Programs as models: Execution

Class Notes SB200@Harvard (Fall 2008)

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

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

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

[†]Corresponding author

Version 1.0 [October 24, 2008]

Contents

1 Modeling in Kappa **3**

1.1 "Hello World" in Kappa: Do/Undo loops 3

2 Execution **8**

2.1 Stochastic rate constants and the rescale factor 9

2.2 Gillespie for rules 10

Appendix **15**

A Stochastic rate constants **15**

B Gillespie's method, briefly **16**

C Time advance with void events **17**

1 Modeling in Kappa

In this section we represent a few small and well-known molecular signaling motifs in Kappa. There is nothing here that could not be done with any other approach, especially differential equations. Indeed, we hope this invites and facilitates comparison. The main purpose is to introduce the fundamentals of modeling in Kappa, including a quick overview of the stochastic simulator. While Kappa scales naturally to systems of staggering complexity and size, the practice of modeling large systems requires a substantial theoretical and algorithmic infrastructure, which we have begun to develop and whose detailed characterization constitutes the objective of a different manuscript.

1.1 "Hello World" in Kappa: Do/Undo loops

A model of a molecular interaction system consists of a list of Kappa rules with rate parameters and an initial mixture of agents or complexes in defined states. As an example, consider an ubiquitous pattern in cellular signaling: A futile, energy-consuming cycle, in which one enzyme modifies a substrate protein and another enzyme undoes the modification, returning the substrate to its previous state. The kinetics of such a cycle has been studied computationally by Goldbeter and Koshland in 1981 for the case of phosphorylation and dephosphorylation of a target protein by a kinase and a phosphatase, respectively. Do-Undo motifs may involve other post-translational modifications, such as the methylation and demethylation of Tar receptors in bacterial chemotaxis; the GDP/GTP exchange and subsequent GTP hydrolysis characteristic of small G proteins; or the fission and fusion of a complex, as they occur with heterotrimeric G proteins. Multiple Do-Undo patterns can be deployed "in parallel", as in multisite phosphorylation, or "in series", as in MAP kinase cascades. The variations on this basic motif are so manifold that one is inclined to think of it as the"hydrogen atom" of molecular control.

In the present example we have the case of Ras in mind. Ras is a small G protein that plays a role in EGFR signaling. A guanine nucleotide-exchange factor, or GEF, swaps the target-bound GDP with a GTP and a GTPase-activating protein, or GAP, hydrolyzes the target-bound GTP into GDP. The loop is schematized in Figure 1A and B, where G stands for the small G-protein, G° denotes the GDP-bound inactive state, and $G^\bullet$ refers to the GTP-bound active state. The GEF (the "Do"-enzyme) and the GAP (the "Undo"-enzyme) are usually different proteins, but here we think of them as different states of the same protein S ("S" as in signal). Indeed, as long as the states don't interconvert, they behave kinetically as if they were two different proteins. Figure 1B depicts an interaction diagram, where solid arrows express substrate transformations, and dotted arrows that point at solid arrows indicate catalytic activity. Oftentimes, a modeler infers a set of differential equations by translating such a diagram into reactions. In the case of Figure 1B (and given that the reactions involve protein-protein interactions) one might translate the first step in the activation of G° as a reversible binding between G° and $S^\bullet$: $G^\circ + S^\bullet \leftrightarrow G^\circ.S^\bullet$ (where we informally use the dot to suggest a complex). From this we recover mass action terms in the rate of change for $G^\circ.S^\bullet$, free G°, and free $S^\bullet$. Yet, when translating such a reaction into Kappa, one realizes that certain mechanistic aspects have remained tacit and probably never attracted the modeler's attention. For example, to express the binding between G° and $S^\bullet$ in Kappa, we must define a binding site on each agent. This detail is pretty innocuous, if it wasn't for potential binding conflicts. For example, let us add a binding site named s to agent G: $G(s)$;

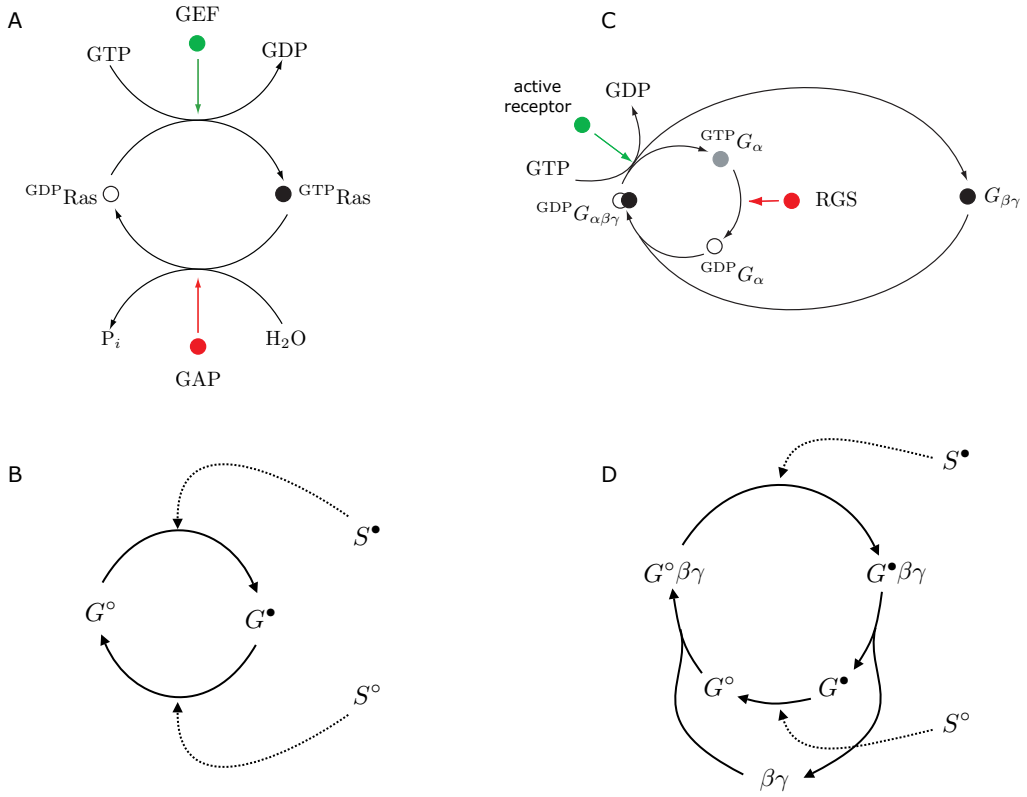

Figure 1: Do/Undo loops. Two versions of Do/Undo loops are shown. A: The case of a small G protein like Ras. A GDP is exchanged for GTP by a guanine nucleotide-exchange factor (GEF), effectively re-tagging Ras. The hydrolysis of GTP to GDP undoes what the swap accomplished. The phosphorylation and dephosphorylation of a protein by a kinase and phosphatase, respectively, constitutes an analogous logical and kinetic case. B: A skeletal form of the reaction system A, as modeled in this section. C: The activation and inactivation of a heterotrimeric G-protein constitutes another Do/Undo scenario. The Do step (catalyzed, for example, by an activated receptor) is the fission of the heterotrimeric $G_{\alpha\beta\gamma}$ complex into G_α and $G_{\beta\gamma}$, while the Undo step (catalyzed by a "regulator of G-protein signaling") consists in the fusion of G_α and $G_{\beta\gamma}$ to reconstitute $G_{\alpha\beta\gamma}$. The fission of a complex is a way of "forking" a signal into two concurrent threads and synchronizing them again. D: The skeletal form of C, as contemplated in this section.

