A Concurrent Object Calculus

Andy Gordon, Microsoft Research
Paul Hankin, Cambridge University

\[ \text{conc}_\zeta = \text{imp}_\zeta + \pi \]
Why is a solution of \( \text{concc}_\zeta = \text{imp}_\zeta + \pi \) interesting?

- A solution \( \text{concc}_\zeta \) could help describe aspects of the vast amount of software coded using objects and threads.
- The \( \text{imp}_\zeta \)-calculus enjoys lots of expressive type systems (supporting functions, classes, subtyping, inheritance, . . .). The \( \pi \)-calculus enjoys lots of powerful proof tools. Therefore, a \( \text{concc}_\zeta \)-calculus would enjoy both!
Syntax of (one form of) the imp\(\zeta\)-calculus

\[ u, v ::= \]
\[ x \quad \text{variable} \]
\[ p \quad \text{name (pointer to an object)} \]
\[ a, b, c ::= \]
\[ u \quad \text{result} \]
\[ [\ell_i = \zeta(x_i) b_i \quad i \in \{1 \ldots n\}] \quad \text{object} \]
\[ a.\ell \quad \text{select method} \]
\[ a.\ell \leftarrow \zeta(x) b \quad \text{update method} \]
\[ \text{clone}(a) \quad \text{clone object} \]
\[ \text{let } x = a \text{ in } b \quad \text{let} \]
An Example Computation in imp\(\zeta\)

Let the object \(d = [\ell = \zeta(x)x.\ell \leftarrow \zeta(y)x]\).

Running the expression \(d.\ell\) goes as follows:

- \(d.\ell\) stores the object \(d\) at a fresh location \(p\), and runs \(p.\ell\),

- which fetches the method \(\zeta(x)x.\ell \leftarrow \zeta(y)x\),
  and runs \(p.\ell \leftarrow \zeta(y)p\),

- which updates location \(p\) with the method \(\zeta(y)p\),
  and returns the result \(p\).
Semantics of \( \text{imp}\zeta \) in More Detail

- Abadi and Cardelli’s relation \( \sigma \cdot S \vdash a \leadsto v \cdot \sigma' \) is a big-step environment-based semantics.

- Gordon, Hankin, and Lassen’s relation \( \langle \sigma \parallel a \rangle \rightarrow \langle \sigma' \parallel a' \rangle \) is an equivalent small-step substitution-based semantics, where
  - \( \langle \sigma \parallel a \rangle \) and \( \langle \sigma' \parallel a' \rangle \) are configurations, and
  - \( \sigma \) and \( \sigma' \) are stores of the form \( p_1 \leftarrow d_1, \ldots, p_n \leftarrow d_n \) where each \( d_i \) is an object.
Small-step Reductions of the Earlier Example

To illustrate Gordon, Hankin, and Lassen’s semantics:

\[
\langle \emptyset \parallel [\ell = \zeta(x)x.\ell \leftarrow \zeta(y)x].\ell \rangle \\
\rightarrow \langle p \mapsto [\ell = \zeta(x)x.\ell \leftarrow \zeta(y)x] \parallel p.\ell \rangle \\
\rightarrow \langle p \mapsto [\ell = \zeta(x)x.\ell \leftarrow \zeta(y)x] \parallel p.\ell \leftarrow \zeta(y)p \rangle \\
\rightarrow \langle p \mapsto [\ell = \zeta(y)p] \parallel p \rangle
\]

where \( p \) is any fresh name.
We seek a concurrent extension of the imp$\zeta$-calculus whose semantics has the simplicity and elegance of the Berry/Boudol/Milner chemical semantics of $\pi$.

