A Generic Framework to Model, Simulate and Verify Genetic Regulatory Networks

Alejandro Arbeláez, Julian Gutiérrez, Carlos Olarte and Camilo Rueda
Pontificia Universidad Javeriana
Calle 18 No. 118 - 250 , Cali-Colombia
aarbelaez@puj.edu.co, {jg,caolarte,crueda}@cic.puj.edu.co

Abstract

Process calculi are formalisms to model concurrent systems. Their mathematical basis and compositional style make possible to decompose a system into simple and well defined processes. Interaction among them is formally defined by the semantic of the calculi. These characteristics allow to study systems coming from different areas such as arts, engineering and sciences. In this paper we propose a generic framework to model, simulate and verify genetic regulatory networks based on a non-deterministic timed concurrent constraint calculus. This framework provides a set of process definitions to model generic/parametric components in a biological context, a simulator to observe the system evolution in time and some insights to perform formal proofs to verify and make inferences over the systems. An instantiation of the framework is presented by modeling the lactose operon.

Keywords: Process calculi, Concurrent Constraint Programming, ntcc, Genetic Regulatory Networks, Lac Operon

Resumen

Los cálculos de procesos son formalismos para modelar sistemas concurrentes. Sus fundamentos matemáticos y estilo de diseño composicional hacen posible descomponer un sistema en procesos simples y bien definidos. La interacción entre los procesos es formalmente definida por la semántica de los cálculos. Estas características permiten estudiar sistemas provenientes de diferentes áreas tales como las artes, la ingeniería y las ciencias. En este artículo se propone un marco genérico para modelar, simular y verificar redes de regulación genética con el uso de un cálculo de procesos temporal y no determinístico basado en restricciones. Este marco provee un conjunto de definiciones de procesos para modelar componentes genéricos/paramétricos en un contexto biológico, un simulador para observar la evolución del sistema en el tiempo y algunas directrices para desarrollar pruebas formales para verificar y hacer inferencia sobre el sistema. Un caso de estudio es presentado con el modelado del operón de la lactosa, una red de regulación genética compleja.

Palabras claves: Cálculos de Procesos, Programación Concurrente por Restricciones, ntcc, Redes de Regulación Genética, Operón de la Lactosa

1 Introduction

The study of concurrent systems is often carried out with the aid of process calculi. These are very expressive formalisms centered on the notion of interaction. Systems are understood as interacting complex processes composed of smaller ones following a compositional approach. Such an approach is encouraged by the (usually) few mathematical constructs provided by the calculus that defines how processes interact among them and with their environment. The mathematical foundations underlying process calculi allow to both model and verify properties of a system, thus providing a concrete design methodology for complex systems. These appealing properties of process calculi have been a strong motivation for its use in a wide spectrum of fields including, distributed systems [10], systems biology [15], visual/object-oriented languages [16] and reactive systems [18] among others. The interest of the scientific community on process calculi is also reflected by the extensions proposed in order to cope with a number of widely occurring concepts such as time ([10, 18]), mobility [8], probabilistic/stochastic behavior [13, 11], and partial information [17].

Process calculi have been previously used to model biological functions (see [5, 12]). Most of this work has been conducted using (extensions of) the π calculus ([13, 2]) and the Ambient calculus ([14]). By the other side, calculi devised for specific biological systems have also been proposed. For instance, calculi to model membranes ([1]), protein interaction ([3]) and reversibility in bio-molecular processes ([6]).

We are interested in the study of biological/molecular systems using process calculi following the concurrent constraint programming model (ccp) [17]. ccp is based on the concept of constraint as an entity carrying partial information, i.e., conditions for the values that variables can take. Constraints are accumulated monotonically over a so-called store, an entity that contains all information produced by the system during its execution. In this way, the problem of finding higher-order biological function of systems can be taken up by relying in the fundamental mathematical approach those process calculi provide. This approach is justified by the following facts: (i) A clear separation among processes can be achieved by considering concurrent agents as basic components of programs making possible a straightforward model refinement in context-dependent models. (ii) Because of the declarative nature of ccp, only the constraints of the problems have to be stated, whereas system control (i.e., evolution) is formally defined by the semantic of the calculus. (iii) Constraints can be seen as a representation of incomplete knowledge. This is an important consideration in biological domains where the exact function of several systems and mechanisms is still a matter of research. Finally, (iv) the construction of simulation tools and model verifiers can be formally done.

Models presented in this paper are built using a non-deterministic timed extension of ccp called `ntcc` [10] and a stochastic extension of it [11]. These models allow to express non-determinism, asynchronous and stochastic behavior. We aim to establish how such constraint-based process calculi can help to design a formal language suitable to model molecular processes. The objective is then using this language to develop highly accurate models, discover from these models behavioral properties of biological systems and develop semi-automated tools to verify and simulate large (complex) systems in molecular biology. We believe that languages and tools based on the ccp paradigm can thus constitute a valuable methodology to design and test coherent bio-molecular models.

The main contribution of this paper is to provide a generic framework to: (i) Model genetic regulatory networks by using a set of process definitions to model biological components. (ii) Observe the evolution of the system during the time by means of a simulation tool executing `ntcc` programs. And (iii) make quantitative inferences when a complete mathematical model of the system is not available by means of formal proofs in the linear temporal logic associated with `ntcc`. In addition, we provide a complete model of the lactose operon (i.e., *lac operon*), a non-trivial biological system not previously modeled using process calculi. The rest of the paper is structured as follow: Section 2 gives some preliminars about `ntcc` calculus. Additionally, a structural and functional description of the lac operon is given. Section 3 shows the generic and parametric process definitions provided by the framework and how they can be used to model the lac operon. Some graphics showing the quantitative measures taken from the simulator of the calculus are presented in section 3.7. Section 4 is devoted to present how formal proofs can be performed in the framework and how they can be used to infer future behavior in the system. Finally, section 5 concludes the paper and gives some research direction.