and let us do the same for agent S: $S(s)$. Assigning the same name to both sites has no formal significance, since the scope of a site name is confined to the agent owning that site. A binding site, however, leads to the question whether the GEF ($S^\bullet$) and the GAP (S°) share the same binding site s on G, whether the binding site is the same site that is also modified, and whether a binding event tests both the modification state of G and S.

In Kappa, a site's internal state can change independently of its binding state. The modeler has to judge whether it is mechanistically sensible for one protein to be bound at site s, while another (or the same) protein changes the state at s (say the phosphorylation state). To keep our example simple, the binding site s of G shall also be the carrier of internal state, which we denote by a

subscripted *gtp* or *gdp*. (In a more detailed representation, one might imagine a separate GTP-agent that gets bound to $S^\bullet$ and then transferred and bound to G.) A set of Kappa rules expressing the two modification legs of the loop as a Michaelis-Menten scheme might thus read:

Model 1: "Do/Undo" (vanilla)

$$S^\bullet + G^\circ \rightleftharpoons S^\bullet.G^\circ \longrightarrow S^\bullet + G^\bullet$$
$$S^\circ + G^\bullet \rightleftharpoons S^\circ.G^\bullet \longrightarrow S^\circ + G^\circ$$

`R1, R1r`	$S(s_y), G(s_{gdp})$	$\longleftrightarrow$	$S(s^1_y), G(s^1_{gdp})$	@	γ_1, γ_{1r}
`R2, R2r`	$S(s_n), G(s_{gtp})$	$\longleftrightarrow$	$S(s^1_n), G(s^1_{gtp})$	@	γ_2, γ_{2r}
`R3`	$G(s^1_{gdp}), S(s^1_y)$	$\longrightarrow$	$G(s_{gtp}), S(s_y)$	@	γ_3
`R4`	$G(s^1_{gtp}), S(s^1_n)$	$\longrightarrow$	$G(s_{gdp}), S(s_n)$	@	γ_4

In the leftmost column, we assign an arbitrary name to each rule. The first row has two names listed, because we notated the rule as reversible, using the double arrow $\longrightarrow$. The first rule is therefore really two rules, expressing a binding and an unbinding action. On the right of each rule we notate its rate constant. Since the first row consists of two rules, we specify two rate constants — one for the rule reading from left to right and the other for the backward rule, right to left. Likewise for the second row. R1/R1r and R2/R2r thus represent the binding/unbinding steps of both modification legs. Note that $S(s_y)$, which represents $S^\bullet$, only binds G when the internal state of s is *gdp*, in agreement with the diagram 1B showing $S^\bullet$ to only act on G°. We say that rule $R1$ *tests* both the internal state of s (it has to be in the *gdp* state) and its binding state (s has to be free). We say that the *action* of $R1$ is a binding, which is an elementary action. Everything on the left hand side of a rule is a condition that must be satisfied for the rule to apply. The action, in contrast, is the *difference* between the right and the left. Obviously, by mentioning agents S and G, $R1$ also tests for the existence of S and G. The execution of a rule creates an *event* that involves particular instances of agents in the reaction mixture. Rule $R3$ expresses the change of internal state of G (the hydrolysis of GTP) by $S(s_y)$ *and* the simultaneous dissociation of G from $S(s_y)$. $R3$ is therefore not an elementary rule, as its action is a composite of two atomic actions. The same holds, *mutatis mutandis*, for rule $R4$. Note that the vanilla version is a case in which every rule expresses one reaction, because all left hand sides completely specify the state of the entire interface for each type of agent. Version vanilla has as many rules as differential equations needed to represent the system in the traditional way. In this case, the only difference between the rule-based approach and ODEs is that the former explicitly represents agents with an internal structure, thus forcing mechanistic choices that remain at best implicit when representing the system with differential equations.

Before discussing the execution (simulation) of this model, we briefly consider alternative mechanisms to highlight a few aspects of modeling in Kappa. Take, for example, the Do/Undo model, version 2:

Model 2: "Do/Undo" (conflict)

$$S^\bullet + G^\circ \rightleftharpoons S^\bullet.G^\circ \longrightarrow S^\bullet.G^\bullet \rightleftharpoons S^\bullet + G^\bullet$$
$$S^\circ + G^\bullet \rightleftharpoons S^\circ.G^\bullet \longrightarrow S^\circ.G^\circ \rightleftharpoons S^\circ + G^\circ$$

R5	$S(s), G(s)$	$\longrightarrow$	$S(s^1), G(s^1)$	@	γ_5
R6	$S(s^1_y), G(s^1_{gdp})$	$\longrightarrow$	$S(s^1_y), G(s^1_{gtp})$	@	γ_6
R7	$S(s^1_n), G(s^1_{gtp})$	$\longrightarrow$	$S(s^1_n), G(s^1_{gdp})$	@	γ_7
R8	$S(s^-)$	$\longrightarrow$	$S(s)$	@	γ_8

As in version 1, both Do and Undo enzymes, $S(s_y)$ and $S(s_n)$, bind to the same site s of G. However, in version 1, the binding was contingent on the state of that site. The different forms of S were never in each other's way, as $S(s_y)$ could only bind the GDP-bound form of G, $G(s_{gdp})$, and $S(s_n)$ only the GTP-bound form $G(s_{gtp})$. In version 2, S binds regardless of the internal state of s at G, which leads to a competition between $S(s_y)$ and $S(s_n)$ for G. We could have directly modified rules R1 and R2 of version 1 as

R5′	$S(s_y), G(s)$	$\longrightarrow$	$S(s^1_y), G(s^1)$	@	γ'_5
R5″	$S(s_n), G(s)$	$\longrightarrow$	$S(s^1_n), G(s^1)$	@	γ''_5

but R5′ and R5″ can be abstracted into R5, as they constitute an exhaustive refinement of R5. Such abstraction, however, would only be warranted if there was no reason to distinguish between R5′ and R5″ on the basis of rate constants, as we are assuming here. A further change in version 2 is the atomicity of the modifcation steps, rules R6 and R7. Unlike for rules R3 and R4, a separate unbinding event has to occur to dissociate S from G. Rule R8 states, in somewhat succinct form, that whatever is bound to S at site s can dissociate. This is an example of a contextual rule, which does not mention all the agents affected by its action: the partner that becomes unbound is determined at runtime. (The present case is simple enough that we know it can only be a G, and that's why we write it this way.)