Moreover, we seek to avoid:

- Auxiliary notions of stores, threads, configurations, evaluation contexts, labelled transitions
- Continuation-passing encodings of expressions
- Encodings of objects (All the known typed encodings of the $\zeta$-calculus into the $\lambda$-calculus or the $\pi$-calculus are fairly complex.)
Our Solution

\[
\text{conc}_\zeta \triangleq \text{imp}_\zeta + \left\{a \vdash b, (\forall p) a\right\} + \left\{p \mapsto [l_i = \zeta(x_i)b_i]\right\}
\]

- Syntax is a superset of the \text{imp}_\zeta-calculus
- Channels, and hence the \pi-calculus, may be encoded
- Like in CML semantics, \(a \vdash b\) is asymmetric
- Well-formedness conditions enforced by a type system
- A small-step substitution-based semantics: \(a \equiv b, a \rightarrow b\)
Syntax of the concζ-calculus

\[ \alpha, \beta, \gamma ::= \]

\[ u \]

\[ \rho \mapsto [\ell_i = \zeta(x_i) b_i \mid i \in 1..n] \]

\[ u.\ell \]

\[ u.\ell \leftarrow \zeta(x) b \]

\[ \text{clone}(u) \]

\[ \text{let } x := \alpha \text{ in } b \]

\[ \alpha \uparrow b \]

\[ (\forall \rho) \alpha \]

- terms
  - result (a variable \( x \) or a name \( \rho \))
  - object denomination
  - select method
  - update method
  - clone object
  - let
  - composition
  - restriction
Chemical Reductions of the Earlier Example

The following illustrates some conventions, structural congruences, and reductions of \texttt{conc}\(\zeta\):

\[
[l = \zeta(x)x.l \equiv \zeta(y)x].l
\]

\[
= \textit{let } z = [l = \zeta(x)x.l \equiv \zeta(y)x] \textit{ in } z.l
\]

\[
= \textit{let } z = (\forall p)(p \mapsto [l = \zeta(x)x.l \equiv \zeta(y)x] \uparrow p) \textit{ in } z.l
\]

\[
\equiv (\forall p)(p \mapsto [l = \zeta(x)x.l \equiv \zeta(y)x] \uparrow \textit{let } z = p \textit{ in } z.l)
\]

\[
\rightarrow (\forall p)(p \mapsto [l = \zeta(x)x.l \equiv \zeta(y)x] \uparrow p.l)
\]

\[
\rightarrow (\forall p)(p \mapsto [l = \zeta(x)x.l \equiv \zeta(y)x] \uparrow p.l \equiv \zeta(y)p)
\]

\[
\rightarrow (\forall p)(p \mapsto [l = \zeta(y)p] \uparrow p)
\]
Example of Interdependent Denominations

We may generate a cyclic dependency between denominations:

\[
\begin{align*}
\text{let } x_1 = [\ell = \zeta(y_1)x_1] \text{ in } & \rightarrow^* (\forall p_1)(\forall p_2)( \\
\text{let } x_2 = [\ell = \zeta(y_2)x_1] \text{ in } & p_1 \mapsto [\ell = \zeta(y_1)p_2] \triangleright \\
x_1.\ell \leftarrow \zeta(y_1)x_2 & p_2 \mapsto [\ell = \zeta(y_2)p_1] \triangleright p_1 
\end{align*}
\]

This illustrates the need in our calculus for name scoping (e.g., \((\forall p_1)\)) to be syntactically separate from denomination (e.g., \(p_1 \mapsto [\ell = \zeta(y_1)p_2]\)).
Semantics Rules of $\text{conc}_\zeta$

**Structural congruence** $a \equiv b$

\[
\begin{align*}
(a \overset{\nu}{\Rightarrow} b) \overset{\nu}{\Rightarrow} c & \equiv a \overset{\nu}{\Rightarrow} (b \overset{\nu}{\Rightarrow} c) \\
(a \overset{\nu}{\Rightarrow} b) \overset{\nu}{\Rightarrow} c & \equiv (b \overset{\nu}{\Rightarrow} a) \overset{\nu}{\Rightarrow} c \\
(\forall p)(\forall q)a & \equiv (\forall q)(\forall p)a \\
(\forall p)(a \overset{\nu}{\Rightarrow} b) & \equiv a \overset{\nu}{\Rightarrow} (\forall p)b \text{ if } p \notin \text{fn}(a) \\
(\forall p)(a \overset{\nu}{\Rightarrow} b) & \equiv ((\forall p)a) \overset{\nu}{\Rightarrow} b \text{ if } p \notin \text{fn}(b) \\
\text{let } x = (\text{let } y = a \text{ in } b) \text{ in } c & \equiv \text{let } y = a \text{ in (let } x = b \text{ in } c) \text{ if } y \notin \text{fv}(c) \\
(\forall p)\text{let } x = a \text{ in } b & \equiv \text{let } x = (\forall p)a \text{ in } b \text{ if } p \notin \text{fn}(b) \\
a \overset{\nu}{\Rightarrow} \text{let } x = b \text{ in } c & \equiv \text{let } x = (a \overset{\nu}{\Rightarrow} b) \text{ in } c
\end{align*}
\]
**Reduction** \( \alpha \to \beta \): basic reduction rules