2 Computational and Biological Foundations

Here we present some of the theoretical background of our work. First, the main features of the `ntcc` process calculus are discussed. Later, an intuitive description of the Lac Operon is given. This system will be used as a case study in forthcoming sections.

2.1 ntcc: A timed, process calculus

`ntcc` is a temporal concurrent constraint calculus suitable to model non-deterministic and asynchronous behavior. As such, it is particularly appropriate to model reactive and concurrent systems. One of the main features of this calculus is that it is equipped with a proof system for linear-temporal properties of `ntcc` processes. In this section we briefly describe the syntax and proof system of the `ntcc` calculus, referring the reader to [10, 21] for further details.

First we recall the notion of *constraint system*. Basically, a constraint system provides a signature from which syntactically denotable objects in the language called *constraints* can be constructed, and an entailment relation ($\models$) specifying interdependencies among such constraints. The underlying language $\mathcal{L}$ of the constraint system contains the symbols $\neg, \wedge, \Rightarrow, \exists$, `true` and `false` which denote logical negation, conjunction, implication, existential quantification, and the always true and always false predicates, respectively. *Constraints*, denoted by $c, d, \ldots$ are first-order formulae over $\mathcal{L}$. We say that c *entails* d in Δ, written $c \models_\Delta d$ (or just $c \models d$ when no confusion arises), if $c \Rightarrow d$ is true in all models of Δ. For operational reasons we shall require $\models$ to be decidable.

In `ntcc` time is divided into discrete *intervals* (or *time units*), each one of them having its own *constraint store* (or simply *store*). Intuitively, a store accumulates all the information available in the system at a given time. The basic actions for communication with the store are *tell* and *ask* operations. While the former adds new pieces of information to the store, the latter enquires the store to check if some information can be inferred from its current content. Moreover, synchronisation among processes is only based on these two actions. In this way, each time unit can be understood as a reactive entity, where a process P_i receives an input from the environment (i.e., a constraint). The process P_i is then executed considering this input, responding with some output (that is, new constraints) once no further processing over the store is possible. Computation in the next time unit is then based on a residual process resulting from P_i and on new inputs provided by the environment.

Process Syntax

`ntcc` 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 the following syntax.

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

The only move or action of process $\mathbf{tell}(c)$ is to add the constraint c to the current store, thus making c available to other processes in the current time interval. The guarded-choice $\sum_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i$, where I is a finite set of indexes, represents a process that, in the current time interval, non-deterministically chooses 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 shall use "+" for binary summations.

Process $P \parallel Q$ represents the parallel composition of P and Q. In one time unit (or interval) P and Q operate concurrently, "communicating" via the common store. We use $\prod_{i \in I} P_i$, where I is a finite set, to denote the parallel composition of all P_i . Process $\mathbf{local}\ x\ \mathbf{in}\ P$ behaves 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. We abbreviate $\mathbf{local}\ x_1\ \mathbf{in}\ (\ \mathbf{local}\ x_2\ \mathbf{in}\ (\ \ldots\ (\ \mathbf{local}\ x_n\ \mathbf{in}\ P\ )\ \ldots\ )\ )$ as $\mathbf{local}\ x_1, x_2, ..., x_n\ \mathbf{in}\ P$.

The process $\mathbf{next}\ (P)$ represents the activation of P in the next time interval. Hence, a move of $\mathbf{next}\ (P)$ is a unit-delay of P. The process $\mathbf{unless}\ c\ \mathbf{next}\ (P)$ is similar, but P will be activated only if c cannot be inferred from the current store. The "unless" processes 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. We shall use $\mathbf{next}^n\ (P)$ as an abbreviation for $\mathbf{next}\ (\mathbf{next}\ (\ldots \mathbf{next}\ (P)) \ldots))$, where $\mathbf{next}$ is repeated n times.

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

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

The operator ! is a delayed version of the replication operator for the π-calculus ([8]): $!P$ 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. The replication operator is a way of defining infinite behavior through the time intervals.

The operator *star* (i.e., $\star$) allows us to express asynchronous behavior. The process $\star P$ represents an arbitrary long but finite delay for the activation of P. For example, $\star\mathbf{tell}(c)$ can be viewed as a message c that is eventually delivered but there is no upper bound on the delivery time. We shall use $\star_{\mathbf{n}}\mathbf{P}$ as an abbreviation of $\mathbf{next}^n\ (\star P)$ to represent a delayed version of the operator *star*.

Using `ntcc` it is also possible to encode *process definitions* as procedures and *recursion*. We shall use a definition of the form $A(x) \stackrel{\text{def}}{=} P_x$ where P_x is a process using a variable x. A "call" of the form $A(c)$ would then launch a process P_x once the variable x is substituted by c. We can rely on the usual intuitions concerning procedure calls in a programming language. We shall use *recursive process 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"** . As in [21] we consider call-by-value for variables in recursive process calls. Moreover, the encodings generalize easily to the case of definitions with an arbitrary number of parameters. These kinds of definition do not add functionality to `ntcc` since they can be defined in terms of the standard `ntcc` constructs.

Linear-temporal Logic in ntcc

The linear-temporal logic associated with `ntcc` is defined as follows. Formulae $A, B, \ldots \in \mathcal{A}$ are defined by the grammar:

$$A, B, \ldots := c \mid A \dot{\Rightarrow} A \mid\ \dot{\neg} A \mid\ \dot{\exists}_x A \mid\ \circ A \mid \Box A \mid \Diamond A.$$

Here c denotes an arbitrary constraint which acts as an atomic proposition. Symbols $\dot{\Rightarrow}$, $\dot{\neg}$ and $\dot{\exists}_x$ represent linear-temporal logic implication, negation and existential quantification. These symbols are not to be confused with the logic symbols $\Rightarrow$, $\neg$ and $\exists_x$ of the constraint system. Symbols $\circ$, $\Box$ and $\Diamond$ denote the linear-temporal operators *next*, *always* and *eventually*. We use $A \dot{\vee} B$ as an abbreviation of $\dot{\neg} A \dot{\Rightarrow} B$ and $A \dot{\wedge} B$ as an abbreviation of $\dot{\neg}(\dot{\neg} A \dot{\vee} \dot{\neg} B)$. The standard interpretation structures of linear temporal logic are infinite sequences of states. In `ntcc`, states are represented with constraints, thus we consider as interpretations the elements of $\mathcal{C}^\omega$. When $\alpha \in \mathcal{C}^\omega$ is a model of A, we write $\alpha \models A$.

