
Linear Logic and Process Algebras

David Baelde — LIX

directed by

Dale Miller — LIX

Outline

- Linear Logic
- Process Algebras
- CCS encoding, correctness and completeness
- Join-calculus encoding
- Synchronism
- Testing semantics
- Conclusion

Our target logic will be (mostly) multiplicative linear logic, with exponentials and first-order quantification.

$$\frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B} \otimes \quad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \wp B} \wp$$

$$\frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A} ?C \quad \frac{\vdash \Gamma}{\vdash \Gamma, ?A} ?W$$

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, ?A} ?D \quad \frac{\vdash A, ?\Gamma}{\vdash !A, ?\Gamma} !$$

$$\frac{\vdash \Gamma, Bc}{\vdash \Gamma, \forall x . Bx} \quad \frac{\vdash \Gamma, Bt}{\vdash \Gamma, \exists x . Bx}$$

$$P ::= 0 \mid P|P \mid a.P \mid \bar{a}$$

$$\bar{a}|a.Q|\Gamma \longrightarrow Q|\Gamma$$

$$P ::= 0 \mid P|P \mid a.P \mid \bar{a}$$

$$\bar{a}|a.Q|\Gamma \longrightarrow Q|\Gamma$$

A few extensions :

Value passing : $\bar{a}(v)|a(x).Q|\Gamma \longrightarrow Q[v/x]|\Gamma$

Synchronization : $\bar{a}(v).P|a(x).Q|\Gamma \longrightarrow P|Q[v/x]|\Gamma$

Choice : $P + Q \longrightarrow P$

$$P + Q \longrightarrow Q$$

Replication : $!P \equiv !P|P$

Fresh names : $\nu(x)(P)|Q \equiv \nu(x)(P|Q)$

if x is not free in Q

$$\begin{aligned}[0] &= \perp \\ [P \mid Q] &= [P] \wp [Q] \\ [\bar{a}] &= a \\ [a.P] &= a^\perp \otimes [P]\end{aligned}$$

Theorem 1 (Reduction & Deduction).

$$\forall P . \forall Q . \quad P \rightarrow^* Q \quad \Rightarrow \quad \frac{\vdash [Q]^\uparrow}{\vdash [P]} \quad (1)$$

$$\forall P . \quad \frac{\vdash \Gamma}{\vdash [P]} \quad \Rightarrow \quad \begin{cases} \exists Q . \Gamma^\uparrow = [Q]^\uparrow \\ \forall Q . \Gamma^\uparrow = [Q]^\uparrow \Rightarrow P \rightarrow^* Q \end{cases} \quad (2)$$

To the reduction

$$\bar{c}|a.\bar{b}|\bar{a}|b.P|c.0 \longrightarrow^* P$$

we associate the following 1-open derivation :

$$\begin{array}{c}
 \frac{}{\vdash a^\perp, a} \quad \frac{}{\vdash c, c^\perp} \quad \frac{}{\vdash b, b^\perp} \quad \frac{\vdash P}{\vdash P, \perp} \\
 \hline
 \frac{}{\vdash c, c^\perp} \quad \frac{}{\vdash b, b^\perp \otimes P, \perp} \\
 \hline
 \frac{}{\vdash a^\perp, a} \quad \frac{}{\vdash c, c^\perp} \quad \frac{}{\vdash b, b^\perp \otimes P, \perp} \\
 \hline
 \frac{}{\vdash c, c^\perp} \quad \frac{}{\vdash b, b^\perp \otimes P, \perp} \\
 \hline
 \frac{}{\vdash c, a^\perp \otimes b, a, b^\perp \otimes P, c^\perp \otimes \perp} \\
 \hline
 \frac{}{\vdash c \wp a^\perp \otimes b \wp a \wp b^\perp \otimes P \wp c^\perp \otimes \perp}
 \end{array}$$

Every 1-open derivation can be turned into a “reduction-derivation” :

$$\frac{\frac{\overline{\vdash a^\perp, a}}{\vdash a^\perp, a, \perp} \quad \overline{\vdash b^\perp, b}}{\vdash a^\perp, a, b^\perp \otimes \perp, b} \quad \vdash P}{\vdash a^\perp \otimes P, a, b^\perp \otimes \perp, b}$$

Every 1-open derivation can be turned into a “reduction-derivation” :

$$\begin{array}{c}
 \frac{}{\vdash a^\perp, a} \\
 \hline
 \vdash a^\perp, a, \perp \quad \frac{}{\vdash b^\perp, b} \\
 \hline
 \vdash a^\perp, a, b^\perp \otimes \perp, b \quad \vdash P \\
 \hline
 \vdash a^\perp \otimes P, a, b^\perp \otimes \perp, b
 \end{array}
 \Downarrow
 \begin{array}{c}
 \frac{}{\vdash b^\perp, b} \quad \frac{\frac{}{\vdash a^\perp, a} \quad \frac{\vdash P}{\vdash P, \perp}}{\vdash a^\perp \otimes P, a, \perp} \\
 \hline
 \vdash a^\perp \otimes P, a, b^\perp \otimes \perp, b
 \end{array}$$