For the first three rules, let \( \mathbf{d} = [\ell_i = \zeta(x_i) b_i \mid i \in 1..n] \).

\[
(p \mapsto \mathbf{d}) \widehat{\rightarrow} p.\ell_j \to (p \mapsto \mathbf{d}) \widehat{\rightarrow} b_j[x_j \leftarrow p] \text{ if } j \in 1..n
\]

\[
(p \mapsto \mathbf{d}) \widehat{\rightarrow} (p.\ell_j \leftarrow \zeta(x) b) \to (p \mapsto \mathbf{d}') \widehat{\rightarrow} p
\]

if \( j \in 1..n \), \( \mathbf{d}' = [\ell_j = \zeta(x) b, \ell_i = \zeta(x_i) b_i \mid i \in (1..n) \setminus \{j\}] \)

\[
(p \mapsto \mathbf{d}) \widehat{\rightarrow} \text{clone}(p) \to (p \mapsto \mathbf{d}) \widehat{\rightarrow} (\forall q)(q \mapsto d \widehat{\rightarrow} q) \text{ if } q \not\in \text{fn}(d)
\]

*let* \( x=p \) *in* \( b \to b[x \leftarrow p] \)

---

**Reduction** \( \alpha \to \beta \): congruence and structural rules

\[
(\forall p)\alpha \to (\forall p)\alpha' \text{ if } \alpha \to \alpha'
\]

\( a \widehat{\rightarrow} b \to a' \widehat{\rightarrow} b \text{ if } a \to a' \)

\( b \widehat{\rightarrow} a \to b \widehat{\rightarrow} a' \text{ if } a \to a' \)

*let* \( x=a \) *in* \( b \to *let* \( x=a' \) *in* \( b \text{ if } a \to a' \)

\( a \to b \text{ if } a \equiv a', a' \to b', b' \equiv b \)
Synchronisation primitives

\[
\text{conc}_{\text{m}} \triangleq \text{conc}_{\text{m}+} \quad \text{mutex denominations}
\]

\[
\{ \text{acquire}(u), \text{release}(u) \} + \{ p \rightarrow \text{locked}, p \rightarrow \text{unlocked} \}
\]

Reduction \( \alpha \rightarrow \beta \): additional rules for mutexes

\[
(p \rightarrow \text{unlocked}) \vdash \text{acquire}(p) \rightarrow (p \rightarrow \text{locked}) \vdash p
\]

\[
(p \rightarrow d) \vdash \text{release}(p) \rightarrow (p \rightarrow \text{unlocked}) \vdash p \quad \text{if } d \in \{ \text{locked}, \text{unlocked} \}
\]

- Mutexes may be implemented using a single queue.
- Mutexes may be encoded within \( \text{conc}_{\text{m}} \) using a shared memory synchronisation algorithm (Dijkstra 1965).
Example: Critical regions

Many programming languages offer support for critical regions:

\[
\text{lock } u \text{ in } a \triangleq \text{acquire}(u); \text{let } y = a \text{ in } (\text{release}(u); y)
\]

(where \(a; b \triangleq \text{let } x = a \text{ in } b \text{ for } x \notin \text{fv}(b)\))

For example, we synchronise access to a shared resource:

\[
\text{let } x = \text{unlocked} \text{ in let } f = \text{fact} \text{ in } \\
(\text{lock } x \text{ in } f(10)) \Rightarrow (\text{lock } x \text{ in } f(20))
\]
**Example: Asynchronous channels**

\[ \text{newChan} \triangleq \]

\[(\forall rd)(\forall wr) (\text{rd} \mapsto \text{locked} \Rightarrow \text{wr} \mapsto \text{unlocked} \Rightarrow \]

\[\text{val} = \zeta(s) s.\text{val},\]

\[\text{read} = \zeta(s) \text{acquire(\text{rd})}; \text{let } x = s.\text{val} \text{ in } (\text{release(\text{wr})}; x),\]

\[\text{write} = \zeta(s) \lambda(x) \text{acquire(\text{wr})}; (s.\text{val} \leftarrow \zeta(s)x); \text{release(\text{rd})}; x)\]

- Invariant: at any time at most one of \text{rd} and \text{wr} is unlocked.

