Timed Concurrent Constraint Programming for Analysing Biological Systems

Gutierrez, Julian; Pérez, Jorge A.; Rueda, Camilo; Valencia, Frank D.

Published in:
Electronic Notes in Theoretical Computer Science

DOI:
10.1016/j.entcs.2007.05.012

Publication date:
2007

Citation for published version (APA):
Gutierrez, J., Pérez, J. A., Rueda, C., & Valencia, F. D. (2007). Timed Concurrent Constraint Programming for Analysing Biological Systems. *Electronic Notes in Theoretical Computer Science*, *171*(2), 117-137. https://doi.org/10.1016/j.entcs.2007.05.012

Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 171 (2007) 117–137
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Timed Concurrent Constraint Programming for Analysing Biological Systems

Julian Gutiérrez[a], Jorge A. Pérez[a], Camilo Rueda[a] and Frank D. Valencia[b]

[a] *Department of Science and Engineering of Computing Pontificia Universidad Javeriana, Cali, Colombia*

[b] *CNRS and LIX, École Polytechnique, Palaiseau, France*

Abstract

In this paper we present our first approach to model and verify biological systems using `ntcc`, a concurrent constraint process calculus. We argue that the *partial information* constructs in `ntcc` can provide a suitable language for such systems. We also illustrate how `ntcc` may provide a *unified framework* for the analysis of biological systems, as they can be described, simulated and verified using the elements provided by the calculus.

Keywords: Process Calculi, Verification of Biological Systems, Partial Information, Concurrent Constraint Programming (CCP)

1 Introduction

Partial information arises naturally in the description of biological systems. It is possible to distinguish two main kinds of partial information when modeling those systems: *quantitative* and *behavioral.* While *partial quantitative information* usually involves incomplete information on the *state of the system* (e.g., the set of possible values that a variable can take), *partial behavioral information* refers to the uncertainty associated to behavior of interactions (e.g., the unknown relative speeds on which two systems interact). Finding precise ways of expressing these kinds of partial information can help to better understand complex pattern behaviors, frequent in biological systems.

Partial information is a central feature of Concurrent Constraint Programming (CCP) [19], a well-established formalism for concurrency. In CCP, processes interact with each other by telling and asking partial information represented as *constraints* (e.g., $x < 42$). Perhaps the most appealing and distinctive feature of CCP is that it combines the traditional *operational* view of process calculi with a *declarative* one

doi:10.1016/j.entcs.2007.05.012

based upon logic. In other words, the process terms can be viewed at the same time as computing agents and logic formulas. This combination allows CCP to benefit from the large body of techniques of both process calculi and logic. For these reasons CCP can be a convenient framework to describe and reason about biological systems.

In this paper we propose `ntcc` [14], a timed process calculus based on CCP, as a suitable language for analyzing biological systems. In `ntcc` the above-mentioned kinds of partial information are naturally captured. On the one hand, partial quantitative information is captured by the notion of *constraint system*, a structure that gives coherence and defines (logic) inference capabilities over constraints. Since constraint systems are *parametric* to `ntcc`, by choosing the appropriate constraint system(s) several kinds of conditions, at different levels of detail, can be stated. This could be particularly useful in the description of quantitative information. For instance, one could think of a constraint system over differential equations interacting with others over, say, integers or real intervals. On the other hand, partial behavioral information is represented by *non-deterministic and asynchronous operators* available in `ntcc`. The interplay of these operators in the discrete time of `ntcc` allows to explicitly describe and reason about the uncertainty in the time occurrence of many biological phenomena.

Furthermore, `ntcc` provides reasoning techniques to prove that a given process P satisfy a given property F. In fact, the calculus offers a linear-temporal specification logic and its corresponding proof system in which reachability analysis can be formally carried out. Reachability analysis is central in the biological context. Consider, for instance, bacterial transcription: it can be seen as a reachability analysis problem in which one wants to know if there is a gene expression possible in a given gene regulatory network.

We shall take advantage of these features by modeling biological systems as processes and their properties as linear-temporal formulas, all *in a single framework*. That is, `ntcc` provides a description language for biological systems that is tightly related to powerful reasoning techniques. An additional advantage of using `ntcc` for the study of biological systems consists in the possibilities of turning this theoretical framework into software tools. As a matter of fact, the AVISPA Research Group [1] (of which the authors are members) has recently built a prototype tool [2,3] that admits the description of biological systems expressed as `ntcc` processes and allows to observe their behavior over time.

The main contribution of this paper is presenting `ntcc` as a *unified framework* for the study of biological systems involving partial information and showing how its constructs naturally capture many biological phenomena. More specifically, we propose the use of *constraint systems* to represent partial *quantitative* information and the modeling of partial *behavioral* information as *non-deterministic and asynchronous* `ntcc` processes. We take the Sodium-Potassium pump [20], a mechanism that influence active transport in eukaryote cells, as a compelling example of the applicability of our approach. In fact, we will use the inference system to give a

[1] URL: `http://avispa.puj.edu.co`

proof of the occurrence of a general malfunction of the pump in the presence of an unpredictable, malicious agent.

Related Work

The use of certain process calculi, such as the π-calculus [17, 18], BioAmbients [16], the Brane calculus [7], Beta binders [15] and the κ-calculus [8], as description languages for Biology has been studied in recent years. This "language approach" for the analysis of biological systems however, has payed little attention to reasoning techniques based on linear-temporal logic such as those available in `ntcc`. Other constraint-based calculi have been studied in the biological context. For instance, in [5, 10, 4], the hcc calculus [11] is used to study dynamic systems. However, since hcc does not provide non-deterministic/asynchronous operators, representing partial behavioral information turns out to be difficult. Only in one of such works ([5]), the logic nature of hcc is exploited, using a model-checking approach for qualitative validation of biological systems. No proof system or similar procedures are used, though. Other works involving the use of logic in the biological context are [1] and [6]. On the one hand, [1] proposes the use of hybrid automata to model and analyze the behavior of biological systems. Supporting tools such as Simpathica [13], allow to query such models using a temporal logic language. On the other hand, in [6] a rule-based language for describing biological systems is proposed. Reasoning techniques include three independent semantic structures (each one with associated logics), which are used depending on the desired level of detail. We believe that by the appropriate use of constraint systems in the description of systems, analysis at several levels of detail are possible, preserving the *same unified framework*.

Structure of this document

The `ntcc` process calculus is described next: the intuitions given above, regarding the use of `ntcc` for the modeling and verification of biological systems, are thoroughly explained. Section 3 summarizes the main results concerning specification and verification for `ntcc` processes. They will be used in Section 4 where the Sodium-Potassium pump is presented. In that section, we propose an `ntcc` model of such a system as well as verify a non-trivial property of this model, using the above-mentioned inference system. Section 5 concludes.

2 `ntcc` as a Calculus for Describing Biological Systems

In this section we present the `ntcc` process calculus and, by means of examples, show how it can be an appropriate language for modeling biological phenomena. For the sake of space, some formal details are elided from this presentation; an in-depth description of `ntcc` is given in [14].

Let us start with an intuitive description of *reactive computation* in `ntcc`. In `ntcc`, time is conceptually divided into *discrete intervals (or time units)*. In a particular time unit, a process P gets an input (an item of information represented as a *constraint*) c from the environment, it executes with this input as the initial *store*, and when it reaches its resting point, it *outputs* the resulting store d to the environment. The resting point determines a residual process Q, which is then executed in the next time unit. Information is not automatically transferred from one time unit to the following.

In CCP, a fundamental notion is that of a *constraint system*. Intuitively, a constraint system provides a signature from which constraints can be constructed, and an entailment relation which specifies the inter-dependencies among them. More formally, a constraint system is a pair (Σ, Δ) where Σ is a signature of function and predicate symbols, and Δ is a decidable theory over Σ. Given a constraint system (Σ, Δ), let $(\Sigma, \mathcal{V}, \mathcal{S})$ be its underlying first-order language, where $\mathcal{V}$ is a set of variables $x, y, \ldots$, and $\mathcal{S}$ is the set of logic symbols $\neg, \wedge, \vee, \Rightarrow, \exists, \forall,$ `true` and `false`. *Constraints* $c, d, \ldots$ are formulas over this first-order language. We say that c *entails* d in Δ, written $c \models d$, iff $c \Rightarrow d$ is true in all models of Δ. For operational reasons, we shall require $\models$ to be decidable. Henceforth, $\mathcal{C}$ denotes the set of constraints in the underlying constraint system.

