct enumeration of cases when more efficient means of enumeration are available. We present an extension of the Kappa language which attaches to agents a notion of *level*. We detail two encodings that are more concise than the former practice.

Rule-based languages are a well-established framework for modeling protein-protein interactions.

Kappa [2,1] is a rule-based language relying on *site-graphs*. The nodes of site-graphs are called *agents*. Agents interact by binding/unbinding through *sites*. Sites are binding resources, each site is part of at most one edge.

A model in Kappa consists of a set of graph rewrite rules with rates. A rule describes a potential interaction given a context. Rates represent probability to fire.

In a biological context, it is often the case that a notion of internal state (such as `active`, `methilated`, ...) is required in order to describe possible interactions. In Kappa, sites are equipped with internal states, facilitating the modeling efforts of the user. However, as shown in the following sections a more systematic encoding of internal states is possible.

Another common practice is to attach a *level* to agents and make an interaction sensitive on the level of its participating agents.

We propose here a language extension to store, test and change levels explicitly. Moreover, we present an encoding of levels that induce a linear (in number of levels) blow up of the number of rules. This is in contrast to previous encodings, which induce an exponential blow up in the number of rules.

[1] Email: Pierre_Boutillier@hms.harvard.edu
[2] Email: Ioana_Cristescu@hms.harvard.edu

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.com/locate/entcs`

1 When enumeration is necessary

The following motivating example [4] demonstrates a typical problem in which levels are necessary:

> KaiC proteins have 6 independent phosphorylation sites. (De)phosphorylation of every site is independent. The more sites are phosphorylated, the bigger the probability that KaiC binds KaiA is.

A typical way to deal with this example consist in explicitly encoding rules for the internal states of the sites of interest. However, doing so induces an exponential blow-up in the number of rules (in the number of levels).

The BNGL [3] language introduces a notion of *indistinguishable* sites. i.e. one can define an agent with n sites that all have the same name. Consequently, a single rule to specify that k sites (out of n) are phosphorylated is enough. Moreover, the number of species is also reduced. Still, there are exponentially many ways to go from one level to another and enumeration is necessary to faithfully respect the dynamics of the system.

We now show that adding a syntactic layer to Kappa that offers support for counting will avoid the explosion in the number of rules.

2 Encoding counters

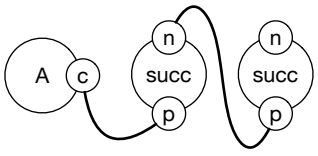
In Kappa, agents are typed and their *signature* is given. In the extension we are presenting, counters are part of an agent's signature and the upper bound of the counter has to be specified.

Extended rules can test counter values ("equals" and "bigger than"). They can also modify their value (increase, decrease or assign a new value). Lastly, levels can be used in the rates of the rules.

We now present two possible ways of encoding counters.

2.1 Unary numbers

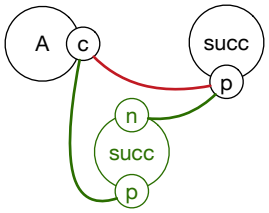
We define a new agent type `succ` with 2 sites `p` and `n`. A chains of $k+1$ `succ` agents denotes a counter with value k. Agents equipped with counters are encoded as agents with an extra site for each counter, which is used to bind to a chain of `succ` agents. For example, in the figure on the right, an agent `A` has a counter `c` set to 1[3].



Testing whether a counter is equal to k consists in checking whether there is a chain of $k+1$ `succ` agents. A greater than k test checks whether there is a chain $k+1$ `succ` agents, where the site `n` of the last `succ` agent does not matter.

[3] Representing level 0 by a chain of length 1 means that we do not have to make a special case for it in the rules!

A counter is incremented by adding `succ` agents between the agent and its `succ` chain. The rule depicted on the right increments `c` of `A` by one. Removing the beginning of the `succ` chain decreases the level.
It is important to stress that counter modifications are independent of its value as the encoding only manipulates the beginning of a `succ` chain.



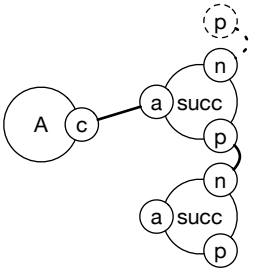
In our encoding, a rule whose rate depends on levels is expanded into several rules, one for each level. This operation imposes the user-defined upper bound on the levels.

Creation of agents with levels also creates the necessary chain of `succ` to represent the levels.

Deletion is more problematic: the chain of `succ` is disconnected from the deleted agent, but not deleted. A possible solution is to collect free chains of `succ` by using a rule that says that `succ` agents with their sites `p` free are deleted "at infinite speed".

2.2 Ruler

A second encoding allows tests "smaller than" in addition of "equal" and "greater than", but is also more verbose and increases the size of states.
As before, counters are encoded as a chain of `succ` agents. In this encoding however, a `succ` agent has 3 sites: `p` and `n` to form chains but also `a`, which is where the other agents bind. Every agent with a counter bounded by n has attached a chain of (always) n `succ`. The value of the counter is given by which `succ` agent it is bound to on site `a`. For example, in the figure on the right, an agent `A` has a counter `c` set to 1.



Since site `p` and `n` are distinct, a direction can be attached to the chain of `succ`. The level is 0 if the agent is bound to the "bottom" of the chain, and it is n if it is bound to the "top".

Incrementing/decrementing the levels are implemented by sliding on the `succ` chain. Testing whether a level is equal or greater than k consists of inspecting the chain of `succ` "below" the connection to the agent. Whereas inspecting the length of the chain "above" the connection to the agent enables to test if the level is smaller than a value k.

3 Conclusions

While fairly trivial, encoding counters in the Kappa simulator has greatly simplified some of the models written by the Kappa team. However, it comes with two drawbacks. First, there is a computational cost (mainly in term of memory management) to synthetize and degrade `succ` agents. Secondly, the simulator works with a plain Kappa model. There is an implementation cost to go back from the plain Kappa model to the user-written model, which is necessary when giving feedback to the

user (for example, dumping the current state of the system). As a result, we plan to implement a native treatment of the counters in the Kappa simulator.

Another direction for future work is the static detection of overflow for counters. Currently, the maximal level of the counter has to be declared by the user. Nevertheless, this means the rules have to be written such that they can never increase a level beyond the defined boundary. The level of the counter can be checked by static analysis using interval computation, but for the meantime we have simply added watchdogs that dynamically raise the alarm if the site of a level becomes free, or if a chain of `succ` that is too long appears.

References

[1] Pierre Boutillier, Jérôme Feret, and Jean Krivine. Kappa and KaSim development page. http://kappalanguage.org.

[2] Vincent Danos, Jérôme Feret, Walter Fontana, Russell Harmer, and Jean Krivine. Rule-based modelling, symmetries, refinements. In *Formal Methods in Systems Biology, First International Workshop, FMSB 2008, Cambridge, UK, June 4-5, 2008. Proceedings*, pages 103–122, 2008.

[3] James R. Faeder, Michael L. Blinov, and William S. Hlavacek. *Rule-Based Modeling of Biochemical Systems with BioNetGen*, pages 113–167. Humana Press, Totowa, NJ, 2009.

[4] Susan S Golden. Integrating the circadian oscillator into the life of the cyanobacterial cell. In *Cold Spring Harbor symposia on quantitative biology*, volume 72, pages 331–338. Cold Spring Harbor Laboratory Press, 2007.