R8 illustrates an important point. In most biological systems, the contexts permissive of a dissociation event are simpler than those that must be satisfied for a binding event. Consider version 3, where we replaced rule R5 of version 2 with rules R1 and R2 from version 1. This model thus agrees with version 1 in the state-sensitive binding of S to G. Like version 2, it uncouples the modification event from the unbinding event. It is good practice to write separate forward and backward reactions, because the contexts in which they occur differ. A single dissociation rule thus makes several association rules reversible. For example, in version 3, each form of S binds to a specific form of G, but any form of S can dissociate from any form of G.

Model 3: "Do/Undo" (full)

$$S^\bullet + G^\circ \rightleftharpoons S^\bullet.G^\circ \longrightarrow S^\bullet.G^\bullet \longrightarrow S^\bullet + G^\bullet$$
$$S^\circ + G^\bullet \rightleftharpoons S^\circ.G^\bullet \longrightarrow S^\circ.G^\circ \longrightarrow S^\circ + G^\circ$$

R1	$S(s_y), G(s_{gdp})$	$\longrightarrow$	$S(s_y^1), G(s_{gdp}^1)$	@	γ_1
R2	$S(s_n), G(s_{gtp})$	$\longrightarrow$	$S(s_n^1), G(s_{gtp}^1)$	@	γ_2
R6	$G(s_{gdp}^1), S(s_y^1)$	$\longrightarrow$	$G(s_{gtp}^1), S(s_y^1)$	@	γ_6
R7	$G(s_{gtp}^1), S(s_n^1)$	$\longrightarrow$	$G(s_{gdp}^1), S(s_n^1)$	@	γ_7
R8	$S(s^-)$	$\longrightarrow$	$S(s)$	@	γ_8

Version 4 represents the case of a heterotrimeric G-protein, as depicted in Figure 1D.

Model 4: "Do/Undo" (heterotrimeric G protein)

$$S^\bullet + G^\circ.\beta\gamma \rightleftharpoons S^\bullet.G^\circ.\beta\gamma \longrightarrow S^\bullet.G^\bullet.\beta\gamma \longrightarrow S^\bullet + G^\bullet.\beta\gamma$$
$$S^\circ + G^\bullet \rightleftharpoons S^\circ.G^\bullet \longrightarrow S^\circ.G^\circ \longrightarrow S^\circ + G^\circ$$
$$G^\bullet.\beta\gamma \longrightarrow G^\bullet + \beta\gamma$$
$$G^\circ + \beta\gamma \longrightarrow G^\circ.\beta\gamma$$

R1′	$S(s_y), G(s_{gdp}, b^2), BeGa(b^2)$	$\longrightarrow$	$S(s_y^1), G(s_{gdp}^1, b^2), BeGa(b^2)$	@	γ_1'
R2′	$S(s_n), G(s_{gtp}, b)$	$\longrightarrow$	$S(s_n^1), G(s_{gtp}^1, b)$	@	γ_2'
R6	$G(s_{gdp}^1), S(s_y^1)$	$\longrightarrow$	$G(s_{gtp}^1), S(s_y^1)$	@	γ_6
R7	$G(s_{gtp}^1), S(s_n^1)$	$\longrightarrow$	$G(s_{gdp}^1), S(s_n^1)$	@	γ_7
R8	$S(s^-)$	$\longrightarrow$	$S(s)$	@	γ_8
R9	$G(s_{gtp}^?, b^1), BeGa(b^1)$	$\longrightarrow$	$G(s_{gtp}^?, b), BeGa(b)$	@	γ_9
R10	$G(s_{gdp}, b), BeGa(b)$	$\longrightarrow$	$G(s_{gdp}, b^1), BeGa(b^1)$	@	γ_{10}

The model highlights in red the differences between the small G-protein case, version 3, and the heterotrimeric case. Rule R1 becomes R1', as it is now executed in the context of G (which stands for the α subunit in Figure 1C) bound to the $\beta\gamma$-subunit, here named $BeGa$. Thus, we have to extend the interface of G with a binding site for $BeGa$. Rule R2' mentions the free $BeGa$-binding site b of G. This represents the fact that the Undo agent (the "regulator of G-protein signaling"), $S(s_n)$, can only bind the free α-unit. If we had omitted the mention of b, as in rule R2, the Undo agent could *in addition* bind the heterotrimer prior to its fission (but after its activation) and thus potentially inactivate the α-subunit preventing further fission. Whether this can or cannot be the case requires the modeler to evaluate available empirical data or hypothesize. The easy modifiability of rules makes exploring many distinct hypotheses, or extending and overhauling models, an almost effortless process. Finally, we need two new rules to represent the fission and fusion of the α and $\beta\gamma$ subunits. In rule R9 we condition fission on the activation of the α subunit (site s of agent G must be in the gtp state), but we do not care whether the Do agent (typically an activated receptor), $S(s_y)$, is still bound to the heterotrimer. This is expressed by the indeterminate (?) binding state of site s at G. Rule R10, however, requires the α subunit to be free (and deactivated) to recombine with the $\beta\gamma$ subunit.

We next turn to the execution of a model.

2 Execution

So far we focused on collections of rules and rate parameters (left unspecified) that represent a system of molecular interactions. While this constitutes a minimal notion of model, it isn't executable yet. An executable model requires the definition of an initial condition specifying which complexes are present in how many copies at the beginning of a simulation. Here is Model 3 with an executable dressing:

Model 3: "Do/Undo" (executable)

R1	$S(s_y), G(s_{gdp})$	$\longrightarrow$	$S(s^1_y), G(s^1_{gdp})$	@	$\gamma_1 = 6.3 \cdot 10^{-7}\ s^{-1} molecule^{-1}$
R2	$S(s_n), G(s_{gtp})$	$\longrightarrow$	$S(s^1_n), G(s^1_{gtp})$	@	$\gamma_2 = 6.3 \cdot 10^{-7}\ s^{-1} molecule^{-1}$
R6	$G(s^1_{gdp}), S(s^1_y)$	$\longrightarrow$	$G(s^1_{gtp}), S(s^1_y)$	@	$\gamma_6 = 0.4\ s^{-1}$
R7	$G(s^1_{gtp}), S(s^1_n)$	$\longrightarrow$	$G(s^1_{gdp}), S(s^1_n)$	@	$\gamma_7 = 0.4\ s^{-1}$
R8	$S(s^-)$	$\longrightarrow$	$S(s)$	@	$\gamma_8 = 0.23\ s^{-1}$

init	10^6	$G(s_{gdp})$
init	11000	$S(s_y)$
init	89000	$S(s_n)$

Table 1: Table.

