Bounded Asynchrony: A Notion of Concurrency Tailored for Modeling Cell-Cell Interactions*

Jasmin Fisher, Thomas A. Henzinger, and Nir Piterman

EPFL, Lausanne

Abstract. The new emerging field of Systems Biology aims to gain a system-level understanding of biology. In order to achieve such an understanding we need to establish the methodologies and techniques that will enable us to understand biological systems as systems. One such attempt is to use existing formal methods designed for the construction of digital systems to model biological systems. While using existing formal methods to model biological processes leads to fast and impressive results, a more thorough understanding requires specialized formalisms that are custom-made for modeling biology. Here we suggest a notion of concurrency that is tailored to the construction of high-level executable models in biology. We find that the two standard notions of concurrency, synchrony and asynchrony, are either too constrained or too loose when modeling certain biological behaviors, such as cell-cell interactions. Thus, we introduce *bounded asynchrony*, a notion that allows the components of a system to move separately while keeping them strongly coupled. We allow processes to move asynchronously while bounding the number of steps that a process can be ahead of other processes. This results in a natural framework to model races between cells. Indeed, a global system step is partitioned to local process steps where a process can move ahead of its neighbors and break the symmetry between the cells. We illustrate the need of this notion through a model representing the intercellular signaling events underlying cell fate decisions during *Caenorhabditis elegans* vulval development. Bounded asynchrony introduces non-determinism, which captures the variability observed under certain mutations leading to multiple fate patterns; and breaks the symmetry between equally potent cells, enabling adjacent cells to assume distinct cell fates.

1 Introduction

Formal languages have been recently used to create executable models of biological systems. For example, models of biochemical processes [21, 22, 9, 6, 5], T cell activation [15] and differentiation in the thymus [7, 8], and more recently to model vulval development in *C. elegans* [16, 11].[1] The latter models are high-level abstractions of the system's behavior, constructed as discrete, state-based models rather than quantitative continuous models (using differential equations for example).

* This work was supported in part by the Swiss National Science Foundation (grant 205321-111840).

[1] A different line of research is using hybrid systems to create a combined model of continuous and discrete aspects of a biological system [1, 13, 12, 20]. This approach is quite different from our own and we do not relate to it in this paper.

One of the main features of biological systems is that they are highly concurrent. When we construct a computational model, we are facing the question of how to compose smaller components together. There are two standard notions of composition: *synchronous* and *asynchronous* [3]. In synchronous composition, the components of a system proceed in lock-step, with every global state transition consisting of many local state transitions (one for each component). In asynchronous composition, the components of a system proceed independently, with every global state transition consisting of a single local state transition, belonging to a single component, which is chosen nondeterministically by a scheduler. Biological systems, however, while highly concurrent, are neither completely synchronous nor completely asynchronous. Different molecules, or cells, do not progress in perfect lock-step, and neither does any molecule or cell rest for arbitrary amounts of time. Indeed, when we try to model cell-cell interaction we find that synchronous composition is to rigid, making it impossible to break symmetry between processes without the introduction of additional artificial mechanisms. On the other hand, asynchronous composition introduces a difficulty in deciding when to stop waiting for a signal that may never come, again requiring artificial mechanisms.

For this reason, we introduce a notion of *bounded asynchrony* into our biological models, which captures the phenomenon that components of a biological system proceed approximately along the same time-line. In order to implement bounded asynchrony, we associate a rate with every process. The rate determines the time t that the process takes to complete an action. A process that works according to rate t makes an action slightly before time t. Then again slightly before time $2t$, $3t$, etc. This way, processes that work according to the same rate work more or less concurrently, and are always at the same stage of computation, however, the action itself can be taken first by either one of the processes or concurrently[2]. Other notions of bounded asynchrony either allow processes to 'drift apart', allowing one process to take arbitrarily more actions than another process, or do not generalize naturally to processes working according to different rates. In Section 3 we explain in detail the motivation and the reasoning that led us to construct bounded asynchrony in this way.

Using this intuition we define the notion of bounded asynchrony by means of a clocked transition system that serves as a scheduler. Some of the transitions of the scheduler are marked with actions that instruct other processes when to move. We then show that for our needs we can replace the real-time scheduler with a discrete-time scheduler. Thus, our final suggestion is a synchronous system with a nondeterministic scheduler instructing which processes to move when.

We find this notion of bounded asynchrony extremely useful, in the context of modeling cell-cell interactions. On the one hand, asynchrony is essential in order to break the symmetry between cells (processes). Adding a mechanism that breaks the symmetry is not supported by the biological data and seems artificial. On the other hand, much like in distributed protocols, a process has to know when to give up on waiting for messages that do not arrive. With classical asynchrony this is impossible and we are forced to add some synchronizing mechanism. Again, in the context of biology, such a mechanism is not supported by the data. When introducing bounded asynchrony both problems are solved. The asynchrony breaks the symmetry and the bound allows processes to de-

[2] We note that this process is not memoryless, making continuous time Markov chains inappropriate. This issue is discussed in detail in Section 6.

cide when to stop waiting. In addition, the asynchrony introduces nondeterminism that captures the diversity of results often observed in biology.

In order to model bounded asynchrony we currently use synchronous composition with a nondeterministic scheduler. When coming to analyze bounded-asynchronous systems we find that this solution is cumbersome. The scheduler introduces many state variables into the system and is hard to ignore when analyzing the system. We believe that further studies into this notion of concurrency are needed in order to solve these problems.

Bounded asynchrony is in a sense the dual of GALS (globally-asynchronous-locally-synchronous): it represents systems that look globally, viewed at a coarse time granularity, essentially synchronous, while they behave locally asynchronous, at a finer time granularity. Efficient implementations of synchronous embedded architectures also fall into this category. For example, time-triggered languages such as Giotto [14] have a synchronous semantics, yet may be implemented using a variety of different scheduling and communication protocols.

2 Transition Systems and Clocked Transition Systems

2.1 Transition Systems

We describe *transition systems* (TS) [18]. A TS $\mathcal{D} = \langle V, W, \Theta, \rho \rangle$ consists of the following components.

- $V = \{u_1, \ldots, u_n\}$: A finite set of typed *state variables* over finite domains. We define a *state* s to be a type-consistent interpretation of V, assigning to each variable $u \in V$ a value $s[u]$ in its domain. We denote by Σ the set of all states. An *assertion* over V is a Boolean combination of comparisons $u = a$ or $u = v$ where $u, v \in V$ range over the same domain and a is a value in the domain of u. A state s satisfies an assertion φ denoted $s \models \varphi$, if φ evaluates to $\mathbf{true}$ by assigning $s[u]$ to every one of the variables appearing in φ. We say that s is a φ-state if $s \models \varphi$.
- $W \subseteq V$: A set of *owned* variables. These are the variables that are actively changed by $\mathcal{D}$. Usually W includes the Boolean *scheduling variable* a.
- Θ : The *initial condition*. This is an assertion characterizing all the initial states of the TS. A state is called *initial* if it satisfies Θ.
- ρ: A *transition relation*. This is an assertion $\rho(V, V')$, relating a state $s \in \Sigma$ to its $\mathcal{D}$-successor $s' \in \Sigma$ by referring to both unprimed and primed versions of the state variables. The transition relation $\rho(V, V')$ identifies state s' as a $\mathcal{D}$-*successor* of state s if $(s, s') \models \rho(V, V')$, where (s, s') is the joint interpretation which interprets $x \in V$ as $s[x]$, and x' as $s'[x]$. Usually, the transition relation ρ has the form $(a{\neq}a' \wedge \rho') \vee (W{=}W')$, where a is the scheduling variable.

