A Logic of Behaviour in Context and the Continuous π-calculus

Chris Banks and Ian Stark

lfcs | Laboratory for Foundations of Computer Science

THE UNIVERSITY of EDINBURGH informatics

INRIA Paris, July 2012

Introduction

In this talk I will

- show how we are applying formal computational techniques to systems biology, using
  - process calculi
  - modal/temporal logic
  - model checking
- present a language for modelling biochemical processes
- and a *logic for querying and classifying model behaviour in context*.

Continuous π-calculus (cπ)

A name-passing, continuous time, continuous state-space process algebra for modelling behaviour and variation in molecular systems.

Based on Milner's π-calculus and sharing an approach common to process algebras for biomodelling (after Regev et al.), some features are distinctive. For example, by comparison with the stochastic π-calculus:

- ODEs are the primary mode of execution, not stochastic simulation
- Continuous concentrations of chemicals replace discrete individuals
- End-to-end channels are replaced by multiple competing names


Marek Kwiatkowski and Ian Stark.
On Executable Models of Molecular Evolution. In *Proc. 8th International Workshop on Computational Systems Biology WCSB 2011*, pp. 105–108.


Marek Kwiatkowski.
A Formal Computational Framework for the Study of Molecular Evolution
PhD Dissertation, University of Edinburgh, December 2010.

Basics of cπ

cπ has two levels of system description:

- Species
  - Individual molecules (proteins)
  - Transition system semantics
- Processes
  - Bulk population (concentration)
  - Differential equations

- *Process space* is a real-valued vector space over species
  - each point in this space is a possible state of the system
  - the behaviour of a system with some initial values is a trajectory through the space.

Biomodelling in cπ

For individual species, cπ uses a modelling idiom based on that of Regev and Shapiro:

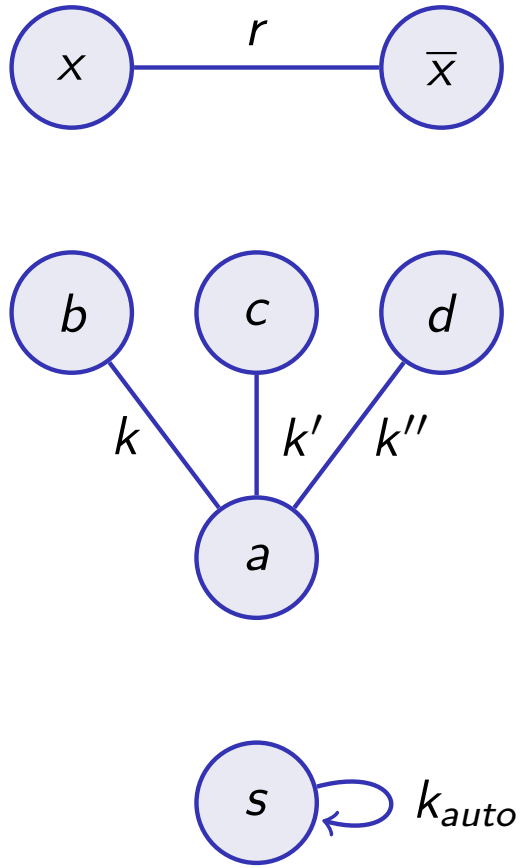
- Reagent-centric rather than rule-based
- Individual species are represented by processes
- Complexes are modelled by name restriction $(\nu x)(A|B)$
- Interaction is modelled by communication between names
- ...but with competition between multiple alternatives.

Names in $c\pi$

As in standard π-calculus, *names* indicate a potential for interaction: for example, the docking sites on an enzyme where other molecules may attach.

These sites may interact with many different other sites, to different degrees.

This variation is captured by an *affinity network*: a graph setting out the interaction potential between different names.



