A Trace Query Language for Rule-based Models

Jonathan Laurent[1], Hector F. Medina-Abarca[2], Pierre Boutillier[2], Jean Yang[1] and Walter Fontana[2]

[1] Carnegie Mellon University
[2] Harvard Medical School

Abstract. In this paper, we introduce a unified approach for querying simulation traces of rule-based models about the statistical behavior of individual agents. In our approach, a query consists in a trace pattern along with an expression that depends on the variables captured by this pattern. On a given trace, it evaluates to the multiset of all values of the expression for every possible matching of the pattern. We illustrate our proposed query language on a simple example, and then discuss its semantics and implementation for the Kappa language. Finally, we provide a detailed use case where we analyze the dynamics of β-catenin degradation in Wnt signaling from an agent-centric perspective.

Keywords: Rule-based modeling · Query Language · Kappa.

1 Introduction

Rule-based modeling languages such as Kappa [2] and BioNetGen [6] can be used to write mechanistic models of complex reaction systems. Models in these languages consist of stochastic graph-rewriting rules that are equipped with rate constants indicating their propensity to apply. Together with an initial mixture graph, these rules constitute a dynamical system that can be simulated using Gillespie's algorithm [5,3,1]. Each run of simulation results in a sequence of transitions that we call a trace.

In practice, simulation traces are often discarded in favor of a limited number of global features, such as the concentration curves of a set of observables. However, a more detailed analysis of their structure and statistical properties can provide useful insights into a system's dynamics. For example, causal analysis methods exist [2,4] that compress a large trace into a minimal subset of events that are necessary and jointly sufficient to replicate an outcome of interest, and then highlight causal influences between those remaining events. Queries about the statistical behavior of individual agents can lead to complementary insights. Examples include (i) measuring the average lifespan of a complex under different conditions, (ii) computing a probability distribution over the states in which a particular type of agent can be when targeted by a given rule, and (iii) estimating how much of a certain kind of substrate getting phosphorylated is due to a particular pathway at different points in time.

In this paper, we propose a unifying language to express queries of this kind, that are concerned with statistical features of groups of molecular events that

are related in specific motifs. These motifs are formalized using a notion of *trace pattern*. Then, evaluating a query comes down to computing the value of an expression for every matching of a pattern into a trace. We give a first illustration of this paradigm on a toy example in section 2. After that, we introduce our query language in section 3 and give it a formal semantics. We then characterize a natural subset of this language for which an efficient evaluation algorithm exists and discuss our implementation for the Kappa language (section 4). Finally, we leverage our query engine to explore aspects of the dynamics of the Wnt signaling pathway in a detailed use case (section 5).

2 A Starting Example

In order to illustrate our Trace Query Language, we introduce a toy Kappa model in Figure 1. It is described using a rule notation that has been introduced in the latest release of the Kappa simulator and which we borrow in our query language. With this notation, a rule is described as a pattern that is annotated with rewriting instructions. The pattern denotes a precondition that is required for a rule to target a collection of agents. Rewriting instructions are specified by arrows that indicate the new state of a site after transformation.

The model of Figure 1 features two types of agents: substrates S and kinases K. Both kinds of agents have two different sites, named x and d. In addition, x-sites can be in two different internal states: *unphosphorylated* and *phosphorylated*. We write those states u and p, respectively. Rule b expresses the fact that a substrate and a kinase with free d-sites can bind at rate λ_b. Rules u and u^* express the fact that the breaking of the resulting complex happens at different rates, depending on the phosphorylation state of the kinase involved. Finally, rule p expresses the fact that a substrate that is bound to a kinase can get phosphorylated at rate λ_p. In all our examples, we consider initial mixtures featuring free substrates and kinases in smiliar quantity. Substrates are initially unphosphorylated and kinases are present in both phosphorylation states.

By playing with this model a bit, one may notice that the concentration of phosphorylated substrate reaches its maximal value faster when the ratio of phosphorylated kinases is high (given the rules of our model, the latter quantity is invariant during the simulation). This phenomenon cannot be explained by looking at rule p alone. The query provided in (1) can be run to estimate the probability that a substrate is bound to a phosphorylated kinase when it gets phosphorylated:

$$\mathsf{match}\ t : \left\{ S\left(x_{\,u \to p},\ d^{\,1}\right),\ k{:}K\left(d^{\,1}\right) \right\}\ \ \mathsf{return}\ \ \mathsf{int\_state}\left[\bullet t\right]\left(k, \texttt{"x"}\right) \qquad (1)$$

Given a trace, this query matches every transition where a substrate is getting phosphorylated and outputs the phosphorylation state of the attached kinase. The variables t and k denote a transition and an agent, respectively. Moreover, the expression $\mathsf{int\_state}\left[\bullet t\right]\left(k, \texttt{"x"}\right)$ refers to the internal state of the site of agent k with name `"x"` in the mixture preceding transition t.

$$\lambda_u \gg \lambda_{u^*} \approx \lambda_p$$

$$\begin{array}{lcll} b & : & S\left(d^{\bullet\to 1}\right),\ K\left(d^{\bullet\to 1}\right) & @\ \lambda_b \\ u & : & S\left(d^{1\to\bullet}\right),\ K\left(d^{1\to\bullet},\ x_u\right) & @\ \lambda_u \\ u^* & : & S\left(d^{1\to\bullet}\right),\ K\left(d^{1\to\bullet},\ x_p\right) & @\ \lambda_{u^*} \\ p & : & S\left(d^{1},\ x_{u\to p}\right),\ K\left(d^{1}\right) & @\ \lambda_p \end{array}$$

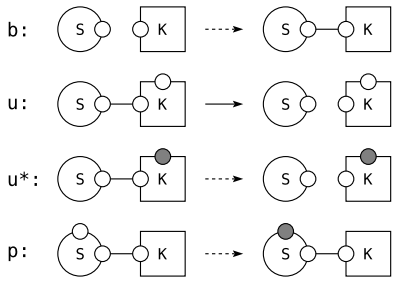

Fig. 1: An Example Kappa Model. On the left, it is described using the *edit notation* introduced in KaSim 4. Numbers in a rule expression correspond to local bond identifiers and • indicates a free site. Sites not mentioned in a rule are left unchanged by it. A graphical representation is provided on the right. Phosphorylated sites are indicated in grey. Dotted and solid arrows indicate *slow* and *fast* reactions, respectively.

