Counters in Kappa: Semantics, Simulation, and Static Analysis

Pierre Boutillier[1], Ioana Cristescu[2], and Jérôme Feret[3]

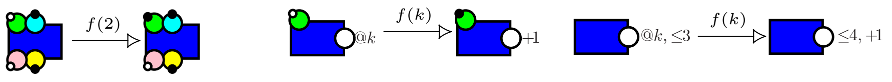
[1] Harvard Medical School, Boston, USA
`Pierre_Boutillier@hms.harvard.edu`
[2] Inria Rennes - Bretagne Atlantique, Rennes, France
`ioana-domnina.cristescu@inria.fr`
[3] DI-ENS (INRIA/ÉNS/CNRS/PSL*), Paris, France
`feret@ens.fr`

Abstract. Site-graph rewriting languages, such as Kappa or BNGL, offer parsimonious ways to describe highly combinatorial systems of mechanistic interactions among proteins. These systems may be then simulated efficiently. Yet, the modeling mechanisms that involve counting (a number of phosphorylated sites for instance) require an exponential number of rules in Kappa. In BNGL, updating the set of the potential applications of rules in the current state of the system comes down to the sub-graph isomorphism problem (which is NP-complete).
In this paper, we extend Kappa to deal both parsimoniously and efficiently with counters. We propose a single push-out semantics for Kappa with counters. We show how to compile Kappa with counters into Kappa without counters (without requiring an exponential number of rules). We design a static analysis, based on affine relationships, to identify the meaning of counters and bound their ranges accordingly.

1 Introduction

Site-graph rewriting is a paradigm for modeling mechanistic interactions among proteins. In Kappa [18] and BNGL [3,40], rewriting rules describe how instances of proteins may bind and unbind, and how each protein may activate the interaction sites of each others, by changing their properties. Sophisticated signaling cascades may be described. The long term behavior of such models usually emerges from competition against shared-resources, proteins with multiple-phosphorylation sites, scaffolds, separation of scales, and non-linear feedback loops.

It is often desirable to add more structure to states in order to describe generic mechanisms more compactly. In this paper, we consider extending Kappa with counters with numerical values. As opposed to the properties of classical Kappa sites, which offer no structure, counters allow for expressive preconditions (such as the value of a counter is less than 2), but also for generic update functions (such as incrementing or decrementing the current value of a counter by a given value independently of its current value). Without counters, such



(a) With phosphorylation sites only.

(b) With a counter and phosphorylation sites

(c) With a counter only

Fig. 1. Three representations for the phosphorylation of a site. We assume that the rate of phosphorylation of a site in a protein in which exactly k sites are already phosphorylated, is equal to the value $f(k)$. The function f is left as a parameter of the model. In 1(a), we do not use counters. In order to get the number of sites that are already phosphorylated, we have to document the state of all the sites of the protein. In this rule, there are exactly 2 sites already phosphorylated, thus the rate of the rule is equal to $f(2)$. In 1(b), we use a counter to encode the number of sites already phosphorylated. The variable k, that is introduced by the notation $@k$, contains the number of sites that are phosphorylated before the application of the rule. Thus, the rate of the rule is equal to $f(k)$. In the right hand side, the notation $+1$ indicates that the counter is incremented at each application of the rule. The rule in 1(b) summarizes exactly 8 rules of the kind of the one in 1(a) (it defines the phosphorylation of the site a regardless of the states of the three other phosphorylation sites). In 1(c), we abstract away the sites and keep only the counter. The notation $@k$ binds the variable k to the value of the counter. The left hand side also indicates that the rule may be applied only if the value of the counter is less than or equal to 3 (so that at least one site is not already phosphorylated). The right hand side specifies that the value of the counter is incremented at each application of the rule and that after the application of a rule, the value of the counter is always less than or equal to 4. The rule in 1(c) stands for 32 rules of the kind of the one in 1(a) (it depends neither on which site is phosphorylated, nor on the state of the three other sites).

update functions would require one rule per potential value of the counter. This raises efficiency issues for the simulation and also blurs any potential reasoning on the causality of the system.

However adding counters cannot be done without consequences. The efficiency of Kappa simulations mainly relies on two ingredients. Firstly, Kappa graphs are rigid [16,39]: an embedding from a connected site-graph into a site-graph, when it exists, is fully determined by the image of one node. Thanks to rigidity, searching for the occurrences of a sub-graph into another graph (up-to isomorphism) may be done without backtracking (once a first node has been placed), and embeddings can be described in memory very concisely. Secondly, the representation of the set of potential applications of rules relies on a categorical construction [6] that optimizes sharing among patterns. Yet this construction cannot cope with the more expressive patterns that involve counters. In order to efficiently simulate models with counters, we need an efficient encoding that preserves rigidity and that use classical site-graph patterns.

Let us consider a case study so as to illustrate the need for counters in Kappa. This example is inspired from the behavior of the protein *KaiC* that is involved in the synchronization of the proteins in the circadian clock. We consider one kind of protein with n identified sites that can get phosphorylated. Indeed, n is equal to 6 in the protein $KaiC$. We take n equal to 4 to make graphical representation lighter. We will make n diverge towards the infinity so as to empirically estimate the combinatorial complexity of several encoding schemes.

The rate of phosphorylation/dephosphorylation of each site, depends on the number of sites that are already phosphorylated. In Fig. 1(a), we provide the example of a rule that phosphorylates the site a of the protein, assuming that the sites b and c are already phosphorylated and that the site d is not. Proteins are depicted as rectangles. Sites are depicted clockwise from the site a to the site d starting at the top left corner of the protein. Phosphorylation states are depicted with a black mark when the site is phosphorylated, and with a white mark otherwise. To fully encode this model in Kappa, we would require $n \cdot 2^n$ rules. Indeed, we need to decide whether this is a phosphorylation or a dephosphorylation (2 possibilities), then on which site to apply the transformation (n possibilities), then what the state of the other sites is (2^{n-1} possibilities). This combinatorial complexity may be reduced by the means of counters. We consider a fresh site (this site is depicted on the right of the protein) and we assume that this site takes numerical values. Writing each rule carefully, we can enforce that the value of this site is always equal to the number of the sites that are phosphorylated in the protein instance. Thanks to this invariant, describing our model requires $2{\cdot}n$ rules according to whether we describe a phosphorylation or a dephosphorylation (2 possibilities) and to which site the transformation is applied (n possibilities). An example of rule for the phosphorylation of the site a is given in Fig. 1(b). The notation $@k$ assigns the value of the counter before the application of the rule to the variable k. Then the rate of the rule may depend on the value of k. This way, we can make the rate of phosphorylation depend on the number of sites already phosphorylated in the protein. Since there are only n sites that may be phosphorylated, it is straightforward to see that the counter may range only between the values 0 and n.

If only the number of phosphorylated sites matters, we can go even further: we need just one counter and two rules, one for phosphorylating a new site (e. g. see Fig. 1(c)) and one for dephosphorylating it. The value of the counter is no longer related explicitly to a number of phosphorylated sites, thus we need another way to specify that the value of the counter is bounded. We do this, by specifying in the precondition of the rule that the phosphorylation rule may be applied only if the value of the counter is less or equal to $n-1$, which entails that the value of the counter may range only between the values 0 and n.

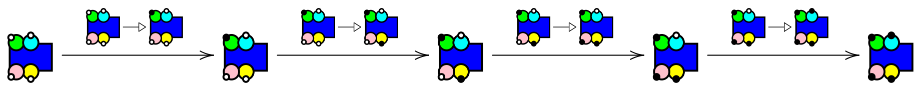
Not only parsimonious description of the mechanistic interactions in a model eases the process of writing a model, enhances readability and leads to more efficient simulation, but also it may provide better grain of observation of the system behavior. In Fig. 2, we illustrate this by looking at three causal traces that denote the same execution, but for three different encodings. Intuitively, causal traces [15,14] are inspired by event structures [43]. They describe sets of traces seen up to permutation of concurrent computation steps. The level of representation for the potential configurations of each protein impacts the way causality is defined, because what is tested in rules depends on the representation level. In our case study, the phosphorylation of each site is intuitively causally independent: one site may be phosphorylated whatever the state of the other sites is. Without counters, the only way to specify that the rate of phosphorylation depends on the number of the sites that are already phosphorylated, is to detail the state of every site of the protein in the precondition of the rule. This induces spurious causal relations (e. g. see Fig. 2(a)). Utilizing counters relaxes this constraint. However it is important to equip counters with arithmetic. Without arithmetic, a rule may only set the value of a counter to a constant value. Thus for implementing counter increment, rules have to enumerate the potential values of the counter before their applications, and set the value of this counter accordingly. This induces again spurious causal relations (e. g. see Fig. 2(b)). With arithmetic, incrementing counters becomes a generic operation that may be applied independently of the current value of the counter. As a result the phosphorylation of the four sites can be seen as causally independent (e. g. see Fig. 2(c)). This faithfully represents the fact that the phosphorylation of the four sites may happen in arbitrary order.

Contribution. Now we describe the main contributions of this paper.

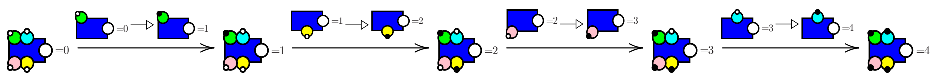
In Sect. 2, we formalize a single push-out (SPO) semantics for Kappa with counters. Having a categorical framework dealing with counters, as opposed to implementing counters as syntactic sugar, is important. Firstly, this semantics will serve as a reference for the formal specification of the behavior of counters. Secondly, the categorical setting of Kappa provides efficient ways to define causality [15,14], symmetries [25], and some sound symbolic reasonings on the behavior of the number of occurrences of patterns [26,1] that are used in model reduction. Including counters in the categorical semantics of Kappa allows for extending the definition of these concepts to Kappa with counters for free.

Yet different encodings of counters may be necessary to extend other tools for Kappa. In Sect. 3, we propose a couple of translations from Kappa with counters into Kappa without counters. The goal is to simulate models with counters efficiently without modifying the implementation of the Kappa simulator, KaSim [17]. The first encoding requires counters to be bounded from below and it supports only two kinds of preconditions over counters: a rule may require the value of a counter to be equal to a given value, or to be greater than a given value. Requiring the value of a counter to be less than a given value is not supported. The second encoding supports equality and inequality (in both directions) tests. But it requires the value of each counter to be bounded also from above.

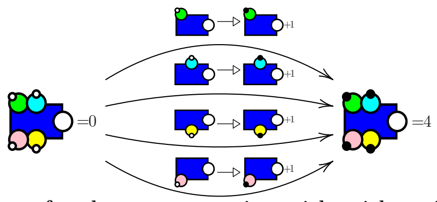
Static analysis is needed not only to prove these requirements, but also to retrieve the meaning of counters. In Sect. 4, we introduce a generic abstract interpretation framework [9] to infer the properties of reachable states of a model. This framework is parametric with respect to a class of properties. In Sect. 5, we instantiate this framework with a relational numerical analysis aiming at relating the value of each counter to its interpretation with respect to the state of the other sites. This is used to detect and prove bounds on the range of counters.



(a) Causal trace for the representation without counters.



(b) Causal trace for the representation with flat counters.



(c) Causal trace for the representation with arithmetical counters.

Fig. 2. Three causal traces. Each causal trace is made of a set of partially ordered computation steps. Roughly speaking, a computation step precedes another one, if the former is necessary to perform the later. Each computation step is denoted as an arrow labeled with the rule that implements it. In 2(a), counters are not used. Every rule tests the full configuration of the protein. At this level of representation, the k-th phosphorylation causally precedes the $k+1$-th one, whatever the order in which the sites have been phosphorylated. In 2(b), an additional site is used to record the number of phosphorylated sites in its internal state. With this encoding, the number of phosphorylated sites cannot be incremented without testing explicitly the internal state of the additional site. As a consequence, here again, at this level of representation, each phosphorylation causally depends on the previous one. In 2(c), we use the expressiveness of arithmetic. We use generic rules to increment the counter regardless of its current value. Hence, at this level of representation, the phosphorylation of the four sites become independent, which flatten the causal trace.