A widely known constraint system is FD [12]. In FD variables are assumed to range over finite domains and, in addition to equality, we may have predicates that restrict the possible values of a variable to some finite set. More formally, FD$[n]$ $(n > 0)$ is the constraint system where Σ is given by the constant symbols $0, \ldots, n-1$ as well as by the equality $=$, and Δ is given by the axioms of equational theory $x = x$, $x = y \Rightarrow y = x$, $x = y \wedge y = z \Rightarrow x = z$, and $v = w \Rightarrow$ `false` for each two different constants $v, w \in \Sigma$. Intuitively FD$[n]$ provides a theory of variables ranging over a finite domain of values $\{0, \ldots, n-1\}$ with syntactic equality over these values.

2.1 Process Syntax

Processes P, Q, $\ldots \in$ *Proc* are built from constraints $c \in \mathcal{C}$ and variables $x \in \mathcal{V}$ in the underlying constraint system by:

$$\begin{array}{rlllll} P, Q, \ldots ::= & \mathbf{tell}(c) & \mid \sum\limits_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i & \mid P \parallel Q & \mid \mathbf{local}\ x\ \mathbf{in}\ P \\ \mid & \mathbf{next}\,(P) & \mid \mathbf{unless}\ c\ \mathbf{next}\, P & \mid \star\ P & \mid\ !\, P \end{array}$$

Below we provide some intuitions regarding the behavior of `ntcc` processes.

Including and Querying (Partial) Information

Process $\mathbf{tell}(c)$, the simplest operation to express *partial information*, includes a constraint c into the current store, thus making it available to other processes in the same time interval.

In the biological context, **tell** operations allow to represent at least two kinds of *partial information* statements: so-called *ground rules* and *state definition* statements. The first ones precisely state certain conditions that apply during the life of the biological system. A clear advantage here w.r.t. other calculi for biology is that these conditions can be expressed by exploiting the available (possibly incomplete) knowledge.

Example 2.1 Let process $M = \textbf{tell}(l < pH_{in} < u)$ represent a rule establishing the acceptable levels of internal pH for some system. It establishes that such a level must fall into some real interval (here given by variables l and u) during the whole experiment or simulation; the exact value of pH_{in} in each time unit could be unknown.

Remarkably, the *declarative flavor* in this kind of statements could favor the definition of essential properties in (biological) models. Complementary to ground rules, *state definition* statements refers to those constraints intended to define the exact values for the variables in the system. This is particularly useful when one exactly knows the set of possible states for the system at a given time; series of such statements (for different time units) thus constitute a detailed view of the behavior of the system. In the context of Example 2.1, $M' = \textbf{tell}(pH_{in} = f(pH_{old}, k))$ is a process defining the value for the variable pH_{in} in the current time unit. It associates such a value with a function f applied to a variable and a constant k.

Guarded operations of the form **when** c **do** P are complementary to **tell** operations and constitute the basic means for *querying* (or *asking*) information about the state of a system. Intuitively, a **when** c **do** P process queries the current constraint store: if the guard c is present in such a store then the execution of P is enabled. The "presence" of c depends on the inference capabilities associated with the store. That is, a particular constraint could not be explicitly present in the store, but it could be inferred from the available information.

From this description, it is straightforward to interpret **when** operations as a way of formally expressing the required *preconditions* for establishing a particular state of the system. The behavior of the system can be precisely stated in this way. Returning to Example 2.1, one could express that when the level of pH reaches a threshold, then the interval for valid values for pH_{in} should reduce, i.e., **when** $pH_{in} > l * 2$ **do** $\textbf{tell}(u = u - k_1)$.

Non-deterministic Choices

Non-determinism is a valuable way of representing several possible courses of action from the same initial state without providing any information on how one of such courses is selected. In `ntcc`, non-deterministic behavior is obtained by generalizing processes of the form **when** c **do** P: a *guarded-choice summation* $\sum_{i\in I}$ **when** c_i **do** P_i, where I is a finite set of indexes, represents a process that, in the current time interval, must non-deterministically choose one of the P_j $(j \in I)$ whose corresponding constraint c_j is entailed by the store. The chosen alternative, if any, precludes the others. If no choice is possible then the summation is precluded. We use $\sum_{i\in I} P_i$ as an abbreviation for the "blind-choice" process $\sum_{i\in I}$ **when** `true` **do** P_i. We use **skip** as an abbreviation of the empty summation and "+" for binary summations.

In the biological context, the combination of guarded choices and partial information represents an appropriate mechanism to formalize the inherent *unpredictability* in system interactions. In this sense, non-determinism is one way of explicitly representing *partial behavioral information.* The following example illustrates these ideas.

Example 2.2 Process P below is an abstract model of a biological system: in the presence of a certain amount of ATP (i.e., energy) the system releases an enzyme; in the case some ATP is present and the conditions of some electrochemical gradient are appropriate, it emits a positive signal:

$$P = \begin{array}{l} \textbf{when } \mathrm{ATP} > 0 \textbf{ do } \textbf{tell}(\mathrm{releaseEnzyme} = 1) \\ + \\ \textbf{when } \mathrm{ATP} > 0 \ \wedge \ \mathrm{elecGradient} = 1 \textbf{ do } \textbf{tell}(\mathrm{emitSignal} = 1). \end{array}$$

The evolution of P depends on the information in the current store. The simplest case is with the (empty) store `true`: P cannot add any further information. In the store $d = (\mathrm{ATP} \geq 50)$, P causes the store to become $d \wedge (\mathrm{releaseEnzyme} = 1)$ since in the first alternative it holds that $(\mathrm{ATP} \geq 50) \models (\mathrm{ATP} > 0)$ and the guard of the second alternative does not entail from d. The interesting case is when both guards in P are enabled; as in the store $e = (\mathrm{ATP} > 0) \ \wedge \ (\mathrm{elecGradient} = 1)$. Depending on which process is chosen for execution, the final store could be either $e \wedge (\mathrm{releaseEnzyme} = 1)$ or $e \wedge (\mathrm{emitSignal} = 1)$. Based on partial information, P constitutes a succinct representation of an unpredictable behavior.

Communication

Process $P \parallel Q$ represents the parallel composition of P and Q. In one time unit P and Q operate concurrently, "communicating" via the common store by adding and querying information. We use $\prod_{i \in I} P_i$, where I is a finite set of indexes, to denote the parallel composition of all P_i.

Example 2.3 Assume process P as in the Example 2.2 and the following process Q:

$$Q = \begin{array}{l} \textbf{when } \mathrm{releaseEnzyme} = 1 \textbf{ do } \textbf{tell}(\mathrm{promoteReaction} = 1) \\ + \\ \textbf{when } \mathrm{emitSignal} = 1 \textbf{ do } \textbf{tell}(\mathrm{promoteReaction} = 0). \end{array}$$

Informally, Q promotes a reaction to occur once the presence of an enzyme has been detected and opposes to such a reaction if a particular signaling process has been activated. The parallel composition $P \parallel Q$ in the store $e = (\mathrm{ATP} > 0) \ \wedge \ (\mathrm{elecGradient} = 1)$ behaves as follows. Since the choice in P guarantees the presence of either $\mathrm{releaseEnzime} = 1$ or $\mathrm{emitSignal} = 1$, process $P \parallel Q$ would cause the store to become either $e \wedge (\mathrm{releaseEnzyme} = 1) \wedge (\mathrm{promoteReaction} = 1)$ or $e \wedge (\mathrm{emitSignal} = 1) \wedge (\mathrm{promoteReaction} = 0)$.

Local Information

In ntcc, as in most process calculi, there is a construct that restricts the interface through which a process can interact with each other, thus allowing for the modeling of local behavior. Processes of the form **local** x **in** P behave like P, except that all the information on x produced by P can only be seen by P and the information on x produced by other processes cannot be seen by P.

