Computation Orchestration

Jayadev Misra

Department of Computer Science
University of Texas at Austin

Email: misra@cs.utexas.edu
web: http://www.cs.utexas.edu/users/psp

Collaborators: William Cook, Tony Hoare, Galen Menzel, David Kitchin

Example: Airline

- Contact two airlines simultaneously for price quotes.
- Buy ticket from either airline if its quote is at most $300.
- Buy the cheapest ticket if both quotes are above $300.
- Buy any ticket if the other airline does not provide a timely quote.
- Notify client if neither airline provides a timely quote.

Example: workflow

- An office assistant contacts a potential visitor.
- The visitor responds, sends the date of her visit.
- The assistant books an airline ticket and
  contacts two hotels for reservation.
- After hearing from the airline and any of the hotels:
  he tells the visitor about the airline and the hotel.
- The visitor sends a confirmation which the assistant notes.

Example: workflow, contd.

After receiving the confirmation, the assistant

- confirms hotel and airline reservations.
- reserves a room for the lecture.
- announces the lecture by posting it at a web-site.
- requests a technician to check the equipment in the room.

Wide-area Computing

Acquire data from remote services.

Calculate with these data.

Invoke yet other remote services with the results.

Additionally

Invoke alternate services for failure tolerance.

Repeatedly poll a service.

Ask a service to notify the user when it acquires the appropriate data.

Download an application and invoke it locally.

Have a service call another service on behalf of the user.

The Nature of Distributed Applications

Three major components in distributed applications:

Persistent storage management

databases by the airline and the hotels.

Specification of sequential computational logic

does ticket price exceed $300?

Methods for orchestrating the computations

contact the visitor for a second time only after hearing from the airline and one of the hotels.

We look at only the third problem.

Related Models and Languages

- Process Calculi: CSP, CCS, π-calculus, Join Calculus
- Petri Net
- Statechart
- Programming Languages
  - Pict: Based on π-calculus
  - Cω: Based on Join Calculus
  - Concurrent ML, Concurrent Haskell: Based on CCS (see List Monads)
  - Esterel, Lustre

Related Work, Applications

- Workflow: Based on extensions to petri nets, π-calculus
- Busieness Process Orchestration: BPEL, OWL-S, ...

Site

Compose basic computing elements called Sites. A site is a

- function: Compress MPEG file
- method of an object: LogOn procedure at a bank
- monitor procedure: read from a buffer
- web service: CNN, get a stock quote
- transaction: check account balance
- distributed transaction: move money from one bank to another
- Humans: Send email, expect report

More on Sites

- Site calls are strict: Arguments must be defined.
- A site returns at most one value.
- A site may not respond.

  Its response at different times (for the same input) may be different.
- A site call may change states (of external servers) tentatively or permanently.

  Tentative state changes are made permanent by explicit commitment.
- A site may be an argument of a site call.

Some Fundamental Sites

0: never responds.

$let(x, y, \cdots)$: returns a tuple of its argument values.

$if(b)$: boolean b,
returns a signal if b is true; remains silent if b is false.

$Signal$ returns a signal immediately. Same as $if(\textit{true})$.

$Rtimer(t)$: integer t, $t \geq 0$, returns a signal t time units later.

Orc

An Orc expression is

1. Simple: just a site call, or

2. composition of two Orc expressions

Evaluation of Orc expression:

calls some sites,

publishes some values

Simple Orc Expression

$CNN(d)$

calls site CNN,

publishes the value, if any, returned by the site.

Combinators (Composition Operators)

do f and g in parallel	$f \mid g$	Symmetric composition
for all x from f do g	$f \; {>}x{>} \; g$	Sequencing, or Piping
for some x from g do f	$f \; \mathrm{where} \; x{:}{\in} \; g$	Asymmetric composition

Composition Operators, Examples

- $CNN \mid BBC$ Symmetric composition
- $CNN\ >x>\ Email(address, x)$ Sequencing
- $(Email(address, x)\ \text{where}\ x{:}\in (CNN \mid BBC))$ Asymmetric composition

Convention, Note

- Precedence of binding: where , $|$, $\gg$

- No arithmetic or logic capability in Orc.
  Can't write $u + v$ or $x \vee y$.
  Write $add(u, v)$ and $or(x, y)$, where add and or are sites.

Centralized Execution Model

- An expression is evaluated on a single machine (client).
- Client communicates with sites by messages.
- All fundamental sites are local to the client.
  All except $Rtimer$ respond immediately.
- Concurrent and distributed executions are possible.