Running the previous query, we learn that substrates are much more likely to be phosphorylated by kinases that are phosphorylated themselves, even when such kinases are in minority in the mixture. This leads us to conjecture a causal link between the phosphorylation state of a kinase and its efficiency. After some thoughts, this link can be easily interpreted: because $\lambda_u \gg \lambda_{u^*}$, phosphorylated kinases form more stable complexes with substrates, leaving more chances for a phosphorylation interaction to happen. In fact, the average lifespan of a kinase-substrate complex is exactly $\lambda_{u^*}^{-1}$ when the kinase is phosphorylated and λ_u^{-1} when it is not. We can check these numbers experimentally by running the following query:

$$\begin{array}{ll} \mathsf{match} & b : \left\{\, s{:}S\left(d^{\bullet\to 1}\right),\ K\left(d^{\bullet\to 1},\ x_p\right)\,\right\} \\ \mathsf{and} & \mathsf{first}\ \ u : \left\{\, s{:}S\left(d^{\to\bullet}\right)\,\right\}\ \ \mathsf{after}\ b \\ \mathsf{return} & \mathsf{time}\left[\,u\,\right] - \mathsf{time}\left[\,b\,\right] \end{array} \tag{2}$$

This query outputs a multiset of numbers, whose mean is the average lifespan of a complex formed by a substrate and a phosphorylated kinase. The same quantity can be computed for unphosphorylated kinases by replacing x_p by x_u in the first line of (2). The pattern in this query does not match single transitions but pairs of related transitions (b, u), where b is a binding transition and u the first unbinding transition to target the same substrate.

More generally, a query is defined by a pattern $P[\boldsymbol{t}, \boldsymbol{a}]$ and an expression $E[\boldsymbol{t}, \boldsymbol{a}]$, which feature a shared set $\boldsymbol{t}$ of transition variables and a shared set $\boldsymbol{a}$ of agent variables. The pattern P can be regarded as a predicate that takes as its arguments a trace τ and a *matching* ϕ mapping the variables in $\boldsymbol{t}$ and $\boldsymbol{a}$ to actual transitions and events in τ. The expression E can be regarded as a function that

maps such (τ, ϕ) pairs to values. Then, the query evaluates on a trace τ to the multiset of all values of E, for every matching ϕ that satisfies P in τ.

3 The Core Query Language

In this section, we introduce the extensible core of our proposed query language and give it a formal semantics.

3.1 Meaning and Structure of Queries

As shown in Figure 2, a query Q consists in a pattern P and an expression E. It can be interpreted as a function $[\![Q]\!]$ from traces to multisets[3] of values. The set of allowed values can grow larger as richer expressions are added to the language. Our current implementation defines a value as a tuple of base values and features the following types for base values: bool, int, float, string, agent, agent_set and snapshot.

A pattern P is interpreted as a function $[\![P]\!]$ that maps a trace to a set of matchings. A matching ϕ is defined by two functions ϕ_{t} and ϕ_{a}, which map variable names to transition identifiers and agent identifiers, respectively. We call ϕ_{t} a transition matching and ϕ_{a} an agent matching. Given a trace τ and a matching ϕ, the transition variable v denotes the transition $\tau[\phi_{\mathsf{t}}(v)]$, where $\tau[i]$ is a notation for the i^{th} transition of a trace. In addition, an expression E is interpreted as a function $[\![E]\!]$ that maps a pair of a trace and a matching to a value. The expression language is extensible and is discussed in section 3.3. Its syntax is documented in Figure 3. Then, the semantics of a query can be formally defined as follows:

$$[\![\mathsf{match}\ P\ \mathsf{return}\ E]\!](\tau) \;=\; \wr\, [\![E]\!](\tau, \phi) \mid \phi \in [\![P]\!](\tau) \,\wr.$$

Our language constraints the structure of possible patterns. As shown in Figure 2, a pattern consists in a sequence of clauses, which can take one of three different forms: $(t : T)$, $(\mathsf{first}\ \ t : T\ \mathsf{after}\ t')$ and $(\mathsf{last}\ \ t : T\ \mathsf{before}\ t')$. Here, t and t' are transition variables and T is a *transition pattern*. In all three cases, we say that t is *constrained* by the clause.

3.2 Transition Patterns

A transition pattern can be thought as a predicate that takes as its argument a pair $(\tau, \phi_{\mathsf{a}})$ of a transition and an agent matching. Our current implementation supports specifying transition patterns using KaSim's *edit notation*. Transition patterns defined this way are enclosed within curly brackets. For example, in query (1) of section 2,

$$\left\{\, S\left(x_{\,u \to p},\ d^{\,1}\right),\ k{:}K\left(d^{\,1}\right) \right\}$$

[3] Note that multisets are indicated in Figure 2 using Dijkstra's bag notation, whereas sets are indicated using the standard curly brackets notation.

is true for a transition t and a matching $\phi_{\mathtt{a}}$ if and only if t has the effect of phosphorylating a substrate that is bound to the kinase with identifier $\phi_{\mathtt{a}}(k)$. Formally, a transition pattern T is interpreted as a function $[\![T]\!]$ that maps transitions into sets of agent matchings. Using the predicate terminology, one may say that $\phi_{\mathtt{a}} \in [\![T]\!](t)$ if and only if $(t, \phi_{\mathtt{a}})$ satisfies T.

Our query language can be instantiated with any choice of a language specifying transition patterns. The only requirement is that transition patterns should be decidable efficiently in the following sense. Given a transition pattern T and a transition t, one should be able to efficiently compute whether $[\![T]\!](t)$ is empty and generate an element of it in the case it is not. Our evaluation algorithm relies on this property.

3.3 Expression Language

We show in Figure 3 the syntax of our expression language. An expression can consist of an agent variable, a constant, a parenthesized expression, a binary operation, a function[4] of an expression, a tuple of expressions or a *measure*.

