Bounded Asynchrony: A Biologically-inspired Notion of Concurrency*

Jasmin Fisher, Thomas A. Henzinger, Maria Mateescu, and Nir Piterman

EPFL, Switzerland

Abstract. We introduce *bounded asynchrony*, a notion of concurrency tailored to the modeling of biological cell-cell interactions. Bounded asynchrony is the result of a scheduler that bounds the number of steps that one process get ahead of other processes; this allows the components of a system to move independently while keeping them coupled. Bounded asynchrony accurately reproduces the experimental observations made about certain cell-cell interactions: its constrained nondeterminism captures the variability observed in cells that, although equally potent, assume distinct fates. Cells are not "scheduled", but we show that distributed real-time behavior can lead to component interactions that are observationally equivalent to bounded asynchrony; this provides a possible mechanistic explanation for the phenomena observed during cell fate specification.

We use model checking to determine cell fates. The nondeterminism of bounded asynchrony causes state explosion during model checking, but partial-order methods are not directly applicable. We present a new algorithm that reduces the number of states that need to be explored: our optimization takes advantage of the bounded-asynchronous progress and the spatially local interactions of components that model cells. Our experiments illustrate that using our algorithm leads to considerable saving when compared to partial order reduction.

1 Introduction

Recently, various discrete, state-based models have been suggested as high-level abstractions of biological systems' behavior [16, 8, 17, 12, 22, 9] (see below a short discussion of other approaches as well as [10] for a review of other uses of formal methods to model biological behaviors). Although very abstract, these models can shed light on complex biological phenomena (e.g., [9, 11] where insights gained by these methods are described as well as their biological validation). When coming to construct a computational model describing a biological behavior, we are facing the question of how to compose its components. 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. When we try to model cell-cell interactions, we find that synchronous composition is too rigid, making it impossible to break the 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 arrive, again requiring artificial mechanisms.[1]

* Supported in part by the Swiss National Science Foundation (grant 205321-111840).

[1] We treat here biological processes as computer processes. For example, when we say 'waiting', 'message', or 'decide' we relate to biological processes that take time to complete, and if

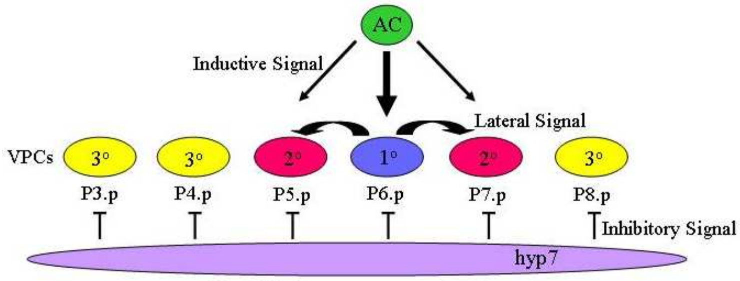


Fig. 1. The intercellular signaling specifying three cell fates during *C. elegans* vulval development

Biological observation: cell-cell interactions. We further explain why the standard notions of concurrency may be inappropriate for modeling biology, through a model representing the intercellular signaling events underlying cell-fate decisions during the earthworm *Caenorhabditis elegans* vulval development. 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. 1). Each of the six VPCs is multipotent, capable of adopting one of three cell fates (1°, 2°, or 3°) [24–26]. The actual fate each cell adopts depends upon three intercellular signals: the inductive signal emanating from the anchor cell (AC), the lateral signal operating between VPCs, and the inhibitory signal coming from the surrounding hypoderm (Fig. 1). 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° [20, 24, 27].

Consider the following three behaviors observed in experiments. First, under a particular gene mutation, all cells start an internal process that, if completed, leads to the assumption of a 1° fate. Then some cells manage to stop this process 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. Under another gene mutation, cells that start the same internal process ignore the signals sent by their neighbors and all assume a 1° fate. This suggests that in this case, at some point, a cell should decide that the message from neighbors does not arrive. Finally, in a normal (unperturbed) system, it is always the case that if the internal process initiates slowly, then it is counteracted by messages from the neighbors. However, if we use asynchrony in the unperturbed system, the scheduler may decide to delay the messages from neighbors arbitrarily long; and the cells in which the internal process is initiated slowly would assume a 1° fate, which is impossible in the real system.

Formal modeling: bounded asynchrony. For this reason, we introduce a notion of *bounded asynchrony* into our biological models, which allows components of a biological system to proceed approximately along the same time-line. In order to implement bounded asynchrony, we associate a rate with every process. The rate determines the

allowed to continue undisturbed may lead to irreversible consequences. Thus, as long as the process is going on the system 'waits', and if the process is not disturbed ('does not receive a message'), it 'decides'.

time t that the process takes to complete an action. A process that works according to rate t performs, in the long run, one action every tth round. 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, and the order may change from round to round[2]. Other notions of bounded asynchrony either permit 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.

Having the above mentioned example in mind, we define the notion of bounded asynchrony by introducing an explicit scheduler that instructs each of the cells when it is allowed to move. Thus, our system is in fact a synchronous system with a nondeterministic scheduler instructing which processes to move when. We find this notion of bounded asynchrony consistent with the observations made in cell-cell interactions. As explained, asynchrony is essential in order to break the symmetry between cells (processes). Adding a separate 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 decide when to stop waiting. In addition, the asynchrony introduces limited nondeterminism that captures the diversity of results often observed in biology.

Possible mechanistic explanation: real time. In some cases, biological systems allow central synchronization. For example, during animal development, it may happen that several cells are arrested in some state until some external signal tells all of them to advance. However, these synchronization mechanisms operate on a larger scale and over time periods that are much longer than the events described by our model. Thus, we do not believe that there is a centralized scheduler that instructs the processes when to move. The behaviors we describe are observed in practice, suggesting that there is some mechanism that actually makes the system work this way. This mechanism has to be distributed between the cells. We show that bounded asynchrony arises as a natural abstraction of a specific type of clocked transition systems, where each component has an internal clock. This may be related to the actual mechanism that creates the emergent property of bounded asynchrony.