$c\pi$ syntax

Species syntax

$$\begin{aligned} \text{(Species) } A, B &::= 0 \mid \sum_{i=0}^{n} \pi_i.S_i \mid A|B \mid (\nu M)A \\ \text{(Prefix) } \pi &::= a(\vec{x};\vec{y}) \mid \tau @ k \\ S &::= D(\vec{a}) \mid A \end{aligned}$$

Process syntax

$$\begin{aligned} \text{(Process) } P, Q &::= c \cdot A \mid P||Q \\ &\quad \text{where } c \in \mathbb{R} \text{ is the species concentration} \\ \text{(Affinity Network) } M &::= \text{a labelled undirected graph: } a \overset{k}{-} b \\ &\quad \text{where } a, b \text{ are sites and } k \in \mathbb{R} \text{ is a reaction rate} \end{aligned}$$

$c\pi$ example

Enzyme catalysed reaction

$$S + E \rightleftarrows ES \rightarrow E + P$$

$c\pi$ example

Enzyme catalysed reaction

$$S + E \rightleftarrows ES \rightarrow E + P$$

$$E \triangleq (\nu\{{}_x\angle^u_r\})e\langle u, r\rangle.x.E$$

$$S \triangleq s(u, r).(u.S + r.P)$$

$c\pi$ example

Enzyme catalysed reaction

$$S + E \rightleftarrows ES \rightarrow E + P$$

$$E \triangleq (\nu\{{}_x\angle^u_r\})e\langle u, r\rangle.x.E$$

$$S \triangleq s(u, r).(u.S + r.P)$$

$$ES \equiv (\nu\{{}_x\angle^u_r\})(x.E|(u.S + r.P))$$

$c\pi$ example

Enzyme catalysed reaction

$$S + E \rightleftarrows ES \rightarrow E + P$$

$$E \triangleq (\nu\{_x\angle^u_r\})e\langle u, r\rangle.x.E$$

$$S \triangleq s(u, r).(u.S + r.P)$$

$$P \triangleq 0$$

$$\Pi \triangleq c_s \cdot S(s)||c_e \cdot E(e)||c_p \cdot P()$$

$$Aff = \{s - e\}$$

cπ example

Syntax

Compiler

ODEs

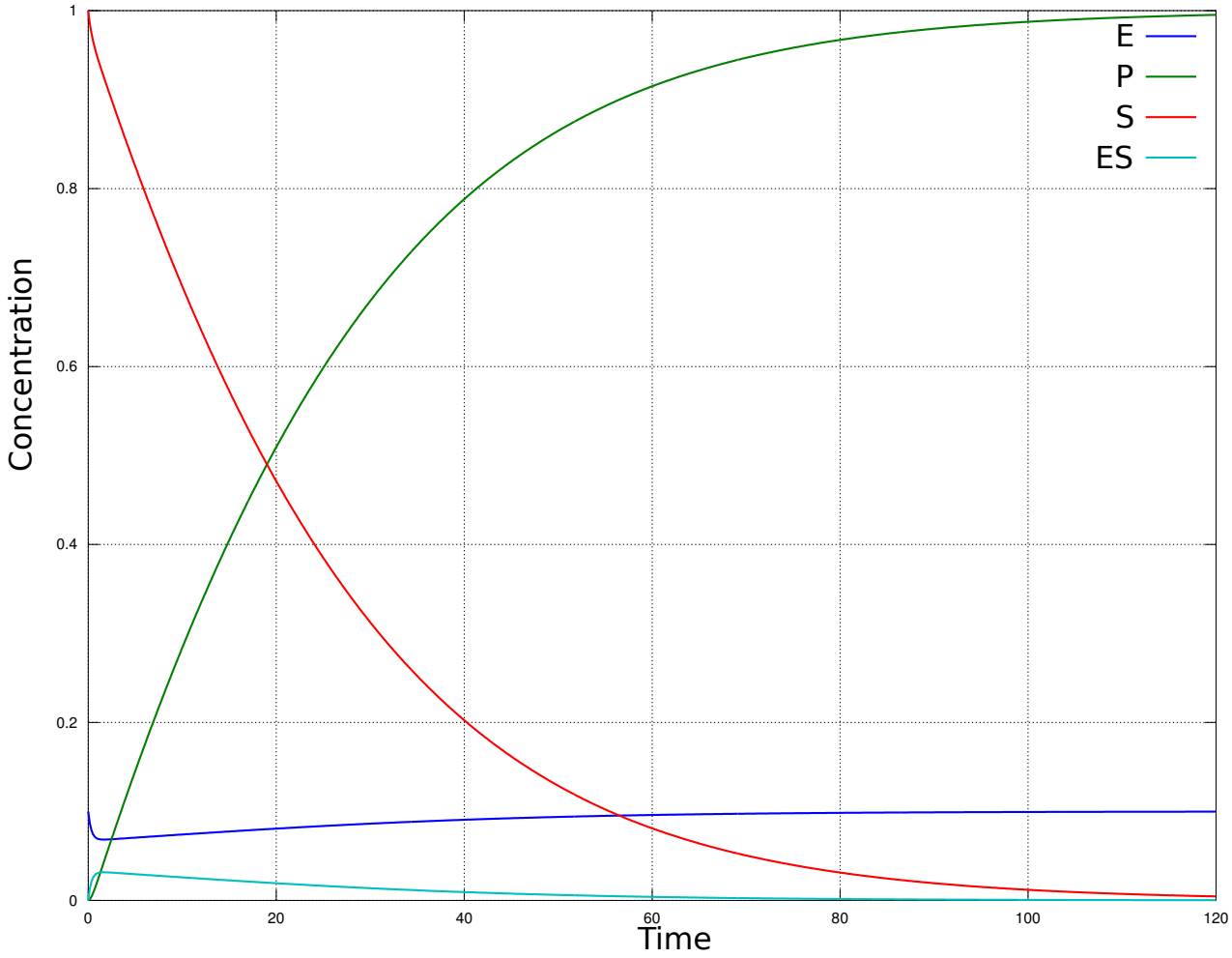
Solver

Time Series

Plotter

Plots

cπ example



$c\pi$ features

- Language reflects the structure of biochemical processes:
  - Prefix actions – reaction sites
  - Affinity network – complex reaction capabilities
  - Restriction – internal capabilities of complexes
- Compositional semantics – easier to construct large models from components
- Compilation to ODEs for standard analysis
- Computationally less expensive for large models than other approaches
  - than e.g. approaches using stochastic simulation or exhaustive state-space exploration

A logic for $c\pi$

We have a model with some dynamic behaviour. Now we wish to classify this behaviour:

- Requirements?
  - Temporal logic
  - Real valued constraints
  - Contextual properties
- $LTL(\mathbb{R})$
  - LTL is sufficient for deterministic processes