We shall say that P satisfies A if every infinite sequence that P can possibly output satisfies the property expressed by A. A relatively complete proof system for assertions $P \vdash A$, whose intended meaning is that P satisfies A, is given in Table 1. We shall write $P \vdash A$ if there is a derivation of $P \vdash A$ in this system. Finally, the following lemma will be useful in derivations:

Lemma 1 (Nielsen et al. [10]) *For every process P,*

1. $P \vdash \dot{\mathtt{true}}$, 2. $P \nvdash \mathtt{false}$, 3. $\dfrac{P \vdash A}{P \parallel Q \vdash A}$ *and* 4. $\dfrac{P \vdash A \qquad P \vdash B}{P \vdash A \dot{\wedge} B}$.

2.2 The Lac Operon: structure and behavior

The lac operon [7] is one of the most important genetic regulatory networks [7] present in living cells. This regulatory system deals with the sources of energy needed to accomplish the functions of the cell. The

genetic regulatory network related with the lac operon has been extensively studied by biologist due to its biological importance. Next, we will present a general description of the main features in the structure and behavior of the lac operon.

An operon is a genetic cluster comprising a control region and some structural genes. The control region determines the operon status. In particular, the lac operon has three genes in the structural region: LacZ, LacY and LacA (see Figure 1). Gene *LacZ* codifies for β-galactosidase protein which hydrolyses (a term used for some bio-molecular divisions) lactose into glucose and galactose. Gene *LacY* codifies for permease protein. This molecule allows to lactose outside the cell to move across the cell membrane to increase the concentration levels of lactose inside the cell. Finally, gene *LacA* codifies for β-galatoside transacetylase protein. The function of this protein is still undetermined but biologists believe that it has no influence on the lac operon regulatory system. Another important gene related to the lac operon regulatory system is *LacI*. This gene codifies a protein that precludes activation of the operon (i.e., it is a so-called repressor protein).

In this genetic regulatory network we can identify two important regulatory processes: repression and induction. The former favors turning the genes off while the latter favors the opposite behavior. The regulating mechanism enforced by the lac operon system is as follows: there is repression when a cell is in an environment plenty in glucose. In this case, the repressor protein produced by LacI can bind to the control region thus preventing RNA polymerase (an enzyme) to transcribe the operon. But when there is lack of glucose in the environment an induction process is triggered. In induction a protein called CAP-cAMP is produced in the cell, helping RNA polymerase to transcribe the operon. In this situation, β-galactosidase, permease and β-galatoside transacetylase proteins increment their concentration inside the cell. In addition, the concentration of internal lactose induces the formation of a molecule called allolactose. This sugar cooperates in the induction process blocking the repressor proteins.

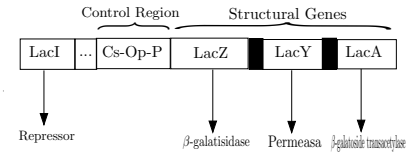


Figure 1: Lac Operon

3 Genetic Regulatory Networks in ntcc

In this section we will present a set of `ntcc` processes to model the behavior of genetic regulatory networks. The lac operon regulatory system is used as a case study. First we explain how we model continuous systems with the discrete-temporal features of `ntcc`. Then, in sections 3.2 and 3.3, formal `ntcc` definitions of molecular events and of regulation and status values in biological entities are given. The `ntcc` processes shown in these sections are the basis for the model of the lac operon regulatory system given in subsequent sections. Finally, in section 3.7 the whole model and some results of its simulation are presented.

3.1 Continuous systems in ntcc

Continuity is required to model this kind of systems because their regulation is determined by the concentration levels of different biological entities along time. We consider two different kinds of continuity: persistence in the values of the variables and continuous time.

To model the former we define a process that explicitly transfers the current value of a variable of the system from one `ntcc` time unit to the next. We shall use m_i and m_i' for the value of a variable in the current and the last time unit, respectively:

$$State_i(v_i) \stackrel{\text{def}}{=} \mathbf{tell}(m_i' = v_i) \parallel \mathbf{next}\ (State_i(m_i))$$

This process is used to define the state of the system in every `ntcc` time unit:

$$State(\rho_1, ..., \rho_n) \stackrel{\text{def}}{=} \prod_{i \in I} (\ \mathbf{tell}(m_i = \rho_i) \parallel \mathbf{next}\ (State_i(\rho_i))\ )$$

where I is the set of indexes of variables in the biological system and ρ_i the initial value of m_i. The above process is also used to configure the system for real system simulations with parameters coming from actual biological measurements.

The temporal kind of continuity is achieved by considering many `ntcc` time units as "samples" of one system unit:

$$Time(t) \stackrel{\text{def}}{=} \textbf{tell}(Ts = t) \parallel \textbf{next}\ (Time(t + Dt))$$

where Ts is the "continuous" time value of the system in the current `ntcc` time unit and Dt a constant value representing the *resolution* of the system. Lower values of Dt give better approximations of real continuous systems. Obviously, the value of Dt has strong practical consequences in system simulations.

The following process represents the continuous behavior of whole system:

$$Dynamic \stackrel{\text{def}}{=} State(\rho_1, ..., \rho_n) \parallel Time(0.0)$$

A very important feature of *Dynamic* is its generality. This process is not restricted to any particular system, not even biological ones, so it can be used to model the dynamic behavior of many continuous systems.