Measures are the basic constructs through which information is extracted from a trace. They come in two different kinds: *state measures* and *event measures*. State measures are used to extract information about the state of the mixture at different points in the trace. They are parametered with *state expressions* that can take the form $\bullet t$ or $t \bullet$, denoting the states before and after transition t, respectively. For example, the `int_state` measure that is used in (1) is a state measure. In addition, event measures are used to extract information that is about a transition itself (in contrast to the states that it connects). They are parametered by transition variables. For example, the `time` measure that is used in (2) is an event measure.

The expression language can be easily extended with new operators, functions, measures and types. In the same way than the language for specifying transition patterns, it should be regarded as a parameter of our query language and not as a rigid component.

4 Evaluating Queries

In this section, we introduce a natural subset of the language described in section 3, for which we provide an efficient implementation. Queries in this subset are said to be *regular*, and they display an interesting rigidity property.

4.1 Rigidity

Intuitively, a pattern is said to be rigid if its matchings are completely determined by the value of a single transition variable.

[4] Note that functions always take a single argument, which can be a tuple.

query	Q	$:=$	match P return E	$[\![Q]\!] \in \mathsf{Trace} \to \wr\, \mathsf{Value} \,\wr$
pattern	P	$:=$	C	$[\![P]\!] \in \mathsf{Trace} \to \{\, \mathsf{Matching} \,\}$
		$\mid$	C and C	
clause	C	$:=$	$t : T$	$[\![C]\!] \in \mathsf{Trace} \to \{\, \mathsf{Matching} \,\}$
		$\mid$	first $t : T$ after t'	
		$\mid$	last $t : T$ before t'	
transition pat.	T	$:=$	$\cdots$	$[\![T]\!] \in \mathsf{Transition} \to \{\, \mathsf{Matching}_{\mathsf{a}} \,\}$
expression	E	$:=$	$\cdots$	$[\![E]\!] \in \mathsf{Trace} \times \mathsf{Matching} \to \mathsf{Value}$

$$[\![\text{match } P \text{ return } E]\!](\tau) \;= \wr\, [\![E]\!](\tau, \phi) \mid \phi \in [\![P]\!](\tau) \,\wr$$

$$[\![C \text{ and } C']\!](\tau) \;=\; [\![C]\!](\tau) \cap [\![C']\!](\tau)$$

$$[\![t : T]\!](\tau) \;=\; \{\, \phi \mid \phi_{\mathsf{a}} \in [\![T]\!](\tau[\phi_{\mathsf{t}}(t)]) \,\}$$

$$[\![\text{first } t : T \text{ after } t']\!](\tau) \;=\; \left\{ \phi \;\middle|\; \begin{array}{l} \phi_{\mathsf{a}} \in [\![T]\!](\tau[\phi_{\mathsf{t}}(t)])\,,\;\; \phi_{\mathsf{t}}(t') < \phi_{\mathsf{t}}(t)\,, \\ \forall i.\; \phi_{\mathsf{t}}(t') < i < \phi_{\mathsf{t}}(t) \implies \phi_{\mathsf{a}} \notin [\![T]\!](\tau[i]) \end{array} \right\}$$

$$[\![\text{last } t : T \text{ before } t']\!](\tau) \;=\; \left\{ \phi \;\middle|\; \begin{array}{l} \phi_{\mathsf{a}} \in [\![T]\!](\tau[\phi_{\mathsf{t}}(t)])\,,\;\; \phi_{\mathsf{t}}(t) < \phi_{\mathsf{t}}(t')\,, \\ \forall i.\; \phi_{\mathsf{t}}(t) < i < \phi_{\mathsf{t}}(t') \implies \phi_{\mathsf{a}} \notin [\![T]\!](\tau[i]) \end{array} \right\}$$

Fig. 2: Syntax and semantics of the Trace Query Language

expression	E	$:=$	$a \mid C \mid (E) \mid E \bowtie E \mid f(E) \mid E, E \mid$ $M_{\mathsf{s}}[S] \mid M_{\mathsf{s}}[S](E) \mid M_{\mathsf{e}}[t] \mid M_{\mathsf{e}}[t](E)$
constant	C	$:=$	$0 \mid 1 \mid \cdots \mid$ `"foo"` $\mid \cdots$
binary operator	$\bowtie$	$:=$	$+ \mid - \mid = \mid < \mid \cdots$
function	f	$:=$	agent_id $\mid$ size $\mid$ count $\mid \cdots$
state measure	M_{s}	$:=$	int_state $\mid$ component $\mid$ snapshot $\mid \cdots$
state expression	S	$:=$	$\bullet t \mid t \bullet$
event measure	M_{e}	$:=$	time $\mid$ rule $\mid \cdots$

(with a an agent variable and t a transition variable)

Fig. 3: Syntax of expressions

Definition 1. *Given a Kappa model, a pattern P is said to be* rigid *if and only if it features a transition variable r called* root variable *such that for any trace τ that is valid in the model, we have*

$$\forall\, \phi, \phi' \in [\![P]\!](\tau),\ \phi_{\,t}(r) = \phi'_{\,t}(r) \implies \phi = \phi'.$$

For example, the pattern P of query (2) is rigid, with root variable b. Indeed, suppose that b is matched to a specific transition t. Then, the agent variable s is determined by t as no more than one substrate can get bound during a single transition given the rules of our model (Figure 1). Finally, u is uniquely determined as the first unbinding event that targets s after b.

An easy consequence of Definition 1 is that the number of matchings of a rigid pattern into a trace is bounded by the size of this trace.

4.2 Regular Queries

Our evaluation algorithm handles a subset of queries whose patterns admit a certain tree structure. For those patterns, rigidity is implied by a weaker notion of *local rigidity*.

Definition 2. *Given a Kappa model, a transition pattern T is said to be* rigid *if and only if for any agent variable a that appears in T and every valid transition t, we have*

$$\forall\, \phi_{\,a}, \phi'_{\,a} \in [\![T]\!],\ \phi_{\,a}(a) = \phi'_{\,a}(a).$$

Intuitively, a transition pattern is rigid if matching it to a transition determines all its agent variables.