In addition to the conventional spirit of this kind of operators, in the context of partial information, local information may represent a valuable help in the analysis of systems. When performing overall analyzes of complex systems, local variables may help to "hide" the behavior of such components that are irrelevant in the interactions to be analyzed.

Example 2.4 Consider a complex system (e.g., a cell) represented by a process C. Assume that the definition of C involves a set of variables $X = \{x_1, x_2, \ldots, x_n\}$ which represent some features of interest. In this way, in a "standalone" analysis of C, variables in X would give a comprehensive view of its behavior over time.

Assume now that we are interested in a process T consisting in the interaction of a large number of identical cells, i.e., $T = C_1 \parallel \ldots \parallel C_m$. In this case, as the focus of the analysis has moved from a local level (a single cell) to a global one (a tissue), it is necessary to abstract from the behavior induced by those variables in each X_i (associated with C_i) that do not participate in the interaction that is being modeled. Let $X_i \supset X_i^* = \{x_1^*, x_2^*, \ldots, x_n^*\}$ be the set containing those "irrelevant" variables[2]. Therefore, each cell C_i could be better represented as $C_i{}^* =$ **local** $x_1^*, x_2^*, \ldots, x_n^*$ **in** C_i[3], and the process $T^* = C_1{}^* \parallel \ldots \parallel C_m{}^*$ would represent cells' interaction.

Note that the internal structure of each cell remains unchanged by this hiding. Further, from an operational point of view, such a hiding is required to preserve the coherence in the values observed from P: an inconsistency may arise as each C_i can assign a different value to each x_i.

From the example, it is possible to observe how the interplay of hiding and the notion of partial information may allow to analyze systems at different levels of detail.

Basic Timed Behavior

ntcc provides two basic time operators: **next** (P) and **unless** c **next** (P). Let us analyze them separately. **next** (P) represents the activation of P in the next time interval. Hence, a move of **next** (P) is a unit-delay of P. **next** (P) can be also considered as the simplest way of expressing the dynamical behavior over time. This is fundamental in ntcc, since information is *not automatically* transferred from one time interval to the next. Building up on **next** (P), it is easy to think

[2] Note that X_i^* should not contain the same variables that X_i since this would represent that every cell is isolated from each other.

[3] Notation **local** $x_1, \ldots, x_n$ **in** P abbreviates the process **local** x_1 **in** (**local** x_2 **in** $(\ldots($**local** x_n **in** $P)\ldots))$.

in more sophisticated delay constructs: we use $\mathbf{next}^n\ (P)$ as an abbreviation for $\mathbf{next}\ (\mathbf{next}\ (\ldots \mathbf{next}\ (P))\ldots))$, where **next** is repeated n times.

In the context of partial information, to be able of reasoning about *absence* of information is both important and necessary. Although sometimes it is possible to predict some of the possible future states for a system, usually there is a strong need of expressing *unexpected behavior.* In this kind of scenarios, processes of the form $\mathbf{unless}\ c\ \mathbf{next}\, P$ may come in handy: P will be activated only if c cannot be inferred from the current store. The "unless" processes thus add (weak) time-outs to the calculus, i.e., they wait one time unit for a piece of information c to be present and if it is not, they trigger activity in the next time interval. To illustrate this consider the example below.

Example 2.5 Process $R = \mathbf{when}\ a\ \mathbf{do}\ P_1 + \mathbf{when}\ b\ \mathbf{do}\ P_2 + \mathbf{when}\ c\ \mathbf{do}\ P_3$ models the prediction of three possible evolutions for a system (i.e., P_1, P_2 and P_3). Notice that since they might be just a small part of a complex behavior that is partially understood, a considerable amount of uncertainty has not been included. Defining a process $R^* = R \parallel \mathbf{unless}\ (a \vee b \vee c)\ \mathbf{next}\, S$ would ensure that in the case of a stimuli different from a, b or c occurs, a consistent default state in the system (here represented by S) will be preserved.

Definitions following this style of modeling not only allow more complete models but also permit to exploit the advantages of counting with partial information in a safe manner.

Asynchrony

The $\star$ operator allows to express asynchronous behavior through the time intervals. Process $\star P$ represents an arbitrary long but finite delay for the activation of P. For example, the process $D = \star\mathbf{tell}(\text{enzymeReleased} = 1)$ could represent the eventual presence of a particular enzyme in the environment, but without providing an upper bound on when such a thing will actually occur.

This kind of asynchronous behavior therefore constitutes another instance of partial behavioral information: in addition to the partial information *on the variables* that are part of the state of the system (and that is expressed by the operators discussed above), the $\star$ operator allows to express partial information *on the time units* where processes are executed. This is particularly interesting when describing (biological) processes that interact at *unknown relative speeds.* For instance, a process $D \parallel S$ (with D defined as above) could represent a flexible representation of the interaction between a system S (which may require the presence of the enzyme) and the process which ensures the arrival of such an enzyme.

The partial information spirit of the asynchronous behavior in `ntcc` is strengthened by the following derived operator, expressing *bounded eventuality*:

$$\star_{[n,m]}\ P = \mathbf{next}^n\ (P) + \mathbf{next}^{n+1}\ (P) + \cdots + \mathbf{next}^{m-1}\ (P) + \mathbf{next}^m\ (P).$$

This operator thus represents an additional amount of temporal (partial) information, as it ensures that P will be activated at some point within the time units in the closed interval of naturals $[n, m]$. As in the original operator, there is no additional information of when this restricted eventuality will take place.

Persistent Behavior

Somehow opposed to the eventual behavior enforced by asynchronous behavior, *persistent* (or infinite) behavior serves to express conditions that are valid during every possible state of the system. The replication operator $!P$ represents $P \parallel \mathbf{next}\ (P) \parallel \mathbf{next}^2(P) \parallel \ldots$, i.e. unboundedly many copies of P but one at a time. As such, persistent behavior is an appropriate way of enforcing conditions stating ground rules of the systems of interest.

A process illustrating this kind of behavior is $D' = !\ \mathbf{tell}(\text{enzymeReleased} = 1)$, the persistent version of the enzyme-related signal. D' simply represents the fact that in every future time unit the constraint it involves will be available. Persistent behavior can also be understood as a mechanism that allows to move from *static* descriptions or conditions (valid only in one state of the system) to *dynamic* statements that are always valid.

As in the asynchronous case, it is possible to derive a bounded version of the persistent operator:

$$!_{[n,m]}\ P = \mathbf{next}^n\ (P) \parallel \mathbf{next}^{n+1}\ (P) \parallel \cdots \parallel \mathbf{next}^{m-1}\ (P) \parallel \mathbf{next}^m\ (P).$$

This operator represents the fact that P is always active during all the time units in the interval $[n, m]$. As its eventual counterpart, this derived operator (known as *bounded invariance*) may come in handy when certain additional information regarding the (persistent) execution of P is available.

2.2 Operational Semantics

The intuitive behavior for `ntcc` processes described above is formalized by means of a structural operational semantics (SOS) that considers *transitions* between process-store *configurations* of the form $\langle P, c\rangle$ with stores represented as constraints. The transitions of the SOS are given by the relations $\longrightarrow$ and $\Longrightarrow$. They are formally defined in Appendix A. Intuitively, the *internal* transition $\langle P, d\rangle \longrightarrow \langle P', d'\rangle$ should be read as "P with store d reduces, in one internal step, to P' with store d' ". The *observable transition* $P \xRightarrow{(c,d)} R$ should be read as "P on input c, reduces in one *time unit* to R and outputs d". The observable transitions are obtained from terminating sequences of internal transitions.

Let us now consider an infinite sequence of observable transitions (or *run*) $P = P_1 \xRightarrow{(s_1,r_1)} P_2 \xRightarrow{(s_2,r_2)} P_3 \xRightarrow{(s_3,r_3)} \ldots$. This sequence can be interpreted as an *interaction* between the system P and an environment. At a time unit i, the environment provides a stimulus s_i and P_i produces r_i as a response. If $\alpha = s_1.s_2.s_3\ldots$ and $\alpha' = r_1.r_2.r_3\ldots$, then the above interaction is represented as $P \xRightarrow{(\alpha,\alpha')}{}^{\omega}$.

