Machine Learning (2022) 111:2435–2487
https://doi.org/10.1007/s10994-021-06038-y

Learning explanations for biological feedback with delays using an event calculus

Ashwin Srinivasan[1] · **Michael Bain**[2] · **A. Baskar**[3]

Received: 27 May 2020 / Revised: 27 May 2021 / Accepted: 9 July 2021 /
Published online: 18 August 2021

Abstract
We propose the identification of feedback mechanisms in biological systems by learning logical rules in R. Thomas' Kinetic Logic (Thomas and D'Ari in Biological feedback. CRC Press, 1990). The principal advantages claimed for Kinetic Logic are that it captures an important class of regulatory networks at an appropriate level of precision, and that the representation is close to that used routinely by biologists, with a well-understood relationship to a differential description. In this paper we present a formalisation of Kinetic Logic as a labelled transition system and provide a provably correct implementation in a modified form of the Event Calculus. The behaviour of a system is then a logical consequence of the core-axioms of a (modified) Event Calculus C, the axioms K implementing Kinetic Logic and the axioms H describing the system. This formulation allows us to specify system identification in the manner adopted in Inductive Logic Programming (ILP), namely, given C, K, system behaviour S and possibly some additional domain-knowledge B, find H s.t. $B \wedge C \wedge K \wedge H \models S$. Identifying a suitable Kinetic Logic hypothesis requires the simultaneous identification of definite clauses for: (a) logical definitions relating the occurrence of events to values of fluents; (b) delays in changes of the values of fluents arising from the occurrence of events; and possibly (c) exceptions to changes in fluent values, arising from asynchronous behaviour inherent to the system. We use a standard ILP engine for (a), and special-purpose abduction procedures for (b) and (c). We demonstrate this combination of induction and abduction on several canonical feedback patterns described by Thomas, and to identify the regulatory mechanism in two well-known biological problems (immune-response and phage-infection).

Keywords Biological feedback · Kinetic logic · Event calculus · Inductive logic programming

Editors: Nikos Katzouris, Alexander Artikis, Luc De Raedt, Artur d'Avila Garcez, Sebastijan Dumančić, Ute Schmid, Jay Pujara.

✉ Ashwin Srinivasan
ashwin@goa.bits-pilani.ac.in

Extended author information available on the last page of the article

1 Introduction

Feedback control is a common thread running through all levels of biological organisation. For example, it is needed for the production of proteins in a cell, cell-differentiation and cell-death. It is employed in signal transduction, enables fish to live in salt water, mammals to maintain body temperature, fruits to ripen, blood-pressure to be regulated, predator and prey populations to stay in balance, and any number of other biological processes. Much of this control is achieved by employing one or more of two elementary forms of feedback loops: positive or negative. It is known that the presence of a *positive* feedback loop is a necessary condition for a system to exhibit multi-stationarity; and a *negative* feedback loop is a necessary condition for it to have a point or oscillatory attractor. Broadly speaking, the wide variety of bistable switches found in biological systems (Wilhelm, 2009) need a positive feedback loop, and achieving homeostasis requires a negative feedback loop (see (Cinquin & Demongeot, 2002) for an assessment of the role of feedback in biological systems using differential equations and, more generally, see Robertson (1991) for a description of the role of feedback loops in shaping evolutionary change). So ubiquitous are these two elementary control mechanisms that it would only be a small exaggeration to say that in order to understand a biological system, we need to understand the underlying positive and negative feedback loops.

The idea that feedback control mechanisms developed by engineers for modelling physical systems should be applicable to biological systems is at least as old as the field of Cybernetics (Wiener, 1961). Early limitations on the applicability of quantitative methods developed for physical systems due to a lack of data are now being overcome by high-throughput methods in biology. However, at least three important sources of difficulty remain: (a) the lack of sufficient theoretical knowledge to be able to formulate the structure of quantitative models; (b) the data, although increased in volume, continue to be very noisy, making accurate parameter estimation difficult; and (c) the high precision of values employed by continuous models can obscure the basic biological principles.

The seminal work by René Thomas on the development of Kinetic Logic (Thomas and D'Ari, 1990) was principally motivated by two considerations. First, to move system understanding in Biology from the level of verbal descriptions to mathematical ones which were still comprehensible to biologists (Thomas, 1977). Secondly, the (still) prevalent description of systems in the form of differential equation models of continuous variables was seen both to be overly precise and yet not to represent accurately the non-linear dynamics of certain kinds of regulatory networks. Kinetic Logic is a *qualitative* representation that is suitable for systems that satisfy the following assumptions: (a) a system variable has little or no effect on other variables until its value reaches a threshold; and (b) at high values, the effect of a system variable reaches a plateau. These two assumptions have been shown to be sufficient to model adequately the dynamic effects of positive and negative feedback. In effect, a sigmoid-shaped change in concentration is approximated using a step-function, in which the regulator acts as a 2-way switch. That is, at concentrations below some (lower) threshold, the effect is 0, and above some (upper) threshold, the effect stays at a maximum.[1]

Kinetic Logic is, in effect, a sequential logic for describing the behaviour of asynchronous automata. That is, variables do not necessarily change values simultaneously

[1] Generalised Kinetic Logic extends this by employing multi-valued variables, which results in a ramp-shaped approximation. We will describe this extension in a later section.

(although this possibility is not precluded). Changes in values of variables are controlled by time-delays. An example, from Thomas (1991), describes the situation thus: "The signal which has switched on the gene functions as a *command* or *order* whose effective *realization* will take place only after a time t_x ...unless a counter order—gene off—has taken place before the expiration of the delay."

There is a natural fit between this kind of time-dependent behaviour of system-variables and that of fluents in the Event Calculus (Mueller, 2008). In this paper we show that Kinetic Logic can be implemented as domain-axioms in an Event Calculus. Since the Event Calculus is readily implemented as a logic program, our formulation of Kinetic Logic is also implemented as a logic program.[2] This also allows us to examine methods for the automatic identification of system-representations in Kinetic Logic. The main contributions of this paper are:

- A formal specification of Kinetic Logic with a declarative implementation in an Event Calculus. To the best of our knowledge this is the first time such a specification and implementation have been proposed. We also provide a proof of correctness of the implementation for the case of feedback without asynchronous change;
- The formulation of data-driven identification of systems represented in Kinetic Logic. This is implemented using methods developed in Inductive Logic Programming (ILP) and special-purpose methods for abduction to deal with the requirements of identifying feedback loops with delays and asynchronous change;[3] and
- Empirical results supporting our approach on 9 simple loops identified in the literature as building blocks for larger systems; and on 2 classic biological systems (immune-response and phage-infection). Together, the data cover systems ranging from 1 to 7 positive or negative feedback loops, containing 2–10 interactions.

The rest of the paper is organised as follows. In Sect. 2 we present two canonical examples of biological feedback (differentiation and homeostasis). In Sect. 3, we introduce Kinetic Logic, developed by Rene Thomas and colleagues as a means of representing and reasoning with biological feedback. The biological descriptions up to the end of Sect. 3.2 are known in the relevant literature. Readers familiar with them can proceed directly to Sect. 3.3, where we present a formalisation of Kinetic Logic as a transition-based system. The Event Calculus and an implementation of the formalised Kinetic Logic is in Sect. 4. In Sect. 5 we address the problem of learning Kinetic Logic models using the Event Calculus implementation. Section 6 demonstrates the performance of the learning methods using some canonical feedback loops, and two well-known biological systems. Section 7 summarises related work, and Sect. 8 concludes the paper.

[2] This viewpoint is consistent with Inoue (2011) who, as far as we know, was the first to formulate genetic regulatory networks in Biology as a logic program.

[3] ILP has been used for identifying regulatory biological networks (see Sect. 7). However, identification of the mechanisms of asynchronous control of regulatory loops as originally formulated in Kinetic Logic has not so far been a primary focus there.

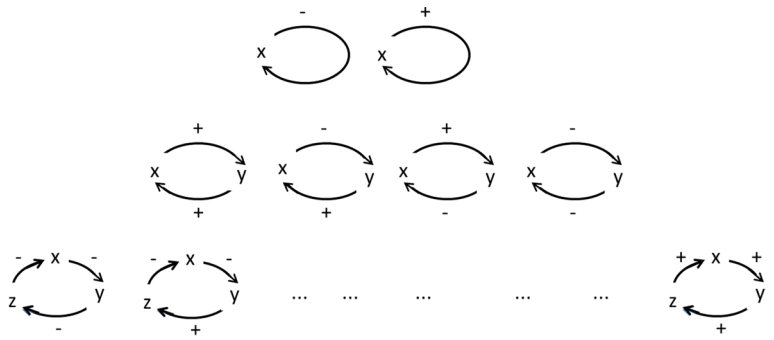


Fig. 1 Feedback loops formed from simple positive and negative interactions

2 Biological feedback loops

The principle underlying feedback is that a change in the value of some variable x, say Δx, directly or indirectly results in a (re-)adjustment of the value of x. At the outset, we clarify a distinction between individual interactions between variables, which can be *positive* or *negative*, and feedback loops, which denote cycles consisting of positive or negative interactions. An individual interaction[4] between variables x and y is said to be positive if an increase (or decrease) in the value of x results in turn in an increase (respectively, a decrease) in the value of y (usually after some delay). Similarly, the interaction is negative if an increase (decrease) in x results in a decrease (respectively, an increase) in the value of y, again after some delay. That is, x is an *activator* of y in the former case, and an *inhibitor* of y in the latter. In this paper, a *feedback loop* is a closed loop, or cycle, comprised of several such activators and inhibitors (see Fig. 1).

It is evident from Fig. 1 that the number of distinct loops possible grows rapidly with the number of edges. Despite this, it is remarkable that each variable in the loop only affects itself positively or negatively. This is obvious enough if all interactions in the loop are positive or negative, but is slightly less so if the interactions are mixed. With a single negative interaction, for example, the accumulated effect up to the negative edge is inverted by the (negative) edge, and that inverted effect is amplified further by the subsequent (positive) edges. The net effect over a single traversal of the loop starting at any variable is thus negative. The result is a *negative feedback* loop. More generally, any loop consisting of an *odd* number of negative interactions results in negative feedback, and a loop consisting of an *even* number of negative interactions results in positive feedback. Positive and negative feedback loops form the basis of regulatory control in a very wide variety of biological phenomena.

Two classic examples are *homeostasis* and *differentiation*. In homeostasis, feedback control is used to maintain the value of a variable at, or close to, some optimal value (that is, the biological equivalent of a thermostat). For example, the production of an amino acid may be activated at low levels of concentration, and inhibited at high levels. If the amino acid's concentration is initially high but is reducing then its production is initially inhibited but is later activated once its concentration levels fall below a threshold. The level of

[4] Typically, interaction edges in regulatory network graphs are directed (Klipp et al., 2016).

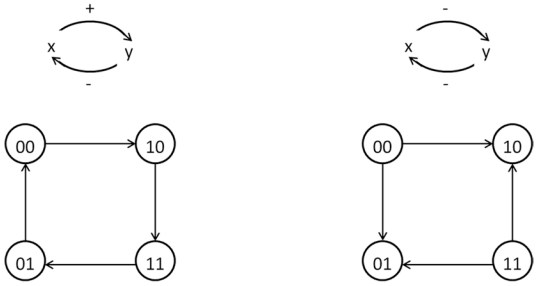


Fig. 2 Simple feedback loops and their state-transition diagrams (from Thomas and D'Ari, 1990). The loop on the left is a negative feedback loop and the one on the right is a positive feedback loop. States are represented by values for variables in the loops: here the variables are Boolean-valued, and 00 represents $x = 0, y = 0$ and so on. For the diagram on the right, we note state-transitions are possible from 00 to either of the stable states 01 or 10 (similarly, two state-transitions are possible from 11). The actual transition that does occur will depend on system-delays, which results in an *asynchronous* update of values of the state-variables. The exact meaning of this will be clarified later

the amino acid thus tends to oscillate around some level. A simple example of differentiation is evident when a gene's expression is dependent on its own product. Thus, the gene continues in one of two states, "on" or "off", depending on whether the product is present or absent. More complex situations arise in cases of cell-differentiation: although all cells have the same set of genes, some are "on" in one cell-line (heart, for example), and "off" in another (intestine, for example). Each of these can be understood abstractly as an instance of a stable-state in a system with multiple stable-states.

Homeostasis and differentiation are instances of two well-known consequences of negative- and positive-feedback loops. That is: the presence of *negative feedback* is necessary for oscillatory behaviour (as exhibited in homeostasis); and the presence of *positive feedback* is necessary for multi-stationary behaviour (as exhibited in cell-differentiation).

Differential equations describing positive and negative feedback can be found in Thomas and D'Ari (1990). Here we present instead the simple Boolean characterisation from Thomas and D'Ari (1990). In principle, a 2-element loop suffices as an abstraction of more complex loops consisting of even- and odd-numbers of negative interactions. Figure 2 shows two such abstract loop structures along with their state-transition diagrams for asynchronous updates of the values of variables. In this formulation (Thomas, 1983) the dynamics of the system (i.e., the exact sequence of states) is determined by the *delays* associated with the change of value of a variable (here from 0 to 1, or vice versa).

Let us denote by d_v^+ the time delay for the variable v to increase from one value to the next larger one (here from 0 to 1, but in general the values of v may be discretised into several bins), and d_v^- to decrease from one value to the next smaller one (here from 1 to 0). Then, in the state-transition diagram on the left in Fig. 2, assuming we start in state 00, state 10 is reached after d_x^+, 11 after a further d_y^+, 01 after d_x^-, and 00 after d_y^-. After this the system behaviour repeats: that is, it oscillates with period $d_x^+ + d_y^+ + d_x^- + d_y^-$. The system with the positive feedback on the right has two stable states. Starting from 00, if $d_x^+ > d_y^-$ the system transits to state 10. If $d_x^+ < d_y^+$ then it transits to 01 (for the present, we resolve the case of $d_x^+ = d_y^+$ by an arbitrary choice). Similarly, if the system starts at 11, it will transit to 10 or 01 depending on the relative values of d_y^- and d_x^-. The system thus exhibits *bistable* behaviour, depending on the initial conditions and the values of the delays involved. More general interaction networks will involve several variables, and include logical constraints

on the interaction. For example, the presence of x or y could act as an inhibitor for z, and so on. Kinetic Logic, initially proposed in the 1970s, and described in detail in Thomas and D'Ari (1990), provides a unified treatment of feedback systems consisting of logical combinations of positive and negative interactions with delays. In essence, it is a sequential logic that deals with asynchronous changes in the state of a system.

3 Kinetic logic

Thomas and his co-workers have provided several accounts of Kinetic Logic in the literature. Here, we present only the main concepts, starting from the simplest form of Kinetic Logic ("Naive" Kinetic Logic), in which all quantities are Boolean-valued, and then proceeding to the multi-valued variant ("Generalised" Kinetic Logic). Both forms are motivated by attempting to arrive at qualitative abstractions of differential equations for a system consisting of variables $x_1, x_2, \ldots, x_n$. It is assumed that the rate of change in the value of each variable is given by an equation of the form:

$$\dot{x_i} = F_i(x_1, x_2, \ldots x_n) - k_i x_i \tag{1}$$

(This is a standard "stock-flow" model, with the negative term denoting a spontaneous decay in the value of the x_i)

3.1 Simple kinetic logic

The simplified (or Naive) form of Kinetic Logic (see Chapter 6 of Thomas (1977)) is a qualitative formulation of Eq. 1, resulting from the following assumptions:

Representation — The values of x_i are 0 or 1. Correctly the "value of the variable x_i" stands for the value of the Boolean function $x_i(t)$ but we will often use them interchangeably. Practically 0 or 1 here denotes "absent or present in sufficient amount". The F_i's are taken to be Boolean functions of $x_1, x_2, \ldots x_n$. Correctly we mean that the values of $F_i(t)$, which is $F_i(x_1(t), x_2(t) \ldots, x_n(t))$, are 0 or 1. The value of the constant k_i is taken to be 1. In the Kinetic Logic literature, the symbol X_i is used interchangeably to denote both the function $F_i(x_1, x_2, \ldots, x_n)$ and it's value at a particular time instant t. We will adopt the convention that $X_i(t)$ denotes $F_i(t)$ which is $F_i(x_1(t), x_2(t), \ldots, x_n(t))$.[5]

Dynamics — With these assumptions, the net rate of change $\dot{x_i}(t)$ is $X_i(t) - x_i(t)$. There is no change in the value of x_i when $X_i(t) - x_i(t) = 0$. Of the two other cases, if $X_i(t) = 1$ and $x_i(t) = 0$ (i.e., $X_i(t) - x_i(t) > 0$), then the net rate of change is $+1$. In this case, x_i increases by 1 after delay d_i^+, unless $X_i(t') - x_i(t') = 0$ at some instant $t' \in (t, t + d_i^+)$. Similarly, if $X_i(t) = 0, x_i = 1$, then $X_i(t) - x_i(t) < 0$. The net rate of change is -1 and

[5] In the biological setting, X_i usually denotes the occurrence of some meaningful biological event. The classic example is that $X_i = 1$ denotes a gene, or a set of genes (an *operon*), being expressed or "switched on". It will be assumed that the occurrence of such events is detectable (for example, we can detect if a gene is being expressed by using a microarray experiment), and re-use $X_i = 1$ to denote the event's detection.

x_i decreases by 1 after delay d_i^-, unless $X_i(t') - x_i(t') = 0$ at some instant $t' \in (t, t + d_i^-)$.

Example 1 (Simple Kinetic Logic: Representation) For the positive feedback loop in Fig. 2, let x and y denote proteins. The loop expresses the conditions that the presence of x inhibits the synthesis of y, and the presence of y inhibits the synthesis of x. A simple formulation of this in Kinetic Logic is the following:

1. x and y are binary valued variables taking values 0 or 1. It is understood that although we will use the terms "absent" and "present" for these values, this will mean that the concentrations are below or above some threshold.
2. Boolean functions $X(t)$ and $Y(t)$ denote the conditions under which the genes coding for the proteins x and y are expressed ("switched on") at time t. Thus, $Y(t) = \overline{x(t)}$ denotes that the presence of x at time t ($x(t) = 1$) will result in the inhibition of the gene for y at time t ($Y(t) = 0$) . Similarly, $X(t) = \overline{y(t)}$ denotes that the presence of y at time t ($y(t) = 1$) will suppress the expression of the gene for x at time t (that is, $X(t) = 0$).
3. We will take the state of the system to be the tuple ($x(t)$, $y(t)$, $X(t)$, $Y(t)$), although sometimes it is useful to distinguish the values of ($x(t)$, $y(t)$) and ($X(t)$, $Y(t)$).
4. Delays $d_x^+, d_x^-, d_y^+, d_y^-$ denote the time (in a simulation, time-steps) for a change to occur. For example, at $t = t_1$ let protein x be absent and corresponding gene be expressed. That is, $x(t_1) = 0$ and $X(t_1) = 1$. Then d_x^+ denotes that $x(t_1 + d_x^+) = 1$ provided $X(t) = 1$ for all $t \in (t_1, t_1 + d_x^+)$ (there is a little more complexity in the kinetic logic formulation arising from asynchronous changes in the values of x and y which we ignore until later in the paper).

Example 2 (Simple Kinetic Logic: Dynamics) For the previous example, let us assume the following values for delays: $d_x^+ = 1, d_x^- = 2, d_y^+ = 2, d_y^- = 2$. Let $x = 0, y = 0$ at $t = 0$:

1. $x(0) = 0$, $y(0) = 0$. Since $X(t) = \overline{y(t)}$ and $Y(t) = \overline{x(t)}$, $X(0) = 1$ and $Y(0) = 1$. The state of the system at $t = 0$ is (0, 0, 1, 1).
2. Since $X(0) = 1$ and $x(0) = 0$, we have $x(t) = 1$ at time $t = 0 + d_x^+ = 1$, unless X changes value in the interval (0, 1). Since $Y(0) = 1$ and $y(0) = 0$, $y(t) = 1$ at time $t = 0 + d_y^+ = 2$, unless Y changes value in the interval (0, 2). So at $t = 1$, $x = 1$ and $y = 0$, $X(1) = \overline{y(1)} = 1$ and $Y(1) = \overline{x(1)} = 0$. So the state at $t = 1$ is (1, 0, 1, 0).
3. However, $Y(1) = 0$ overrides $y(2) = 1$ which was imposed by $Y(0) = 1$ with delay $d_y^+ = 2$. Further, since $X(1) = x(1)$ and $Y(1) = y(1)$, there will be no change in the values of x, y, and the system reaches a stable state (1, 0, 1, 0).

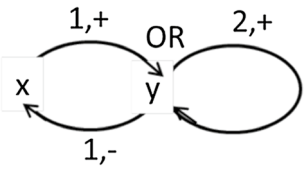
3.2 Generalised kinetic logic

A generalised form of Kinetic Logic (Thomas, 1991) removes the restriction to Boolean-valued variables and functions. Informally, the generalised form of Kinetic Logic is the following:

Representation The $x_i(t)$'s are multi-valued and the corresponding $X_i(t) = F_i(x_1(t), \ldots, x_n(t))$ is a function.

Dynamics The net rate of change of a variable will be +1, 0 or −1, depending on the value of $X_i(t) - x_i(t)$ (+1 if this difference is > 0; 0 if the difference is 0; and −1 if the difference is < 0).

There is no change in the value of x_i when $X_i(t) - x_i(t) = 0$ (that is, the net rate of change is 0). Of the two other cases, if $X_i(t) - x_i(t) > 0$ then x_i increases by 1 after some delay d_i^+ unless X_i changes value in $(t, t + d_i^+)$.[6s] Similarly, if $X_i(t) - x_i(t) < 0$, then x_i decreases by 1 after some delay d_i^-, unless X_i changes value in $(t, t + d_i^-)$.



Example 3 (Generalised Kinetic Logic: Representation) Consider the connected feedback loops from Thomas and D'Ari (1990), annotated slightly differently here:

1. x is a 2-valued variable (with values 0, 1) and y is a 3-valued variable (with values 0, 1, 2). The corresponding functions X and Y are 2- and 3-valued respectively.
2. The label on a directed edge (i, j) is the pair (l_{ij}, s_{ij}) where l_{ij} denotes a level and s_{ij} the sign of the interaction. So $(1, -)$ on the edge (y, x) denotes that y inhibits x once $y \geq 1$; $(1, +)$ on the edge (x, y) denotes that x activates y once $x \geq 1$; and $(2, +)$ on the edge (y, y) denotes that y auto-activates itself once $y \geq 2$.
3. The tabulation:

$x(t)$	$y(t)$	$X(t)$	$Y(t)$
0	0	K_{12}	0
0	1	0	0
0	2	0	K_{22}
1	0	K_{12}	K_{21}
1	1	0	K_{21}
1	2	0	K_{21+22}

specifies the functions:

$$X(t) = F_1(x(t), y(t)) = \begin{cases} K_{12}, & \text{if } \overline{y(t) \geq 1} \\ 0, & \text{otherwise} \end{cases}$$

and:

[6] The condition is slightly more complicated than this and this will be clarified in Sect. 3.3.

$$Y(t) = F_2(x(t), y(t)) = \begin{cases} K_{22}, & \text{if } \overline{x(t) \geq 1} \wedge \overline{y(t) \geq 2} \\ K_{21}, & \text{if } x(t) \geq 1 \wedge \overline{y(t) \geq 2} \\ K_{21+22} & \text{if } x(t) \geq 1 \wedge y(t) \geq 2 \\ 0, & \text{otherwise} \end{cases}$$

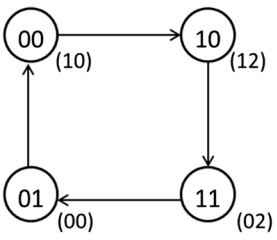
Here, $K_{12} \in \{0, 1\}$ and $K_{21}, K_{22}, K_{21+22} \in \{0, 1, 2\}$ are constants denoting sets of values, with the actual value in each case to be specified by the choice of logical function. Although it is easier to write $\overline{y(t) \geq 2}$ simply as $y(t) < 2$, we have continued to use the "negated" form to allow Simple Kinetic Logic to be a special case of generalised kinetic logic. That is $y(t) = 0$ will be represented as $\overline{y(t) \geq 1}$. The Ks are scaled versions of the rate parameters of the differential representations: The details can be found in Thomas (1991) and Snoussi (1989).