Model checking: scheduler optimization. The scheduler we introduce to define bounded asynchrony consists of adding variables that memorize which of the processes has already performed an action in the current round. When we come to analyze such a system we find that this solution is cumbersome. The state variables introduced by the scheduler are hard to ignore; and we find that, much like in asynchronous systems, many different choices of the scheduler lead to the same states. Motivated by partial order

[2] We note that this process is not memoryless, making continuous time Markov chains inappropriate. This issue is discussed further below.

reduction [7], we show that in configurations of the system where communication is locally restricted, we can suggest alternative schedulers that explore only a fraction of the possible interleavings, however, visit all necessary states in order to explore all possible computations of the system. Partial order reduction techniques are not directly applicable in this setting. Experimental evaluation shows that our techniques lead to substantial improvement when compared with partial order reduction.

Related (and unrelated) models. There are mainly two approaches to handle concurrency in abstract biological models. One prevalent approach is to create a continuous time Markov chain (CTMC). This approach is usually used with models that aim to capture molecular interactions [13, 21]. Then, the set of enabled reactions compete according to a continuous probability rate (usually, the χ-distribution). Once one reaction has occurred, a new set of enabled reactions is computed, and the process repeats. The fact that our model is very abstract makes it very hard to use this approach. First, exact data on probabilities is difficult to obtain; indeed, even the data as to exactly which molecules are involved in the process may be missing (as it is the case in the *C. elegans* model). Second, our scheduler choices are made on the cellular level. This process is no longer memoryless, making CTMCs inappropriate. Consider a CTMC setting cells in motion according to the same rate (same distribution). Then, at any given state, all cells have the same probability of moving. Thus, a sequence of moves all made by the same cell is just as probable as a sequence of moves made by different cells.

A different approach, common in *Boolean networks* [18, 4, 6], is to use asynchrony between the substances. Again, this approach is usually applied to models that aim to capture molecular interactions, however, in an abstract way. Asynchronous updates of the different components is used as an over-approximation of the actual updates. If the system satisfies its requirements under asynchronous composition, it clearly satisfies them under more restricted compositions. We note, however, that these models are used primarily to analyze the steady-state behavior of models (i.e., loops that have no outgoing edges). As asynchrony over-approximates the required composition, such steady-state attractors are attractors also in more restricted compositions, justifying this kind of analysis. For our needs, we find unbounded asynchrony inappropriate.

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 Bounded Asynchrony

In this section we define the notion of bounded asynchrony. We first define transition systems and then proceed to the definition of bounded asynchrony.

2.1 Transition Systems

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. For an assertion 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 only $\mathcal{D}$ may change. 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 $(s, s') \models \rho(V, V')$. Usually, the transition relation ρ has the form $(a{\neq}a' \wedge \rho') \vee (W{=}W')$, where a is the scheduling variable. In what follows we restrict our attention to systems that use a scheduling variable.

A *run* of $\mathcal{D}$ is a sequence of states $\sigma : s_0, s_1, ...,$ satisfying the requirements of (a) *Initiality:* s_0 is initial, i.e., $s_0 \models \Theta$; (b) *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 $\mathcal{D}$. If $\mathcal{D}$ 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 *parallel composition*, denoted by $\mathcal{D}_1 \parallel \mathcal{D}_2$, is the TS $\langle V, W, \Theta, \rho \rangle$ where $V = V_1 \cup V_2$, $W = W_1 \cup W_2$, $\Theta = \Theta_1 \wedge \Theta_2$, and $\rho = \rho_1 \wedge \rho_2$. [3] For more details, we refer the reader to [19].

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 Explicit Scheduler

We define bounded asynchrony by supplying an explicit scheduler that lets all processes proceed asynchronously, however, does not permit any process to proceed faster than other processes. Intuitively, the system has one macro-step in which each of the processes performs one micro-step (or sometimes none), keeping all processes together (regarding the number of actions). 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. We start with a scheduler that allows all processes to proceed according to the same rate. We then explain how to generalize to a scheduler that implements bounded asynchrony between processes with different rates.

We start by considering a set of processes all working according to the same rate (without loss of generality the rate is 1). In this case, the resulting behavior is that every process does one micro-step in every macro-step of the system. Namely, we can choose a subset of the processes, let them take a move, then continue recursively with the remaining processes until completing one macro-step. We create a TS that sched-

[3] Notice that, in the case that $\mathcal{D}_1$ and $\mathcal{D}_2$ have stutter transitions, this composition is neither synchronous nor asynchronous in the classical sense.

ules actions accordingly. The scheduler has a Boolean variable b_i associated with every process P_i. A move of P_i is forced when b_i changes from false to true. Once all b_is are set to true, they are all set concurrently to false (and no process moves).

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 $V = W = \{b_1, \ldots, b_n\}$ and b_i is Boolean for all $1 \leq i \leq n$, $\Theta = \bigwedge_{i=1}^{n} \overline{b_i}$, and ρ is defined as follows:

$$\rho = \left(\Omega \rightarrow \bigwedge_{i=1}^{n} (b_i \rightarrow b'_i) \quad \wedge \quad \overline{\Omega} \rightarrow \bigwedge_{i=1}^{n} \overline{b'_i} \right) \tag{1}$$

Where $\Omega = \bigvee_{i=1}^{n} \overline{b_i}$ denotes the assertion that at least one variable b_i is still false.

