Università degli Studi "G. D'Annunzio"
Dipartimento di Scienze

Constraint-based Languages to Model the Blood Coagulation Cascade

Stefano Bistarelli Marco Bottalico
Francesco Santini

November 12, 2009

Technical Report no. R-2009-002 *Research Series*

Constraint-based Languages to Model the Blood Coagulation Cascade

Stefano Bistarelli [1] Marco Bottalico [2] Francesco Santini [1]

[1] *Dipartimento di Scienze, Università "G. d'Annunzio", Pescara, Italy*
Istituto di Informatica e Telematica (CNR), Pisa, Italy
ipartimento di Matematica e Informatica, Università di Perugia, Italy
bista@sci.unich.it
[2] *Dipartimento di Scienze, Università "G. d'Annunzio", Pescara, Italy*
bottalic@sci.unich.it
[2] *Dipartimento di Scienze, Università "G. d'Annunzio", Pescara, Italy*
Istituto di Informatica e Telematica (CNR), Pisa, Italy
santini@sci.unich.it

November 12, 2009

Abstract. In this paper, we use different formal languages based on constraints to model biological reactions and use the blood coagulation cascade as a running example to analyze similarities and differences. Moreover we compare the results of the simulation with in vitro experiments in the medical scientific literature by also considering an hepatic inhibitor drug. Our study show how assets obtained with in vitro experiments could be modeled in silico using constraint languages.

Keywords: *Biochemical Reactions, Blood Coagulation, Concurrent Constraint Programming*

Contents

1 Introduction

System biology is an interdisciplinary science, integrating experimental activity and mathematical modeling, which studies the dynamical behaviors of biological systems. While current genome projects provide a huge amount of data on genes or proteins, lot of research is still necessary to understand how the different parts of a biological system interact. Mathematical and computational techniques are central in this approach to biology, as they provide the capability of formally describing living systems and studying their proprieties.

A variety of formalisms for modeling biological systems has been proposed in literature. In [4], the author distinguishes three basic approaches: discrete, stochastic and continuous. Discrete models are based on discrete variables and discrete state changes; continuous models are based on differential equations that typically model biochemical reactions; finally in the stochastic ones the probabilities are introduced through random variables, which are usually defined by taking into account the kinetics laws. In the latest approach there is a simplified representation of the processes and an integration of the stochastic noise in order to get more realistic models.

In [16] it is shown that there are two formalisms for mathematically describing the time behavior of a spatially homogeneous chemical systems: the deterministic approach and the stochastic one. The first regards the time evolution as a continuous and predictable process which is governed by a set of ordinary differential equations (the "reaction-rate equations"), while the seconds regard the time evolution as a kind of random-walk process which is governed by a single differential-difference equation (the "master equation").

The goal of this paper is to show how different kinds of constraint-based languages can model biochemical reactions and to compare and to use their features. The languages studied are the *Stochastic Concurrent Constraint Programming* (*sCCP*) [8], the *non-Deterministic Temporal Concurrent Constraint Programming* (*ntCC*) [18] and the *Hybrid Concurrent Constraint Programming* (*HCC*) [24].

We want to show that these formalisms can be used to simulate in vitro reactions, and can be then adopted to save time and costs by using an automated simulation in silico. The obtained in silico results have been produced by using the in vitro parameters in medical literature (i.e. the six reaction definitions, the stoichiometric coefficients and the factor's concentration). In vitro and in silico results are the same in terms of reduction of the thrombin formations.

2 Constraint-based Languages for Biology

Concurrent Constraint Programming (*CCP*) languages [22] concern the behaviour of a set of concurrent agents with a shared store, which is a conjunction of constraints. Each computation step possibly adds new constraints to the store. Thus information is monotonically added to the store until all agents have evolved. The final store is a refinement of the initial one and it is the result of the computation. The concurrent agents do not communicate

directly with each other, but only through the shared store, by either checking if it entails a given constraint (*ask* operation) or adding a new constraint to it (*tell* operation).

We use languages based on *Concurrent Constraint Programming* [22], because we want a powerful framework which provides the fine grained concurrency desirable for compositionality. For example, the blood cascade coagulation is modelled by composing (parallel execution) programs representing each step of the cascade. In this context we use compositionality with the meaning of "compositionality of biochemical reactions".
The languages based on *Concurrent Constraint Programming* [22] are very expressive, being built on top of arbitrary constraint systems, and are also declarative. Each program is a logical formula, that make easy the reasoning about the models. The matching between the logic programming and the real behavior is given by the notion of competition to obtain the shared resources (i.e. the *race competition*). In nature, if we have a reaction which involves many components, these components compete to reach the single necessities; for example, the species less suited to compete for limited resources should either adapt or face extinction.

2.1 Stochastic Concurrent Constraint Programming

sCCP [8] derives from classical CCP [22] by adding a stochastic duration to the instructions interacting with the constraint store C, i.e. *ask* and *tell* by means of stream variables.
ask and *tell* are identified by a rate function λ: $tell_\lambda(c)$ and $ask_\lambda(c)$, with the meaning that the reaction occurs in a stochastic time T, following the probability law $f(\tau) = \lambda e^{-\lambda\tau}$; $tell_\infty$ stay instead for an instantaneous execution while $tell_0$ never occurs.