Alternatively, if $\alpha = \mathtt{true}^\omega$, we can interpret the run as an interaction among the parallel components in P without the influence of an external environment (i.e., each component is part of the environment of the others). In this case α is called the *empty* input sequence and α' is regarded as a *timed observation* of such an interaction in P. We will say that the *strongest postcondition* of a process P, denoted $sp(P)$, denotes the set of all infinite sequences that P can possibly output. More precisely, $sp(P) = \{\alpha' \mid \text{for some } \alpha : P \xRightarrow{(\alpha,\alpha')}{}^{\omega}\}$.

3 Specification and Verification for ntcc Processes

In this section we summarize some results regarding to Linear Temporal Logic (LTL) associated to `ntcc`. This particular LTL expresses properties over sequences of constraints and we shall refer to it as **CLTL**. A sound, partially complete proof system for this logic is also described. Further details of this logic (including decidability results) can be found in [14, 21].

The importance of the strong relationship between **CLTL** and `ntcc` is that a *logic-based* methodology for verification of properties of biological systems can be adopted, in addition to the observational approach that is induced by the operational semantics given above. That is, *simulations* of an `ntcc` process (i.e., its timed observations) could be complemented by proofs of essential properties (stated as temporal formulas).

We begin giving the syntax of LTL formulas and then interpret them with the **CLTL** semantics. The formulas $F, G, ... \in \mathcal{F}$ are built from constraints $c \in \mathcal{C}$ and variables $x \in \mathcal{V}$ in the underlying constraint system by:

$$F, G, \ldots := c \mid \dot{\mathtt{true}} \mid \dot{\mathtt{false}} \mid F \mathbin{\dot\wedge} G \mid F \mathbin{\dot\vee} G \mid \dot\neg F \mid \dot\exists_x F \mid \circ F \mid \Box F \mid \Diamond F$$

The constraint c (i.e., a first-order formula in the constraint system) represents a *state formula.* The dotted symbols represent the usual (temporal) Boolean and existential operators. The dotted notation is needed as in **CLTL** these operators do not always coincide with those in the constraint system. The symbols $\circ$, $\Box$, and $\Diamond$ denote the LTL modalities *next*, *always* and *eventually.* We use $F \mathbin{\dot\Rightarrow} G$ for $\dot\neg F \mathbin{\dot\vee} G$. Below we give the formulas a **CLTL** semantics. We first introduce some notation and the notion of *x-variant.* Intuitively, d is an x-variant of c iff they are the same except for the information about x. More formally, given a sequence $\alpha = c_1.c_2.\ldots$, we use $\exists_x\alpha$ to denote the sequence $\exists_x c_1 \exists_x c_2 \ldots$. We shall use $\alpha(i)$ to denote the $i - th$ element of α.

Definition 3.1 [x-variant] A constraint d is an *x-variant* of c iff $\exists_x c = \exists_x d$. Similarly α' is an *x-variant* of α iff $\exists_x \alpha = \exists_x \alpha'$.

Definition 3.2 [**CLTL** Semantics] We say that α satisfies (or that it is a model of) F in **CLTL** , written $\alpha \models_{\text{CLTL}} F$, iff $\langle \alpha, 1\rangle \models_{\text{CLTL}} F$, where:

$\langle \alpha, i\rangle \models_{\text{CLTL}} \dot{\mathtt{true}}$		$\langle \alpha, i\rangle \not\models_{\text{CLTL}} \dot{\mathtt{false}}$
$\langle \alpha, i\rangle \models_{\text{CLTL}} c$	iff	$\alpha(i) \models c$
$\langle \alpha, i\rangle \models_{\text{CLTL}} \dot{\neg} F$	iff	$\langle \alpha, i\rangle \not\models_{\text{CLTL}} F$
$\langle \alpha, i\rangle \models_{\text{CLTL}} F \dot{\wedge} G$	iff	$\langle \alpha, i\rangle \models_{\text{CLTL}} F$ and $\langle \alpha, i\rangle \models_{\text{CLTL}} G$
$\langle \alpha, i\rangle \models_{\text{CLTL}} F \dot{\vee} G$	iff	$\langle \alpha, i\rangle \models_{\text{CLTL}} F$ or $\langle \alpha, i\rangle \models_{\text{CLTL}} G$
$\langle \alpha, i\rangle \models_{\text{CLTL}} \bigcirc F$	iff	$\langle \alpha, i+1\rangle \models_{\text{CLTL}} F$
$\langle \alpha, i\rangle \models_{\text{CLTL}} \Box F$	iff	for all $j \geq i$ $\langle \alpha, j\rangle \models_{\text{CLTL}} F$
$\langle \alpha, i\rangle \models_{\text{CLTL}} \Diamond F$	iff	there is a $j \geq i$ such that $\langle \alpha, j\rangle \models_{\text{CLTL}} F$
$\langle \alpha, i\rangle \models_{\text{CLTL}} \dot{\exists}_x F$	iff	there is an x-variant α' of α such that $\langle \alpha', i\rangle \models_{\text{CLTL}} F$.

Define $[\![F]\!]=\{\alpha \,|\, \alpha \models_{\text{CLTL}} F\}$. F is **CLTL** *valid* iff $[\![F]\!] = \mathcal{C}^\omega$, and **CLTL** *satisfiable* iff $[\![F]\!] \neq \emptyset$.

Process Verification.

Intuitively, $P \models_{\text{CLTL}} F$ iff every sequence that P can possibly output, on inputs from arbitrary environments, satisfies F.

Definition 3.3 We say that a process P *satisfies* F, written $P \models_{\text{CLTL}} F$, iff $sp(P) \subseteq [\![F]\!]$.

Example 3.4 Assume $R = \star\,\mathbf{tell}(c)$ and $F = \Diamond c$. Then $R \models_{\text{CLTL}} F$ as in every sequence output by R there must be an e entailing c. Also $P = \mathbf{tell}(c) + \mathbf{tell}(d) \models_{\text{CLTL}} c \vee d$ and $P \models_{\text{CLTL}} c \,\dot{\vee}\, d$ as every e output by P entails either c or d. Notice, however, that $Q = \mathbf{tell}(c \vee d) \models_{\text{CLTL}} c \vee d$ but $Q \not\models_{\text{CLTL}} (c \,\dot{\vee}\, d)$ in general, since Q can output an e which certainly entails $c \vee d$ and still entails neither c nor d —take $c = (x = 42), d = (x \neq 42)$ and $e = c \vee d$. Therefore, $c \,\dot{\vee}\, d$ distinguishes P from Q.

In order to reason about statements of the form $P \models_{\text{CLTL}} F$, `ntcc` is equipped with a *proof (or inference) system* for assertions of the form $P \vdash F$. The system is presented in Table 1. We say that $P \vdash F$ iff the assertion $P \vdash F$ has a proof in the system in Table 1. The assertion $P \vdash F$ is intended to be the "counterpart" of $P \models F$ in the sense that $P \vdash F$ should approximate $P \models_{\text{CLTL}} F$ as closely as possible (ideally, they should be equivalent). The following proposition from [14] states the correspondence between $\models$ and $\vdash$. We say that a process P is locally independent iff the guards of every non-unary sum in P contains no local variables.

Proposition 3.5 *(Soundness) If $P \vdash F$ then $P \models F$. Furthermore, (Completeness) if P is locally-independent and $P \models F$ then $P \vdash F$.*

Hence the proof system is sound, and also complete for locally independent processes —which represent a substantial family of `ntcc` processes. It is worth noticing that our compelling example is in fact locally independent. Finally, the following lemma will be useful in derivations (see [14] for further details):

Lemma 3.6 *For every process P,*

1. $P \vdash \dot{\mathtt{true}}$, 2. $P \not\vdash \dot{\mathtt{false}}$, 3. $\dfrac{P \vdash A}{P \parallel Q \vdash A}$ *and* 4. $\dfrac{P \vdash A \qquad P \vdash B}{P \vdash A \,\dot{\wedge}\, B}$.