Related works Many modeling languages support arbitrary data-types. In Spatial-Kappa [41], counters encode the discrete position of agents. More generally, in Chromar [29] and in colored Petri nets [30,35], agents may be tagged with values in arbitrary auxiliary programming languages. In ML-Rules [28], agents with attributes continuously diffuse within compartments and collide to interact.

We have different motivations. Our goal is to enrich the state of proteins with some redundant information, so as to reduce the number of rules that are necessary to describe their mechanistic interactions. Also we want to avoid too expressive data-types, which could not be integrated within simulation, causality analysis, and static analysis tools, without altering their performance. For instance, analysis of colored Petri nets usually relies on unfolding them into classical ones. Unfolding rule sets into classical ones does not scale because the number of rules would become intractable. Thus we need tools which deal directly with counters.

An encoding of two-counter machines has been proposed to show that most problems in Kappa are undecidable [19,34]. We represent counters the same way in our first encoding, but we provide atomic implementation for more primitives.

The number of isomorphic classes of connected components that may occur in Kappa models during simulation is usually huge (if not infinite), which prevents from using agent-centric approaches [4]. For instance, one of the first non-toy model written in Kappa was involving more than 10^{19} kinds of biomolecular complexes [26,16]. Kappa follows a rule-centric approach which allows for the description and the execution of models independently from the number of potential complexes. Also, Kappa disallows to describe diffusion of molecules. Instead the state of the system is assumed to satisfy the well-mixed assumption. This provides efficient ways to represent and update the distribution of potential computation steps, along a simulation [17,6].

Equivalent sites [3] or hyperlinks [31] offer promising solutions to extend the decision procedures to extract minimal causal traces in the case of counters, but the rigidity of graphs is lost. Our encodings rely neither on the use of equivalent sites, nor on expanding the rules into more refined and more numerous ones. Hence our encodings preserve the efficiency of the simulation.

Our analysis is based on the use of affine relationships [32]. It relates counter values to the state of the other variables. Such relationships look like the ones that help understanding and proving the correctness of semaphores [20,21]. We use the decision procedure that is described in [23,24] to deduce bounds on the values of counters from the affine relationships. The cost of each atomic computation is cubic with respect to the number of variables. Abstract multi-sets [27,38] may succeed in expressing the properties of interest, but they require a parameter setting a bound on the values that can abstract precisely. In practice, their time-cost is exponential as soon as this bound is not chosen big enough. Our abstraction has an infinite height. It uses widening [11] and reduction [12] to discover the bounds of interest automatically. Octagons [36,37] have a cubic complexity, but they cannot express the properties involving more than two variables which are required in our context. Polyhedra [13] express all the properties needed for an exponential time-cost in practice.

2 Kappa

In this section, we enrich the syntax and the operational semantics of Kappa so as to cope with counters. We focus on the single push-out (SPO) semantics.

2.1 Signature

Firstly we define the signature of a model.

Definition 1 (signature). *The signature of a model is defined as a tuple* $\Sigma = (\Sigma_{ag}, \Sigma_{site}, \Sigma_{int}, \Sigma_{ag\text{-}st}^{int}, \Sigma_{ag\text{-}st}^{lnk}, \Sigma_{ag\text{-}st}^{\$}, Prop_{\$}, Update_{\$})$ *where:*

1. *Σ_{ag} is a finite set of agent types,*
2. *Σ_{site} is a finite set of site identifiers,*
3. *Σ_{int} is a finite set of internal state identifiers,*
4. *$\Sigma^{lnk}_{ag\text{-}st}$, $\Sigma^{int}_{ag\text{-}st}$, and $\Sigma^{\$}_{ag\text{-}st}$ are three site maps (from Σ_{ag} into $\wp(\Sigma_{site})$)*
5. *$Prop_{\$}$ is a potentially infinite set of non-empty subsets of $\mathbb{Z}$,*
6. *$Update_{\$}$ is a potentially infinite set of functions from $\mathbb{Z}$ to $\mathbb{Z}$ containing the identity function.*

For every $G \in Prop_{\$}$, we assume that for every function $f \in Update_{\$}$, the set $\{f(k) \mid k \in G\}$ belongs to the set $Prop_{\$}$, and that for every element $k \in G$, the set $\{k\}$ belongs to the set $Prop_{\$}$ as well.

Agent types in Σ_{ag} denote the agents of interest, the different kinds of proteins for instance. A site identifier in Σ_{site} represents an identified locus for a capability of interaction. Each agent type $A \in \Sigma_{ag}$ is associated with a set of sites $\Sigma^{int}_{ag\text{-}st}(A)$ with an internal state (i.e. a property), a set of sites $\Sigma^{lnk}_{ag\text{-}st}(A)$ which may be linked, and a set of sites $\Sigma^{\$}_{ag\text{-}st}(A)$ with a counter. We assume without any loss of generality that the three sets $\Sigma^{lnk}_{ag\text{-}st}(A)$, $\Sigma^{int}_{ag\text{-}st}(A)$, and $\Sigma^{\$}_{ag\text{-}st}(A)$ are disjoint pairwise. The set *$Prop_{\$}$* contains the set of valid conditions that may be checked on the value of counters, whereas the set *$Update_{\$}$* contains all the possible update functions for the value of counters. We assume that every singleton that is included in a valid condition is a valid condition as well. In this way, a valid condition may be refined to a fully specified value. Additionally, the image of a valid condition is required to be valid, so that the post-condition obtained by applying an update function to a valid precondition, is valid as well.

Example 1 (running example). We define the signature for our case study as the tuple $(\Sigma_{ag}, \Sigma_{site}, \Sigma_{int}, \Sigma^{int}_{ag\text{-}st}, \Sigma^{lnk}_{ag\text{-}st}, \Sigma^{\$}_{ag\text{-}st}, Prop_{\$}, Update_{\$})$ where:

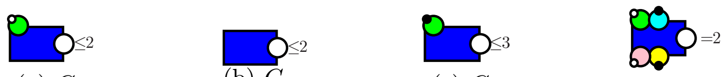
1. *$\Sigma_{ag} := \{P\}$;*
2. *$\Sigma_{site} := \{a, b, c, d, x\}$;*
3. *$\Sigma_{int} := \{\circ, \bullet\}$;*
4. *$\Sigma^{int}_{ag\text{-}st} := [P \mapsto \{a, b, c, d\}]$;*
5. *$\Sigma^{lnk}_{ag\text{-}st} := [P \mapsto \emptyset]$;*
6. *$\Sigma^{\$}_{ag\text{-}st} := [P \mapsto \{x\}]$;*
7. *$Prop_{\$}$ is the set of all the convex parts of $\mathbb{Z}$;*
8. *$Update_{\$}$ contains the function mapping each integer $n \in \mathbb{Z}$ to its successor, and the function mapping each integer $n \in \mathbb{Z}$ to its predecessor.*

The agent type P denotes the only kind of proteins. It has four sites a, b, c, d carrying an internal state and one site x carrying a counter. □

Until the rest of the paper, we assume given a signature Σ.

2.2 Site-graphs

Site-graphs describe both patterns and chemical mixtures. Their nodes are typed agents with some sites which may carry internal and binding states, and counters.



(a) G_1. (b) G_2. (c) G_3. (d) G_4.

Fig. 3. Four site-graphs G_1, G_2, G_3, and G_4.

Definition 2 (site-graph). *A site-graph is a tuple $G = (\mathcal{A}, type, \mathcal{S}, \mathcal{L}, p\kappa, c\kappa)$ where:*

1. *$\mathcal{A}$ is a finite set of agents,*
2. *$type : \mathcal{A} \to \Sigma_{ag}$ is a function mapping each agent to its type,*
3. *$\mathcal{S}$ is a set of sites satisfying the following property:*
$$\mathcal{S} \subseteq \{(n,i) \mid n \in \mathcal{A}, i \in \Sigma_{ag\text{-}st}(type(n))\},$$
4. *$\mathcal{L}$ maps the set:*
$$\{(n,i) \in \mathcal{S} \mid i \in \Sigma^{lnk}_{ag\text{-}st}(type(n))\}$$
to the set:
$$\{(n,i) \in \mathcal{S} \mid i \in \Sigma^{lnk}_{ag\text{-}st}(type(n))\} \cup \{\dashv, -\},$$
such that:
   (a) for any site $(n,i) \in \mathcal{S}$, we have $\mathcal{L}(n,i) \neq (n,i)$;
   (b) for any two sites $(n,i), (n',i') \in \mathcal{S}$, we have $(n',i') = \mathcal{L}(n,i)$ if and only if $(n,i) = \mathcal{L}(n',i')$;
5. *$p\kappa$ maps the set $\{(n,i) \in \mathcal{S} \mid i \in \Sigma^{int}_{ag\text{-}st}(type(n))\}$ to the set Σ_{int};*
6. *$c\kappa$ maps the set $\{(n,i) \in \mathcal{S} \mid i \in \Sigma^{\$}_{ag\text{-}st}(type(n))\}$ to the set $Prop_{\$}$.*

For a site-graph G, we write as $\mathcal{A}_G$ its set of agents, $type_G$ its typing function, $\mathcal{S}_G$ its set of sites, and $\mathcal{L}_G$ its set of links. Given a site-graph G, we write as $\mathcal{S}^{lnk}_G$ (resp. $\mathcal{S}^{int}_G$, resp. $\mathcal{S}^{\$}_G$) its set of binding sites (resp. property sites, resp. counters) that is to say the set of the sites (n,i) such that $i \in \Sigma^{lnk}_{ag\text{-}st}(type_G(n))$ (resp. $i \in \Sigma^{int}_{ag\text{-}st}(type_G(n))$, resp. $i \in \Sigma^{\$}_{ag\text{-}st}(type_G(n))$).

Let us consider a binding site $(n,i) \in \mathcal{S}^{lnk}_G$. Whenever $\mathcal{L}_G(n,i) = \dashv$, the site (n,i) is free. Various levels of information may be given about the sites that are bound. Whenever $\mathcal{L}_G(n,i) = -$, the site (n,i) is bound to an unspecified site. Whenever $\mathcal{L}_G(n,i) = (n',i')$ (and hence $\mathcal{L}_G(n',i') = (n,i)$), the sites (n,i) and (n',i') are bound together.

A *chemical mixture* is a site-graph in which the state of each site is fully specified. Formally, a site-graph G is a chemical mixture, if and only if, the three following properties:

1. the set $\mathcal{S}_G$ is equal to the set $\{(n,i) \mid n \in \mathcal{A}_G,\ i \in \Sigma_{ag\text{-}st}(type_G(n))\}$;
2. every binding site is free or bound to another binding site (i. e. for every $(n,i) \in \mathcal{S}_G \cap \Sigma^{lnk}_{ag\text{-}st}(type_G(n))$, $\mathcal{L}_G(n,i) \neq -$);
3. every counter has a single value (i. e. for every $(n,i) \in \Sigma^{\$}_{ag\text{-}st}$, $c\kappa_G(n,i)$ is a singleton);

are satisfied.