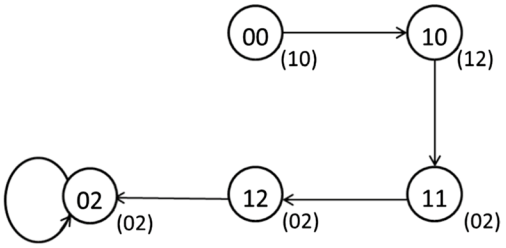
4. As before, the state of the system at time t is the tuple $(x(t), y(t), X(t), Y(t))$.
5. Delays $d_x^+, d_x^-, d_y^+, d_y^-$ that denote the time (or time-steps, for simulation). Here d_x^+ denotes the delay for increasing the value of x by 1 (level-)unit, and d_x^- denotes the delay in decreasing the value of x by 1 (level-)unit. Similarly for y.

Example 4 (Generalised Kinetic Logic: Dynamics) For the previous example, let us assume the following specification of $x(t)$ and $y(t)$. From now onward, we will remove reference to the time whenever it is clear from the context.

x	y	X	Y
0	0	1	0
0	1	0	0
0	2	0	2
1	0	1	2
1	1	0	2
1	2	0	2

The diagram on the left assumes $d_x^- < d_y^+$ and on the right $d_x^- > d_y^+$. For clarity, we include the values of X and Y in parentheses next to each state.





3.3 Specification

In this subsection, we provide a formal specification for a variant of a Kinetic Logic that is consistent with the informal description above.

Remark 1 *(Time)* Although not required by Kinetic Logic, we will adopt a discrete-time model, in which time is treated as a discrete variable that takes values from a set T. For convenience, we assume T is the set of natural numbers $\mathbb{N}$ ($\{0, 1, 2, \dots\}$.

Definition 1 *(System)* A system σ is a structure $(Xs, Vs, Fs, Ps, Ds, \succ)$. Here:

(a) $Xs = \{x_1, x_2, \dots, x_n\}$ is a set of identifiers denoting system variables for some $n \in \mathbb{N}$.
(b) $Vs = \{V_1, V_2, \dots, V_n\}$ where for each $i = 1, 2, \dots, n$, $V_i = \{0, 1, \dots, n_i\}$ for some $n_i \in \mathbb{N}$.
(c) $Fs = \{F_1, F_2, \dots, F_n\}$ is a set of functions of the system-variables. For each $i = 1, 2, \dots, n$, F_i denotes a function from $V_1 \times V_2 \times \cdots \times V_n$ to V_i.
(d) Ps is a set of identifiers $\{p_1, p_2, \dots, p_k\}$ denoting labels for some $k \in \mathbb{N}$.
(e) Ds is a function from $Ps \times Xs$ to $\mathbb{Z}^+ \times \mathbb{Z}^+$ denoting system delays.
(f) $\succ = \bigcup_{p,t} \succ_{p,t}$ where $\succ_{p,t} \subseteq Xs \times Xs$.

Informally, Ps is a set of identifiers that allows us to capture the notion of multiple pathways in a (biological) system. For convenience, we will use a set notation for the function Ds, and say $(x, p, d^+, d^-) \in Ds$ iff $Ds(x, p) = (d^+, d^-)$, for $x \in Xs$, $p \in Ps$ and $d^+, d^- \in \mathbb{Z}^+$. In the biological context, if (p, x, d^+, d^-) in Ds, d^+ is the time to increase x by 1 unit and d^- is the time to decrease x by 1 unit in the pathway identified with p. The relation $\succ$ is intended to capture the asynchrony inherent in Kinetic Logic, which forces at most 1 variable to change value at any time-instant on a pathway. Again for convenience we will use $(p, t, x_i) \succ (p, t, x_j)$ to denote that $(x_i, x_j) \in \succ_{p,t}$.

Definition 2 *(States)* Let $\sigma = (Xs, Vs, Fs, Ps, Ds, \succ)$ be a system. The state-space of the system σ is the set $\mathcal{S}_\sigma = \{(v_1, \dots, v_n, v'_1, \dots, v'_n) : \text{ for each } i = 1, \dots, n, v_i, v'_i \in V_i, \text{ and } F_i(v_1, \dots, v_n) = v'_i\}$. Elements in $\mathcal{S}_\sigma$ are called the states of the system σ.

Let s be a state in $\mathcal{S}_\sigma$. For each $i = 1, 2, \dots, n$, the notation $(x_i = v_i) \in s$ is used to denote there exists $v_1, v_2, \dots, v_{i-1}, v_{i+1}, \dots, v_n$ and $v'_1, \dots v'_n$ such that $s = (v_1, \dots, v_n, v'_1, \dots, v'_n)$. Similarly for each $i = 1, \dots, n$, the notation $(X_i = v'_i) \in s$ is used to denote there exists $v_1, \dots, v_n$ and $v'_1, \dots, v'_{i-1}, v'_{i+1}, \dots, v'_n$ such that $s = (v_1, \dots, v_n, v'_1, \dots, v'_n)$. In order to describe the dynamic behaviour of a system, we introduce the notion of a configuration.

Definition 3 *(Configurations)* Let σ be a system and S_σ be the state space of the system σ. A configuration γ of σ is an element of $T \times \mathcal{S}_\sigma$. We denote the set of all configurations of σ as Γ_σ.

Definition 4 *(Labelled Transition System)* Let $\sigma = (Xs, Vs, Fs, Ps, D, \succ)$ be a system, S_σ be the state space of the system σ and Γ_σ be the set of all configurations of σ. A labelled transition system LT_σ defined on σ is the triple $(\Gamma_\sigma, \rightarrow, Ps)$ where $\rightarrow \subseteq \Gamma_\sigma \times Ps \times \Gamma_\sigma$, and for each $i \in T, s \in \mathcal{S}_\sigma, p \in Ps, ((i, s), p, (i + 1, s))$ is in $\rightarrow$.

The notation $\gamma \xrightarrow{p} \gamma\prime$ is used to denote that $(\gamma, p, \gamma') \in \rightarrow$. We use labelled transitions to identify a specific sequence (or trace) of transitions. The reason for adding the self transitions (for each $i \in T, s \in \mathcal{S}_\sigma, p \in Ps, ((i,s),p,(i+1,s))$ is in $\rightarrow$) is to allow the system to continue in the same state if no change is required.

Definition 5 (Trace) Let $\sigma = (Xs, Vs, Fs, Ps, Ds, \succ)$ be a system, S_σ be the state space of the system σ, Γ_σ be the set of all configurations of σ and $LT_\sigma = (\Gamma, \rightarrow, Ps)$ be a labelled transition system defined on σ. A finite sequence of configurations $\langle \gamma_0, \gamma_1, \ldots, \gamma_l \rangle$ (where $l \in \mathbb{N}$) of σ is a p-trace of LT_σ iff for each $i = 0, 1, \ldots, l$ there exists $s_i \in \mathcal{S}_\sigma$ such that $\gamma_i = (i, s_i)$ and for each $i = 0, 1, \ldots, l-1$, $\gamma_i \xrightarrow{p} \gamma_{i+1}$. Let $Traces_{LT_\sigma}(p)$ be the set of all p-traces of LT_σ.

If σ is clear from the context, we just use $\mathcal{S}, \Gamma$ to denote the state space $\mathcal{S}_\sigma$ of σ, the set of all configurations Γ_σ of σ respectively. Similarly if LT_σ is clear from the context, we use $Traces(p)$ to denote the $Traces_{LT_\sigma}(p)$. We use the term trace to denote any p-trace of LT_σ where $p \in Ps$.

It is evident from Defnitions 4 and 5 that the time-instant associated with a configuration is the same as the position of the configuration in a p-trace, and therefore can be discarded. Nevertheless, for clarity, and possible future generalisations to other timed-automata, we will continue to treat configurations as time-state pairs. Let $\tau_p = \langle \gamma_0, \gamma_1, \ldots, \gamma_l \rangle$ be a p-trace of a labelled transition system LT_σ defined on σ. We will say $\gamma \in \tau_p$ if there exists an $i \leq l$ such that $\gamma = \gamma_i$. By $t = t_k$ in τ_p we mean the time-instant associated with the k^{th} configuration in τ_p. If τ_p is clear from the context, we just use $t = t_k$ instead of $t = t_k$ in τ_p. We note for the specific formulation here, $t = t_k$ is identical to $t = k$.

Kinetic Logic uses some additional notions defined on configurations in p-traces.

Definition 6 *(Net Rate of Change)* Let $\sigma = (Xs, Vs, Fs, Ps, Ds, \succ)$ be a system, LT_σ be a labelled transition system defined on σ, and τ_p a p-trace in LT_σ. Let $\gamma_k = (t_k, s_k)$ in τ_p, $(x_i = v_i) \in s_k$, and $X_i = v_i' \in s_k$. Then the net rate of change of variable x_i at $t = t_k$ in τ_p is $v_i' - v_i$ and is denoted by $Rate(\tau_p, x_i, t_k)$. $Rate(\tau_p, x_i, t_k)$ is (a) Positive, iff $v_i' - v_i > 0$; (b) Negative, iff $v_i' - v_i < 0$; and (c) Zero, iff $v_i' - v_i = 0$.

Positive or negative net rates of change result in *orders* to increase or decrease the value of a system variable x_i after some delay.

Definition 7 *(Orders)* Let $\sigma = (Xs, Vs, Fs, Ps, Ds, \succ)$ be a system, LT_σ be a labelled transition system defined on σ, and τ_p a p-trace in LT_σ. Let $(p, x_i, d_i^+, d_i^-) \in Ds$ and $\gamma_k = (t_k, s_k)$ in τ_p. Then we will say that there exists an order at $t = t_j$ in τ_p to increase (resp. decrease) x_i at $t = t_k$, iff: (a) there is a $\gamma_j = (t_j, s_j)$ in τ_p s.t. $t_j = t_k - d_i^+$ (resp. $t_k - d_i^-$); and (b) $Rate(\tau_p, x_i, t_j) > 0$ (resp. < 0).

The predicate $Order(\tau_p, x_i^+, [t_j, t_k])$ (resp. $Order(\tau_p, x_i^-, [t_j, t_k])$) is true iff there exists an order at $t = t_j$ in τ_p to increase (resp. decrease) x_i at $t = t_k$.

An order to increase (or decrease) is necessary for a variable to change. However, we will see below that this will not be sufficient. It is easy to see that there cannot exist orders at $t = t_i$ to both increase and decrease x (even with different delays). That is, for any x, p, we can show:

$$\nexists\, t_j, t_k(Order(\tau_p, x^+, [t_i, t_j]) \wedge Order(\tau_p, x^-, [t_i, t_k]))$$

Orders to change the value of x_i can be revoked if X_i changes value within the delay period. It is useful to introduce the notion of values of functions changing in a sequence of configurations.

Definition 8 *(Change in Function Value)* Let $\sigma = (Xs, Vs, Fs, Ps, Ds, \succ)$ be a system, LT_σ be a labelled transition system defined on σ, and τ_p a p-trace in LT_σ. Let $x_m \in Xs$. Then we will say that X_m changes in the interval $[t_a, t_b]$ in τ_p ($t_a < t_b$) iff: there exists $\gamma_i = (t_i, s_i)$ and $\gamma_j = (t_j, s_j)$ in τ_p s.t. (a) $t_a \leq t_i$, $t_i < t_j$ and $t_j \leq t_b$; and (b) $(X_m = v') \in s_i$, and $(X_m = v'') \in s_j$; and $v' \neq v''$.

The predicate $FuncChanges(\tau_p, X_i, [t_a, t_b])$ is true iff X_i changes in the interval $[t_a, t_b]$ in τ_p.

Definition 9 *(Cancellation)* Let $\sigma = (Xs, Vs, Fs, Ps, Ds, \succ)$ be a system, LT_σ be a labelled transition system defined on σ, and τ_p a p-trace in LT_σ. Let $x_m \in Xs$. Then an order at $t = t_i$ to increase (resp. decrease) x_m at $t = t_j$ is cancelled iff: (a) the predicate $Order(\tau_p, x_m^+, [t_i, t_j])$ (resp. $Order(\sigma, x_m^-, [t_i, t_j])$) is true; and (b) the predicate $FuncChanges(\tau_p, X_m, [t_i, t_j - 1)$ is true. That is, we define a predicate *Cancelled* as follows:

$$\begin{aligned} Cancelled(\tau_p, x_m^+, [t_i, t_j]) \leftrightarrow \\ Order(\tau_p, x_m^+, [t_i, t_j]) \wedge \\ FuncChanges(\tau_p, x_m^+, [t_i, t_j - 1]) \end{aligned} \qquad \begin{aligned} Cancelled(\tau_p, x_m^-, [t_i, t_j]) \leftrightarrow \\ Order(\tau_p, x_m^-, [t_i, t_j]) \wedge \\ FuncChanges(\tau_p, x_m^-, [t_i, t_j - 1]) \end{aligned}$$

A different variant of Kinetic Logic results if the F_i's are allowed to change their values within the delay period, as long as the net rates remain positive (or negative) for increase (or decrease). We do not consider this variant here. Orders that are not cancelled initiate updates of a system (there is an exception to this, which we will deal with below).

Definition 10 *(Necessary Conditions for Change)* Let $\sigma = (Xs, Vs, Fs, Ps, Ds, \succ)$ be a system, LT_σ be a labelled transition system defined on σ, and $\tau_p = \langle \gamma_0, \gamma_1, \ldots, \gamma_k \rangle$ be a p-trace of LT_σ. Then x can increase at $t = t_k$ iff there exists $t_i \in T$ s.t.: (a) $Order(\tau_p, x^+, [t_i, t_k])$; (b) $\neg Cancelled(\tau_p, x^+, [t_i, t_k])$; and (c) $Rate(\tau_p, x, t_k - 1) > 0$. We will say the predicate $Increase(\tau_p, x, t_k)$ is true iff these conditions are true. Similarly we can define the predicate $Decrease(\tau_p, x, t_k)$. We use the predicate $PendingOrder(\tau_p, x, t_k)$ to denote $Increase(\tau_p, x, t_k) \vee Decrease(\tau_p, x, t_k)$.

Remark 2 Let $\sigma = (Xs, Vs, Fs, Ps, Ds, \succ)$ be a system, LT_σ be a labelled transition system defined on σ, and $\tau_p = \langle \gamma_0, \gamma_1, \ldots, \gamma_k \rangle$ be a p-trace of LT_σ. We note that the following statement is true, for x:

$$\nexists t_k\ (Increase(\tau_p, x, t_k) \wedge Decrease(\tau_p, x, t_k))$$

Let us assume:

$$\exists t_k \ (Increase(\tau_p, x, t_k) \wedge Decrease(\tau_p, x, t_k))$$

From Definition 10, since *Increase* is true, $Rate(\tau_p, x, t_k - 1) > 0$ and since *Decrease* is true $Rate(\tau_p, x, t_k - 1) < 0$, which is a contradiction, and the result follows.

Usually, if $PendingOrder(\tau_p, x, t_j)$ is true, then x will change by 1 unit at $t = t_j$. An exception arises however, if pending orders are true for a pair of variables x_i, x_j to change values at the same instant. If this happens, then changes are decided by a mechanism of preference.

Definition 11 *(Overridden)* Let $\sigma = (Xs, Vs, Fs, Ps, Ds, \succ)$ be a system, LT_σ be a labelled transition system defined on σ and τ_p a p-trace in LT_σ. Let $x_i, x_j \in Xs$. Then any change in variable x_i at $t = t_k$ is overridden by x_j, and the predicate $Overridden(\tau_p, x_i, t_k)$ is true, iff: (a) $PendingOrder(\tau_p, x_i, t_k)$ is true; (b) $PendingOrder\ (\tau_p, x_j, t_k)$ is true; and (c) $(p, t_k, x_j) \succ (p, t_k, x_i)$.

In the original formulation of Kinetic Logic by Thomas changes in values of system variables ($x \in Xs$) happen asynchronously. That is, only one variable can change value at any time instant. In effect, this requires $\succ$ to be a total ordering.

We are now able to state necessary and sufficient conditions for a system-variable to change value.

Definition 12 *(Necessary and Sufficient Conditions for Change)* Let $\sigma = (Xs, Vs, Fs, Ps, Ds, \succ)$ be a system, and LT_σ be a labelled transition system defined on σ, and τ_p be a p-trace in LT_σ. Then a variable $x \in Xs$ changes value at $t = t_k$ in τ_p, and the predicate $VarChanges(\tau_p, x, t_k)$ is true, iff: (a) $PendingOrder(\tau_p, x, t_k)$ is true; and (b) $Overridden(\tau_p, x, t_k)$ is false.

We finally have all the pieces for specifying a Kinetic Logic system.

Definition 13 *(Kinetic Logic System)* Let $\sigma = (Xs, Vs, Fs, Ps, Ds, \succ)$ be a system. We will say a labelled transition system LT_σ is a Kinetic Logic system iff for all p-traces τ_p in LT_σ, for all $x_i \in Xs$, for all $\gamma_j = (t_j, s_j)$ in τ_p $(j > 0)$, for all $v \in V_i$ (where $V_i \in Vs$),

1. if $\neg VarChanges(\tau_p, x_i, t_j)$ and $(x_i = v) \in s_{j-1}$ then: $(x_i = v) \in s_j$;
2. if $VarChanges(\tau_p, x_i, t_j)$ and $Increase(\tau_p, x_i, t_j)$ and $(x_i = v) \in s_{j-1}$ then: $(x_i = v + 1) \in s_j$;
3. if $VarChanges(\tau_p, x_i, t_j)$ and $Decrease(\tau_p, x_i, t_j)$ and $(x_i = v) \in s_{j-1}$ then: $(x_i = v - 1) \in s_j$.

Definition 14 *(Pathway)* Let $\sigma = (Xs, Vs, Fs, Ps, Ds, \succ)$ be a system and LT_σ be a Kinetic Logic system. A pathway of LT_σ is any p-trace of LT_σ.

Example 5 (Kinetic Logic: Formal Representation) We now re-examine the system in Example 4 in light of the formalisation proposed. The system is now denoted by $\sigma = (Xs, Vs, Fs, Ps, Ds, \succ)$, where $Xs = \{x_1, x_2\}$. $Vs = \{\{0, 1\}, \{0, 1, 2\}\}$ and $Fs = \{F_1, F_2\}$ where the F_i are represented by the table:

x_1	x_2	$X_1 = F_1(x_1, x_2)$	$X_2 = F_2(x_1, x_2)$
0	0	1	0
0	1	0	0
0	2	0	2
1	0	1	2
1	1	0	2
1	2	0	2

Let $Ps = \{p_1, p_2\}$ and $Ds = \{(p_1, x_1, 2, 1), (p_1, x_2, 3, 2), (p_2, x_1, 2, 3), (p_2, x_2, 1, 2)\}$. Let $\succ = \emptyset$.

Example 6 (Generalised Kinetic Logic Dynamics: again) Continuing Example 5, given the system σ, let Γ be the set of all configurations of σ (as defined in Definition 3), and $LT_\sigma = (\Gamma, \rightarrow, Ps)$ be a a labelled-transition system where $\rightarrow = \Gamma \times Ps \times \Gamma$). The dynamics allowed by our representation will be encoded by traces of LT_σ. One example is the trace $\tau_{p_1} = \langle$ (0, 0010), (1, 0010), (2, 1012), (3, 1012), (4, 1012), (5, 1102), (6, 0100), (7, 0100), (8, 0010) $\rangle$.

This trace is a pathway in this paper. The states in the sequence of configurations in the pathway are consistent with the states and transitions depicted in the left diagram in Example 4, given the delays for p_1:

We also note the following about τ_{p_1}: (a) At $t = 0$ an order to increase x_1 is placed to be executed at $t = 2$. Since X_1 does not change its value between 0 and 1, this order is not cancelled. So the value of x_1 increases by 1 unit at $t = 2$; (b) At $t = 2$ an order to increase x_2 is placed to be executed at $t = 5$. Since X_2 does not change its value between 2 and 4, this order is not cancelled. So the value of x_2 increases by 1 unit at $t = 5$; (c) At $t = 5$, two orders are placed: One is to decrease x_1 by one unit at $t = 6$ and another order is to increase x_2 by one unit at $t = 8$; (d) At $t = 6$, x_1 is decreased by one unit since X_1 does not change its value from $t = 5$. Using the definition of the function F_2 given in the table, $X_2 = 0$ at $t = 6$; (e) Since the value of X_2 changes from 1 to 0, the order at $t = 5$ to increase x_2 is cancelled; (f) At $t = 6$, an order to decrease x_2 is placed to be executed at $t = 8$. Since $x_1 = X_1$ there is no order placed to change x_1 and its value will be unchanged at $t = 7$ and $t = 8$; (g) At $t = 8$, x_2 decreases by one unit and we reach the state 0010.

In the following section, we describe an implementation, using an Event Calculus, of a program capable of inference in Kinetic Logic. Looking ahead, we are also able to state the system-identification task addressed in this paper, namely: Given Xs, Vs, Ps, and data Ts consisting of p-traces of the system, find Fs, Ds, and $\succ$ such that $\sigma = (Xs, Vs, Fs, Ps, Ds, \succ)$ is a Kinetic Logic System and every element of Ts is a p-trace of LT_σ.

4 Implementing kinetic logic in an event calculus

In this section, we present an implementation of the Kinetic Logic defined in Sect. 3.3 using a form of the Event Calculus. Theorem 1 at the end of the section establishes the correctness of the implementation. The Event Calculus (EC) is a logic-based formulation for temporal reasoning. We refer the reader to Kowalski and Sergot (1986), Shanahan (1999), Mueller (2008) and Katzouris et al. (2016) for an introduction to basic and extended forms of the EC, efficient implementations and practical applications.