We define a *run* of $\mathcal{D}$ to be a sequence of states $\sigma : s_0, s_1, \ldots$, satisfying the requirements of

- *Initiality:* s_0 is initial, i.e., $s_0 \models \Theta$.
- *Consecution:* For every $j \geq 0$, the state s_{j+1} is a $\mathcal{D}$-successor of the state s_j.

We denote by *runs*$(\mathcal{D})$ the set of runs of $\mathcal{D}$. Note that in case that $\mathcal{D}$ has a scheduling variable a, we can divide the run to transitions where $\mathcal{D}$ stutters (i.e., a and all variables in W do not change) and where $\mathcal{D}$ moves (i.e., a flips its value and variables in W may change).

Given systems $\mathcal{D}_1 : \langle V_1, W_1, \Theta_1, \rho_1\rangle$ and $\mathcal{D}_2 : \langle V_2, W_2, \Theta_2, \rho_2\rangle$ such that $W_1 \cap W_2 = \emptyset$, the *synchronous parallel composition*, denoted by $\mathcal{D}_1 \,\|\, \mathcal{D}_2$, is the TS whose set of variables and set of owned variables are the union of the corresponding sets in the two systems, whose initial condition is the conjunction of the two initial conditions, and whose transition relation is the conjunction of the two transition relations. Thus, a step in an execution of the combined system is a joint step of systems $\mathcal{D}_1$ and $\mathcal{D}_2$. For more details, we refer the reader to [17].

The *projection* of a state s on a set $V' \subseteq V$, denoted $s\Downarrow_{V'}$, is the interpretation of the variables in V' according to their values in s. Projection is generalized to sequences of states and to sets of sequences of states in the natural way.

2.2 Clocked Transition Systems

A *clocked transition system* (CTS) [2] $\Phi = \langle V, W, \theta, \rho, \Pi\rangle$ consists of

- V: a finite set of *state variables*. The set $V = D \cup C$ is partitioned into $D = \{u_1, \ldots, u_n\}$ the set of *discrete variables* and $C = \{t_1, \ldots, t_k\}$ the set of *clocks*. Clocks range over non-negative values in $\mathbb{R}$. Discrete variables are as in TS.
- $W \subseteq C$: A set of *owned* variables. These are the variables that are actively changed by $\mathcal{D}$. Note that only discrete variables are owned.
- Θ: The *initial condition*. An assertion characterizing the initial states. For every $c \in C$ we require that $\Theta \to c{=}0$, i.e., all clocks are initialized to zero.
- ρ: A *transition relation*. An assertion $\rho(V, V')$, referring to both unprimed and primed versions of the state variables. For every $c \in C$ we require that $\rho \to (c{=}c' \vee c'{=}0)$, i.e., all clocks are either not modified or reset to zero by transitions.
- Π - a *time progress condition*. An assertion over V. The assertion is used to specify a global restriction over the progress of time.

A *run* of Φ is an infinite sequence of states $\sigma : s_0, s_1, \ldots$ satisfying

- *Initiality*: $s_0 \models \Theta$.
- *Consecution*: For each $j \geq 0$ we have either
  - s_{j+1} is a ρ successor of s_j, or
  - for every $u \in D$ we have $s_j[u] = s_{j+1}[u]$ and there exists $\Delta > 0$ such that for every $c \in C$ we have $c'{=}c{+}\Delta$ and forall $\delta{\in}[0, \Delta)$ the condition $\Pi(D, C{+}\delta)$ holds.
- *Time Divergence*: There exists some clock $c \in C$ such that $\Sigma_{i=1}^{\infty} s_i[c]{-}s_{i-1}[c]$ increases beyond any bound. That is, time grows beyond any bound.

Given two CTS $\Phi_1 : \langle V_1, W_1, \Theta_1, \rho_1, \Pi_1\rangle$ and $\Phi_2 : \langle V_1, W_2, \Theta_2, \rho_2, \Pi_2\rangle$, the *parallel composition*, denoted by $\Phi_1 \,\|\, \Phi_2$, is the CTS whose set of variables is the union of the corresponding sets in the two CTS, whose initial condition, transition relation, and time progress condition are the conjunction of the corresponding conditions.

Notice that a TS is a CTS whose set of clocks is empty and whose time progress condition is **true**. Thus, the composition of TS and CTS is defined.

3 Real-time Scheduler

In this section we explain in detail the intuition that lead us to suggest the notion of bounded asynchrony. As mentioned, biological systems are neither completely synchronous nor completely asynchronous. Events do not occur according to some uniform clock and yet no process is neglected beyond other processes, keeping all processes more or less at the same stage. Here we define the notion of *bounded asynchrony* using a CTS. The CTS sets the time for a discrete processes to take actions.

A typical biological model follows the behavior of some components of the system. We view these components as separate concurrent processes. Thus, we build a TS where values of variables relate to quantities of certain compounds. Suppose that our abstract model tells us that a compound is increasing to a 'high' level. In practice, the sequence of events that lead to this rise in the level of the compound is the creation of a very large number of molecules, each created by some stochastic process. We can say that the amount of molecules reaches the 'high' level in time that is determined by these stochastic processes. It follows that the time it takes a compound to reach a certain level is determined stochastically. With high probability the time would range around some average reaction time. Given that the probabilities involved in this processes are unknown to us[3], we would like the event (e.g., compound reaches a 'high' level) to take place sometime between $1-\epsilon$ and 1 (i.e., the average time until the event occurs is $1-\frac{\epsilon}{2}$; with high probability it occurs between $1-\epsilon$ and 1).

We model this by combining a TS P with the CTS Φ depicted in Fig. 1. Let a_p be the scheduling variable of P, and let the transition relation of P be $(a_p{\neq}a_p' \wedge \rho) \vee (a_p{=}a_p' \wedge W{=}W')$. The CTS has two Boolean variables s and a_p and one clock x_p. The initial condition sets s to true and x_p to 0. The transition relation either (a) changes s from true to false when x is in the range $[1-\epsilon, 1)$ and flips the value of a_p or (b) changes s from false to true when x_p is 1, leaves a_p unchanged, and resets x_p to 0. Formally, $\Phi = \langle V, W, \Theta, \rho, \Pi\rangle$, where the components of Φ are as follows.

- $D = \{s, a_p\}$.
- $C = \{x_p\}$.
- $\Theta = s \wedge (x_p{=}0)$.
- $\rho =$
  $(s{\wedge}\overline{s}'{\wedge}(x_p{\geq}1{-}\epsilon){\wedge}(x_p{<}1){\wedge}x_p{=}x_p'{\wedge}a_p{\neq}a_p'){\vee}(\overline{s}{\wedge}s'{\wedge}(x_p{=}1){\wedge}(x_p'{=}0){\wedge}a_p{=}a_p')$.
- $\Pi = (s \rightarrow x_p{<}1) \wedge (\overline{s} \rightarrow x_p{\leq}1)$.

