Stochastic Concurrent Constraint Programming

Luca Bortolussi

1Department of Mathematics and Computer Science
University of Udine, Italy.

QAPL 2006, Wien, 2nd April 2006
Outline

1. Introduction
   - Preliminaries

2. Syntax and Operational Semantic
   - Syntax and Rates
   - Configurations
   - Operational Semantic
   - Discrete and Continuous Time Observables

3. Examples
   - Random Walk
   - Modeling Biochemical Reactions
   - Modeling Gene Regulatory Networks
Outline

1. Introduction
   - Preliminaries

2. Syntax and Operational Semantic
   - Syntax and Rates
   - Configurations
   - Operational Semantic
   - Discrete and Continuous Time Observables

3. Examples
   - Random Walk
   - Modeling Biochemical Reactions
   - Modeling Gene Regulatory Networks
Concurrent Constraint Programming

Store as a valuation replaced by a constraint store, modeled by a constraint system (extension of cylindric algebra). Computations evolve monotonically on the constraint system.

Syntax of CCP

Program = Decl . A

D = ε | Decl . Decl | p(x) : − A

A = 0
tell(c) . A
ask(c1) . A1 + ask(c2) . A2
A1 | A2
∃ x A
p(x)

SOS for CCP

⟨tell(c) . A, d⟩ → ⟨A, d ⊔ c⟩
⟨ask(c1) . A1 + ask(c2) . A2, d⟩ → ⟨A1, d⟩ if d ⊩ c1
⟨A, d⟩ → p⟨A′, d′⟩
⟨A | B, d⟩ → ⟨A′ | B, d′⟩
⟨A, d ⊔ ∃ x c⟩ → ⟨B, d′⟩
⟨∃ x A, c⟩ → ⟨∃ x B, c ⊔ ∃ x c⟩
⟨p(y), c⟩ → ⟨Δx A, c⟩ if p(x) : − A ∈ Decl
Non-Determinism is replaced by **probabilistic branching**. Sum and parallel operator are weighted by **probability distributions**. Transition relation is labeled with the probability associated to it.

**Syntax of pCCP**

\[
\text{Program} = \text{Decl}.A \\
D = \varepsilon \mid \text{Decl}.\text{Decl} \mid p(x) : \neg A \\
A = \begin{array}{l}
0 \\
tell(c).A \\
\exists x A \\
p(x) \\
\sum_{i=1}^{k} q_i : \text{ask}(c_i).A_i \\
|_{i=1}^{k} q_i : A_i 
\end{array}
\]

**SOS of pCCP**

\[
\begin{align*}
\langle \text{tell}(c).A, d \rangle & \rightarrow_1 \langle A, d \sqcup c \rangle \\
\langle \sum_{i=1}^{k} q_i : \text{ask}(c_i).A_i, d \rangle & \rightarrow q_j \langle A_j, d \rangle \text{ if } d \vdash c_j \\
\langle A, d \rangle & \rightarrow_p \langle A', d' \rangle \\
\langle q_1 : A \mid_{i=2}^{k} q_i : B_i, d \rangle & \rightarrow_p q_1 \langle q_1 : A' \mid_{i=2}^{k} q_i : B_i, d' \rangle \\
\text{where } \tilde{q}_j & = q_j / \sum_{k \in \text{active}} q_k 
\end{align*}
\]
Outline

1. Introduction
   - Preliminaries

2. Syntax and Operational Semantic
   - Syntax and Rates
   - Configurations
   - Operational Semantic
   - Discrete and Continuous Time Observables

3. Examples
   - Random Walk
   - Modeling Biochemical Reactions
   - Modeling Gene Regulatory Networks
Syntax and Rates

Syntax of sCCP

Syntax of Stochastic CCP

\begin{align*}
\text{Program} &= \text{Decl}.A \\
D &= \varepsilon \mid \text{Decl}.\text{Decl} \mid p(x) : -A \\
A &= 0 \mid \text{tell}_\lambda(c).A \mid \text{ask}_\lambda(c).A \mid [p(x)]_\lambda \mid \\
& \quad \exists xA \mid (A_1 + A_2) \mid (A_1 \parallel A_2)
\end{align*}

Each basic instruction (tell, ask, procedure call) has a rate attached to it. Rates are positive real numbers.
Rates can be interpreted as priorities or as frequencies.

**Rates as Priorities**
- A rate can represent the *priority of execution* of a process.
- There is a *global scheduler* choosing probabilistically between active processes, according to their priority.
- *Discrete time evolution.*

**Rates as Frequencies**
- A rate can represent the *frequency* or *speed* of a process.
- The higher the speed, the higher the probability of seeing a certain process executed.
- *Continuous time evolution.*
Hiding has no rate attached. It is removed by substituting the hidden variable with a fresh free variable.