The *bounded asynchronous parallel composition* of $P_1, \ldots, P_n$ according to the rate 1, denoted $P_1^1 \|_{_{ba}} \cdots \|_{_{ba}} P_n^1$, is $S \parallel P_1 \parallel \cdots \parallel P_n$ with the following additional conjunct added to the transition:

$$\bigwedge_{i=1}^{n} (a_i \neq a'_i \iff (\overline{b_i} \wedge b'_i)) \tag{2}$$

Thus, the scheduling variable of P_i is forced to change when b_i is set to true.

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 $S = \langle V, W, \Theta, \rho \rangle$ with the following components:

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

$$\rho = \left(\Omega \rightarrow \bigwedge_{i=1}^{n} (b_i \rightarrow b'_i) \right) \wedge \left(\overline{\Omega} \rightarrow \bigwedge_{i=1}^{n} (\overline{b'_i} \wedge (c'{=}c \oplus 1)) \right) \wedge \left(\bigwedge_{i=1}^{n} ((\overline{b_i} \wedge (c \; mod \; t_i{\neq}0)) \rightarrow \overline{b_i}') \right) \tag{3}$$

The *bounded asynchronous parallel composition* of $P_1, \ldots, P_n$ according to rates $t_1, \ldots, t_n$, denoted $P_1^{t_1} \|_{ba} \cdots \|_{ba} P_n^{t_n}$, is $S \parallel P_1 \parallel \cdots \parallel P_n$ with the the conjunct in Equation (2) added to the transition.

3 Modeling Cell-Cell Interactions

We describe at a high level the way we use bounded asynchrony to model the biological process of cell fate acquisition. Our model represents aspects of the *C. elegans* earthworm vulval development (cf. [17, 12, 22]). It is a discrete model constructed according to the current understanding of vulval formation [25, 5, 23, 28] (cf. Fig. 1 and relevant text in Section 1).

The postulated mechanism that drives VPC fate specification is as follows. The internal process mentioned earlier is a cascade that consists of a few proteins. An activation of this cascade starts by an increase in the level of the first protein. An increase in the level of a protein causes the next in line to increase as well until full activation of the cascade. There is a signal emanating from the dermis of the worm inhibiting this cascade (i.e., keeping it inactive). In the absence of this signal the cascade is initiated without an external stimulus. The fate specification process starts when the anchor cell (AC) sends an inductive signal (IS) to the VPCs. In a VPC receiving a low level of IS, the cascade is kept inactive and the VPC adopts a 3° fate. A high IS causes the cascade to initiate. The cascade also causes a lateral signal (LS) to be sent to the immediate surrounding of the cell only after disabling the receptors to this signal 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 cascade to initiate, however slower. If the cascade is fully activated, the cell assumes a 1° fate. The perception of the lateral signal sent by a neighbor cell turns off the cascade and causes the cell to assume a 2° fate. By experimenting with the system (creating mutations and perturbing the system in various ways) 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.

In order to explain the importance of bounded asynchrony we concentrate on three experiments. First, in the case that the inhibitory signal is shut-down, all cells initiate the protein cascade, then some cells manage to stop the 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 the receptor of the lateral signal (LS) is shut-down, cells that initiate the cascade do not perceive the LS and assume a 1° fate. This suggests that in the case that the cascade initiates, then at some point, it should decide that the LS does not arrive. Finally, in a normal (unperturbed) system it is always the case that if the 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 cascade is initiated slowly would assume a 1° fate.

It is relatively simple to see that when using bounded asynchrony, a cell that awaits an 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 (the same cell assumes different fates in repetitions of the same experiment) observed in experiments involving the inhibitory signal shut-down.

We find that synchronous composition is too rigid, making it impossible to break the symmetry between cells. On the other hand, 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.

Using the language of reactive modules [3] we have constructed a model describing the behavior of this system, as explained above. In this paper, intended for a Computer Science audience, we present the model of bounded asynchrony, as well as verification algorithms that take advantage of this model. In a separate paper, intended for a biological audience, we describe the insights this model gives to the biological system and the experiments that were done in order to validate these predictions [11]; bounded-asynchrony and its algorithmic aspect are not discussed in that paper. Using the techniques described in the next section we reduce model checking time from 13 to 5 seconds. As explained, we find it very important to be able to combine synchronous moves with asynchronous moves. Thus, we use Mocha [1] and reactive modules that allow to combine the two.

4 Model Checking

Partial Order Reduction (POR) [7] is a technique that takes advantage of the fact that in asynchronous systems many interleavings lead to the same results. It does this by not exploring some redundant interleavings, more accurately, by shrinking the set of successors of a state while preserving system behavior. Existing algorithms are designed for (unbounded) asynchronous systems and do not directly adapt to our kind of models (see below). Although, at the moment, we are unable to suggest POR techniques for bounded asynchrony, we propose an algorithm that exploits the restricted communication encountered in systems that model cell-cell interaction, we refer to our algorithm as *communication based reduction*, or CBR for short. Like POR, our algorithm searches only some of the possible interleavings, while covering all possible behaviors of the system. We reduce the reachable region of the scheduler from exponential size to polynomial size in the number of processes, and thus we have a direct and important impact on enumerative model checking.

4.1 Optimized Scheduler

Let $\mathcal{P} = P_1^1 \parallel_{ba} \cdots \parallel_{ba} P_n^1$ be the bounded asynchronous composition of $P_1, P_2, \ldots, P_n$ according to rate 1 (as defined in Section 2).[4] The scheduler S allows all possible interleavings of processes within a macro-step. We show that we can construct a new scheduler that preserves system macro-behavior (macro-step level behavior) but allows fewer interleavings.