Symmetric composition: $f \mid g$

Evaluate f and g independently.

Publish all values from both.

Example:
$CNN \mid BBC$: calls both CNN and BBC simultaneously.

Publishes values returned by both sites. (0, 1 or 2 values)

Note:

No direct communication or interaction between f and g.

They may communicate only through sites.

Sequencing: $f \; {>}x{>} \; g$

For all values published by f do g. Publish only the values from g.

- $CNN \; {>}x{>} \; Email(address, x)$

  Call CNN. Name any value returned x. Call $Email(address, x)$.

  Publish the value (a signal), if any, returned by $Email$.

- $(CNN \mid BBC) \; {>}x{>} \; Email(address, x)$

  May call $Email$ twice. Publishes up to two signals.

Notation:
Write $f \gg g$ for $f \; {>}x{>} \; g$ if x unused in g.

Schematic of Sequencing

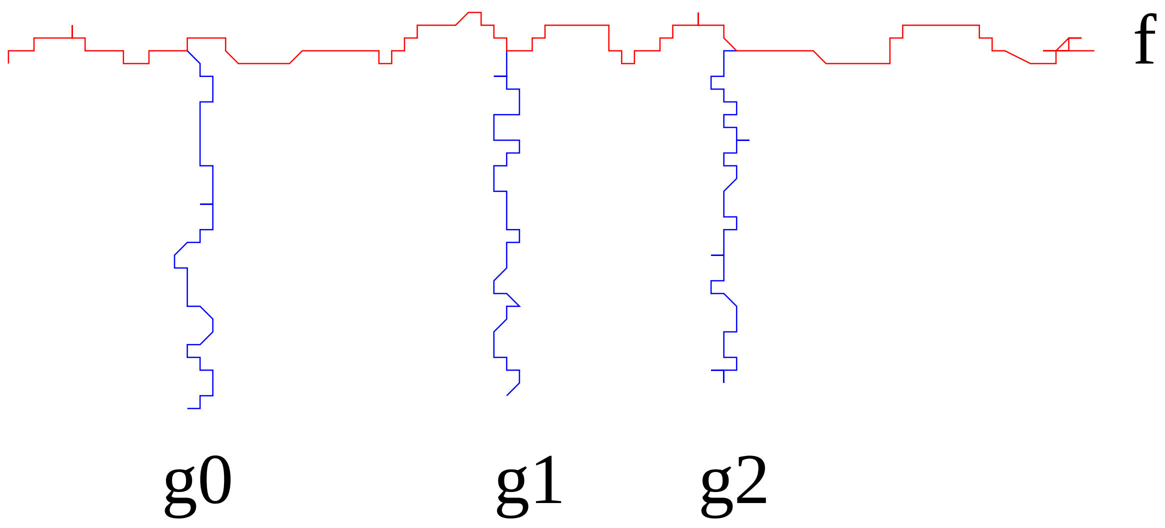


Figure 1: Schematic of $f\ {>}x{>}\ g$

Notes on Sequencing

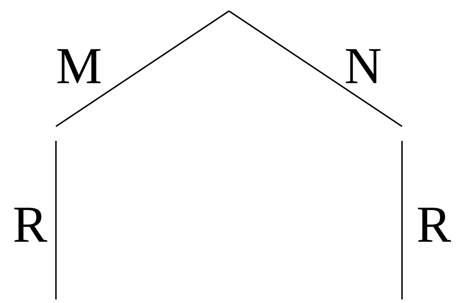
- $\gg$ is associative. $>x>$ is right associative.
- A fresh evaluation of g is started with each returned value x. Many copies of g may be executing concurrently, along with f.
- If f publishes at most one value and halts, $f >x> g$ is $f;g$.
- If f publishes no value, g is never evaluated in $f >x> g$.

Questions

$$
\begin{array}{lcl}
M \mid M & \stackrel{?}{=} & M \\
(M \mid N) \gg R & \stackrel{?}{=} & M \gg R \mid N \gg R \\
M \gg (N \mid R) & \stackrel{?}{=} & M \gg N \mid M \gg R \\
\mathit{if}\,(b) \gg M \mid \mathit{if}\,(\neg b) \gg M & \stackrel{?}{=} & M
\end{array}
$$

$$\boxed{(M \mid N) \gg R \;=\; M \gg R \mid N \gg R}$$

Evaluate M and N. For each published value, call R.