The stream variables are time-varying variables, they can be easily modeled in sCCP as growing lists with a unbounded tail: $X = [a_1, ..., a_n|Y]$. When the quantity changes, we simply need to add the new value, say b, at the end of the list by replacing the old tail variable with a list containing b and a new tail variable: $Y = [b|Y']$. When we need to know the current value of the variable X, we need to extract from the list, the value immediately preceding the unbounded tail. These variables are connected to the non-monotonic behaviour in sCCP.

With the *sCCP* the author in [5] models some biochemical reactions: an enzymatic reaction and a MAP-kinase cascade; in [8] a gene regulatory networks and in particular a bistable circuit, a repressilator and a circadian clock; in [6] models the dealing with protein complexes: a agent-based protein structure prediction and a protein folding simulation. in [7] describes and models a molecular interaction maps.

In the sCCP the author provides a model checker implemented by PRISM [5], a work on bisimulation and Temporal logic in [5].

2.2 Non Deterministic Temporal Concurrent Constraint Programming

In ntCC [18], time is conceptually divided into discrete intervals. In a time unit, a process P gets an input c (a constraint) from the environment; it executes with this input as the initial

store and it outputs the resulting store d to the environment, when it reaches its resting point. The resting point determines a residual process Q, which is then executed in the next time unit. With the "next" operator we can transfer information from one time unit to the following one.

The constructs different from classical cc are the following:

- *when c do P* is equivalent to $ask(c) \rightarrow P$. Its function is asking information about the state of the system;
- $next(P)$ represents the activation of P in the next time interval;
- *unless c next* (P): P will be activated only if c cannot be inferred from the current store (it is connected to the non-monotonic behaviour in the ntCC);
- $\star\ P$ allow us to express partial information on the time units where processes are executed. Process $\star\ P$ represents an arbitrary long but finite delay for the activation of P. $\star\ P_{[n,m]} = next^n\ (P)\ +\ next^{n+1}\ (P)\ +\ ...\ +\ next^{m-1}\ (P)\ +\ next^m\ (P)$;
- $!\ P$ represents $P \parallel next\ (P) \parallel next^2\ (P) \parallel ...$, many copies of P but one at a time unit.

In the implementation we used the $next(P)$ operator encoded with the symbol $next([[P]])$, the $\star P$ operator encoded with the symbol $rep([[P]])$ and the $!P$ operator encoded with the symbol $star([[P]])$.
With the *ntCC* in [18] the author models a SP-pump and an interaction between genes.
In ntCC we have a Model checking for a finite time interval [14], a Weak bisimulation [25] and a Linear temporal logic.

2.3 Hybrid Concurrent Constraint Programming

HCC is a powerful framework for modeling, analyzing and simulating hybrid systems, i.e., systems that exhibit both discrete and continuous change. It is an extension of *Timed Default CCP* [23] over continuous time. One of the major problems in the original CCP framework is that CCP programs can detect only the presence of information, not the absence. Timed Default CCP extends CCP by a negative *ask* combinator ($if\ a\ else\ A$) which imposes the constraint a at the program A.

The CCP paradigm has no concept of timed execution. For modeling discrete, reactive systems, it was introduced the idea (from synchronous programming) that the environment reacts with a system (program) at discrete time ticks. At each time tick, the program executes a CCP program, outputs the resulting constraint, and sets up another program for execution at the next clock tick. Concretely, this lead to the addition of two control constructs to the language *next A* (execute A at the next time instant), and *always A* (execute A at every time instant). Thus, intuitively, the discrete timed language was obtained by uniformly extending the not-timed language (CCP or Default CCP) across (integer) time [17]. *Next A* and *always A* are connected to the non-monotonic behaviour in HCC.

The authors of [17] allow constraints expressing initial value (integration) problem, e.g. constraints of the form $init(X = 0)$; $cont(dot(X) = 1)$ read as follows: the initial value of X is 0, the first derivative of X is 1 and from these we can infer that $X = t$ at time t.
In HCC a new temporal control construct has been added to the not-timed Default CCP: *hence A*. Declaratively, *hence A* imposes the constraints of A at every time instant after the current one. Operationally, if *hence A* is invoked at time t, a new copy of A is invoked at each instant in $(t; 1)$.
With the *HCC* [24] we have two running examples: the cell differentiation and the interaction between 2genes.
Concerning HCC we can find a model checker providing a simulation of hybrid automata, the HyTech model checker [26] that performs a bisimulation with Labeled Markow processes and a linear Temporal logic verification [24].

3 Biochemical Reactions and Blood Coagulation

The blood coagulation process can be defined by a set of biochemical reactions among proteins. In general, all the interactions that take place in a cell, can be arranged into a diagram, thus obtaining a biochemical network. Biochemical networks can be represented using equations, usually described as follows:

$$m_1R_1 + ... + m_kR_k \longrightarrow n_1P_1 + ... + n_hP_h. \tag{1}$$

In the equation (1) R_i are the reactants, P_i the products, m_i and n_i are the stoichiometric coefficients[1]. Along with this expression, there is a real number representing its basic expected "frequency"; this number is related to the adopted kinetic model [5]: the most important kinetic laws that we consider are Michaelis-Menten (MM), Hill's kinetics (HK) and Mass Action (MA) [5]. According to how we want to describe a reaction, we can use one or another kinetic law, and in this way we can model different behaviors.
This is the scheme to represent the Michaelis-Menten kinetics [12]:

$$E + S \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} ES \overset{k_2}{\rightharpoonup} E + P. \tag{2}$$

In the equation (2) the enzyme E does not magically convert S into P, it must first come into a physical contact with it, i.e. E binds S to form an enzyme-substrate complex ES. The terms k_1, k_{-1} and k_2 are rate constants for, respectively, the association of substrate and enzyme, the dissociation of unaltered substrate from the enzyme and the dissociation of product from the enzyme.
At this point there are two important hypotheses in order to safely use the Michaelis-Menten kinetics:

[1]The stoichiometric coefficients of a chemical equation represent the molar ratio among material quantities (expressed in nanomolars).