A *macro-step* of $\mathcal{P}$ is a sequence of states $g : s = s_0, s_1, \ldots, s_m$ satisfying:
- g is a subsequence of a run,
- s_0 is initial with respect to the scheduler, i.e., $\overline{s_0[b_k]}$ holds for all $0 \leq k \leq n$,
- s_m is final with respect to the scheduler, i.e., $s_m[b_k]$ holds for all $0 \leq k \leq n$,
- s_m is the only final state in g.

Within $\mathcal{P}$, we say that s_m is a *macro-successor* of s with respect to interleaving $\mathcal{I}$ if there exists a macro-step g with initial state s, interleaving $\mathcal{I}$ and final state s_m.

A macro-step is formed by steps, $r_i : s_{i-1} \rightarrow s_i$. As processes run at equal rates, P_k moves in exactly one step, i_{g,P_k}, within macro-step g. We say that there is a channel c_{kl} connecting P_k and P_l if $V_k \cap V_l \neq \emptyset$. Given a macro-step g, a channel c_{kl} may have one of three states: enabled from k to l, if $i_{g,P_k} < i_{g,P_l}$, enabled from l to k, if $i_{g,P_l} < i_{g,P_k}$, disabled, if $i_{g,P_k} = i_{g,P_l}$. Intuitively, a channel is enabled if it may propagate a value generated in the current macro-step.

Consider a macro-step $g : s = s_0, s_1, \ldots, s_m$ of $\mathcal{P}$. An *interleaving* of g, denoted $\mathcal{I}_g$, is the projection of g on the variables controlled by the scheduler. An interleaving establishes a total order on the steps i_{g,P_k} in which processes move within g. A *channel configuration* of g, denoted $\mathcal{C}_g$, is the state of the channels in g. A channel configuration establishes a partial order on the steps of execution of the processes. The states of the channels imply the order in which the processes that are connected by that channel move. An interleaving (total order) induces a channel configuration (partial order). Two interleavings are $\mathcal{P}$*-equivalent* if they induce the same channel configuration. The following lemma establishes that two equivalent interleavings have the same set of successors. A full proof is given in Appendix A.

Lemma 1. *Consider two $\mathcal{P}$-equivalent interleavings $\mathcal{I}$ and $\mathcal{I}'$. If s' is a macro-successor of s with respect to $\mathcal{I}$, then s' is a macro-successor of s with respect to $\mathcal{I}'$.*

A scheduler that allows only one of two $\mathcal{P}$-equivalent interleavings preserves system macro-behavior. Actually, it is enough for a scheduler to generate only one interleaving per channel configuration. We choose this representative interleaving so that the scheduler reaches as few states as possible, and thus, reduce model checking time.

We show how to pick an interleaving for a channel configuration in the case of *line communication scheme* ($V_k \cap V_l = \emptyset$, for all $l \notin \{k-1, k+1\}$, $k \in (1..n)$). This is the channel configuration in the *C. elegans* model, as every cell has two neighbors. Let c_k denote the channel $c_{k,k+1}$. In interleaving g, we say that c_k is *enabled-right* if it is enabled from k to $k{+}1$, *enabled-left* is it is enabled from $k{+}1$ to k, and disabled as before.

[4] Here, we only describe the case of processes running at equal rates. The same ideas can be easily extended to general rates.

Given a channel configuration we choose one interleaving that induces this configuration. Consider the minimal right-enabled channel c_{l_0}. Process l_0 is oblivious to whatever happens in the same macro step in processes $l_0+1,\ldots,n$. Thus, whatever actions are performed by processes $l_0+1,\ldots,n$ they do not affect the actions of processes $1,\ldots,l_0$. Thus, we may shuffle all the actions of processes $1,\ldots,l_0$ to the beginning of the interleaving without affecting its results. The new interleaving starts by handling all processes $1,\ldots,l_0$ from right to left. Let $k_1,k_2,\ldots,k_m$ be the left-enabled channels in $1,\ldots,l_0$. Then, the order of moves is: first processes $k_m+1,\ldots,l_0$, then $k_{m-1}+1,\ldots,k_m$, and so on until $0,\ldots k_1$. Consider the minimal right-enabled channel c_{l_1} for some $l_1>l_0$. Using this reasoning, we can handle the processes in the range $l_0+1,\ldots,l_1$ from right to left according to the left-enabled channels, and so on.

The CBR scheduler also uses the Boolean variables $b_1,\ldots,b_n$, however, the reachable configurations described above are of the forms $0^+1^+0^+$, 1^+0^+, $1^+0^+1^+0^+$, and 1^+. With similar intuition configurations of the form 0^+, 0^+1^+, and $1^+0^+1^+$ are also reachable. There are only $O(n^3)$ such reachable states, compared to 2^n reachable states in the original scheduler[5]. A full implementation is given in Appendix A. Fig. 2(a) compares the number of states and transitions of the two schedulers (none, the scheduler described in Section 2 with no reduction vs. CBR, the scheduler described above) for different number of processes.

4.2 Experimental Evaluation

We compare experimentally the behavior of the CBR scheduler with POR methods. While Mocha [1] is used for the biological model as it allows concurrent moves, here, we use Spin [15] for a thorough analysis of the behavior of CBR.

We explain, intuitively, why POR is inappropriate for bounded asynchrony. First, we find it very important to allow processes to move concurrently. POR is developed for 'classical' asynchronous systems, thus, it does not allow for processes to move concurrently. Second, a macro-step in bounded asynchrony is a sequence of at most n local steps, and noticing that one interleaving is redundant may require exploration of more than 1 lookahead. Let us further explore this with an example. Suppose that we give up on concurrent moves and would like to use POR for reasoning about the same bounded asynchronous system. That is, processes in a line configuration where only neighbor processes may communicate. In the beginning of a macro-step, all processes are enabled. Communication between processes implies that we cannot find independent processes (such that the order of scheduling them does not matter), and we have to explore all possible n processes as the first process to move. With one process ahead of others, it is clear that the processes to the left of this process and to its right are no longer connected and the order between scheduling every process on the left and every process on the right can be exchanged. However, among the processes on one side, there is still dependency and the same selection by the scheduler has to be applied recursively. Overall, the number of possible interleavings to be checked is still exponential in n.[6]

[5] More accurately, the number of reachable states is $\frac{n^3}{6}+\frac{5n}{6}+1$.