Definition 3. *Given a model, a pattern P is said to be* locally rigid *if it features only rigid transition patterns. Then, a transition variable t is said to* determine *an agent variable a if there is a clause of P that constrains*[5] *t and features a.*

For patterns with a particular structure, local rigidity implies rigidity. This structural assumption can be expressed in terms of a pattern's *dependency graph*.

Definition 4. *The* dependency graph *of a pattern P is a graph whose nodes are the transition variables of P and for which there is an edge from t to t' if and only if P contains a clause of the form $(\mathsf{first}\ t' : T\ \mathsf{after}\ t)$ or $(\mathsf{last}\ t' : T\ \mathsf{before}\ t)$.*

We can now define the notion of a regular pattern, and thus of a regular query.

Definition 5. *A pattern is said to be* regular *if the following three conditions hold: (i) it is locally rigid (ii) its dependency graph is a tree (iii) whenever two of its transition variables determine a same agent variable, one of them has to be a descendent of the other in the dependency tree.*

This structure enables an efficient enumeration of the matchings of a regular pattern into a trace. Moreover, the number of these matchings is bounded by the size of the trace, as regular patterns can be proven to be rigid.

[5] As defined in section 3.1.

Proposition 1. *Regular patterns are rigid.*

Finally, regular queries are defined as expected.

Definition 6. *A query is said to be* regular *if its pattern is regular.*

This notion of regularity may appear unintuitive at first, and we agree that its formal definition is somewhat involved. However, we argue that regular queries are exactly those queries that admit a natural operational interpretation. Therefore, experimentalists tend to think in terms of regular queries instinctively.

4.3 Evaluating Regular Queries Efficiently

When designing an algorithm for evaluating trace queries, one has to keep in mind that the corresponding sequence of state mixtures cannot fit in random-access memory all at once, even for small traces. In fact, even the most economic representation of a trace, which is specified by an initial mixture and a sequence of labeled rewriting events, may fail to fit in memory in some cases. Therefore, as often as possible, one should only be allowed to stream such a representation from disk, recomputing intermediate states dynamically and never keeping more than a small number of them in memory at once (two in our case).

Our algorithm for evaluating a regular query proceeds in two steps. First, it streams the trace to compute the set of all matchings of the pattern. Then, it streams the trace a second time to compute the value of the expression for all these matchings. The second step is quite simple to implement. Indeed, once the matchings are known, it is easy to compute the sequence of all measures that need to be performed and order them in increasing order of time. The first step attempts to match the root variable of the pattern to every transition in the trace. For each candidate matching, it uses rigidity to determine all other variables progressively as the trace is streamed, in an order that is determined by the dependency tree and with a minimal amount of caching. Overall, the algorithm runs in linear time with the length of the trace.

4.4 Our Implementation

We provide an implementation of our proposed trace query language, which relies on the algorithm that is mentioned in section 4.3 for evaluating regular queries. Our query engine takes for inputs (i) a file that contains a list of queries written in the same syntax that we use in our examples and (ii) a trace file that has been generated by the Kappa simulator using the `-trace` option. It evaluates all queries at once and generates one output file per query, in comma-separated values (CSV) format.[6]

Queries that are non-regular for structural reasons – i.e. that fail to meet criterion (ii) or (iii) of Definition 5 – are rejected immediately. As there is no

[6] Every line of an output file represents a single value. In our expression language, values are tuples of *base values*. These are separated by commas within a line.

easy static check for local rigidity, queries that do not meet this criterion will be rejected at runtime.

We now introduce a use case in which we leverage our query engine to explore aspects of the dynamics of the Wnt signaling pathway.

5 A Use Case on Wnt Signaling

In this use case, we are focusing on a simplified model of the β-catenin destruction complex from canonical Wnt signaling. This complex is highly conserved in animals, and operates from humans to nematodes to insects to amphibians, regulating the establishment of the dorso-ventral axis. It is also heavily involved in colon cancer.

A source of complexity in our model is the fact that none of the enzymes involved in destroying β-catenin bind it directly. Instead they are loaded onto a scaffold. Moreover, the scaffold can head-to-tail homopolymerize, in addition to having three independent binding sites on a second scaffold, itself capable of dimerization. This allows a complex of scaffolds, where connection paths or stoichiometries are dynamic. It is this complex that acts as a super-scaffold to bring the substrate in contact with the enzymes. Considering both scaffolds contain large regions of disorder (i.e. chunks of unfolded peptide with high flexibility), it is sensible to believe an enzyme loaded on one scaffold could modify the substrate loaded on the neighboring scaffold. Lacking experimental evidence to suggest a ballpark limit for this reachable horizon, we leave it unconstrained: an enzyme will be able to modify any substrate loaded onto its complex.

Another source of complexity is that having kinases (i.e. enzymes that add a phosphate group) and phosphatases (i.e. enzymes that remove a phosphate group) loaded on the same complex will result in unimolecular do-undo loops. Conceivably the kinetics of complexes will vary heavily with the amount of kinases, phosphatases, and substrates loaded onto them. These are all dynamic properties.

We leverage our trace query engine to explore the dynamics of this system. More precisely, we develop queries to probe the agent-centric dephosphorylation dynamics, to measure the time it takes for an agent to navigate the modification steps, and to explore the complexes at which events happen. Our results are relevant to other pathways in addition to Wnt, from NFκB to RAS/ERK to the most studied protein on the world, P53; the pathways these proteins regulate make heavy use of polymeric scaffold complexes, sequential modifications, and do-undo loops.

5.1 Experimental Protocol and Queries

To explore our system, we create a Kappa model with three parametrizations. The model contains the scaffold proteins Axin1 (Axn) and APC, the kinases CK1α (CK1) and GSK3β (GSK), the protein phosphatases PP1 and PP2, and the substrate of all these reactions, β-catenin (Cat). The destruction complex recruits Cat through Axn. It then gets phosphorylated at the Serine on position 45 (S45) by CK1. While S45-phosphorylated, it can be phosphorylated at the Threonine on position 41 (T41) by GSK. While T41-phosphorylated, it can be phosphorylated on both Serines on positions 37 and 33 (S37 and S33). Once S37- and S33-phosphorylated, Cat is degraded. Meanwhile, PP1 undoes the phospho-

rylations of CK1, while PP2 undoes those of GSK. Each kinase-phosphatase pair also compete against each other for binding sites on Axn.