3.2 Modeling molecular events

In molecular systems several events have to be considered, such as pointing out when a group of molecules interacts with others, performs a specific task or produces a biological control signal. We shall use several discrete variables to indicate either presence or absence of some molecular actions or events in models. The variables representing the events or actions described in this section will be called *signaling variables* in the rest of the paper. A *generic* `ntcc` process to model this kind of molecular behavior can be defined as follows:

$$Signal \stackrel{\text{def}}{=} !\prod_{e \in E, svar \in S} (\ \textbf{when}\ \ e\ \ \textbf{do}\ \ \ \textbf{next}\ (\textbf{tell}(svar = 1)) \parallel \textbf{unless}\ \ e\ \ \textbf{next}\ \ \ \textbf{tell}(svar = 0)\ )$$

where E is the set of constraints expressing molecular events and S the set of signaling variables in the system. This specification is not the same as an *if, then, else* kind of thing. Notice that unlike an *if-then-else* structure, process *Signal* can reason over the lack of information. So, it is always possible to determine *svar* despite constraint e holds or not in the store.

More complex signaling processes and variables can be constructed with the process presented above, e.g., a molecular event with delay conditions using temporal `ntcc` operators. In some cases, to achieve more accurate descriptions of these kinds of molecular behavior, stochastic processes are needed. A stochastic extension of `ntcc` recently proposed in [11] is effectively applied in our case study to model a particular binding process of molecules.

3.3 Modeling regulation and status value in biological entities

Most of the processes used to represent dynamic behavior of a biological entity have a similar structure. They can be modeled as a regulated process controlled by a signaling variable. We define a *parametric* process $Regulate_i$ to model the behavior of biological entity i under the control of a signaling variable. This parametric process can be constructed as follows:

$$Regulate_i(svar, P_i, N_i) \stackrel{\text{def}}{=} \textbf{when}\ \ svar = 1\ \ \textbf{do}\ \ \ P_i\ +\ \textbf{when}\ \ svar = 0\ \ \textbf{do}\ \ \ N_i$$

In the above, process P_i is executed when the biological event marked by signaling variable *svar* occurs. Otherwise process N_i is executed. Notice that operator "+" chooses between the two kinds of regulation for the biological entity i. So, only one type of regulation is perfomed over i since the chosen alternative precludes the other one.

To model *status values* (e.g., level of gene expression, location, etc.), we use template $Status_i$ to define a wide variety of biological situations in which we want to determine particular conditions in/of a biological entity:

$$Status_i \stackrel{\text{def}}{=} \; !\ (\ (\ \textstyle\sum_{c\in C} \textbf{when}\ \ condition_c\ \ \textbf{do}\ \ \textbf{next}\ (\textbf{tell}(m_i = fc_i(m_i'))))\ )\ \parallel \\ \textbf{unless}\ \ knownConditions\ \ \textbf{next}\ \ \ \textbf{tell}(m_i = m_i')\ )$$

where C is the set of indexes of conditions for changes in the status of a biological entity i. The new value is defined by a control function fc_i. When no conditions for change holds, the state of the system remains the same in the next time unit.

3.4 Control Region and Structural Genes

In this section the control region and structural genes of a regulatory network are modeled. We use the $Status_i$ process as a *template* to model the sites or places in which a control event may happen. We also propose a parametric process to model the behavior of the most important biological entity present in a genetic regulatory network: a single gene.

In the particular case of the lac operon three places have relevance in the control process: CAPsite, Operator and Promoter regions. We use discrete variables m_1,m_2 and m_3 to represent the operon status: $CAPsite$ process to indicate induction, *Operator* process to indicate repression and *Promoter* process to indicate the level of transcription. We also use some signaling variables to determine when these biological processes occur. Processes $CAPsite$, *Operator* and $Promoter$ are formally integrated in $ControlRegion$ by the parallel composition operator:

$$ControlRegion \stackrel{\text{def}}{=} CAPsite_1 \parallel Operator_2 \parallel Promoter_3$$

Process Gen_x below is a *parametric* `ntcc` specification to model the structure and behavior of a single gene. This specification is parameterized by constants representing the degradation and production rates of mRNAs and proteins produced in the transcription and translation of a gene. We consider three important entities: level (i.e., status) of transcription and concentrations of mRNAs and proteins produced by the gene. Process Gen_x is defined using parametric/generic processes $Regulate_i$ and $Status_i$:

$$\begin{array}{lcl}
GenStatus_i & \stackrel{\text{def}}{=} & !\ (\ (\ \textbf{when}\ \ tbegin = 1 \wedge tend = 0\ \ \textbf{do}\ \ \textbf{next}\ (\textbf{tell}(m_i = m_i' + 1)) + \\
 & & \quad \textbf{when}\ \ tbegin = 0 \wedge tend = 1\ \ \textbf{do}\ \ \textbf{next}\ (\textbf{tell}(m_i = m_i' - 1))\ )\ \parallel \\
 & & \quad \textbf{unless}\ \ tbegin \neq tend\ \ \textbf{next}\ \ \textbf{tell}(m_i = m_i')\ ) \\
MRNA_j(p_j, d_j) & \stackrel{\text{def}}{=} & Regulate_j(tbegin, \textbf{next}\ (\textbf{tell}(m_j = m_j' + p_j - Dt \times (d_j \times m_j'))), \\
 & & \quad \textbf{next}\ (\textbf{tell}(m_j = m_j' - Dt \times (d_j \times m_j')))) \\
PROTEIN_k(p_k, d_k) & \stackrel{\text{def}}{=} & Regulate_k(mrnah, \textbf{next}\ (\textbf{tell}(m_k = m_k' + Dt \times (p_k \times m_j' - d_k \times m_k'))), \\
 & & \quad \textbf{next}\ (\textbf{tell}(m_k = m_k' - Dt \times (d_k \times m_k')))) \\
Gen_x(p_j, d_j, p_k, d_k) & \stackrel{\text{def}}{=} & GenStatus_i \parallel !\ MRNA_j(p_j, d_j) \parallel !\ PROTEIN_k(p_k, d_k)
\end{array}$$

where m_i, m_j and m_k are variables representing the status of the gene (i.e., level of expression), mRNA concentration and protein concentration, respectively. Moreover, d_j and d_k represent the rate of natural molecular degradation of mRNAs and proteins, respectively. The production rate of these biological entities is determined by the constants p_j and p_k and two signaling variables *tbegin* and *tend*. These denote starting and ending time of RNA polymerase gene transcription. Signaling variable *mrnah* is used to indicate when the mRNA concentration is "high enough" to begin the translation of the protein.