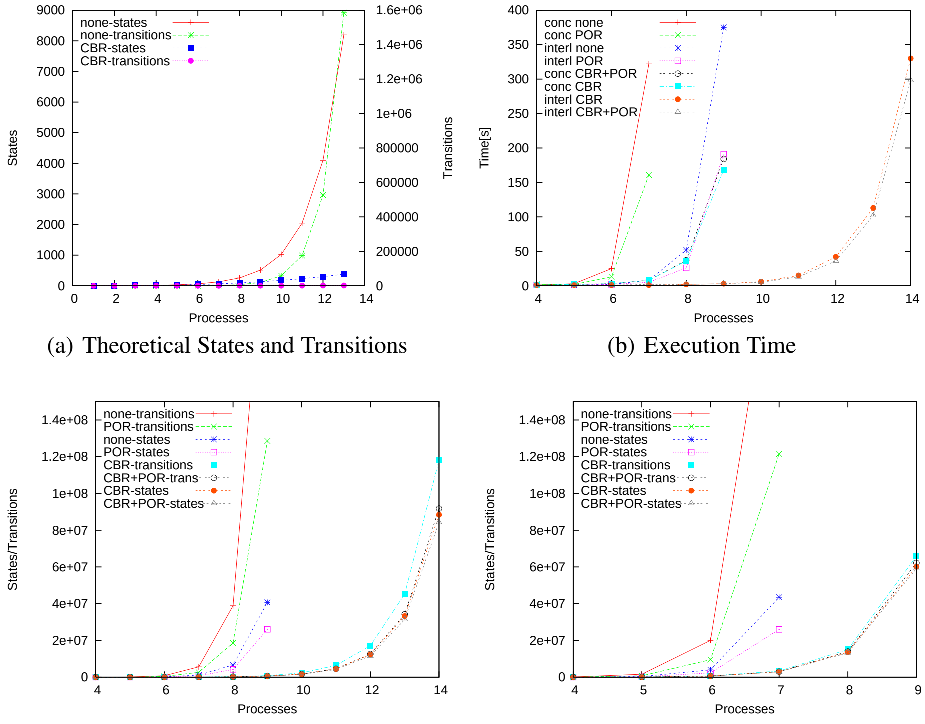
[6] More accurately, the analysis is as follows. The number of interleavings of one process is $f(1)=1$, the number of interleavings of zero processes is $f(0)=0$. Generally, $f(n)=\Sigma_{i=1}^{n}(f(i-1)+f(n-i))=2f(n-1)+f(n-1)=3f(n-1)$ and $f(n)=2\cdot 3^{n-2}$.

As exhibited by our experiments, POR does offer some reduction, however, this cannot be compared to the order of magnitude saving offered by using communication-based reduction.

For the experimental evaluation we suggest a model that is intended to capture channel configurations. It is the intention that from a given state, two different channel configurations lead to different states. The model has n processes in a line configuration scheme. Each process controls one variable and reads the variables controlled by its neighbors. In each step a process assigns its variable x to $max(x_{left}, x_{right}) + 1$. We evaluate the CBR scheduler by considering the time for enumerative model checking and the number of states and transitions explored during model checking. We compare the behavior of the CBR scheduler with the basic scheduler described in Section 2 (simple scheduler) when POR is enabled and disabled. We perform two sets of experiments, both using Spin. The first set of experiments uses the normal interleaving semantics of Spin. In this case the size of the CBR scheduler is reduced from $O(n^3)$ to $O(n^2)$ states. This set of experiments includes running the simple scheduler without any reductions (none), the simple scheduler with POR (POR), the CBR scheduler (CBR), and the CBR scheduler with POR (CBR+POR). The second set of experiments includes a mechanism that makes Spin mimic the possibility of concurrent moves. We note that this additional mechanism increases the size of each process and that in order to communicate with the CBR scheduler each process has additional variables. Thus, the experiment is unfair with respect to the CBR scheduler. As before, this set of experiments includes running the simple scheduler (conc none), simple scheduler with POR (conc POR), CBR scheduler (conc CBR), and CBR scheduler with POR (conc CBR+POR). In all experiments, increasing the number of processes by one leads to memory overflow (10GB). For example, for the experiment with 10 processes, with the simple scheduler where POR is enabled, Spin requires more than 10GB of memory. Fig. 2(b) compares the model-checking time for the different experiments. Figures 2(c) and 2(d) compare the numbers of states and transitions explored in the first (interleaving semantics) and second (with mechanism mimicking concurrent moves) sets of experiments, respectively. For better scaling, the range of values covered by these figures does not include the number of transitions for the none-experiments in the cases of 9 and 7 processes, respectively. Notice that the size of the system itself increases exponentially with the number of processes. The experiments confirm that POR offers some improvement while the communication-based reduction affords a significant improvement when compared with the simple scheduler with POR.

5 A Possible Mechanistic Explanation for Bounded Asynchrony

It is rather obvious that a scheduler such as the one we describe in Section 2 does not exist in real biological systems. While trying to describe biological behavior (of this type) in high-level requires us to use a notion like bounded asynchrony, it is not clear what is responsible for this kind of behavior in real systems. Obviously, no centralized control exists in this case, and there has to be some distributed mechanism that creates this kind of behavior. In this section we show that bounded asynchrony can be naturally used to abstract a special kind of distributed real-time mechanism. It is an interesting challenge to design biological experiments that will confirm or falsify the hypothesis that



(a) Theoretical States and Transitions

(b) Execution Time

(c) Interleaving States and Transitions

(d) Concurrent States and Transitions

Fig. 2. Comparing theory and practice

internal clock-like mechanisms are responsible for the emerging behavior of bounded asynchrony.