The three parametrizations explore the relationship between phosphatase/kinase ratio and the distribution of do-undo events. The three parameter pairs are 50/10, 10/10, and 10/50, all in units of number of agents, and represent the number of kinases and phosphatases in the model (e.g. 10/50 presents 10 copies of PP1, 10 copies of PP2, 50 copies of CK1, and 50 copies of GSK). The scaffolds remain at an abundance of 100 each. The models begin with an initial amount of Cat of 500 agents, and the models are run for 500 simulated seconds. We use global stochastic rates for our reactions, a bi-molecular binding of 10^{-4} per second per agent, a uni-molecular binding of 10^{-2} per second, an unbinding of 10^{-2} per second, and a catalytic of 1.0 per second.

For all three parametrizations, we run the following queries on the resulting traces.

Undoing S45, T41, S37 and S33 phosphorylation Considering phosphatases undoing the phosphorylation of sites, does this happen to all agents? Does it happen to just a few agents? What is the distribution of dephosphorylation events per agent? (Figure 4)

$$
\begin{array}{ll}
\mathsf{match} & e : \{\, c{:}\mathrm{Cat}\,(\,S45\,{}^{1}_{p\,\to\,u}\,)\,\} \\
\mathsf{return} & \mathsf{agent\_id}\,(\,c\,)\,,\ \mathsf{time}\,[\,e\,]
\end{array}
\qquad
\begin{array}{ll}
\mathsf{match} & e : \{\, c{:}\mathrm{Cat}\,(\,T41\,{}^{1}_{p\,\to\,u}\,)\,\} \\
\mathsf{return} & \mathsf{agent\_id}\,(\,c\,)\,,\ \mathsf{time}\,[\,e\,]
\end{array}
$$

$$
\begin{array}{ll}
\mathsf{match} & e : \{\, c{:}\mathrm{Cat}\,(\,S37\,{}^{1}_{p\,\to\,u}\,)\,\} \\
\mathsf{return} & \mathsf{agent\_id}\,(\,c\,)\,,\ \mathsf{time}\,[\,e\,]
\end{array}
\qquad
\begin{array}{ll}
\mathsf{match} & e : \{\, c{:}\mathrm{Cat}\,(\,S33\,{}^{1}_{p\,\to\,u}\,)\,\} \\
\mathsf{return} & \mathsf{agent\_id}\,(\,c\,)\,,\ \mathsf{time}\,[\,e\,]
\end{array}
$$

Wait times What is the distribution of times spent between the first phosphorylation on an agent, and the time it gets degraded? (Figure 5)

$$
\begin{array}{ll}
\mathsf{match} & i : \{\, c : \mathrm{Cat}{+}\,\} \\
\mathsf{and} & \mathsf{first}\ \ p : \{\, c{:}\mathrm{Cat}\,(\,S45\,{}_{u\,\to\,p}\,)\,\}\ \ \mathsf{after}\ i \\
\mathsf{and} & \mathsf{first}\ \ d : \{\, c : \mathrm{Cat}{-}\,\}\ \ \mathsf{after}\ p \\
\mathsf{return} & \mathsf{time}\,[\,d\,] - \mathsf{time}\,[\,p\,]
\end{array}
\tag{3}
$$

About this query. Agent creation and destruction is expressed by suffixing agent names with + and −, respectively.

Component size and enzyme identity Where do the phosphorylation steps that actually lead to degradation occur? Do they happen mostly on large complexes? What is the composition in units of Axn and APC of the complexes where the phosphorylation events leading to degradation took place? What is

the distribution of kinase identifiers for the last phosphorylation events that lead to degradation? (Figure 6)

$$
\begin{array}{ll}
\mathsf{match} & d : \{\, c : \mathrm{Cat} - \,\} \\
\mathsf{and} & \mathsf{last}\ p_1 : \left\{\, c{:}\mathrm{Cat}\left(S45\,{}^{1}_{u\,\to\,p}\right),\ k_1{:}\mathrm{CK1}\left(c^{\,1}\right)\right\}\ \mathsf{before}\ d \\
\mathsf{and} & \mathsf{last}\ p_2 : \left\{\, c{:}\mathrm{Cat}\left(T41\,{}^{1}_{u\,\to\,p}\right),\ k_2{:}\mathrm{GSK}\left(c^{\,1}\right)\right\}\ \mathsf{before}\ d \\
\mathsf{and} & \mathsf{last}\ p_3 : \left\{\, c{:}\mathrm{Cat}\left(S37\,{}^{1}_{u\,\to\,p}\right),\ k_3{:}\mathrm{GSK}\left(c^{\,1}\right)\right\}\ \mathsf{before}\ d \\
\mathsf{and} & \mathsf{last}\ p_4 : \left\{\, c{:}\mathrm{Cat}\left(S33\,{}^{1}_{u\,\to\,p}\right),\ k_4{:}\mathrm{GSK}\left(c^{\,1}\right)\right\}\ \mathsf{before}\ d \\
\mathsf{return} & \mathsf{agent\_id}\,(\,k_1\,),\ \mathsf{count}\,(\,\mathsf{component}\,[\,\bullet p_1\,]\,(\,k_1\,),\ \texttt{"Axn"},\ \texttt{"APC"}\,),\ \\
 & \mathsf{agent\_id}\,(\,k_2\,),\ \mathsf{count}\,(\,\mathsf{component}\,[\,\bullet p_2\,]\,(\,k_2\,),\ \texttt{"Axn"},\ \texttt{"APC"}\,),\ \\
 & \mathsf{agent\_id}\,(\,k_3\,),\ \mathsf{count}\,(\,\mathsf{component}\,[\,\bullet p_3\,]\,(\,k_3\,),\ \texttt{"Axn"},\ \texttt{"APC"}\,),\ \\
 & \mathsf{agent\_id}\,(\,k_4\,),\ \mathsf{count}\,(\,\mathsf{component}\,[\,\bullet p_4\,]\,(\,k_4\,),\ \texttt{"Axn"},\ \texttt{"APC"}\,)
\end{array}
\tag{4}
$$

About this query. The component state measure computes the connected component that contains an agent in a mixture. It returns a set of agents S. The count function takes such a set S along with n strings denoting agent types and returns an n-tuple of integers indicating how many agents of each type appear in S.