LTELL	$\mathbf{tell}(c) \vdash c$	LSUM	$\dfrac{\forall i \in I \quad P_i \vdash A_i}{\sum_{i\in I} \mathbf{when}\ c_i\ \mathbf{do}\ \ P_i \vdash \dot{\bigvee}_{i\in I}(c_i \mathrel{\dot\wedge} A_i) \mathrel{\dot\vee} \dot{\bigwedge}_{i\in I} \dot{\neg}\, c_i}$			
LPAR	$\dfrac{P \vdash A \quad Q \vdash B}{P \parallel Q \vdash A \mathrel{\dot\wedge} B}$	LUNL	$\dfrac{P \vdash A}{\mathbf{unless}\ \ c\ \ \mathbf{next}\ \ \ P \vdash c \mathrel{\dot\vee} \bigcirc A}$			
LREP	$\dfrac{P \vdash A}{!P \vdash \Box A}$	LLOC	$\dfrac{P \vdash A}{\mathbf{local}\ x\ \mathbf{in}\ P \vdash \dot{\exists}_x\, A}$			
LSTAR	$\dfrac{P \vdash A}{\star P \vdash \Diamond A}$	LNEXT	$\dfrac{P \vdash A}{\mathbf{next}\ (P) \vdash \bigcirc A}$	LCONS	$\dfrac{P \vdash A}{P \vdash B}$	if $A \mathrel{\dot\Rightarrow} B$

Table 1
A proof system for (linear-temporal) properties of `ntcc` processes

4 Analysing a Biological System in `ntcc`

In this section we show the use of our approach to model and verify biological systems using the *Sodium-Potassium pump* as case study. We first give a short biological description of the system and propose an `ntcc` model representing its behavior. Later, we verify a non-trivial property over this model using the `ntcc` reasoning techniques.

4.1 Biological Description

An *ion pump* is a natural channel connecting the two sides of a membrane. The function of these pumps is to move ions across the membrane in a process called *transport*. Depending on the source of the required energy, the transport can be either *passive* or *active*. In passive transport ions freely move across the membrane following an electrochemical gradient. As ions move in the direction of the gradient then the cell does not need to provide energy for the transport. Since in active transport ions move against the direction of the gradient, the cell has to supply energy (usually in form of ATP) to accomplish this movement.

In particular, the Sodium-Potassium pump [20] (SP-pump in the sequel) is a system for active transport of ions in animal eukaryotic cells. It exchanges Sodium ions inside the cell with Potassium ions outside of it. The pump is composed of two proteins known as the alpha and beta subunits. The purpose of the pump is to keep the concentration of sodium inside the cell lower than outside. This difference of concentrations generates an electrochemical gradient that leads the passive transport of Sodium ions towards the cytoplasm in the cell. If the pump does not work well then the gradient becomes weak for transport, thus affecting the entrance of required substances into the cell.

The pumping process in the SP-pump can be divided in six phases. At the beginning there is a pump conformation with high affinity for Sodium ions inside the cell (1). This conformation encourages the binding of three Sodium ions with the pump. Then the alpha subunit is phosphorylated by ATP hydrolysis (2), leaving a residual ADP molecule in the cytoplasm. This chemical reaction provides the needed energy for the pumping process. Once this occurs, the pump conformation changes and then the Sodium ions can leave the cell (3).

At this point, there is a pump conformation with high affinity for Potassium ions outside the cell (4). This results in the binding of two Potassium ions with the pump. Hence, the alpha subunit is dephosphorylated (5) and the pump conformation returns to the initial state. At this moment Potassium ions can enter the cell (6). The pumping process is always performed regulating the concentration of Sodium in the cell.

In parallel to this active transport movement, there is a *passive* transport movement that allows Potassium and Sodium ions to move against the direction of the active transport. This complementary movement is induced by an electrochemical gradient present in the cell.

4.2 An `ntcc` *model of the SP-pump*

Here we propose an `ntcc` model of the SP-pump. We use non-deterministic and asynchronous behavior for modeling partial behavioral information regarding temporal responses of certain components. Before entering into the detailed description of the model let us informally describe two encodings for recursive functions and mutable entities that will allow for cleaner model descriptions. A detailed account of their definition can be found in [14].

Recursive Definitions It is possible to encode recursive definitions of the form $q(x) \stackrel{\text{def}}{=} P_q$; where q is the process name and P_q calls q only once and such a call must be within the scope of a "**next**". Moreover, we can rely on the usual intuitions concerning procedure calls in a programming language.

Cells Using the basic `ntcc` syntax it is possible to provide *cells*, a basis for the specification and analysis of mutable and persistent data structures. A *cell* can be thought of as a structure that contains a value, and if tested, it yields this value. A cell keeps its value over the time units until it is modified. We use notations $x : v$ and $x := v$ to represent the *initialization* and the *assignment* of a cell x with value v, respectively. Also, we shall use notation $x := x + z$ as an abbreviation of the assignment $x := x' + z$, where x' is the value of the cell x in the previous time unit and z is a fixed value. The operation $x := x - z$ can be encoded analogously.

We now enter to describe the `ntcc` model representing the SP-pump, which is presented in Figures 1 and 2. Let us first describe the main principles underlying its modeling. The model assumes a constraint system over finite domains of integers, considering three places for interaction: inside and outside the cell, and an intermediate place where ions stay before entering or flowing out of the cell (i.e., the pump). The model involves a series of persistent variables (modeled as cells) that store useful quantities about the pumping process. Output and input operations of the pump are then modeled as modifications over variables representing the number of ions both inside and outside the cell. In particular, variables Na_O, Na_I, K_O and K_I represent the amount of Sodium and Potassium ions placed outside and inside the cell, respectively. In addition, a certain amount of each kind of ion needed for the correct functioning of the cell is assumed. Such amounts are denoted by Na_{IDEAL} and K_{IDEAL}. Finally, the model includes additional variables capturing other details of the pump: $OPump$ represents the orientation of the pump (either inside or outside the cell), $Alpha$ denotes the current binding of the alpha subunit and $Pump$ represents the current content of the pump. These three variables will be instantiated with constants that can be encoded by integers: for instance, possible values for $Alpha$ are `P`, `free` and `null` (note the special font style given to constants). Finally, integer variables ATP and ADP represent the presence of ATP and ADP inside the cell, respectively.

The model in Figures 1 and 2 reflect the complementary nature of active and passive transport in the SP-pump, represented as $ActiveTrans$ and $PassiveTrans$ processes, which are integrated as the $NaKPump$ process. From this process it is then possible to assume some environment in which the pump is placed. This is the intuition behind process $System$. We now proceed to explain in a greater detail the ideas behind these processes.

Active Transport Phases

Process $ActiveTrans$ integrates sub-processes for the six phases described before; these processes invoke each other. Some processes (i.e., $NaPhase1$, $NaPhase2$ and $KPhase1$) include possible recursive calls to themselves. This intends to represent the possibility that the system remains stuck in certain phases in spite of all the conditions needed to evolve are given. That is, we are trying to model "reversible" phases, a behavior that is represented by non-deterministic choices. As a result, those phases could be executed several times therefore delaying system execution in at least one time unit. Such a delay occurs because the system waits for the presence of some substances at a specific place of the pump. In fact, those substances could be available but not in the required place. This non-deterministic and asynchronous behavior could represent other conditions on component binding, such as an appropriate physical contact among elements that (chemically) react with components of the pump. Similarly, non-deterministic behavior can also represent some kind of malfunction. For instance, it is possible that in phase $NaPhase2$ the phosphate could not bind to the alpha subunit, which would result in a malfunction of the system that could be directly observed from the evolution of the pump in time.