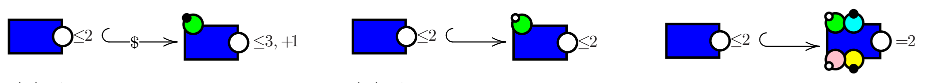
Example 2 (running example). In Fig. 3, we give a graphical representation of the four site-graphs, G_1, G_2, G_3, and G_4 that are defined as follows:

1. (a) $\mathcal{A}_{G_1} = \{1\}$,
   (b) $type_{G_1} = [1 \mapsto P]$,
   (c) $\mathcal{S}_{G_1} = \{(1,a),(1,x)\}$,
   (d) $\mathcal{L}_{G_1} = \emptyset$,
   (e) $p\kappa_{G_1} = [(1,a) \mapsto \circ]$,
   (f) $c\kappa_{G_1} = [(1,x) \mapsto \{k \in \mathbb{Z} \mid k \leq 2\}]$;
2. (a) $\mathcal{A}_{G_2} = \{1\}$,
   (b) $type_{G_2} = [1 \mapsto P]$,
   (c) $\mathcal{S}_{G_2} = \{(1,x)\}$,
   (d) $\mathcal{L}_{G_2} = \emptyset$,
   (e) $p\kappa_{G_2} = []$,
   (f) $c\kappa_{G_2} = [(1,x) \mapsto \{k \in \mathbb{Z} \mid k \leq 2\}]$;
3. (a) $\mathcal{A}_{G_3} = \{1\}$,
   (b) $type_{G_3} = [1 \mapsto P]$,
   (c) $\mathcal{S}_{G_3} = \{(1,a),(1,x)\}$,
   (d) $\mathcal{L}_{G_3} = \emptyset$,
   (e) $p\kappa_{G_3} = [(1,a) \mapsto \bullet]$,
   (f) $c\kappa_{G_3} = [(1,x) \mapsto \{k \in \mathbb{Z} \mid k \leq 3\}]$;
4. (a) $\mathcal{A}_{G_4} = \{1\}$,
   (b) $type_{G_4} = [1 \mapsto P]$,
   (c) $\mathcal{S}_{G_4} = \{(1,a),(1,b),(1,c),(1,d),(1,x)\}$,
   (d) $\mathcal{L}_{G_4} = \emptyset$,
   (e) $p\kappa_{G_4} = [(1,a) \mapsto \circ, (1,b) \mapsto \bullet, (1,c) \mapsto \bullet, (1,d) \mapsto \circ]$,
   (f) $c\kappa_{G_4} = [(1,x) \mapsto \{2\}]$;

The white site on the side of proteins is always the site x. The other sites, starting from the top-left one denote the sites a, b, c, and d clockwise. □

2.3 Sliding embeddings

In classical Kappa, two site-graphs may be related by structure-preserving injections, which are called embeddings. Here, we extend their definition to cope with counters. There are two main issues: a rule may require the value of a given counter to belong to a non-singleton set; also updating counters may involve arithmetic computations. The smaller the set of the potential values for a counter is, the more information we have. Thus, embeddings may map the potential values of a given counter into a subset. In order to cope with update functions, we equip embeddings with some arithmetic functions which explain how to get from the value of the counter in the source of the embedding to its value in the target. This way, our embeddings not only define instances of site-graphs, but they also contain the information to compute the values of counters.



(a) A sliding embedding. (b) A pure embedding. (c) A pure embedding.

Fig. 4. Three sliding embeddings from the G_2 respectively into the site-graphs G_3, G_1, and G_4. Only the second and the third embeddings are pure.

Definition 3 (sliding embedding). *A* sliding embedding $h : G \overset{\$}{\hookrightarrow} H$ *from a site-graph G into a site-graph H is a pair $(h_e, h_\$)$ where h_e is a function of agents $h_e : \mathcal{A}_G \to \mathcal{A}_H$ and $h_\$$ is a function mapping the counters of the site-graph G to update functions $h_\$: \mathcal{S}^{\$}_G \to Update_\$$ such that for all agent identifiers m, n, $n' \in \mathcal{A}_G$ and for all site identifiers $i \in \Sigma_{ag\text{-}st}(type_G(n))$, $i' \in \Sigma_{ag\text{-}st}(type_G(n'))$, the following properties are satisfied:*

1. *if $m \neq n$, then $h_e(m) \neq h_e(n)$;*
2. *$type_G(n) = type_H(h_e(n))$;*
3. *if $(n,i) \in \mathcal{S}_G$, then $(h_e(n), i) \in \mathcal{S}_H$;*
4. *if $(n,i) \in \mathcal{S}^{lnk}_G$ and $\mathcal{L}_G(n,i) = (n', i')$, then $\mathcal{L}_H(h_e(n), i) = (h_e(n'), i')$;*
5. *if $(n,i) \in \mathcal{S}^{lnk}_G$ and $\mathcal{L}_G(n,i) = \dashv$, then $\mathcal{L}_H(h_e(n), i) = \dashv$;*
6. *if $(n,i) \in \mathcal{S}^{lnk}_G$ and $\mathcal{L}_G(n,i) = -$, then $\mathcal{L}_H(h_e(n), i) \in \{-\} \cup \mathcal{S}_H$;*
7. *if $(n,i) \in \mathcal{S}^{int}_G$ and $p\kappa_G(n,i) = \iota$, then $p\kappa_H(h_e(n), i) = \iota$;*
8. *if $(n,i) \in \mathcal{S}^{\$}_G$, then $c\kappa_H(h(n), i) \subseteq \{h_\$(k) \mid k \in c\kappa_G(n,i)\}$.*

Two sliding embeddings between site-graphs, from E to F, and from F to G respectively, compose to form a sliding embedding from E to G (functions compose pair-wise). A sliding embedding $(h_e, h_\$)$ such that $h_\$$ maps each counter to the identity function is called a *pure embedding.* A pure embedding from E to F is denoted as $E \hookrightarrow F$. Pure embeddings compose. Two site-graphs E and F are isomorphic if and only if there exist a pure embedding from E to F and a pure embedding from F to E. A pure embedding between two isomorphic site-graphs is called an isomorphism. When it exists, the unique pure embedding $(h_e, h_\$)$ from a site-graph E into the site-graph F such that $\mathcal{A}_E \subseteq \mathcal{A}_F$ and $h_e(n) = n$ for every agent $n \in \mathcal{A}_E$, is called the *inclusion* from E to F and is denoted as $i_{E,F}$ or as $E \overset{\subseteq}{\hookrightarrow} F$. In such a case, we say that the site-graph E is included in the site-graph F. The inclusion from a site-graph into itself always exists and is called an identity embedding.

Example 3 (running example). We show in Fig. 4 three sliding embeddings from the site-graph G_2 respectively into the site-graphs G_3, G_1, and G_4. The first of these three sliding embeddings is assumed to increment the value of the counter of the site x. The last two embeddings are pure. □

Let L, R, and D be three site-graphs, such that R is included in D, and let ϕ be a sliding embedding from L into D. Then there exist a site graph D' that is included in L and a sliding embedding ψ from D' to R such that $i_{R,D}\psi =$

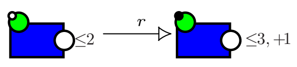
$$\begin{tikzcd}
 & L \arrow[r, hook, "\phi", "\$"'] & D \\
 & D' \arrow[u, hook, "\subseteq"] \arrow[r, hook, "\psi", "\$"'] \arrow[ur, phantom, "\lrcorner"] & R \arrow[u, hook, "\subseteq"'] \\
D'' \arrow[uur, hook, bend left, "\subseteq"] \arrow[ur, hook, dotted, "\subseteq"] \arrow[urr, hook, bend right, "\psi'", "\$"'] & &
\end{tikzcd}$$

(a) Pull-back.

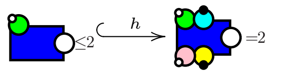
$$\begin{tikzcd}
L & & R & & R' \\
 & D \arrow[ul, hook, "\subseteq"] \arrow[ur, hook, "\$"'] & & D' \arrow[ul, hook, "\subseteq"'] \arrow[ur, hook, "\$"'] & \\
 & & D'' \arrow[ul, hook, "\subseteq"] \arrow[ur, hook, "\$"'] \arrow[u, phantom, "\wedge"] & &
\end{tikzcd}$$

(b) Composition.

Fig. 5. Composition of partial sliding embeddings.



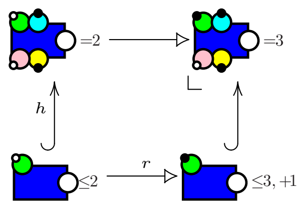
(a) A rule r.



(b) A pure embedding h from the lhs of the rule r to the site-graph G_4.

$$\begin{tikzcd}
 & & R'' \\
L' \arrow[r, "r'"'] \arrow[urr, bend left, "r''"] & R' \arrow[ur, hook, dotted, "!h"'] \arrow[d, phantom, "\llcorner"] & \\
L \arrow[u, hook, "h_L"] \arrow[r, "r"] & R \arrow[u, hook, "h_R"] \arrow[uur, hook, bend right, "h'_R"'] &
\end{tikzcd}$$

(c) Universal construction



(d) Application of the rule r on the site-graph G_4.

Fig. 6. Rule application.

$\phi i_{D',L}$ and such that D' is *maximal* (w.r.t. inclusion among site-graphs) for this property. The pair $(D', i_{D',L}, \psi)$ is called the *pull-pack* of the pair $(\phi, i_{R,D})$.

Let L, R, and D be three site-graphs such that D is included in L. A *partial sliding embedding* from L into R is defined as a pair made of the inclusion $i_{D,L}$ and a sliding embedding from D to R. Sliding embeddings may be considered as partial sliding embeddings with the inclusion as the identity embedding. Partial sliding embeddings compose by the means of a pull-back (e.g. see Fig. 5(b)).

2.4 Rules

Rules represent transformations between site-graphs. For the sake of simplicity, we only use a fragment of Kappa (we assume here that there are no *side effects*). Rules may break and create bonds between pairs of sites, change the properties of sites, update the value of counters. They may also create and remove agents. When an agent is created, all its sites must be fully specified: binding sites may be either free, or bound to a specific site, and the value of counters must be singletons. So as to ensure that there is no side-effect when an agent is removed, we also assume that the binding sites of removed agents are fully specified. These requirements are formalized as follows:

Definition 4 (rule). *A* rule *is a partial sliding embedding* $L \leftarrowtail\!\!\supseteq\!\!\hookleftarrow D \xhookrightarrow[\$]{(h_e,h_\$)} R$ *such that:*

1. *(modified agents) for all agents* $n \in \mathcal{A}_D$ *such that* $h_e(n) \in \mathcal{A}_R$ *and for every site identifier* $i \in \Sigma_{site}(type_L(n))$*,*
   (a) *the site* (n,i) *belongs to the set* $\mathcal{S}_L$ *if and only if* $(h_e(n),i)$ *belongs to set* $\mathcal{S}_R$*;*
   (b) *if the site* (n,i) *belongs to the set* $\mathcal{S}_L^{lnk}$*, then either* $\mathcal{L}_L(n,i) = -$ *and* $\mathcal{L}_R(h_e(n),i) = -$*, or* $\mathcal{L}_L(n,i) \in \mathcal{S}_L^{lnk} \cup \{\dashv\}$ *and* $\mathcal{L}_R(h_e(n),i) \in \mathcal{S}_R^{lnk} \cup \{\dashv\}$*;*
   (c) *if the site* (n,i) *belongs to the set* $\mathcal{S}_L^{\$}$*, then the sets* $c\kappa_R(h_e(n),i)$ *and* $\{h_\$(v) \mid v \in c\kappa_L(n,i)\}$ *are equal.*
2. *(removed agents) for all agents* $n \in \mathcal{A}_L$ *such that* $n \notin \mathcal{A}_D$*, for every site identifier* $i \in \Sigma_{ag\text{-}st}^{lnk}(type_L(n))$*,* $(n,i) \in \mathcal{S}_L^{lnk}$ *and* $\mathcal{L}_L(n,i) \in \mathcal{S}_L^{lnk} \cup \{\dashv\}$*.*
3. *(created agents) for all agents* $n \in \mathcal{A}_R$ *for which there exists no* $n' \in \mathcal{A}_D$ *such that* $n = h_e(n')$*, and for every site identifier* $i \in \Sigma_{site}(type_R(n))$*,*
   (a) *the site* (n,i) *belongs to the set* $\mathcal{S}_R$*;*
   (b) *if the site* (n,i) *belongs to the set* $\mathcal{S}_R^{lnk}$*, then the binding state* $\mathcal{L}_R(n,i)$ *belong to the set* $\mathcal{S}_R^{lnk} \cup \{\dashv\}$*;*
   (c) *if the site* (n,i) *belongs to the set* $\mathcal{S}_R^{\$}$*, then* $c\kappa_R(n,i)$ *is a singleton.*

In Def. 4, each agent that is *modified* occurs on both hand sides of a rule. Constraint 1a ensures that they document the same sites. Constraint 1b ensures that, if the binding state of a site is modified, then it has to be fully specified (either free, or bound to a specific site) in both hand sides of the rule. Constraint 1c ensures that the post-condition associated to a counter is the direct image of its precondition by its update function. Constraint 2 ensures that the agents that are *removed* have their binding sites fully specified. Constraint 3a ensures that, in the agents that are *created*, all the sites are documented. Beside, constraint 3b requires that the state of their binding site is either free or bound to a specific site. Constraint 3c ensures that their counters have a single value.