Predicate	Meaning
$initially(F)$	Fluent F holds at time 0
$holds(F, T)$	Fluent F holds at time T
$initiates(E, F, T)$	Fluent F starts to hold at time T if the event E occurs at time T
$terminates(E, F, T)$	Fluent F ceases to hold at time T if the event E occurs at time T
$happens(E, T)$	Event E occurs at time T

(a)

$$holds(Fluent, 0) \leftarrow initially(Fluent)$$

$$holds(Fluent, T) \leftarrow happens(Event, T0), initiates(Event, Fluent, T0), not\ clipped(Fluent, T0, T)$$

$$clipped(Fluent, T1, T2) \leftarrow terminates(Event, Fluent, T), T1 \leq T, T \leq T2$$

(b)

Fig. 3 **a** The main predicates used for reasoning in a simple form of the Event Calculus, and **b** the core axioms of the Event Calculus. This simple version is included here for illustration: the actual axioms we use in the paper (see later) are more complex

The main predicates in a basic EC and their meanings are in the tabulation in Fig. 3a. The predicate definitions forming the basis of the core-axioms of this variant of the EC are in Fig. 3b. In Fig. 3, we use a Prolog-like notation, in which predicate-symbols, function-symbols and ground-terms are in lower-case, and variables are in upper-case. We abuse the notation *S/n* to denote predicates or functions with *n* arguments with the understanding the context will make it disambiguate the kind of symbol involved. The actual implementation (see below) that we will use for this paper is written in Prolog and requires the representation power of logic programs with negation-as-failure (Lloyd, 1987 for details). The event calculus axioms are written as clauses of the form $h \leftarrow l_1, l_2, \ldots, l_k$, to be read as "if l_1 and l_2 and ...l_k then h". If any of the l_i are of the form *not l*, then this should be read as "*l* is not provable".

There are two main representation choices we make to implement Kinetic Logic in the EC:

1. Values of system-variables (the x_i's in the previous section) are taken as fluents. When implemented as a logic program, these are represented by a function-symbol *val*/2, and fluents are ground-instances of this function. The predicate *holds*/3 is used to specify that a fluent holds at some time-instant. For example, $holds(\tau_p, val(x, 0), 5)$ denotes that the system-variable $x = 0$ at time 5 in p-trace τ_p.[7]
2. Values of system-functions (the X_i's) constitute events. Here, the function *occurs*/2 will be used to denote an event's occurrence, and ground instances of the *happens*/3 predicate will specify an event's occurrence at a specific time. When implemented as a logic program, we will use $f(x_i)$, rather than $X_i = F_i(\cdot)$. Thus, $happens(\tau_p, occurs(f(x_i), 1), 3)$ denotes that $X_i = 1$ at time 3.[8]

[7] We use a cached implementation of the EC that ensures that inconsistencies like $holds(\tau_p, val(x, 0), 5)$, $holds(\tau_p, val(x, 1), 5)$ do not arise. The cached implementation also avoids repeated recomputation of values of fluents.

[8] Again, the cached implementation ensures inconsistencies do not occur. Also, the notation $f(x)$ is a computational convenience suitable for logic programs, and is more correctly understood as the function f_x. The image of x continues to be a function of all system variables.

(C1)
$holds(P, val(X, V), 0) \leftarrow$
$\quad pathid(P),$
$\quad initially(P, val(X, V))$

(C2)
$holds(P, val(X, V), T_k) \leftarrow$
$\quad pathid(P),$
$\quad happens(P, tick, T_k),$
$\quad T_k > 0,$
$\quad initiates(P, Event, val(X, V), [T_i, T_k])$
$\quad not\ clipped(P, Event, val(X, V), [T_i, T_k])$

(C3)
$clipped(P, Event, val(X, V), [T_i, T_k]) \leftarrow$
$\quad terminates(P, Event, val(X, V), [T_i, T_k])$

Fig. 4 A variant of the Event Calculus to account for delays in Kinetic Logic. There is now a delay $T_k - T_i$ between the occurrence of an event and its effect on the corresponding fluent. *tick* is an event that happens on every time-instant. Upper-case letters denote variables in the logic-programming sense. So, the X here should not be confused with the $X(t)$ from the previous section. The functions $X(t), Y(t) \ldots$ will be represented by the ground-terms $f(x), f(y) \ldots$

There are some good reasons for these choices: (a) In the biological setting introduced by Thomas, the images (functions) act as the events triggering the change in values of the corresponding system variables; and (b) The value of the function is determined at every instant by the values of one or more system variables. Thus, there is no notion of persistence as with a fluent (although the impression of persistence may be apparent due to the function having the same value in successive time-instants).

4.1 Implementation in a variant of EC

A difficulty now arises from the fact that in the simplified EC in Fig. 3 events *instantaneously* initiate (or terminate) fluents, but we know from the previous section that in Kinetic Logic *delays* play a crucial role. Fig. 4 presents the implementation used in the paper. We note that *clipped* is a trivial rewrite of *terminates* in this implementation. Also, *terminates* is used in a slightly different manner to its usual formulation in the Event Calculus: Here we are interested in checking in if the action of the initiating event is terminated. This is similar to the use in Moyle (2003) where the initiating event is included in the calls to terminate.

All axioms also are now additionally augmented with an argument to identify the pathway on which the inference is to be performed. Recall that pathways are sequences of configurations $\langle \gamma_0, \ldots, \gamma_n \rangle$. Besides events associated with system-functions, we will use a *tick* event, which acts as follows: *happens*(*tick*, t) is *true* for each $t \in T$, where $T = \{0, 1, \ldots\}$ as in Sect. 3.3.

At the heart of any EC axiomatisation are domain-specific axioms. The axioms specific to Kinetic Logic are shown in Fig. 5. The primary axioms dealing with the delayed increase or decrease of system-variables by the corresponding functions are in Fig. 5a.

Before we establish a correctness result about the implementation, we illustrate its working based on the system described in Examples 3 and 4, and using the axioms in Figs. 4 and 5. For simplicity, we initially assume that there are no definitions for *overrides*/4 (that is, we ignore any asynchronous changes). Later we provide an example of the role played by this predicate.

Example 7 (Delayed fluent increase) For some p, assume we know the following: (a) At $t = 0$, *happens*(p, *occurs*(*f*(y), 2), 0), is true and *holds*(p, *val*(y, 0), 0) is true; (b) The delay for initiating an increase in y is 1 time unit, i.e., *delay*($p, y, +1, 1$) is true. We want to know

(KI1)
$initiates(P, occurs(f(X), V_i'), val(X, V_k), [T_i, T_k]) \leftarrow$
$\quad delay(P, X, +1, D),$
$\quad T_i \ is \ T_k - D,$
$\quad happens(P, occurs(f(X), V_i'), T_i),$
$\quad rate(P, X, T_k, D, R_i),$
$\quad R_i > 0,$
$\quad rate(P, X, T_k, 1, R_{k-1}),$
$\quad R_{k-1} > 0,$
$\quad holds(P, val(X, V_{k-1}), T_{k-1}),$
$\quad V_k \ is \ V_{k-1} + 1$

(KI2)
$initiates(P, occurs(f(X), V_i'), val(X, V_k), [T_i, T_k]) \leftarrow$
$\quad delay(P, X, -1, D),$
$\quad T_i \ is \ T_k - D,$
$\quad happens(P, occurs(f(X), V_i'), T_i),$
$\quad rate(P, X, T_k, D, R_i),$
$\quad R_i < 0,$
$\quad rate(P, X, T_k, 1, R_{k-1}),$
$\quad R_{k-1} < 0,$
$\quad holds(P, val(X, V_{k-1}), T_{k-1}),$
$\quad V_k \ is \ V_{k-1} - 1$

(KI3)
$initiates(P, tick, val(X, V), [T_i, T_k]) \leftarrow$
$\quad T_i \ is \ T_k - 1,$
$\quad happens(P, tick, T_i),$
$\quad holds(P, val(X, V), T_i)$

$rate(P, X, T, D, R) \leftarrow$
$\quad T_d \ is \ T - D,$
$\quad happens(P, occurs(f(X), V_1), T_d),$
$\quad holds(P, val(X, V), T_d),$
$\quad R \ is \ V_1 - V$

(a) Initiation of fluents: increase (top-left), decrease (top-right), and no change (lower-left and lower-right). The two calls to *rate*/5 in KI1 and KI2 implement the requirements in conditions (a) and (c) in Defn. 10.

(KT1)
$terminates(P, occurs(f(X), V_i'), val(X, V_k), [T_i, T_k]) \leftarrow$
$\quad T_{k-1} \text{ is } T_k - 1,$
$\quad fchanges(P, f(X), T_i, T_{k-1})$

(KT2)
$terminates(P, occurs(f(X), V_i'), val(X, V_k), [T_i, T_k]) \leftarrow$
$\quad T_{k-1} \text{ is } T_k - 1,$
$\quad overridden(P, X, T_k)$

(KT3)
$terminates(P, tick, val(X, V_k), [T_{k-1}, T_k]) \leftarrow$
$\quad initiates(P, occurs(f(X), V_i'), val(X, V_k), [T_i, T_k]),$
$\quad not \ terminates(P, occurs(f(X), V_i'), val(X, V_k), [T_i, T_k])$

$fchanges(P, f(X), T_1, T_2) \leftarrow$
$\quad T_1 \leq T_2,$
$\quad happens(P, occurs(f(X), V_1), T_1),$
$\quad happens(P, occurs(f(X), V_2), T_2),$
$\quad V_1 \neq V_2$
$fchanges(P, f(X), T_1, T_2) \leftarrow$
$\quad T_1 < T_2,$
$\quad T \ is \ T_1 + 1,$
$\quad fchanges(P, f(X), T, T_2)$

$overridden(P, X, T) \leftarrow$
$\quad overrides(P, Y, X, T),$
$\quad Y \neq X,$
$\quad initiates(P, occurs(f(Y), V_i'), val(Y, V), [T_i, T]),$
$\quad not \ terminates(P, occurs(f(Y), V_i'), val(Y, V), [T_i, T])$

(b) Termination of changes in fluents (top-left) and persistence of fluents (top-right), along with auxiliary predicates (lower-left, lower-right)

Fig. 5 Domain-specific axioms for Kinetic Logic. $f(X)$ is used to denote the logical function associated with system variable X. *tick* is an event that happens at every time-instant. *overrides*/4 implements the $\succ$ ordering in the specification (see text for more details)

the value of y at $t = 1$. That is, we want an answer to the query: $holds(p, val(y, V), 1)$. Using axiom $C2$ in Fig. 4, an SLDNF refutation proof for the goal $(\leftarrow holds(p, val(y, V), 1))$ succeeds with answer-substitution $\{V/1\}$ using Axioms KI1 (for proving *initiates*), KT1 and KT2 (for proving *not clipped*) in Fig. 5. Some relevant goals in the successful proof using KI1 are: $(\leftarrow initiates(p, occurs(f(y), 2)), val(y, 1), [0, 1]))$; $(\leftarrow delay(p, y, +1, 1))$; $(\leftarrow happens(p, occurs(f(y), 2), 0))$; $(\leftarrow rate(p, y, 1, 1, 2))$; $(\leftarrow holds(p, val(y, 0), 0))$. Inference using C2 in Fig. 4 then attempts to establish $not\ clipped(p, occurs(f(y), 2), val(y, 1), [0, 1])$. This succeeds since the calls to $KT1$ and $KT2$ finitely fail.

The core axioms of the EC only encode the effect of change caused by the logical functions (events), and not their non-effects. Therefore we need to capture the notion that the values of system-variables will continue unchanged, unless an event occurred to change it. This is a variant of the frame problem, and general techniques for addressing it in the EC are described in Shanahan (1999), Mueller (2008). Here, we employ a *tick* event, that occurs once every clock-tick. Every time *tick* happens, it initiates the continuation of the value for a system-variable x (Axiom KI3 in Fig. 5a). This is terminated by Axiom KT3, if the occurrence of an event $f(x)$ results in a change in the value of x.

Example 8 (Persistence of fluents) We continue the exposition from the previous example. If $f(y) = 2$ at $t = 1$, then *initiates*(p, *occurs*($f(y)$, 2), *val*(y, 2), [1, 2]) is true (KI1: delayed fluent increase). Since *not clipped*(p, *occurs*($f(y)$, 2), *val*(y, 2), [1, 2]) succeeds, *holds*(p, *val*(y, 2), 2) is true. That is, $y = 2$ at $t = 2$. Now: (c) $f(y) = 2$ at $t = 2$ (that is, *happens*(p, *occurs*($f(y)$, 2), 2) is true). We want to know the value of y at $t = 3$: that is, we want an answer to the query *holds*(p, *val*(y, V), 3), using Axiom C2. The goal ($\leftarrow$ *initiates*(p, *val*($f(y)$, V_i'), *val*(y, V), $[T_i, 3]$)) using KI1 fails, since ($\leftarrow$ *rate*($p, y, 3, 1, R$)) fails. Similarly, a proof of ($\leftarrow$ *initiates*(p, *val*($f(y)$, V_i'), *val*(y, V), $[T_i, 3]$)) using KI2 fails. However, ($\leftarrow$ *initiates*(p, *tick*, *val*(y, 2), [2, 3])) succeeds using KI3. Furthermore *not clippe d*(p, *tick*, *val*(y, 2), [2, 3]) succeeds and ($\leftarrow$ *holds*(p, *val*(y, 2), 3)) succeeds using Axiom C2. That is, at $t = 3$ the value of y is unchanged from its value at $t = 2$ (the values of both $f(y)$ and y are equal at $t = 2$).

Example 9 (Delayed fluent decrease.) For the previous example suppose: (d) The delay for initiating a *decrease* in y is 2 time units (that is, *delay*($p, y, -1, 2$) is true); and (e) $f(y) = 1$ at $t = 3$ and at $t = 4$. That is, *happens*(p, *occurs*($f(y)$, 1), 3) and *happens*(p, *occurs*($f(y)$, 1), 4) are both true. What are the values of y at $t = 4$ and $t = 5$? For this, we need answer substitutions for the goals ($\leftarrow$ *holds*(p, *val*(y, V), 4) and ($\leftarrow$ *holds*(p, *val*(y, V), 5) using Axiom C2. We consider proofs for ($\leftarrow$ *holds*(p, *val*(y, V), 4)) first. From the previous examples, we know: $y = 2$ at $t = 3$ (steady state). Since $f(y) = 1$ at $t = 3$, any proof using Axiom KI1 will fail, since the value of $f(y)$ at $t = 3$ is less than y at $t = 3$. A proof using KI2 is not possible, since at $t = 4$ the delay period of 2 has not run out. This leaves Axiom KI3, which will succeed. The proof for ($\leftarrow$ *holds*(p, *val*(y, V), 4)) then attempts to prove *not clipped*(p, *tick*, *val*(y, 2), 4), which succeeds, since $f(y)$ cannot change value at $t = 3$, and there are no definitions for *overrides*/4. Thus, the proof for ($\leftarrow$ *holds*(p, *val*(y, V), 4) succeeds with answer-substitution $\{V/2\}$. That is, $y = 2$ at $t = 4$ Now we consider the proof for ($\leftarrow$ *holds*(p, *val*(y, V), 5)) using C2. At $t = 5$, the goal ($\leftarrow$ *initiates*(p, *occurs*($f(y)$, 1), *val*(y, 1), [3, 5])) succeeds using KI2. Since there are no definitions for *overrides*/4, and $f(y)$ does not change value at $t = 3$ and $t = 4$, we can see that *not clipped*(p, *occurs*($f(y)$, 1), *val*(y, 1), [3, 5]) succeeds, and the proof for ($\leftarrow$ *holds*(p, *val*(y, V), 5)) succeeds with answer-substitution $\{V/1\}$. That is, $y = 1$ at $t = 5$.

Example 10 (Termination of persistence) In the previous example, when attempting to prove ($\leftarrow$ *holds*(p, *val*(y, V), 5)) using C2, suppose the the inference mechanism selected axiom KI3, which will succeed with a refutation of ($\leftarrow$ *initiates*(p, *tick*, *val*(y, 2), [4, 5])). That is, a persistence of $y = 2$ is initiated by KI3 (recall $y = 2$ at $t = 4$). However to complete the proof using C2 for ($\leftarrow$ *holds*(p, *val*(y, V), 5)) with substitution $\{V/2\}$, it is necessary that *not clipped*(p, *tick*, *val*(y, 2), [4, 5]) must succeed. But *clipped*(p, *tick*, *val*(y, 2), [4, 5]) is provable using KT3 since $f(y) = 1$ has initiated the delayed decrease $y = 1$ at $t = 5$, and there are no definitions for *overrides*/4. So *clipped*(p, *tick*, *val*(y, 2), [4, 5]) succeeds and the proof for ($\leftarrow$ *holds*(p, *val*(y, 2), 5)) using KI3 fails. Persistence of the fluent is thus terminated by a pending order to decrease that has not been overridden.

Finally, we provide an example of asynchronous change using *overrides*/4.

Example 11 (Overrides) Suppose there exists a definition (*overrides*(p, z, y, T) $\leftarrow$) (that is, a change in the value of some system-variable z will always override any change in the value of system-variable y. Suppose the value of z increases at $t = 5$. Then the

proof of ($\leftarrow holds(p, val(y, V), 5)$) in Example 9 using C2 and KI2 fails when attempting to establish (*not clipped*(p, *occurs*($f(y)$, 1), *val*(y, 1), [3, 5])). This fails because *terminates*(p, *occurs*($f(y)$, 1), *val*(y, 1), [3, 5]) succeeds through the use of axiom KT2 and the definition of *overridden*/3. The latter succeeds since the proof for ($\leftarrow$ *overrides*($p, z, y, 5$), $z \neq y$, *initiates*(p, *occurs*($f(z), V_i'$), *val*(z, V), [T_i, 5])), *not terminates* (p, *occurs*($f(z), V_i'$), *val*(z, V), [T_i, 5])) succeeds for some V_i', V, T_i. That is, no (delayed) decrease of y is now possible. However, the proof for ($\leftarrow$ *initiates*(p, *tick*, *val*(y, 2), [4, 5])) will succeed, and *not clipped*(p, *tick*, *val*(y, 2), [4, 5])) also succeeds, since no change in y occurs. Thus the proof for ($\leftarrow holds(p, val(y, V), 5)$) succeeds using C2 and KI3, with answer-substitution $\{V/2\}$. That is, the change in y is overridden by a change in z at $t = 5$. In this case, the value of y at $t = 5$ persists from the previous time instant.

4.2 Relationship to the specifications

Here we show a form of correctness of the implementation with respect to the specification in Sect. 3.3. For a system $\sigma = (Xs, Vs, Fs, Ps, Ds, \succ)$, let τ_p be a trace in a labelled transition system LT_σ defined on σ. For all $x_i \in Xs$, and for all $\gamma_k = (t_k, s_k)$ in τ_p, $(x_i = v_i) \in s_k$ is denoted in the implementation by $holds(\tau_p, val(x_i, v_i), t_k)$. Denote $(X_i = v_i') \in s_k$ by $happens(\tau_p, occurs(f(x_i), v_i'), t_k)$. Thus the state s_k corresponds to the conjunction:

$$\bigwedge_{i=1}^{n} (holds(\tau_p, val(x_i, v_i), t_k) \wedge happens(\tau_p, occurs(f(x_i), v_i'), t_k))$$

where n is the number of system-variables in Xs. We will need the following.

Definition 15 *(Consistent Initialisation of τ_p)* Let $\sigma = (Xs, Vs, Fs, Ps, Ds, \succ)$ be a system. Let LT_σ be a Kinetic Logic system defined on σ. Let τ_p be a p-trace defined in LT_σ, and $\gamma_0 = (0, s_0) \in \tau_p$. Then the set $I_p = \{initially(p, val(x, v)) : x \in Xs\}$ is said to be a consistent initialisation of τ_p if $initially(p, val(x_i, v_{i,0})) \in I_p$ iff $(x_i = v_{i,0}) \in s_0$.

Let C denote the definitions in Fig. 4, K denote the definitions in Fig. 5. Let additional definitions describing the system be denoted by A_σ, and let $B = C \cup K \cup A_\sigma$. We are concerned with the correctness of a proof of *holds*/3 using B and a consistent initialisation of a p-trace.

Theorem 1 (Correctness of the EC Implementation) *Let $\sigma = (Xs, Vs, Fs, Ps, Ds, \succ)$ be a system and LT_σ be a Kinetic Logic system. Let τ_p be a p-trace of LT_σ. Assume*: (*a*) *correct definitions Fs, Ps, Ds*; (*b*) $\succ = \emptyset$; (*c*) *a consistent initialisation I_p of τ_p*; *and* (*d*) $B \cup I_p \nvdash \square$. *We show, for all $\gamma_k = (k, s_k) \in \tau_p$, for all $x_i \in Xs$ and $v_i \in V_i$, that* ($B \cup I_p \vdash holds(p, val(x_i, v_i), k)$) *iff* $(x_i = v_i) \in s_k$.

Here $\vdash$ denotes derivation using a theorem-prover, implementing SLDNF resolution using the usual Prolog search and computation rules. The restriction of $\succ$ to be $\emptyset$ results in a significant simplification of the proof. In practice too, this may not be an overly serious restriction, if the discretisation of time is sufficiently fine-grained.

We prove Theorem 1 by using induction. To apply induction we have to strengthen the claim as in Lemma 2 in "Appendix A".

We turn now to the learning task addressed in this paper. The examples to this point would have made clear that the two essential parts of Kinetic Logic are: when do events occur, and when are their effects felt? In the Event Calculus formulation we have just

described, answers to these questions are in the definitions of *happens*/3 and *holds*/3 predicates (which in turn, depends on *delay*/4). In addition—although the proof of correctness ignores it—we may be required to identify definitions for *overrides*/4, if the data contain asynchronous updates. Definitions for *happens*/3, *delay*/4 and *overrides*/4 have to be in place for inference of *holds*/3 to proceed. We will address this multi-predicate learning problem using Inductive Logic Programming (ILP).

5 Learning kinetic logic programs

5.1 Specification

We motivate the learning task in biological terms: given data in the form of the dynamic behaviour of a biological system, can we identify the regulatory interactions, activation and inhibition delays, and overrides of the system?

In terms of the Kinetic Logic we have specified, this amounts to identifying the Fs, Ds and $\succ$ of a system $\sigma = (Xs, Vs, Fs, Ps, Ds, \succ)$ given the Xs, Vs, and Ps. We will use the term "partially-specified system" to denote a system σ for which Xs, Vs, Ps are specified and Fs, Ds and $\succ$ are not specified. Then, the learning task we consider is: given a partially-specified system σ, and some p-traces of a Kinetic Logic system LT_σ, find definitions for Fs, Ds and $\succ$.

We now translate this requirement in terms of the implementation provided in the previous section:

Given the following:

- A partially-specified system $\sigma = (Xs, Vs, \cdot, Ps, \cdot, \cdot)$;
- $B = C \cup K \cup A_\sigma$, where C are the core-axioms of the Event Calculus in Fig. 4, K are the Kinetic Logic axioms in Fig. 5, and A_σ are auxiliary predicate definitions relevant to σ
- A language $\mathcal{L}$ for acceptable hypotheses;
- $S = E \cup F$ denoting system behaviour, consisting of the set of ground atoms $E = \{e_1, \ldots, e_j\}$ representing the conjunction of event occurrences $\bigwedge_i e_i$, and ground atoms $F = \{f_1, \ldots, f_k\}$ representing the conjunction of fluents $\bigwedge_i f_i$, of which the subset F_0 to denotes the values of fluents at $t = 0$.[9]

Where it is the case that:

$$B \cup F_0 \nvDash E \cup F$$

Find a set of clauses $H \in \mathcal{L}$ s.t.:

$$B \cup F_0 \cup H \vDash E \cup F$$

Of course $H = E \cup F$ would trivially achieve the requirement. But we want to be able to identify explanations in terms of *feedback mechanisms*, which require a generalisation of

[9] The set Fs in the specification of a system in Sect. 3.3 consists of functions $F_1, \ldots, F_n$, which is unrelated to F_0 here.

the specific facts observed. To achieve this, we will use the ILP notion of generalisation based on the relation of logical entailment (and the weaker relation of subsumption).

