The reflexive CHAM and the join-calculus

Jason Reed

October 17, 2001

The CHAM

The **Ch**emical **A**bstract **M**achine can be thought of as a *model of computation* for concurrent calculi, for instance the π-calculus.

> *"Intuitively, the state of a system is like a* chemical solution *in which the floating* molecules *can interact with each other according to* reaction rules*; a* magical mechanism *stirs the solution, allowing for possible contacts between molecules"*

- The reaction rules are specified before any computation begins. They form a signature of sorts of what calculus we are dealing with. For instance, for the π-calculus, we would have a rule that would specify that reading molecules and writing molecules can synchronize and pass information.
- We can think of the rules as *catalysts* of reactions, and also as *places* where molecules must travel to react.

Problems with the CHAM

If we must specify one, fixed set of reaction rules at the outset, then

- All communication is constrained to this fixed set of 'reaction sites'. (Fournet and Gonthier say "Catalysts are bottlenecks")
- All the expressivity of pattern matching we want to have in the core calculus must be present in this fixed set of rules, which means that actually matching patterns against molecules in the CHAM may become complicated. (Fournet and Gonthier say "Catalysts clog up")

The reflexive CHAM

We solve both of these problems by permitting molecules to add new reactions to their environment.

Example:

$$D ::= \mathsf{ready}\langle printer\rangle|\mathsf{job}\langle file\rangle \Rrightarrow printer\langle file\rangle$$

$$\cdot \vdash \textbf{def}\, D\, \textbf{in}\, \mathsf{ready}\langle\mathsf{laser}\rangle|\mathsf{job}\langle 1\rangle|\mathsf{job}\langle 2\rangle$$

$$\rightleftharpoons D \vdash \mathsf{ready}\langle\mathsf{laser}\rangle|\mathsf{job}\langle 1\rangle|\mathsf{job}\langle 2\rangle$$

$$\rightleftharpoons D \vdash \mathsf{ready}\langle\mathsf{laser}\rangle, \mathsf{job}\langle 1\rangle, \mathsf{job}\langle 2\rangle$$

$$\rightleftharpoons D \vdash \mathsf{ready}\langle\mathsf{laser}\rangle|\mathsf{job}\langle 1\rangle, \mathsf{job}\langle 2\rangle$$

$$\rightarrow D \vdash \mathsf{laser}\langle 1\rangle, \mathsf{job}\langle 2\rangle$$

We'll see later the benefits of this style of model.

The reflexive chemistry (1)

But first, a formal definition:

$$
\begin{array}{lcl}
\text{Processes } P & ::= & x\langle\vec{v}\rangle \\
 & & \mathbf{def}\, D \,\mathbf{in}\, P \\
 & & P_1|P_2 \\
\text{Join patterns } J & ::= & x\langle\vec{v}\rangle \\
 & & J_1|J_2 \\
\text{Definitions } D & ::= & J \Rrightarrow P \\
 & & D \wedge D
\end{array}
$$

The reflexive chemistry (2)

$$
\begin{array}{rcl}
rv(x\langle\vec{v}\rangle) & := & \vec{v} \\
rv(J_1|J_2) & := & rv(J_1) \uplus rv(J_2)
\end{array}
$$

$$
\begin{array}{rcl}
dv(x\langle\vec{v}\rangle) & := & \{x\} \\
dv(J_1|J_2) & := & dv(J_1) \cup dv(J_2) \\
dv(J \Rrightarrow P) & := & dv(J) \\
dv(D_1 \wedge D_2) & := & dv(D_1) \cup dv(D_2)
\end{array}
$$

The reflexive chemistry (3)

$$
\begin{aligned}
fv(J \Rrightarrow P) &:= dv(J) \cup (fv(P) - rv(J)) \\
fv(D_1 \wedge D_2) &:= fv(D_1) \cup fv(D_2)
\end{aligned}
$$

$$
\begin{aligned}
fv(x\langle\vec{v}\rangle) &:= \{x\} \cup \vec{v} \\
fv(\mathbf{def}\, D \,\mathbf{in}\, P) &:= (fv(P) \cup fv(D)) - dv(D) \\
fv(P_1 | P_2) &:= fv(P_1) \cup fv(P_2)
\end{aligned}
$$

Operational Semantics (1)

All rules operate on *higher-order solutions* $\mathcal{R} \vdash \mathcal{M}$ which consist of a multiset $\mathcal{M}$ of processes ("molecules") on the right and a multiset $\mathcal{R}$ of definitions ("reactions") on the left. There are reversible structural "heating/cooling" rules (reversibility is denoted by $\rightleftharpoons$) and one irreversible *reaction* rule (red).