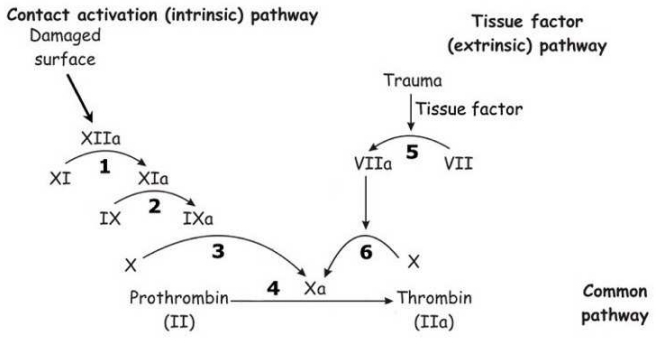

Figure 1: Coagulation Cascade.

1. $[S] >> [E]$ i.e. the quantity of the substrate S is significatively bigger than the quantity of the enzyme E.

2. The system is in a quasi steady-state i.e. the ES complex is being formed and broken down at the same rate, so overall $[ES]$ is constant.

Under these hypotheses, the most important equations in the Michaelis-Menten kinetics are:
$K_M = \frac{k_{-1}+k_2}{k_1}$ Michaelis constant. It measures the affinity of the enzyme for the substrate: if K_M is small there is a high affinity, and viceversa.
$V_{MAX} = V_0 = k_2[E_0]$. This is the maximum rate, would be achieved when all of the enzyme molecules have substrate bound (Hp1). $[E_0]$ is the starting quantity of enzyme E. k_2 is also called k_{cat}.

In the following of the paper we will use blood coagulation phenomenon as a running example for our study. The blood coagulation is part of an important host defense mechanism termed *hemostasis*, that is the cessation of blood loss from a damaged vessel [27]. Blood clotting is a very delicately balanced system; when hemostatic functions fail, hemorrhage or thromboembolic phenomena may result. The chemical reactions that constitute the whole process can be seen as a decomposition of many kinds of enzymatic reactions, involving reactants, products, enzymes, substrates, stoichiometric coefficients, proteins, inhibitors and chemical accelerators.

In our work we use an exemplification model, given in Fig. 1. In the intrinsic and the extrinsic pathways (see Fig. 1), the chain of events leading to coagulation is the set in motion merely by the exposure of plasma to non endothelial surfaces such as a glass in vitro, or a collagen fibres in basement membranes in vivo [2].

The downward sequence of reactions in Fig. 1 justifies the term “cascade”. In the extrinsic pathway, coagulation is achieved as a result of an injury to the vessel wall from the “outside”. This pathway is initiated when the tissue factor becomes mixed with factors *II*, *VII*, *X*, *XII* and calcium of the blood plasma. In the extrinsic pathway we have a sequence

$react(S, P, K_M, V_0) :-$
$\quad ask_{r_{MM}(K_M,V_0,S)}(S > 0).\ (tell_\infty(S\ \$= S - 1) || tell_\infty(P\ \$= P + 1)).\ react(S, P, K_M, V_0)$

Figure 2: Blood coagulation in sCCP

of reactions leading to fibrin formation, beginning with the contact activation of factor *XII*, and resulting in the activation of factor *X* to initiate the common pathway of coagulation. The extrinsic pathway merges with the intrinsic one after the activation of factor *X*.

We have both pathways and all the interactions to get to the thrombin formation (Factor *IIa*). We have labeled each reaction, in order to make every steps of our analysis as clear as possible.

4 Modeling Blood Coagulation with sCCP, ntCC and HCC

In this section we show how the simplified cascade in Fig. 1 can be properly modeled with the three languages. For sake of brevity we only show the piece of codes used to model the fist reaction in Fig. 1 ($XI + XIIa = XIa$) labeled with the number 1. However all the code is available in [10].

4.1 sCCP

The first biochemical equation given in (2), can be modelled in sCCP with the following recursively defined method [8] in Fig. 2.
The rate λ of the *ask*, is computed by the Michaelis-Menten kinetics $r_{MM(K_M,V_0,\,S)} = \frac{V_0\ S}{S+K_M}$ in Fig. 2. Roughly the program inserts in the store the current value for the variables, it checks the value of the factor S, then, with an immediate effect, it updates the values for the factors S (reagent) and P (product) with the new values. Subsequently it executes a new instance of the program.