We will take $H = H_E \cup H_F$, where H_E refers to clauses needed for explaining the events, and H_F refers to clauses needed for explaining the fluents (these cannot be obtained entirely independently of each other, as will be seen below).

Example 12 (Learning Kinetic Logic Programs) For the system in Example 5, recall the dynamics of the Kinetic Logic system was given by the trace of a labelled transition system in Example 6: ⟨ (0, 0010), (1, 0010), (2, 1012), (3, 1012), (4, 1012), (5, 1102), (6, 0100), (7, 0100), (8, 0010) ⟩. Each element of this trace is the pair $(t, x_1x_2X_1X_2)$, where t is a time instant; x_1, x_2 denote the values of system variables; and X_1, X_2 denote the values $F_1(x_1, x_2)$, $F_2(x_1, x_2)$. Then, given traces of system-behaviour in this form, the main steps involved in learning a Kinetic Logic program are: (1) Obtaining data on the events and fluents for each time-instant t in the trace. Data on the events are the values of X_1, X_2 at each time-instant (constituting E), and data on the fluents are the values of x_1, x_2 (constituting F). F_0 will be the values of x_1, x_2 at $t = 0$; (2) Given the axioms in Fig. 4 (constituting C), Fig. 5 (constituting K) and any auxiliary definitions needed (A), obtain definitions for H_E and H_F. Learning definitions for H_E and H_F in this paper will require the following: (a) Learning definite-clause definitions for F_1 and F_2. In the implementation of the Kinetic Logic in Fig. 5, these are the definitions for *happens*/3 and constitute H_E; and (b) Definite clause definitions for the set of delays. This translates to definitions for *delay*/4 in Fig. 5, which are part of H_F; and (c) Definite clause definitions for exceptions caused by asynchronous change. These are definitions for *overrides*/4 in Fig. 5, which complete the definition of H_F. Usually, (b) and (c) will simply be ground unit clauses. We refer the reader to "Appendix C" for examples of actual clauses constructed.

5.2 Implementation

There are several ways a suitable H_E could be found: later we describe how this could be accomplished straightforwardly using ILP engine based on the techniques described in Muggleton (1995). For H_F it is apparent that we seek a hypothesis that entails the *holds*/3 facts provided as F. But a definition for *holds*/3 already exist in the core-axioms C: what is missing are definitions for *delay*/4 and—if we allow asynchronous updates—*overrides*/4. That is, the learning task is one of augmenting the existing background definitions in B. We look at identifying delays first. For the present, we will assume that changes in a variable are not overridden (that is, $>= \emptyset$ in Sect. 3).

In principle, identifying definitions for *delay*/4 and *overrides*/4 could be done using an ILP engine. In Muggleton (1994), it is shown that the ILP specification allows the construction of *abductive* hypotheses that extend the definitions in the background knowledge B by adding definitions of predicates that appear in the body of clauses in B (clearly, *delay*/4 and *overrides*/4 are such predicates). The ILP implementation we use in this paper (Srinivasan, 1999) does contain machinery for constructing abductive hypotheses, but it is inadequate for the requirements here.[10] Instead, we employ two procedures specifically

[10] Specifically, the implementation can construct a definition for *delay*/4, but it is not guaranteed to be consistent in the sense defined in Definition 16. It will also not construct a definition for *overrides*/4, since this predicate involves a proof using negation-as-failure.

for constructing the kinds of abductive explanations we seek here. We will first need the following:

Definition 16 *(Consistent Set of Delays)* Let $\sigma = (Xs, Vs, \cdot, Ps, \cdot, \cdot)$ be a partially-specified system, and $x \in Xs$. Then, for any $p \in Ps$, let $A_{x,p}$ denote the set of possible abducible *delay*/4 atoms. $D_{x,p} \subseteq A_{x,p}$ is said to be a consistent set of delays for x if it is one of: (a) $\emptyset$; or (b) $\{delay(p, x, +1, d^+)\}$; or (c) $\{delay(p, x, -1, d^-)\}$; or (d) $\{delay(p, x, +1, d^+), delay(p, x, -1, d^-)\}$.

A consistent set of delays for p is $Ds_p = \bigcup_x D_{x,p}$, where $D_{x,p}$ is a consistent set of delays for x in p. A consistent set of delays for the system is then $Ds = \bigcup_p Ds_p$.

Then, H_F can be any set Ds of consistent delays for the system s.t. $B \cup F_0 \cup H_E \cup Ds \models F$. The immediate question that arises is whether such a Ds is always guaranteed to exist? The short answer to this is "no": even in the case where the data are noise-free, the presence of asynchronous changes may result in some fluents not being entailed with any set of consistent delays. Also of interest is this: if a Ds exists, is it guaranteed to be unique? This is a property of Kinetic Logic, for which we do not have a definitive answer at this point.[11] A reasonably intuitive objective is to identify a set of consistent delays that entails the most number of fluents. Given this, identifying a Ds is better treated as an optimisation problem.

Definition 17 *(Event and Fluent subsets)* Let $\sigma = (Xs, Vs, \cdot, Ps, \cdot, \cdot)$ be a partially-specified system. Given a set of fluents F, we define the following subsets: F_p: the subset of fluents of F for $p \in Ps$; and $F_{p,t}$: the subset of fluents F for $p \in Ps$ at time t.

Given a set of events E, we define the following subsets: E_p: the subset of events E for $p \in Ps$; and $E_{p,t}$: the subset of events E for $p \in Ps$ at time t.

It is easy to see that $F_p = \bigcup_t F_{p,t}$ and $E_p = \bigcup_t E_{p,t}$.[12]

Definition 18 *(Optimal set of delays)* Let $\sigma = (Xs, Vs, \cdot, Ps, \cdot, \cdot)$ be a partially-specified system. Ds^*_p is said to be an optimal set of delays for $p \in Ps$ if: (a) DS^*_p is a consistent set of delays for p; and (b) there exists $F^*_p \subseteq F_p$ such that $B \cup F_{p,0} \cup H_E \cup DS^*_p \models F^*_p$; and (c) For any consistent set of delays DS'_p and $F'_p \subseteq F_p$ if $B \cup F_{0,p} \cup H_E \cup DS'_p \models F'_p$ then $|F'_p| \leq |F^*_p|$.

In this paper, we employ a search procedure that approximates DS^*_p by making three simplifying assumptions: (a) Delays are found one variable at-a-time, maximising the number of fluents provable for that variable; (b) At any time-instant t, fluents up to time t are correctly provable; and (c) The *delay*/4 atoms needed to prove fluents are obtained from a pre-defined set of abducible atoms A_p for $p \in Ps$. The details are in Procedure 1.

[11] We conjecture that the answer to this question is also "no", at least for the specification of Kinetic Logic as we have proposed.

[12] We note that this usage of F_p does not conflict with the functions of system-variables $F_1, \ldots, F_n$ defined in Sect. 3.3.

Procedure 1: (AbduceDelays) Greedy search for a consistent set $delay/4$ atoms for $p \in Ps$.

Input: A partially-specified system $\sigma = (Xs, Vs, \cdot, Ps, \cdot, \cdot)$; background knowledge B; $p \in Ps$; data consisting of events $E_p = E_{p,0} \cup \cdots \cup E_{p,k}$ and fluents $F_p = F_{p,0} \cup \cdots \cup F_{p,k}$; and a set A_p of abducible $delay/4$ atoms for $p \in Ps$
Output: A consistent set of delays Ds_p

for x *in* Xs **do**
  for $t = 1, 2, \ldots, k$ **do**
    Let $f_{p,x,t} \in F_{p,t}$ be the atom $holds(p, val(x, v), t)$;
    Let $A_{p,x,t} \subseteq A_p$ denote the set of $delay(p, x, \pm 1, \cdot)$ s.t. for each $a_{p,x,t} \in A_{p,x,t}$ $B \cup E_{p,0} \cup \cdots E_{p,t-1} \cup F_{p,0} \cup \cdots F_{p,t-1} \cup \{a_{p,x,t}\} \vdash f_{p,x,t}$;
    Let $A^+_{p,x,t} = \{(d, t) : delay(p, x, +1, d) \in A_{p,x,t}\}$;
    Let $A^-_{p,x,t} = \{(d, t) : delay(p, x, -1, d) \in A_{p,x,t}\}$;
  end
  Let $A^+_{p,x} = \bigcup_{t=1\ldots k} A^+_{p,x,t}$;
  Let $A^-_{p,x} = \bigcup_{t=1\ldots k} A^-_{p,x,t}$;
  if $A^+_{p,x} \neq \emptyset$ **then**
    Let d^+ be s.t. $(d^+, \cdot)$ is maximally frequent in $A^+_{p,x}$;
    $D^+_{p,x} = \{delay(p, x, +1, d^+)\}$
  else
    $D^+_{p,x} = \emptyset$
  end
  if $A^-_{p,x} \neq \emptyset$ **then**
    Let d^- be s.t. $(d^-, \cdot)$ is maximally frequent in $A^-_{p,x}$;
    $D^-_{p,x} = \{delay(p, x, -1, d^-)\}$
  else
    $D^-_{p,x} = \emptyset$
  end
  $D_{p,x} = D^+_{p,x} \cup D^-_{p,x}$;
end
Let $Ds_p = \bigcup_{x \in Xs} D_{p,x}$;
return Ds_p

Example 13 (Identifying delays) Assume we are given the fluents F_1 = { *holds*(*p*, *val*(*x*, 1), 1), *holds*(*p*, *val*(*y*, 0), 1) }; F_2 = { *holds*(*p*, *val*(*x*, 1), 2), *holds* (*p*, *val*(*y*, 1), 2) }, etc. Then $f_{x,1}$ = *holds*(*p*, *val*(*x*, 1), 1), $f_{y,1}$ = *holds*(*p*, *val*(*y*, 0), 1), $f_{x,2}$ = *holds*(*p*, *val*(*x*, 1), 2), $f_{y,2}$ = *holds*(*val*(*y*, 1), 2), and so on. Assume there are no definitions for *delay*/4. Let the possible abducibles allowed be A_p = {*delay*(*p*, *x*, +1, 1), *delay*(*p*, *x*, +1, 2), …, delay(p,x,-1,1),delay(p,x,-1,2), …, delay(p,y,+1,1), delay(p,y,+1,2), … }. Suppose in order to prove $f_{x,1}$ the atom $d_{x,1}$ = *delay*(*p*, *x*, +1, 1) is required. Similarly, suppose the proof for $f_{y,2}$ requires $d_{y,2}$ = *delay*(*p*, *y*, +1, 1); the proof for $f_{x,3}$ requires $d_{x,3}$ = *delay*(*p*, *x*, −1, 1); and for $f_{y,4}$ requires $d_{y,4}$ = *delay*(*p*, *y*, −1, 1). Then $A^+_{p,x,1}$ = { *delay*(*p*, *x* + 1, 1) }; $A^+_{p,y,2}$ = { *delay*(*p*, *y*, +1, 1) } and so on. From these, $A^+_{p,x,1}$ = {(1, 1)}; $A^-_{p,x,1} = A^+p, y, 1 = A^-_{p,y,1} = \emptyset$; and so on. Then $A^+_{p,x}$ = {(1, 1)}; $A^-_{p,x}$ = {1, 3)}; $A^+_{p,y}$ = {(1, 2)}; and $A^-_{p,y}$ = {(1, 4)}. Eventually, this yields

$D_{p,x} = \{delay(p, x, +1, 1), \quad delay(p, x, -1, 1)\}$; and $D_{p,y} = \{delay(p, y, +1, 1), \quad delay(p, y, -1, 1)\}$.[13]

Proposition 1 *The set DS_p constructed by Procedure* 1 *is a consistent set of delays.*

Proof In Procedure 1, for any variable x, $D_{p,x}$ will be obtained under one of the following conditions: (1) $A^+_{p,x} = \emptyset$, $A^-_{p,x} = \emptyset$; (2) $A^+_{p,x} \neq \emptyset$, $A^-_{p,x} = \emptyset$; (3) $A^+_{p,x} = \emptyset$, $A^-_{p,x} \neq \emptyset$; (4) $A^+_{p,x} \neq \emptyset$, $A^-_{p,x} \neq \emptyset$; The corresponding values of $D_{p,x}$ are: (a) $\emptyset$; (b) $\{delay(p, x, +1, d^+)\}$ for some d^+; (c) $\{delay(p, x, -1, d^-)\}$ for some d^-; and (d) $\{delay(p, x, +1, d^+), delay(p, x, -1, d^-)\}$ for some d^+ and d^-. Therefore $D_{p,x}$ is a consistent set of delays. It follows that $DS_p = \bigcup_x D_{p,x}$ is also a consistent set of delays. □

Remark 3 Step 4 in Procedure 1 is implemented with the usual SLDNF resolution theorem-prover used by Prolog. The total number of calls to the theorem-prover is $n \times k \times |A_p|$. Finding maximally-frequent delays in Steps 11,17 is $O(k|A_p|\log(k|A_p|))$. For all variables, this is $O(nk|A_p|\log(k|A_p|))$.

Since no delays are defined, they have to be abduced. Step 4 in Procedure 1 identifies abducible values for delays needed to derive a fluent at time t, assuming that we have already derived fluents up to $t-1$, using the events up to $t-1$. However, the delay atoms abduced by Procedure 1 may not allow an explanation of all fluents in a pathway, because of the sub-optimality of the set of delays found; asynchronous change that overrides increase or decrease in a variable; or of some other exceptional conditions (for example, noise in the data).

Example 14 (Asynchronous change) Continuing the previous examples, assume that given data on events and fluents concerning variables x, y, z, Procedure 1 returns the following: Ds ={delay(p,x,+1,1), delay(p,y,+1,1), delay(p,z,+1,1)}. Suppose the data for time-instants 0, 1 contains the events and fluents: $E_0 = \{ happens(p, occurs(f(x), 1), 0), happens(p, occurs(f(y), 0), 0), happens(p, occurs(f(z), 1), 0) \}$, and $F_0 = \{ holds(p, val(x, 0), 0), holds(p, val(y, 1), 0), holds(p, val(z, 0), 0) \}$, $F_1 = \{ holds(p, val(x, 1), 1), holds(p, val(y, 0), 1), holds(p, val(z, 0), 1) \}$. Then $F_{x,1} = holds(p, val(x, 1), 1)$ is provable from $B \cup F_0 \cup E_0 \cup Ds$ but $F_{z,1} = holds(p, val(z, 0), 1)$ is not provable. However, with $O = \{overrides(p, x, z, 1)\}$ (denoting that a change in x overrides the change in z at time-instant 1), then $B \cup F_0 \cup E_0 \cup Ds \cup O$ correctly proves $F_{x,1}$ and $F_{z,1}$, because:

1. $initiates(p, inc, z, [0, 1])$ will succeed, but $not\ clipped(p, inc, z, [0, 1])$ will fail, since $overridden(p, z, 1)$ is provable;
2. $initiates(p, std, z, [0, 1])$ will succeed and $not\ clipped(p, std, z, [0, 1])$ will succeed since there exists a variable x s.t. $overrides(p, x, z, 1)$ is provable, $initiates(p, inc, x, [0, 1])$ is provable, and $overridden(p, x, 1)$ is not provable. Therefore $holds(p, val(z, 0), 1)$ will succeed (that is, the value of z continues without change from the previous time-instant).

We will need the following:

[13] It is possible that multiple delays end up having the same frequency, an we require a mechanism for choosing amongst them. In the experiments in this paper, we select the longer delay.

Definition 19 *(Sufficient set of overrides)* Let $\sigma = (Xs, Vs, \cdot, Ps, \cdot, \cdot)$ be a partially-specified system. For any $p \in Ps$ with a set of fluents F_p and $x \in Xs$, let $holds(p, val(x, v_{x,t}, t) \in F_p$ and $holds_p, val(x, v_{x,t-1}, t-1) \in F_p$ for $t > 0$. Let $\Delta_{p,x,t} = |v_{x,t} - v_{x,t-1}|$. A set O_p is said to be a sufficient set of overrides for $p \in Ps$ iff for every $overrides(p, y, x, t) \in O_p$ it is the case that $\Delta_{p,y,t} \neq 0$ and $\Delta_{p,x,t} = 0$.

Procedure 2 returns a sufficient set of *overrides*/4 atoms for $p \in Ps$. In the procedure, a change in variable x is overridden at time t if: (a) a change (increase or decrease) in x is derivable at t, but not observed in F_t (these are identified in Step 13); and (b) there exists at least one other variable y for which a change is derivable at t, and the change is also observable at t (this is identified in Step 12).

Procedure 2: (AbduceOverrides) Identify a set of $overrides/4$ atoms for $p \in Ps$.

Input: A partially-specified system $\sigma = (Xs, Vs, \cdot, Ps, \cdot, \cdot)$; Background knowledge B; $p \in Ps$; data consisting of events $E_p = E_{p,0} \cup \cdots \cup E_{p,k}$ and fluents $F_p = F_{p,0} \cup \cdots \cup F_{p,k}$; and a consistent set of delays Ds_p
Output: A sufficient set of overrides O_p

1 **if** $Ds_p = \emptyset$ **then**
2 $\quad O_p = \emptyset$
3 **else**
4 $\quad$ **for** $t = 1, 2, \ldots, k$ **do**
5 $\quad\quad$ Let $F_{p,t} \subseteq F_p$ denote the set of fluents for time t;
6 $\quad\quad$ Let $F_{p,t-1} \subseteq F_p$ denote the set of fluents for time $t-1$;
7 $\quad\quad$ **for** $x \in Xs$ **do**
8 $\quad\quad\quad$ Let $f_{p,x,t} \in F_{p,t}$ be the atom $holds(p, val(x, v_{x,t}), t)$;
9 $\quad\quad\quad$ Let $f_{p,x,t-1} \in F_{p,t-1}$ be the atom $holds(p, val(x, v_{x,t-1}), t-1)$;
10 $\quad\quad\quad$ Let $\Delta f_{p,x,t} = |v_{x,t} - v_{x,t-1}|$
11 $\quad\quad$ **end**
12 $\quad\quad$ Let $F^1_{p,t} = \{y : y \in Xs,\ \Delta f_{p,y,t} \neq 0,$
$\quad\quad\quad B \cup E_{p,0} \cup \cdots E_{p,t-1} \cup F_{p,0} \cup \cdots F_{p,t-1} \cup Ds_p \vdash f_{p,y,t}\}$;
13 $\quad\quad$ Let $F^0_{p,t} = \{x : x \in Xs,\ \Delta f_{p,x,t} = 0,$
$\quad\quad\quad B \cup E_{p,0} \cup \cdots E_{p,t-1} \cup F_{p,0} \cup \cdots F_{p,t-1} \cup Ds_p \nvdash f_{p,x,t}\}$;
14 $\quad\quad$ Let $O_{p,t} = \{overrides(p, y, x, t) : f_{p,y,t} \in F^1_{p,t}, f_{p,x,t} \in F^0_{p,t}\}$;
15 $\quad$ **end**
16 $\quad O_p = \bigcup_{t=0\ldots k} O_{p,t}$
17 **end**
18 **return** O_p

Example 15 Continuing Example 14, $\Delta_{p,x,1} = 1$, $\Delta_{p,y,1} = 1$, and $\Delta_{p,z,1} = 0$. Then $F^0_{p,1} = \{z\}$; $F^1_{p,1} = \{x, y\}$ and $O = \{overrides(p, x, z, 1), overrides(p, y, z, 1)\}$.

Proposition 2 *The set O_p constructed by Procedure 2 is a sufficient set of overrides.*

Proof Steps 12 and 13 in Procedure 2 ensures that if $overrides(p, y, x, t) \in O_p$ then $\Delta f_{p,y,t} \neq 0$ and $\Delta f_{p,x,t} = 0$. By definition 19, the set O_p is therefore a sufficient set of overrides. □

Remark 4 In Procedure 2, Steps 12, 13 are implemented with the usual SLDNF resolution theorem-prover used by Prolog. The total number of calls to the theorem-prover is

$2 \times n \times k$, where n is the number of system variables and k is the number of time-instances for the pathway.

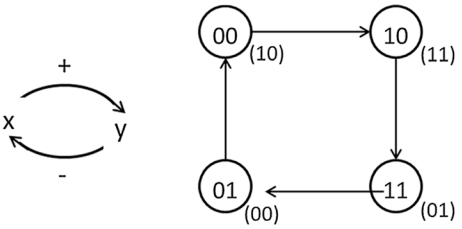
We now have all the pieces to construct H_E and H_F. This is done in Procedure 3. The procedure that invokes an ILP engine to construct generalisations where appropriate. Practical details related to the use of ILP are in "Appendix B".

Input: A partially-specified system $\sigma = (Xs, Vs, \cdot, Ps, \cdot, \cdot)$; background knowledge B; a set of abducible delays As; data for time-instants $t = 0, \ldots, k$ consisting of events $E = E_0 \cup \cdots \cup E_k$ and fluents $F = F_0 \cup \cdots \cup F_k$; ILP language restrictions $\mathcal{L}$; and an ILP implementation ILP
Output: H s.t. $B \cup F_0 \cup H \models E \cup F$

Let $E^+ = E$;
Generate E^- using the closed-world assumption and E^+;
Let $H_E = ILP(A_\sigma \cup F, \mathcal{L}, E^+, E^-)$;
for $p \in Ps$ **do**
  Let $E_p \subseteq E$ denote the events for $p \in Ps$;
  Let $F_p \subseteq F$ denote the fluents for $p \in Ps$;
  Let $A_p \subseteq A$ denote the set of abducibles for $p \in Ps$;
  Let $Ds_p = AbduceDelays(B, Xs, p, E_p \cup F_p, A_p)$;
  Let $O_p = AbduceOverrides(B, Xs, p, E_p \cup F_p, DS_p)$;
end
Let $Ds = \bigcup_{p \in Ps} Ds_p$;
Let $O = \bigcup_{p \in Ps} O_p$;
Let $F' \subseteq F$ s.t. $B \cup F_0 \cup H_E \cup Ds \cup O \vdash F'$;
Let $H_F = Ds \cup O \cup (F - F')$;
Let $H = H_E \cup H_F$;
return H

We have already presented examples of identifying Ds_p and O_p. In Procedure 3, F' are the fluents derivable using the definitions in Ds_p and O_p, given B, F_0 and H_E (that is, F' is redundant given $B \cup F_0 \cup H_E \cup Ds_p \cup O_p$). $(F - F')$ are the fluents not derivable using the abduced atoms for delays and overrides. In the following we restrict ourselves to present an example of the use of ILP to construct H_E in Step 3 of Procedure 3.