Operational Semantics (2)

$$
\begin{array}{llcl}
\text{(str-join)} & \mathcal{R} \vdash \mathcal{M}, P|Q & \rightleftharpoons & \mathcal{R} \vdash \mathcal{M}, P, Q \\
\text{(str-and)} & \mathcal{R}, D \wedge E \vdash \mathcal{M} & \rightleftharpoons & \mathcal{R}, D, E \vdash \mathcal{M} \\
\text{(str-def)} & \mathcal{R} \vdash \mathcal{M}, \mathbf{def}\, D \,\mathbf{in}\, P & \rightleftharpoons & \mathcal{R}, D\sigma_{dv} \vdash \mathcal{M}, P\sigma_{dv} \\
\text{(red)} & \mathcal{R}, J \Rrightarrow P \vdash \mathcal{M}, J\sigma_{rv} & \rightarrow & \mathcal{R}, J \Rrightarrow P \vdash \mathcal{M}, P\sigma_{rv}
\end{array}
$$

Side conditions:

- $\mathrm{dom}(\sigma_{dv}) \subseteq dv(D)$ and substitutes for these variables distinct, fresh names. Fresh means $\mathrm{rng}(\sigma_{dv}) \cap (fv(\mathcal{R}) \cup fv(\mathcal{M}) \cup fv(\mathbf{def}\, D \,\mathbf{in}\, P)) = \emptyset$.
- $\mathrm{dom}(\sigma_{rv}) \subseteq rv(J)$.

The Join Calculus (1)

We can think of the CHAM as *computational model* derived from *process calculi* like the π-calculus. We can go in the opposite direction and produce a process calculus, the *join calculus*, from the reflective CHAM. The terms of the join calculus are just the *molecules* of the reflexive CHAM, and the structural equivalence and transition rules correspond to the reaction rules of the CHAM.

However, we're going to look at a smaller version of the full join calculus, the *core join calculus*. It turns out we lose no expressive power.

The Join Calculus (2)

$$P ::= x\langle u\rangle \mid (P_1|P_2) \mid (\mathbf{def}\, x\langle u\rangle|y\langle v\rangle \Rrightarrow P_1 \,\mathbf{in}\, P_2)$$

$$\begin{array}{rcl}
P|Q & \equiv & Q|P \\
P|(Q|R) & \equiv & (P|Q)|R \\
P|\,\mathbf{def}\, D \,\mathbf{in}\, Q & \equiv & \mathbf{def}\, D \,\mathbf{in} (P|Q) \\
\mathbf{def}\, D \,\mathbf{in}\, \mathbf{def}\, D' \,\mathbf{in}\, P & \equiv & \mathbf{def}\, D' \,\mathbf{in}\, \mathbf{def}\, D \,\mathbf{in}\, P \\
P \equiv_\alpha P & \Rightarrow & P \equiv P \\
P \equiv Q & \Rightarrow & P|R \equiv Q|R \\
R \equiv S, P \equiv Q & \Rightarrow & \mathbf{def}\, J \Rrightarrow R \,\mathbf{in}\, P \equiv \mathbf{def}\, J \Rrightarrow S \,\mathbf{in}\, Q
\end{array}$$

The Labelled Transition System

We define a labelled transition relation $\xrightarrow{\delta}$ where δ ranges over $D \cup \{\tau\}$. The relation is the smallest such that