In the particular case of the lac operon two processes are needed to model when RNA polymerase is placed between GenZ and GenY, and when it is placed between GenY and GenA (see Figure 1). This biological situation is modeled as a $Status_i$ process:

$$\begin{array}{lcl}
DelayGG_i & \stackrel{\text{def}}{=} & !\ (\ (\ \textbf{when}\ \ tend_1 = 1 \wedge tbegin_2 = 0\ \ \textbf{do}\ \ \textbf{next}\ (\textbf{tell}(m_i = m_i' + 1)) + \\
 & & \quad \textbf{when}\ \ tend_1 = 0 \wedge tbegin_2 = 1\ \ \textbf{do}\ \ \textbf{next}\ (\textbf{tell}(m_i = m_i' - 1))\ )\ \parallel \\
 & & \quad \textbf{unless}\ \ tend_1 \neq tbegin_2\ \ \textbf{next}\ \ \textbf{tell}(m_i = m_i')\ )
\end{array}$$

where $tend_1$ indicates the time when RNA polymerase finishes the transcription of the first gene and $tbegin_2$ the time when it begins the transcription of the second gene. Thus m_i is the number of molecules of RNA polymerase moving between the two consecutive genes.

To present a complete model of the structural genes in the lac operon, we define $GenZ$, $GenY$, $GenA$, $DelayZY$ and $DelayYA$ in a similar way as the *parametric and generic* specifications proposed for Gen_x and $DelayGG_i$.

$$StructuralGenes \stackrel{\text{def}}{=} GenZ(\kappa_1, ..., \kappa_4) \parallel DelayZY \parallel GenY(\sigma_1, ..., \kappa_4) \parallel DelayYA \parallel GenA(\rho_1, \ldots, \rho_4)$$

The above processes could be used to model the structure and behavior of different genes by changing biological parameters and signaling variables in the model. In subsequent sections we present formal models of particular biological processes in the lac operon (i.e., induction, repression and hydrolysis).

3.5 Induction and Repression

Induction and repression are biological conditions inside the cell determining whether the lac operon turns on or off. In induction, glucose concentration is low (i.e., signaling variable $glucl = 1$). This allows to increase the concentration of a protein called cAMP, thus increasing also that of CAP-cAMP protein. This protein is the biological entity that enhances transcription in the lac operon. Levels of CAP-cAMP protein are indirectly modeled from the explicit concentrations of CAP and cAMP. The concentrations of AMP and ADP are also modeled to calculate the value of cAMP inside the cell. To model induction the `ntcc` processes $CAMP$, AMP, ADP and CAP are defined. Biological details about the parameters in these processes are omitted due to space restrictions (see [9] for a more complete account).

$$\begin{array}{rcl}
CAMP & \stackrel{\text{def}}{=} & Regulate_5(glucl, \textbf{next } (\textbf{tell}(m_5 = m'_5 + Dt \times (0.1m'_{11} - 0.1001m'_5))), \\
 & & \qquad \textbf{next } (\textbf{tell}(m_5 = m'_5 - Dt \times 0.1001m'_5))) \\
AMP & \stackrel{\text{def}}{=} & Regulate_{11}(glucl, \textbf{next } (\textbf{tell}(m_{11} = m'_{11} + Dt \times (0.1m'_5 + 0.1m'_{12} - 0.2001m'_{11}))), \\
 & & \qquad \textbf{next } (\textbf{tell}(m_{11} = m'_{11} + Dt \times (0.1m'_5 + 0.1m'_{12} - 0.1001m'_{11})))) \\
ADP & \stackrel{\text{def}}{=} & ! \textbf{ next } (\textbf{tell}(m_{12} = m'_{12} + Dt \times (0.1m'_{11} - 0.2m'_{12}))) \\
CAP & \stackrel{\text{def}}{=} & ! \textbf{ next } (\textbf{tell}(m_4 = m'_4 + Dt \times (1.0 - 0.1m'_4))) \\
Induction & \stackrel{\text{def}}{=} & ! CAMP \parallel ! AMP \parallel ADP \parallel CAP
\end{array}$$

Process *Repression* below models the repressor gene, the repressor protein and the way in which repressor protein binds to the lac operon. The repressor gene $GenI$ is defined with the same `ntcc` specification used for the other genes in the lac operon. The behavior of the repressor protein modeled in *Repressor* and its binding to the DNA of the lac operon is controlled by *allolach*, a signaling variable indicating when allolactose concentration inside the cell reaches a threshold. When this happens, repressor and allolactose react forming a biological complex that prevents the repressor binding to the lac operon. Signaling variable *allolach* is defined with a stochastic process ρP (see [11]) to model probabilistic binding to the allolactose once the threshold is reached.