An example of a rule is given in Fig. 6(a).

A rule $L \leftarrowtail\!\!\supseteq\!\!\hookleftarrow D \xhookrightarrow[\$]{} R$ is usually denoted as $L \;\rightharpoonup\; R$ (leaving the common region and the sliding embedding implicit). Rules are applied to site-graphs via pure embeddings using the *single push-out* construction [22].

Definition 5 (rule application [14]). *Let* r *be a rule* $L \rightharpoonup R$*,* L' *be a site-graph, and* h_L *be a pure embedding from* L *to* L'*. Then, there exists a rule* $r' : L' \rightharpoonup R'$ *and a pure embedding* $h_R : R \hookrightarrow R'$ *such that the following properties are satisfied (e. g. see Fig. 6(c)):*

1. $h_R r = r' h_L$*;*
2. *for all rules* r'' *between the site-graph* L' *and a site-graph* R'' *and all embeddings* h'_R *from* R *into* R'' *such that* $h'_R r = r'' h_L$*, there exists a unique pure embedding* h *from* R' *into* R'' *such that* $r'' = hr'$ *and* $h'_R = hh_R$*.*

Moreover, whenever the site-graph L' *is a chemical mixture, the site-graph* R' *is a chemical mixture as well.*

We write $L' \xrightarrow{r} R'$ for a transition from the state L' into the state R' via an application of a rule r. Usually transition labels also mention the pure embedding (h_L here), but we omit it since we do not use it in the rest of the paper.

Example 4 (running example). An example of rule application is depicted in Fig. 6. We consider the rule r that takes a protein with the site a unphosphorylated and a counter with a value at least equal to 2*, and that phosphorylates the site a while incrementing the counter by* 1 *(e. g. see Fig. 6(a)). Note that the update function of the counter is written next to its post-condition in the right hand side of the rule. We apply the rule to a protein with the sites b and c phosphorylated, the site d unphosphorylated, and the counter equal to* 2 *(e. g. see Fig. 6(b)). The result is a protein with the sites a, b, and c phosphorylated, the site d unphosphorylated and the counter equal to* 3 *(e. g. see Fig. 6(d)).* $\square$

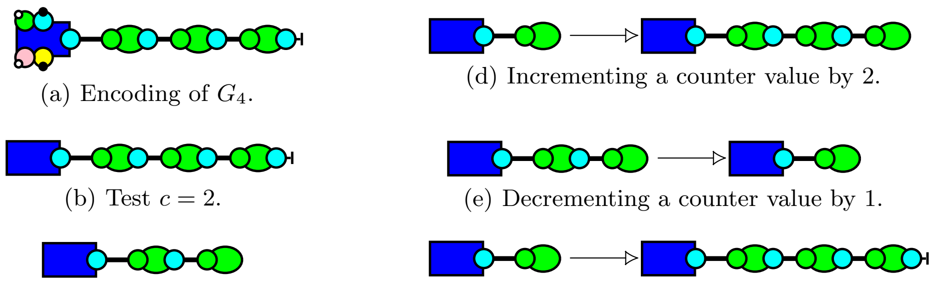
A model $\mathcal{M}$ over a given signature Σ is defined as the pair $(G_0, \mathcal{R})$ where G_0 is a chemical mixture, representing the initial state, and $\mathcal{R}$ is a set of rules. Each rule is associated with a functional rate which maps each potential tuple of values for the counters of the left hand side of the rule to a non negative real number. We write $\mathcal{C}(\mathcal{M})$ for the set of states obtained from G_0 by applying a potentially empty sequence of rules in $\mathcal{R}$.

3 Encoding counters

In this section, we introduce two encodings from Kappa with counters into Kappa without counters. As explained in Sect. 1, our goal is to preserve the rigidity of site-graphs and to avoid the blow-up of the number of rules in the target model. This is mandatory to preserve the good performances of the Kappa simulator. Both encodings rely on syntactic restrictions over the preconditions and the update functions that may be applied to counters and on semantics ones about the potential range of counters. In Sects. 4 and 5, we provide a static analysis to check whether, or not, these semantics assumptions hold.

3.1 Encoding the value of counters as unbounded chains of agents

In this encoding, each counter is bound to a chain of fictitious agents the length of which minus 1 denotes the value of the counter (another encoding not requiring the subtraction is possible but it would require side-effects). Encoding counters as chains of agents has already been used in the implementation of two-counter machines in Kappa [19,34]. We slightly extend these works to implement more atomic operations over counters. We assume that the value of counters is bounded from below. For the sake of simplicity, we assume that counters range in $\mathbb{N}$, but arbitrary lower bounds may be considered by shifting each value accordingly. We denote by Ω_1 the set of the site-graphs that have a counter with a negative value. They are considered as erroneous states, since they may not be encoded with chains of agents.


(a) Encoding of G_4.

(b) Test $c = 2$.

(c) Test $c \geq 1$.

(d) Incrementing a counter value by 2.

(e) Decrementing a counter value by 1.

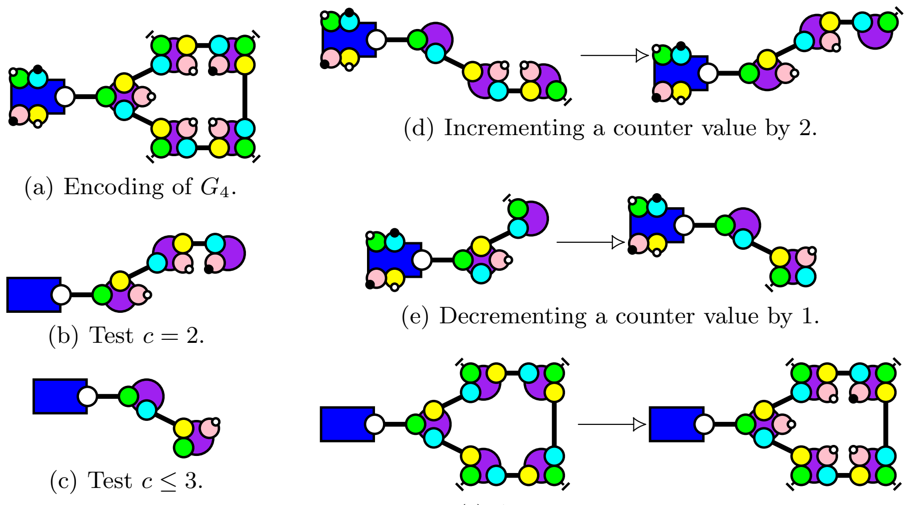
(f) Assigning 2 to a counter.

Fig. 7. Encoding the value of counters as unbounded chains of agents.

Only two kinds of guards are handled. A rule may require the value of a counter to be equal to a given number or that the value of a counter is greater than a given number. Rules testing whether a value is less than a given number require unfolding each such rule into several ones (one per potential value). Also when the rate of a rule depends on the value of some counters, we unfold each rule according to the value of these counters, so that the rate of each unfolded rule is a constant (the Kappa simulator requires all the instances of a given rule in a given simulation state to have the same rate, for efficiency concerns). For update functions, we only consider constant functions and the functions that increase/decrease the value of counters by a fixed value. Testing whether the value of a counter is equal to (resp. greater than) n, can be done by requiring the corresponding chain to contain exactly (resp. at least) $n+1$ agents (e. g. see Figs. 7(b) and 7(c)). Incrementing (resp. decrementing) the value of a counter is modeled by inserting (resp. removing) agents at the beginning its chain (e. g. see Fig. 7(d), resp. Fig. 7(e)). Setting a counter to a fixed value, requires to detach its full chain in order to create a new one of the appropriate length (e. g. see Fig. 7(f)). In such a case, the former chain remains as a junk. Thus the state of the model must be understood up to insertion of junk agents. We introduce the function gc_1 that removes every chain of spurious agents not bound to any counter. We denote as $[\![G]\!]_1^g$ (resp. $[\![r]\!]_1^r$) the encoding of a site-graph G (resp. of a rule r).

3.2 Encoding the value of counters as circular lists of agents

In this second encoding, each counter is bound to a ring of agents. Each such agent has three binding sites *zero*, *pred*, and *next*, and a property site *value* which may be activated, or not. In a ring, agents are connected circularly through their site *pred* and *next*. Exactly one agent per ring is bound to a counter and exactly one agent per ring has the site *value* activated. The value of the counter is encoded by the distance between the agent bound to the counter and the agent that is activated, scanning the agents by following the direction given by the site *next* of each agent (clock-wisely in the graphical representation). We have to



(a) Encoding of G_4.

(b) Test $c = 2$.

(c) Test $c \leq 3$.

(d) Incrementing a counter value by 2.

(e) Decrementing a counter value by 1.

(f) Assigning 2 to a counter.

Fig. 8. Encoding the value of counters as circular lists of agents.

consider that counter values are bounded from above and below. Without any loss of generality, we assume that the length of each ring is the same, that is to say that counters range from 0 to $n-1$, for a given $n \in \mathbb{N}$. We denote by Ω_2 the set of the site-graphs with at least one counter not satisfying these bounds.

Compared to the first encoding, this one may additionally cope with testing that a counter has a value less than a given constant without having to unfold the rule. Both encodings may deal with the same update functions. Testing whether a counter is equal to a value is done by requiring that the activated agent is at the appropriate distance of the agent that is connected to the counter (e. g. see Fig. 8(b)). It is worth noting that the intermediary agents are required to be not activated. This is not mandatory for the soundness of the encoding, this is an optimization that helps the simulator for detecting early that no embedding may associate a given agent of the left hand side of a rule to a given agent in the current state of the system. Inequalities are handled by checking that enough agents starting from the one that is connected to the counter and in the direction specified by the direction of the inequality, are not activated (e. g. see Fig. 8(c)). Incrementing/decrementing the value of a counter is modeled by making counter glide along the ring (e. g. see Figs. 8(d) and 8(e)). Special care has to be taken to ensure that the activated agent never crosses the agent linked to the counter (which would cause a numerical wrap-around). Assigning a given value to a counter requires to entirely remove the ring and to replace it with a fresh one (e. g. see Fig. 8(f)). It may be efficiently implemented without memory allocation. As in the first encoding, when the rate of a rule depends on the value of some counters, we unfold each rule according to the value of these counters, so that the rate of each unfolded rule is constant.

We introduce the function gc_2 as the identity function over site-graphs (there are no junk agent in this encoding). We denote as $[\![G]\!]_2^g$ (resp. $[\![r]\!]_2^r$) the encoding without counter of a site-graph G (resp. of a rule r).

3.3 Correspondence

The following theorem states that, whenever there is no numerical overflow and providing that junk agents are neglected, the semantics of Kappa with counters and the semantics of their encodings are in bisimulation.

Theorem 1 (correspondence). *Let i be either 1 or 2. Let G be a fully specified site-graph such that $G \notin \Omega_i$ and r be a rule. Both following properties are satisfied:*

1. *whenever there exists a site-graph G' such that $G \xrightarrow{r} G'$ and $G' \notin \Omega_i$, there exists a site-graph $G'_{\$}$, such that $[\![G]\!]_i^g \xrightarrow{[\![r]\!]_i^r} G'_{\$}$ and $[\![G']\!]_i^g = gc_i(G'_{\$})$;*
2. *whenever there exists a site-graph $G'_{\$}$ such that $[\![G]\!]_i^g \xrightarrow{[\![r]\!]_i^r} G'_{\$}$, there exists a site-graph G' such that $G \xrightarrow{r} G'$, $G' \notin \Omega_i$, and $[\![G']\!]_i^g = gc_i(G'_{\$})$.*

3.4 Benchmarks

The experimental evaluation of the impact of both encodings to the performance of the simulator KaSim [17,6] is presented in Fig. 9. We focus on the example that has been presented in Sec. 1. We plot the number of events that are simulated per second of CPU. For the sake of comparison, we also provide the simulation efficiency of the simulator NFSim [40] on the models written in BNGL with equivalent sites (with a linear number of rules only).