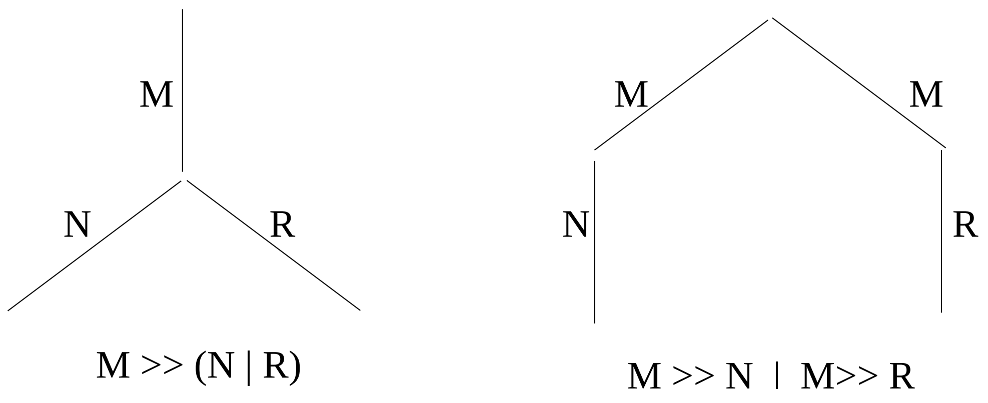


Thus, R may be called twice.

$$(Email(address1, message) \mid Email(address2, message)) \gg Notify$$

Double notification.

$$M \gg (N \mid R) \neq M \gg N \mid M \gg R$$



Asymmetric parallel composition: $(f \text{ where } x{:}\in g)$

For some value published by g do f. Publish only the values from f.

- Evaluate f and g in parallel.
- When g returns a value, assign it to x and terminate g.
- Any site call in f which does not name x can proceed.
  $(M \mid N(x)) \text{ where } x{:}\in f$
- A site call which names x waits until x gets a value.
- Values published by f are the values of $(f \text{ where } x{:}\in g)$.

Pruning the computation

$(CNN \mid BBC) \; > x > \; Email(address, x)$

May send two emails.

To send just one email:

$$Email(address, x) \; \text{where} \; x :\in (CNN \mid BBC)$$

Notify Once after both M and N respond

$(M \mid N)) \gg Notify$: notifies twice.

Use

$((let(u, v) \gg Notify$
where
$u{:}\in M)$
where
$v{:}\in N)$

Adopt the notation:

$(let(u, v) \gg Notify$
where
$u{:}\in M$
$v{:}\in N)$

Fundamental Site 0

0 is a site which never responds.

Example: send an email but do not wait for its response:

$$(Email(address1, message) \gg 0 \;\;|\;\; Notify)$$

Expression Definition

An expression is defined like a procedure.

$$MailOnce(a) \;\underline{\Delta}$$
$$\quad Email(a, m) \;\; \text{where}\; m{:}\in (CNN \mid BBC)$$

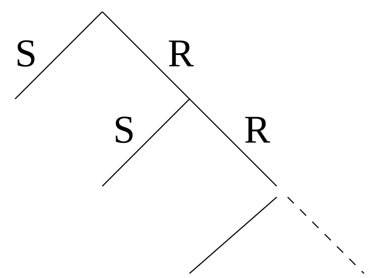
$$MailLoop(a, t) \;\underline{\Delta}$$
$$\quad MailOnce(a) \gg Rtimer(t) \gg MailLoop(a, t)$$

Note: $MailLoop$ does not publish a value.

Metronome

Publish a signal at every time unit.

$$Metronome \;\underline{\Delta}\; Signal \mid Rtimer(1) \gg Metronome$$



Publish n signals.

$$\begin{array}{ll} BM(0) & \underline{\Delta}\; 0 \\ BM(n) & \underline{\Delta}\; Signal \mid Rtimer(1) \gg BM(n-1) \end{array}$$

Example of Expression call

- Site $Query$ returns a value (different ones at different times).
- Site $Accept(x)$ returns x if x is acceptable;
  it is silent otherwise.
- Produce all acceptable values by calling $Query$ at unit intervals forever.

$$Metronome \gg Query \; {>}x{>} \; Accept(x)$$

Synchronous Semantics

Call sites as soon as possible.

Consequently:

- In $M \mid N$, both sites are called simultaneously.
- A response is processed only if no site can be called.
- In $(f \text{ where } x{:}{\in}\, g)$, x gets the first value from f.