We suggest clocked transition systems (CTS) as a possible distributed mechanism that produces bounded asynchrony. The systems we consider use a single clock, perform actions when this clock reaches a certain value, and reset the clock. We give a high-level description of the CTS we have in mind. A formal treatment is in Appendix B.

Consider the CTS Φ depicted in Fig. 3. The CTS has two Boolean variables s and a_p and one clock x_p. The values of s correspond to the two states in the figure. The CTS is allowed to move from s to $\overline{s}$ when the clock x is in the range $[1-\epsilon, 1)$, for some ϵ. When the CTS moves from $\overline{s}$ to s, it resets the clock back to 0. The variable a_p is the scheduling variable that this CTS sets; it changes when the system moves from s to $\overline{s}$, and does not change when the system moves from $\overline{s}$ to s. The possible computations of this system include the clock progressing until some point in $[1-\epsilon, 1)$, then the system makes a transition from s to $\overline{s}$ while changing a_p, then the clock progresses until it is 1, and finally the system makes a transition from $\overline{s}$ to s. Then, the process repeats itself when the global time is $[2-\epsilon, 2)$, $[3-\epsilon, 3)$, and in general $[i-\epsilon, i)$ for every i.

Consider now the composition of Φ with a TS P that uses a_p as its scheduling variable. The composition of the two is a CTS in which moves of the TS P happen in

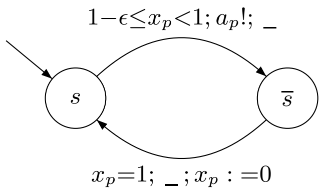


Fig. 3. CTS for one rate

the time range $[i-\epsilon, i)$ for every $i \in \mathbb{N}$. 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 CTSs 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 TSs 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). We can show that under appropriate projection, the sequence of actions taken by the composition of the four systems, is equivalent to the sequence of actions taken by the bounded-asynchronous composition of P and Q with rate 1.

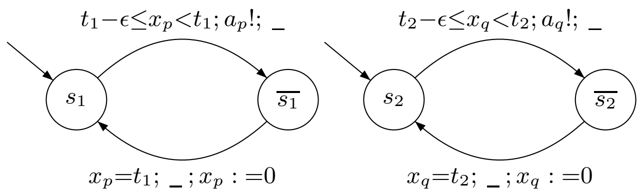


Fig. 4. CTSs for different rates

We now turn to consider the more general scheduler. Consider the CTSs in Fig. 4. They resemble the simple CTS presented above, however use the bounds of t_1 and t_2 time units, respectively. Denote the CTS using bound t_1 by Φ_1, and the CTS using bound t_2 by Φ_2. A computation of Φ_1 is a sequence of steps where time progresses until the range $[i \cdot t_1 - \epsilon, i \cdot t_1)$, then the system takes a step, then the time progresses until $i \cdot t_1$, and the system takes a step that resets the local clock. A computation of Φ_2 is similar, with t_2 replacing t_1.

Let P and Q be two TSs 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 units and

Q every t_2 time units[7]. Every t_1 time units P performs an action, and every t_2 time units Q performs an action. At time t such that both t_1 and t_2 divide t, both P and Q make moves, however, the order between P and Q is not determined. We can show that under appropriate projection, the sequence of actions taken by the composition of the four systems, is equivalent to the sequence of actions taken by the bounded-asynchronous composition of P and Q with rates t_1 and t_2, respectively.

A full formal treatment of these two systems, including proofs that the runs induced by them can be abstracted by bounded-asynchronous composition, is in Appendix B. It is an interesting question whether similar ideas can be used for the abstraction of real time and probabilistic systems.

6 Concluding Remarks

In this paper we introduce bounded asynchrony and study how to improve model checking if the communication between components is locally restricted, as in the interaction of adjacent cells. Bounded asynchrony allows processes to move separately but in a strongly coupled manner, disallowing interleavings where one process advances faster than the others. Bounded asynchrony naturally captures the phenomena of equally potent cells choosing distinct cell fates. It can also arise in the context of abstracting real-time systems of a certain type, which may help to explain the biological observations.

References

1. R. Alur, L. de Alfaro, R. Grosu, T.A. Henzinger, M. Kang, C.M. Kirsch, R. Majumdar, F.Y.C. Mang, and B.Y. Wang. jmocha: a model-checking tool that exploits design structure. In *Proceedings of the 23rd IEEE/ACM International Conference on Software Engineering*, pages 835–836. IEEE Computer Society Press, 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. G. Bernot, J.P. Comet, A. Richard, and J. Guespin. Application of formal methods to biological regulatory networks: extending thomas asynchronous logical approach with temporal logic. *Journal of Theoretical Biology*, 229(3):339–347, 2004.
5. 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.
6. L. Calzone, N. Chabrier-Rivier, F. Fages, and S. Soliman. Machine learning biochemical networks from temporal logic properties. *Transactions on Computational Systems Biology*, 4:68–94, 2006.
7. E.C. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
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. S. Efroni, D. Harel, and I.R. Cohen. Emergent dynamics of thymocyte development and lineage determination. *PLoS Computational Biology*, 3(1):127–136, 2007.

[7] 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, restricting t_1 and t_2 to range over integer values seems reasonable.