5.2 Results and Interpretation

Distribution of undo events per agent To study the effect of adding phosphatase, we look at the distribution of dephosphorylation events per agent in Figure 4. S45 is the first residue to be modified in the causal chain leading to degradation; S37 is the last. Based on the 1:1 system, it is surprising to see increasing the phosphatase level five-fold maintains a similar total number of dephosphorylation events (compare curves' integrals). However, their distribution is quite different. Interestingly, increasing the amount of kinase to 1:5 led to decrease in dephosphorylation events, even though the dephosphorylation enzyme's abundance and rates were kept at the same levels. It is also worth noting, the 1:1 saw almost 30 thousand dephosphorylation events of S45, occurring on a shrinking pool of at most 500 copies of Cat. Clearly certain agents are caught in the do-undo loop; some specific agents are getting dephosphorylated almost 800 times. It is worth noting these levels of dephosphorylation imply a comparable number of phosphorylation events.

To answer the question that motivated this query, for S45 under 1:5 regime, most agents don't get sabotaged by the phosphatase: the blue line is quite flat. Decreasing the amount of kinase changes this, and under a 1:1 regime some agents get undone multiple times, a quarter seeing upwards of hundreds of undo events (e.g. from id 300 onward). Increasing the phosphatase to a 5:1 regime further exacerbates this, with over half the agents receiving undo events hundreds of times. The unavailability of phosphorylated S45 in turn inhibits the phosphorylation of T41, and so forth to S37 and S33. It is worth noting that, based on the

1:1 system, *increasing* the phosphatase five-fold *decreases* the number and extent of advanced dephosphorylation events, such as S33 and S37. Paradoxically, increasing the kinase five-fold has this same effect. We attribute the former to decreased availability of the intermediate phosphorylated states (i.e. if T41 is not phosphorylated, S37 can't be phosphorylated, ergo can't be dephosphorylated), and the latter to increased throughput to degradation (i.e. Cat is not around for long enough to get dephosphorylated, as once it gets fully phosphorylated it quickly proceeds to get degraded).

We call attention to the number of agents whose final sites got dephosphorylated (Figure 4), vs. the number of agents who got degraded (Figure 7, in Appendix A). The 1:5 or 1:1 systems both degraded over 450 agents each, but the former undid around 160 agents (Figure 4 S37, domain of blue curve) while the latter undid over 350 (Figure 4 S37, domain of red curve). For the 1:5 and 5:1 systems, both undid around 160 agents (Figure 4 S37, domain of blue and yellow curves), but the former degraded over 450 agents (Figure 7, blue curve) while the latter less than 50 (Figure 7, yellow curve). This argues the notion of efficiency (e.g. minimizing the amount of undo steps) can't readily be inferred from the throughput of the system.

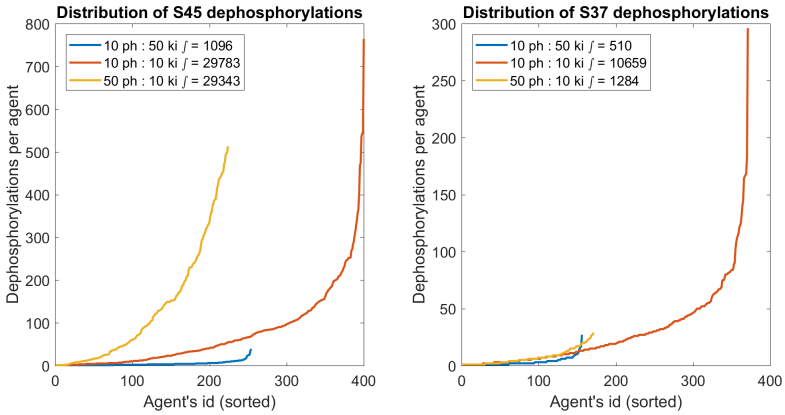

Fig. 4: Distribution of dephosphorylation events per agent. Each time an agent gets dephosphorylated, its ID is registered. After sorting, we plot the distribution of these IDs for two residues in the three parameter regimes. The area under the curve is also presented on each legend. S45 is the first residue to get phosphorylated, S37 (along with S33) is the last.

Wait times Looking at the distribution of wait times (Figure 5), from first phosphorylation to degradation, we note the bulk of degradation events occur rapidly, in less than 50 seconds. Worth noting that, from the 1:1 regime, increasing the amount of kinase five-fold marginally reduced wait times.

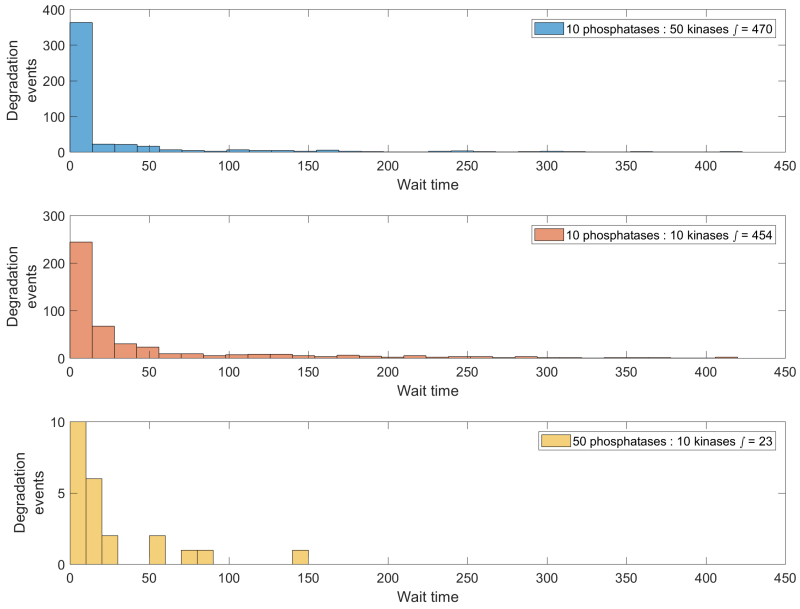

Fig. 5: Distribution of wait times from first phosphorylation until degradation. The sum of the bins is presented in the legend of each plot, and corresponds to the total number of degradation events, matching what is seen on Figure 7. The height of each bin represents the number of agents that waited the bin's position (in seconds) since they were first modified until they were degraded.