Fundamental sites get priority:

Process responses from fundamental sites before any other response.

Some Fundamental Sites

0: never responds.

$let(x, y, \cdots)$: returns a tuple of its argument values.

$if(b)$: boolean b,
returns a signal if b is true; remains silent if b is false.

$Signal$ returns a signal immediately. Same as $if(\textit{true})$.

$Rtimer(t)$: integer t, $t \geq 0$, returns a signal t time units later.

Small Examples

- Call site M four times, at unit time intervals.

$$M \;|\; Rtimer(1) \gg M \;|\; Rtimer(2) \gg M \;|\; Rtimer(3) \gg M$$

- Time-out: return M's response if it arrives before t, return *false* after t.

$$let(z) \text{ where } z{:}{\in}\; M \;|\; Rtimer(t) \gg let(\textit{false})$$

Priority

- Receive N's response asap, but no earlier than 1 unit from now.

$$Delay(N) \ \underline{\Delta} \quad (Rtimer(1) \gg let(u)) \text{ where } u{:}\in N$$

- Call M, N together.

  If M responds within one unit, take its response.

  Else, pick the first response.

$$let(x) \text{ where } x{:}\in (M \mid Delay(N))$$

Recursive definition with time-out

Call a list of sites.

Count the number of responses received within 10 time units.

$$
\begin{array}{ll}
tally([\,]) & \underline{\Delta} \;\; let(0) \\
tally(M : MS) & \underline{\Delta} \\
& \quad u + v \\
& \quad\quad \text{where} \\
& \quad\quad\quad u{:}\in \; M \gg let(1) \mid Rtimer(10) \gg let(0) \\
& \quad\quad\quad v{:}\in \; tally(MS)
\end{array}
$$

Sequential Computing

- $(S;\ T)$ is $(S \gg T)$

- **if** b **then** S **else** T

  is

  $$if(b) \gg S \mid if(\neg b) \gg T$$

- **while** $B(x)$ **do** $x := S(x)$

  $$loop(x) \ \underline{\Delta} \ B(x) > b > (if(b) \gg S(x) > y > loop(y) \mid if(\neg b) \gg let(x))$$

Kleene Star

- For a given x, produce the set of values

$$x,\ M(x),\ M(x) \; >y> \; M(y),\ M(x) \; >y> \; M(y) \; >z> \; M(z),\ \ldots$$

$$Mstar(x) \;\underline{\Delta}\; let(x) \mid M(x) \; >y> \; Mstar(y)$$

- Produce the same set of values without x, i.e.,

$$M(x),\ M(x) \; >y> \; M(y),\ M(x) \; >y> \; M(y) \; >z> \; M(z),\ \ldots$$

$$Mplus(x) \;\underline{\Delta}\; M(x) \; >y> \; (let(y) \mid Mplus(y))$$

Arbitration

In CCS: $\alpha.P + \beta.Q$

In Orc:

$$if\,(b) \gg P \mid if\,(\neg b) \gg Q$$

where

$$b :\in Alpha \gg let(\textsf{true}) \mid Beta \gg let(\textsf{false})$$

Time-out

Return $(x, \textit{true})$ if M returns x before t, and $(-, \textit{false})$ otherwise.

$let(z, b)$
where
$(z, b) :\in M \;>x>\; let(x, \textit{true}) \;\mid\; Rtimer(t) \;>x>\; let(x, \textit{false})$

Fork-join parallelism

Call M and N in parallel.

Return their values as a tuple after both respond.

$$
\begin{array}{l}
let(u, v) \\
\quad \text{where} \quad u :\in M \\
\qquad\qquad\quad v :\in N
\end{array}
$$

Return a signal after both respond.

$$
\begin{array}{l}
let(u) \gg let(v) \gg Signal \\
\quad \text{where} \quad u :\in M \\
\qquad\qquad\quad v :\in N
\end{array}
$$

Barrier Synchronization

Synchronize $M \gg f$ and $N \gg g$:

f and g start only after both M and N complete.

Rendezvous of CSP or CCS; M and N are complementary actions.

$$
\begin{array}{l}
(\ \ let(u, v) \\
\quad \text{where } u{:}{\in}\ M \\
\qquad\qquad\ \ v{:}{\in}\ N\,) \\
\gg\ (f \mid g)
\end{array}
$$

To pass values from M and N to f and g, modify last line:

$$>(u, v)>\ (f \mid g)$$

Interrupt handling

- Orc statement can not be directly interrupted.
- $Interrupt$ site: a monitor.
- $Interrupt.set$: to interrupt the Orc statement
- $Interrupt.get$: responds after $Interrupt.set$ has been called.