10. J. Fisher and T.A. Henzinger. Executable biology. In *Proc. of the Winter Simulation Conference - Track on Modeling and Simulation in Computational Biology*, pages 1675–82. IEEE press, 2006.
11. J. Fisher, N. Piterman, A. Hajnal, and T.A. Henzinger. Predictive modeling of signalling crosstalk during *C. elegans* vulval development. *PLoS Computational Biology*, 2007. To appear. http://mtc.epfl.ch/∼piterman/stuff/sbio.pdf.
12. 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.
13. D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. of Phys. Chemistry*, 81(25):2340–61, 1977.
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. Springer-Verlag, 2001.
15. G.J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Engineering*, 23(5):279–295, May 1997. Special issue on Formal Methods in Software Practice.
16. 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., 2003.
17. 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.
18. S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22(3):437–467, 1969.
19. 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.
20. 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.
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. Sadot, J. Fisher, D. Barak, Y. Admanit, M. J. Stern, E. J. Hubbard, and D. Harel. Towards verified biological models. *IEEE Transactions in Computational Biology and Bioinformatics*, 2007. To appear.
23. 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.
24. P. W. Sternberg and H. R. Horvitz. Pattern formation during vulval development in C. elegans. *Cell*, 44(5):761–72, 1986.
25. 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.
26. J. E. Sulston and J. G. White. Regulation and cell autonomy during postembryonic development of Caenorhabditis elegans. *Dev Biol*, 78(2):577–97, 1980.
27. 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.
28. 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 Model checking

In the following we prove Lemma 1.

Lemma 1. *1 Consider two $\mathcal{P}$-equivalent interleavings $\mathcal{I}$ and $\mathcal{I}'$. If s' is a macro-successor of s with respect to $\mathcal{I}$, then s' is a macro-successor of s with respect to $\mathcal{I}'$.*

Proof. Let $g : s_0, \ldots, s_m$ be a macro-step consistent with interleaving $\mathcal{I}$ such that $s = s_0$ and $s' = s_m$. Let $I_1, \ldots, I_{m'}$ denote the equivalence classes on processes induced by the interleaving $\mathcal{I}'$, that is, for every macro-step h consistent with $\mathcal{I}'$ and for every j, j', $a \in I_j$, and $a' \in I_{j'}$ we have $j = j'$ implies $i_{h,P_a} = i_{h,P_{a'}}$ and $j < j'$ implies $i_{h,P_a} < i_{h,P_{a'}}$. Let G_0 be the set of all macro-steps with initial state s consistent with interleaving $\mathcal{I}'$. For all $0 \leq i \leq m'$, let G_i be the set of all macro-steps $h : t_0, \ldots, t_{m'}$ in G_{i-1} such that for every TS $P_a \in I_i$ we have $t_i[W_k] = s'[W_k]$.

We prove by induction that $G_{m'}$ is not empty, implying that, for some macro-step g' in G_0, we have s' is the result of g'. Clearly, $G_0 \neq \emptyset$. Suppose that $G_{i-1} \neq \emptyset$. Consider the set I_i of processes that perform an action in the transition from t_{i-1} to t_i and consider some variable v read by a process P_a for $a \in I_i$. Let $P_{a'}$ be the process who owns v. By the definition of equivalent interleavings, it follows that P_a and $P_{a'}$ move in the same order in $\mathcal{I}$ and $\mathcal{I}'$. If $P_{a'}$ moves before P_a, then by induction, the value of $t_{i-1}[v] = s'[v]$. If $P_{a'}$ moves concurrently or after P_a, then $t_{i-1}[v] = s[v]$. In addition, for every process $P_{a'}$ such that $P_{a'} \notin I_i$ but $P_{a'}$ moves concurrently with P_a in $\mathcal{I}$, by the definition of equivalent interleavings, it follows that $C_{a',a}$ is not defined. Thus, the transition of P_a does not depend on the values set by $P_{a'}$ in $\mathcal{I}$. It follows that P_a has the same input when it moves in g as in t_{i-1}. It follows that every possible transition of the processes in I_i that is enabled for their concurrent move in g is enabled also in t_{i-1}, and thus we can find a transition such that for every $P_a \in I_i$ we have $t_i[W_a] = s'[W_a]$. ∎

We now describe formally the scheduler introduced in Section 4. We start from an abstraction A, whose transition graph is given in Fig. 5. A state of the scheduler is a valuation of the Boolean variables $b_1, \ldots, b_n$. The abstract states of A are labeled with patterns that describe the concrete states they represent.

The formal definition is as follows. The scheduler S' is $\langle V, W, \Theta, \rho \rangle$, where $V = W = \{b_1, \ldots, b_n\}$ and b_i is Boolean for all $1 \leq i \leq n$, $\Theta = \bigwedge_{i=1}^{n} \overline{b_i}$, 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. Let Ψ_0 denote the assertion that for no value of i, $\overline{b_i} b_{i+1}$ holds, and Ψ_1 denote the assertion that for exactly one value of i, $\overline{b_i} b_{i+1}$ holds. For simplicity of notation we denote false also by b_{n+1}.

$$\rho = \Omega \rightarrow \bigwedge_{i=1}^{n} \begin{pmatrix} (b_i \rightarrow b_i') & \wedge \\ (\overline{b_i} \wedge \Psi_1 \rightarrow b_i' \wedge (b_{i+1} \vee b_{i+1}')) & \wedge \\ (\overline{b_i} \wedge \overline{\Psi_1} \rightarrow b_i' \wedge (\Psi_0' \vee \Psi_1')) & \end{pmatrix} \quad \wedge \quad \overline{\Omega} \rightarrow \bigwedge_{i=1}^{n} \overline{b_i'} \tag{4}$$

The scheduler S' replaces the scheduler S defined in Section 2. The proof that S' includes one representative for every equivalence class of interleavings in S resembles the intuition explained in Section 4.