The "init" rows specify the initial mixture: a million substrate agents in the gdp-form, $G(s_{gdp})$, and a total of 10^5 enzyme agents, 11 thousand of which act as Do-agents and 89 thousand as Undo-agents.

Since a Kappa model is not translated into differential equations, for the reasons elaborated in the introductory sections, its simulation is event-based or stochastic. When a particular set of agents in the mixture is matched by the pattern specified on the left of a rule, the rule is applied by converting that instance of reactants into products as discussed in the Kappa Language Basics. In this way, application after application, a list of rules drive changes in the composition of the mixture. To generate a traditional dynamical trace of time versus the number of instances of a molecular species, we need a scheduler that decides which rule should be applied when. This scheduler is a continuous time Monte-Carlo algorithm, such as the one proposed by Doob and Gillespie (see also Appendix B), suitably adapted to efficiently operate on a set of rules. In a very concrete sense, a model is a computer program, whose instructions (the rules) continually operate on the program's state (the mixture). Unlike more familiar programs, in which instructions are executed in a deterministic (though state-dependent) sequence, the rules of a model are executed probabilistically, like in a concurrent computer program.

We shall describe our adaptations of Gillespie's algorithm to rules after a few comments on rate

constants in stochastic systems.

2.1 Stochastic rate constants and the rescale factor

In Appendix A we recall that stochastic rate constants, while related to deterministic rate constants, must be understood as probabilities per unit time. To convert a deterministic bimolecular constant into a probability, requires knowing the volume V to which the reactants of a system are confined. A bimolecular stochastic rate constant γ, expressed in $s^{-1}molecule^{-1}$, is related to its deterministic counterpart k, expressed in $s^{-1}M^{-1}$, as

$$\gamma = \frac{k}{A\,V}, \tag{1}$$

where A is Avogadro's number, $A \approx 6.022\ 10^{23}$. In general, for a reaction of molecularity n, $\gamma = k/(A\,V)^{(n-1)}$. While Kappa rules can be of any molecularity, we will confine ourselves to mono- and bimolecular cases.

The parameters of Model 3 reflect plausible ballpark figures. For the deterministic bimolecular binding constants we chose a value of $k_1 = k_2 = 8.5\ 10^5\ (s\ M)^{-1}$. Assuming a mammalian cell volume of $V = 2.25\ 10^{-12}l$, equation (1) yields the stochastic rate constants $\gamma_1 = \gamma_2 = 6.3\ 10^{-7}\ (s\ molecule)^{-1}$. The monomolecular dissociation constants have no volume dependence, $k_8 = \gamma_8 = 0.23\ s^{-1}$, and, likewise, the catalytic rate constants, $k_6 = k_7 = \gamma_6 = \gamma_7 = 0.4$. This combines to a Michaelis constant for the GEF enzyme, $S(s_y)$, of $K_m = (k_8 + k_6)/k_1 = 0.74\ 10^{-6}M$. Michaelis constants for signaling proteins, such as kinases, are roughly in the 0.1-10μM range (but we have not surveyed the literature systematically). A $K_m = 0.74\ 10^{-6}M$ corresponds to a stochastic $\mathcal{K}_m = K_m\ A\ V = 10^6\ molecules$. Ditto for the GAP, $S(s_n)$, as we are defining a kinetically symmetric Do/Undo loop. To play with the system at various saturation levels, we will need to cover a few orders of magnitude around 10^6 substrate agents G. If we wish to compare simulations of Model 3 with deterministic computations based on a Michaelis-Menten approximation, we also need a substantial excess of G over S. With just one order of magnitude difference, Model 3 would be barely satisfying that criterion.

The drawback with stochastic simulations is the substantial amount of computing time required to simulate large systems of components whose copy numbers differ by several orders of magnitude. In some situations it helps to rescale a system. Consider two systems that share the same bimolecular deterministic rate constants, but differ in size: $V_2 = \lambda V_1$, $\lambda \neq 0$. Their stochastic rate constants thus must be related like

$$\gamma_1 = \frac{k}{A\,V_1} \qquad \gamma_2 = \frac{k}{A\,\lambda V_1} \qquad \Rightarrow \qquad \gamma_2 = \frac{1}{\lambda}\gamma_1. \tag{2}$$

Since we rescale the volume, we also must rescale the particle numbers of any component A, $n_{A,2} = \lambda n_{A,1}$, to keep the particle densities the same. In sum, rescaling a system with a factor λ means (i) multiply all particle numbers with λ, (ii) multiply the volume with λ and, consequently, (iii) divide all bimolecular rate constants by λ. Typically we want to reduce computing time and therefore $\lambda < 1$. While downscaling the system retains the same average (deterministic) behavior, the fluctuations will increase. For example, we might consider executing Model 3 with a rescale factor $\lambda = 10^{-3}$, which puts the number of G and S agents at 1000 and 100, respectively. Since

the S agents are divided into two forms, each ends up with only several tens of instances. If we wish to study the system at saturation, we need to increase the original number of G agents to, say, 10^8, to be well above the stochastic $\mathcal{K}_m$ of 10^6. Yet, we cannot go much lower than a rescale factor of 10^{-3} without drastically increasing the fluctuations resulting from low copy numbers in the two S-forms. Thus, we need to simulate about $100,000$ agents to see saturation effects.

2.2 Gillespie for rules

Given a set of rules with rate constants and an initial mixture of agents, we use a continuous-time Monte-Carlo method, specifically an adaptation of Gillespie's algorithm, to simulate sample trajectories consistent with stochastic chemical kinetics as described by a master equation.

We define some terminology first. By a *reaction* we mean an event in which specific individual molecules react in a particular way. By definition, a reaction can fire only once, since it uses up the particular reactant combination. A given mixture may contain many reactant combinations capable of reacting according to the same scheme. We call that scheme a reaction channel. Thus, when we write down a chemical "reaction" on paper, we really write down a reaction channel - a type of interaction. (We continue using the terms reaction and reaction channel interchangeably, when we have no need to differentiate) It should be clear by now, that a reaction channel differs from a rule. A reaction channel fully specifies all agents involved as reactants or products, while a rule mentions patterns of agents, and thus typically expands into more than one reaction channel.