The binding process *Binding* includes three processes (i.e., $OperatorBinding$, $DNABinding$ and $NotBinding$) to model the fact that the repressor could interact directly with the operator region or, with less probability, bind to the structural genes. It may also happens that the repressor does not bind to the lac operon. We formally express this kind of behavior in process *Repression*:

$$\begin{array}{rcl}
Repressor & \stackrel{\text{def}}{=} & Regulate_{16}(allolach, \textbf{next } (\textbf{tell}(m_{16} = m'_{16} + Dt \times (0.2m'_{15} - m'_8 \times m'_{16}))), \\
 & & \qquad \textbf{next } (\textbf{tell}(m_{16} = m'_{16} + Dt \times (0.2m'_{15} - m'_{16}))) \\
DNABinding & \stackrel{\text{def}}{=} & Regulate_{17}(allolach, \textbf{next } (\textbf{tell}(m_{17} = m'_{17} - Dt \times 0.1m'_{17})), \\
 & & \qquad \textbf{next } (\textbf{tell}(m_{17} = m'_{17} + Dt \times (0.0399m'_{16} - 0.1m'_{17}))) \\
NotBinding & \stackrel{\text{def}}{=} & Regulate_{18}(allolach, \textbf{next } (\textbf{tell}(m_{18} = m'_{18} - Dt \times 0.1m'_{18})), \\
 & & \qquad \textbf{next } (\textbf{tell}(m_{18} = m'_{18} + Dt \times (0.001m'_{16} - 0.1m'_{18}))) \\
OperatorBinding & \stackrel{\text{def}}{=} & Regulate_7(allolach, \textbf{next } (\textbf{tell}(m_7 = m'_7 - Dt \times 0.1m'_7)), \\
 & & \qquad \textbf{next } (\textbf{tell}(m_7 = m'_7 + Dt \times (0.96m'_{16} - 0.1m'_7))) \\
Binding & \stackrel{\text{def}}{=} & DNABinding \parallel NotBinding \parallel OperatorBinding \\
Repression & \stackrel{\text{def}}{=} & GenI(\kappa_1, ..., \kappa_4) \parallel ! Repressor \parallel ! Binding
\end{array}$$

3.6 Lactose Hydrolysis

In the hydrolysis of lactose into glucose and galactose we observe the real purpose of this genetic regulatory network. In this section we model in `ntcc` three biological processes present in the lac operon: the entrance of lactose into the cell, the division of internal lactose into glucose and galactose and the production of allolactose enhanced by lactose concentration. To determine the behavior of these processes we use four signaling variables: *permh*, *bgalh*, *allolach* and *lacinh*. We also use two functions, *vp* and *vg*, to calculate the degree of regulation produced by permease and β-galactosidase, respectively.

Signaling variables *permh* and *bgalh* indicate when the concentration of permease and β-galactosidase is high enough to enable the biological processes they regulate. When $permh = 1$, the concentration of lactose inside the cell increases and reciprocally the lactose outside the cell decreases. When $bgalh = 1$, lactose is converted into glucose and galactose. Finally, signaling variables *allolach* and *lacinh* are used to determine the behavior of allolactose. While *allolach* has the same meaning as in *Repression*, signaling variable *lacinh* indicates the time when lactose inside the cell reaches a threshold thus improving the production of allolactose. This system is modelled as follows:

$$
\begin{array}{rcl}
LacOut & \stackrel{\text{def}}{=} & Regulate_{29}(permh, \textbf{next}\ (\textbf{tell}(m_{29} = m'_{29} - Dt \times (vp + 0.0001m'_{29}))), \\
 & & \qquad \textbf{next}\ (\textbf{tell}(m_{29} = m'_{29} - Dt \times (0.0001m'_{29})))) \\
LacIn & \stackrel{\text{def}}{=} & Regulate_{19}(bgalh, \\
 & & \quad Regulate_{19}(permh, \textbf{next}\ (\textbf{tell}(m_{19} = m'_{19} + Dt \times (vp - vg - 0.0001m'_{19}))), \\
 & & \qquad \textbf{next}\ (\textbf{tell}(m_{19} = m'_{19} - Dt \times (vg + 0.0001m'_{19})))), \\
 & & \quad Regulate_{19}(permh, \textbf{next}\ (\textbf{tell}(m_{19} = m'_{19} + Dt \times (vp - 0.0001m'_{19}))), \\
 & & \qquad \textbf{next}\ (\textbf{tell}(m_{19} = m'_{19} - Dt \times (0.0001m'_{19}))))) \\
Glucose & \stackrel{\text{def}}{=} & Regulate_{6}(bgalh, \textbf{next}\ (\textbf{tell}(m_{6} = m'_{6} + Dt \times (vg - 0.0001m'_{6}))), \\
 & & \qquad \textbf{next}\ (\textbf{tell}(m_{6} = m'_{6} - Dt \times (0.0001m'_{6})))) \\
Galactose & \stackrel{\text{def}}{=} & Regulate_{30}(bgalh, \textbf{next}\ (\textbf{tell}(m_{30} = m'_{30} + Dt \times (vg - 0.0001m'_{30}))), \\
 & & \qquad \textbf{next}\ (\textbf{tell}(m_{30} = m'_{30} - Dt \times (0.0001m'_{30})))) \\
Allolactose & \stackrel{\text{def}}{=} & Regulate_{8}(allolach, \\
 & & \quad Regulate_{8}(lacinh, \\
 & & \qquad \textbf{next}\ (\textbf{tell}(m_{8} = m'_{8} + Dt \times ((0.1m'_{29} + 0.2m'_{9}) - (0.5m'_{8} + m_{8} \times m'_{16})))), \\
 & & \qquad \textbf{next}\ (\textbf{tell}(m_{8} = m'_{8} + Dt \times (0.1m'_{29} - (0.5m'_{8} + m'_{8} \times m'_{16}))))), \\
 & & \quad Regulate_{8}(lacinh, \\
 & & \qquad \textbf{next}\ (\textbf{tell}(m_{8} = m'_{8} + Dt \times ((0.1m'_{29} + 0.2m'_{9}) - 0.5m'_{8}))), \\
 & & \qquad \textbf{next}\ (\textbf{tell}(m_{8} = m'_{8} + Dt \times (0.1m'_{29} - 0.5m'_{8})))))
\end{array}
$$

$$
Hydrolysis \stackrel{\text{def}}{=} !\ LacOut \parallel !\ LacIn \parallel !\ Glucose \parallel !\ Galactose \parallel !\ Allolactose
$$