- For all $D = x\langle u\rangle | y\langle v\rangle \Rrightarrow R$, we have $x\langle s\rangle | y\langle t\rangle \xrightarrow{D} R[s/x, t/y]$.
- If $P \xrightarrow{\delta} P'$, then
  - $P|Q \xrightarrow{\delta} P'|Q$
  - $\mathbf{def}\, D \,\mathbf{in}\, P \xrightarrow{\delta} \mathbf{def}\, D \,\mathbf{in}\, P'$ (if $fv(D) \cap dv(\delta) = \emptyset$)
  - $\mathbf{def}\, \delta \,\mathbf{in}\, P \xrightarrow{\tau} \mathbf{def}\, \delta \,\mathbf{in}\, P'$ (if $\delta \neq \tau$)
  - $Q \xrightarrow{\delta} Q'$ (if $P \equiv Q$ and $P' \equiv Q'$)

Relating the Reflexive CHAM to the Join Calculus

Lemma 1.1 *The structural congruence* $\equiv$ *is the smallest congruence that contains all pairs of processes* P, Q *such that* $\vdash P \rightleftharpoons^* \vdash Q$. *The silent transition relation* $\xrightarrow{\tau}$ *contains exactly the pairs of processes* P, Q *up to* $\equiv$ *such that* $\vdash P \rightarrow \vdash Q$.

Programming in the Join Calculus

Everyone's favorite example; the ref cell.

$$\mathbf{def}\, mkcell\langle v_0, \kappa_0\rangle \Rrightarrow \begin{pmatrix} \mathbf{def}\, get\langle\kappa\rangle | s\langle v\rangle \Rrightarrow \kappa\langle v\rangle | s\langle v\rangle \\ \wedge\ set\langle u, \kappa\rangle | s\langle v\rangle \Rrightarrow \kappa\langle\rangle | s\langle u\rangle \\ \mathbf{in}\, set\langle v_0\rangle | \kappa_0\langle get, set\rangle \end{pmatrix} \mathbf{in} \ldots$$

Intuitively $mkcell\langle v_0, \kappa_0\rangle \approx \mathbf{let}\, x = \mathbf{ref}\, v_0\, \mathbf{in}\, \kappa_0$ except that instead of a process κ_0 with a free x in it, we have a 'message' on channel κ_0 containing the get and set operations. We can make use of these operations (that is, 'read on the channel') by making another definition, say $\mathbf{def}\, \nu\langle g, s\rangle \Rrightarrow P$ inside the $\ldots$ above, and then invoking $mkcell\langle v_0, \nu\rangle$. The variables g and s are bound in P, and they will be instantiated with the fresh *get* and *set* operations when the reaction rule of *mkcell* is invoked.

Encoding the λ-calculus (1)

Everyone's other favorite example.

Call-by-name:

$$
\begin{array}{rcl}
[\![x]\!]_v & := & x\langle v\rangle \\
[\![\lambda x.T]\!]_v & := & \mathbf{def}\,\kappa\langle x,w\rangle \Rrightarrow [\![T]\!]_w\,\mathbf{in}\,v\langle\kappa\rangle \\
[\![T\ U]\!]_v & := & \mathbf{def}\,x\langle u\rangle \Rrightarrow [\![U]\!]_u\,\mathbf{in} \\
 & & \ \mathbf{def}\,w\langle\kappa\rangle \Rrightarrow \kappa\langle x,v\rangle\,\mathbf{in}\,[\![T]\!]_w
\end{array}
$$

The interpretation $[\![T]\!]_v$ means that T should send its *value* on channel v. A value is a channel κ which, if sent $\langle x,w\rangle$, uses x to look up an argument and sends the result of applying itself to the argument on w. To lookup the value of x, send z to x, and the value of x will be sent on z.

Encoding the λ-calculus (2)

Parallel Call-by-value:

$$
\begin{aligned}
[\![x]\!]_v &:= v\langle x\rangle \\
[\![\lambda x.T]\!]_v &:= \mathbf{def}\,\kappa\langle x,w\rangle \Rrightarrow [\![T]\!]_w\,\mathbf{in}\,v\langle\kappa\rangle \\
[\![T\ U]\!]_v &:= \mathbf{def}\,t\langle\kappa\rangle|u\langle w\rangle \Rrightarrow \kappa\langle w,v\rangle\,\mathbf{in}\,[\![T]\!]_t|[\![U]\!]_u
\end{aligned}
$$

This differs in that instead of sending a channel which serves up the unreduced argument, we send the actual argument in the 'application request'. We can do this in the definition of application because we wait until both of the terms in the application have converged to a value.