4.2 ntCC

In the following we encode in ntCC the first reaction in Fig. 1, we show only the enzyme formation (*DEe*). E (*DEe*), S (*DEs*), ES (*DEes*) and P (*DEp*) can combined to show the collaborative function.

```
% Title :  Blood Coagulation
% Authors :  Marco Bottalico
% Created :  September 1 / 2009
% Last Modified :  September 1 / 2009
% Description :  Sixth Reaction
```

```
% functor import
% Application System
% RI at ’./../../ntccSim/ri/RI.ozf’
% NTCC at’./../../ntccSim/ntccSim.ozf’
% Open define
DEVars = vars(e:_ s:_ es:_ p:_ s1:_ es1:_ p1:_ s2:_ es2:_ p2:_ s3:_
es3:_ p3:_ s4:_ es4:_ p4:_ es5:_ t:_)
T = vars(tntcc:_)
SVars = Record.adjoin T DEVars
Resolution = 10.0
Dt = 0.1 / Resolution
MaxTime = 200
K1 = 1.0
Km1 = 0.0908
K2 = 0.0092
K3 = 1.0
Km3 = 2.09
K4 = 2.4
K5 = 1.0
Km5 = 0.267
K6 = 0.223
K7 = 1.0
Km7 = 0.0325
K8 = 0.0075
K9 = 1.0
Km9 = 0.02
K10 = 0.22
K11 = 1.0
Km11 = 0.0325
K12 = 0.04


T:::0#1000000
Record.forAll DEVars proc{$ Xi}
Xi = {RI.var.exp "[-10000.0,10000.0]"} end


%Time Process (discrete and continuous)
TimeS = par(tell(proc{$ Root} Root.current.tntcc =:  0 end)
tell(proc{$ Root} {RI.eq Root.current.t 0.0} end)
rep(par(next(tell(proc{$ Root}
Root.current.tntcc =:  Root.residual.tntcc + 1 end))
next(tell(proc{$ Root}
```

```
{RI.eq Root.current.t {RI.plus Root.residual.t Dt}} end)))))

Start = par(
tell(proc{$ Root} {RI.eq Root.current.e 10.0} end)
tell(proc{$ Root} {RI.eq Root.current.s 10.0} end)
tell(proc{$ Root} {RI.eq Root.current.es 0.0} end)
tell(proc{$ Root} {RI.eq Root.current.p 0.0} end)
tell(proc{$ Root} {RI.eq Root.current.s1 10.0} end)
tell(proc{$ Root} {RI.eq Root.current.es1 0.0} end)
tell(proc{$ Root} {RI.eq Root.current.p1 0.0} end)
tell(proc{$ Root} {RI.eq Root.current.s2 10.0} end)
tell(proc{$ Root} {RI.eq Root.current.es2 0.0} end)
tell(proc{$ Root} {RI.eq Root.current.p2 0.0} end)
tell(proc{$ Root} {RI.eq Root.current.s3 10.0} end)
tell(proc{$ Root} {RI.eq Root.current.es3 0.0} end)
tell(proc{$ Root} {RI.eq Root.current.p3 0.0} end)
tell(proc{$ Root} {RI.eq Root.current.s4 10.0} end)
tell(proc{$ Root} {RI.eq Root.current.es4 0.0} end)
tell(proc{$ Root} {RI.eq Root.current.p4 0.0} end)
tell(proc{$ Root} {RI.eq Root.current.es5 0.0} end)
)

DEe = rep(
next(
tell(
proc{$ Root}
Ec Er Ed DeE Sr ESr in
Ec = Root.current.e
Er = Root.residual.e
Sr = Root.residual.s
ESr = Root.residual.es
Ed = sub(times(ESr plus(Km1 K2)) (times(Sr times(K1 Er))))
DeE = eq(Ec plus(Er times(Ed Dt)))
{RI.hc4 DeE}
end )))

DEs = rep(
next(
tell(
proc{$ Root}
Sc Sr Sd DeS Er ESr in
```

```
Sc = Root.current.s
Sr = Root.residual.s
Er = Root.residual.e
ESr = Root.residual.es
Sd = sub(times(Km1 ESr) times(Sr times(K1 Er)))
DeS = eq(Sc plus(Sr times(Sd Dt)))
RI.hc4 DeS end )))

DEes = rep(
next(
tell(
proc{$ Root}
ESc ESr Er Sr ESd DeES in
ESc = Root.current.es
ESr = Root.residual.es
Er = Root.residual.e
Sr = Root.residual.s
ESd = sub(times(K1 times(Er Sr)) times(ESr plus(Km1 K2)))
DeES = eq(ESc plus(ESr times(ESd Dt)))
{RI.hc4 DeES}
end )))

DEp = rep(
next(
tell(
proc{$ Root}
Pc Pr ESr ES1c Pr S1c ALFA1 ALFA2 ALFA3 Pd DeP in
Pc = Root.current.p
Pr = Root.residual.p
ESr = Root.residual.es
ES1c = Root.current.es1
Pr = Root.residual.p
S1c = Root.current.s1
ALFA1 = times(K3 times(Pr S1c))
ALFA2 = times(ES1c plus(Km3 K4))
ALFA3 = times(K2 ESr)
Pd = sub(plus(ALFA3 ALFA2) ALFA1)
DeP = eq(Pc plus(Pr times(Pd Dt)))
{RI.hc4 DeP}
end )))
```

```
DEs1 = rep(
next(
tell(
proc{$ Root}
S1c S1r E1r ES1r S1d DeS1 in
S1c = Root.current.s1
S1r = Root.residual.s1
E1r = Root.residual.p
ES1r = Root.residual.es1
S1d = sub(times(Km3 ES1r) times(S1r times(K3 E1r)))
DeS1 = eq(S1c plus(S1r times(S1d Dt)))
{RI.hc4 DeS1}
end )))

DEes1 = rep(
next(
tell(
proc{$ Root}
ES1c ES1r E1r S1r ES1d DeES1 in
ES1c = Root.current.es1
ES1r = Root.residual.es1
E1r = Root.residual.p
S1r = Root.residual.s1
ES1d = sub(times(K3 times(E1r S1r)) times(ES1r plus(Km3 K4)))
DeES1 = eq(ES1c plus(ES1r times(ES1d Dt)))
{RI.hc4 DeES1}
end )))

DEp1 = rep(
next(
tell(
proc{$ Root}
P1c P1r ES1r ES2c P1r S2c BETA1 BETA2 BETA3 P1d DeP1 in