Example 16 (Identifying H_E) Let us suppose that we are given fluents and events for a system consisting of the simple negative feedback loop in Fig. 2, reproduced here, with the values of the functions (X, Y) in parentheses:



For a partially-specified system $\sigma = (Xs, Vs, \cdot, Ps, \cdot, \cdot)$ and $p \in Ps$, system behaviour consists of (a) E, the events, represented by:

$$happens(p, occurs(f(x), 1), 0), happens(p, occurs(f(y), 0), 0),$$
$$happens(p, occurs(f(x), 1), 1), happens(p, occurs(f(y), 1), 1), etc.,$$

and (b) F, the fluents, represented by: {*initially*$(p, val(x, 0))$, *initially*$(p,$ *val*$(y,$ 0)), *holds*$(p,$ *val*$(x,$ 1), 1), *holds*$(p,$ *val*$(y,$ 0), 1), *holds*$(p,$ *val*$(x,$ 1), 2), *holds*$(p,$ *val*$(y,$ 1), 2), …}. Although, correctly, F_0 = {*initially*$(p, val(x, 0))$, *initially*$(p, val(y, 0))$}, for uniformity we will take F_0 = {*holds*$(p, val(x, 0), 0)$, *holds*$(p, val(y, 0), 0)$}. Let $e_{x,0}$ = *happens*$(p,$ *occurs*$(f(x),$ 1), 0), and $e_{y,0}$ = *happens*$(p,$ *occurs*$(f(y),$ 0), 0), and so on. Thus, E^+ in Step 1 of Procedure 3 is the set $\{e_{x,0}, e_{y,0}, e_{x,1}, e_{y,1} \ldots\}$. Each variable is Boolean-valued, so E^- in Step 2 is the set {¬*happens*$(p,$ *occurs*$(f(x), 0), 0)$, ¬*happens*$(p,$ *occurs* $(f(y), 1), 0), \ldots\}$.

Suppose we use an ILP implementation based on the description in Muggleton (1995). This first constructs "most-specific clauses" and then constructs generalisations. Using language restrictions $\mathcal{L}$ that ensure that, along with the background knowledge B, only fluents F_t are used when constructing most-specific clauses for events in E_t, suppose the following represent most-specific clauses for $e_{x,0}$ and $e_{y,0}$:

$$\begin{array}{ll}
\bot_{x,0}: & \bot_{y,0}: \\
happens(p, occurs(f(x), 1), 0) \leftarrow & happens(p, occurs(f(y), 0), 0) \leftarrow \\
\quad holds(p, val(y, 0), 0), & \quad holds(p, val(y, 0), 0), \\
\quad holds(p, val(x, 0), 0), & \quad holds(p, val(x, 0), 0), \\
\quad opposite(1, 0) & \quad same(0, 0)
\end{array}$$

(where *opposite*/2 is assumed to be an auxiliary predicate defined in the background knowledge B). The ILP engine is capable of finding generalisations like: H_x and H_y (here upper-case letters are variables, as usual):

$$\begin{array}{ll}
H_x: & H_y: \\
happens(P, occurs(f(x), V), T) \leftarrow & happens(P, occurs(f(y), V), T) \leftarrow \\
\quad holds(P, val(y, V_1), T), & \quad holds(P, val(x, V), T) \\
\quad opposite(V, V_1) &
\end{array}$$

$B \cup F \cup H_x \cup H_y \models E^+$ and $B \cup F \cup H_x \cup H_y \cup E^-$ is consistent, hence $H_x \cup H_y$ is a possible hypothesis from the ILP engine. To accommodate generalised Kinetic Logic, we need to go beyond the expressive power of predicates like *same*/2 and *opposite*/2. It is useful to have instead predicates for generalised forms for positive and negative interactions in B:

$$\begin{array}{ll}
pos(P, val(Var, V), T) \leftarrow & neg(P, val(Var, V), T) \leftarrow \\
\quad values(Var, Vals), & \quad values(Var, Vals), \\
\quad member(V, Vals), & \quad member(V, Vals), \\
\quad Fluent = val(Var, V1), & \quad Fluent = val(Var, V1), \\
\quad holds(P, Fluent, T), & \quad holds(P, Fluent, T), \\
\quad V1 \geq V & \quad V1 < V.
\end{array}$$

Then, the most-specific clauses become:

$$\begin{array}{ll} \bot_{x,0}: & \bot_{y,0}: \\ happens(p, occurs(f(x),1),0) \leftarrow & happens(p, occurs(f(y),0),0) \leftarrow \\ \quad neg(p, val(y,0),0), & \quad pos(p, val(x,0),0), \\ \quad neg(p, val(x,0),0) & \quad pos(p, val(x,0),0) \end{array}$$

Possible generalisations are:

$$\begin{array}{ll} H_x: & H_y: \\ happens(P, occurs(f(x),1),T) \leftarrow & happens(P, occurs(f(y),0),T) \leftarrow \\ \quad neg(P, val(y,0),T) & \quad pos(P, val(x,0),T) \end{array}$$

Remark 5 In Procedure 3, the number of clauses examined for the construction of H_E in Step 3 can be estimated as follows. In the worst-case, for each $e \in E^+$, the ILP engine: (a) constructs a most-specific clause $\bot_e$; and (b) constructs generalisations of $\bot_e$. With the use of definitions for positive and negative interactions as in Example 16, and the use of a $\mathcal{L}$ that prevents the use of fluents across time-points, $\bot_e$ will have at most $n = |Xs|$ literals. If the number of interacting variables is bound to a value m, then the number of generalisations of any $\bot_e$ will be $O(n^m)$. The total number of clauses to be tested is therefore at most $|E^+|O(n^m) = |E|O(n^m)$. If the maximum number of time-instants on any pathway is k, then this bound is $k|Ps|O(n^m)$. The complexities of Steps 8, 9 are in Remarks 3, 4. Each of those are now multiplied by a factor of $|Ps|$. Thus, we expect the complexity of constructing explanations to be dominated by: (a) the number of system-variables n; and (b) the maximum number of interactions m allowed for a variable.

Proposition 3 *Let the background knowledge B be $C \cup K \cup A_\sigma$, where C, K are the axioms in Figs. 4, 5; and A_σ are auxiliary definitions. Let ILP be a procedure s.t. given: B, a language $\mathcal{L}$, positive examples E^+ and negative examples E^-, ILP returns a set of clauses $C \in \mathcal{L}$ that satisfies $B \cup C \models E^+$ and $B \cup C \cup E^-$ is consistent. If H is constructed by Procedure 3 then $B \cup F_0 \cup H \models E \cup F$.*

Proof

(a) Given the constraints on *ILP*, H_E in Step 3 of Procedure 3 satisfies $B \cup F \cup H_E \models E$
(b) Let $B \cup F_0 \cup H_E \cup Ds \cup O \vdash F'$, where $F' \subseteq F$ in Step 13 of Procedure 3. Since $\vdash$ is sound, $B \cup F_0 \cup H_E \cup Ds \cup O \models F'$. Since $H_F = Ds \cup O \cup (F - F')$, $B \cup F_0 \cup H_E \cup H_F \models F' \cup (F - F')$

From Steps a, b and $H = H_E \cup H_F$ in Procedure 3, the result follows. □

We note that although we are able to establish a form of algorithmic correctness for Procedures 1–3, there at least two assumptions that may not be immediately apparent. First, since Procedure 3 abduces facts for delays before it abduces overrides, there is a possibility that the resulting set of delays, although consistent, may not be the same as the true delays characterising the system. There is an assumption that changes are not overridden very often (this is consistent with the biological observations by Thomas that simultaneous requirements of change would occur very rarely, if at all in biological systems). If this is so, then since Procedure 1 attempts to find maximally frequent values for delays, we would expect that incorrect delay values would usually not be returned. Procedures 1 and 2 may also not find any definitions simply because the data do not contain sufficient information of changes in values of system-variables. For example, variable x may not change value at all in the data provided: in

this case, no delays for x would be abduced. Avoiding this requires an assumption that the data always contain sufficient information to abduce complete definitions for delays and overrides.

5.3 Evaluation of explanations

We use some simple quantitative and qualitative assessments of the explanations constructed (H's from Procedure 3). Quantitative assessments are the usual ones of Precision, Recall, and Accuracy. From the logical model, we can obtain: (a) The number of interactions correctly identified (N_1); (b) The number of interactions incorrectly identified (N_2); and (c) The number of interactions not identified (N_3). Then $Precision = N_1/(N_1 + N_2)$ and $Recall = N_1/(N_1 + N_3)$. Further, the *Accuracy* of the logical model allows us to assess if the model has correctly identified the logical relations between the interactions. *Accuracy* is simply the proportion of states correctly explained by the logical model. For experiments with synthetic data (see Sect. 6), we are additionally able to directly compare the delays abduced against the values used by the simulator to generate the synthetic data. We obtain the number of delays correctly identified (D_1) and the number of delays incorrectly identified (D_2). Then the accuracy of prediction is $Accuracy = D_1/(D_1 + D_2)$. In addition to these measures, we will also be interested in the complexity of explanations, measured simply by the number of clauses.

Qualitative evaluation of the explanations are in the form of interaction diagrams as directed graphs, with system-variables at nodes and edges labelled either "+V" or "-V", where $V > 0$ denotes the value taken by a function. Details of extracting interaction diagrams are in "Appendix B".

6 Case studies of synthetic and real systems

In this section we demonstrate the identification of feedback loops and delays. Specifically, we consider identification of the following:

(a) Simple synthetic structures consisting of negative and positive feedback loops, that form the basis of many complex biological systems; and
(b) Structures of real biological phenomena containing feedback loops. We focus on the invasion of a host organism by a phage, and the production of antibodies by a host organism on the introduction of a foreign body (antigen).

For (a), we will use data from simulations resulting from combinations of delays. For (b), we will use data from the biological literature. In each case, we assess the identification of interactions in the manner adopted in network identification, namely, by the correctness (precision) and completeness (recall) of the interactions in the logical model. For synthetic data (a), we are able to check the delays identified directly against those used for simulation. For the systems in (b), we compare the delays identified against what is reported in the literature. This is usually in the form of a relative ordering amongst the values. Additionally, the representation employed for (a) is simple (or naive) Kinetic Logic, in which variables are Boolean-valued. For (b), we are able to consider both simple and generalised Kinetic Logic.

In all cases, we distinguish between identifying *structure* and identifying *parameters*. By structure, we mean: (a) identification of the interaction edges along with their labels, and (b) the logical interaction between these interactions (for example, variable $y = \neg x \vee z$ to denote that y is the disjunction of a negative interaction with x and a positive interaction with z). By parameters, we mean the identification of delays for the (unit) increase and decrease of the variables.

6.1 Synthetic systems

6.1.1 Materials

In a previous section, negative and positive feedback were described as being determined by the parity of negative interactions. Specifically, an odd number of negative interactions in a loop results in negative feedback, and an even number of negative interactions results in positive feedback. In Chapter 7 of Thomas (1977), several simple forms of loop-templates are described, which are intended as building-blocks for many biological systems. The principal structures identified are: (a) Simple 2- or 3-variable positive or negative feedback loops; (b) Positive or negative loops with external "grafts" joined by AND or OR connections; and (c) Tangent loops, consisting of positive- or negative-loops with a common element. In this section we consider identifying each of these structures using positive- or negative- feedback loops. The structures we consider are in the table below:

S.No.	Name	Description
1	P_LOOP	2-var positive loop
2	N_LOOP	2-var negative loop
3	P_LOOP_AND	2-var positive loop with AND graft
4	P_LOOP_OR	2-var positive loop with OR graft
5	N_LOOP_AND	2-var negative loop with AND graft
6	P_LOOP_OR	2-var negative loop with OR graft
7	T_LOOP_PP	Tangent loop with two coupled 2-var positive loops
8	T_LOOP_NN	Tangent loop with two coupled 2-var negative loops
9	T_LOOP_PN	Tangent loop with coupled 2-var positive and 2-var negative loops

The template-structures for these loops are in Fig. 6.

6.1.2 Method

Our method consists of the following stages:

Data generation. We will use simulated data for all the systems (1)–(9) above. The simulations require: (a) A start state; (b) Delays for increase and decrease in the value of the variables. In this section, all variables are Boolean-valued. Kinetic Logic does not require delays $d_x{}^+$ and $d_x{}^-$ of a variable x to be equal to each other, or to the delays of any other variable. For experiments here, we will assume each delay is in the range $1 \ldots n$, where n is the number of variables in the system considered. That is, for a 2-variable system, the delays $d_x{}^+$ and $d_x{}^-$ for a variable x can each take values of 1, 2 and there will be 16 combinations of delays possible. Each combination will result in a pathway. The simulator obtains the sequence of states in a pathway using the axioms of the Event Calculus, Kinetic Logic, the domain-specific axioms describing the logical functions in the system, and a specific combination of delays for the variables. The simulator also ensures that transitions between states are asynchronous (that is, only one variable changes value from one state to another). Finally, we

assume the grafts required in tasks (3)–(6) are Boolean-valued input variables. A summary of the data generated by the simulator is below:

S.No.	Name	Data summary
1–2	P_LOOP	16 pathways for each loop,
	N_LOOP	arising from 4 delays for each of 2 variables
3–6	P_LOOP_AND, P_LOOP_OR	16 pathways for each loop, 8 with graft-variable = 1
	N_LOOP_AND, N_LOOP_OR	and 8 with graft-variable = 0
7–9	T_LOOP_PP,	16 pathways for each loop, sampled from
	T_LOOP_NN	729 possible pathways arising from
	T_LOOP_PN	9 delays for each of 3 variables

In all cases, each pathway consists of values for an initial state (that is, fluents and events) at $t = 0$ followed by the values of the states at 10 successive instants ($t = 1 \ldots 10$).

Learning. Given a set of pathways, the axioms of the Event Calculus and Kinetic Logic and any auxiliary definitions, we construct clauses to complete the Kinetic Logic axioms by learning definitions for the occurrence of logical functions and for the delays as described in Sect. 4.

Evaluation. Quantitative measures of evaluation are Precision, Recall, Accuracy and Size as described in Sect. 5.3. For problems involving sampled data (that is (7)–(9)), we repeat the experiment

6.1.3 Results

The results obtained are tabulated below.

S.No.	Name	Structure			Parameters	Theory
		Prec.	Recall	Acc.	Acc.	Size
1	P_LOOP	1.0	1.0	1.0	1.0	12
2	N_LOOP	1.0	1.0	1.0	1.0	12
3	P_LOOP_AND	1.0	1.0	1.0	1.0	12
4	P_LOOP_OR	1.0	1.0	1.0	1.0	12
5	N_LOOP_AND	1.0	1.0	1.0	1.0	4
6	N_LOOP_OR	1.0	1.0	1.0	1.0	12
7	T_LOOP_PP	1.0	1.0	1.0	1.0	25
8	T_LOOP_NN	1.0	1.0	1.0	1.0	25
9	T_LOOP_PN	1.0	1.0	1.0	1.0	25

That is, in all cases, the theory identified reconstructs the structure and parameters perfectly. These results provide the springboard for examining data for real biological systems.

We note that the purpose of the experiments here has not been to check the validity of the upper-bound on time-complexity obtained in Remark 5. Synthetic experiments that examine increase in time for theory construction could be devised: for example, identification of networks with varying numbers of interacting variables. Remark 5 suggests that increasing the number of interactions would increase the time for theory construction. This

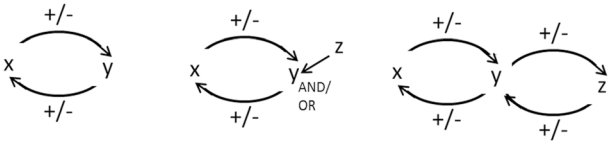


Fig. 6 Template structures for the loops considered in this section. The leftmost structure is for loops (1)–(2), the middle for loops (3)–(6), and the rightmost for loops (7)–(9)

is consistent with results obtained elsewhere in network identification (Friedman et al., 2000). We defer such experiments for future work, but note that some evidence for this is apparent in the identification of tangent loops, which have more interactions and take longer to identify than other simple feedback loops.

6.2 Real systems

We now consider identification of two biological systems concerned with the immune system response in vertebrates (interactions between lymphocytes), and the classic model of invasion of a bacterial cell by a virus (the well-known phage λ model). Brief biological descriptions of these systems are in "Appendix C": here we simply note that the models considered involve both simple and generalised Kinetic Logic.

6.2.1 Method

Data Extraction Data for the immune system are states identified in Thomas and D'Ari (1990). Data for the Boolean-model for phage infection are from Chapter 17 of Thomas (1977), and for the generalised model are from Thieffry and Thomas (1995). The details are summarised below:

S. No.	Name	Description
1.	IMMUNE_B	Data for 6 pathways from a Boolean-valued model for T_h and T_s lymphocyte-interaction. 3 pathways are in the presence of an antigen, and for 3 pathways the antigen is absent. The pathways with the antigen demonstrate development of immunity or immune paralysis and the pathways without the antigen contain a "memory" state that enables faster response to the antigen
2.	IMMUNE_G	Data for 13 pathways from a multi-valued model for T_h and T_s lymphocyte-interaction. 4 pathways are in the absence of an antigen; 4 with antigen present in moderate levels and 4 at high levels. Lymphocyte-cells are also considered at 3-levels of concentration
3.	PHAGE_B	Data for 7 pathways from a Boolean-valued model for the interaction between *cI*, *cro* and *cII*.
4.	PHAGE_G	Data for 4 pathways from a multi-valued model for interaction between *cI*, *cro*, *cII* and *n*. 1 pathway is the one most likely to have been followed for establishing host immunity (lysogeny)and 3 pathways leading to cell lysis. *cI* has 3 values, *cro* has 4 values, *cII* and *n* have 2-values each.

As with the synthetic data, pathways will consist of the initial state, followed by states for 10 additional time-steps.

Learning — As with the synthetic systems, given the set of pathways described above, and the axioms of the Event Calculus and Kinetic Logic, we construct clauses to complete the Kinetic Logic axioms by learning definitions for the occurrence of logical functions and for the delays.

Evaluation — As was done with synthetic data, we are able to assess the efficacy of structure identification by examining precision, recall, and accuracy. The literature does not provide explicit values of delays, and we are therefore not able to obtain accuracy estimates for delays, and rely instead on the accuracy of the theory to guide us on whether the delays hypothesised are appropriate.

We also tabulate the size of the theory constructed (measured by the number of clauses). We only consider learning *overrides* clauses if the abduction of delays does not explain all of the data.

6.2.2 Results

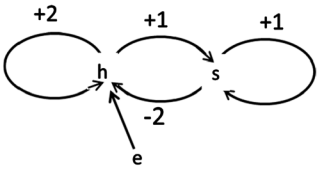
The results obtained are tabulated below.

S.No.	Name	Prec.	Recall	Accuracy	Size	Time(s)
1	IMMUNE_B	1.00	1.00	1.00	9	0.03
2	IMMUNE_G	0.71	1.00	0.92	17	0.12
3	PHAGE_B	1.00	1.00	‘.00	17	0.06
4	PHAGE_G	0.89	0.80	0.92	28	0.

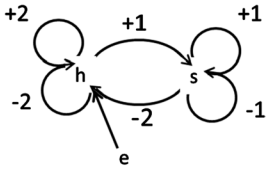
The accuracy on 3 of the 4 problems suggest that the data may contain asynchronous changes. The result with additionally learning *overrides*/4 rectifies this, as shown below:

S.No.	Name	Prec.	Recall	Accuracy	Size	Time(s)
2	IMMUNE_G	0.71	1.00	1.00	21	0.07
4	PHAGE_G	0.89	0.80	1.00	31	0.006

The theories for the generalised Kinetic Logic models are reproduced in “Appendix C”, Differences exist between the actual model and the model identified. The tabulation above suggests that deficiencies in prediction—such as they are—appear to be with models in generalised Kinetic Logic (models IMMUNE_G and PHAGE_G). The actual interactions and those obtained from the models identified for these cases are shown below:

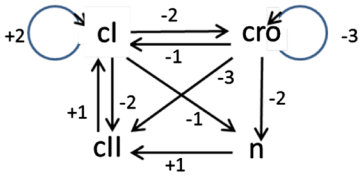


Actual

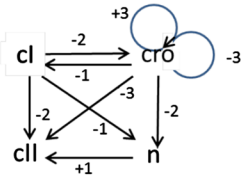


Identified

IMMUNE_G



Actual



Identified

PHAGE_G

The structures on the left are representations of the models for immune response and phage infection in generalised Kinetic Logic. For the immune system h and s denote the presence of cells from the T_h and T_s compartments, and for the phage infection, *cI*, *cro*, *cII* and *n* denote products of the genetic regulators *CI*, *Cro*, *CII* and *N* respectively. For models in generalised Kinetic Logic, a $+n$ on an edge from x to y denotes that $x \geq n$ exerts a positive interaction on y. A value of $-n$ on an edge from x to y denotes that $\overline{x \geq n}$ exerts a negative interaction on y. The edges in these models are all conjunctions.

To understand possible deficiencies in our approach, we focus here on the PHAGE_G model, since this is the most complex model identified in this paper. The tabulation suggests two kinds of errors. Errors of omission result in interaction edges not being identified, leading to reduced *Recall*. Errors of commission result in additional interaction edges being identified, leading to reduced *Precision*. For PHAGE_G, the first error is due to lack of data, and the second is an artefact of some shortcomings in the interaction diagram. We consider each in turn.

The interaction edge not identified is the autocatalytic loop (self-loop) on *cI*. It is relevant to note the following discussion in Thomas and D'Ari (1990), p. 242 on this:

> The positive loop [which] provides indirect autocatalysis of *cI* ...Knowing that it exists, we must now go back one step and ask whether it is still necessary to postulate a direct positive control of *cI* on itself. The fact that this loop has been demonstrated experimentally does not mean that it is necessary to account for the observed behaviour: it could be a case of "belt and suspenders". However, it is, in fact, essential. In the absence of the cI^+ loop, we cannot account for the block of *CI* expression at high temperatures ...

The data used here do not contain temperature-specific expression of *CI*: it is therefore unsurprising that autocatalysis of *cI* is not discovered. We would expect inclusion of pathways with temperature values of *low* or *high* will allow the identification of this edge.

The interaction diagram from the hypothesised model shows two auto-catalytic loops on *cro*, one of which is apparently spurious. The paradox of proposing both "+3" and a "-3" auto-catalytic loops on *cro* is resolved if we consider the model with the multi-level values for $f(cro)$ (that is, the levels for the gene complex *Cro*). It then becomes clear that the "+3"

loop is used to raise the expression level of $f(cro)$ from 2 to 3; and the "-3" loop is used to lower the expression level of $f(cro)$ from 3 to 2. In fact, this is the correct behaviour of the lysis pathways in Thieffry and Thomas (1995). All these pathways end with an oscillation between $cro = 2$ and $cro = 3$. Correctly, therefore, this is not a case of lower precision: the original interaction diagram in Thieffry and Thomas (1995) does not capture this oscillatory behaviour. The extra edges in the interaction diagram for IMMUNE_G can similarly be explained: we do not pursue this further here.

7 Related work

There is a very large literature on modelling in systems biology and, increasingly, machine learning is viewed as a method to acquire such models from data (see, e.g., de Jong, 2002; Klipp et al., 2016; Delgado & Gomez-Vela, 2019). However, most current approaches lack the ability to learn *executable* models, which has been proposed as a key requirement for models that can assist in understanding biological systems (Clarke & Fisher, 2020).

Logic-based approaches to event representation and modelling have the advantage of a formal, declarative semantics compared to process or stream-based systems that have informal or procedural semantics, which can be important for validation and explainability (Artikis et al., 2010).

The Event Calculus has many variants, e.g., for online reasoning (Artikis et al., 2012) and probabilistic inference (Artikis et al., 2019); see Mueller (2008) for an overview. In the Event Calculus, due to the default assumption of persistence (Shanahan 1999) of fluents holding values, it is not necessary to specify the entire state vector of the system at every time instant. Two further advantages of the Event Calculus are that it enables implementing temporal reasoning in a non-monotonic way (the so-called "common-sense law of inertia" using the closed-world assumption as in logic programming), as well as domain-specific knowledge in a declarative way (Shanahan 1999).