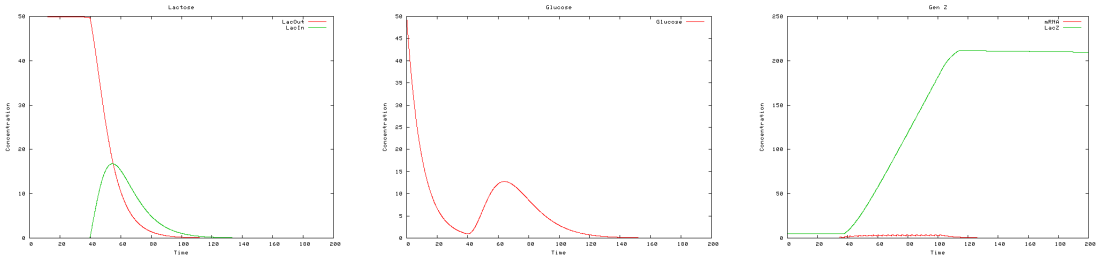
3.7 An integrated model with system simulations

Processes defined in previous sections are integrated in process *GRN*:

$$
\begin{array}{rcl}
GRN & \stackrel{\text{def}}{=} & \textbf{local}\ Ts, svar_1, ..., svar_k, m_1, ..., m_n, m'_1, ..., m'_n\ \textbf{in} \\
 & & Dynamic \parallel Signal \parallel ControlRegion \parallel StructuralGenes \parallel Induction \parallel Repression \parallel Hydrolysis
\end{array}
$$

This concurrent model is implemented in **sntccSim**, a simulation tool we developed in the concurrent constraints language Mozart [20]. **sntccSim** runs both `sntcc` and `ntcc` specifications. **sntccSim** allows to define procedures and recursive processes. A very important feature of **sntccSim** is that several constraint systems can be included in the same model. Indeed, in our case study, we use constraints over finite domains [19] and real intervals [4] to implement the constraint-based model of the lac operon described in this paper.

Reading the resulting store of each time unit it is possible to visualize the evolution of concentrations of lactose (i.e., inside and outside the cell), glucose and LacZ (i.e., mRNA and β-galatosidase protein). They are plotted in figure 2.



(a) Lactose (b) Glucose (c) LacZ

Figure 2: Simulation Results

4 A logic-based approach to verity system properties

In this section we describe a logic-based approach to verify system properties using the inference system associated with `ntcc`. We focus in a proof of *stability* in the system. As case of study, we verify that if CAP protein reaches a stable state, variable m_4 has value Vs.

Rewriting the definition of CAP we have:

$$CAP \stackrel{\text{def}}{=} \; !\,\mathbf{next}\,(\mathbf{tell}(m_4 = m_4' + Dt \times (1.0 - 0.1m_4'))) \;\equiv\; !\,\mathbf{next}\,(\mathbf{tell}(m_4 = m_4' + \Delta m_4))$$

The formula for process CAP is: $CAP \vdash \Box \circ (\; m_4 = m_4' + \Delta m_4 \;)$. From the definition of CAP it follows that when CAP protein is stable, the statement $\Delta m_4 = 0.0$ must be true. So, the condition for stability in CAP could be represented in the following `ntcc` process definition:

$$StableProperty \stackrel{\text{def}}{=} \star_n \mathbf{tell}(\Delta m_4 = 0.0)$$

where n is a time delay long enough to reach stability in CAP.

The formula for $StableProperty$ is: $StableProperty \vdash \Diamond \Delta m_4 = 0.0$.

The following assertion represents a stable state of CAP protein:

$$CAP \parallel StableProperty \vdash \Diamond \Box m_4 = Vs$$

The proof is formally expressed using the inference system associated with `ntcc`:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{StableProperty \vdash \Diamond \Delta m_4 = 0.0}}{StableProperty \vdash \Diamond \;(\; \Delta m_4 = 0.0 \,\dot{\wedge}\, Dt \times (1.0 - 0.1m_4') = 0.0 \;)}\,\text{LCONS} \quad \dfrac{}{StableProperty \vdash \Diamond \;(\; \Delta m_4 = 0.0 \,\dot{\wedge}\, m_4' = 10.0 \;)}\,\text{LCONS}}{CAP \vdash \Box \circ (\; m_4 = m_4' + \Delta m_4 \;) \quad CAP \parallel StableProperty \vdash \;(\; \Box \circ (\; m_4 = m_4' + \Delta m_4 \;)\;) \,\dot{\wedge}\, (\; \Diamond \;(\; \Delta m_4 = 0.0 \,\dot{\wedge}\, m_4' = 10.0 \;)\;)}\,\text{LPAR}}{CAP \parallel StableProperty \vdash \;(\; \Box \circ (\; m_4 = m_4' + \Delta m_4 \;)\;) \,\dot{\wedge}\, (\; \Diamond m_4 = 10.0 \;)\;)}\,\text{LCONS}}{CAP \parallel StableProperty \vdash \Diamond \;(\; \Box \circ (\; m_4 = m_4' + \Delta m_4 \;) \,\dot{\wedge}\, (\; m_4 = 10.0 \;)\;)}\,\text{LCONS}$$

Since the value of m_4 at time t is equal to that of m_4' at the next time unit, we can perform the following deduction:

$$\dfrac{\dfrac{\dfrac{}{CAP \parallel StableProperty \vdash \Diamond \;(\; \Box \circ (\; m_4 = m_4' + \Delta m_4 \;) \,\dot{\wedge}\, (\; m_4 = 10.0 \;)\;)}}{CAP \parallel StableProperty \vdash \Diamond \;(\;\; (\; \Box \circ (\; m_4 = m_4' + \Delta m_4 \;)\;) \,\dot{\wedge}\, (\; m_4 = 10.0 \;) \,\dot{\wedge}\, (\;\; \circ (\; m_4' = 10.0 \;)\;)\;)}\,\text{LCONS}}{CAP \parallel StableProperty \vdash \Diamond \Box m_4 = 10.0}\,\text{LCONS}$$

The above logical expression proves stability in CAP protein when $m_4 = Vs = 10.0$ in an undetermined time in the future. This value continues for the future. Finally, due to *Lemma 1.3* we can be sure that

the proof is also valid taking into account the rest of the system. So, the stability value of CAP can be obtained by two different approaches: following the steps in the operational semantics of `ntcc` simulated with **sntccSim** (see figure 3) or by means of a logical-temporal proof done with the inference system associated with `ntcc`.