The composition of the two is a CTS in which moves of the TS P happen in the time range $[i{-}\epsilon, i)$ for every $i \in \mathbb{N}$.

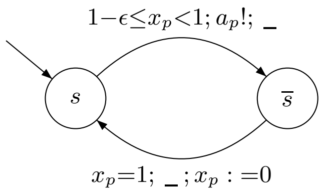

Fig. 1. CTS for one rate

[3] In addition, we are usually talking about thousands if not millions of molecules, the probability of the time diverging from average by more than a small fraction is negligible.

Suppose that we are following the level of component p in two separate cells. In both cells the level of component p increases to 'high'. We would like to model this by parallel composition. Suppose that we have two TS P and Q with scheduling variables a_p and a_q, respectively. We take the composition of two CTS as above using clocks x_p and x_q and the variables a_p and a_q. It follows that P and Q take approximately one time unit to make one move. However, the exact timing is not set. In a run of the system combined of the four TSs the order of actions between P and Q is not determined. Every possible ordering of the actions is possible. In addition, the transitions that reset the clocks x_p and x_q ensure that the two TS stay coupled. However long the execution, it cannot be the case that P takes significantly more actions than q (in this case more than one).

One may ask why not reset the clock immediately upon taking the action (*immediate reset* vs. *rounding reset* as above). Thus, a process would take its first action at time $s_1 \in [1-\epsilon, 1]$, its second action at time $s_2 \in [s_1+1-\epsilon, s_1+1]$, and in general action $i+1$ is taken at time $s_{i+1} \in [s_i + 1 - \epsilon, s_i + 1]$. This is a reasonable solution when the exact time of action is set stochastically. Indeed, we expect the action to occur sometimes early in the time window and sometimes late resulting in overall behavior in which actions occur every 1 time unit. However, when the scheduler is non-deterministic, we have to take into account the behavior where process 1 always takes actions as soon as possible and process 2 always takes actions as late as possible. In long enough executions, process 1 can make arbitrarily more actions than process 2. Much like the standard asynchrony, this may lead to undesirable results. Similar intuition prohibits the use of similar solutions that could be called 'bounded asynchrony'. For example, giving just an upper bound on the time of an action would result in executions where a slow process is able to make two very fast consecutive moves (with rounding reset) or executions where some processes act very fast and others very slow (with immediate reset). A different approach would be to say that the difference in number of steps between two processes is bounded. When restricting this approach to a distance of one step it turns out to be quite similar to our approach. It is, however, hard to generalize to processes that move according to different rates (see below).

We may want to work with two processes; one requiring t_1 time units to complete an action and the other requiring t_2 time units to complete an action. In this case we replace the bound 1 above by the bounds t_1 and t_2. Consider the CTSs in Fig. 2. Denote the CTS using bound t_1 by Φ_1 and the CTS using bound t_2 by Φ_2.

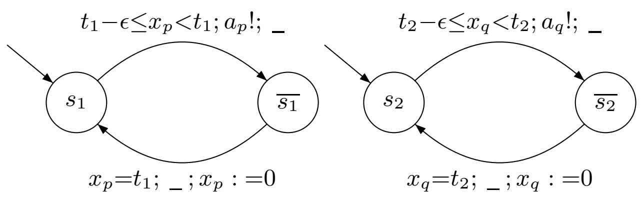

Fig. 2. CTSs for different rates

Let P and Q be TS with scheduling variables a_p and a_q as above. Consider the composition of P and Q with Φ_1 and Φ_2. It follows that P moves every t_1 time unites and Q every t_2 time units. For every ϵ and for every values t_1 and t_2 there are some integers i_1 and i_2 such that $[i_1 \cdot t_1 - \epsilon, i_1 \cdot t_1]$ intersects $[i_2 \cdot t_2 - \epsilon, i_2 \cdot t_2]$. As we are interested only in the sequence of actions taken by p_1 and p_2, and as the models we wish to consider result from a non-formal analysis of the biological system, restricting t_1 and t_2 to range over integer values does not seem like a major restriction.

Finally, we give a full formal definition. Define the CTS $T(t) = \langle V, \Theta, \rho, \Pi \rangle$ with the following components.

- $D = \{s, a\}$.
- $C = \{x\}$.
- $\Theta = s \wedge x = 0$.
- $\rho = (s \wedge \overline{s'} \wedge x{\geq}t - \epsilon \wedge x{<}t \wedge x{=}x' \wedge a{\neq}a') \vee (\overline{s} \wedge s' \wedge x{=}t \wedge x'{=}0 \wedge a{=}a')$.
- $\Pi = (s \rightarrow x_p{<}t) \wedge (\overline{s} \rightarrow x_p{\leq}t)$.

Consider n TSs $P_1, \ldots, P_n$. Let a_i be the scheduling variable of P_i and let $(\rho_i \wedge a_i \neq a'_i) \vee (a_i = a'_i \wedge W_i = W'_i)$ be the transition relation of P_i. For $1 \leq i \leq n$, let t_i denote the rate of P_i, let Φ_i denote $T(t_i)$, and let s_i, a_i, and x_i denote the variables of Φ_i.

The bounded-asynchronous composition of $P_1, \ldots, P_n$, denoted $\underset{i=1}{\overset{n}{|||}} P_i$, is $P_1 \parallel \Phi_1 \parallel \ldots \parallel P_n \parallel \Phi_n$. Intuitively a run of $\underset{i=1}{\overset{n}{|||}} P_i$ is $s_0, s_1, \ldots$ such that a transition from s_j to s_{j+1} either corresponds to the flow of time or to moves by a subset of the processes that does not change the owned variables of the rest of the processes.

More formally, a run of $\mathcal{P}$ is $\sigma : s_0, s_1, \ldots$ such that for every j one of the following holds:

- s_j differs from s_{j-1} only in all clocks increasing by some $\Delta > 0$,
- s_j differs from s_{j-1} only in some clocks being reset to 0,
- or there exists a nonempty set I_j of TS such that:
  - for every $P_i \in I_j$ we have (a) $x_i \in [t_i - \epsilon, t_i)$ (b) $(s_{j-1}, s_j) \models \rho_i \wedge a_i{\neq}a'_i$
  - and for every $P_{i'} \notin I_j$ we have $(s_{j-1}, s_j) \models W_i{=}W_{i'} \wedge a_i{=}a'_i$.

For the class of CTS considered above, most interesting questions we wish to consider are decidable and implementations are available. Below we explain how to replace the real time scheduler presented above by a discrete time scheduler. Going from real time to discrete time using variables ranging over finite domains enables us to use more efficient algorithms for simulations and model checking.

4 Discrete-Time Scheduler

We show how to replace the CTS scheduler presented above by a discrete scheduler that ensures that the actions of processes are the projection of the timed actions on un-timed actions, restricted to certain time points. The combination of the processes with the scheduler can be described as a system that takes global steps in which each subprocess takes one local step (or none in some cases). The order of actions between the subprocesses is completely non-deterministic. Thus, some of the processes may move together and some one after the other. In every global round each process makes one move keeping all processes together (regarding the number of actions).