In Appendix B we briefly review and derive Gillespie's scheme. In the more familiar context of reactions, the setup phase of Gillespie's algorithm consists in determining the activity α_r of a reaction channel. As defined in equation (11) (Appendix A), α_r is r's rate constant γ_r times the number of distinct reactant combinations that can react along r in a given mixture. The total activity of the system $\lambda = \sum_r \alpha_r$ is the parameter of an exponential probability distribution for the time to the next reaction event (i.e., on average, every λ^{-1} time units a reaction occurs).

Given a mixture M at time t and a list of all possible reaction channels $r = 1, \ldots, n$, the core loop of the original Gillespie algorithm proceeds in 4 steps. (i) Select a reaction channel r with probability α_r/λ. (ii) Draw a δt (time to the next reaction) exponentially distributed with parameter λ, and advance the simulated wall-clock time t to $t + \delta t$. (iii) Execute a reaction according to the selected channel r by removing an instance of each r-reactant from M and adding an instance of each r-product to M. (iv) Update all α_r, $r = 1, \ldots, n$, and the total activity λ to reflect the changed composition of M. Repeat.

The implementation of Gillespie's algorithm in the context of rules requires some adaptation, if the goal is to formulate a scalable procedure whose time cost per update step is insensitive to the number of potential molecular species implied by the rules and the size of the system (the number of agents populating a mixture). The former means we must avoid ever expanding the list of rules into a list of reaction channels, since the number of channels n might be astronomical or outright infinite, as in the case of polymer formation. The latter means that we chose to represent each agent in M individually. For example, if M contains 1 million instances of the mTORC2 complex $mTOR(sin^1, r^2, l), SIN1(m^1, r^3, akt), Rictor(t^2, sin^3)$, we explicitly represent 1 million $mTOR$ agents, 1 million $SIN1$ agents, and 1 million *Rictor* agents. In other words, we literally work with the (large) Kappa expression representing M.

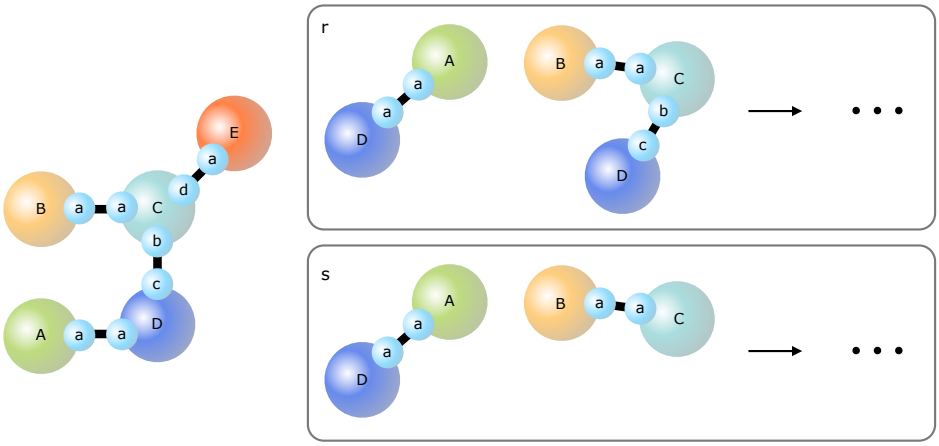

Figure 2: Clashes.

In analogy to the traditional reaction-based procedure, the setup phase consists in computing the activity α_r of each rule $r = 1, \ldots, n$ as the number of distinct ways in which r can be applied to the mixture M. This is done by identifying all matching instances between the left hand side pattern r_{lhs} of a rule r and M, as detailed in Kappa Language Basics and its appendix on matching. The number c of connected patterns that make up r_{lhs} is the arity of r. To match r_{lhs} to M means finding a disjoint match for each connected pattern $P_i^{(r)}, i = 1, \ldots, c$ of r. A match for a given $P_i^{(r)}$ is attempted by first identifying an agent in M whose name occurs in $P_i^{(r)}$. Given such an anchor, the rigidity property of site-graphs, then assures that we can decide an attempted match at a computational cost linear in the size of $P_i^{(r)}$. Although the matchings for each $P_i^{(r)}, i = 1, \ldots, c$ must not overlap, they may well occur in the same complex in M. An example is shown in Figure 2. The two components on the left of the binary rule r each mention an agent of type D. As in chemical notation, each mention of an agent type means a separate resource. Both components must therefore match different instances of D in the mixture M. Thus, rule r does not match the complex specified on the left of Figure 2, whereas rule s does. (However, rule r matches two *distinct* instances of the complex on the left.)

By scanning over M we can determine all possible matchings of rule r in M. A symmetry in r_{lhs}, however, gives rise to physically indistinguishable matchings that involve identical agents in M. For example, case A in Figure 3 might represent the dissociation of a dimerized receptor A. Regardless of differences in the context of the two matched agents A in M, r_{lhs} can match them in two ways by assigning either #1 or #2 to the first A in M (the second A is then assigned #2 or #1, respectively). In general, two matchings are equivalent when they are related by an automorphism of r_{lhs}. The left hand side in Figure 3A has one automorphism in addition to the identity, and the number of matchings must, therefore, be divided by two. It will be useful (in a moment) to keep in mind automorphisms within a connected component of r_{lhs} and across connected components of r. For example, case B in Figure 3 has an automorphismn across components, and case C has one automorphisms within each component and one across both

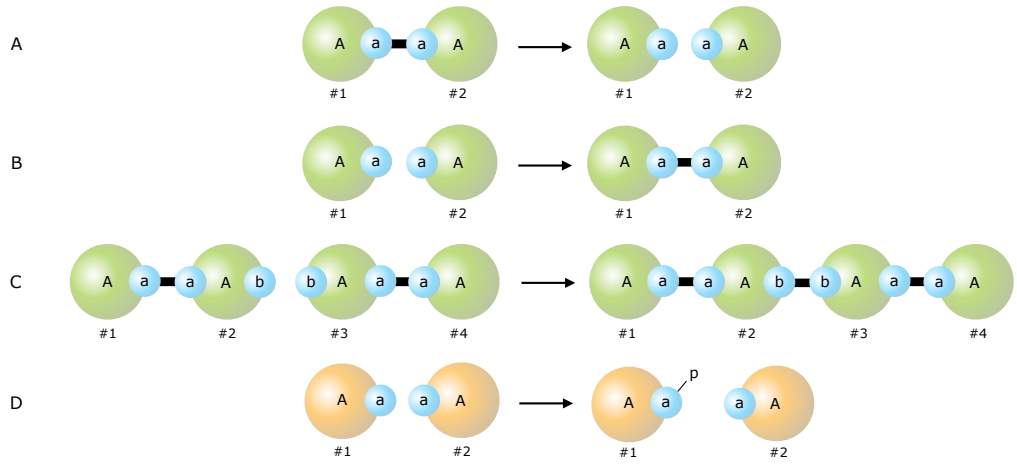

Figure 3: Isos and autos.

components.