A configuration is a point in the space $\mathcal{P} \times \mathcal{C} \times \overline{\mathcal{V}}$, where:

- $\mathcal{P}$ is the space of processes (modulo a congruence relation $\equiv$);
- $\mathcal{C}$ is the constraint system;
- $\overline{\mathcal{V}}$ is the family of finite subsets of (fresh free) variables.
Operational Semantic

SOS

\[
\langle \text{tell}_\lambda (c).A, d, V \rangle \rightarrow_{(1,\lambda)} \langle A, d \uplus c, V \rangle
\]

\[
\langle \text{ask}_\lambda (c), d, V \rangle \rightarrow_{(1,\lambda)} \langle A, d, V \rangle \quad \text{if } d \vdash c
\]

\[
\langle [\rho(y)]_\lambda, d, V \rangle \rightarrow_{(1,\lambda)} \langle A[y/x], d, V \rangle \quad \text{if } \rho(x) : -A
\]

\[
\langle A_1, d, V \rangle \rightarrow_{(p,\lambda)} \langle A'_1, d', V \rangle
\]

\[
\langle A_1 + A_2, d, V \rangle \rightarrow_{(p',\lambda')} \langle A'_1, d', V \rangle
\]

\[
\langle A_1 \parallel A_2, d, V \rangle \rightarrow_{(p',\lambda')} \langle A'_1 \parallel A_2, d', V \rangle
\]

with \( p' = \frac{p^\lambda}{\lambda + \text{rate}(A_2, d)} \) and \( \lambda' = \lambda + \text{rate}(A_2, d) \).

rate returns the sum of rates of all active agents.
Examples

Example

\[
\langle \text{tell}_1(c), \top, \emptyset \rangle \rightarrow_{(1,1)} \langle 0, c, \emptyset \rangle
\]

\[
\langle \text{ask}_1(c).\text{tell}_{100}(d) \parallel \text{tell}_1(c), \top, \emptyset \rangle \\
\rightarrow_{(1,1)} \langle \text{ask}_1(c).\text{tell}_{100}(d'), c, \emptyset \rangle \\
\rightarrow_{(1,1)} \langle \text{tell}_{100}(d'), c, \emptyset \rangle \\
\rightarrow_{(1,100)} \langle 0, c \sqcup d, \emptyset \rangle.
\]

Example

\[
\langle \text{tell}_1(c) + \text{tell}_1(d), \top, \emptyset \rangle \rightarrow_{(0.5,2)} \langle 0, c, \emptyset \rangle
\]

\[
\langle \text{tell}_1(c) + \text{tell}_1(d), \top, \emptyset \rangle \rightarrow_{(0.5,2)} \langle 0, d, \emptyset \rangle
\]
Discrete and Continuous Time Observables

**Discrete time I/O observables**

\[ \mathcal{O}_d(\langle A, d \rangle) = \left\{ (d', p) \mid p = \text{Prob} \left( \langle A, d \rangle \rightarrow \langle 0, d' \rangle \right) \right\}. \]

**Continuous time I/O observables**

\[ \mathcal{O}_c(\langle A, d \rangle)(t) = \left\{ (d', p) \mid p = \text{Prob} \left( \langle A, d \rangle \rightarrow \langle 0, d' \rangle \right)(t) \right\}. \]

**Theorem**

\[ \lim_{t \to \infty} \mathcal{O}_c(\langle A, d \rangle)(t) = \mathcal{O}_d(\langle A, d \rangle). \]
Discrete and Continuous Time Observables

Discrete time I/O observables

\[ O_d(\langle A, d \rangle) = \{(d', p) \mid p = \text{Prob}(\langle A, d \rangle \rightarrow \langle 0, d' \rangle)\} \].

Continuous time I/O observables

\[ O_c(\langle A, d \rangle)(t) = \{(d', p) \mid p = \text{Prob}(\langle A, d \rangle \rightarrow \langle 0, d' \rangle)(t)\} \].

Theorem

\[ \lim_{t \to \infty} O_c(\langle A, d \rangle)(t) = O_d(\langle A, d \rangle). \]
Discrete and Continuous Time Observables

Discrete time I/O observables

\[ \mathcal{O}_d(\langle A, d \rangle) = \left\{ (d', p) \mid p = \text{Prob}(\langle A, d \rangle \rightarrow \langle 0, d' \rangle) \right\} . \]

Continuous time I/O observables

\[ \mathcal{O}_c(\langle A, d \rangle)(t) = \left\{ (d', p) \mid p = \text{Prob}(\langle A, d \rangle \rightarrow \langle 0, d' \rangle)(t) \right\} . \]

Theorem

\[ \lim_{t \to \infty} \mathcal{O}_c(\langle A, d \rangle)(t) = \mathcal{O}_d(\langle A, d \rangle). \]
The language has been implemented by writing a meta-interpreter in SICStus Prolog.

We can model random walk as a stochastic process increasing or diminishing of one unit the value of a variable $X$.

**random**($X$) :-

$$\exists Y \ ( \text{tell}_1(Y = X + 1) + \text{tell}_1(Y = X - 1) ) \text{.random}(Y)$$
The stochastic extension of Concurrent Constraint Programming can be used to model biological systems, similarly to $\pi$-calculus.