We start by considering a set of processes working all according to the same rate (wlog 1). In this case, the resulting behavior is that every process does one action in every step of the global system however the order between all the steps is completely free. Namely, we can choose a subset of the processes, let them take a move, then continue recursively with the remaining processes until completing one round. We create a TS that schedules actions accordingly. The scheduler has a Boolean variable b_i associated with every process P_i. Process P_i is allowed to make a move when b_i is false and it is set to true. Once all variables b_i are set to true, they are all set concurrently to false (and no process makes a move).

More formally, consider n TSs $P_1, \ldots, P_n$. For $1 \leq i \leq n$, let a_i be the scheduling variable of P_i and let $(\rho_i \wedge a_i \neq a_i') \vee (W_i = W_i')$ be the transition relation of P_i. We define a scheduler $S = \langle V, W, \Theta, \rho\rangle$ where $V = \{b_1, \ldots, b_n, a_1, \ldots, a_n\}$, $W = \{b_1, \ldots, b_n\}$, $\Theta = \bigwedge_{i=1}^n \overline{b_i}$, and ρ is defined below. Let $\Omega = \bigvee_{i=1}^n \overline{b_i}$ denote the assertion that at least one variable b_i is still false.

$$\rho = \bigwedge_{i=1}^{n} \Big((\Omega \wedge b_i \rightarrow b_i') \wedge (\overline{\Omega} \rightarrow \overline{b_i}') \Big)$$

Reactive modules code implementing this scheduler is given in Appendix B.

Let $\mathcal{P} = |||_{i=1}^n P_i$ and let $\mathcal{P}' = S \parallel \|_{i=1}^n P_i$. We show that under appropriate projections the sets *runs*$(\mathcal{P})$ and *runs*$(\mathcal{P}')$ are equivalent. The projection restricts the run to states that are the result of transitions that changed at least one scheduling variable. More formally, for a run $\sigma : s_0, s_1, s_2, \ldots$, let $\overline{\sigma}$ denote the maximal subsequence of states $s_{j_0}, s_{j_1}, s_{j_2}, \ldots$ such that $j_0 = 0$ and for every $k > 0$ there exists some process P_i such that $s_{j_k - 1}[a_i] \neq s_{j_k}[a_i]$. Given a set R of runs, let $\overline{R} = \{\overline{\sigma} \mid \sigma \in R\}$. Let V' denote the set of variables of $P_1, \ldots, P_n$ (i.e., we ignore all the variables of the different schedulers).

Lemma 1. $\overline{\text{runs}(\mathcal{P})}\Downarrow_{V'} = \overline{\text{runs}(\mathcal{P}')}\Downarrow_{V'}$.

Given a run σ of $\mathcal{P}$, we construct a run σ' of $\mathcal{P}'$ by forcing the same order on the processes as in σ. Similarly in the other direction.

Proof. Consider a run $\sigma : s_0, s_1, \ldots$ of $\mathcal{P}$ and its projection $\overline{\sigma} : s_0, s_{j_1}, \ldots$. We construct a run σ' of $\mathcal{P}'$ by induction. Let r_0 be the state such that $r_0[V'] = s_0[V']$ and forall i we have $r_0[b_i] = \textbf{false}$. Consider a prefix of a run $r_0, r_1, \ldots, r_m$ that corresponds to the prefix $s_0, s_{j_1}, \ldots, s_{j_k}$. If forall i we have $r_m[b_i] = \mathbf{true}$ then r_{m+1} is the state such that $r_{m+1}[V'] = r_m[V']$ and for every i we have $r_{m+1}[b_i] = \textbf{false}$. Otherwise, r_{m+1} is the state such that $r_{m+1}[V'] = s_{j_{k+1}}[V']$ and forall i we have $r_{m+1}[b_i] = r_m[b_i] \vee (s_{j_{k+1}-1}[a_i] \neq s_{j_{k+1}}[a_i])$. It is simple to see that the resulting sequence $\sigma' : r_0, r_1, \ldots$ is a run of $\mathcal{P}'$ and that $\overline{\sigma'}\Downarrow_{V'} = \overline{\sigma}\Downarrow_{V'}$.

In the other direction the proof is similar. We take the sequence of decisions made by the scheduler S and add exact time stamps to it. ∎

We consider now the more general case of processes working with general rates. In this case, we use the same system of Boolean variables but in addition have a counter that counts the number of steps. A process is allowed to make a move only when its rate divides the value of the counter.

More formally, let the rates of $P_1, \ldots, P_n$ be $t_1, \ldots, t_n$. For $1 \leq i \leq n$, let a_i be the scheduling variable of P_i and let $(\rho_i \wedge a_i \neq a_i') \vee (W_i = W_i')$ be the transition relation of P_i. We define a scheduler $S = \langle V, W, \Theta, \rho\rangle$, where the components of S are as follows.

- $V = \{b_1, \ldots, b_n, c, a_1, \ldots, a_n\}$. Forall i we have b_i and a_i are Boolean, c ranges over $\{1, \ldots, lcm(t_1, \ldots, t_n)\}$, and $lcm(t_1, \ldots, t_n)$ is the least common multiplier.
- $W = \{b_1, \ldots, b_n, c\}$.
- $\Theta = c{=}1 \wedge \bigwedge_{i=1}^{n} \overline{b_i}$.
- Let $\Omega = \bigvee_{i=1}^{n} (\overline{b_i} \wedge c \textit{ mod } t_i{=}0)$ denote the assertion that at least one variable b_i for which the rate t_i divides the counter is still false.

$$\rho = \bigwedge_{i=1}^{n} (\Omega \wedge b_i \rightarrow b_i') \wedge (\overline{b_i} \wedge c \textit{ mod } t_i{\neq}0 \rightarrow \overline{b_i}') \wedge (\overline{\Omega} \rightarrow \overline{b_i}' \wedge c'{=}c \oplus 1)$$

Reactive modules code implementing this scheduler is given in Appendix B.

Let $\mathcal{P} = \mathop{|||}\limits_{i=1}^{n} P_i$ and let $\mathcal{P}' = S \parallel \mathop{\parallel}\limits_{i=1}^{n} P_i$. We show that under the appropriate projections the sets *runs*$(\mathcal{P})$ and *runs*$(\mathcal{P}')$ are equivalent. The projection restricts the run to the states that are the result of transitions that changed at least one scheduling variable. Given a run $\sigma : s_0, s_1, \ldots$ we reuse the definition of $\overline{\sigma}$. As before, let V' denote the set of variables of $P_1, \ldots, P_n$.

Lemma 2. $\overline{\text{runs}(\mathcal{P})}\Downarrow_{V'} = \overline{\text{runs}(\mathcal{P}')}\Downarrow_{V'}$.

As before, given a run σ of $\mathcal{P}$, we construct a run σ' of $\mathcal{P}'$ by forcing the same order on the processes as in σ.