In the so-called "normal" setting ILP can be viewed in terms of theory revision via the operations of generalisation or specialisation of an existing theory (Nienhuys-Cheng & de Wolf, 1997). When the task is to generalise or "complete" an existing theory by adding clauses this has been referred to as "theory completion", with a particular focus within ILP on the learning of non-observational predicates, i.e., predicates for which instances are not given in the training set (Muggleton & Bryant, 2000). Learning an Event Calculus logic program fits naturally into the framework of theory completion, since there is an existing theory (the Event Calculus axioms) and typically the goal is to learn the domain-specific axioms, where these clauses are not restricted to be definitions of observational predicates (Moyle & Muggleton, 1997). Deduction or abduction can be used to derive additional instances to add to the training examples, which can then be used in induction (Kakas & Michael, 2020). Such approaches have often been proposed for learning in dynamic domains, where the problem has the following characteristics: observed data tend to be values of system variables over time, the task usually requires learning (non-observational) predicates defining how the observed data represents the effects of events or actions occurring in components of the system (non-observational predicates), and available background knowledge on the system structure is incomplete (Muggleton & Bryant, 2000; Moyle, 2003; Ray, 2008; Akutsu et al., 2009; Inoue et al., 2013).

Several such approaches study applications to biological systems, such as metabolic networks (Muggleton & Bryant, 2000; Ray, 2008), signalling networks (Inoue et al., 2013)

and gene regulatory networks (Akutsu et al., 2009), but not in the Event Calculus. Using abduction, the work by Maimari et al. (2014) identified logical models for regulatory networks. An Event Calculus representation was used to model a system of metabolic reasoning and learn definitions of the "*happens*" predicate, although without the ability to learn delays, in XHAIL (Ray, 2008). Learning the "*happens*" predicate can be thought as a form of learning non-observational predicates, in which the target predicate is not the predicate used in examples but a predicate used in the body of background clauses. This facility is available in a number of systems like XHAIL, TAL (Corapi et al., 2010), ASPAL (Corapi et al., 2012), ILED (Katzouris et al., 2015), ILASP (Law et al., 2020) and FastLAS (Law et al., 2020). The formalism of Logic Production Systems (LPS) has semantics based on an event theory, which is related to the Event Calculus, but is more oriented towards reactive applications such as agent programming, and does not include any methods for learning (Kowalski & Sadri, 2015). It will be interesting to see how to formulate learning Kinetic Logic programs within these systems, and to this extent the approach in this paper should be seen as a baseline technique rather than a definitive way to learn such programs.

A relational sequence pattern mining approach, based on a representation with several delay-type predicates where fluents representing a multi-dimensional data stream are indexed by events, was able to learn sequences defining stable states and attractors for 3D cellular automata, but did not use Event Calculus inference (Mauro et al., 2007). Event Calculus inference was used in relational reinforcement learning to enrich and constrain the exploration space with planning to reach goals, and by representing reward values as fluents could acquire domain-specific axioms about the learning task (Nickles, 2012). Business process mining using first-order decision trees (TILDE) proposed the use of an Event Calculus representation for events, but required the generation of negative events (Goedertier et al., 2007).

However the Event Calculus is known to be difficult to learn, due to the need for non-observational predicate learning where multiple predicates may need to be learned, and the dependency of clauses on negated literals with inference by negation as failure (Moyle, 2003). ILP was used to learn programs in the Event Calculus in Moyle and Muggleton (1997). However, this required changing the EC representation by merging the *initiates* and *terminates* predicates into a single predicate to allow learning in the setting of Inverse Entailment. This caused difficulties when learning definitions of domain-specific axioms, leading to subsequent work that applied SOLD-resolution to enable the derivation of abducibles to be used in learning (Moyle, 2003), although this did not study biological applications.

To scale Event Calculus learning an online version of XHAIL called OLED was developed for application to human activity recognition from streaming data (Katzouris et al., 2016). This was able to learn the definitions of the domain-specific predicates *initiated_at* and *terminated_at* in a simplified Event Calculus for real-time processing (Artikis et al., 2012), that is, to learn the conditions under which and the time at which a fluent will begin, or cease, to hold. These definitions in turn then determine the occurrence of events.

An issue here is that what is observed is not these predicates on fluents, but the "happens' predicate on events. As with earlier theory completion approaches, a key problem addressed in OLED is how to apply abduction through negation and avoid the limitation to observational predicate learning of other online learners (Katzouris et al., 2016); the online learning method was subsequently extended with a statistical relational framework in which using online gradient descent and Hoeffding bounds it was able efficiently to learn Event Calculus rules from noisy data (Katzouris et al., 2018). However, the Event Calculus representation used was the same as in OLED.

There has been a large amount of work on learning Boolean networks from data (see the methods and reviews in, e.g., (Akutsu et al., 2009; Videla et al., 2015), but this work does not involve feedback loops. Inoue and co-workers (Inoue, 2011; Inoue et al., 2014) established the very important connection between the state transition relation of dynamical systems models and the immediate consequence operator of logic programming semantics. By representing system states before and after a transition as interpretations of a logic program (Lloyd, 1987), a paradigm for identification of dynamic models called "learning from interpretation transitions" (LFIT) was introduced. Normal logic program clauses are learned to define, for each component of the successor state, the relevant conditions on the predecessor state. Clauses are indexed with a time value, so feedback and stable states or attractors (cycles of length ≥ 1) can be modelled. Since the representation is non-monotonic, both positive and negative loops can be identified. However, time delays in the system cannot be modelled.

In several follow-up works this approach was extended in number of directions, which can be viewed as a kind of parameterisation of the LFIT logical framework (Inoue et al., 2014). For example, learning with delays in the system was achieved by extending one-step transitions to allow values of system variables from up to k previous steps to be included in clauses (Ribeiro et al., 2015). This approach also used multi-valued variables, but was restricted to synchronous updates. Additionally, in Ribeiro et al. (2018) the structure of normal logic programs to compute successor states as in LFIT was preserved while changing the form of atoms to consist of variables defined on continuous domains. This has the advantage of removing the need for discretization of real-valued datasets prior to learning, which can increase error, since effectively the thresholds on variable values are determined as part of the learning process.

The LFIT logical framework has also been extended to enable learning of multi-valued logic programs with synchronous or asynchronous updates in system model dynamics (Ribeiro et al., 2018). Essentially, the asynchronous semantics imposes a restriction that no more than one variable at a time can be updated in the successor state of a transition, which differs from the more common synchronous semantics in which multiple variables can change their values (in fact, arbitrary subsets of variables whose values should change can actually be updated in the successor state—this is called the "general semantics"). This framework also allows concurrent rules, i.e., rules matching the current state that update different variables in the successor state, which permits non-deterministic transitions. However, this paper did not enable delays to be learned.

The formalism of timed automata networks enables modelling with delays, and allows for model checking, with inference of delay constraints (Ahmad et al., 2008). This was used in Ben Abdallah et al. (2017) to learn networks with delays, but unlike our approach this requires the system topology (or wiring diagram, in the form of all component interactions) to be given to the algorithm as an input. This is also required in several approaches for identification of biological system networks (as in, e.g., Akutsu et al., 2009).

8 Conclusions

Feedback loops control the dynamic behaviour of a wide-range of biological systems, most prominently those exhibiting either oscillatory control (like homeostasis) or multi-stability (like cell differentiation). In this paper, we revisited seminal work by Rene Thomas and his colleagues, that explicitly focuses on a logic-based approach to modelling biological

feedback. We have proposed a formal specification for Thomas' Kinetic Logic, and a method of implementing the specification using an Event Calculus. We have also a method of learning explanations for data using a combination of a standard form of Inductive Logic Programming and a specialised form of abduction. There are three questions that can be raised immediately.

First, why Kinetic Logic? Our motivation for the choice of Kinetic Logic is the same as the motivation proposed by Thomas, namely, the need for a formal language that is more precise than verbal descriptions employed by biologists, yet able to approximate sufficiently closely the precision afforded by differential equations. While there are several hybrid languages combining logical and differential descriptions that are particularly relevant to the modelling of cyber-physical systems (e.g., Platzer (2018)), Kinetic Logic has had nearly 4 decades of research specifically on modelling biological feedback (the early papers on Boolean-valued Kinetic Logic date to Thomas, 1977 and recent work is in Bibi et al., 2016). This has resulted in demonstrations of how a wide range of biological phenomena can be modelled. Additionally, there is clarity also on the theoretical connections to a category of differential equations (Snoussi, 1989). It is, therefore, the natural first choice to consider when constructing logical models for such systems.

Granting the choice of Kinetic Logic, why use the Event Calculus? The dynamics of a form Kinetic Logic (albeit not exactly as proposed by Thomas and colleagues) has been modelled elsewhere by the transitions of a stopwatch automaton (Ahmad et al., 2008), and formalisations exist of aspects of Kinetic Logic as a temporal logic (Bernot et al., 2004). These formalisations–in our view–do not represent some key concepts of Kinetic Logic as naturally as is done by the Event Calculus. We have in mind here: the treatment of logical functions as events acting as initiators of change values of system variables; that events triggering change can be overridden; and values persist unless a change is initiated. We are conscious, however, that naturalness of a representation is not necessary for effective modelling or effective learning of models.

Thirdly, why ILP? Logic programming is the usual choice for implementing an Event Calculus formulation, and an answer to this question is obvious enough, especially given past work on the use of ILP to learn clauses for Event Calculus programs (Moyle, 2003). But the use ILP as the vehicle for learning is not simply one forced by the choice of the Event Calculus or its implementation. Rather, our motivation for ILP stems primarily from its flexibility to use background knowledge for system identification. We have demonstrated the utility of this in our previous work on system-identification of transition systems (Bain and Srinivasan, 2018) and is shown here to some extent by the use of background knowledge defined using predicates used by the Event Calculus formulation. The use of ILP allows the incorporation of domain-knowledge, which has not been exploited in the case studies we have shown here (for example in the form of the logical functions, based on what may be biologically known about interactions between variables). Ignoring domain-knowledge, for Boolean-valued variables, any technique for hypothesising DNF formulae in propositional logic could be used to identify the logical functions. If the structure of interactions is known, then the approach in Ahmad et al. (2008) allows the synthesis of delays for a timed automaton, which in turn can be used as abductions in proofs for the fluents. The specification of an ILP system is sufficiently general to allow both of these forms of learning to be incorporated: the implementation of ILP used here

(Aleph) certainly allows this. Additionally, of course, it handles the non-Boolean case, as shown here.

This paper does not report on the identification of Kinetic Logic theories in the presence of noisy or missing data. We are nevertheless able to say something about this. Currently, we rely on the ILP and abductive machinery to deal with noisy data: usually, this results in occurrences of events being left ungeneralised at time-instants containing noise. Delays abduced for such time-instants will typically be less frequent, and therefore not selected for abduction. In some ways, this is akin to the approach taken in Bain and Srinivasan (2018) where (ultimately infrequent) transitions are invented to account for noisy data. The further step taken in Bain and Srinivasan (2018) is however not in place here, namely the use of probabilities on transitions, estimated from data. Here, this will require us to move from the classical Event Calculus to a probabilistic form of the Event Calculus (such as those developed in McAreavey et al., 2017 and Artikis et al., 2019). The current approach also has no mechanism for dealing with missing data: we expect a form of EM-like guesswork and checkwork will be needed to address this. We reinforce that in Kinetic Logic, the logical functions correspond to actual biological entities (like genes or operons), that are in principle, observable. Therefore, missing values, can be both for system-variables or the values of logical functions.

Finally, the reader may be concerned about the scalability of the approach when identifying complex systems. The complexity analysis we have in the paper indicates that important factors are the number of variables, and the number of interactions between variables. More generally, our position is similar to that taken by Thomas, in that the only scalable way to understand complex biological systems is to consider its composition by simpler sub-units. The formulation of Kinetic Logic does this by using independent definitions of logical functions each of which typically depends only a few system-variables. As we have seen in the case studies here, each logical function can represent the collective behaviour of several regulatory elements (*N* and *Cro* for example represent the action of many genes). Similarly, the system-variables can refer to more than one entity (like groups of T_h and T_s cells). The apparent simplicity of Kinetic Logic models is therefore for conceptual understanding, rather than computational convenience.

Appendix A: Correctness of the event calculus implementation of kinetic logic

Let $\sigma = (Xs, Vs, Fs, Ps, Ds, \succ)$ be a system and LT_σ be a Kinetic Logic system, with τ_p having its usual meaning. Let I_p be a consistent initialisation for τ_p (see Sect. 4.2). We refer to the definitions in Fig. 4 as the core-axioms of the event-calculus, and denote it by C. The definitions in Fig. 5 are the axioms of Kinetic Logic, and denoted by K. We assume that correct definitions of $Fs, Ps, Ds, \succ$ are encoded by *happens*/3, *pathid*/1, *delay*/4, and *overrides*/4. We collectively denote these by A_σ; and let $B = C \cup K \cup A_\sigma$. We want to establish the following:

Soundness	If $B \cup I_p \vdash holds(p, val(x_i, v_{i,k}), k)$ then $(x_i = v_{i,k}) \in s_k$
Completeness	If $(x_i = v_{i,k}) \in s_k$ then $(B \cup I_p \vdash holds(p, val(x_i, v_{i,k}), k))$

In this section, we prove the following about the implementation, for the restricted case $\succ = \emptyset$. That is, we ignore overrides, which may be justifiable in practice if the time-steps refer to sufficiently small time intervals. A proof for $\succ \neq \emptyset$ can be constructed, but involves a complex double-induction step (over ordering of variables, and over time). We do not present this here.

Before proceeding to the proof, we will introduce the following macros and use them in proving the correctness of the implementation.

initiates$(p, inc, val(x, v), [j, k])$ denotes *initiates*$(p, occurs(f(x), \cdot), val(x, v), [j, k]) \wedge$
holds $(p, val(x, v_1), k-1) \wedge v = v_1 + 1$

initiates$(p, dec, val(x, v), [j, k])$ denotes *initiates*$(p, occurs(f(x), \cdot), val(x, v), [j, k]) \wedge$
holds $(p, val(x, v_1), k-1) \wedge v = v_1 - 1$

initiates$(p, std, val(x, v), [j, k])$ denotes *initiates*$(p, tick, val(x, v), [j, k]) \wedge$
holds $(p, val(x, v_1), k-1) \wedge v = v_1$

We note the following remarks before proceeding to prove the correctness of the implementation. The remarks assume a system σ, a Kinetic Logic system LT_σ. $B = C \cup K \cup A_\sigma$ and I_p is a consistent initialisation (see Sect. 4.2).

Remark 6 *(Consequences of* $\succ = \emptyset$*)* If $\succ$ is the $\emptyset$, the following hold:

- *terminates*$(p, occurs(f(x), v'_j), val(x, v_k), [j, k])$ iff *fchanges*$(p, f(x), j, k-1)$.
- if $B \cup I_p \vdash$ *initiates*$(p, inc, val(x_i, v_{i,k}), [j, k])$ then $B \cup I_p \vdash$ *terminates*$(p, tick, val(x_i, v), [k-1, k])$ for all v.
- if $B \cup I_p \vdash$ *initiates*$(p, dec, val(x_i, v_{i,k}), [j, k])$ then $B \cup I_p \vdash$ *terminates*$(p, tick, val(x_i, v), [k-1, k])$ for all v.

Remark 7 *(Proof constraints for initiates/4)*

1. The refutation proof for *initiates*$(p, inc, val(x_i, v_{i,k+1}), [j, k+1])$ from $B \cup I_p$ will only succeed if the following constraints hold:
   - *delay*$(p, x_i, +1, d) \in B$ such that $j + d = k + 1$; and
   - $B \cup I_p \vdash$ *rate*$(p, x_i, k+1, d, r_{k+1-d}) \wedge r_{k+1-d} > 0$
   - $B \cup I_p \vdash$ *rate*$(p, x_i, k+1, 1, r_k) \wedge r_k > 0$
   - $B \cup I_p \vdash$ *holds*$(p, val(x_i, v_{i,k}), k) \wedge v_{i,k+1}$ is $v_{i,k} + 1$
   - $B \cup I_p \vdash$ *happens*$(p, occurs(f(x_i), v'_{i,j}), j)$ for some $v'_{i,j}$
2. The refutation-proof for *initiates*$(p, dec, val(x_i, v_{i,k+1}), [j, k+1])$ from $B \cup I_p$ will only succeed if the following constraints hold:
   - *delay*$(p, x_i, -1, d) \in B$ such that $j + d = k + 1$; and
   - $B \cup I_p \vdash$ *rate*$(p, x_i, k+1, d, r_{k+1-d}) \wedge r_{k+1-d} < 0$
   - $B \cup I_p \vdash$ *rate*$(p, x_i, k+1, 1, r_k) \wedge r_k < 0$
   - $B \cup I_p \vdash$ *holds*$(p, val(x_i, v_{i,k}), k) \wedge v_{i,k+1}$ is $v_{i,k} - 1$
   - $B \cup I_p \vdash$ *happens*$(p, occurs(f(x_i), v'_{i,j}), j)$ for some $v'_{i,j}$
3. The refutation-proof for *initiates*$(p, std, val(x_i, v_{i,k+1}), [j, k+1])$ from $B \cup I_p$ will only succeed if the following constraint holds:
   - $B \cup I_p \vdash$ *holds*$(p, val(x_i, v_{i,k}), k) \wedge v_{i,k+1}$ is $v_{i,k}$

We will need the following on the correctness of *fchanges*/4 for establishing a proof of correctness of the implementation.

Lemma 1 (Correctness of *fchanges*/4) *Let I_p be a consistent initialisation of τ_p. $B \cup I_p \vdash fchanges(p, f(x_i), [j,k])$, iff $FuncChanges(\tau_p, X_i, [j,k])$ is true.*

Proof We assume $FuncChanges(\tau_p, X_i, [j,k])$ is true and prove $B \cup I_p \vdash fchanges(p, f(x_i), [j, k])$. Since $FuncChanges(\tau_p, X_i, [j,k])$ is true there exists l such that $j \leq l < k$, $(X_i = v'_{i,l}) \in s_l$, and $v'_{i,l} \neq v'_{i,k}$ where $(X_i = v'_{i,k}) \in s_k$. Since we assume *happens*/3 is correct, $B \cup I_p \vdash happens(p, occurs(f(x_i), v'_{i,l}), l) \wedge happens(p, occurs(f(x_i), v'_{i,k}), k)$. Hence by using the first axiom for *fchanges*, we can conclude $B \cup I_p \vdash fchanges(p, f(x_i), [l, k])$. Now using the second axiom of *fchanges*, we can derive $fchanges(p, f(x_i), [j,k])$ from $B \cup I_p$.

We prove the other direction by induction on $k - j$.

Base case $k - j = 1$. In this case only the first axiom of *fchanges* can be used as the second axiom of *fchanges* cannot be used after one unfolding. Hence $B \cup I_p \vdash happens(p, occurs(f(x_i), v'_{i,j}), j) \wedge happens(p, occurs(f(x_i), v'_{i,k}), k)$ and $v'_{i,j} \neq v'_{i,k}$. Hence $(X_i = v'_{i,j}) \in s_j$ and $(X_i = v'_{i,k}) \in s_k$. Hence $FuncChanges(\tau_p, X_i, [j,k])$ is true.

Induction step Assume the statement for $k - j < n$ and $B \cup I_p \vdash fchanges(p, f(x_i), [j,k])$ and $k - j = n$. The proof is similar to the base case if the first axiom of *fchanges* is used to derive $fchanges(p, f(x_i), [j,k])$ in the last step. Suppose the second axiom of *fchanges* is used in the last step, then $B \cup I_p \vdash fchanges(p, f(x_i), [j+1, k])$. Since $k - j - 1 < n$, using induction hypothesis, we conclude that $FuncChanges(\tau_p, X_i, [j+1, k])$ is true. Hence $FuncChanges(\tau_p, X_i, [j,k])$ is true. □

Lemma 2

1. $B \cup I_p \vdash rate(p, x_i, k, d, r)$ *iff* $Rate(\tau_p, x_i, k - d) = r$.
2. $B \cup I_p \vdash initiates(p, inc, val(x_i, v_{i,k}), [j,k]) \wedge$ *not terminates*$(p, occurs(f(x_i), v'_{i,j}), val(x_i, v_{i,k}), [j,k])$ *for some* $v_{i,k}, v'_{i,j}$ *and* j *iff* $Increase(\tau_p, x_i, k)$ *is true*.
3. $B \cup I_p \vdash initiates(p, dec, val(x_i, v_{i,k}), [j,k]) \wedge$ *not terminates*$(p, occurs(f(x_i), v'_{i,j}), val(x_i, v_{i,k}), [j,k])$ *for some* $v_{i,k}, v'_{i,j}$ *and* j *iff* $Decrease(\tau_p, x_i, k)$ *is true*.
4. $B \cup I_p \vdash initiates(p, std, val(x_i, v_{i,k-1}), [k-1,k]) \wedge$ *not terminates*$(p, tick, val(x_i, v_{i,k-1}), [k-1,k])$ *for some* $v_{i,k-1}$ *iff* $VarChanges(\tau_p, x_i, k)$ *is false*.; *and finally*
5. $B \cup I_p \vdash holds(p, val(x_i, v_{i,k}), k)$ *iff* $(x_i = v_{i,k}) \in s_k$.

Proof We proceed by induction on t.

Base Case ($t = 0$).

The base case for (1)–(4) follow trivially since the antecedent is false in all cases, due to calls at time-points < 0 for the *holds*/3 predicate. We omit these proofs here, and focus instead on proving the base case for (5).

We want to show that if $B \cup I_p \vdash holds(p, val(x_i, v_{i,0}), 0)$, then $(x_i = v_{i,0}) \in s_0$. If $B \cup I_p \vdash holds(p, val(x_i, v_{i,0}), 0)$, then this will be the result of a refutation-proof for $\rightarrow holds(p, val(x_i, v_{i,0}), 0)$ using $B \cup I_p$. It is evident that such any such refutation-proof must involve either $C1$ or $C2$. However, the proof cannot involve $C2$, since that a successful refutation-proof will require $t_k > 0$. Thus the proof must use $C1$. It is evident that this requires a refutation of the goal $\rightarrow initially(p, val(x_i, v_{i,0}))$. Since $B \nvdash initially(p, val(x_i, v_{i,0}))$, and I_p is a set of ground facts, it must follow that

initially$(p, val(x_i, v_{i,0})) \in I_p$. Since I_p is a consistent initialisation of τ_p, it follows that $(x_i = v_{i,0}) \in s_0$.

Hypothesis ($t \leq k$). We assume the statements (1)–(5) are true when $t \leq k$.

Induction ($t = k + 1$). We prove the statements for $t = k + 1$:

1. Let us assume $B \cup I_p \vdash rate(p, x_i, k+1, d, r)$ and prove that $Rate(\tau_p, x_i, k+1-d) = r$. $rate(p, x_i, k+1, d, r)$ can be derived from $B \cup I_p$ only if $B \cup I_p \vdash happens(p, occurs(f(x_i), v'_{i,l}), l) \wedge holds(p, val(x_i, v_{i,l}), l)$ where $l = k+1-d$ and $r = v'_{i,l} - v_{i,l}$. Using induction hypothesis for holds for $l = k+1-d$, we know that $x_i = v_{i,l} \in s_l$. Since we assume *happens*/3 is correct, $X_i = v'_{i,l} \in s_l$. Hence net rate change is $v'_{i,l} - v_{i,l}$ which is r.