<table>
<thead>
<tr>
<th>$\pi$-calculus for system biology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Molecule</td>
</tr>
<tr>
<td>Interaction capability</td>
</tr>
<tr>
<td>Interaction</td>
</tr>
<tr>
<td>Modification</td>
</tr>
<tr>
<td>(of cellular components)</td>
</tr>
</tbody>
</table>
The stochastic extension of Concurrent Constraint Programming can be used to model biological systems, similarly to $\pi$-calculus.

### CCP for Biological Simulation

<table>
<thead>
<tr>
<th>Molecular (Type) / Reaction Modification</th>
<th>Process State change</th>
</tr>
</thead>
<tbody>
<tr>
<td>(of cellular components)</td>
<td>(state transition systems and memory)</td>
</tr>
<tr>
<td>Environment</td>
<td>Constraint Store</td>
</tr>
<tr>
<td>Interaction with Environment</td>
<td>Asynchronous Communication</td>
</tr>
<tr>
<td>Direct Interaction capability</td>
<td>Channel</td>
</tr>
<tr>
<td>Interaction</td>
<td>Synchronous Communication</td>
</tr>
</tbody>
</table>

We need to extend the concept of rate: a rate needs to be a function $\lambda : C \rightarrow \mathbb{R}^+$. 
A simple reaction: $H + Cl \rightleftharpoons HCl$

**π-calculus**

We model atom $H$ and atom $Cl$. Reaction happens by a synchronous communication of these two processes. We need several copy of these processes.

Covalent Bonding: $H + Cl \rightleftharpoons HCl$

- $H$ has a private electron $e$.
- $H$ can share its electron with $Cl$ to form $HCL$, with $rate(share) = 100s^{-1}$
- $HCl$ can break its private bond, with $rate(e) = 10s^{-1}$
A simple reaction: $H + Cl \rightleftharpoons HCl$

We write a reaction agent, while the reagents and the product are modeled in the constraint store (put down in the environment). Independent on the number of agents.

```
reaction(H, CL, HCL) :-
    (tell_{shareRate}(H, CL) (share(H, CL, HCL))) +
    tell_{relRate}(H, CL) (rel(H, CL, HCL))
).reaction(H, CL, HCL)
```
Modeling Biochemical Reactions

A simple reaction: $\text{H} + \text{Cl} \rightleftharpoons \text{HCl}$

**sCCP**

We write a reaction agent, while the reagents and the product are modeled in the constraint store (put down in the environment). Independent on the number of agents.
Another reaction: $\text{Na} + \text{Cl} \iff \text{Na}^+ + \text{Cl}^-$

\[
\text{ionization}(Na, Cl, Na^+, Cl^-) \ :- \\
( \text{tell}_{\text{ionizeRate}}(Na^+, Cl^-) (\text{ionize}(Na, Cl, Na^+, Cl^-)) + \\
\text{tell}_{\text{deionizeRate}}(Na^+, Cl^-) (\text{deionize}(Na, Cl, Na^+, Cl^-)) ) ....\text{ionization}(Na, Cl, Na^+, Cl^-)
\]
The gene machine

DNA → messenger RNA → PROTEIN → SYSTEMS

regulation  transcription  translation  interaction
The instruction set

degradator(X) :- tell\text{\textunderscore degRate}(X)(\text{degrade}(X))\text{.degradator}(X)

null(X) :- tell\text{\textunderscore prodRate}(X)(\text{produce}(X))\text{.null}(X)

pos(X, Y) :- (tell\text{\textunderscore prodRate}(X)(\text{produce}(X))
                       + ask\text{\textunderscore enhanceRate}(Y)(Y > 0).tell\text{\textunderscore enhProdRate}(X)(\text{produce}(X))
                     ).pos(X, Y)

neg(X, Y) :- (tell\text{\textunderscore prodRate}(X)(\text{produce}(X))
                       + ask\text{\textunderscore inhibitRate}(Y)(Y > 0).ask\text{\textunderscore delayRate}(X)(\text{true})
                     ).neg(X, Y)

null\text{\textunderscore gate}(X) :- null(X) \parallel \text{degradator}(X)

pos\text{\textunderscore gate}(X, Y) :- pos(X, Y) \parallel \text{degradator}(X)

neg\text{\textunderscore gate}(X, Y) :- neg(X, Y) \parallel \text{degradator}(X)

Null Gate

null$\rightarrow$ $b$

neg_gate(A)
Pos Gate

pos_gate(a, b)

Modeling Gene Regulatory Networks
Neg Gate

$$\text{neg}_\text{gate}(A, B)$$
Modeling Gene Regulatory Networks

Self Repression

$$\text{neg} \quad a$$

$$\text{neg}_\text{gate}(A, A)$$

Graph showing self regulating neg gate in sCCP
Mutual Repression

\[ \text{neg\_gate}(A, B) \parallel \text{neg\_gate}(B, A) \]
We have introduced a stochastic version of CCP. With the same syntax and SOS, we can have a discrete model of time (DTMC - probabilistic computation) or a continuous model of time (CTMC).

We introduced the concept of observables both for discrete and continuous time, and compared them.

We argued that sCCP may be used for modeling biological systems, via examples.
The End

THANKS FOR THE ATTENTION!

QUESTIONS?