Proof. Consider a run $\sigma : s_0, s_1, \ldots$ of $\mathcal{P}$ and its projection $\overline{\sigma} : s_0, s_{j_1}, s_{j_2}, \ldots$. We construct a run σ' of $\mathcal{P}'$ by induction. Let r_0 be the state such that $r_0[V'] = s_0[V']$, forall i we have $r_0[b_i] = \mathbf{false}$, and $r_0[c] = 1$. Consider a prefix of a run $r_0, r_1, \ldots, r_m$ that corresponds to the prefix $s_0, s_{j_1}, \ldots, s_{j_k}$. If forall i we have either $r_m[b_i] = \mathbf{true}$ or $c \textit{ mod } t_i{\neq}0$ then r_{m+1} is the state such that $r_{m+1}[V'] = r_m[V']$, forall i, $r_{m+1}[b_i] = \mathbf{false}$, and $r_{m+1}[c] = r_m[c] \oplus 1$. Otherwise, r_{m+1} is the state such that $r_{m+1}[V'] = s_{j_{k+1}}[V']$, forall i we have $r_{m+1}[b_i] = r_m[b_i] \vee (s_{j_{k+1}-1}[a_i]{\neq}s_{j_{k+1}}[a_i])$. It is simple to see that the resulting run $\sigma' : r_0, r_1, \ldots$ is a run of $\mathcal{P}'$ and that $\overline{\sigma'}\Downarrow_{V'} = \overline{\sigma}\Downarrow_{V'}$.

In the other direction the proof is similar. ∎

We note that the discrete scheduler is quite large. If we consider the case of four processes whose actions are coordinated using bounded asynchrony, it follows that there are 75 possible ways to complete a system step (i.e., one global step in which each of the processes moved once). In general, if we denote the number of possible ways to complete a system step for a system with n processes by $f(n)$ then $f(0) = 1$ and $f(n) = \Sigma_{i=1}^{n} \binom{n}{i} f(n-i)$. Obviously, when the system makes several global steps the number of possible interactions increases by multiplication. Even for relatively small systems, the number of possible interactions is huge (notice that this is only a subset of the possible interactions of the asynchronous composition of these processes, see also discussion in Section 6). In addition, bounded asynchrony allows all possible orders of interactions. As the variables $b_1, \ldots, b_n$ are maintained explicitly by the scheduler, using bounded asynchrony between n processes creates a system where at least 2^n different states are reachable (multiplied by the reachable assignments to other variables).

5 Usefulness of Bounded Asynchrony for Modeling Cell Fate Decisions

As part of an ongoing effort to model *C. elegans* vulval development [16, 11, 23] we have constructed a discrete dynamic state-based mechanistic model according to the current understanding of vulval formation [26, 4, 24, 29]. The *C. elegans* vulva (the egg-laying system) normally derives from three vulval precursor cells (VPCs) that are members of a larger set of six VPCs, named P3.p – P8.p (see Fig. 3). Each of the six VPCs is multipotent, capable of adopting one of three cell fates (1°, 2°, or 3°) [25–27]. The actual fate each cell adopts depends upon three intercellular signals: the epidermal growth factor receptor (EGFR) inductive signal emanating from the gonadal anchor cell (AC), the LIN-12/Notch lateral signal operating between VPCs, and the inhibitory signal coming from the surrounding hypodermal syncytium (Fig. 3). VPC fates in wild-type animals are influenced by their distance from the AC: the cell closest to the AC (P6.p) becomes 1°, the next closest (P5.p and P7.p) become 2°, and the most distant cells (P3.p, P4.p, and P8.p) become 3° [19, 25, 28].

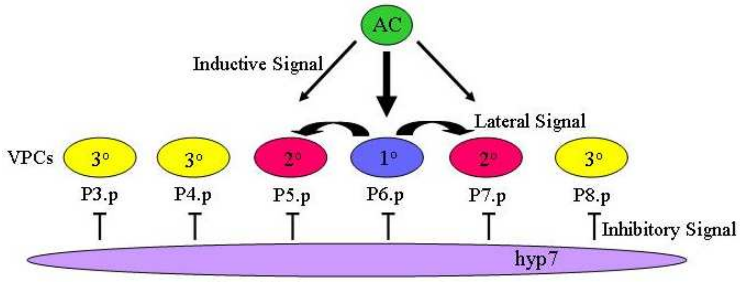

Fig. 3. The intercellular signaling specifying three cell fates during *C. elegans* vulval development

The postulated mechanism that drives VPC fate specification is as follows. The key player is the EGFR cascade that consists of four main proteins: LET-23, SEM-5, LET-60, and MPK-1. An activation of this cascade starts by an increase in the level LET-23. An increase in the level of a protein causes the next in line to increase as well until full activation of all four. There is a signal emanating from the dermis of the worm inhibiting the EGFR (i.e., keeping it inactive). This inhibitory signal is transduced by a protein called LIN-15. In the absence of this signal the EGFR cascade is initiated without external stimulus. The fate specification process starts when the anchor cell (AC) sends an inductive signal (IS) to the vulval precursor cells (VPCs). In a VPC receiving a low level of IS, the EGFR cascade is kept inactive by LIN-15 and the VPC adopts a 3° fate. A high IS causes the EGFR cascade to initiate. The cascade also causes a signal called lateral signal (LS) to be sent to the immediate surrounding of the cell only after disabling the receptors to this signal (LIN-12) in the same cell. Finally, the full activation of the cascade causes the cell to adopt a 1° fate. A medium level of IS also causes the EGFR cascade to initiate, however slower. Should it get to full activation, the cell assumes a 1° fate. The perception of the lateral signal sent by a neighbor cell causes an increase in the *lst* genes which turn the EGFR cascade off and cause the cell to assume a 2° fate. This is described in Fig. 4 by a static diagram as is the convention in biology. An arrow between two substances indicates activation (thickness relates to strength) and a bar indicates inhibition. Components in black are active while components in grey are inactive. By experimenting with the system (creating mutations) certain signals / receptors can be shut-down or encouraged, and the behavior of the system is observed in these cases. A model should be able to reproduce the behaviors observed in actual experiments.

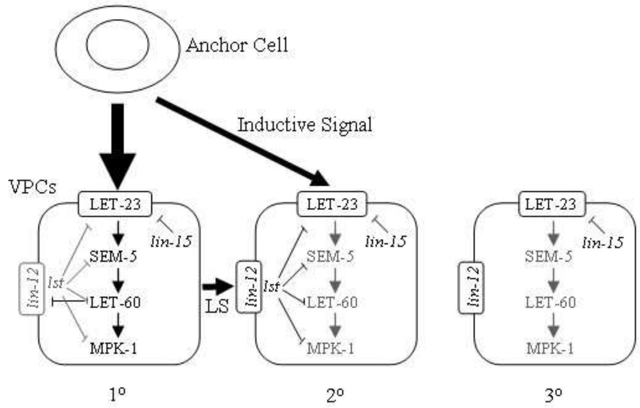

Fig. 4. Proposed model for the signaling events underlying VPC fate specification

Using the language of reactive modules [3] we have constructed a model of this system, as explained above. In order to explain the importance of bounded-asynchrony we concentrate on three experiments. First, in the case that LIN-15 is shut-down, all cells initiate the EGFR cascade, then some cells manage to stop the EGFR cascade in their neighbors. These cells adopt a 1° fate and their neighbors a 2° fate. In addition, in repeating this experiment, different cells assume a 1° fate. This suggests that we have to introduce asynchrony in order to break the symmetry between the cells. In the case that LIN-12 is shut-down, cells that initiate the EGFR cascade do not perceive the LS and assume a 1° fate. This suggests that in the case that EGFR cascade initiates, then at some point, it should decide that the LS does not arrive. Finally, in the unperturbed system it is always the case that if the EGFR cascade initiates slowly then it is counteracted by the LS. However, if we use asynchrony in the unperturbed system the scheduler may decide to delay the LS arbitrarily long; and the cells in which the EGFR is initiated slowly would assume a 1° fate.