Let the set of these nonequivalent matchings for r be $\mathcal{R}$. The activity of rule r then is

$$\alpha_r = \gamma_r |\mathcal{R}|, \tag{3}$$

where γ_r is the rate constant and $|\mathcal{R}|$ the size of set $\mathcal{R}$. Once the α_r have been established for all $r = 1, \ldots, n$, we can proceed essentially as in the traditional reaction-based version of the Gillespie algorithm. As indicated previously, it is very time-efficient to keep an explicit agent-based representation of M. In this way, we avoid having to check whether the products generated at every execution of a rule are already in M (to update their counter in a counter-based implementation). While such checks are not too onerous for many standard Kappa models, they are, in principle, NP-complete, as they require deciding whether two graphs are isomorphic. An agent-based representation forgoes such a check, or delays it to a later moment when, for example, M is displayed. In addition, an agent-based representation enables us to explicitly keep pointers from agent combinations in M back to the rule r that matches them. In this way we operate with an explicit representation of all possible next events in M. This is like keeping a list of all reactions that can fire next - which is quite different from a list of all possible reactions.

This approach suffers from rules of arity $c > 1$. Even if rules were at most binary - which in practice they are (but they don't have to be) - the number of possible next events (and pointers associated with them) scales with the power of the arity. Rather than establishing and storing matchings between r_{lhs} as a whole and the mixture M, we establish matchings between the connected component patterns of r_{lhs}, $P_i^{(r)}, i = 1, \ldots, c$, individually. Consider, thus, the sets of nonequivalent matchings $\mathcal{P}_i^{(r)}, i = 1, \ldots, c$; $r = 1, \ldots, n$. The activity of rule r is then given as

$$\alpha'_r = \gamma_r \frac{1}{auto(r_{lhs})} \prod_{i=1}^{c} |\mathcal{P}_i^{(r)}|, \tag{4}$$

where $auto(r_{lhs})$ is the number of automorphisms (including the identity) *between* components of r_{lhs}, as the automorphisms within a component have already been taken care of by definition of the $\mathcal{P}_i^{(r)}$ as sets of nonequivalent matchings. This approach is vastly more efficient, but it comes with a trade-off. Because pattern components are matched individually to M and the numbers of their matchings are multiplied, as if there was a mass-action law for patterns, we no longer catch overlaps between pattern matchings. In this approach, rule r in Figure 2 is assigned an activity α'_r, equation (4), potentially larger than its true activity α_r, equation (3), in M. The product in equation (4) now includes a combination in which, for example, both the first and second pattern component of r in Figure 2 match the same instance of the complex depicted on the left in Figure 2. We refer to this as a "clash", because it violates the condition stipulated by r_{lhs} that two distinct instances of D be involved in the action. We resolve clashes at runtime. When matchings for pattern components of a rule are chosen that clash (a condition that is easy to detect), the attempted event is rejected, a new rule selection is made (using the α'_r), but the simulated wall-clock time is advanced as if the rejected event was a productive event. In appendix C we prove that this approach fires rules and advances time with the correct probability distribution. It might help to consider the example in Figure 3B, assuming that the patterns on the left hand side of the rule actually represent the complete interface of agent A, in which case the rule expresses a reaction channel. In the traditional reaction-based algorithm, the activity of the channel would be $\gamma_B(1/2)n_A(n_A-1)$, with γ_B the rate constant of the rule in Figure 3B and n_A the number of particles of A in M. Our procedure computes the activity as $\gamma_B(1/2)n_A^2$. The factor $1/2$ is the automorphism count between pattern components, $auto(B_{lhs})$, and n_A is the number of matchings for each pattern component $A(a)$. Clearly, we overestimate the activity by a factor of $\gamma_B(1/2)n_A$ which comes from attempted "collisions" between a particle of A and itself, which are precisely the clashes from selecting twice the same matching for both pattern components of the rule. In this example, such clashes occur with probability $1/n$, and are rejected. A clash is an instance of a "void event" - an event that leaves the system unchanged, but causes the simulated wall-clock time to update. Void events can also arise from rejecting attempted reactions that do not meet certain constraints specified by the modeler, such as polymer formation (see Kappa Language Basics) or limiting complexes to a maximum size. The snag with this approach is that a rule might have a nonzero overestimated activity, while its true activity might have gone to zero (in our example: when only one particle of A remains). In that case, our algorithm will loop forever, as it keeps proposing clashes which it then rejects. This is referred to in the computer science literature of concurrent systems as a deadlock. We declare a deadlock when a long enough string of rejected events has occurred. "Long enough" is a parameter set by the modeler.

We have sketched a strategy that assigns an (overestimated) activity to each rule in a model. Recall that we also keep track of all matchings between the patterns of a rule and the mixture M. We call this the matching map. It turns out to be useful to also keep for each site x in M a set of pairs, with each pair containing (i) a matching ϕ of which that site is a part of and (ii) the pattern component $P_i^(r)$ for which ϕ constitutes a match. We call this the extension (lift) map of x. Once this information is established in the setup phase, the core loop of the Gillespie algorithm for rules is analogous to the usual scheme:

i Select a rule r with probability α'_r/λ', where $\lambda' = \sum_r \alpha_r$.

ii Draw a δt exponentially distributed with parameter λ', and advance the simulated

wall-clock time t to $t + \delta t$.

iii Select at random one matching for each pattern component of the rule r chosen in step (i). If the matching combination is a clash, continue with step (i).

iv On the matching chosen in step (iii), execute the action represented by the rule r selected in step (i).

v Exploit the pointer databases to perform a

(a) Negative update: For each site that has been modified by the action of r, remove all pattern matchings that involve that site and decrease all affected α'_r accordingly. Likewise, update the extension maps of all sites that participate in the matchings that were removed. These are very fast operations typically limited to a few pointers.

(b) Positive update: For each agent who had a site modified by the action of r, identify all the new matchings for the $P_i^{(r)}$ in the model. This is done by using each modified agent, in turn, as an anchor for a match (see rigidity property). Increase the α'_r accordingly and update the extension maps for all sites that participate in the new matchings.

vi Update the overall system activity λ'.

The actual implementation of the positive update step exploits a map of causal relationships between rules (which are a static property of the model that can be determined ahead of a simulation) to only attempt matchings for those rules that can possibly be affected by the action of r. We detail this aspect in a subsequent manuscript.