Complex composition A way of looking at the question of complex contribution is to query the size of the complex at the last phosphorylation event before degradation. Taking S45 as representative of all the other residues (see Figure 8 in Appendix A for a residue comparative), we plot the size of the complex, in terms of Axn and APC, at the time the final S45 occurred. Overall, we see a broad distribution of sizes, with some phosphorylation events occurring in large complexes (i.e. > 80 Axn, > 40 APC), but a significant number occurring in far smaller complexes (i.e. < 10 Axn, < 10 APC). Changing the parameter regime of kinase to phosphatase does not seem to alter this behavior significantly.

5.3 Summary of Findings

1. The number of undo events does not inform us of overall throughput (contrast Figure 4 and Figure 7).
2. How a step may be affected by changing abundances depends greatly on its upstream context (Figure 4).

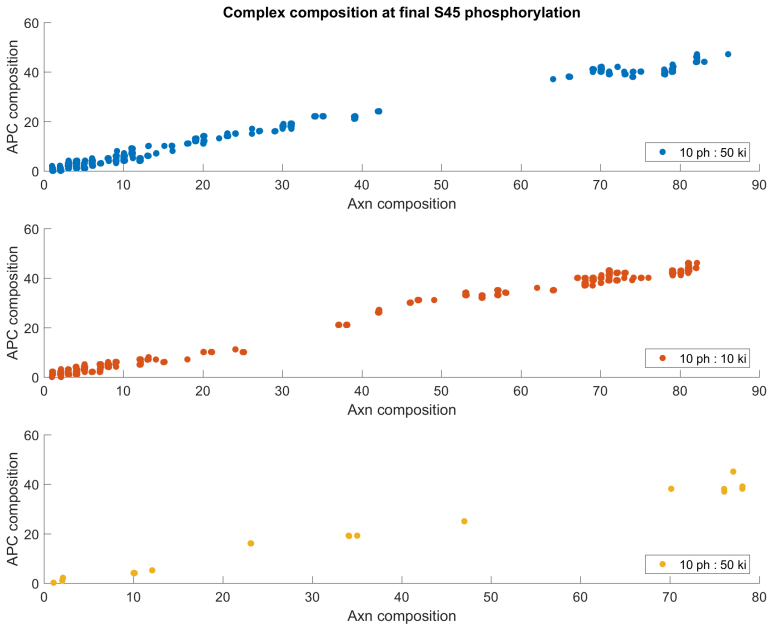

Fig. 6: Composition of the complex, in terms of Axn and APC components, at the last event where Cat got S45 phosphorylated before being degraded. The number of points corresponds to the number of degradation events. The points of this scatter plot have been nudged with a random noise factor of 0.2 to increase visual perception of discrete points where the data overlap.

3. Entities that got degraded waited a short while since their first modification (Figure 5), and yet most modifications were futile (Figure 4).
4. We can't argue that giant complexes, nor small complexes, nor medium complexes, are the sole entities responsible for performing the effective (i.e. final) phosphorylation events (Figure 6); the distribution of complexes is wide, and they all contribute to the kinetics.

The capacity of querying a simulation's trace offers a mechanistic description of the inner workings of our system. Since this description uses the vocabulary of molecular biology, it can greatly inform the search for drug targets.

For example, in our setup, complexes with over 60 copies of Axn and over 40 copies of APC contributed a large amount of degradation events (Figure 6). Considering there were a total of 100 copies of each scaffold, these large complexes are giant components, having recruited the majority of scaffolds into a single entity. If a single entity is contributing an amount of degradation events comparable to the rest of the mixture, it means its *effective* catalytic rate is greater than that of smaller entities. One could therefore reduce overall degradation of

Cat by destabilizing any of the three scaffold interactions (i.e. Axn-Axn, APC-APC, Axn-APC), without affecting the enzymes directly. Since these enzymes are also involved in metabolism, it would be desirable to avoid affecting their behaviors outside our pathway of interest.

6 Conclusion

How could one have explored a question like "which complexes contribute to kinetics"? Experimental biologists have been using labeling techniques for decades, but implementing this in a modeling framework requires being able to track individual agents, and query particular events. Implementing a framework to query events on the trace of an agent and rule simulation seems a natural way of tackling these classes of problems.

Moreover, once a sufficiently rich *mechanistic* model is available, questions on *mechanism* arise. For a subset of these, a satisfying answer will require a change of vocabulary; the explanations desired use the individual's lexicon (e.g. it bound, it unbound, it got dephosphorylated 800 times), rather than a whole system lexicon (e.g. the abundance changed from 500 to 50). Thus, rather than tracking the whole model's behavior (akin to a top-down approach), one needs to focus on agents, and observe their individual experiences (akin to a bottom-up approach). These approaches are complementary, as they explore a model's intricacies from very different viewpoints. We hope that the query language proposed in this paper will contribute to make agent-centric analysis more widespread and accessible.