- If \text{rd} is unlocked, the result in \text{val} is the contents of the channel. If \text{wr} is unlocked, the channel is empty.
Encoding the asynchronous $\pi$-calculus

\[
\begin{align*}
    [\overline{x} y] &= x.\text{write}(y) \\
    [x(y).P] &= \text{let } y = x.\text{read} \text{ in } [P] \\
    [P \mid Q] &= [P] \triangleright [Q] \\
    [(\text{new } x)P] &= \text{let } x = \text{newChan} \text{ in } [P] \\
    [!x(y).P] &= [\text{rep} = \zeta(s)\text{let } y = x.\text{read} \text{ in } ([P] \triangleright s.\text{rep})].\text{rep}
\end{align*}
\]

- We conjecture that this translation is sound with respect to a suitable notion of contextual equivalence.

- This particular translation is not fully abstract.
Well-formed Terms

A simple type system for well-formed terms makes these guarantees:

(1) The top-level denominations in a term represent a partial function, whose domain is preserved by reduction.

Neither \( p \leftarrow d_1 \mapsto p \leftarrow d_2 \) nor \( p \leftarrow [\ell = \zeta(x) q \leftarrow d] \) is well-formed.

(2) A denomination does not occur where a result is expected.

The term \( \text{let } x = p \leftarrow d \text{ in } b \) is not well-formed.

(3) Each restriction of \( p \) includes a denomination of \( p \) in its scope.

The term \( \forall p. p. \ell \) is not well-formed.
Well-formed terms: \( a : T \) where \( T ::= \text{Exp} \mid \text{Proc} \)

(Well Result) (Well Object)

\[
\begin{array}{c}
\text{u : Exp} \\
\text{p} \mapsto [l_i = \zeta(x_i) b_i]_{i \in 1..n} : \text{Proc}
\end{array}
\]

(Well Select) (Well Update) (Well Clone) (Well Res)

\[
\begin{array}{c}
\text{u} : \text{Exp} \\
\text{u} \text{l} \leftarrow \zeta(x) b : \text{Exp} \\
\text{clone(u)} : \text{Exp} \\
\text{a} : T \quad p \in \text{dom(a)}
\end{array}
\]

(Well Let) (for \( \text{dom(b)} = \emptyset \)) (Well Par) (Well Concur)

\[
\begin{array}{c}
\text{let x = a in b : Exp} \\
\text{a} : \text{Proc} \quad \text{b} : T \quad \text{dom(a)} \cap \text{dom(b)} = \emptyset \\
\text{a} : \text{Exp} \\
\text{a} : \text{Proc}
\end{array}
\]

Lemma Suppose \( a : T \). If \( a \equiv b \) or \( a \rightarrow b \) then \( b : T \) and \( \text{dom(a)} = \text{dom(b)} \).
A Single-Threaded Fragment

We define $\alpha : ^1 T$ by the same rules but omitting (Well Concur).

**Lemma** Suppose $\alpha : ^1 T$. If $\alpha \equiv b$ or $\alpha \rightarrow b$ then $b : ^1 T$ and $\text{dom}(\alpha) = \text{dom}(b)$.

If $\alpha$ is a term of the imp$\zeta$-calculus, then $\alpha : ^1 \text{Exp}$.

**Theorem** Suppose $\alpha : ^1 \text{Exp}$. If $\alpha \rightarrow \alpha'$ and $\alpha \rightarrow \alpha''$ then $\alpha' \equiv \alpha''$.

If $\alpha : ^1 \text{Exp}$ then $\alpha \equiv (\nu \bar{p})(q_1 \mapsto d_1 \xrightarrow{\bar{r}} \cdots \xrightarrow{\bar{r}} q_n \mapsto d_n \xrightarrow{\bar{r}} t)$, where the thread $t$ is not a denomination.
A First-Order Type System

- Types $\mathcal{A} ::= \text{Proc} \mid \text{Exp} \mid [\ell_i: \mathcal{A}_i \ i \in 1 \ldots n]$, where the $\ell_i$ are distinct, and no $\mathcal{A}_i = \text{Proc}$.