P1c = Root.current.p1
P1r = Root.residual.p1
ES1r = Root.residual.es1
ES2c = Root.current.es2
P1r = Root.residual.p1
S2c = Root.current.s2
BETA1 = times(K5 times(P1r S2c))
BETA2 = times(ES2c plus(Km5 K6))
```

```
BETA3 = times(K4 ES1r)
P1d = sub(plus(BETA3 BETA2) BETA1)
DeP1 = eq(P1c plus(P1r times(P1d Dt)))
{RI.hc4 DeP1}
end )))

DEs2 = rep(
next(
tell(
proc{$ Root}
ES2r E2r S2r ES5c E5r S2r EPSILON1 EPSILON2 EPSILON3 EPSILON4
EPSILON DeS2 S2c in
ES2r = Root.residual.es2
E2r = Root.residual.p1
S2r = Root.residual.s2
ES5c = Root.current.es5
E5r = Root.residual.p4
S2c = Root.current.s2
EPSILON1 = times(K11 times(E5r S2r))
EPSILON2 = times(Km11 ES5c)
EPSILON3 = times(K5 times(E2r S2r))
EPSILON4 = times(Km5 ES2r)
EPSILON = sub(EPSILON4 plus(EPSILON3 sub(EPSILON2 EPSILON1)))
DeS2 = eq(S2c plus(S2r times(EPSILON Dt)))
{RI.hc4 DeS2}
end )))

DEes2 = rep(
next(
tell(
proc{$ Root}
ES2c ES2r E2r S2r ES2d DeES2 in
ES2c = Root.current.es2
ES2r = Root.residual.es2
E2r = Root.residual.p1
S2r = Root.residual.s2
ES2d = sub(times(K5 times(E2r S2r)) times(ES2r plus(Km5 K6)))
DeES2 = eq(ES2c plus(ES2r times(ES2d Dt)))
{RI.hc4 DeES2}
end )))
```

```
DEp2 = rep(
next(
tell(
proc{$ Root}
ES2r ES3c ES5c P2c P2r S3c GAMMA1 GAMMA2 GAMMA3 GAMMA4 GAMMA DeP2
in
ES2r = Root.residual.es2
ES3c = Root.current.es3
ES5c = Root.current.es5
P2c = Root.current.p2
P2r = Root.residual.p2
S3c = Root.current.s3
GAMMA1 = times(K6 ES2r)
GAMMA2 = times(Km7 ES3c)
GAMMA3 = plus(times(K8 ES3c) times(K12 ES5c))
GAMMA4 = times(K7 times(P2r S3c))
GAMMA = plus(GAMMA1 plus(GAMMA2 sub(GAMMA3 GAMMA4)))
DeP2 = eq(P2c plus(P2r times(GAMMA Dt)))
{RI.hc4 DeP2}
end )))

DEs3 = rep(
next(
tell(
proc{$ Root}
S3c S3r E3r ES3r S3d DeS3 in
S3c = Root.current.s3
S3r = Root.residual.s3
E3r = Root.residual.p2
ES3r = Root.residual.es3
S3d = sub(times(Km7 ES3r) times(S3r times(K7 E3r)))
DeS3 = eq(S3c plus(S3r times(S3d Dt)))
{RI.hc4 DeS3}
end )))

DEes3 = rep(
next(
tell(
proc{$ Root}
ES3c ES3r E3r S3r ES3d DeES3 in
ES3c = Root.current.es3
```

```
ES3r = Root.residual.es3
E3r = Root.residual.p2
S3r = Root.residual.s3
ES3d = sub(times(K7 times(E3r S3r)) times(ES3r plus(Km7 K8)))
DeES3 = eq(ES3c plus(ES3r times(ES3d Dt)))
{RI.hc4 DeES3}
end )))

DEp3 = rep(
next(
tell(
proc{$ Root}
P3c P3r ES3r ES4c P3r S4c DELTA1 DELTA2 DELTA3 P3d DeP3 in
P3c = Root.current.p3
P3r = Root.residual.p3
ES3r = Root.residual.es3
ES4c = Root.current.es4
P3r = Root.residual.p3
S4c = Root.current.s4
DELTA1 = times(K9 times(P3r S4c))
DELTA2 = times(ES4c plus(Km9 K10))
DELTA3 = times(K8 ES3r)
P3d = sub(plus(DELTA3 DELTA2) DELTA1)
DeP3 = eq(P3c plus(P3r times(P3d Dt)))
{RI.hc4 DeP3}
end )))

DEs4 = rep(
next(
tell(
proc{$ Root}
S4c S4r E4r ES4r S4d DeS4 in
S4c = Root.current.s4
S4r = Root.residual.s4
E4r = Root.residual.p3
ES4r = Root.residual.es4
S4d = sub(times(Km9 ES4r) times(S4r times(K9 E4r)))
DeS4 = eq(S4c plus(S4r times(S4d Dt)))
{RI.hc4 DeS4}
end )))
```

```
DEes4 = rep(
next(
tell(
proc{$ Root}
ES4c ES4r E4r S4r ES4d DeES4 in
ES4c = Root.current.es4
ES4r = Root.residual.es4
E4r = Root.residual.p3
S4r = Root.residual.s4
ES4d = sub(times(K9 times(E4r S4r)) times(ES4r plus(Km9 K10)))
DeES4 = eq(ES4c plus(ES4r times(ES4d Dt)))
{RI.hc4 DeES4}
end )))

DEp4 = rep(
next(
tell(
proc{$ Root}
ES4r ES5c P4c P4r S2r ZETA1 ZETA2 ZETA3 ZETA DeP4 in
ES4r = Root.residual.es4
ES5c = Root.current.es5
P4c = Root.current.p4
P4r = Root.residual.p4
S2r = Root.residual.s2
ZETA1 = times(K10 ES4r)
ZETA2 = times(ES5c plus(Km11 K12))
ZETA3 = times(K11 times(P4r S2r))
ZETA = plus(ZETA1 sub(ZETA2 ZETA3))
DeP4 = eq(P4c plus(P4r times(ZETA Dt)))
{RI.hc4 DeP4}
end )))

DEes5 = rep(
next(
tell(
proc{$ Root}
E5r S2r ES5c ES5r ES5d DeES5 in
E5r = Root.residual.p4
S2r = Root.residual.s2
ES5c = Root.current.es5
ES5r = Root.residual.es5
```

```
ES5d = sub(times(K11 times(E5r S2r))
times(ES5r plus(Km11 K12)))
DeES5 = eq(ES5c plus(ES5r times(ES5d Dt)))
{RI.hc4 DeES5}
end )))

DESystem = par(Start TimeS DEe DEs DEes DEp DEs1 DEes1
DEp1 DEs2 DEes2 DEp2 DEs3 DEes3 DEp3 DEs4 DEes4 DEp4
DEes5)
```