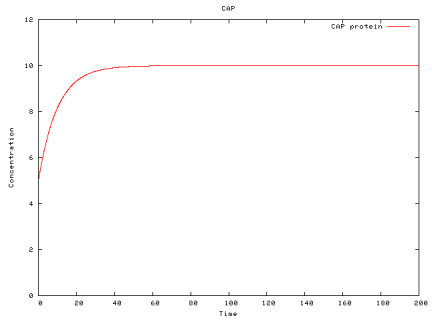


Figure 3: CAP protein

5 Concluding Remarks

In this paper we have proposed a framework to model, simulate and verify genetic regulatory networks and illustrate its use in a model of the lac operon. This framework is formally based on `ntcc`, a constraint-based process calculus. A simulation tool to execute `ntcc` processes following the operational semantics of the calculus is presented. An important feature is that `ntcc` allows to *reason* about the models specified using the calculus. In this way, we have shown how we can predict future behavior using the proof system associated with the `ntcc` temporal logic. We will present some concluding ideas about the most important features related with the framework proposed in this paper.

A Suitable Methodology to Model, Simulate and Verify Biological Systems. Although finding an appropriate language for modeling biological systems is an important task, devising a complete methodology to model, simulate and verify such systems is perhaps more crucial. We believe that a methodology based on the timed ccp model has considerable advantages for modeling and verifying biological systems. Reasons supporting this claim include: the natural use of concurrent processes to model biological entities, the notion of time to express the evolution of dynamic biological systems, the notion of constraint as a way of modeling incomplete information, and the inclusion of quantitative parameters in the models by choosing an appropriate constraint system. Moreover, this conceptual framework can be directly supported both by programming languages and tools based on the ccp model to perform system simulations.

Time and Non-deterministic/Asynchronous Behavior. In some biological scenarios the most important things to observe are the initial and final states of the system. In other situations, however, having a strict control of the evolution of the system might be required. That is the case when a dynamic biological system is modeled and simulated. We believe that having an explicit notion of time is fundamental to achieve such a control. The discrete-time features of `ntcc` allow very expressive system specifications in which the evolution of the system can be observed step-by-step. Moreover, these discrete time features can be used to model continuous time systems. Our case study is a good example of this. Although the issue of modeling the function of biological systems using process calculi has been previously studied [5, 12], to the best of our knowledge none of these calculi allow to represent time, non-deterministic and asynchronous behavior within the same model. `ntcc` provides explicit constructions to express these kinds of behavior, which may be very useful to represent several biological situations.

As future work, we plan to proposed components in `ntcc` to model malfunctions in biological models. We believe that with the use of some `ntcc` operators, in particular $\star$, we could model in a very realistic way the unpredictable behavior of several diseases. Moreover, based on the results achieved so far, we plan to extend our work to admit the inclusion of ordinary differential equations in models. This allows us to formally model very complex biological systems which already have a mathematical formulation.

References

[1] L. Cardelli. Brane calculi. In V. Danos and V. Schachter, editors, *CMSB*, volume 3082 of *Lecture Notes in Computer Science*, pages 257–278. Springer, 2004.

[2] G. Ciobanu, V. Ciubotariu, and B.Tanasa. A pi-calculus model of the na pump. *Genome Informatics*, pages 469–472, 2002.

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

[4] AVISPA Research Group. *XRI: Extended Real Interval.*, 2004. Available at `http://home.gna.org/xrilpoz`.

[5] J. Gutiérrez, J. A. Pérez, and C. Rueda. Modelamiento de sistemas biológicos usando cálculos de procesos concurrentes. *Epiciclos*, 4, 2005.

[6] J. Krivine and V. Danos. Formal molecular biology done in CCS-R. In *BioConcur 2003, Workshop on Concurrent Models in Molecular Biology*, 2003.

[7] B. Lewin. *Genes VII.* Oxford University Press, 2000.

[8] R. Milner. *Communicating and Mobile Systems: The π-Calculus.* Cambridge University Press, 1999.

[9] S. Miyano and H. Matsuno. How to model and simulate biological pathways with petri nets - a new challenge for systems biology -. *25th International Conference on Application and Theory of Petri Nets*, June 2004.

[10] M. Nielsen, C. Palamidessi, and F. D. Valencia. Temporal concurrent constraint programming: Denotation, logic and applications. *Nord. J. Comput.*, 9(1):145–188, 2002.

[11] C. Olarte and C. Rueda. A stochastic non-deterministic temporal concurrent constraint calculus. In IEEE Computer Society, editor, *Proceedings of XXV International conference of the chilean computer science society*, 2005.

[12] D. Prandi, C. Priami, and P. Quaglia. Process calculi in a biological context. *Concurrency Column*, February 2005.

[13] C. Priami. Stochastic pi-calculus with general distributions. In CLUP, editor, *Proceedings of PAPM '96*, 1996.

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

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

[16] C. Rueda, G. Alvarez, L. O. Quesada, G. Tamura, F. D. Valencia, J. F. Diaz, and G. Assayag. Integrating constraints and concurrent objects in musical applications: A calculus and its visual language. *Constraints*, 6(1):21–52, 2001.

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

[18] V. A. Saraswat, R. Jagadeesan, and V. Gupta. Timed default concurrent constraint programming. *Journal of Symbolic Computation*, 22(5/6):475–520, 1996.

[19] C. Schulte and G. Smolka. *Finite Domain Constraint Programming in Oz. A Tutorial.*, 2004. Available at `www.mozart-oz.org`.

[20] G. Smolka. The Oz programming model. In Jan van Leeuwen, editor, *Computer Science Today*, volume 1000 of *LNCS*, pages 324–343. Springer - Verlag, 1995.

[21] F.D. Valencia. *Temporal Concurrent Constraint Programming.* PhD thesis, University of Aarhus, November 2002.