- Subtyping:
  
  \[ [\ell: \mathcal{A}, \ell_i: \mathcal{A}_i \ i \in 1 \ldots n] <: [\ell_i: \mathcal{A}_i \ i \in 1 \ldots n] <: \text{Exp} <: \text{Proc}. \]

- Environments $E ::= \chi_1 : \mathcal{A}_1, \ldots, \chi_n : \mathcal{A}_n$.

- Typing judgment $E \vdash \alpha : \mathcal{A}$.

- If $E \vdash \alpha : \mathcal{A}, \mathcal{A} <: T$, and $T \in \{\text{Proc}, \text{Exp}\}$ then $\alpha : T$.

- Suppose $E \vdash \alpha : \mathcal{A}$. If $\alpha \equiv b$ or $\alpha \rightarrow b$ then $E \vdash b : \mathcal{A}$.
(Val Subsumption)  
\[ E \vdash a : A \quad \text{and} \quad A <: B \]  
\[ \quad \Rightarrow \quad E \vdash a : B \]

(Val \( u \))  
\[ E, u : A, E' \vdash \diamond \quad \Rightarrow \quad E \vdash u : [\ell_i:B_i \ i \in 1..n] \]

(Val Select) (where \( j \in 1..n \))  
\[ E \vdash u : A, E' \vdash u : A \quad \Rightarrow \quad E \vdash u.\ell_j : B_j \]

(Val Object) (where \( A = [\ell_i:B_i \ i \in 1..n] \) and \( E = E', p : A, E'' \))  
\[ E, x_i : A \vdash b_i : B_i \quad \text{dom}(b_i) = \emptyset \quad \forall i \in 1..n \]
\[ \quad \Rightarrow \quad E \vdash p \mapsto [\ell_i = \zeta(x_i)b_i \ i \in 1..n] : \text{Proc} \]

(Val Let)  
\[ E \vdash a : A \quad E, x : A \vdash b : B \quad \text{dom}(b) = \emptyset \quad A <: \text{Exp} \quad B <: \text{Exp} \]
\[ \quad \Rightarrow \quad E \vdash \text{let } x=a \text{ in } b : B \]

(Val Par) (where \( \text{dom}(a) \cap \text{dom}(b) = \emptyset \))  
(Val Res)  
\[ E \vdash a : \text{Proc} \quad E \vdash b : B \quad \Rightarrow \quad E \vdash a \uparrow b : B \]
\[ E, p : A \vdash a : B \quad p \in \text{dom}(a) \quad \Rightarrow \quad E \vdash (\forall p)a : B \]
Examples of Typing and Subtyping

\[ \begin{align*}
A \rightarrow B & \triangleq [\text{arg:} A, \text{val:} B] \\
\uparrow A & \triangleq [\text{read:} A, \text{write:} A \rightarrow A] \\
\uparrow A & \triangleq [\text{write:} A \rightarrow A] \\
\downarrow A & \triangleq [\text{read:} A] \\
\end{align*} \]

- \( \emptyset \vdash \text{newChan} : \uparrow A \)
- \( \emptyset \vdash \text{newChan} : \uparrow A \) from \( \downarrow A <: \uparrow A \)
- \( \emptyset \vdash \text{newChan} : \downarrow A \) from \( \downarrow A <: \downarrow A \)
Alternative Styles of Semantics

Our chemical semantics based on $a \xrightarrow{} b$ is an alternative to:

Conclusion

We solved \( \text{conc} \eta = \text{imp} \eta + \pi \) by setting

\[
\text{conc} \eta \overset{\Delta}{=} \text{imp} \eta + \{ a \rightarrow b, (\forall p) \alpha \} + \{ p \leftrightarrow [l_i = \eta(x_i)b_i] \}
\]

This appears to be the first detailed study of a chemical semantics for an asymmetric \( a \rightarrow b \), and the first typed concurrent extension of \( \text{imp} \eta \).

As a sanity check, we proved the equivalence of our chemical semantics and a classical SOS semantics.

Future work: equivalence, enriched type systems, mobility.