Comparison with the π-calculus

We can think of the join calculus is an asynchronous version of the π-calculus, except with the restrictions that

- All binding happens with one construct, the definition.
- *Synchronization* only happens with defined names; Processes cannot pass messages over free names. Even though one can *write* on a channel with a free name, communication only occurs when a defined rule executes.
- For every defined name, there is *exactly one* replicated read. If we think of the calculus in a distributed setting, this means that for every defined name, there is *exactly one place* where synchronization can occur for that name.

The Asynchronous π-calculus

$$P ::= (P|Q) \mid \mathbf{new}\, u\, \mathbf{in}\, P \mid \bar{x}\langle u\rangle \mid x(u).P \mid !x(u).P$$

Naïve Representation of the π-calculus

$$
\begin{aligned}
[\![P|Q]\!]_\pi &:= [\![P]\!]_\pi | [\![Q]\!]_\pi \\
[\![\mathbf{new}\, x \,\mathbf{in}\, P]\!]_\pi &:= \mathbf{def}\, x_o\langle v_o, v_i\rangle | x_i\langle \kappa\rangle \Rrightarrow \kappa\langle v_o, v_i\rangle \,\mathbf{in} [\![P]\!]_\pi \\
[\![\bar{x}\langle v\rangle]\!]_\pi &:= x_o\langle v_o, v_i\rangle \\
[\![x(v).P]\!]_\pi &:= \mathbf{def}\, \kappa\langle v_o, v_i\rangle \Rrightarrow [\![P]\!]_\pi \,\mathbf{in}\, x_i\langle \kappa\rangle \\
[\![!x(v).P]\!]_\pi &:= \mathbf{def}\, \kappa\langle v_o, v_i\rangle \Rrightarrow x_i\langle \kappa\rangle | [\![P]\!]_\pi \,\mathbf{in}\, x_i\langle \kappa\rangle
\end{aligned}
$$

No good!

For instance, $[\![\bar{x}\langle a\rangle|\bar{x}\langle b\rangle|x(u).\bar{y}\langle u\rangle]\!]_\pi$ can't make any transition, even though $\bar{x}\langle a\rangle|\bar{x}\langle b\rangle|x(u).\bar{y}\langle u\rangle$ can in the π-calculus. This is because we have no enclosing $\mathbf{new}\, x\, \mathbf{in} \ldots$ to provide the definition that allows synchronization to occur.

Additionally, this encoding is not *fully abstract* in the sense that it is not robust given an arbitrary concurrent context that the encoded process lives in. Even if we make sure all of our free names have been appropriately defined, a channel we are reading on might be written to by some malicious process, with the message being a free name. Again, if we try to do anything to that free name, we become stuck!

The Equator

A tool we will use to patch up this problem is the *equator*

$$M^{\pi}_{x,y} :=!x(u).\bar{y}\langle u\rangle | !y(u).\bar{x}\langle u\rangle$$

This is an asynchronous π-calculus program which conflates the two channels x and y. If $M^{\pi}_{x,y}$ is in the 'solution', then no process can tell the difference between them.

Firewalls

$$
\begin{array}{rcl}
\mathcal{P}_x[P] & := & \mathbf{def}\, x_\ell\langle v_o, v_i\rangle | x_i\langle \kappa\rangle \Rrightarrow \kappa\langle v_o, v_i\rangle \\
& & \mathbf{in}\, \mathbf{def}\, x_o\langle v_o, v_i\rangle \Rrightarrow p\langle v_o, v_i, x_\ell\rangle\, \mathbf{in}\, P \\
\mathcal{E}_x[P] & := & \mathcal{P}_x[x_e\langle x_o, x_i\rangle | P] \\
\mathcal{M}[P] & := & \mathbf{def}\, p\langle x_o, x_i, \kappa\rangle \Rrightarrow \mathcal{P}_y[\kappa\langle y_o, y_i\rangle | [\![M^\pi_{x,y}]\!]_\pi]\, \mathbf{in}\, P \\
\mathcal{E}[P] & := & \mathcal{M}[\mathcal{E}_{x_1}[\cdots \mathcal{E}_{x_n}[P]\cdots]] \qquad (\text{for } x_1, \ldots, x_n = fv(P))
\end{array}
$$

Full Abstraction Result

Theorem 1.2 $Q \approx_\pi R$ *iff* $\mathcal{E}[[\![Q]\!]_\pi] \approx \mathcal{E}[[\![R]\!]_\pi]$.