Appendix

A Stochastic rate constants

In a chemical reaction R of the kind

$$A + B \quad \stackrel{k}{\rightarrow} \quad \text{something}$$

the rate constant k fixes the proportionality between the velocity of the reaction v_R (or reaction rate) and the product of the reactant concentrations:

$$v_R = k[A][B] \tag{5}$$

The form of (5) factors the rate of R into (i) the density dependence of encounters between A and B (kinetic law of mass action) and (ii) the "rest". This rest contains dependencies such as the collision cross-section of molecules, their speed and kinetic energy (and thus molecular mass). The separation in (5) is based on the idea that the "rest" is constant over time, unlike the concentrations of the reactants, which change as the reaction proceeds. Underlying the constancy of k is the idea of a single *mechanism* – a particular causal path from reactants to products. Indeed, a time-dependent "constant" k would be a telltale sign that the mechanism of the reaction is concentration dependent, or rather, that there are multiple mechanisms whose relative significance shifts with the concentration of the reactants.

The rate constant k is an average property characterizing a reactive encounter between one molecule of A and one molecule of B. For hard spheres with no activation barrier, i.e. a fictitious situation in which every collision is reactive, k is straightforward to compute from gas theory. Essentially, a particle of type A with an average velocity v relative to a particle of type B and cross-section σ sweeps out a collision volume $\delta V = v\sigma$ per unit time. This volume δV contributes, along with the density of B (which is recognized in the density dependent portion of v_R) to the rate at which a particle of A reacts. Thus k has a dimension of volume/time. The deterministic approach is characterized by expressing everything in terms of densities, but a particle-based, stochastic approach hinges on probabilities. The theoretical calculation of k is no different than in the deterministic case. However, k needs to become a *probability* (per unit time), say γ. A (dimensionless) probability is obtained by dividing the sweep volume δV by the total volume V that confines the reactants. This requires that we know the volume V. Thus, for a bimolecular reaction, the stochastic rate constant γ indicates a probability (per unit time) and is volume dependent:

$$\gamma = \frac{k}{V}. \tag{6}$$

This generalizes to reactions of molecularity n: $\gamma = k/V^{(n-1)}$. The dimension of a bimolecular γ is $s^{-1}mol^{-1}$, if k is given in $s^{-1}\ M^{-1}$. A monomolecular stochastic reaction rate constant has no volume dependence, since it does not require a collision. It is like radioactive decay. We often want to express γ in "per molecule" rather than "per mol". Naturally, this only changes the numerical value, not the dimension (both are numbers of molecules). Thus

$$\gamma = \frac{k}{A\,V}, \tag{7}$$

where A is Avogadro's number, $A \approx 6\ 10^{23}$.

The Michaelis constant of an enzymatic reaction is $K_m = (k_{-1} + k_2)/k_1$, with k_{-1} and k_1 denoting the off and on constants of the enzyme-substrate interaction, and k_2 denoting the catalytic rate constant. The stochastic $\mathcal{K}_m$ has monomolecular rate constants in the numerator and a bimolecular rate constant in the denominator from which it inherits the dependence on volume:

$$\mathcal{K}_m = K_m\ A\ V. \tag{8}$$

Consider a single pair of molecules A and B and a stochastic rate constant γ, as described above. Assuming the occurrence of a reaction event is a memoryless process and that no reaction event has occurred at time $t_0 = 0$, the probability density of an event happening at time t is given by an exponential density:

$$p(t) = \gamma e^{-\gamma t}. \tag{9}$$

If the system contains n_A and n_B particles of type A and B, respectively, we can form $n_A n_B$ possible events, each of which has the same exponential likelihood (9). Using exactly the same argument leading up to equation (15) in section B (where we provide a brief summary of the Gillespie method), it is straightforward to show that the probability density for a reaction event to occur at time t in a mixture containing n_A molecules of A and n_B molecules of B is given by

$$p(t) = \gamma n_A n_B\ e^{-\gamma n_A n_B t}. \tag{10}$$

Let us call

$$\alpha = \gamma n_A n_B \tag{11}$$

the "activity" of the reaction. This activity corresponds to the instantaneous reaction velocity in the deterministic setting – except that here n_A and n_B are particle numbers, not concentrations, and γ depends on the volume in which reactions take place. Generally, the activity factors into a stochastic rate constant and a count of the number of ways in which a reaction event can occur. For a reaction of the sort $2A \longrightarrow$ *something* that number is $(1/2)n_A(n_A - 1)$.

B Gillespie's method, briefly

Consider a system of n reaction types, i, $i = 1, \ldots, n$, with respective activities α_i. Let the system start at $t_0 = 0$ and let t_j be the time at which the first reaction event of type $j = 1, \ldots, n$ occurs. We want to know the probability density that the first reaction in the system is of type i, i.e. that reaction i occurs before all others.

We first compute the probability density that reaction i it occurs before all others, given that it occurs at time t_i:

$$p(t_i < t_1)p(t_i < t_2)\cdots p(t_i < t_{i-1})p(t_i < t_{i+1})\cdots p(t_i < t_n) = \prod_{j\neq i} p(t_i < t_j) = \prod_{j\neq i} p(t_j > t_i). \tag{12}$$

Using equations (10) and (11):

$$\prod_{j\neq i} p(t_j > t_i) = \prod_{j\neq i} \int_{t_i}^{\infty} \alpha_j e^{-\alpha_j t} dt = \prod_{j\neq i} e^{-\alpha_j t_i} = e^{-\lambda t_i} e^{\alpha_i t_i}, \tag{13}$$

with $\lambda = \sum_{j=1}^{n} \alpha_j$. λ is the "total activity" of the system.

From this we compute the probability density $p(i,t)$ that reaction i is the first reaction and that it occurs *at* time t:

$$p(i,t) = e^{-\lambda t} e^{\alpha_i t} \alpha_i e^{-\alpha_i t} = \alpha_i e^{-\lambda t}. \tag{14}$$

$p(i,t)$ is the probability needed to simulate the system. Given an initial condition, one calculates the α_i, $i = 1, \ldots, n$, and generates a random pair (i,t) whose joint distribution is $p(i,t)$. Once the chosen reaction has been carried out, the α_i are recomputed (they depend on the molecule numbers, which are changed by reactions), the simulated wall-clock is advanced by t time units, and the process is repeated.

Here is a convenient way to choose i and t. We first determine a time t' at which some reaction occurs and in a second step we decide which reaction among the n possible ones $i = 1, \ldots, n$ is the one to occur at t'. The probability density $p(t)$ that *some* reaction is the first to occur at time t is

$$p(t) = \sum_{i}^{n} \alpha_i e^{-\lambda t} = \lambda e^{-\lambda t}. \tag{15}$$

Suppose we draw a t' distributed according to $p(t)$. Given t', the probability density that reaction i is the first one and occurs at t' is $\alpha_i \exp(-\lambda t')$, which is $p(i,t')$. We therefore draw the first reaction i according to the distribution $p(i)$ given by

$$p(i|t') = \frac{p(i,t')}{\sum_j p(j,t')} = \frac{\alpha_i e^{-\lambda t'}}{\sum_j \alpha_j e^{-\lambda t'}} = \frac{\alpha_i}{\lambda}. \tag{16}$$

Notice that $p(i,t) = p(t)p(i|t)$.