We notice that, with KaSim, the direct approach (without counter) is the most efficient when there are less than 9 phosphorylation sites. We explain this overhead, by the fact that each encoding utilizes spurious agents that have to be allocated in memory and relies on rules with bigger left hand sides. Nevertheless this overhead is reasonable if we consider the gain in conciseness in the description of the models. The versions of models with counters rely on a linear number of rules, which make models easier to read, document, and update. For more phosphorylation sites, simulation time for models written without counters blow up very quickly, due to the large number of rules. The simulation of the models with counters scales much better for both encodings.

Models can be concisely described in BNGL without using counters, by the means of equivalent sites. Each version of the model uses n indistinguishable sites and only a linear number of rules is required. However, detecting the potential applications of rules in the case of equivalent sites relies on the sub-graph isomorphism problem on general graphs, which prevent the approach to scale to large value of n. We observe that the efficiency of NFSim on this family of examples is not as good as the ones of KaSim (whatever which of the three modeling methods is used). We also observe a very quick deterioration of the performances starting at n equal to 5.

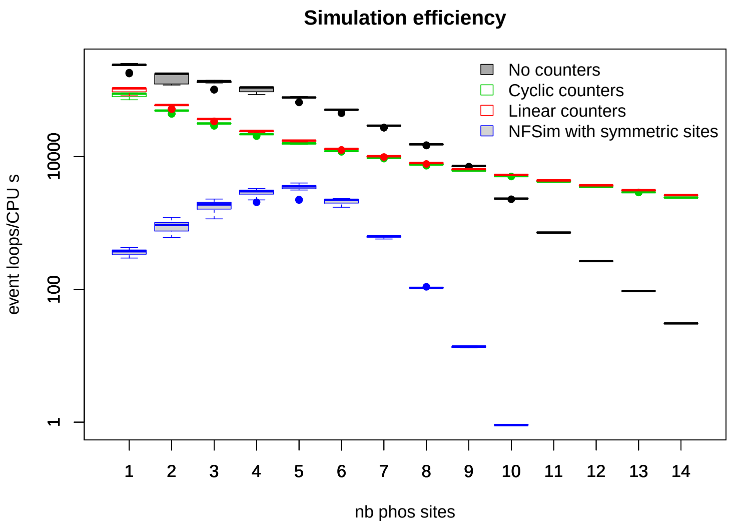


Fig. 9. Efficiency of the simulation for the example in Sec. 1 with n ranging between 1 and 14. We test the simulator KaSim with a version of the models written without counters and versions of the models according to both encodings (including the n phosphorylation sites). For the sake of comparison, we also compare with the efficiency of the simulator NFSim with the same model but written in BNGL by the means of equivalent sites. For each version of the model and each simulation method, we run 15 simulations of 10^5 events on an initial state made of 100 agents and we plot the number of computation steps computed in average per second of CPU on a log scale. Every simulation has been performed on 4 processors: Intel(R) Xeon(R) CPU E5-2609 0 @ 2.40GHz 126 Go of RAM, running ubuntu 18.04.

4 Generic abstraction of reachable states

So far, we have provided two encodings to compile Kappa with counters into Kappa without counters. These encodings are sound under some assumptions over the range of counters. Now we propose a static analysis not only to check that these conditions are satisfied, but also to infer the meaning of the counters (in our case study, that they are equal to the number of phosphorylated sites).

Firstly, we provide a generic abstraction to capture the properties of the states that a Kappa model may potentially take. Our abstraction is parametric with respect to the class of properties. It will be instantiated in Sect. 5. Our analysis is not complete: not all the properties of the program are discovered; nevertheless, the result is sound: all the properties that are captured, are correct.

4.1 Collecting semantics

Let $\mathcal{Q}$ be the set of all the site-graphs. We are interested in the set $\mathcal{C}(\mathcal{M})$ of all the states that a model $\mathcal{M} = (G_0, \mathcal{R})$ may take in 0, 1, or more computation steps. This is the collecting semantics [7]. By [33], it may be expressed as the least fixpoint of the $\cup$-complete endomorphism $\mathbb{F}$ on the complete lattice $\wp(\mathcal{Q})$ that is defined as $\mathbb{F}(X) = \{G_0\} \cup \{q' \mid \exists q \in X, r \in \mathcal{R} \text{ such that } q \xrightarrow{r} q'\}$. By [42], the collecting semantics is also equal to the meet of all the post-fixpoints of the function $\mathbb{F}$ (i. e. $\mathcal{C}(\mathcal{M}) = \bigcap\{X \in \wp(\mathcal{Q}) \mid \mathbb{F}(X) \subseteq X\}$), that is to say the strongest inductive invariant of our model that is satisfied by the initial state.

4.2 Generic abstraction

The collecting semantics is usually not decidable. We use the Abstract Interpretation framework [9,10] to compute a sound approximation of it.

Definition 6 (abstraction). *A tuple $\mathcal{A} = (\mathcal{Q}^\sharp, \sqsubseteq, \gamma, \sqcup, \bot, \mathcal{I}^\sharp, t^\sharp, \nabla)$ is called an abstraction when all following conditions are satisfied:*

1. *the pair $(\mathcal{Q}^\sharp, \sqsubseteq)$ is a pre-order of abstract properties;*
2. *the component $\gamma : \mathcal{Q}^\sharp \to \wp(\mathcal{Q})$ is a monotonic map (i. e. for every two abstract elements $q_1^\sharp$, $q_2^\sharp \in \mathcal{Q}^\sharp$ such that $q_1^\sharp \sqsubseteq q_2^\sharp$, we have $\gamma(q_1^\sharp) \subseteq \gamma(q_2^\sharp)$);*
3. *the component $\sqcup$ maps each finite set of abstract properties $X^\sharp \in \wp_{finite}(\mathcal{Q}^\sharp)$ to an abstract property $\sqcup(X^\sharp) \in \mathcal{Q}^\sharp$ such that for each abstract property $q^\sharp \in X^\sharp$, we have: $q^\sharp \sqsubseteq \sqcup(X^\sharp)$;*
4. *the component $\bot \in \mathcal{Q}^\sharp$ is an abstract property such that $\gamma(\bot) = \emptyset$;*
5. *the component $\mathcal{I}^\sharp$ is an element of the set $\mathcal{Q}^\sharp$ such that $\{G_0\} \subseteq \gamma(\mathcal{I}^\sharp)$;*
6. *the component $t^\sharp$ is a function mapping each pair $(q, r) \in \mathcal{Q}^\sharp \times \mathcal{R}$ to an abstract property $t^\sharp(q, r) \in \mathcal{Q}^\sharp$ such that: $\forall q^\sharp \in \mathcal{Q}^\sharp$, $\forall q \in \gamma(q^\sharp)$, $\forall r \in \mathcal{R}$, $\forall q' \in \mathcal{Q}$, we have $q' \in \gamma(t^\sharp(q^\sharp))$ whenever $q \xrightarrow{r} q'$;*
7. *the component $\nabla : \mathcal{Q}^\sharp \times \mathcal{Q}^\sharp \to \mathcal{Q}^\sharp$ satisfies both following properties:*
   (a) $\forall q_1^\sharp$, $q_2^\sharp \in \mathcal{Q}^\sharp$, $q_1^\sharp \sqsubseteq q_1^\sharp \nabla q_2^\sharp$ and $q_2^\sharp \sqsubseteq q_1^\sharp \nabla q_2^\sharp$,
   (b) $\forall (q_n^\sharp)_{n\in\mathbb{N}} \in \left(\mathcal{Q}^\sharp\right)^{\mathbb{N}}$, the sequence $(q_n^\nabla)_{n\in\mathbb{N}}$ that is defined as $q_0^\nabla = q_0^\sharp$ and $q_{n+1}^\nabla = q_n^\nabla \nabla q_{n+1}^\sharp$ for every integer $n \in \mathbb{N}$, is ultimately stationary.

The set $\mathcal{Q}^\sharp$ is an abstract domain. It captures the properties of interest, and abstracts away the others. Each property $q^\sharp \in \mathcal{Q}^\sharp$ is mapped to the set of the concrete states $\gamma(q^\sharp)$ which satisfy this property by the means of the concretization function γ. The pre-order $\sqsubseteq$ describes the amount of information which is known about the properties that we approximate. We use a pre-order to allow some concrete properties to be described by several unrelated abstract elements. The abstract union $\sqcup$ is used to gather the information described by a finite number of abstract elements. It may not necessarily compute the least upper bound of a finite set of abstract elements (this least bound may not even exist). The abstract element $\bot$ provides the basis for abstract iterations. The concretization function is strict which means that it maps the element $\bot$ to the empty set. The abstract property $\mathcal{I}^\sharp$ is satisfied by the initial state. The function $t^\sharp$ is used to mimic concrete rewriting steps in the abstract. The operator ∇ is called a widening. It ensures the convergence of the analysis in finitely many iterations.

Given an abstraction $(\mathcal{Q}^\sharp, \sqsubseteq, \gamma, \sqcup, \bot, \mathcal{I}^\sharp, t^\sharp, \nabla)$, the abstract counterpart $\mathbb{F}^\sharp$ to $\mathbb{F}$ is defined as $\mathbb{F}^\sharp(q^\sharp) = \sqcup^\sharp \left(\{q^\sharp, \mathcal{I}^\sharp\} \cup \{t^\sharp(q^\sharp, r) \mid r \in \mathcal{R}\}\right)$. The function $\mathbb{F}^\sharp$ satisfies the soundness condition $\forall q^\sharp \in \mathcal{Q}^\sharp$, $[\mathbb{F} \circ \gamma](q^\sharp) \subseteq [\gamma \circ \mathbb{F}^\sharp](q^\sharp)$. Following [7], we compute a sound and decidable approximation of our abstract semantics by using the widening operator ∇. The abstract iteration [10,11] of $\mathbb{F}^\sharp$ is defined by the following induction: $\mathbb{F}_0^\nabla = \bot$ and, for each integer $n \in \mathbb{N}$, $\mathbb{F}_{n+1}^\nabla = \mathbb{F}_n^\nabla$ whenever $\mathbb{F}^\sharp(\mathbb{F}_n^\nabla) \sqsubseteq \mathbb{F}_n^\nabla$, and $\mathbb{F}_{n+1}^\nabla = \mathbb{F}_n^\nabla \nabla \mathbb{F}^\sharp(\mathbb{F}_n^\nabla)$ otherwise.

Theorem 2 (Termination and soundness). *The abstract iteration is ultimately stationary and its limit* $\mathbb{F}^\nabla$ *satisfies* $\mathcal{C}(\mathcal{M}) \subseteq \gamma(\mathbb{F}^\nabla)$.

Proof. By construction, $\mathbb{F}^\sharp(\mathbb{F}^\nabla) \sqsubseteq \mathbb{F}^\nabla$. Since γ is monotonic, it follows that: $\gamma(\mathbb{F}^\sharp(\mathbb{F}^\nabla)) \subseteq \gamma(\mathbb{F}^\nabla)$. Since, $\mathbb{F} \circ \gamma \dot{\subseteq} \gamma \circ \mathbb{F}^\sharp$, $\mathbb{F}(\gamma(\mathbb{F}^\nabla)) \subseteq \gamma(\mathbb{F}^\nabla)$. So $\gamma(\mathbb{F}^\nabla)$ is a post-fixpoint of $\mathbb{F}$. By [42], we have *lfp* $\mathbb{F} \subseteq \gamma(\mathbb{F}^\nabla)$. □

4.3 Coalescent product

Two abstractions may be combined pair-wise to form a new one. The result is a coalescent product that defines a mutual induction over both abstractions.