Passive Transport Phases

Process $PassiveTrans$ defines two sub-processes: one for the entrance of Sodium ions and another for the output of Potassium ions. It is worth noticing that in the modeling of these sub-processes we are considering partial behavioral information on the actual time when the ion movement really occurs, which is represented by a bounded asynchronous operator.

$$
\begin{aligned}
NaPhase1 &\stackrel{\text{def}}{=} \textbf{when } (Na_I > Na_{IDEAL} \vee K_I < K_{IDEAL}) \wedge Pump = \texttt{Empty} \wedge OPump = \texttt{In} \textbf{ do}\\
&\quad (\textbf{next } (Na_I := Na_I - 3 \parallel Pump := \texttt{Na} \parallel \textbf{tell}(unchangedK = 1) \parallel NaPhase2) \ +\\
&\qquad \textbf{next } (NaPhase1 \parallel \textbf{tell}(unchangedK = 1) \parallel \textbf{tell}(unchangedNa = 1)))\\
NaPhase2 &\stackrel{\text{def}}{=} \textbf{when } Pump = \texttt{Na} \wedge Alpha = \texttt{free} \wedge ATP > 0 \textbf{ do}\\
&\quad (\textbf{next } (OPump := \texttt{Out} \parallel Alpha := \texttt{P} \parallel ADP := 1 \parallel\\
&\qquad\qquad \textbf{tell}(unchangedK = 1) \parallel \textbf{tell}(unchangedNa = 1) \parallel NaPhase3)\\
&\qquad + \textbf{next } (NaPhase2 \parallel \textbf{tell}(unchangedK = 1) \parallel \textbf{tell}(unchangedNa = 1)))\\
NaPhase3 &\stackrel{\text{def}}{=} \textbf{when } Pump = \texttt{Na} \wedge OPump = \texttt{Out} \textbf{ do}\\
&\quad \textbf{next } (Na_O := Na_O + 3 \parallel Pump := \texttt{Empty} \parallel \textbf{tell}(unchangedK = 1) \parallel KPhase1)\\
KPhase1 &\stackrel{\text{def}}{=} \textbf{when } Pump = \texttt{Empty} \wedge OPump = \texttt{Out} \textbf{ do}\\
&\quad (\textbf{next } (Pump := \texttt{K} \parallel K_O := K_O - 2 \parallel \textbf{tell}(unchangedNa = 1) \parallel KPhase2) \ +\\
&\qquad \textbf{next } (KPhase1 \parallel \textbf{tell}(unchangedK = 1) \parallel \textbf{tell}(unchangedNa = 1)))\\
KPhase2 &\stackrel{\text{def}}{=} \textbf{when } Alpha = \texttt{P} \wedge Pump = \texttt{K} \textbf{ do}\\
&\quad \textbf{next } (OPump := \texttt{In} \parallel ADP := 0 \parallel Alpha := \texttt{free} \parallel\\
&\qquad\quad \textbf{tell}(unchangedK = 1) \parallel \textbf{tell}(unchangedNa = 1) \parallel KPhase3)\\
KPhase3 &\stackrel{\text{def}}{=} \textbf{when } Pump = \texttt{K} \wedge OPump = \texttt{In} \textbf{ do}\\
&\quad \textbf{next } (K_I := K_I + 2 \parallel Pump := \texttt{Empty} \parallel \textbf{tell}(unchangedNa = 1) \parallel NaPhase1)\\
ActiveTrans &\stackrel{\text{def}}{=} NaPhase1
\end{aligned}
$$

Figure 1. An `ntcc` model for the Sodium-Potassium pump (Part 1 of 2)

Additional Processes

The integration of the above processes as the $NakPump$ process is straightforward. There is an additional process (i.e., $Control$) which governs the global behavior of the pump w.r.t. the equilibrium of the ions amounts; in the case an equilibrium on the amount of one of the ions is reached, a general system malfunction (denoted as $death = 1$) is established. As the other processes, the structure of this control process makes it possible the inclusion of additional features. Process $Start$, which receives a group of six parameters (denoted as $\sigma_{1\ldots6}$), is self-explanatory.

A remarkable feature of our model is that it can be parameterized with actual quantitative values extracted from experimentation. In our model ion concentrations depend on *parameters* which make it more accurate; more detailed models involving other biological components (such as, e.g., the electrochemical gradients governing the dynamics of the passive transport and the magnitude of forces related with the physical contact between ions and the pump) would then require the inclusion of more sophisticated numerical parameters. In this sense, considering a constraint system over real numbers would not only allow to include more sophisticated conditions but also would allow to perform analyzes at different levels of detail.

$$
\begin{aligned}
\mathit{PassiveNa} &\stackrel{\text{def}}{=} \textbf{unless}\ \ Na_O = Na_I\ \ \textbf{next}\\
&\qquad (\textbf{next}^5\ (\mathit{PassiveNa}) \parallel\\
&\quad \star_{[0,5]}(\textbf{unless}\ \ \mathit{unchangedNa} = 1\ \ \textbf{next}\ \ (Na_I := Na_I + 3 \parallel Na_O := Na_O - 3) \parallel\\
&\qquad\quad \textbf{when}\ \mathit{unchangedNa} = 1\ \textbf{do}\ \ (Na_I := Na_I + 3 \parallel Na_O := Na_O - 3)))\\
\mathit{PassiveK} &\stackrel{\text{def}}{=} \textbf{unless}\ \ K_O = K_I\ \ \textbf{next}\\
&\qquad (\textbf{next}^5\ (\mathit{PassiveK}) \parallel\\
&\qquad \star_{[0,5]}(\textbf{unless}\ \ \mathit{unchangedK} = 1\ \ \textbf{next}\ \ (K_I := K_I - 2 \parallel K_O := K_O + 2) \parallel\\
&\qquad\quad \textbf{when}\ \mathit{unchangedK} = 1\ \textbf{do}\ \ (K_I := K_I - 2 \parallel K_O := K_O + 2)))\\
\mathit{PassiveTrans} &\stackrel{\text{def}}{=} \mathit{PassiveNa} \parallel \mathit{PassiveK}\\
\mathit{Control} &\stackrel{\text{def}}{=}\ !\ (\textbf{when}\ Na_I = Na_O\ \textbf{do}\ \ \textbf{tell}(\mathit{equilNa} = 1) \parallel\\
&\qquad \textbf{when}\ K_I = K_O\ \textbf{do}\ \ \textbf{tell}(\mathit{equilK} = 1) \parallel\\
&\qquad \textbf{when}\ \mathit{equilNa} = 1 \vee \mathit{equilK} = 1 \vee M\ \textbf{do}\ \ !(\textbf{tell}(\mathit{death} = 1)))\\
\mathit{Start}(\sigma_{1\ldots6}) &\stackrel{\text{def}}{=}\ !(\textbf{tell}(ATP > 0) \parallel \textbf{tell}(Na_{IDEAL} = \sigma_5) \parallel \textbf{tell}(K_{IDEAL} = \sigma_6))\\
&\quad ADP : 0 \parallel \mathit{Alpha} : \texttt{free} \parallel \mathit{OPump} : \texttt{In} \parallel \mathit{Pump} : \texttt{Empty} \parallel\\
&\quad Na_I : \sigma_1 \parallel Na_O : \sigma_2 \parallel K_I : \sigma_3 \parallel K_O : \sigma_4\\
\mathit{NaKPump} &\stackrel{\text{def}}{=} \textbf{local}\ Na_I, Na_O, K_I, K_O, \mathit{Alpha}, \mathit{ADP}, \mathit{Pump}, \mathit{OPump}\ \textbf{in}\\
&\qquad \mathit{Start}(\sigma_{1\ldots6}) \parallel \mathit{ActiveTrans} \parallel \mathit{PassiveTrans} \parallel \mathit{Control}\\
\mathit{System} &\stackrel{\text{def}}{=} \mathit{NaKPump} \parallel \mathit{Environment}
\end{aligned}
$$

Figure 2. An `ntcc` model for the Sodium-Potassium pump (Part 2 of 2)

4.3 *Proving Properties About Biological Models: A logic-based approach*

In this section we give a non-trivial biological example of the reasoning capabilities of `ntcc`. In particular, the example deals with an *inhibition process* over the SP-pump. This inhibition may represent both a drug and a disease: to prevent circulatory problems, certain medicines induce a partial inhibition of the pump to augment the strength of heart's contractions, thus improving blood circulation. On the other hand, certain substances may cause a complete inhibition process over the pump, therefore causing the death of the cell.