  - $LTL(\mathbb{R})$ for real valued constraints on species concentrations

A logic for $c\pi$

We have a model with some dynamic behaviour. Now we wish to classify this behaviour:

- Requirements?
  - Temporal logic
  - Real valued constraints
  - Contextual properties
- $LTL(\mathbb{R})$
  - LTL is sufficient for deterministic processes
  - $LTL(\mathbb{R})$ for real valued constraints on species concentrations
- Guarantee operator from spatial logic
  - $\phi \triangleright \psi$
    - $P \models \phi \triangleright \psi \iff \forall Q.Q \models \phi \implies P||Q \models \psi$

A logic for $c\pi$

We have a model with some dynamic behaviour. Now we wish to classify this behaviour:

- Requirements?
  - Temporal logic
  - Real valued constraints
  - Contextual properties
- $LTL(\mathbb{R})$
  - LTL is sufficient for deterministic processes
  - $LTL(\mathbb{R})$ for real valued constraints on species concentrations
- Guarantee operator from spatial logic
  - $\phi \triangleright \psi$
    - $P \models \phi \triangleright \psi \iff \forall Q.Q \models \phi \implies P||Q \models \psi$
- Context introduction operator:
  - $Q \triangleright \psi$
    - $P \models Q \triangleright \psi \iff Q||P \models \psi$

Logic syntax

A logic of behaviour in context:

$$
\begin{aligned}
\text{(Formula) } \phi, \psi \quad &::= \quad \mathit{Atom} \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \Rightarrow \psi \mid \neg\phi \mid \\
&\qquad \phi \mathbf{U}_t \psi \mid \mathbf{F}_t \phi \mid \mathbf{G}_t \phi \mid Q \triangleright \phi \\
\mathit{Atom} \quad &::= \quad \top \mid \bot \mid \mathit{Val}\ \mathit{BOp}\ \mathit{Val} \\
\mathit{Val} \quad &::= \quad v \in \mathbb{R} \mid [A] \mid \mathit{Val}\ \mathit{AOp}\ \mathit{Val}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{BOp} \quad &::= \quad > \mid < \mid \geq \mid \leq \\
\mathit{AOp} \quad &::= \quad + \mid - \mid \times \mid \div
\end{aligned}
$$

- Where t is a sub-interval of $[0, \infty) \in \mathbb{R}$.
- Abbreviations $\mathbf{U}$, $\mathbf{F}$, $\mathbf{G}$ denote $\mathbf{U}_{[0,\infty)}$, $\mathbf{F}_{[0,\infty)}$, $\mathbf{G}_{[0,\infty)}$ respectively
- and $\mathbf{U}_x$, $\mathbf{F}_x$, $\mathbf{G}_x$ denote $\mathbf{U}_{[0,x]}$, $\mathbf{F}_{[0,x]}$, $\mathbf{G}_{[0,x]}$ respectively.