Definition 7 (coalescent product). *The coalescent product between two abstractions* $(\mathcal{Q}_1^\sharp, \sqsubseteq_1, \gamma_1, \sqcup_1, \bot_1, \mathcal{I}_1^\sharp, t_1^\sharp, \nabla_1)$ *and* $(\mathcal{Q}_2^\sharp, \sqsubseteq_2, \gamma_2, \sqcup_2, \bot_2, \mathcal{I}_2^\sharp, t_2^\sharp, \nabla_2)$. *is defined as the tuple* $(\mathcal{Q}^\sharp, \sqsubseteq, \gamma, \sqcup, \bot, \mathcal{I}^\sharp, t^\sharp, \nabla)$ *where*

1. $\mathcal{Q}^\sharp = \mathcal{Q}_1^\sharp \times \mathcal{Q}_2^\sharp$;
2. $\sqsubseteq$, $\sqcup$, $\bot$, *and* ∇ *are defined pair-wise;*
3. γ *maps every pair* $(q_1^\sharp, q_2^\sharp)$ *to the meet* $\gamma_1(q_1^\sharp) \cap \gamma_2(q_2^\sharp)$ *of their respective concretization;*
4. $\mathcal{I}^\sharp = (\mathcal{I}_1^\sharp, \mathcal{I}_2^\sharp)$;
5. $t^\sharp$ *maps every pair* $((q_1^\sharp, q_2^\sharp), r) \in \mathcal{Q}^\sharp \times \mathcal{R}$ *made of a pair of abstract properties and a rule to the abstract property* $(t_1^\sharp(q_1^\sharp, r), t_2^\sharp(q_2^\sharp, r))$ *whenever* $t_1^\sharp(q_1^\sharp, r) \neq \bot_1$ *and* $t_2^\sharp(q_2^\sharp, r) \neq \bot_2$, *and to the pair* $(\bot_1, \bot_2)$ *otherwise.*

Theorem 3 (Soundness of the coalescent product). *The coalescent product of two abstractions is an abstraction as well.*

We notice that if either of both abstractions proves that the precondition of a rule is not satisfiable, then this rule is discarded in the other abstraction (hence the term coalescent). By mutual induction, the composite abstraction may detect which rules may be safely discarded along the iterations of the analysis.

We may now define an analysis modularly with respect to the class of considered properties. We use the coalescent product to extend the existing static analyzer KaSa [5] with a new abstraction dedicated to the range of counters.

5 Numerical abstraction

Now we specialize our generic abstraction to detect and prove safe bounds to the range of counters. In general, this requires to relate the value of the counters to the state of others sites. Our approach consists in translating each protein configuration into a vector of relative numbers and in abstracting each rule by its potential effect on these vectors. We obtain an integer linear programming problem that we will solve by choosing an appropriate abstract domain.

The set of convex parts of $\mathbb{Z}$ is written as $\mathcal{I}_{\mathbb{Z}}$. We assume that guards on counters are element of $\mathcal{I}_{\mathbb{Z}}$ and that each update function either set counters to a constant value, or increment/decrement counters by a constant value.

5.1 Encoding states and preconditions

We propose to translate each agent into a set of numerical constraints. A protein of type A is associated with one variable χ_i^{λ} for each binding site i and each binding state λ, one variable χ_i^{ι} for each property site i and each internal state identifier ι, and one variable val_i for each counter in i.

Definition 8 (numerical variables). *Let $A \in \Sigma_{ag}$ be an agent type. We define the set $\mathcal{V}ar_A$ as the set of variables $\mathcal{V}ar_A^{lnk} \cup \mathcal{V}ar_A^{int} \cup \mathcal{V}ar_A^{\$}$ where:*

1. $\mathcal{V}ar_A^{lnk} = \{\chi_i^{\lambda} \mid i \in \Sigma_{ag\text{-}st}^{lnk}(A), \lambda \in \{\dashv\} \cup \{(A', i') \mid A' \in \Sigma_{ag}, i' \in \Sigma_{ag\text{-}st}^{lnk}(A')\}\}$;
2. $\mathcal{V}ar_A^{int} = \{\chi_i^{\iota} \mid i \in \Sigma_{ag\text{-}st}^{int}(A), \iota \in \Sigma_{int}\}$;
3. $\mathcal{V}ar_A^{\$} = \{val_i \mid i \in \Sigma_{ag\text{-}st}^{\$}\}$.

Intuitively, variables of the form χ_i^{λ} (resp. χ_i^{ι}) take the value 1 if the binding (resp. internal) state of the site i is λ (resp. ι), whereas the variables of the form val_i takes the value of the counter i.

Each agent of type A may be translated into a function mapping each variable in the set $\mathcal{V}ar_A$ into a subset of the set $\mathbb{Z}$. Such a function is called a guard.

Definition 9 (Encoding of agents). *Let G be a site-graph and n be an agent in $\mathcal{A}_G$. We denote by A the type $type_G(n)$. We define as follows the function $guard_G(n)$ from the set $\mathcal{V}ar_A$ into the set $\mathcal{I}_{\mathbb{Z}}$:*

1. *$guard_G(n)(\chi_i^{\dashv})$ is equal to the singleton $\{1\}$ whenever $(n,i) \in \mathcal{S}_G^{lnk}(A)$ and $\mathcal{L}_G(n,i) = \dashv$, to the singleton $\{0\}$ whenever $(n,i) \in \mathcal{S}_G^{lnk}(A)$ and $\mathcal{L}_G(n,i) \neq \dashv$, and to the set $\{0,1\}$ whenever $(n,i) \notin \mathcal{S}_G^{lnk}(A)$;*
2. *$guard_G(n)(\chi_i^{(A',i')})$ is equal to the singleton $\{1\}$ whenever $(n,i) \in \mathcal{S}_G^{lnk}(A)$ and there exists $n' \in \mathcal{A}_G$ such that both conditions $type_G(n') = A'$ and $\mathcal{L}_G(n,i) = (n',i')$ are satisfied, to the singleton $\{0\}$ whenever $(n,i) \in \mathcal{S}_G^{lnk}(A)$ and either $\mathcal{L}_G(n,i) = \dashv$, or there exist an agent identifier $n'' \in \mathcal{A}_G$ and a site name $i'' \in \Sigma_{site}$ such that $(type_G(n''), i'') \neq (A', i')$, and to the set $\{0,1\}$ whenever $(n,i) \notin \mathcal{S}_G^{lnk}(A)$ or $\mathcal{L}_G(n,i) = -$;*
3. *$guard_G(n)(\chi_i^{\iota})$ is equal to the singleton $\{1\}$ whenever $(n,i) \in \mathcal{S}_G^{int}(A)$ and $p\kappa_G(n,i) = \iota$; to the singleton $\{0\}$ whenever $(n,i) \in \mathcal{S}_G^{int}(A)$ and $p\kappa_G(n,i) \neq \iota$; and to set $\{0,1\}$ whenever $(n,i) \notin \mathcal{S}_G^{int}(A)$.*

4. $guard_G(n)(val_i)$ is equal to the set $c\kappa_G(c)$ whenever $(n,i) \in \mathcal{S}_G^{\$}$ and to the set $\mathbb{Z}$ otherwise.

The variable $\chi_i^{\dashv}$ takes the value $\{1\}$ if we know that the site i is free, the value $\{0\}$ if we know that it is bound, and the value $\{0,1\}$ if we do not know whether the site is free or not. This is the same for binding type, the variable $\chi_i^{(A',i')}$ takes the value $\{1\}$ if we know that the site is bound to the site i' of an agent of type A', the value $\{0\}$ if we know that this is not the case, and the value $\{0,1\}$ otherwise. Property sites work the same way. Lastly, the variable val_i takes as value the set attached to the counter or the value $\mathbb{Z}$ if the site is not mentioned in the agent. We notice that when n is a fully-specified agent of type A, the function $guard_G(n)$ maps every variable in the set $\mathcal{V}ar_A$ to a singleton.

Example 5 (running example). We provide the translation of the unique agent of the site-graph G_1 (e. g. see Fig. 3(a)) and the one of the unique agent of the site-graph G_4 (e. g. see Fig. 3(d)).

The agent of the site-graph G_1 is translated as follows:

$$\left\{\begin{array}{l} \chi_a^{\circ} = \{1\}; \chi_a^{\bullet} = \{0\}; \\ \chi_b^{\circ} = \{0,1\}; \chi_b^{\bullet} = \{0,1\}; \\ \chi_c^{\circ} = \{0,1\}; \chi_c^{\bullet} = \{0,1\}; \\ \chi_d^{\circ} = \{0,1\}; \chi_d^{\bullet} = \{0,1\}; \\ val_x = \{z \in \mathbb{Z} \mid z \leq 2\} \end{array}\right\}.$$

According to the first two constraints, the site a is unphosphorylated. According to the next six ones, the sites b, c, and d have an unspecified state. According to the last constraint, the value of the counter must be less than or equal to 2.

The translation of the agent of the site-graph G_4 is obtained the same way:

$$\left\{\begin{array}{l} \chi_a^{\circ} = \{1\}; \chi_a^{\bullet} = \{0\}; \\ \chi_b^{\circ} = \{0\}; \chi_b^{\bullet} = \{1\}; \\ \chi_c^{\circ} = \{0\}; \chi_c^{\bullet} = \{1\}; \\ \chi_d^{\circ} = \{1\}; \chi_d^{\bullet} = \{0\}; \\ val_x = \{2\} \end{array}\right\}.$$

This means that the sites b and c are phosphorylated while the sites a and d are not. According to the last constraint, the value of the counter is equal to 2.

5.2 Encoding rules

In Kappa, a rule may be applied only when its precondition is satisfied. Moreover, the application of a rule modifies the state of some sites in agents. We translate each rule into a tuple of guards that encodes its precondition, a set of non-invertible assignments (when a site is given a new state that does not depend on the former one), and a set of invertible assignments (when the new state of a site depends on the previous one). Such a distinction is important as we want to establish relationships among the value of some variables [32]: a non-invertible assignment completely hides the former value of a variable. This is not the case with invertible assignments for which relationships may be propagated more easily. The agents that are created (which have no precondition) and the ones that are removed (which disappear), have a special treatment.

Definition 10 (Encoding of rules). *Each rule* $r : L \overset{\supseteq}{\hookleftarrow} D \overset{(h_e,h_\$)}{\hookrightarrow} R$ *is associated with the tuple* $(pre_r, \textit{not-invert}_r, invert_r, new_r)$ *where:*

1. pre_r *maps every agent* $n \in \mathcal{A}_L$ *in the left hand side of the rule* r *to its guard* $guard_L(n)$*;*
2. $\textit{not-invert}_r$ *maps every agent* $n \in \mathcal{A}_D$ *and every variable* $v \in \mathcal{V}ar_{type_D(n)}$ *such that the set* $guard_R(h_e(n))(v)$ *is a singleton and* $guard_R(h_e(n))(v) \neq guard_L(n)(v)$ *to the unique element of the set* $guard_R(h_e(n))(v)$*.*
3. $invert_r$ *maps every agent* $n \in \mathcal{A}_D$ *and every variable* $v \in \mathcal{V}ar_{type_D(n)}$ *such that the set* $guard_R(h_e(n))(v)$ *is not a singleton and* $h_\$(n,i)$ *is a function of the form* $[z \in \mathbb{Z} \mapsto z + c]$ *with* $c \in \mathbb{Z}$*, to the relative number* c*.*
4. new_r *maps every agent* $n' \in \mathcal{A}_R$ *such that there is no agent* $n \in \mathcal{A}_D$ *satisfying* $h_e(n) = n'$ *to the guard* $guard_R(n')$*.*

Example 6 (running example). The encoding of the rule of Fig. 6(a) is given as follows:

– the function pre_r maps the agent 1 to the following set of constraints:

$$\left\{\begin{array}{l} \chi_a^\circ = \{1\}; \chi_a^\bullet = \{0\}; \\ \chi_b^\circ = \{0,1\}; \chi_b^\bullet = \{0,1\}; \\ \chi_c^\circ = \{0,1\}; \chi_c^\bullet = \{0,1\}; \\ \chi_d^\circ = \{0,1\}; \chi_d^\bullet = \{0,1\}; \\ val_x = \{z \in \mathbb{Z} \mid z \leq 2\} \end{array}\right\};$$

– the function $\textit{not-invert}_r$ maps the pair $(1, \chi_a^\circ)$ to the value 0, and the pair $(1, \chi_a^\bullet)$ to the value 1;
– the function $invert_r$ maps the pair $(1, x)$ to the successor function;
– the function new_r is the function with the empty domain.