C Time advance with void events

We think of an event as a happening that changes the state of a system. A "void event", then, is defined as an attempted event that is not executed, leaving the state of the system unchanged. The point of the continuous time Monte-Carlo scheme developed by Gillespie is to avoid void events. By contrast, in a pedestrian Monte-Carlo approach one chooses any two molecules and then looks up whether there is a scheme according to which they can react. The pedestrian approach is computationally wasteful, since many attempted interactions will typically be rejected for lack of an appropriate reaction scheme. Such interactions are "void events". Void events, however, are not always avoidable. In Kappa Language Basics, we pointed out that Kappa is a context-free language, capturing events whose conditions for occurrence can be decided locally – at a computational cost that is insensitive to the size of the system. While demanding local

decidability makes physical sense, global constraints are oftentimes useful and even necessary. As discussed in Kappa Language Basics, the absence of geometric features in Kappa creates a problem when we wish to avoid unreasonable polymerization in favor of the intramolecular cyclization of complexes. One way of solving the probem is to introduce constraints, or filters, that reject an attempted but non-compliant interaction. This leads to void events.

In section 2.2, void events arise as a consequence of an implementation trade-off. Rather than generating all matchings between a rule with $m > 1$ connected patterns on the left hand side and a mixture, we generate matches between each component and the mixture separately. While the reduction in storage requirements and setup costs is dramatic, the technique occasionally leads to reactions that violate Kappa semantics – what we characterized as "clashes" in section 2.2. These attempted reactions are identified and must be rejected, thus leading to void events.

The question arises how to properly account for the simulated-time advance, equation (15), in a system throwing up void events. The answer is simple: proceed exactly as in the case without void events, but advance the simulated-time even when drawing a void event (though no state change is executed). This procedure was adopted by Faeder et al. We proposed an equivalent version in our earlier paper that detailed the first implementation of Gillespie's algorithm for general rule-based systems.

At any given time t and for any given rule i (and a source of void events), a mixture defines a set of potential next events, with total activity (as defined in section A) α_i and a set of potential next void events, with total activity $\alpha'_i - \alpha_i$. Thus, α'_i is the total *apparent activity* of rule i, $\alpha'_i \geq \alpha_i$. Likewise, $\sum_i \alpha'_i = \lambda'$ (adopting notation from section 2.2) is the total apparent activity of the system, and $\sum_i \alpha_i = \lambda$ is the true activity of the system. Thus, the likelihood of rejecting an attempted reaction i is given by

$$\frac{\alpha'_i - \alpha_i}{\alpha'_i} \equiv \epsilon_i. \tag{17}$$

Figure 4 depicts the kind of scenario we need to consider. In the absence of void events ($\lambda' = \lambda$) the time interval Δt to the next event is distributed according to equation (15):

$$p(\Delta t) = \lambda e^{-\lambda \Delta t}. \tag{18}$$

The presence of void events causes the apparent system activity to be higher and the time intervals between attempted events (void or executed) to be smaller, like the δt in Figure 4. In fact, by the same arguments that led to equation (15), the δt are distributed as

$$p(\delta t) = \lambda' e^{-\lambda' \delta t}. \tag{19}$$

The example in Figure 4 consists in a run of 5 void events (nulls) followed by an event (checkmark). According to Faeder et al.'s procedure, each δt advances the simulated time. If we have a run of k void events preceding an event, the random variable Δt is the sum of $k + 1$ random variables δts distributed according to equation (19). Because all attempted events prior to the checkmark are void, the state of the system remains unchanged and so does the λ'. It is a textbook case of statistics that the probability density for the sum of $k + 1$ independent

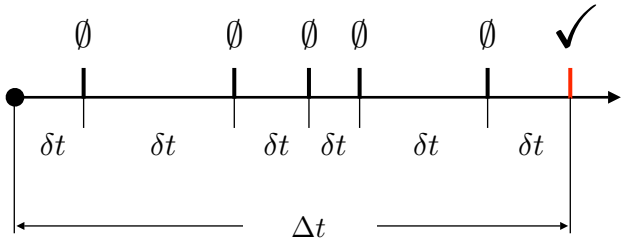

Figure 4: Void events.

exponentially distributed random variables is a Gamma distribution. In our specific case:

$$\Gamma(k+1,\lambda') = \frac{(\lambda' t)^k}{k!}\lambda' e^{-\lambda'\Delta t}. \tag{20}$$

Given that a run of k void events has preceded an event, the time interval Δt to the event is distributed according to $\Gamma(k+1,\lambda')$. The probability p_k that we will observe such a run of k void events is given by

$$p_k = \left(\sum_{i=1}^{n}\frac{\alpha_i'}{\lambda'}\epsilon_i\right)^k\left(1-\sum_{i=1}^{n}\frac{\alpha_i'}{\lambda'}\epsilon_i\right) = \epsilon^k(1-\epsilon), \tag{21}$$

where α_i'/λ' is the probability of drawing an event based on rule i (section 2.2, equation 16), ϵ_i is the probability of rejecting that event (equation 17), and $\epsilon = \sum_i(\alpha_i'/\lambda')\epsilon_i$ is a shorthand for the probability of a void event. Thus, with probability p_k the time advance Δt (which we now refer to as simply t) is drawn from $\Gamma(k+1,\lambda')$. To obtain the probability density $p(t)$ for the time advance t, we need to sum over all possible runs of void events preceding an event:

$$\begin{aligned} p(t) &= \sum_{k=0}^{\infty}\epsilon^k(1-\epsilon)\Gamma(k+1,\lambda') = \sum_{k=0}^{\infty}\epsilon^k(1-\epsilon)\frac{\lambda'}{k!}(\lambda' t)^k e^{-\lambda' t} = (1-\epsilon)\lambda' e^{-\lambda' t}\sum_{k=0}^{\infty}\frac{(\epsilon\lambda' t)^k}{k!} \\ &= (1-\epsilon)\lambda' e^{-\lambda' t}e^{\epsilon\lambda' t} = (1-\epsilon)\lambda' e^{-(1-\epsilon)\lambda' t} = \lambda e^{-\lambda t}, \end{aligned} \tag{22}$$

which yields the correct time-advance distribution for events, equation (15) of section 2.2.

Instead of advancing time at each void event, we can count the length of a run of void events and use equation (20). The two methods are equivalent, because the Gamma distribution is the distribution resulting from a sum of independent exponentially distributed random variables. However, the former method is operationally simpler to implement.