Logic examples

- $\mathbf{F}([A] \geq c)$
  - Eventually $[A] \geq c$.
- $\mathbf{G}([A] > 0)$
  - We always have some A.
- $\mathbf{F}_{24}([A] > 0)$
  - Within 24 we have some A.
- $\mathbf{G}_{[10,15]}([A] \leq 5)$
  - Between times 10 and 15 we always have $[A] \leq 5$.

Logic examples

- $Q \triangleright \mathbf{F}([A] \geq c)$
  - If we introduce Q then eventually $[A] \geq c$.
- $Q \triangleright \mathbf{F}([A] \geq c) \wedge \mathbf{G}([A] < c)$
  - If we introduce Q then $[A]$ rises above c, whereas without Q it doesn't.
- $\mathbf{F}(Q \triangleright \mathbf{F}([A] < c))$
  - If we introduce Q *at some point in time* then $[A]$ is eventually below c.
- $\mathbf{G}_{[0,5]}(Q \triangleright \mathbf{F}_{10}([A] \geq c))$
  - If we introduce Q *at any time* between 0 and 5 then within 10 we get at least c of A.

Logic semantics

For P, a cπ process:

$$
\begin{array}{lcl}
P \models \mathit{Atom} & \iff & \mathit{Atom} \text{ is a property of } P \\
P \models \phi \wedge \psi & \iff & P \models \phi \text{ and } P \models \psi \\
P \models \neg\phi & \iff & P \not\models \phi \\
P \models \phi \mathbf{U}_{[t_0,t_n]} \psi & \iff & \exists t_0 \leq t' \leq t_n . \big((P^{t'} \models \psi) \wedge (\forall t_0 \leq t'' < t' . P^{t''} \models \phi) \big) \\
P \models Q \triangleright \phi & \iff & (Q \parallel P) \models \phi
\end{array}
$$

- P^t is the process reached from process P after time t.
- Q is a cπ process with any new global affinity network.

$$\mathbf{F}_t \phi \equiv \top \mathbf{U}_t \phi$$
$$\mathbf{G}_t \phi \equiv \neg \mathbf{F}_t \neg \phi$$

Model checking

- A numerical ODE solver gives us a *Trace* of the form:

$$(t_0, \vec{c_0}), (t_1, \vec{c_1}), \ldots$$

- A naive model checking algorithm is then easily expressed as a functional program:

Naive model checker

```
check ::  Trace ↦ Formula ↦ Bool

check (t:ts) (atom)          =   valid atom t

check (t:ts) (φ ∧ ψ)         =   (check (t:ts) φ) AND (check (t:ts) ψ)
check (t:ts) (φ ∨ ψ)         =   (check (t:ts) φ) OR (check (t:ts) ψ)
check (t:ts) (¬φ)            =   NOT(check (t:ts) φ)

check (t:ts) (φ U_[t0,tn] ψ) =   if (t< t0) then check ts φ U_[t0,tn] ψ
                                 else (t ≤ tn) AND ((check (t:ts) ψ) OR
                                 ((check (t:ts) φ) AND (check (ts) (φ U_[t0,tn] ψ)))

check (t:ts) (Q ▷ φ)         =   check (solve (compose Q (proc (t:ts)))) φ

solve ::  Process ↦ Trace
compose ::  Process ↦ Process ↦ Process
proc ::  Trace ↦ Process
```

Improved model checking

- The naive algorithm has complexity $O(n^f)$ for trace length n and depth f of nested temporal operators.
  - We repeatedly re-compute sub-formulae along the trace,
  - e.g. in $\mathbf{G}(\mathbf{F}\phi)$ if ϕ isn't true until the end of the trace we have cost n^2.
- To improve on this we can make use of the dynamic programming approach used in:

Calzone, L., Chabrier-Rivier, N., Fages, F., Soliman, S.
Machine learning biochemical networks from temporal logic properties.
Transactions on Computational Systems Biology VI, 2006, pp. 68–94.

- where we traverse the trace only once, evaluating all sub-formulae simultaneously.