The inhibition process example also allows us to take advantage of the flexibility of the presented model. We will assume a (malicious) drug that is present in the environment surrounding the pump. The goal of this drug is to take control of the alpha subunit, thus preventing the phosphate from inducing a conformational change in the pump. In turn, this obstruction will lead to a complete inhibition of the active transport mechanism enforced by the pump. We express this in our model by specifying the *Environment* process as follows:

$$\mathit{Environment} \stackrel{\text{def}}{=} \mathit{Drug} \qquad (1)$$

where $\mathit{Drug} \stackrel{\text{def}}{=} \star_{[m,n]}\ \textbf{when}\ \mathit{Alpha} = \texttt{free}\ \textbf{do}\ \ !\mathit{Alpha} := \texttt{null}$ (with $n > m$). Note that the actual time unit where *Drug* will be active is undetermined, because of the uncertainty induced by the $\star$ operator. It is important to remark that although

$Drug$ is the only component *explicitly described* in the $Environment$ process, other components or systems can be easily included in its definition. In other words, we are focusing on the drug-related part of $Environment$. We will also denote by $Drug'$ the process obtained from the execution of $Drug$ at a time $m \leq j \leq n$ (i.e., $Drug' \overset{\text{def}}{=} !Alpha := \texttt{null})$.

By inhibiting the active transport capabilities of the pump, the cell will reach an equilibrium between the internal and external concentrations of Sodium. Such an equilibrium, that causes the death of the cell, is not reversible and will occur in an undetermined future. These facts suggest us the following assertion to be verified:

$$NaKPump \parallel Drug \vdash \Diamond\Box\ death = 1 \tag{2}$$

where $death = 1$ represents the death of the cell. Intuitively, we want to formally verify that in the presence of the drug described above the cell will die in an undetermined future, with no chance of returning to a previous state.

The complete inhibition of the active transport mechanism can be seen directly on the model. At a certain stage of the process (just after $NaPhase1$), the alpha subunit will be empty, ready for a binding with some substance (`P` in the "healthy" case). The inclusion of $Drug$ in the environment adds a new alternative of execution, as both $NaPhase2$ and $Drug$ have the chance of binding the subunit (with `P` and `null`, respectively). In this (implicit) non-deterministic choice, we assume the success of the drug in binding the alpha subunit. Note that this choice precludes the active transport processes from the execution of the system. Therefore, at that point, we can regard the system as the following processes:

$$Control \parallel PassiveNa \parallel Drug' \parallel RestOfSystem' \tag{3}$$

where $RestOfSystem' \overset{\text{def}}{=} PassiveK \parallel !(\mathbf{tell}(ATP > 0) \parallel \mathbf{tell}(Na_{IDEAL} = \sigma_5) \parallel \mathbf{tell}(K_{IDEAL} = \sigma_6))$. As a result, assertion (2) can be expressed as

$$Control \parallel PassiveNa \parallel Drug' \parallel RestOfSystem \vdash \Diamond\Box\ death = 1. \tag{4}$$

In order to prove (4), we will restrict our attention to the interaction among $Control$, $PassiveNa$ and $Drug'$. Intuitively, due to the absence of the active transport mechanism the passive transport will introduce sodium ions into the cell until reaching an equilibrium (i.e., $Na_I = Na_O$). Once that occurs, $Control$ (that has been awaiting the equilibrium) emits $equilNa = 1$ to the environment. Such a signal is enough to determine the death of the cell.

The proof proceeds as follows. Let us first assume the following abbreviations for processes and guards:

$$G_1 = (G_2 \vee G_3 \vee M) \quad G_2 = (equilNa = 1) \quad G_3 = (equilK = 1)$$
$$G_4 = (Na_I = Na_O) \quad G_5 = (K_I = K_O) \quad A \overset{\text{def}}{=} \mathbf{when}\ G_1\ \mathbf{do}\ !\ \mathbf{tell}(death = 1)$$
$$B \overset{\text{def}}{=} \mathbf{when}\ G_4\ \mathbf{do}\ \ \mathbf{tell}(G_2) \quad C \overset{\text{def}}{=} \mathbf{when}\ G_5\ \mathbf{do}\ \ \mathbf{tell}(G_3)$$

Consequently, and because of the replicated definition of $Control$, we have

$$Control \overset{\text{def}}{=} \; !\, A \parallel !\, B \parallel !\, C.$$

The following proposition represents an intuition derived from the definition of $PassiveNa$ and $Drug'$.

Proposition 4.1 $PassiveNa \parallel Drug' \vdash \Diamond G_4$.

Once $Drug'$ is present in the system and sets the state of $Alpha$ to `null` for every future time unit, process $ActiveTrans$ does not modify anymore neither Na_I or Na_O. As a consequence, process $PassiveNa$ decrements Na_O and increments Na_I until they have the same value (i.e., $Na_I = Na_O$). This will take some time units, depending on the value of Na_I and Na_O when $Drug'$ be active in the system. This behavior can also be verified applying the rules in the operational semantics of `ntcc`.

Finally, using the proof system in Table 1, it is possible to derive a proof for (4). Let us first derive $!(B \parallel C) \vdash \Box(G_4 \dot\Rightarrow G_2)$ (Proposition 4.2):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{B \vdash (G_4 \dot\wedge G_2) \dot\vee \dot\neg G_4}\,\text{LSUM}}{B \vdash G_4 \dot\Rightarrow G_2}\,\text{LCONS} \qquad \dfrac{\dfrac{}{C \vdash (G_5 \dot\wedge G_3) \dot\vee \dot\neg G_5}\,\text{LSUM}}{C \vdash G_5 \dot\Rightarrow G_3}\,\text{LCONS}}{B \parallel C \vdash (G_4 \dot\Rightarrow G_2)\, \dot\wedge (G_5 \dot\Rightarrow G_3)}\,\text{LPAR}}{B \parallel C \vdash G_4 \dot\Rightarrow G_2}\,\text{LCONS}}{!(B \parallel C) \vdash \Box(G_4 \dot\Rightarrow G_2)}\,\text{LREP}$$

With the above result, we can perform the following deductions. Let us first state an auxiliar derivation:

$$D = \dfrac{\dfrac{\dfrac{}{!(B \parallel C) \vdash \Box(G_4 \dot\Rightarrow G_2)}\,\text{Prop. 4.2} \qquad \dfrac{}{PassiveNa \parallel Drug' \vdash \Diamond G_4}\,\text{Prop. 4.1}}{!(B \parallel C) \parallel PassiveNa \parallel Drug' \vdash \Box(G_4 \dot\Rightarrow G_2) \,\dot\wedge \Diamond G_4}\,\text{LPAR}}{!(B \parallel C) \parallel PassiveNa \parallel Drug' \vdash \Diamond G_2}\,\text{LCONS}$$

We then get the following derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash (G_1 \dot\wedge \Box\; death = 1) \dot\vee \dot\neg G_1}\,\text{LSUM}}{A \vdash G_1 \dot\Rightarrow \Box\; death = 1}\,\text{LCONS}}{!A \vdash \Box(G_1 \dot\Rightarrow \Box\; death = 1)}\,\text{LREP}}{!A \vdash \Box((G_2 \dot\vee G_3 \dot\vee M) \dot\Rightarrow \Box\; death = 1)}\,\text{LCONS}}{!A \vdash \Box(G_2 \dot\Rightarrow \Box\; death = 1)}\,\text{LCONS} \qquad D}{!A \parallel !B \parallel !C \parallel PassiveNa \parallel Drug' \vdash \Box(G_2 \dot\Rightarrow \Box\; death = 1) \,\dot\wedge \Diamond G_2}\,\text{LPAR}}{Control \parallel PassiveNa \parallel Drug' \vdash \Diamond\Box\; death = 1}\,\text{LCONS}$$

Finally, using item 3 in Lemma 1, we obtain

$$Control \parallel PassiveNa \parallel Drug' \parallel RestOfSystem \vdash \Diamond\Box\; death = 1$$

hence proving the desired property.

Notice how the partial information constructs helped to better describe the behavior of the SP-pump. They allow for flexible and extensible system specifications. Moreover, since the associated temporal logic naturally captures the spirit of these constructs, the essential properties to be verified can also involve partial information in an explicit way.

5 Concluding Remarks

In this paper we have proposed `ntcc`, a process calculus based on constraints, as a suitable language for modeling and verifying biological systems. We have shown how process constructs in `ntcc` naturally capture two kinds of partial information: quantitative and behavioral. Descriptions of many biological phenomena that are only partially understood could greatly benefit from the use of these kinds of partial information provided by `ntcc`.