It is relatively simple to see that when using bounded asynchrony, a cell that awaits a LS knows when to stop waiting. It can decide that the LS is not going to arrive at all and safely continue with its computation. Bounded asynchrony introduces small races between processes. In our case, cells proceed towards fate acquisition and a cell that adopts a 1° fate inhibits its neighbors from assuming the same fate. If a cell moves slightly faster than its neighbors, it gets to assume a 1° fate and inhibits its neighbors that assume a 2° fate. Dually, if a cell moves slightly slower than its neighbors, the neighbors get to assume the 1° fate and inhibit the cell that assumes a 2° fate. This reproduces very nicely the unstable fate patterns (same cell assumes different fates in repetitions of the experiment) observed in experiments involving LIN-15 shut-down.

We find that synchronous composition is too rigid, making it impossible to break symmetry between the cells. Asynchronous composition is too flexible. Processes cannot make decisions, as they cannot rely on the fact that the data about the state of the system is complete. Unlike the design of distributed algorithms, we cannot come up with our own suggestions as to how to synchronize the system. Bounded asynchrony introduces races between the processes allowing them to diverge and at the same time keeps processes strongly coupled so that decisions can be made without additional mechanisms.

It is an interesting choice whether to allow processes to move concurrently or not. Intuitively, we view concurrent moves of processes as the mirror of the rare event where the difference between biological processes is so small that it is ignored. We allow processes to move concurrently, creating situations in which cells proceed synchronously performing the same sequence of actions (in particular assuming the same fate). In some mutations leading to multiple possible outcomes, we know that some of the outcomes are rarely observed. When we disallow concurrent moves of processes, these rare observations disappear from the model, matching our intuition of bounded asynchrony.

For further details about the model described above, the novel insights it offers to biologists, and the experimental validation of these predictions we refer the reader to [10]. Additional details about the model and reactive modules code of the model are given in Appendix C.

6 Concluding Remarks

In this paper we introduced a notion of concurrency that is tailored specifically for executing abstract mechanistic models representing the signaling events underlying cell fate decisions. This notion, which we call *bounded asynchrony*, allows cells (processes) to move separately but in a strongly coupled manner, disallowing interleavings where one cell (process) advances faster than the others. Bounded asynchrony naturally captures the phenomena of equally potent cells choosing distinct cell fates.

We introduce bounded asynchrony by coding an explicit scheduler. Thus, we use synchronous composition and the scheduler gives explicit commands to some processes when to move. However, further studies are needed in order to understand how to analyze bounded-asynchronous systems. For example, consider the case of model checking. We note that bounded asynchrony creates highly non-deterministic models. However, in the models we consider, communication between processes is very local (e.g., one cell sensing a component level in a neighbor cell). Thus, the order of interaction between processes that do no communicate is not important. The outcome of a global system step does not depend on the order between processes that do not communicate. This observation could be used to improve the algorithms for model checking in these cases. In the context of equivalence checking, it is not clear how to factor out the scheduler to come up with a reasonable equivalence criterion.

The basic intuition that led us to suggest bounded asynchrony was the following. The occurrences we follow are the results of molecular interactions. Molecular interactions are often modeled using continuous time Markov chains (CTMC) that set the exact time of interaction [21, 5]. We are considering an abstraction, where we mimic an 'observable' change in the number of molecules by changes in the values of discrete variables. As the underlying process is stochastic, the time to make an 'observable' change would be more or less the average time for all the interactions to occur. Thus, we change the discrete level at a point in time that ranges between $t - \epsilon$ and t. As discussed in detail in Appendix A, it may be inappropriate to model abstractions of CTMCs as CTMCs. In particular, we cannot use CTMCs for abstract modeling as the abstracted process is no longer memoryless. The probability of a change in some state depends on the way the state is reached (e.g., which was the last variable to change and for how long other variables have not changed). Though CTMCs are not directly applicable, we find it interesting to explore the option of a probabilistic scheduler that sets the time of process actions (according to the exponential distribution, as prevalent in biology, or other). Such a scheduler would enable to quantify over the different interleavings and to measure the probability of results. Developing a theoretical framework that would allow to reason about abstractions of CTMCs is an interesting (and open) research topic.

We note, however, that handling probabilities usually incurs an extra computational price for simulation and analysis. The current discrete model already poses some problems when trying to perform advanced analysis on it (e.g., equivalence checking between models). It seems that adding another measure of complexity by introducing probabilities will make advanced analysis of the model impossible. In addition, our experience with probabilistic systems shows that the resulting distribution of outcomes may change considerably with small changes in the probabilities of transitions. Thus, it may be the case that without exact data on rates (that translate to probabilities) the system may be very hard to calibrate. We intend to further explore the option of using a probabilistic scheduler. However, we believe this requires a major theoretical and tool construction effort that is beyond the scope of this work.

Finally, the concept of bounded asynchrony suggests a certain degree of variability in the order in which cells progress during the process of fate determination. From a biological point of view, it is an interesting question whether such variability can really be observed in biological systems and whether this new notion will eventually provide further comprehension of the mechanisms underlying cell fate decisions.

7 Acknowledgments

We thank Alex Hajnal for fruitful discussions, and Marc Schaub and Maria Mateescu for comments on an earlier draft of the manuscript.

References

1. R. Alur, C. Belta, F. Ivancic, V. Kumar, M. Mintz, G. Pappas, H. Rubin, and J. Schug. Hybrid modeling and simulation of biomolecular networks. In *Fourth International Workshop*