The guard specifies that the site a must be unphosphorylated and the value of the counter less or equal to 2. Applying the rule modifies the value of three variables. The site a gets phosphorylated. This is a non-invertible modification that sets the variable χ_a° to the constant value 0 and the variable $\chi_a^\bullet$ to the constant value 1. The counter x is incremented. This is an invertible modification that is encoded by incrementing the value of the variable val_x.

5.3 Generic numerical abstract domain

We are now ready to define a generic numerical abstraction.

Definition 11 (Numerical domain). *A numerical abstract domain is a family* $(\mathcal{A}_A^{\mathcal{N}})_{A \in \Sigma_{ag}}$ *of tuples* $(\mathcal{D}_A^{\mathcal{N}}, \sqsubseteq_A^{\mathcal{N}}, \gamma_A, \sqcup_A^{\mathcal{N}}, \bot_A^{\mathcal{N}}, \top_A^{\mathcal{N}}, g_A^{\mathcal{N}}, forget_A^{\mathcal{N}}, \delta_A^{\mathcal{N}}, \nabla_A^{\mathcal{N}})$ *that satisfy the following conditions, for every agent type* $A \in \Sigma_{ag}$*:*

1. the pair $(\mathcal{D}^{\mathcal{N}}_A, \sqsubseteq^{\mathcal{N}}_A)$ is a pre-order;
2. the component $\gamma^{\mathcal{N}}_A : \mathcal{D}^{\mathcal{N}}_A \to \wp(\mathbb{Z}^{\mathcal{V}ar_A})$ is a monotonic function;
3. the component $\sqcup^{\mathcal{N}}_A : \wp_{finite}(\mathcal{D}^{\mathcal{N}}_A) \to \mathcal{D}^{\mathcal{N}}_A$ is an operator such that $\forall X^\sharp \in \wp_{finite}(\mathcal{D}^{\mathcal{N}}_A)$, $\forall \rho^\sharp \in X^\sharp$, $\rho^\sharp \sqsubseteq \sqcup(X^\sharp)$;
4. the component $\perp^{\mathcal{N}}_A$ is an element in the set $\mathcal{D}^{\mathcal{N}}_A$ such that $\gamma^{\mathcal{N}}_A(\perp^{\mathcal{N}}_A) = \emptyset$;
5. the component $\top^{\mathcal{N}}_A$ is an element in the set $\mathcal{D}^{\mathcal{N}}_A$ such that $\gamma^{\mathcal{N}}_A(\top^{\mathcal{N}}_A) = \mathbb{Z}^{\mathcal{V}ar_A}$;
6. the component $g^{\mathcal{N}}_A$ is a function mapping each pair $(g, \rho^\sharp)$ where g is a guard and $\rho^\sharp$ an abstract property in $\mathcal{D}^{\mathcal{N}}_A$ to an abstract element in $\mathcal{D}^{\mathcal{N}}_A$ such that the set $\gamma^{\mathcal{N}}_A(g^{\mathcal{N}}_A(g, \rho^\sharp))$ contains at least each function $\rho \in \gamma^{\mathcal{N}}_A(\rho^\sharp)$ that verifies the condition $\rho(v) \in \rho^\sharp(v)$ for every variable $v \in \mathcal{V}ar_A$;
7. the component $forget^{\mathcal{N}}_A$ maps each pair $(V, \rho^\sharp) \in \wp(\mathcal{V}ar_A) \times \mathcal{D}^{\mathcal{N}}_A$ to an abstract property $forget^{\mathcal{N}}_A(V, \rho^\sharp) \in \mathcal{D}^{\mathcal{N}}_A$, the concretization $\gamma(forget^{\mathcal{N}}_A(V, \rho^\sharp))$ of which contains at least each function $\rho \in \mathbb{Z}^{\mathcal{V}ar_A}$ such that there exists a function $\rho' \in \gamma^{\mathcal{N}}_A(\rho^\sharp)$ satisfying $\rho(v) = \rho'(v)$ for each variable $v \in \mathcal{V}ar_A \setminus V$;
8. the component $\delta^{\mathcal{N}}_A$ maps each pair $(t, \rho^\sharp) \in \mathbb{Z}^{\mathcal{V}ar_A} \times \mathcal{D}^{\mathcal{N}}_A$ to an abstract property $\delta^{\mathcal{N}}_A(t, \rho^\sharp) \in \mathcal{D}^{\mathcal{N}}_A$, such for each function $\rho \in \gamma^{\mathcal{N}}_A(\rho^\sharp)$, the function mapping each variable $v \in \mathcal{V}ar_A$ to the value $\rho(v) + t(v)$ belongs to the set $\gamma^{\mathcal{N}}_A(\delta^{\mathcal{N}}_A(t, \rho^\sharp))$;
9. the component $\nabla^{\mathcal{N}}$ is a widening operator. satisfies both following properties:

   (a) $\forall \rho^\sharp_1,\ \rho^\sharp_2 \in \mathcal{D}^{\mathcal{N}}_A$, $\rho^\sharp_1 \sqsubseteq^{\mathcal{N}}_A \rho^\sharp_1 \nabla^{\mathcal{N}} \rho^\sharp_2$ and $\rho^\sharp_2 \sqsubseteq^{\mathcal{N}}_A \rho^\sharp_1 \nabla^{\mathcal{N}} \rho^\sharp_2$,
   (b) $\forall (\rho^\sharp_n)_{n \in \mathbb{N}} \in \left(\mathcal{D}^{\mathcal{N}}_A\right)^{\mathbb{N}}$, the sequence $(\rho^\nabla_n)_{n\in\mathbb{N}}$ that is defined as $\rho^\nabla_0 = \rho^\sharp_0$ and $\rho^\nabla_{n+1} = \rho^\nabla_n \nabla^{\mathcal{N}} \rho^\sharp_{n+1}$ for every integer $n \in \mathbb{N}$, is ultimately stationary.

5.4 Numerical abstraction

The following theorem explains how to build an abstraction (as defined in Sect. 4) from a numerical abstract domain. We introduce an operator $\uparrow$ to extend the domain of functions with default values. Given a function f, a value v and a super-set X of the domain of f, we write $\uparrow^v_X f$ the extension of the function f that maps each element $x \in X \setminus Dom\ (f)$ to the value v. We also write set_A for the function mapping pairs $(f, X^\sharp)$ where f is a partial function from the set $\mathcal{V}ar_A$ into the set of the convex parts of $\mathbb{Z}$ and $X^\sharp$ an abstract property in $\mathcal{D}^{\mathcal{N}}_A$, to the abstract property: $g^{\mathcal{N}}_A(\uparrow^{\mathbb{Z}}_{\mathcal{V}ar_A} f, forget^{\mathcal{N}}_A(dom(f), X^\sharp))$. The function set_A forgets all the information about the variables in the domain of the function f, and reassign their range to their image by f in the abstract.

Theorem 4. *Let $(\mathcal{D}^{\mathcal{N}}_A, \sqsubseteq^{\mathcal{N}}_A, \gamma_A, \sqcup^{\mathcal{N}}_A, \perp^{\mathcal{N}}_A, \top^{\mathcal{N}}_A, g^{\mathcal{N}}_A, forget^{\mathcal{N}}_A, \delta^{\mathcal{N}}_A, \nabla^{\mathcal{N}}_A)_{A \in \Sigma_{ag}}$ be a numerical abstract domain. The tuple $(\mathcal{Q}^\sharp, \sqsubseteq, \gamma, \sqcup, \perp, \mathcal{I}^\sharp, t^\sharp, \nabla)$ that is defined by:*

1. the component $\mathcal{Q}^\sharp$ is the set of the functions mapping each agent type $A \in \Sigma_{ag}$ to an abstract property in the set $\mathcal{D}^{\mathcal{N}}_A$;
2. the component γ is the function mapping a function $X^\sharp \in \mathcal{Q}^\sharp$, to the set of the fully specified site-graph G such that for each agent $n \in \mathcal{A}_G$, we have $guard_G(n) \in \gamma_{type_G(n)}(X^\sharp(type_G(n)))$;
3. the components $\sqsubseteq$, $\sqcup$, $\perp$ are defined component-wise;

4. *the component $\mathcal{I}^\sharp$ maps each agent type $A \in \Sigma_{ag}$ to the abstract property $\sqcup_A^{\mathcal{N}}\{g_A^{\mathcal{N}}(guard_{G_0}(n), \top_A^{\mathcal{N}}) \mid n \in \mathcal{A}_{G_0}\}$;*
5. *the component $t^\sharp$ is a function mapping each pair $(X^\sharp, r) \in \mathcal{Q}^\sharp \times \mathcal{R}$ (we write $r : L \twoheadleftarrow\!\supseteq\!\hookrightarrow D \xhookrightarrow{\$} R$) to the element $\perp_A^{\mathcal{N}}$ whenever there exists an agent n in $\mathcal{A}_L$ such that $g_A^{\mathcal{N}}(pre_r(n), X^\sharp(type_L(n))) = \perp_A^{\mathcal{N}}$, and, otherwise, to the function mapping each agent type A to the numerical property:*
$$\sqcup_A^{\mathcal{N}}(\{X^\sharp(A)\} \cup fresh(r, A) \cup updated(r, A, X^\sharp)),$$
with:
   - *$fresh(r, A)$ the set of the numerical abstract elements $g_A^{\mathcal{N}}(new_r n, \top_A^{\mathcal{N}})$ for every $n \in dom(new_r)$ such that $type_R(n) = A$;*
   - *and $updated(r, A, X^\sharp)$ the set of the elements:*
   $$set_A(not\text{-}invert_r(n), \delta_A^{\mathcal{N}}(\uparrow_A^0 invert_r(n), g_A^{\mathcal{N}}(pre_r(n), X^\sharp(A))))$$
   for each agent $n \in \mathcal{A}_D$ with $type_D(n) = A$;

is a generic abstraction.

Most of the constructions of the abstraction are standard. The expression $g_A^{\mathcal{N}}(pre_r(n), X^\sharp(type_L(n)))$ refines the abstract information about the potential configurations of the n-th agent in the left hand side of the rule, by taking into account its precondition. Whenever a bottom element is obtained for at least one agent, the precondition of the rule is not satisfiable and the rule is discarded at this moment of the iteration. Otherwise, the information about each agent is updated. Starting from the result of the refinement of the abstract element by the precondition, the function $\delta_A^{\mathcal{N}}$ applies the invertible transformations $\uparrow_A^0$ $invert_r(n)$ (the function $\uparrow_A^0$ extends the domain of the function $invert_r(n)$ by specifying that the variables not in the domain of this function remain unchanged), and the function set_A applies non invertible one $not\text{-}invert_r(n)$.

The domain of intervals [8] and the one of affine relationships [32] provide all the primitives requested by Def. 11. We use a product of them, when all primitives are defined pair-wise, except the guards which refine its output by using the algorithm that is described in [23]. We use widening with thresholds [2] for intervals so as to avoid infinite bounds when possible. This way we obtain a domain, where all operations are cubic with respect to the number of variables.

This is a very good trade-off. A relational domain is required. Other relational domain are either too imprecise [37], or to costly [13], or both [27,38].

5.5 Benchmarks

We run our analysis on the family of models of Sec. 1 for n ranging between 1 and 25. For each version of the model, the protein is made of n phosphorylation sites and a counter. Moreover, our analysis always discover that the counter ranges between 0 and n. CPU time is plot in Fig. 10.