The Reverse Encoding

We can also embed the join calculus in the π-calculus. This isn't too surprising considering our claim that the join calculus is just the π-calculus with some restrictions on how things can communicate.

Naïve representation of the Join Calculus

$$
\begin{aligned}
[\![P|Q]\!]_j &:= [\![P]\!]_j \,|\, [\![Q]\!]_j \\
[\![x\langle v\rangle]\!]_j &:= \bar{x}\langle v\rangle \\
[\![\mathbf{def}\, x\langle u\rangle|y\langle v\rangle \Rrightarrow P \,\mathbf{in}\, Q]\!]_j &:= \mathbf{new}\, x, y\, \mathbf{in} \\
&\quad\;\; ((!x(u).y(v).[\![P]\!]_j)\,|\,[\![Q]\!]_j)
\end{aligned}
$$

Still no good!

There is the same problem as with the previous encoding; it's not robust up to arbitrary contexts. We pull the same sort of trick to fix things.

We define a *relay* process

$$R_{x,y} :=!x(v).\,\mathbf{new}\, v_e\, \mathbf{in}\, \bar{r}\langle v_e, v\rangle|\bar{y}\langle v_e\rangle$$

and firewall definitions

$$\begin{array}{rcl} \mathcal{R}[P] & := & \mathbf{new}\, r\, \mathbf{in}\ (!r(x, x_e).R_{x,x_e}|P) \\ \mathcal{E}^{\pi}_{x}[P] & := & \mathbf{new}\, x\, \mathbf{in}(R_{x,x_e}|P) \\ \mathcal{E}^{\pi}[P] & := & \mathcal{R}[\mathcal{E}^{\pi}_{x_1}[\cdots \mathcal{E}^{\pi}_{x_n}[P]\cdots]] \qquad (\text{for } x_1, \ldots, x_n = fv(P)) \end{array}$$

Full Abstraction Result

Theorem 1.3 $P \approx Q$ *iff* $\mathcal{E}^{\pi}[[\![P]\!]_j] \approx_{\pi} \mathcal{E}^{\pi}[[\![Q]\!]_j]$.

Join Calculus vs. the world

Fournet and Gonthier seem to believe that there are serious problems with some of the calculi we've seen. For instance,

- Synchronization is *non-local*. Reads and writes must join up somehow, but we are potentially forced to search the whole universe for a and $\bar{a}$ to synchronize.
- Either there are a few simple primitives, (π-calculus) or else many primitives hardcoded into the language, for communication, choice, locations, agents, etc. With few we may lose the ability to naturally express certain constructions, but with many the language becomes complicated to reason about.

The Join Calculus Way

- *Locality* comes from replacing synchronization

$$a(x).P|\bar{a}\langle v\rangle.Q$$

  in the π-calculus style synchronization with the more general notion of *join patterns*. This means that all forms of communication (reactions) have a unique place (catalyst) where they occur.

- *Reflection* comes from allowing processes to define new reaction rules. Pattern matching is a very expressive yet simple way of encoding synchronization constraints.

But is it really so great?

We do have embeddings going both ways, so we're still left with the problem of subjectively deciding which formulation is more natural.

If we try to examine the DRCHAM rules, the idea of definitions *qua* locations seems elegantly used in the rule

$$\textbf{comm} \vdash x\langle\vec{v}\rangle \parallel J \Rrightarrow P \vdash \cdot \longrightarrow \cdot \vdash \cdot \parallel J \Rrightarrow P \vdash x\langle\vec{v}\rangle$$

but there are a whole host of other mobility and location primitives, with many scoping rules. It doesn't seem that defined reaction rules capture everything we intuitively think of concerning locations, and they also don't appear to be general enough to permit the mobility primitives to be naturally definable.

Finally...

Is garbage collection still problematic? It seems so. Names defined at one site can still float around other sites, even though they cannot react there. (although they can *participate* as the arguments of reactions)