on *Hybrid Systems: Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*, pages 19–32. Springer-Verlag, 2001.
2. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
3. R. Alur and T.A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15:7–48, 1996.
4. T. Berset, E. F. Hoier, G. Battu, S. Canevascini, and A. Hajnal. Notch inhibition of RAS signaling through MAP kinase phosphatase LIP-1 during C. elegans vulval development. *Science*, 291(5506):1055–8, 2001.
5. M. Calder, V. Vyshemirsky, D. Gilbert, and R. Orton. Analysis of signalling pathways using the PRISM model checker. In *Computational Methods in Systems Biology*, Edinburgh, Scotland, 2005.
6. L. Cardelli. Abstract machines of systems biology. *Transactions on Computational Systems Biology*, III:145–168, 2005.
7. S. Efroni, D. Harel, and I. R. Cohen. Modeling and simulation of the thymus. In *Multidisciplinary Approaches to Theory in Medicine*, 2002.
8. S. Efroni, D. Harel, and I. R. Cohen. Toward rigorous comprehension of biological complexity: modeling, execution, and visualization of thymic T-cell maturation. *Genome Res*, 13(11):2485–97, 2003.
9. D. D. Errampalli, C. Priami, and P. Quaglia. A formal language for computational systems biology. *Omics*, 8(4):370–80, 2004.
10. J. Fisher, N. Piterman, A. Hajnal, and T.A. Henzinger. Predictive modeling of *c. elegans* vulval development. 2006. Submitted.
11. J. Fisher, N. Piterman, E. J. Hubbard, M. J. Stern, and D. Harel. Computational insights into Caenorhabditis elegans vulval development. *Proc Natl Acad Sci U S A*, 102(6):1951–6, 2005.
12. R. Ghosh, A. Tiwari, and C. Tomlin. Automated symbolic reachability analysis; with application to delta-notch signaling automata. In *6th International Workshop on Hybrid Systems Computation and Control*, volume 2623 of *Lecture Notes in Computer Science*, pages 233–248, Prague, Czech Republic, 2003.
13. R. Ghosh and C. Tomlin. Lateral inhibition through delta-notch signaling: A piecewise affine hybrid model. In *4th International Workshop on Hybrid Systems Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*, pages 232–246, Rome, Italy, 2001.
14. T.A. Henzinger, B. Horowitz, and C.M. Kirsch. Giotto: A time-triggered language for embedded programming. In *Proc. 1st International Workshop on Embedded Software*, volume 2211 of *Lect. Notes in Comp. Sci.*, pages 166–184, 2001.
15. N. Kam, D. Harel, and I.R. Cohen. The immune system as a reactive system: Modeling T-cell activation with statecharts. In *IEEE Symposium of Visual Languages and Formal Methods*, pages 15–22. IEEE, 2001.
16. N. Kam, D. Harel, H. Kugler, R. Marelly, A. Pnueli, E. J. A. Hubbard, and M. J. Stern. Formal modeling of C. elegans development: A scenario-based approach. In *First International Workshop on Computational Methods in Systems Biology*, volume 2602 of *Lecture Notes in Computer Science*, pages 4–20, Roverto, Italy, 2003. Springer-Verlag.
17. Y. Kesten and A. Pnueli. Control and data abstractions: The cornerstones of practical formal verification. *Software Tools for Technology Transfer*, 2(1):328–342, 2000.
18. Y. Kesten and A. Pnueli. Verification by augmented finitary abstraction. *Inf. and Comp.*, 163:203–243, 2000.
19. J. Kimble. Alterations in cell lineage following laser ablation of cells in the somatic gonad of Caenorhabditis elegans. *Dev Biol*, 87(2):286–300, 1981.
20. C. Piazza, M. Antoniotti, B. Mishra, V. Mysore, A. Policriti, and F. Winkler. Algorithmic algebraic model checking i: Challenges from systems biology. In *17th International Confer-*

ence on Computer Aided Verification, volume 3576 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.

21. C. Priami, A. Regev, E.Y. Shapiro, and W Silverman. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters*, 80(1):25–31, 2001.
22. A. Regev, W. Silverman, and E. Shapiro. Representation and simulation of biochemical processes using the pi-calculus process algebra. In *Pac Symp Biocomput*, pages 459–70, 2001.
23. A. Sadot, J. Fisher, D. Barak, Y. Admanit, M. J. Stern, E. J. Hubbard, and D. Harel. Towards verified biological models. 2006. Submitted.
24. D. D. Shaye and I. Greenwald. Endocytosis-mediated downregulation of LIN-12/Notch upon Ras activation in Caenorhabditis elegans. *Nature*, 420(6916):686–90, 2002.
25. P. W. Sternberg and H. R. Horvitz. Pattern formation during vulval development in C. elegans. *Cell*, 44(5):761–72, 1986.
26. P. W. Sternberg and H. R. Horvitz. The combined action of two intercellular signaling pathways specifies three cell fates during vulval induction in C. elegans. *Cell*, 58(4):679–93, 1989.
27. J. E. Sulston and J. G. White. Regulation and cell autonomy during postembryonic development of Caenorhabditis elegans. *Dev Biol*, 78(2):577–97, 1980.
28. J. H. Thomas, M. J. Stern, and H. R. Horvitz. Cell interactions coordinate the development of the C. elegans egg-laying system. *Cell*, 62(6):1041–52, 1990.
29. A. S. Yoo, C. Bais, and I. Greenwald. Crosstalk between the EGFR and LIN-12/Notch pathways in C. elegans vulval development. *Science*, 303(5658):663–6, 2004.

A Abstraction of Continuous Time Markov Chains

In this section we give an example illustrating the inadequacy of continuous time Markov chains (CTMC) for our case. We start from a simple CTMC and show that an abstraction of this system cannot be modeled as a CTMC. One of the most important characteristics of CTMCs is that they are memoryless. That is, from every state the probability distribution over the next states is fixed regardless of the history that lead to this state. We show that when abstracting CTMCs this quality cannot be maintained (in a way that preserves distributions).

Let us start with a simplified definition of CTMCs. A CTMC is $\langle S, R\rangle$ where S is a finite set of states and $R : S \times S \to \mathcal{P}$ is the *probability matrix*. For every pair of states $R(s, s')$ is a distribution that sets the probability of the transition from s to s' in time t. If $R(s, s')$ is not the constant 0 for more than one state s', a *race* between the outgoing transitions from s exists. That is, the probability $P(s, s')$ of moving from s to s' in a single step equals the probability that the delay of going from s to s' "finishes before" the delays of any other outgoing transitions from s.

We now proceed to describe a simple CTMC. Consider a plant that can produce three types of elements, A, B, and C. In the initial phase of operation the plant produces either an A or a B with a race between A and B. Once 100 As have been produced, the plant starts to produce Cs instead of As. Once we have 200 Bs and 100 Cs, the plant stops. This can be modeled as a CTMC where the set of states is triplets of the form (a, b, c) where a and c range over $[0..100]$ and b ranges over $[0..200]$. Let U denote the uniform distribution over $[0, 1]$ and let Z denote the constant 0 distribution. We define the following probability matrix.

$$R((a, b, c),(a + 1, b, c)) = \begin{cases} U \text{ If } a < 100 \\ Z \text{ If } a = 100 \end{cases}$$

$$R((a, b, c),(a, b + 1, c)) = \begin{cases} U \text{ If } b < 200 \\ Z \text{ If } b = 200 \end{cases}$$

$$R((a, b, c),(a, b, c + 1)) = \begin{cases} U \text{ If } a = 100 \text{ and } c < 100 \\ Z \text{ Otherwise} \end{cases}$$

For example, from states of the form (a, b, c) where both $a < 100$ and $b < 200$, there exists a race between the increment of a and of b and as they use the same distribution the probability of the two transitions is equivalent.

Consider now an abstraction of this system where we care only about the truth value of the assertions $a < 100$, $b < 100$, and $c < 100$. We get a system with 8 states according to the possible truth values of the three assertions. We denote a state of the abstracted process by tuples in $\{\perp, \top\}^3$ where for example $(\perp, \perp, \perp)$ is the state where $a < 100$, $b < 100$, and $c < 100$. This process cannot be modeled as a CTMC as there is no consistent way to associate a distribution with transitions. Consider for example the concrete state $(100, 99, 0)$. It is abstracted to state $(\top, \perp, \perp)$. The rate from $(\top, \perp, \perp)$ to $(\top, \top, \perp)$ should be (much) higher than the rate from $(\top, \perp, \perp)$ to $(\top, \perp, \top)$. However the concrete state $(100, 99, 99)$ is also abstracted to $(\top, \perp, \perp)$, but now the rates to go to $(\top, \top, \perp)$ and $(\top, \perp, \top)$ should be equivalent.