Er, *Sr* and *ESr* represent the residual quantity of enzyme, substrate and enzyme-substrate while *Ec* represents the current quantity of enzyme. *Ed* represents the differential equation for the enzyme formation. *DeE* represents the enzyme quantity changes over time. The authors uses the Finite-difference methods to approximate the solutions to differential equations by replacing derivative expressions with approximately equivalent difference quotients. The finite-difference methods are numerical methods for approximating the solutions to differential equations using finite difference equations to approximate derivatives. In particular, if we have $u(x+h) = u(x) + u'(t)dt$ become $Xc = Xr + XdDt$. The row $\{$*RI.hc4 DeE* $\}$ impose a propagator on the restriction *DeE* using the consistency rule hc4 on the extended real interval (XRI [1]). With the *Start* we can begin the process from the differential equation process, beginning to the initial values (*Root.current.value*).

Finally the result is approximated with the *hc4* operator. The HC4 propagator is used to avoid splitting complex constraints into a set of primitive basic constraints. represented by *XRI.sinXY*, *XRI.cosXZ* and $Y = Z$. Using hc4 instead the user writes this constraint thus $RI.hc4eq(sin(X), cos(X))$. Hc4 is useful for constraints with multiple occurrences of the same variable. The Hc4 propagator was proposed by Benhamou Goualard, Granvilliers and Puget in [3].

4.3 HCC

In HCC, we have instead the following syntax (for all the reactions in Fig. 1).

```
e=100, s=100, es=0, p=0, s1=100, es1=0, p1=0, s2=100, es2=0,
p2=0, s3=100, es3=0, p3=0, e4=100, s4=100, es4=0, p4=0, es5=0,

always {k1=1, km1=0.0908, k2=0.0092, k3=1,
km3=2.09, k4=2.4, k5=1, km5=0.267, k6=0.223,
k7=1, km7=0.0325, k8=0.0075, k9=1, km9=0.02,
k10=0.22, k11=1, km11=0.0325, k12=0.04,

cont(e),cont(s),
```

```
if (s >= 0.00000000001) then {
e' = ((km1+k2)es)-(k1es),
p' = k2es+((km3+k4)es1)-(k3p's1),
s' = (km1es)-(k1es),
es'= (k1es)-((km1+k2)es),
es1' = (k3p's1)-(km3+k4)es1,
s1' = (km3es1)-(k3p's1),
p1' = (k4es1)+((km5+k6)es2)-(k5p1s2),
s2' = (km5es2)+(km11es5)-(k11p4s2)-(k5p1s2),
es2' = (k5p1's2)-((km5+k6)es2),
p2' = (k6es2)+(km7es3)+((k8es3)+(k12es5))-(k7p2s3),
s3' = (km7es3)-(k7p2's3),
es3' = (k7p2's3)-((km7+k8)es3),
p3' = k8es3,
e4' = ((km9+k10)es4)-(k9e4s4),
s4' = (km9es4)-(k9e4s4),
es4' = (k9e4s4)-((km9+k10)es4),
p4' = (k10es4)+((km11+k12)es5)-(k11p4s2),
es5' = (k11p4s2)-((km11+k12)es5) }

else { e'=0, p'=0, s'= 0, es'= 0, p1'= 0, s1'= 0, es1'= 0,
s2'=0, es2'=0, p2'=0, s3'=0, es3'=0, p3'=0, e4'=0, s4'=0,
es4'=0, p4'=0, es5'=0 } }, sample(p4)
```