Furthermore, `ntcc` provides a *single, unified* framework where it is possible to both *model and reason* about biological systems. This approach was illustrated by modeling an ion transport mechanism and verifying one non-trivial property of such a model. While the use of partial behavioral information statements was crucial to describe and reason about a possible system failure, partial quantitative information statements provided flexibility in the modeling process.

References

[1] M. Antoniotti, C. Piazza, A. Policriti, M. Simeoni, and B. Mishra. Taming the complexity of biochemical models through bisimulation and collapsing: theory and practice. *Theor. Comput. Sci.*, 325(1):45–67, 2004.

[2] A. Arbeláez, J. Gutiérrez, C. Olarte, and C. Rueda. A Generic Framework to Model, Simulate and Verify Genetic Regulatory Networks. In *Proc. of 32nd Latin-American Conference on Informatics (CLEI 2006)*, 2006. Santiago, Chile.

[3] AVISPA Research Group. ntccSim: A simulation tool for timed concurrent processes, 2006. Available at http://avispa.puj.edu.co.

[4] A. Bockmayr and A. Courtois. Using hybrid concurrent constraint programming to model dynamic biological systems. In Peter J. Stuckey, editor, *ICLP*, volume 2401 of *LNCS*, pages 85–99. Springer, 2002.

[5] A. Bockmayr, A. Courtois, D. Eveillard, and M. Vezain. Building and Analysing an Integrative Model of HIV-1 RNA Alternative Splicing. In Danos and Schächter [9], pages 43–57.

[6] L. Calzone, N. Chabrier-Rivier, F. Fages, and S. Soliman. Machine learning biochemical networks from temporal logic properties. *Transactions on Computational Systems Biology*, 2006. CMSB'05 Special Issue (to appear).

[7] L. Cardelli. Brane Calculi. In Danos and Schächter [9], pages 257–278.

[8] V. Danos and C. Laneve. Formal molecular biology. *Theor. Comput. Sci.*, 325(1):69–110, 2004.

[9] V. Danos and V. Schächter, editors. *Computational Methods in Systems Biology, International Conference CMSB 2004, Paris, France, May 26-28, 2004, Revised Selected Papers*, volume 3082 of *LNCS*. Springer, 2005.

[10] D. Eveillard, D. Ropers, H. de Jong, C. Branlant, and A. Bockmayr. A multi-scale constraint programming model of alternative splicing regulation. *Theor. Comput. Sci.*, 325(1):3–24, 2004.

[11] V. Gupta, R. Jagadeesan, V. A. Saraswat, and D. G. Bobrow. Programming in hybrid constraint languages. In P. J. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems*, volume 999 of *LNCS*, pages 226–251. Springer, 1994.

[12] P. Van Hentenryck, V. Saraswat, and Y. Deville. Design, Implementation, and Evaluation of the Constraint Language cc(FD). In *Constraint Programming*, volume 910 of *LNCS*, pages 293–316. Springer, 1994.

[13] B. Mishra, M. Antoniotti, S. Paxia, and N. Ugel. Simpathica: A Computational Systems Biology Tool within the Valis Bioinformatics Environment. In E. Eiles and A. Kriete, editors, *Computational Systems Biology*. Elsevier, 2005.

[14] M. Nielsen, C. Palamidessi, and F. Valencia. Temporal Concurrent Constraint Programming: Denotation, Logic and Applications. *Nordic Journal of Computing*, 9:145–188, 2002.

[15] C. Priami and P. Quaglia. Beta Binders for Biological Interactions. In Danos and Schächter [9], pages 20–33.

[16] A. Regev, E. M. Panina, W. Silverman, L. Cardelli, and E. Shapiro. Bioambients: an abstraction for biological compartments. *Theor. Comput. Sci.*, 325(1):141–167, 2004.

[17] A. Regev and E. Shapiro. Cells as Computation. *Nature*, 419:343, September 2002.

[18] A. Regev and E. Shapiro. *Modelling in Molecular Biology*, chapter The π-calculus as an abstraction for biomolecular systems, pages 219–266. Natural Computing Series. Springer, 2004.

[19] V. Saraswat, M. Rinard, and P. Panangaden. The semantic foundations of concurrent constraint programming. In *POPL '91*, pages 333–352, Jan 1991.

[20] G. Scheiner-Bobis. The sodium pump: Its molecular properties and mechanics of ion transport. *Euro. J. Biochem.*, 269:2424–2433, 2002.

[21] F. Valencia. Decidability of Infinite-State Timed CCP Process and First-Order LTL. *Theor. Comput. Sci.*, 330(3):577–607, 2005.

A ntcc Operational Semantics

$$\textsf{TELL}\ \frac{}{\langle \mathbf{tell}(c), d\rangle \longrightarrow \langle \mathbf{skip}, d \wedge c\rangle} \qquad \textsf{SUM}\ \frac{d \models c_j \ \ j \in I}{\left\langle \sum_{i\in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i, d\right\rangle \longrightarrow \langle P_j, d\rangle}$$

$$\textsf{PAR}\ \frac{\langle P, c\rangle \longrightarrow \langle P', d\rangle}{\langle P \parallel Q, c\rangle \longrightarrow \langle P' \parallel Q, d\rangle} \qquad \textsf{LOC}\ \frac{\langle P, c \wedge \exists_x d\rangle \longrightarrow \langle P', c'\rangle}{\langle (\mathbf{local}\, x, c)\, P, d\rangle \longrightarrow \langle (\mathbf{local}\, x, c')\, P', d \wedge \exists_x c'\rangle}$$

$$\textsf{UNL}\ \frac{}{\langle \mathbf{unless}\ c\ \mathbf{next}\, P, d\rangle \longrightarrow \langle \mathbf{skip}, d\rangle}\ \text{if } d \models c$$

$$\textsf{REP}\ \frac{}{\langle !\, P, d\rangle \longrightarrow \langle P \parallel \mathbf{next}\, !\, P, d\rangle} \qquad \textsf{STAR}\ \frac{}{\langle \star\, P, d\rangle \longrightarrow \langle \mathbf{next}^{\,n} P, d\rangle}\ \text{if } n \geq 0$$

$$\textsf{STR}\ \frac{\gamma_1 \longrightarrow \gamma_2}{\gamma_1' \longrightarrow \gamma_2'}\ \text{if } \gamma_1 \equiv \gamma_1' \text{ and } \gamma_2 \equiv \gamma_2'$$

$$\textsf{OBS}\ \frac{\langle P, c\rangle \longrightarrow^* \langle Q, d\rangle \not\longrightarrow}{P \xRightarrow{(c,d)} R}\ \text{if } R \equiv F(Q)$$

Table A.1
Rules for internal reduction $\longrightarrow$ (upper part) and observable reduction $\Longrightarrow$ (lower part). $\gamma \not\longrightarrow$ in OBS holds iff for no γ', $\gamma \longrightarrow \gamma'$.

Note that $\equiv$ (structural congruence) is the smallest congruence satisfying: (1) $P \parallel \mathbf{skip} \equiv P$, (2) $P \parallel Q \equiv Q \parallel P$, and (3) $P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R$.

In rule OBS, the process R to be executed in the next time interval is equivalent to $F(Q)$, the "future" of Q.

Definition A.1 [Future Function] Let $F : \mathit{Proc} \rightharpoonup \mathit{Proc}$ be defined by

$$F(Q) = \begin{cases} \mathbf{skip} & \text{if } Q = \sum_{i\in I} \mathbf{when}\ c_i\ \mathbf{do}\ Q_i \\ F(Q_1) \parallel F(Q_2) & \text{if } Q = Q_1 \parallel Q_2 \\ (\mathbf{local}\, x)\, F(R) & \text{if } Q = (\mathbf{local}\, x, c)\, R \\ R & \text{if } Q = \mathbf{next}\, R \text{ or } Q = \mathbf{unless}\ c\ \mathbf{next}\, R \end{cases}$$

Intuitively, $F(Q)$ is obtained by removing from Q summations that did not trigger activity and any local information which has been stored in Q, and by "unfolding" the sub-terms within "next" and "unless" expressions. Notice that F does not need to be total since whenever we need to apply F to a Q (OBS in Table A.1), every $\mathbf{tell}(c)$, $\star R$ and $!\, R$ in Q will occur within a "next" or "unless" expression.