Consider the following three TSs.

- For $i \in \{1,2\}$, let $\mathcal{D}_i = \langle V_i, W_i, \Theta_i, \rho_i \rangle$ where $V_i = W_i = \{v_i, a_i\}$, $\Theta_i = \overline{v_i}$, and $\rho_i = (a_i{\neq}a_i' \wedge (\overline{v_i} \rightarrow v_i) \wedge (v_i \rightarrow v_i)) \vee (a_i{=}a_i' \wedge v_i{=}v_i')$.
- Let $\mathcal{D}_3 = \langle V_3, W_3, \Theta_3, \rho_3 \rangle$ where $V_3 = \{v_1, v_3, a_3\}$, $W_3 = \{v_3, a_3\}$, $\Theta_3 = \overline{v_3}$, and $\rho_3 = (a_3{\neq}a_3' \wedge (\overline{v_3} \wedge v_1 \rightarrow v_3) \wedge (v_3 \rightarrow v_3) \vee (a_3{=}a_3' \wedge v_3{=}v_3')$.

Consider $\mathcal{P} = \mathcal{D}_1 \,|||\, \mathcal{D}_2 \,|||\, \mathcal{D}_3$ using rate 100 for all three processes. The process $\mathcal{P}$ has only a few possible behaviors. Depending on the choice in the 100th step whether to schedule $\mathcal{D}_1$ before $\mathcal{D}_3$ or not, variable v_3 increases from false to true either in the 199th step or the 200th step. Variable v_2 increases from false to true in the 100th step. By using rate 2 for all three processes and observing only the order of increase we see that the possible behaviors are that the order between the increase of v_1 and v_2 is not determined, however, it is always the case that v_1 and v_2 increase before v_3.

B Reactive Modules Code for Schedulers

We give explicit reactive modules code implementing the schedulers from Section 4. We note that in practice instead of using both the scheduling variable of the processes themselves and the Boolean variables of the scheduler we use the same Boolean variables for both purposes.

In Fig. 5 we have the reactive modules code for the scheduler that assumes all processes are working according to rate 1. The module `smallSched` that controls b_i does the following: The init command sets b_i to false. If b_i is false, the update command either sets b_i to true or leaves it unchanged. If b_i is true and rst' is true, the update command sets b_i to false. Otherwise, the update command leaves b_i unchanged. The module `resetMod` that controls rst does the following: The init command sets rst to false. If all b_i are true, the update command sets rst to true. Otherwise, the update command sets rst to false.

In Fig. 6 we have the reactive modules code for the general scheduler. The module `smallSched_t` that controls b_i does the following: The init command sets b_i to false. If b_i is false and t_i divides c, the update command either sets b_i to true or leaves it unchanged. If the value of c changes, the update command sets b_i to false. Otherwise, the update command leaves b_i unchanged. The module `cntMod` that controls c does the following: The init command sets c to 1. If forall i either b_i is true or t_i does not divide c, then the update command increases c by 1. Otherwise, the update command leaves c unchanged.

C Reactive-modules Code of the Biological Model

We give a few additional details about the structure of the model. Our model consists of a worm module that is comprised of an AC module, and six VPC modules. In addition, we have the scheduler, an additional module that handles the experimental conditions (organizer), and a module that analyzes the entire behavior of the model.

The AC module contains variables that represent the behavior of the inductive signal. An ablated AC does not send inductive signal while a normal AC sends a high inductive signal to P6.p, a medium inductive signal to P5.p and P7.p and low inductive signal to all other cells.

```
module smallSched
   interface b:bool;
   external rst:bool;

atom controls b
     reads b
     awaits rst
init
[] true -> b' := false;
update
[] !b -> b' := true;
[] !b -> b' := false;
[] b & rst' ->
       b' := false;
```

```
module resetMod
   interface rst:bool;
   external b_1,...,b_n:bool;

atom controls rst
     reads rst,b_1,...,b_n

init
[] true -> rst' := false;
update
[] rst -> rst' := false;
[] !rst &
     b_1 & ... & b_n ->
       rst' := true;
```

```
module a_1 is smallSched[b := b_1];
    .    .    .
    .    .    .
module a_n is smallSched[b := b_n];
module Scheduler is a_1 || ... || a_n || resetMod
```

Fig. 5. RM code for the basic scheduler.

The VPC module contains variables that represent the behavior of the following components: (a) lin-15 – a variable that controls the state of the *lin-15* gene (ON or OFF). This variable constitutively inhibits the EGFR pathway. Mutations that inactivate *lin-15* remove this inhibition. (b) lin-12 – a variable that represents the activity level of the *lin-12* gene with the observable values OFF, LOW, MEDIUM, and HIGH. *lin-12* is inhibited by the EGFR pathway and activated by the lateral signal. (c) lst – a variable that represents the state of the *lst* genes (ON or OFF). This variable is activated by *lin-12*. (d) Lateral Signal – a variable that represents the state of the lateral signal (ON or OFF). Once it is turned ON, the signal is sent to the neighboring VPCs. (e) EGFR – a collection of 4 variables representing the states of LET-23, SEM-5, LET-60, and MPK-1. Each of these variables awaits the activation by its predecessor (according to the following order: LET-23, SEM-5, LET-60, MPK-1) with LET-23 awaiting the activation by the inductive signal. During this time, each variable behaves according to the inhibition by *lin-15*, *lin-12*, and the *lst* genes. An additional variable follows the cell's progress towards fate assumption. Totally, our model (composition of all cells) has about 130 Boolean variables (for variables ranging over more than two values we use logarithmic encoding).

Our model enables consideration of the following mutations (and all possible combinations thereof): (a) AC can either be ablated or formed. (b) *lin-15* can be either wild-type or inactivated. (c) *lin-12* can either be wild-type, inactivated, or highly expressed. (d) *lst* can be either wild-type or inactivated (i.e., all *lst* genes are wild-type or all *lst* genes are inactivated). (e) Each of the components of the EGFR can be either wild-type

```
module smallSched_t
   interface b:bool; t:int;
   external c:[1..lcm];

atom controls b,t
     reads c,b,t
     awaits c
init
[] true -> b' := false;
           t' := rate;
update
[] !b -> b' := false;
[] !b & (c mod t = 0) ->
      b' := true;
[] c != c' ->
            b' := false;
```

```
module cntMod
   interface c:[1..lcm];
   external b_1,...,b_n:bool;
            t_1,...,t_n:int;
atom controls c
     reads b_1,...,b_n
           t_1,...,t_n,c
init
[] true -> c' := 1;

update
[] (b_1 | c mod t_1 != 0) &
         ... &
   (b_n | c mod t_n != 0) ->
         c' := (c mod lcm) + 1;
```

```
module a_1 is smallSched_t[b,t := b_1,t_1];
    .    .    .
    .    .    .
module a_n is smallSched_t[b,t := b_n,t_n];
module Scheduler is a_1 || ... || a_n || cntMod
```

Fig. 6. RM code for the full scheduler.

or inactivated. Judgments are used to check that the model reproduces the experimental observations observed under these mutations.