We can observe that in the first block of code above, we have the initial conditions with the quantity of enzyme, substrate, enzyme-substrate and product. In the second one we observe the stoichiometric coefficients, whose values are always valid. In the third block we check the concentrations of e and s, to start the computation with the differential equations, to derive the quantity in the first step. In the fourth block we have the branch *else*; it is used if the if guard results mistaken. Finally in the last block we print only the last p quantity.

5 Results and Comparison of the Frameworks

Warfarin is an anticoagulant also known with several commercial names as Coumadin, Jantoven, Marevan, and Waran. Warfarin inhibits the hepatic secretion of some factors in the blood clotting. In particular, these factors are *II*, *VII*, *IX*, *X* [19]. As we can note from the Fig. 3 it is clearly visible the reduction of the thrombin (Factor *IIa*) produced.

The graph generated with all the six reactions of the simplified cascade in Fig. 1 is shown in Fig. 3: it represents the concentration of thrombin (in nanomolars on the y axis) as time passes (on x axis). This figure is generated by the HCC framework, and we can note that, during time, the production of thrombin (*FIIa*) is increased. The graph in Fig. 3

	ntCC	*HCC*
time	27.23 sec	9.84 sec
% cpu	71.2	31.5
% memory	7.7	2.7

Table 1: Performance for the simulation of the blood coagulation cascade with ntCC and HCC.

has been obtained by using the same input parameters (i.e. the six reaction definitions, the stoichiometric coefficients and the factor's concentration).

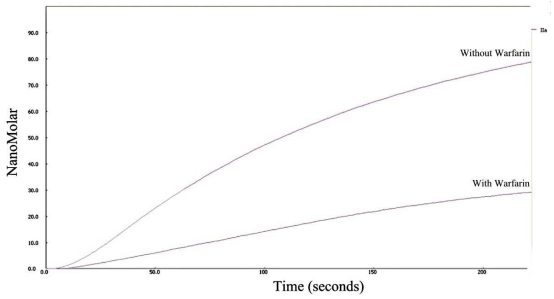

Figure 3: Formation of thrombin (factor *IIa*) with and without Warfarin, in silico.

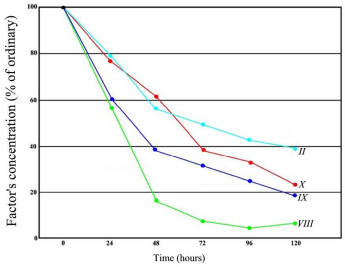

Figure 4: Inhibition of factors *II*, *VII*, *IX*, and *X*, in vitro.

In our work, we have reduced the substances concentration in order to study the production of thrombin (*FIIa*). As we can see in Fig. 3, we have the same reduction in the formation of thrombin (*FIIa*), in all the languages.

The performance in Tab. 1 have been collected by using the *top* utility under *Ubuntu Linux*, with a *Pentium 4, 3.00GHZ* processor. It reports the time, %cpu and %memory used for the simulation of blood coagulation with ntCC and HCC. We cannot compare the results with sCCP because, so far, this language is not equipped with an implemented interpreter-tool. As we can see in Tab. 1, our blood coagulation experiment is executed in less time with HCC.

We think that ntCC offers the best modeling environment: in general it is better in usability/features because we can model a set of important configurations lake as the resolution for the continue model (like the granularity), the possibility of set the time differential (Dt), the domain of each dynamic variable involved in the reactions, the max time of resolution (to generate the ordinary differential equations) and it is also possible to create a graph directly from this framework; As far as we know, the manual is not available for ntCC (while it is for HCC). In reverse, the time, the cpu and the memory in use for our running example in ntCC is greater then HCC, like as we can see to the report in Tab. 1.

6 Related Works

In [21] the authors suggest to model biomolecular process i.e. protein networks, by using the *pi-Calculus*. In the paper the author shows how the pi-calculus can be used to model biochemical networks as a mobile communication systems and the features of this process calculus for modeling various molecular systems, including transcriptional circuits, metabolic pathways, and signal transduction (ST) networks. In [20] we have a stochastic implementation of pi-Calculus for systems biology.

Biochemical Abstract Machine (*Biocham* [13]) is a software environment for modeling complex cell processes. It is characterized by distinct aspects: the analysis and simulation of boolean, kinetic and stochastic model and the simulation of biological proprieties in temporal logic.

Bio-PEPA [11] is a process algebra and is a modification of PEPA to deal with some features of biological models, such as stoichiometry and the use of generic kinetic laws. We have studied blood coagulation with this tool in a master thesis [9].

Even if these languages aren't based on CCP, they can help us to compare the experimental results obtained in the simulation because by having a different approach, they can be able to check small differences among the frameworks.