Dynamic programming algorithm

1. By post-order depth-first traversal of the formula we obtain sub-formulae ordered by dependency.
2. Reverse the ordering of the trace.
3. We traverse the reversed trace once, labelling each time-point with each sub-formula in order if it holds, according to the following rules:

Dynamic programming algorithm

- An atomic proposition holds if its constraint is satisfied.
- $\phi \wedge \psi$ holds if the time-point is already labelled with both ϕ and ψ.
- $\neg\phi$ holds if the time-point is not already labelled with ϕ.

Dynamic programming algorithm

- $\phi \mathbf{U}_{[t_0,t_n]} \psi$ holds if:
  - *either* the time is $\leq t_n$ and:
    - *either* the time-point is already labelled with ψ
    - *or* it is already labelled with ϕ and the previous time point was labelled with $\phi \mathbf{U}_{[t_0,t_n]} \psi$.
  - *or* the time is $< t_0$ and the previous time point is labelled with $\phi \mathbf{U}_{[t_0,t_n]} \psi$.

Dynamic programming algorithm

- $Q \triangleright \phi$ holds if the following procedure returns *true*:
  1. find the cπ process Π of this time point and find $Q||\Pi$,
  2. solve $Q||\Pi$ and apply the algorithm to this new trace and ϕ.

Hybrid algorithm

- The dynamic programming algorithm avoids unnecessary re-computation.
- However we lose the ability to *short-circuit* the checking, e.g. stop when we find a witness for $\mathbf{F}\phi$.
- By using a hybrid of the two algorithms we can have the best of both.
- The naive algorithm can be restructured as:
  - an outer recursion over the trace
  - and an inner recursion over the sub-formulae.
- Implementation in Haskell exploits any potential laziness and only computes over as much of the trace as we need.

Calls to the solver

- By far the greatest computational cost comes from solving ODEs.
- The key to efficient model checking is reducing the number of calls we have to make to the solver.
- Consider $\mathbf{G}(Q \triangleright \phi)$ – we must compute a new trace introducing Q at every time point.
  - Thankfully this is *embarrassingly parallel*.
- Context introduction itself doesn't increase complexity, as we can re-write, e.g.:
  - $(Q \triangleright \phi) \wedge (Q \triangleright \psi) \rightarrow Q \triangleright (\phi \wedge \psi)$
  - $Q \triangleright (Q' \triangleright \phi) \rightarrow (Q||Q') \triangleright \phi$
- We have $O(n^d)$ where d is the depth of temporal operators separated by context introductions – $\mathbf{G}(Q \triangleright (\mathbf{G}\phi))$ has $O(n^2)$

Limitations

- We assume that the ODE solver gives us:
  - a trace over sufficient time to verify the formula,
  - a sufficiently accurate trace to verify the formula.
- The first condition we can satisfy as our temporal modalities specify the times over which we need to check (assuming a closed interval).
- The second condition is harder – a comment about that later.

Tools

- cπ workbench – implementation in Haskell
  - cπ interactive interpreter
  - species transition system exploration
  - ODE output
  - ODE solvers
    - internal GSL solvers: explicit (RKf45) / implicit (BSimp)
    - external: interface to GNU Octave for LSODE
    - Jacobian computed symbolically by the tool.
  - graph plotting
  - logic and model checker (various algorithms)

Future work

- Formula rewriting LTL algorithms (Rosu & Havelund)
- Reduce calls to the numerical solver:
  - solver is the bottleneck
  - formula rewriting
  - can some class of formulae be solved symbolically?
- Parallelising calls to the solver.
- Counterexample generation.
- Continued work on tool support / (G)UI.
- How can we assume the trace is sufficient?
  - exact real ODE solvers? (Edalat & Krznaric)

Fin

- Acknowledgements:
  - Ian Stark (supervisor)
  - Marek Kwiatkowski (cπ)

- Bibliography:

Kwiatkowski, M., & Stark, I.
The continuous π-calculus: A process algebra for biochemical modelling.
In *Proc. Computational Methods in Systems Biology '08*, LNCS 5307, pp.103122. Springer-Verlag, 2008.

Kwiatkowski, M.
A formal computational framework for the study of molecular evolution
Ph.D. Thesis, University of Edinburgh, 2010.

Kwiatkowski, M., & Stark, I.
On Executable Models of Molecular Evolution.
In *Proc. WCSB'11*, pp.105-108, 2011