   Now we will assume that $Rate(\tau_p, x_i, k+1-d) = r$. It means $x_i = v_{i,k+1-d}, X_i = v'_{i,k+1-d}$ and $r = v'_{i,k+1-d} - v_{i,k+1-d}$ for some $v_{i,k+1-d}$ and $v'_{i,k+1-d}$. Since $k+1-d < k+1$ and $x_i = v_{i,k+1-d}$, we can conclude $B \cup I_p \vdash holds(p, val(x_i, v_{i,k+1-d}), k+1-d)$ using induction hypothesis for holds. Since we assume *happens*/3 is correct $B \cup I_p \vdash happens(p, occurs(f(x_i), v'_{i,k+1-d}), k+1-d)$. From axiom for rate, $B \cup I_p \vdash rate(p, x_i, k+1, d, r)$.
2. Let us assume $B \cup I_p \vdash initiates(p, inc, val(x_i, v_{i,k+1}), [j, k+1]) \wedge not\ terminates(p, occurs(f(x_i), v'_{i,j}), val(x_i, v_{i,k+1}), [j, k+1])$ and prove $Increase(\tau_p, x_i, k+1)$ is true. From Remark 7 and induction hypothesis: (a) $Rate(\tau_p, x_i, k) > 0$; and (b) there exists $(p, x_i, d, \cdot) \in Ds$ of σ, such that $j + d = k + 1s$ and $Rate(\tau_p, x_i, k+1-d) > 0$; and (c) $Order(\tau_p, x_i^+, [j, k+1])$ is true. Since $B \cup I_p \vdash not\ terminates(p, occurs(f(x_i), v'_{i,j}), val(x_i, v_{i,k+1}), [j, k+1])$ and *terminates* is ground, $B \cup I_p \nvdash terminates(p, occurs(f(x_i), v'_{i,j}), val(x_i, v_{i,k+1}), [j, k+1])$. From Remark 6, $B \cup I_p \nvdash fchanges(p, f(x_i), [j, k])$ is true. From Remark 1, (d) $FuncChanges(\tau_p, X_i, [j, k])$ is false. From (a)–(d), $Increase(\tau_p, x_i, k+1)$ is true.

   Let us assume that $Increase(\tau_p, x_i, k+1)$ is true and prove $B \cup I_p \vdash initiates(p, inc, val(x_i, v_{i,k+1}), [j, k+1]) \wedge not\ terminates(p, occurs(f(x_i), v'_{i,j}), val(x_i, v_{i,k+1}), [j, k+1])$ for some $v_{i,k+1}, v'_{i,j}$ and j. Since $Increase(\tau_p, x_i, k+1)$ is true, there exists $(p, x_i, d, \cdot) \in Ds$ such that $Rate(\tau_p, x_i, k+1-d) > 0$, $Rate(\tau_p, x_i, k) > 0$, and $FuncChanges(\tau_p, x_i, [k+1-d, k+1])$ is false. Since Ds is encoded by *delay* in B, $B \cup I_p \vdash delay(p, x_i, +1, d)$. By induction hypothesis, $B \cup I_p \vdash rate(p, x_i, k+1, 1, r) \wedge r > 0$ and $B \cup I_p \vdash rate(p, x_i, k+1, d, r_1) \wedge r_1 > 0$. Let $(x_i = v_{i,k}) \in s_k$ for some $v_{i,k}$. Hence by induction hypothesis, $B \cup I_p \vdash holds(p, val(x_i, v_{i,k}), k)$. Let $v_{i,k+1} = v_{i,k} + 1$ and $j = k + 1 - d$. Since $X_i = v'_{i,j} \in s_j$ for some $v'_{i,j}$, $B \cup I_p \vdash happens(p, occurs(f(x_i), v'_{i,j}), j)$. Hence $B \cup I_p \vdash initiates(p, occurs(f(x_i), v'_{i,j}), val(x_i, v_{i,k+1}), [j, k+1])$. Since $B \cup I_p \vdash holds(p, val(x_i, v_{i,k}), k), v_{i,k+1} = v_{i,k} + 1, B \cup I_p \vdash initiates(p, inc, val(x_i, v_{i,k+1}), [j, k+1])$. By Remark 1, $B \cup I_p \nvdash fchanges(p, f(x_i), [k+1-d, k])$. By Remark 6, $B \cup I_p \vdash not\ terminates(p, occurs(f(x_i), v'_{i,j}), val(x_i, v_{i,k+1}), [j, k+1])$.
3. The proof for $B \cup I_p \vdash initiates(p, dec, val(x_i, v_{i,k}), [j, k]) \wedge not\ terminates(p, occurs(f(x_i), v'_{i,j}), val(x_i, v_{i,k}), [j, k])$ for some $v_{i,k}, v'_{i,j}$ and j iff $Decrease(\tau_p, x_i, k)$ is similar to the above.
4. Let us assume $B \cup I_p \vdash initiates(p, std, val(x_i, v_{i,k}), [k, k+1]) \wedge not\ terminates(p, tick, val(x_i, v_{i,k}), [k, k+1])$ (that is, the proof search for $terminates(p, tick, val(x_i, v_{i,k}), [k, k+1])$ from $B \cup I_p$ finitely fails). We show $VarChanges(\tau_p, x_i, k+1)$ is false by contradiction. If $VarChanges(\tau_p, x_i, k+1)$ is true then $Increase(\tau_p, x_i, k+1)$ is true or $Decrease(\tau_p, x_i, k+1)$ is true. We consider the case when $Increase(\tau_p, x_i, k+1)$ is true and the proof is similar when $Decrease(\tau_p, x_i, k+1)$ is true. Since $Increase(\tau_p, x_i, k+1)$ is true, from 2 there exists $(p, x_i, d, \cdot) \in Ds$ such that $B \cup I_p \vdash initiates(p, inc, val(x_i, v_{i,k+1}), [k+1-d, k+1])$. Using Remark 6, $B \cup I_p \vdash terminates(p, tick, val(x_i, v_{i,k}), [k, k+1])$, which is a contradiction.

Now we assume $VarChanges(\tau_p, x_i, k+1)$ is false and prove $B \cup I_p \vdash initiates(p, std, val(x_i, v_{i,k}), [k, k+1]) \wedge not\ terminates(p, tick, val(x_i, v_{i,k}), [k, k+1])$. Since $VarChanges(\tau_p, x_i, k+1)$ is false, either $PendingOrder(\tau_p, x_i, k+1)$ is false or $Overridden(\tau_p, x_i, k+1)$ is true. Since $\succ$ is empty, $Overridden(\tau_p, x_i, k+1)$ is false. So $Increase(\tau_p, x_i, k+1)$ is false and $Decrease(\tau_p, x_i, k+1)$ is false. So if $x_i = v_{i,k} \in s_k$ and $x_i = v_{i,k+1} \in s_{k+1}$, then $v_{i,k+1} = v_{i,k}$. By induction hypothesis $B \cup I_p \vdash holds(p, val(x_i, v_{i,k}), k)$. By using axiom $KI3$, $B \cup I_p \vdash initiates(p, std, val(x_i, v_{i,k}), [k, k+1])$. Since $Increase(\tau_p, x_i, k+1)$ is false and $Decrease(\tau_p, x_i, k+1)$ is false, from 2 and 3, we can conclude for every $v_{i,k}, v'_{i,j}$ and j, either $B \cup I_p \nvdash initiates(p, occurs(f(x_i), v'_{i,j}), val(x_i, v_{i,k}), [j, k+1])$ or $B \cup I_p \nvdash not\ terminates(p, occurs(f(x_i), v'_{i,j}), val(x_i, v_{i,k}), [j, k+1])$. So it is not possible to derive $terminates(p, tick, val(x_i, v_{i,k}), [k, k+1])$ using axiom $KT3$. Hence $B \cup I_p \vdash not\ terminates(p, tick, val(x_i, v_{i,k}), [k, k+1])$.

5. Let us assume $B \cup I_p \vdash holds(p, val(x_i, v_{i,k+1}), k+1)$ and prove $(x_i = v_{i,k+1}) \in s_{k+1}$. Since $k+1 > 0$, any refutation-proof for $holds/3$ has to start with $C2$. Let $B \cup I_p \vdash initiates(p, Event, val(x_i, v_{i,k+1}), [T, k+1])\theta$ and $B \cup I_p \vdash not\ clipped(p, Event, val(x_i, v_{i,k+1}), [T_0, k+1])\theta$ where θ will be one of the following ground substitutions:
$\theta_1 = \{Event/occurs(f(x_i), v'_{i,j}), j), T_0/j\}, (j \leq k)$; or $\theta_2 = \{Event/tick, T_0/j\}\,(j = k)$.
We consider each of the θ's in turn.
$\theta = \theta_1$ That is $B \cup I_p \vdash initiates(p, occurs(f(x_i), v'_{i,j}), val(x_i, v_{i,k+1}), [j, k+1])$. Since $B \cup I_p \vdash not\ terminates(p, occurs(f(x_i), v'_{i,j}), val(x_i, v_{i,k+1}), [j, k+1])$, $FuncChanges(\tau_p, X_i[j, k+1])$ is false. So there will be only two cases to be considered: $v'_{i,j} > v_{i,k+1} - 1$ and $v'_{i,j} < v_{i,k+1} - 1$. If $v'_{i,j} > v_{i,k+1} - 1$, there will be $v_{i,k}$ such that $B \cup I_p \vdash holds(p, val(x, v_{i,k}), k) \wedge v_{i,k+1} = v_{i,k} + 1$. Hence $B \cup I_p \vdash initiates(p, inc, val(x_i, v_{i,k+1}), [j, k+1])$. From 2, this implies $Increase(\tau_p, x_i, k+1)$ is true. From Remark 7, $B \cup I_p \vdash holds(p, val(x_i, v_{i,k}), k)$. By the induction hypothesis for $holds/3$ at k, $(x_i = v_{i,k}) \in s_k$. Since $B \cup I_p \vdash not\ clipped(p, occurs(f(x_i), v'_{i,j}, val(x_i, v_{i,k+1}), [j, k+1])$, $B \cup I_p \vdash not\ overridden(p, x_i, k+1)$. Hence, $Overridden(\tau_p, x_i, k+1)$ is false. Since $Increase(\tau_p, x_i, k+1)$ is true, and $Overridden(\tau_p, x_i, k+1)$ is false, $VarChanges(\tau_p, x_i, k+1)$ is true. It is given that LT_σ is a Kinetic Logic system. Using Definition 13 we conclude $(x_i = v_{i,k} + 1) \in s_{k+1}$. From Remark 7, $v_{i,k+1} = v_{i,k} + 1$, and therefore $(x_i = v_{i,k+1}) \in s_{k+1}$. The proof for $v'_{i,j} < v_{i,k+1} - 1$ is similar.$\theta = \theta_2$

That is, $B \cup I_p \vdash initiates(p, tick, val(x_i, v_{i,k+1}), [k, k+1])$ which is same as $B \cup I_p \vdash initiates(p, std, val(x_i, v_{i,k+1}), [k, k+1]) \wedge holds(p, val(x_i, v_{i,k}), k) \wedge v_{i,k+1} = v_{i,k}$.

By induction hypothesis for $holds/3$, $(x_i = v_{i,k}) \in s_k$. Since $B \cup I_p \vdash not\ clipped(p, tick, val(x_i, v_{i,k+1}), [k, k+1])$, and using $C3$, $B \cup I_p \vdash not\ terminates(p, tick, val(x_i, v_{i,k+1}), [k, k+1])$. Hence $B \cup I_p \vdash initiates(p, std, val(x_i, v_{i,k+1}), [k, k+1]) \wedge not\ terminates(p, std, val(x_i, v_{i,k+1}), [k, k+1])$. Hence by claim 4, $VarChanges(\tau_p, x_i, k+1)$ is false. It is given that LT_σ is a Kinetic Logic system. Using Definition 13 we conclude $(x_i = v_{i,k} + 1) \in s_{k+1}$. From Remark 7, $v_{i,k+1} = v_{i,k}$, and therefore $(x_i = v_{i,k+1}) \in s_{k+1}$.

Let us assume that $x_i = v_{i,k+1} \in s_{k+1}$ and prove that $B \cup I_p \vdash holds(p, val(x_i, v_{i,k+1}), k+1)$. Let $x_i = v_{i,k} \in s_k$ for some $v_{i,k}$. There are three cases to consider:

(a) $(v_{i,k+1} = v_{i,k} + 1)$. By induction hypothesis $B \cup I_p \vdash holds(p, val(x_i, v_{i,k}), k)$. We have to show $B \cup I_p \vdash initiates(p, occurs(f(x_i), v'_{i,j}), val(x_i, v_{i,k+1}), [j, k+1]) \wedge not\ clipped(p, occurs(f(x_i), v'_{i,j}), val(x_i, v_{i,k+1}), [j, k+1])$ for some $j, v'_{i,j}$. Since $v_{i,k+1} = v_{i,k} + 1$: $VarChanges(\tau_p, x_i, k+1)$ is true; and $Increase(\tau_p, x_i, k+1)$ is true. That is, there exists $(p, x_i, d, \cdot) \in Ds$ s.t. $Rate(\tau_p, x_i, k+1-d) > 0$ is true. That is, if $j = k+1-d$, $(X_i = v'_{i,j}) \in s_j$ and $(x_i = v_{i,j}) \in s_j$, then $v'_{i,j} > v_{i,j}$. By the induction hypothesis $B \cup I_p \vdash holds(p, val(x_i, v_{i,j}), j)$. Since *happens*/3 is correct, then $B \cup I_p \vdash rate(p, x_i, k+1, d, r) \wedge r > 0$, for some r. Similarly $B \cup I_p \vdash rate(p, x_i, k+1, -1, r') \wedge r' > 0$ for some r'. So, from $KI1$, $B \cup I_p \vdash initiates(p, occurs(f(x_i), v'_{i,j}), val(x_i, v_{i,k+1}), [j, k+1])$. Since $Increase(\tau_p, x_i, k+1)$ is true, $FuncChanges(\tau_p, x_i, [j, k+1]$ is false. From Remark 1, $B \cup I_p \vdash not\ fchanges(p, f(x_i), [j, k+1])$. From Remark 6, $B \cup I_p \vdash not\ terminates(p, occurs(f(x_i), v'_{i,j}), val(x_i, v_{i,k+1}), [j, k])$ and $B \cup I_p \vdash not\ clipped(p, occurs(f(x_i), v'_{i,j}), val(x_i, v_{i,k+1}), [j, k])$. Using $C2$ $B \cup I_p \vdash holds(p, val(x_i, v_{i,k+1}), k+1)$.

(b) $(v_{i,k+1} = v_{i,k} - 1)$. The proof for $B \cup I_p \vdash holds(p, val(x_i, v_{i,k+1}), k+1)$ is similar to the case above.

(c) $(v_{i,k+1} = v_{i,k})$. First we will show $B \cup I_p \vdash initiates(p, tick, val(x_i, v_{i,k+1}), [k, k+1]) \wedge not\ clipped(p, tick, val(x_i, v_{i,k+1}), [k, k+1])$. Since $x_i = v_{i,k} \in s_k$, by the Induction hypothesis, $B \cup I_p \vdash holds(p, val(x_i, v_{i,k}), k)$. Therefore by using axiom $KI3$, we conclude that $B \cup I_p \vdash initiates(p, tick, val(x_i, v_{i,k}), [k, k+1])$. Suppose $B \cup I_p \vdash terminates(p, tick, val(x_i, v_{i,k+1}), [k, k+1])$. Hence $B \cup I_p \vdash initiates(p, occurs(f(x_i)v'_{i,j}), val(x_i, v_{i,k}), [j, k+1]) \wedge not\ terminates(p, occurs(f(x_i), v'_{i,j}), val(x_i, v_{i,k}), [j, k+1])$ for some $v'_{i,j}$ and j. So by 2, $Increase(\tau_p, x_i, [j, k+1])$ is true. Since $Overridden(\tau_p, x_i, k+1)$ is always false, $VarChanges(\tau_p, x_i, k+1)$ is true. Using the fact LT_σ is a kinetic logic system, it is easy to see $v_{i,k+1} = v_{i,k} + 1$ which is a contradiction to our assumption $v_{i,k+1} = v_{i,k}$. Hence $B \cup I_p \vdash not\ terminates(p, tick, val(x_i, v_{i,k+1}), [k, k+1])$ and $B \cup I_p \vdash not\ clipped(p, std, val(x_i, v_{i,k+1}), [k, k+1])$. Now by using $C2$, we conclude that $B \cup I_p \vdash holds(p, val(x_i, v_{i,k+1}), k+1)$.

□

The proof of Theorem 1 follows directly from the lemma.

Appendix B: Implementation details

We first note some details introduced for efficiency into the implementations of Procedures 1–3: (a) We use a depth-bounded resolution-based theorem-prover; (b) We use an efficient form of the axioms in Figs. 4, 5: the main change is the use of a cached-form of *holds*/3 and *happens*/3 that avoids repeated theorem-proving over time-instants; and (c) We only consider constructing *overrides*/4 definitions if the abduction of delays does not explain of all the fluents (otherwise $O = \emptyset$).

Procedure 3 in this paper assumes the availability of a standard ILP engine (in our case, Aleph Srinivasan, 1999). Here, we provide some practical details concerned with the use of the ILP engine. In the following, we use a Prolog-like syntax, interspersed with some logical constructs for clarity.

Appendix B.1: Representation

We will assume that data are available as time-stamped sequences of values of system-variables x_i along with values of the functions F_i. Each such sequence represents a conjunction of events and fluents thus:

Data:

```
sequence(p1,[
    0-[x=0,y=0,f(x)=1,f(y)=1],
    1-[x=1,y=1,f(x)=0,f(y)=0],
    ...
    ...
    ]).
```

(a) Raw data for a $p1$-trace

Fluents:

$$holds(p1, val(x, 0), 0) \ \wedge \ holds(p1, val(y, 0), 0) \ \wedge$$
$$holds(p1, val(x, 1), 1) \ \wedge \ holds(p1, val(y, 1), 1) \ \wedge$$
$$\ldots$$

Events:

$$happens(p1, occurs(f(x), 1), 0) \ \wedge \ happens(p1, occurs(f(y), 1), 0) \ \wedge$$
$$happens(p1, occurs(f(x), 0), 1) \ \wedge \ happens(p1, occurs(f(y), 0), 1) \ \wedge$$
$$\ldots \qquad happens(tick, 0) \ \wedge \ happens(tick, 1) \ \wedge$$
$$\ldots$$

(b) Data as interpreted by the ILP engine

Background knowledge, besides the definitions in Figs. 4 and 5, contains auxiliary definitions (A_σ in Sect. 5.1 and Procedure 3) which include the following: (a) Definitions that capture positive and negative interactions (defined in terms of the values of fluents); (b) Specification of system-variables; (c) Specification of variables external to the system, along with the predicate *has_value*/4 to represent the actual values of external variables; (d) Specification of the set of abducible values for delays.

$$pos(P, val(Var, V), T) \leftarrow$$
$$\quad Fluent = val(Var, V1),$$
$$\quad holds(P, Fluent, T),$$
$$\quad V1 \geq V.$$

$$neg(P, val(Var, V), T) \leftarrow$$
$$\quad Fluent = val(Var, V1),$$
$$\quad holds(P, Fluent, T),$$
$$\quad V1 < V.$$

(a) Predicates for positive and negative interactions in A_σ

$$domain\_element(x, [0, 1]) \ \wedge$$
$$domain\_element(y, [0, 1]) \ \wedge$$
$$external\_element(e, [0, 1]) \ \wedge$$

$$delays(p1, x, +1, [1, 2, 3]) \ \wedge$$
$$delays(p1, x, -1, [1, 2, 3]) \ \wedge$$
$$delays(p1, y, +1, [1, 2, 3]) \ \wedge$$
$$delays(p1, y, -1, [1, 2, 3]) \ \wedge$$

$$has\_value(p1, e, 1, T)$$

(b)–(d) Specification of system-variables; external-variables; and abducible delays in A_σ

Appendix B.2: Abduction and induction

The specification of *delays* allows us to implement the set of abducibles allowed A_p in Procedure 1. For a partially-specified system $\sigma = (Xs, Vs, \cdot, Ps, \cdot, \cdot)$, for variable $x \in Xs$ and $p \in Ps$, abducible delays are simply combinations of values specified for ± 1. Identifying H_E in Procedure 3 requires the ILP engine to construct clausal explanations for the *happens*/3 facts in the data, using the fluent facts (F) and any auxiliary predicate definitions relevant to identifying the system σ in A_σ. Examples of such explanations are (the explanations do not relate to the example data in (a) above):

$$happens(P, occurs(f(x), 1), T) \leftarrow pos(P, val(y, 1), T)$$

$$happens(P, occurs(f(y), 1), T) \leftarrow neg(P, val(x, 1), T), has\_value(P, e, 1, T)$$

$$\ldots$$
$$\ldots$$

(e) Explanations for events for simple Kinetic Logic

$$happens(P, occurs(f(x), 2), T) \leftarrow pos(P, val(x, 2), T)$$

$$happens(P, occurs(f(x), 1), T) \leftarrow neg(P, val(h, 2), T), neg(P, val(s, 2), T), has\_value(P, e, 1, R)$$

$$happens(P, occurs(f(y), 2), T) \leftarrow \ldots \ldots$$

(f) Explanations of events for generalised Kinetic Logic

We are also able to use an ILP engine to compress the *delay*/4 facts constructed by Procedure 1, as shown below, which can make the definition of *delay*/4 more readable.

$$delay(p1, x, +1, 1) \wedge delay(p1, x, -1, 2) \wedge delay(p1, y, +1, 2) \wedge delay(p1, y, -1, 1)$$

(g) Abduced atoms for delays using Procedure 1.

$$delay(P, x, +1, 1) \leftarrow member(P, [p1, p3, p4])$$
$$delay(P, x, -1, 2) \leftarrow member(P, [p1, p5])$$
$$\ldots$$
$$\ldots$$

(g) Compressed version of a delay definitions using ILP

Similar compressed explanations can also be constructed for *overrides*/4.

Appendix B.2.1: ILP language bias

For all ILP experiments, the ILP system Aleph (Srinivasan, 1999) was used to construct theories for events and delays. Aleph requires the specification of a mode-language $\mathcal{L}$ in the manner described in Muggleton (1995). The following mode declarations were used to construct explanations for the case studies reported here:

```
:- modeh(1,happens(+pathid,occurs(#function,#fval),+time)).
:- modeh(1,delay(+pathid,#variable,#value,#delay)).
:- modeh(1,overrides(#pathid,#variable,#variable,#time)).

:- modeb(*,has_value(+pathid,#input,#val,+time)).
:- modeb(*,pos(+pathid,val(#var,#vval),+time)).
:- modeb(*,neg(+pathid,val(#var,#vval),+time)).
:- modeb(1,member(+pathid,#pathids)).
```

member/2 is additionally declared as a lazily-evaluated literal (see Srinivasan & Camacho, 1999). This allows the construction of compressed explanations of the kind shown above.