In [15] the authors introduce an approach to detecting inconsistencies in large biological networks by using Answer Set Programming; they use a non-monotonic handing of the store.

7 Conclusions and Future Works

The obtained in silico results have been produced by using the in vitro parameters in medical literature (i.e. the six reaction definitions, the stoichiometric coefficients and the factor's concentration). In vitro and in silico results are the same in terms of reduction of the thrombin formations.

These approaches can be used in a silico modeling for drug experimentations (*preclinic*) area; we aim to use our study for an in silico analysis of other biochemical reactions and phenomena beyond the blood coagulation.

We plan to extend our studies to new substances in pharmacological discovery (enzymes like as cytochrome or structure based prediction) and to compare our work with a a set of in vitro test, in order to better estimate the errors of in silico approach.

References

[1] Xri: Extended real interval. AVISPA Research Group, 2004.

[2] S. Avantsa. *Kinetic Analysis and Simulation of the In Vitro Blood Coagulation Mechanism*. PhD thesis, Texas Tech University, 1989.

[3] F. Benhamou, F. Goualard, L. Granvilliers, and J.F. Puget. Revising hull and box consistency. In *ICLP*, pages 230–244, 1999.

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

[5] L. Bortolussi. *Mathematical Modeling of Biological Systems*. PhD thesis, Universitá di Udine, 2007.

[6] L. Bortolussi, A. Dovier, and F. Fogolari. Agent-based protein structure prediction. *Multiagent and Grid Systems*, 3(2):183–197, 2007.

[7] L. Bortolussi, S. Fonda, and A. Policriti. Constraint-based simulation of biological systems described by molecular interaction maps. In *BIBM*, pages 288–293. IEEE Computer Society, 2007.

[8] L. Bortolussi and A. Policriti. Modeling biological systems in stochastic concurrent constraint programming. *Constraints*, 13(1-2):66–90, 2008.

[9] M. Bottalico. Modellazione in silico di pathway biologici: studio della blood coagulation tramite bio-pepa. Master's thesis, Università Degli Studi "G. d'Annunzio", Pescara, 2008. (In italian).

[10] M. Bottalico. Technical Report R-2009-001, Dipartimento di Scienze, Università "G. D'Annunzio" Chieti–Pescara, 2009.

[11] F. Ciocchetta and J. Hillston. Bio-pepa: An extension of the process algebra pepa for biochemical networks. *Electr. Notes Theor. Comput. Sci.*, 194(3):103–117, 2008.

[12] T. Delvin. *Texbook of biochemistry with clinical correlations*. McGraw Hill Book co., 2001.

[13] F. Fages. Temporal logic constraints in the biochemical abstract machine biocham. In P.M. Hill, editor, *LOPSTR*, volume 3901 of *LNCS*, pages 1–5. Springer, 2005.

[14] M. Falaschi, A. Policriti, and A. Villanueva. Modeling concurrent systems specified in a temporal concurrent constraint language-i. *Electr. Notes Theor. Comput. Sci.*, 48, 2001.

[15] M. Gebser, T. Schaub, S. Thiele, B. Usadel, and P. Veber. Detecting inconsistencies in large biological networks with answer set programming. In Maria Garcia de la Banda and Enrico Pontelli, editors, *ICLP*, volume 5366 of *Lecture Notes in Computer Science*, pages 130–144. Springer, 2008.

[16] D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions. In *The Journal of Physical Chemistry*, pages 2340–2352, 1977.

[17] V. Gupta, R. Jagadeesan, V.A. Saraswat, and D.G. Bobrow. Programming in hybrid constraint languages. In *Hybrid Systems*, pages 226–251, 1994.

[18] J. Gutiérrez, J.A. Pérez, C. Rueda, and F.D. Valencia. Timed concurrent constraint programming for analysing biological systems. *Electr. Notes Theor. Comput. Sci.*, 171(2):117–137, 2007.

[19] B.K. Katzung. *Farmacologia generale e clinica*. Piccin, 2003.

[20] C. Kuttler, C. Lhoussaine, and J. Niehren. A stochastic pi calculus for concurrent objects. In H. Anai, K. Horimoto, and T. Kutsia, editors, *AB*, volume 4545 of *Lecture Notes in Computer Science*, pages 232–246. Springer, 2007.

[21] A. Regev, W. Silverman, and E.Y. Shapiro. Representation and simulation of biochemical processes using the pi-calculus process algebra. In *Proc. Pacific Symposium on Biocomputing*, pages 459–470, 2001.

[22] V.A. Saraswat. *Concurrent Constraint Programming Languages*. The MIT Press, Cambridge, MA, 1993.

[23] V.A. Saraswat, R. Jagadeesan, and V. Gupta. Foundations of timed concurrent constraint programming. In *LICS*, pages 71–80. IEEE Computer Society, 1994.

[24] V.A. Saraswat, R. Jagadeesan, and V. Gupta. Timed default concurrent constraint programming. *J. Symb. Comput.*, 22(5/6):475–520, 1996.

[25] F.D. Valencia. Reactive constraint programming. Bric report, Department of Computer Science University of Aarhus, Denmark, 2000.

[26] A. Villanueva. *Model Checking for the Concurrent Constraint Paradigm*. PhD thesis, Università di Udine, Udine, Italy, May 2003.

[27] J. Williams. *Williams Hematology*. McGraw Hill Book co., 2006.