References

1. Boutillier, P., Ehrhard, T., Krivine, J.: Incremental update for graph rewriting. In: Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017. pp. 201–228 (2017)
2. Danos, V., Feret, J., Fontana, W., Harmer, R., Krivine, J.: Rule-Based Modelling of Cellular Signalling, invited paper. In: Proceedings of the Eighteenth International Conference on Concurrency Theory, CONCUR 2007, Lisbon, Portugal. Lecture Notes in Computer Science, vol. 4703, pp. 17–41. Lisbon, Portugal (2007)
3. Danos, V., Feret, J., Fontana, W., Krivine, J.: Scalable Simulation of Cellular Signaling Networks, invited paper. In: Proceedings of the Fifth Asian Symposium on Programming Systems, APLAS 2007, Singapore. Lecture Notes in Computer Science, vol. 4807, pp. 139–157. Singapore (2007)
4. Danos, V., Feret, J., Fontana, W., Harmer, R., Hayman, J., Krivine, J., Thompson-Walsh, C.D., Winskel, G.: Graphs, Rewriting and Pathway Reconstruction for Rule-Based Models. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012. vol. 18, pp. 276–288 (2012)
5. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. The journal of physical chemistry **81**(25), 2340–2361 (1977)
6. Harris, L.A., Hogg, J.S., Tapia, J.J., Sekar, J.A.P., Gupta, S., Korsunsky, I., Arora, A., Barua, D., Sheehan, R.P., Faeder, J.R.: BioNetGen 2.2: Advances in rule-based modeling. Bioinformatics **32**(21), 3366–3368 (2016)

A Use Case Appendix

Concentration Time Traces From the output of the simulator, we get the evolution of the abundance of Cat through time. In Figure 7, we can see that the systems with low phosphatase behave similarly, even though one has five times the amount of kinases than the other (blue vs red traces). In contrast, the system with high phosphatase shows markedly less degradation of Cat; where the other two systems degraded around 450 units, this one has only degraded 23. From this whole-system view, it would seem the amount of phosphatase is more critical than the amount of kinase: based on the 1:1 system, increasing the kinase five-fold has little effect, whereas increasing the phosphatase has a more dramatic effect.

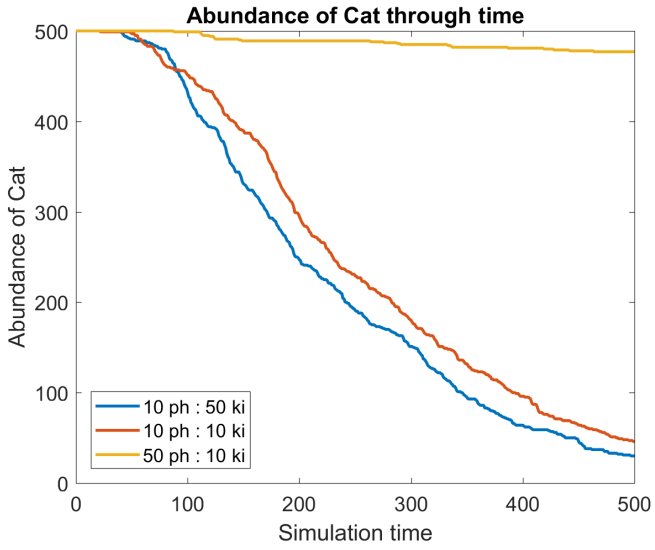

Fig. 7: Tracking the abundance of agent Cat through the simulation. At time $T = 0$, the agents are introduced, all in monomeric form. The simulation was stopped after five hundred simulated seconds. In this legend and throughout the figures, "ph" stands for phosphatase, "ki" stands for kinase, and numbers indicate agent multiplicity. Thus "10 ph : 50 ki" means the system with 10 units of phosphatase and 50 of kinase.

Complex composition: all four on the same component? For the final query, we wonder if all four final phosphorylation events occur on the same complex. Given the short wait time (Figure 5), one might expect so, but the number

of dephosphorylation events is so large (Figure 4), it could be well that a substrate is partially modified on one complex, subsequently modified on another, finalized in yet another. Lacking a metric by which we can compare complexes for distance, we instead compare complex compositions as a proxy.

Seeing how overwhelmingly, for each specific modification on a single Cat, the S45 phosphorylation events happened on complexes of the same Axn and APC composition as the T41 phosphorylation events, as the S37 phosphorylation events, as the S33 phosphorylation events, we feel confident in claiming all four steps occurred on the same complex. This agrees well with the observation that the wait times are fairly short (Figure 5). We did not see an appreciable difference for the other parameter regimes.

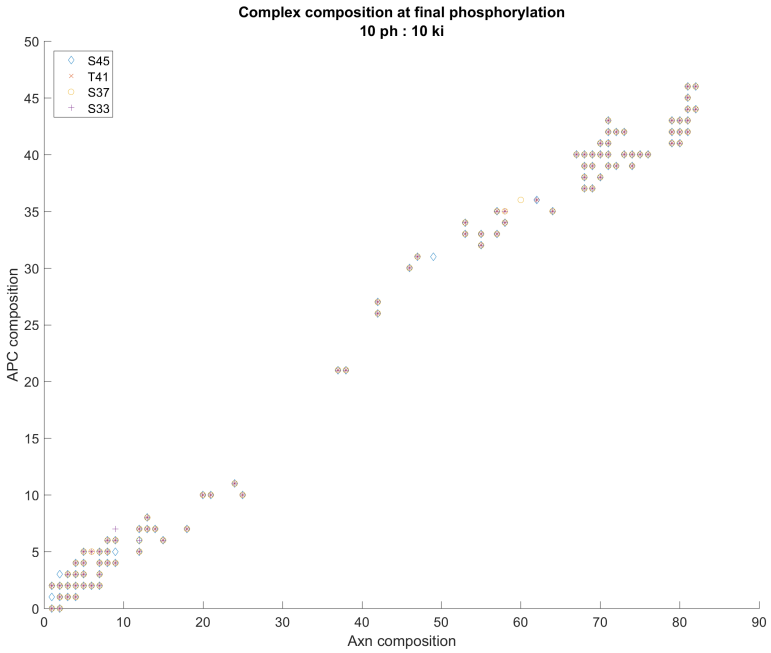

Fig. 8: Complex composition at the time of the last phosphorylation for the 1:1 system. All four residues are shown. A diamond superposed with a cross superposed with a circle superposed with a plus sign indicates that all four modifications for a specific copy of Cat occurred on a complex of the same composition in terms of Axn and APC. We interpret this as having occurred on the exact same complex.