Appendix B.3: Extracting influence diagrams

We obtain interaction diagrams from logical explanations as follows. For any system-variable x, we consider clauses of the form $happens(P, occurs(f(x), N), T) \leftarrow Body$, where $N > 0$. For each such clause (the definitions for *pos*/3 and *neg*/3 are as described above):

1. If *Body* has a literal $pos(P, val(y, V), T)$, then we add an edge from y to x labelled $+V$;
2. If *Body* has a literal $neg(P, val(y, V), T)$, then we add an edge from y to x labelled $-V$;
3. If *Body* has a literal $has\_value(P, e, V, T)$, then we add an edge from e to x labelled $+1$ if $e = 1$ and -1 if $e = 0$.

We note that interaction diagrams omit logical constructs (like ANDs and ORs). They also do not distinguish between different non-zero levels for logical functions. The purpose of an interaction diagram is simply to identify pairwise interactions for display in a graphical form.

Appendix C: Application details

Appendix C.1: Brief descriptions of the biological systems

Immune Response

The immune system in vertebrates responds to the introduction of a foreign body or *antigen* by producing proteins called *antibodies*. Usually one or more antibodies bind, in a kind of lock-and-key manner, to specific parts of the antigen, preventing them from interacting further with the host. It is this immune reaction of antibody production that forms the basis of both vaccination (in which antibody production is deliberately triggered by injection of an antigen), or disease detection (by checking for the presence of antibodies: correctly though this does not tell us if the antigen is present, but only if the host was exposed to it at some point).

The immune system consists of a complex network of interactions, mainly involving two categories of cells. The $B-lymphocytes$ produce the actual antibodies, in stages regulated by the $T-lymphocytes$, itself consisting of sub-categories called T_h- and T_s-lymphocytes. Each category consists of multiple different cells, specialised to the production of antibodies specific to parts of the antigen. Here we use the simplified model in Thomas and D'Ari (1990), that treats each category as a functional *compartment*. In the model, the logical functions H and S denote the biological mechanism triggering the secretion of h and s cells in the T_h and T_s compartments The presence of an antigen (e) triggers the secretion of h-cells. The presence of cells is sustained by an auto-catalytic reaction. In turn, h triggers the production of antibodies by activating cells in the B-lymphocyte compartment. The presence of h also induce the secretion of s-cells, that act in turn to suppress the production of h cells. While the production of antibodies results in eliminating the antigen, h cells may persist, thus

constituting a *memory* state that can effect a quicker response if the antigen re-appears.

Phage Infection — Infection of a bacterium by a phage (virus) results in either the death of the host (lysis) with subsequent release of multiple copies of the phage; or immunity for the host (lysogenesis), resulting from the phage's DNA being integrated into the host's DNA. This combined DNA then produces a suppression of the virus's genes that result in lysis. The suppression is due to a protein produced by a viral gene: in the classic system of the bacteriophage λ that we consider here, the repressor protein is called *cI*. Since the viral genome has been integrated into the bacterial DNA, the ability to repress the infective portions of the viral genome is transferred to subsequent bacterial generations.

In the phage λ, lethal (lytic) activity is the result of positive feedback, largely from two groups of genes (operons) called *N* and *Cro*. The repressor protein encoded by the gene *CI* prevents the transcription of the operons *N* and *cro*. Immunity is in fact the result of two stages. The first, or *establishment* stage requires the products of genes *CII* and *CIII* to promote the activation of the repressor gene *CI*. Subsequent *maintenance* of immunity is controlled by auto-catalytic activity, in which moderate concentrations of the repressor protein encoded by *CI* results in increase in its own synthesis. The genes *CII* and *CIII* are regulated positively by the *N* operon and negatively by *Cro*. Each of these are in turn affected by the activations of the others, resulting in complex feedback circuitry. In the model, the logical functions *CI*, *Cro*, *CII* and *N* are defined in terms of the corresponding protein products *cI*, *cro*, *cII* and *n*.

In the experiments here, we consider both simple and generalised Kinetic Logic models for each system. The models for immunity are from Thomas and D'Ari (1990), and consist of 3 feedback loops between the T_h and T_s lymphocytes.[14] The model also contains a graft representing the presence or absence of the antigen. For the phage λ, the simple Kinetic Logic model is from Thomas (1977) and consists of 3 regulatory elements *cI*, *cro* and *N*. The generalised Kinetic model is from Thieffry and Thomas (1995), and consists of 4 regulatory elements, with 7 feedback loops. Figure 7 shows the interaction diagrams in each of these cases. The simple model for immunity consists of the following equations for the logical functions H and S: $H \equiv (h \vee (e \wedge \overline{s}))$ and $S \equiv (h \vee s)$. The generalised models are more complex. For example, the interaction diagram is consistent with the following: $(H \neq 0) \leftarrow ((h \geq 2) \vee (e \wedge \neg(s \geq 2)))$. The equations of phage λ is complicated and can be found in Thieffry and Thomas (1995).

[14] A Boolean-valued model also exists in Kaufman et al. (1985) that includes the *B*-lymphocytes. We have chosen the simpler model here since both Boolean- and multi-valued treatments are available in the literature for that model.

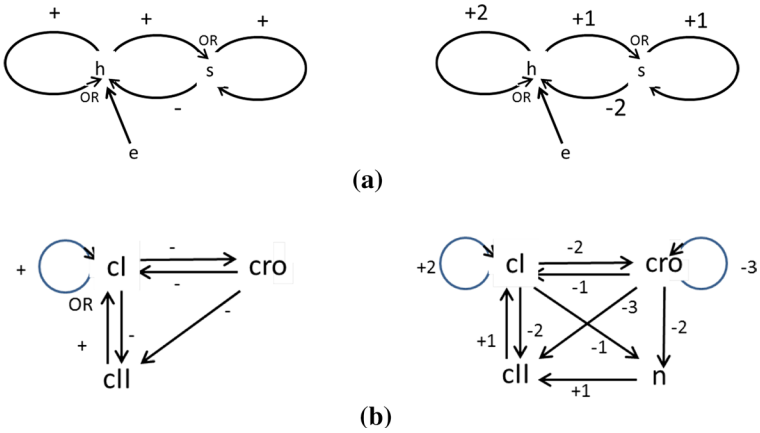


Fig. 7 Interactions for: **a** immune response (Thomas & D'Ari, 1990); and **b** phage λ (Chapter 17 of Thomas (1977), left and Thieffry and Thomas (1995), right). In each case, the structure on the left is a representation of the model in simple Kinetic Logic (all variables are Boolean), and the structure on the right is a representation of the model in generalised Kinetic Logic. For the immune system h and s denote the presence of cells from the T_h and T_s compartments, and for the phage infection, cI, cro, cII and n denote products of the genetic regulators CI, Cro, CII and N respectively. The edge-labels for the diagrams for simple Kinetic Logic represent positive (+) or negative (−) interactions. For generalised models, a $+n$ on an edge from x to y denotes that $x \geq n$ exerts a positive interaction on y. A value of $-n$ on an edge from x to y denotes that $\underline{x \geq n}$ exerts a negative interaction on y. Edges labelled with an OR indicate a disjunction. Edges without any labels indicate a conjunction

Appendix C.2: Examples of results

We show the generalised Kinetic Logic models for the two biological systems described in the paper.

Theory for IMMUNE_G:

```
happens(A,occurs(f(h),0),B):-
    neg(A,val(h,2),B),
    has_value(A,antigen,0,B).
happens(A,occurs(f(h),0),B):-
    neg(A,val(h,2),B),
    pos(A,val(s,2),B).
happens(A,occurs(f(h),2),B):-
    pos(A,val(h,2),B).
happens(A,occurs(f(h),1),B):-
    neg(A,val(h,2),B),
    neg(A,val(s,2),B),
    has_value(A,antigen,1,B).
happens(A,occurs(f(h),2),B):-
    neg(A,val(s,2),B),
    has_value(A,antigen,2,B).
happens(A,occurs(f(s),1),B):-
    neg(A,val(s,1),B),
    pos(A,val(h,1),B).
happens(A,occurs(f(s),0),B):-
    neg(A,val(h,1),B),
    neg(A,val(s,1),B).
happens(A,occurs(f(s),2),B):-
    pos(A,val(s,1),B).

delay(A,h,-1,1):-
    member(A,[e010_a,e100_a,e101_a,e201_b]).
delay(e010_b,h,-1,2):-true.
delay(e010_c,h,-1,3):-true.
delay(A,h,1,1):-
    member(A,[e100_a,e101_a,e200_a,
              e200_b,e201_a,e201_b]).
delay(A,h,1,2):-member(A,[e200_a,e201_a]).
delay(e200_b,h,1,3):-true.
delay(A,s,1,1):-
    member(A,[e010_b,e010_c,e020_a,e100_a,
              e101_b,e120_a,e200_b,e201_c]).
delay(A,s,1,3):-
    member(A,[e010_b,e200_a,e200_b,e201_a]).
delay(A,s,1,2):-
    member(A,[e010_c,e020_a,e100_a,
              e101_a,e120_a,e200_a,e201_b]).

overrides(e010_b,h,s,2):-true.
overrides(e200_b,s,h,2):-true.
overrides(e200_b,h,s,3):-true.
overrides(e201_b,s,h,2):-true.
```

Theory for PHAGE_G:

```
happens(A,occurs(f(cI),2),B):-
    neg(A,val(cro,1),B).
happens(A,occurs(f(cI),0),B):-
    pos(A,val(cro,1),B).
happens(A,occurs(f(cro),3),B):-
    neg(A,val(cI,2),B),
    neg(A,val(cro,3),B).
happens(A,occurs(f(cro),0),B):-
    pos(A,val(cI,2),B).
happens(A,occurs(f(cro),2),B):-
    pos(A,val(cro,3),B).
happens(A,occurs(f(cII),0),B):-
    neg(A,val(n,1),B).
happens(A,occurs(f(cII),1),B):-
    neg(A,val(cI,2),B),
    neg(A,val(cro,3),B),
    pos(A,val(n,1),B).
happens(A,occurs(f(cII),0),B):-
    pos(A,val(cI,2),B).
happens(A,occurs(f(cII),0),B):-
    pos(A,val(cro,3),B).
happens(A,occurs(f(n),1),B):-
    neg(A,val(cI,1),B),
    neg(A,val(cro,2),B).
happens(A,occurs(f(n),0),B):-
    pos(A,val(cI,1),B).
happens(A,occurs(f(n),0),B):-
    pos(A,val(cro,2),B).

delay(phage_4_immune,cI,1,3):-true.
delay(phage_4_immune,cI,1,4):-true.
delay(A,cro,1,1):-
    member(A,[phage_4_lysis_1,phage_4_lysis_2,
            phage_4_lysis_3]).
delay(phage_4_lysis_1,cro,1,2):-true.
delay(A,cro,1,3):-
    member(A,[phage_4_lysis_1,phage_4_lysis_2,
                phage_4_lysis_3]).
delay(A,cro,-1,1):-
    member(A,[phage_4_lysis_1,phage_4_lysis_2]).
delay(A,cro,1,4):-
    member(A,[phage_4_lysis_1,phage_4_lysis_2,
                phage_4_lysis_3]).
delay(A,cro,-1,3):-
    member(A,[phage_4_lysis_1,phage_4_lysis_2,
            phage_4_lysis_3]).
delay(phage_4_lysis_3,cro,-1,2):-true.
delay(phage_4_lysis_3,cro,-1,4):-true.
delay(phage_4_immune,cII,1,1):-true.
delay(phage_4_immune,cII,-1,2):-true.
delay(phage_4_immune,n,1,1):-true.
delay(A,n,-1,2):-
    member(A,[phage_4_immune,phage_4_lysis_3]).
delay(A,n,1,2):-
    member(A,[phage_4_lysis_2,phage_4_lysis_3]).
delay(phage_4_lysis_2,n,-1,1):-true.

overrides(phage_4_lysis_2,n,cro,2):-true.
overrides(phage_4_lysis_2,n,cro,4):-true.
overrides(phage_4_lysis_3,n,cro,2):-true.
```

Acknowledgements The authors would like to thank Krishnamachari Sriram for drawing our attention to the work of René Thomas, and to Steve Moyle for his help with the Event Calculus. AS is a visiting Professorial Fellow at the School of CSE, UNSW.

References

Ahmad, J., Roux, O., Bernot, G., Comet, J. P., & Richard, A. (2008). Analysing formal models of genetic regulatory networks with delays. *International Journal of Bioinformatics Research and Applications, 4*(3), 240–262.

Akutsu, T., Tamura, T., & Horimoto, K. (2009). Completing networks using observed data. In *Algorithmic learning theory. ALT 2009* (pp. 126–140). Springer.

Artikis, A., Makris, E., & Paliouras, G. (2019). A probabilistic interval-based event calculus for activity recognition. *Annals of Mathematics and Artificial Intelligence*.

Artikis, A., Paliouras, G., Portet, F., & Skarlatidis, A. (2010). Logic-based representation, reasoning and machine learning for event recognition. In *DEBS10: Proceedings of the fourth ACM international conference on distributed event-based systems* (pp. 282–293) (2010).

Artikis, A., Skarlatidis, A., Portet, F., & Paliouras, G. (2012). Logic-based event recognition. *The Knowledge Engineering Review, 27*(4), 469–506.

Bain, M., & Srinivasan, A. (2018). Identification of biological transition systems using meta-interpreted logic programs. *Machine Learning, 107,* 1171–1206.

Ben Abdallah, E., Ribeiro, T., Magnin, M., Roux, O., & Inoue, K. (2017). Modeling delayed dynamics in biological regulatory networks from time series data. *Algorithms,10*(8).

Bernot, G., Comet, J. P., Richard, A., & Guespin, J. (2004). Application of formal methods to biological regulatory networks: Extending Thomas' asynchronous logical approach with temporal logic. *Journal of Theoretical Biology,229,* 339–347.

Bibi, Z., Ahmad, J., Ali, A., Siddiqa, A., Shahzad, S., Tareen, S., Janjua, H. A. & Khusro, S. (2016). On the modeling and analysis of the biological regulatory network of NF-kappa B activation in HIV-1 infection. *Complex Adaptive Systems Modeling,4*(1).

Cinquin, O., & Demongeot, J. (2002). Roles of positive and negative feedback in biological systems. *C. R. Biologies, 325,* 1085–1095.

Clarke, M., & Fisher, J. (2020). Executable cancer models: Successes and challenges. *Nature Reviews Cancer*.

Corapi, D., Russo, A., & Lupu, E. (2010). Inductive logic programming as sbductive search. Leibniz international proceedings in informatics (LIPIcs). In M. Hermenegildo & T. Schaub (Eds.), *Technical communications of the 26th international conference on logic programming* (Vol. 7, pp. 54–63). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Corapi, D., A.Russo, & Lupu, E. (2012). Inductive logic programming in answer set programming. In S. Muggleton, A. Tamaddoni-Nezhad, & F. Lisi (Eds.), *Inductive logic programming* (pp. 91–97). Springer.

de Jong, H. (2002). Modeling and simulation of genetic regulatory systems: A literature review. *Journal of Computational Biology, 1*(9), 67–103.

Delgado, F., & Gomez-Vela, F. (2019). Computational methods for Gene Regulatory Networks reconstruction and analysis: A review. *Artificial Intelligence In Medicine, 95,* 133–145.

Friedman, N., Linial, M., Nachman, I., & Pe'er, D. (2000). Using Bayesian networks to analyze expression data. *Journal of Computational Biology,7,* 601–620.

Goedertier, S., Martens, D., B.Baesens, H. R., & Vanthienen, J. (2007). Process mining as first-order classification learning on logs with negative events. In *Proceedings 3rd workshop on business processes intelligence (BPI'07)* (2007).

Inoue, K. (2011). Logic Programming for Boolean networks. *Proceedings of IJCAI, 2011,* 924–930.

Inoue, K., Doncescu, A., & Nabeshima, H. (2013). Completing causal networks by meta-level abduction. *Machine Learning, 91,* 239–277.

Inoue, K., Ribeiro, T., & Sakama, C. (2014). Learning from interpretation transition. *Machine Learning, 94*(1), 51–79.

Kakas, A., & Michael, L. (2020). Abduction and argumentation for explainable machine learning: A position survey. In *Encyclopedia of machine learning and data science* (3rd Edn.). Springer.

Katzouris, N., Michelioudakis, E., Artikis, A., & Paliouras, G. (2018). Online learning of weighted relational rules for complex event recognition. In *ECML-PKDD 2018: Proceedings of joint European conference on machine learning and knowledge discovery in databases* (pp. 396–413). Springer.

Katzouris, N., Artikis, A., & Paliouras, G. (2015). Incremental learning of event definitions with inductive logic programming. *Machine Learning, 100*(2–3), 555–585. https://doi.org/10.1007/s10994-015-5512-1.

Katzouris, N., Artikis, A., & Paliouras, G. (2016). Online learning of event definitions. *Theory and Practice of Logic Programming, 16*(5–6), 817–833.

Kaufman, M., Urbain, J., & Thomas, R. (1985). Towards a logical analysis of the immune response. *Journal of Theoretical Biology, 114,* 527–561.

Klipp, E., Liebermeister, W., Wierling, C., & Kowald, A. (2016). *Systems biology: A textbook*. Weinheim, Germany: Wiley-VCH.

Kowalski, R., & Sadri, F. (2015). Reactive computing as model generation. *New Generation Computing, 33,* 33–67.

Kowalski, R., & Sergot, M. (1986). A logic-based calculus of events. *New Generation Computing, 4*(1), 67–96.

Law, M., Russo, A., & Broda, K. (2020). The ILASP system for inductive learning of answer set programs. CoRR abs/2005.00904. arXiv:2005.00904.

Law, M., Russo, A., Bertino, E., Broda, K., & Lobo, J. (2020). FastLAS: Scalable inductive logic programming incorporating domain-specific optimisation criteria. In *Proceedings of the thirty-fourth AAAI conference on artificial intelligence (AAAI-20)* (pp. 2877–2885).

Lloyd, J. W. (1987). *Logic programming* (2nd ed.). Berlin: Springer-Verlag.

Maimari, N., Broda, K., Kakas, A., Krams, R., & Russo, A. (2014). *Symbolic representation and inference of regulatory network structures* (Chap. 1, pp. 1–48). John Wiley & Sons, Ltd.

Mauro, N.D., Basile, T., Ferilli, S., & Esposito, F. (2007). Mining frequent patterns from multi-dimensional relational sequences. In *Proceedings of sixth interntional workshop on multi-relational data mining* (p. 22).

McAreavey, K., Bauters, K., Liu, W., & Hong, J. (2017). The event calculus in probabilistic logic programming with annotated disjunctions. In *Proceedings of the 16th conference on autonomous agents and multiAgent systems, AAMAS 2017* (pp. 105–113). São Paulo, Brazil, May 8–12, 2017.

Moyle, S. (2003). An investigation into theory completion techniques in inductive logic programming. Ph.D. thesis, University of Oxford.

Moyle, S., & Muggleton, S. (1997). Learning programs in the event calculus. In *Proceedings of the 7th international workshop on inductive logic programming* (pp. 205–212). Springer.

Mueller, E. (2008). Event calculus. In: *Handbook of knowledge representation* (Vol. 3, pp. 671–708). Elsevier.

Muggleton, S. (1994). Inductive logic programming: Derivations, successes and shortcomings. *ACM SIGART Bulletin*.

Muggleton, S., & Bryant, C. (2000). Theory completion using inverse entailment. In *ILP 2000: Proceedings of international conference on inductive logic programming* (pp. 130–146). Springer.

Muggleton, S. (1995). Inverse entailment and progol. *New Generation Computing, 13,* 245–286.

Nickles, M. (2012). Integrating relational reinforcement learning with reasoning about actions and change. In *Inductive logic programming. ILP 2011* (pp. 255–269). Springer.

Nienhuys-Cheng, S., & de Wolf, R. (1997). *Foundations of inductive logic programming. Lecture notes in artificial intelligence* (Vol. 1228). Springer.

Platzer, A. (2018). *Logical foundations of cyber-physical systems*. Springer.

Ray, O. (2008). Nonmonotonic abductive inductive learning. *Journal of Applied Logic, 7,* 329–340.

Ribeiro, T., Folschette, M., Magnin, M., Roux, O., & Inoue, K. (2018). Learning dynamics with synchronous, asynchronous and general semantics. In F. Riguzzi, E. Bellodi & R. Zese (Eds.), *Proceedings of ILP 2018* (pp. 118–140). Springer.

Ribeiro, T., Magnin, M., Inoue, K., & Sakama, C. (2015). Learning multi-valued biological models with delayed influence from time-series observations. In *14th IEEE international conference on machine learning and applications, ICMLA 2015* (pp. 25–31). IEEE.

Ribeiro, T., Tourret, S., Folschette, M., Magnin, M., Borzacchiello, D., Chinesta, F., Roux, O., & Inoue, K. (2018). Inductive learning from state transitions over continuous domains. In N. Lachiche & C. Vrain (Eds.), *Proceedings of ILP 2017* (pp. 124–139). Springer.

Robertson, D. (1991). Feedback theory and Darwinian evolution. *Journal of Theoretical Biology, 152,* 469–484.

Shanahan, M. (1999). The event calculus explained. In *Artificial intelligence today* (pp. 409–430). Springer.

Snoussi, E. H. (1989). Qualitative dynamics of piecewise-linear differential equations: A discrete mapping approach. *Dynamics and Stability of Systems, 4*(3–4), 565–583.

Srinivasan, A. (1999). The Aleph manual. Available at https://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph.

Srinivasan, A., & Camacho, R. (1999). Numerical reasoning with an ILP system capable of lazy evaluation and customised search. *Journal of Logical Programming, 40*(2–3), 185–213. https://doi.org/10.1016/S0743-1066(99)00018-7.
Thieffry, D., & Thomas, R. (1995). Dynamical behaviour of biological regulatory networks—II. Immunity control in bacteriophage lambda. *Bulletin of Mathematical Biology,57*(2), 277–297.
Thomas, R. (1977). *Kinetic logic—A Boolean approach to the analysis of complex regulatory systems*. Springer.
Thomas, R. (1983). Fully asynchronous logical description of networks comprising feedback loops. In *Lecture notes in biomathematics* (Vol. 49). Springer.
Thomas, R., & D'Ari, R. (1990). *Biological feedback*. CRC Press.
Thomas, R. (1991). Regulatory networks seen as asynchronous automata: A logical description. *Journal of Theoretical Biology, 153,* 1–23.
Videla, S., Konokotina, I., Alexopoulos, L., Saez-Rodriguez, J., Schaub, T., Siegel, A., & Guziolowski, C. (2015). Designing experiments to discriminate families of logic models. *Frontiers in Bioengineering and Biotechnology,3*(131).
Wiener, N. (1961). *Cybernetics* (2nd edn.). MIT Press.
Wilhelm, T. (2009). The smallest chemical reaction system with bistability. *BMC Systems Biology,3*(90).

Authors and Affiliations

Ashwin Srinivasan[1] · Michael Bain[2] · A. Baskar[3]

Michael Bain
m.bain@unsw.edu.au

A. Baskar
abaskar@goa.bits-pilani.ac.in

[1] Department of CSIS and APPCAIR, BITS Pilani, Goa Campus, Sancoale, Goa, India

[2] School of Computer Science and Engineering, University of New South Wales, Sydney, NSW, Australia

[3] Department of Computer Science & Information Systems, BITS Pilani, Goa Campus, Sancoale, Goa, India