The Join-calculus is an asynchronous calculus in which receptions are localized :

$$\begin{aligned} P & ::= 0 \quad | \quad P|P \quad | \quad a(\bar{x}) \quad | \quad \mathbf{let\ def\ } D \mathbf{\ in\ } P \\ D & ::= J \triangleright P \quad | \quad D \wedge D \\ J & ::= a(\bar{x}) \quad | \quad J|J \end{aligned}$$

$$\begin{aligned} \vdash P|Q & \rightarrow \vdash P, Q \\ \vdash 0 & \rightarrow \vdash \\ \vdash \mathbf{let\ def\ } D \mathbf{\ in\ } P & \rightarrow D\sigma_{dv} \vdash P\sigma_{dv} \quad \text{for fresh } range(\sigma_{dv}) \\ D_1 \wedge D_2 \vdash & \rightarrow D_1, D_2 \vdash \\ J \triangleright P \vdash J\sigma_{rv} & \rightarrow \vdash J \triangleright P \vdash P\sigma_{rv} \end{aligned}$$

$$\begin{array}{l} \vdash \text{let def } R(a) | R(b) \triangleright a(b) | b(a) \text{ in} \\ \quad R(a) | R(b) | R(c) | R(d) \\ R(x) | R(y) \triangleright x(y) | y(x) \vdash R(a) | R(b) | R(c) | R(d) \\ R(x) | R(y) \triangleright x(y) | y(x) \vdash R(a), R(b), R(c), R(d) \\ R(x) | R(y) \triangleright x(y) | y(x) \vdash c(a) | a(c), R(b), R(d) \\ R(x) | R(y) \triangleright x(y) | y(x) \vdash c(a), a(c), R(b), R(d) \\ R(x) | R(y) \triangleright x(y) | y(x) \vdash c(a), a(c), b(d) | d(b) \\ R(x) | R(y) \triangleright x(y) | y(x) \vdash c(a), a(c), b(d), d(b) \end{array}$$

We encode as follows Join-calculus processes into linear logic formulas :

$$\begin{aligned} [0] &= \perp \\ [P \mid Q] &= [P] \wp [Q] \\ [a(\bar{x})] &= a(\bar{x}) \\ [\mathbf{let\ def\ } D \mathbf{ in\ } P] &= \overline{\forall dv(D)}, \quad [D] \wp [P] \\ [D_1 \wedge D_2] &= [D_1] \wp [D_2] \\ [J \triangleright P] &= ?(\overline{\exists rv(J)}, \quad [J]^\perp \otimes [P]) \end{aligned}$$

It seems that only synchronous languages can be treated. Let's extend the logic :

$$\frac{\vdash \Gamma, P, Q}{\vdash \Gamma, \bullet P, \circ Q}$$

Or, by duplicating the non-canonical connective :

$$\frac{\vdash \Gamma, P, Q}{\vdash \Gamma, a.P, \bar{a}.Q}$$

Theorem 3. *The initial rule can be reduced to the atomic case.*

Theorem 4. *The cut rule is admissible.*

It is now easy to encode finite synchronous CCS :

$$\begin{aligned}[0] &= \perp \\ [P \mid Q] &= [P] \wp [Q] \\ [a.P] &= a.[P]\end{aligned}$$

It is worth noting that everything works the same with the following rule :

$$\frac{\vdash \Gamma, A, B \quad \vdash \Delta, P, Q}{\vdash \Gamma, \Delta, A.P, B.Q}$$

We define the *behaviour* of a process :

$$\llbracket P \rrbracket = \{\Gamma \mid \vdash \Gamma, P\} \quad [P] \vdash [Q] \Leftrightarrow \llbracket P \rrbracket \subset \llbracket Q \rrbracket$$

We get nice properties, but this is *more discriminating* than the usual testing semantics in concurrency.

For example, this proves mutual simulations :

$$\begin{aligned} a^\perp \otimes P &\equiv \forall b . (a^\perp \otimes b \wp b^\perp \otimes P) \\ [a.b.P] &\equiv [b.a.P] \end{aligned}$$

But the \wp has a new meaning :

$$[a.a.a.0 \mid a.0] \not\wp [a.a.0 \mid a.a.0]$$

The correspondance is elementary and looks promising but is not (yet) much understood nor applied.

- A clear limitation to *may* properties at the logic level.
- A unifying framework for reasoning on processes at the meta-level.
- Extend the correspondancy, interpret full logic.