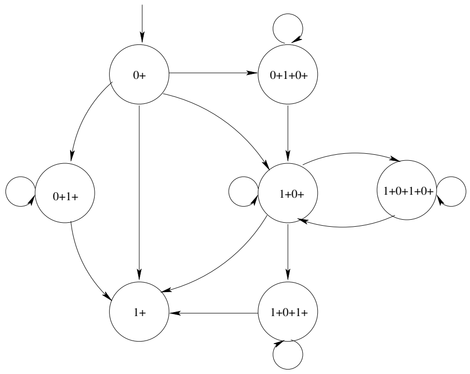


Fig. 5. Scheduler abstraction

Lemma 2. *For every interleaving $\mathcal{I}$ in S, there exists an equivalent interleaving $\mathcal{I}'$ in S'.*

Proof. Consider an interleaving $\mathcal{I}$ in S. Consider the minimal right-enabled channel c_{l_0} in $\mathcal{I}$ and let $k_1, k_2, \ldots, k_n$ be the left-enabled channels in $1, \ldots, l_0$. According to S' the processes $k_n + 1, \ldots, l_0$ can move concurrently in the first step. Consider the process $l_0 + 1$, as the channel c_{l_0} is right-enabled, it follows that $l_0 + 1$ moves after l_0 also in $\mathcal{I}$. Consider the process k_n, as the channel c_{k_n} is left-enabled, it follows that k_n moves after k_n+1 also in in $\mathcal{I}$. In addition, all the channels between processes in $k_n+1, \ldots, l_0$ are disabled (from minimality of l_0 and maximality of k_n), implying that $k_n + 1, \ldots l_0$ move concurrently also in $\mathcal{I}$. We then handle the processes in $1, \ldots, k_{n-1}$ according to the order $k_{n-1} + 1, \ldots k_n$, then $k_{n-2} + 1, \ldots, k_{n-1}$, etc. The equivalence on their order between $\mathcal{I}$ and $\mathcal{I}'$ follows from $k_1, \ldots, k_n$ being all the right-enabled channels in $1, \ldots, l_0$.

We handle the minimal right-enabled channel c_{l_1} such that $l_1 > l_0$ similarly. ∎

B Clocked Transition System Schedulers

We give a formal treatment of the clocked transition systems presented in Section 5. We start with a formal definition of clocked transition systems, proceed with a definition of the systems presented in Figures 3 and 4, and conclude with proofs that the bounded asynchronous composition presented in Section 2 abstracts these systems.

B.1 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.
- $W \subseteq D$: A set of *owned* variables. These are the variables that are actively changed by Φ. 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 $\Theta \rightarrow c{=}0$, 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 $\rho \rightarrow (c{=}c' \vee c'{=}0)$, 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 one of the following.
  - s_{j+1} is a ρ successor of s_j. In this case, j is a *discrete step*.
  - 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. In this case, j is a *time step*.
- *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.

A sequence of states (either finite or infinite) that satisfies only the consecution requirement is called a *partial run*.

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 $\Phi_1 \parallel \Phi_2$, 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 $\mathtt{true}$. In this case, the different definitions of parallel composition coincide.

B.2 Real-time Scheduler

Recall the CTS Φ depicted in Fig. 3. 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. More formally, consider the CTS $\Phi = \langle V, W, \Theta, \rho, \Pi\rangle$, where the components of Φ are as follows.

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

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. Let Φ_i denote a copy of Φ above where every component is indexed by i. Let Ψ denote the parallel composition of $P_1, \ldots, P_n$ with $\Phi_1, \ldots, \Phi_n$. Recall the scheduler $S = \langle V, W, \Theta, \rho\rangle$ defined in Section 2. Here $V = W = \{b_1, \ldots, b_n\}$, $\Theta = \bigwedge_{i=1}^{n} \overline{b_i}$, and ρ is as in Equation (1).

Let $\mathcal{P}$ denote the bounded-asynchronous composition of $P_1, \ldots, P_n$ (i.e., their composition with S and the additional conjunct in Equation (2)). We show that under appropriate projections the sets *runs*$(\mathcal{P})$ and *runs*(Ψ) 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 3. $\overline{\text{runs}(\mathcal{P})}\Downarrow_{V'} = \overline{\text{runs}(\Psi)}\Downarrow_{V'}$.

Given a run σ of Ψ, 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 Ψ 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 $r_0[b_i] = \mathbf{false}$. 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 $r_{m+1}[b_i] = \mathbf{false}$. 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 now turn to consider the more general scheduler. Recall the CTSs in Fig. 4 that use bounds t_1 and t_2. More formally, consider 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.

Let Ψ denote the parallel composition of $P_1, \ldots, P_n$ with $T(t_1), \ldots, T(t_n)$. Intuitively a run of Ψ 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 Ψ 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 $(s_{j-1}, s_j) \models W_i{=}W_i' \wedge a_i{=}a_i'$.

Recall the scheduler $S = \langle V, W, \Theta, \rho \rangle$ with rates $t_1, \ldots, t_n$ defined in Section 2. Here $V = W = \{b_1, \ldots, b_n, c\}$, $\Theta = c{=}1 \wedge \bigwedge_{i=1}^{n} \overline{b_i}$, and ρ is as in Equation (3).

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

Let $\mathcal{P}$ denote the bounded-asynchronous composition of P_1, ..., P_n according to the rates t_1, ..., t_n (i.e., their composition with S and the additional conjunct in Equation (2)). We show that under the appropriate projections the sets *runs*$(\mathcal{P})$ and *runs*(Ψ) 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, ..., P_n.

Lemma 4. $\overline{\text{runs}(\mathcal{P})}\Downarrow_{V'} = \overline{\text{runs}(\Psi)}\Downarrow_{V'}$.

As before, given a run σ of Ψ, 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 Ψ 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 $c \; mod \; t_i \neq 0$ 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. ∎