6 Conclusion

When potential protein transformations depend on the number of sites satisfying a given property, counters offer a convenient way to describe generic mechanisms

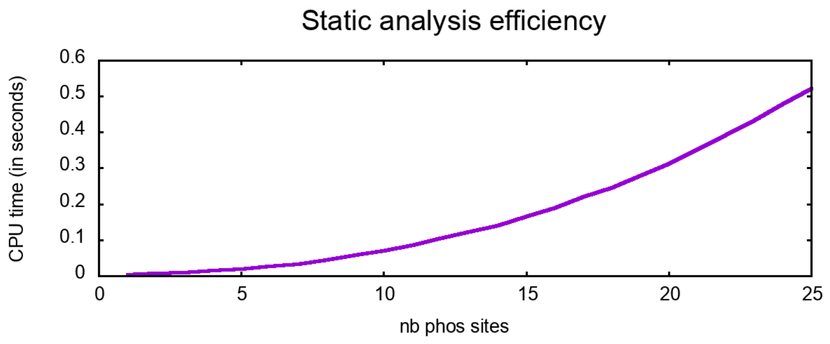


Fig. 10. Efficiency of the static analysis for the example in Sec. 1 with n ranging between 1 and 25. Every analysis has successfully computed the exact range of the counter. The analysis has been performed on a MacBook Pro on a 2.8 GHz intel Core i7, 16 Go of RAM, running under macOS High Sierra version 10.13.6.

while avoiding the explosion in the number of rules. We have extended the semantics of Kappa to deal with counters. We have proposed some encodings to remove counters while preserving the performance of the Kappa simulator. In particular, graphs remain rigid and the number of rules remain the same. Then, we have introduced a static analysis to bound the range of counters.

It is quite common to find proteins with more than 40 phosphorylation sites. Without our contributions, the modeler has no choice but to assume these proteins to be active only when all their sites are phosphorylated. This is a harsh simplification. Modeling simplifications are usually done not only because detailed knowledge is missing, but also because corresponding models cannot be described, executed, or analyzed efficiently. Yet these simplifications are done without any clue of their impact on the behavior of the systems. By providing ways of describing and handling some complex details, we offer the modelers the means to incorporate these details and to test empirically their impact.

Our framework is fully integrated within the Kappa modeling platform which is open-source and usable online (`https://kappalanguage.org`). It is worth noting that we have taken two radically different approaches to deal with counters in simulation and in static analysis. Encodings are good for simulation, but they tend to obfuscate the properties of interest, hence damaging drastically the capability of the static analysis to infer useful properties about them. The extension of the categorical semantics provides a parsimonious definition of causality between computation steps, as well as means to reason symbolically on the behavior of the number of occurrences of patterns. For further works, we will extend existing decision procedures [15,14] that compute minimal causal traces to cope with counters. It is very likely that a third approach will be required. We suggest to use the traces obtained by simulation, then translate the counters in these traces thanks to equivalent sites, and apply existing decision procedures the traces that will be obtained this way.

References

1. Nicolas Behr, Vincent Danos, and Ilias Garnier. Stochastic mechanics of graph rewriting. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, (LICS'16)*, pages 46–55, New York, NY, USA, 2016. ACM.
2. Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. A static analyzer for large safety-critical software. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI'03)*, pages 196–207, San Diego, California, USA, June 7–14 2003. ACM Press.
3. Michael L. Blinov, James R. Faeder, Byron Goldstein, and William S. Hlavacek. Bionetgen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics*, 20(17):3289–3291, 2004.
4. Luca Bortolussi, Rocco De Nicola, Vashti Galpin, Stephen Gilmore, Jane Hillston, Diego Latella, Michele Loreti, and Mieke Massink. CARMA: collective adaptive resource-sharing markovian agents. In Nathalie Bertrand and Mirco Tribastone, editors, *Proceedings of the Thirteenth Workshop on Quantitative Aspects of Programming Languages and Systems (QAPL'15)*, volume 194 of *EPTCS*, pages 16–31, London, UK, 2015.
5. Pierre Boutillier, Ferdinanda Camporesi, Jean Coquet, Jérôme Feret, Kim Quyên Lý, Nathalie Théret, and Pierre Vignet. Kasa: A static analyzer for kappa. In Milan Ceska and David Safránek, editors, *Computational Methods in Systems Biology - 16th International Conference, CMSB 2018, Brno, Czech Republic, September 12-14, 2018, Proceedings*, volume 11095 of *Lecture Notes in Computer Science*, pages 285–291. Springer, 2018.
6. Pierre Boutillier, Thomas Ehrhard, and Jean Krivine. Incremental update for graph rewriting. In Hongseok Yang, editor, *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10201 of *Lecture Notes in Computer Science*, pages 201–228. Springer, 2017.
7. Patrick Cousot. Semantic foundations of program analysis. In *Program Flow Analysis: Theory and Applications*, chapter 10. Prentice-Hall, Inc., 1981.
8. Patrick Cousot and Radhia Cousot. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming*, pages 106–130. Dunod, Paris, France, 1976.
9. Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. POPL'77*. ACM Press, 1977.
10. Patrick Cousot and Radhia Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4), 1992.
11. Patrick Cousot and Radhia Cousot. Comparing the Galois connection and widening-narrowing approaches to abstract interpretation. In *Proc. PLILP'92*, LNCS. Springer-Verlag, 1992.
12. Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. Combination of abstractions in the ASTRÉE static analyzer. In M. Okada and I. Satoh, editors, *Eleventh Annual Asian Computing Science Conference (ASIAN'06)*, Tokyo, Japan, LNCS 4435, December 6–8 2007. Springer, Berlin.

13. Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In Alfred V. Aho, Stephen N. Zilles, and Thomas G. Szymanski, editors, *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages, Tucson, Arizona, USA, January 1978*, pages 84–96. ACM Press, 1978.
14. Vincent Danos, Jérôme Feret, Walter Fontana, Russell Harmer, Jonathan Hayman, Jean Krivine, Christopher D. Thompson-Walsh, and Glynn Winskel. Graphs, rewriting and pathway reconstruction for rule-based models. In Deepak D'Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, volume 18 of *LIPIcs*, pages 276–288. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.
15. Vincent Danos, Jérôme Feret, Walter Fontana, Russell Harmer, and Jean Krivine. Rule-based modelling of cellular signalling. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *Proceedings of the 18th International Conference on Concurrency Theory (CONCUR'07)*, volume 4703 of *Lecture Notes in Computer Science*, pages 17–41, Lisbon, Portugal, September, 2007 2007. Springer.
16. Vincent Danos, Jérôme Feret, Walter Fontana, Russell Harmer, and Jean Krivine. Abstracting the differential semantics of rule-based models: exact and automated model reduction. In Jean-Pierre Jouannaud, editor, *Proceedings of the Twenty-Fifth Annual IEEE Symposium on Logic in Computer Science, LICS '2010*, volume 0, pages 362–381, Edinburgh, UK, 11–14 July 2010. IEEE Computer Society.
17. Vincent Danos, Jérôme Feret, Walter Fontana, and Jean Krivine. Scalable simulation of cellular signaling networks. In Zhong Shao, editor, *Proceedgins of the 5th Asian Symposium on Programming Languages and Systems, 5th Asian Symposium (APLAS'07)*, volume 4807 of *Lecture Notes in Computer Science*, pages 139–157. Springer, 2007.
18. Vincent Danos and Cosimo Laneve. Formal molecular biology. *Theoretical Computer Science*, 325(1), 2004.
19. Giorgio Delzanno, Cinzia Di Giusto, Maurizio Gabbrielli, Cosimo Laneve, and Gianluigi Zavattaro. The *kappa*-lattice: Decidability boundaries for qualitative analysis in biological languages. In Pierpaolo Degano and Roberto Gorrieri, editors, *Computational Methods in Systems Biology, 7th International Conference, CMSB 2009, Bologna, Italy, August 31-September 1, 2009. Proceedings*, volume 5688 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 2009.
20. Edsger W. Dijkstra. Over de sequentialiteit van procesbeschrijvingen. circulated privately, 1962 or 1963.
21. Edsger Wybe Dijkstra. Cooperating sequential processes. Technical Report EWD-123, 1965.
22. H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, and A. Corradini. Algebraic approaches to graph transformation. part ii: Single pushout approach and comparison with double pushout approach. In *Handbook of Graph Grammars and Computing by Graph Transformation*, pages 247–312. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.
23. Jérôme Feret. Occurrence counting analysis for the pi-calculus. *Electronic Notes in Theoretical Computer Science*, 39.(2), 2001. Workshop on GEometry and Topology in COncurrency theory, PennState, USA, August 21, 2000.
24. Jérôme Feret. Abstract interpretation of mobile systems. *J. Log. Algebr. Program.*, 63(1):59–130, 2005.
25. Jérôme Feret. An algebraic approach for inferring and using symmetries in rule-based models. *Electr. Notes Theor. Comput. Sci.*, 316:45–65, 2015.

26. Jérôme Feret, Vincent Danos, Jean Krivine, Russ Harmer, and Walter Fontana. Internal coarse-graining of molecular systems. *PNAS*, 2009.
27. René Rydhof Hansen, Jacob Grydholt Jensen, Flemming Nielson, and Hanne Riis Nielson. Abstract interpretation of mobile ambients. In *Proc. SAS'99*, LNCS. Springer-Verlag, 1999.
28. Tobias Helms, Tom Warnke, Carsten Maus, and Adelinde M. Uhrmacher. Semantics and efficient simulation algorithms of an expressive multilevel modeling language. *ACM Trans. Model. Comput. Simul.*, 27(2):8:1–8:25, 2017.
29. Ricardo Honorato-Zimmer, Andrew J. Millar, Gordon D. Plotkin, and Argyris Zardilis. Chromar, a language of parameterised agents. *Theoretical Computer Science*, 2017.
30. Kurt Jensen. *Coloured Petri Nets (2Nd Ed.): Basic Concepts, Analysis Methods and Practical Use: Volume 1*. Springer-Verlag, Berlin, Heidelberg, 1996.
31. Mathias John, Cédric Lhoussaine, Joachim Niehren, and Cristian Versari. Biochemical reaction rules with constraints. In Gilles Barthe, editor, *Programming Languages and Systems - 20th European Symposium on Programming, ESOP 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, volume 6602 of *Lecture Notes in Computer Science*, pages 338–357. Springer, 2011.
32. Michael Karr. Affine relationships among variables of a program. *Acta Informatica*, 1976.
33. Stephen Cole Kleene. *Introduction to mathematics*. ISHI Press International, 1952.
34. Peter Kreyßig. *Chemical Organisation Theory Beyond Classical Models: Discrete Dynamics and Rule-based Models*. PhD thesis, Friedrich-Schiller-University Jena, 2015.
35. Fei Liu, Mary Ann Blätke, Monika Heiner, and Ming Yang. Modelling and simulating reaction-diffusion systems using coloured petri nets. *Comp. in Bio. and Med.*, 53:297–308, 2014.
36. Antoine Miné. A new numerical abstract domain based on difference-bound matrices. In *Proc. of the Second Symposium on Programs as Data Objects (PADO II)*, volume 2053 of *Lecture Notes in Computer Science (LNCS)*, pages 155–172. Springer, May 2001.
37. Antoine Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation (HOSC)*, 19(1):31–100, 2006.
38. Hanne Riis Nielson and Flemming Nielson. Shape analysis for mobile ambients. In *Proc. POPL'00*. ACM Press, 2000.
39. Tatjana Petrov, Jerome Feret, and Heinz Koeppl. Reconstructing species-based dynamics from reduced stochastic rule-based models. In C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, editors, *Winter Simulation Conference*. WSC, 2012.
40. Michael W. Sneddon, James R. Faeder, and Thierry Emonet. Efficient modeling, simulation and coarse-graining of biological complexity with nfsim. *Nat Methods*, 8(2), 2011.
41. Donald Stewart. Spatial biomodelling, 2010. Master thesis, School of Informatics, University of Edinburgh.
42. Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2), 1955.
43. Glynn Winskel. Event structures. In *Advances in Petri Nets*, volume 255 of *LNCS*, 1987.