Use

$$let(z) \text{ where } z{:}{\in}\ (f \ \mid \ Interrupt.get)$$

Interrupt; contd.

Determine if there has been an interrupt:

$$
\begin{array}{l}
callM \ \underline{\Delta} \\
\quad (let(z,b) \\
\qquad \textsf{where} \\
\qquad\quad (z,b) :\in M \ >x> \ let(x,\textsf{true}) \ \mid \ Interrupt.get \ >x> \ let(x,\textsf{false}) \\
\quad)
\end{array}
$$

Process Interrupt:

$$
\begin{array}{l}
\qquad\qquad callM \\
>(z,b)> \\
\qquad\qquad (\quad if\,(b) \quad \gg \ \text{“Normal processing with value } z\text{”} \\
\qquad\qquad \mid \ if\,(\neg b) \quad \gg \ \text{“Interrupt Processing”} \quad)
\end{array}
$$

Parallel or

Sites M and N return booleans. Compute their parallel or.

$ift(b) = if(b) \gg let(\textit{true})$: returns *true* if b is *true*; silent otherwise.

$$ift(x) \mid ift(y) \mid or(x, y)$$
where
$$x{:}\in M,\ y{:}\in N$$

To return just one value:

$$let(z)$$
where
$$z{:}\in ift(x) \mid ift(y) \mid or(x, y)$$
$$x{:}\in M,\ y{:}\in N$$

Airline quotes: Application of Parallel or

Contact airlines A and B.

Return any quote if it is below c as soon as it is available, otherwise return the minimum quote.

$threshold(x)$ returns x if $x < c$; silent otherwise.
$Min(x, y)$ returns the minimum of x and y.

$let(z)$
where
$z :\in threshold(x) \mid threshold(y) \mid Min(x, y)$
$x :\in A$
$y :\in B$

Backtracking: Eight queens

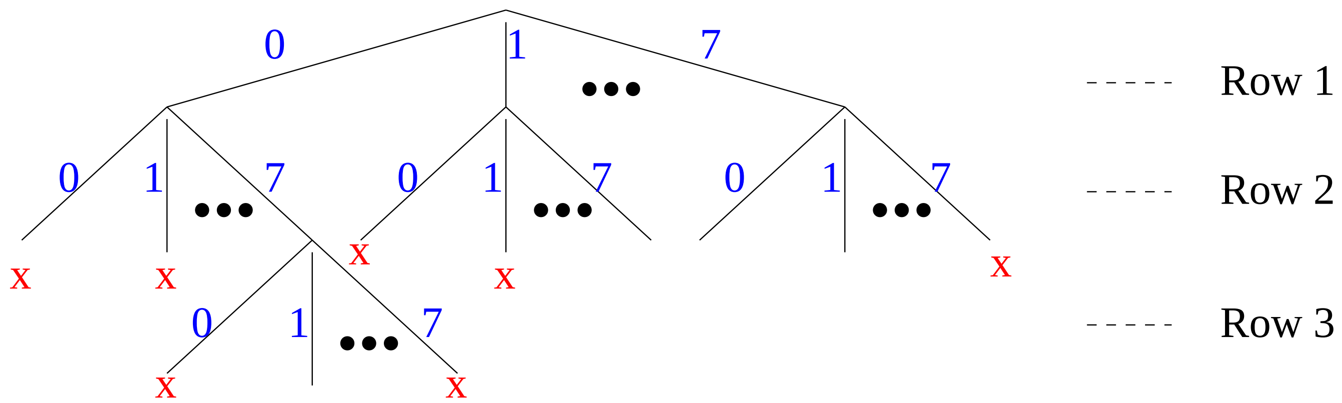


Figure 2: Backtrack Search for Eight queens

Eight queens; contd.

- configuration: placement of queens in the last i rows.
- Represent a configuration by a list of integers j, $0 \le j \le 7$.
- Valid configuration: no queen captures another.

Eight queens; contd.

- Site $check(x{:}xs)$: Given xs is valid,

  return $x{:}xs$, if it is valid; remain silent otherwise

- Produce all valid extensions of z by placing n additional queens:

$$
\begin{array}{lcl}
extend(z,1) & \underline{\Delta} & check(0{:}z) \mid check(1{:}z) \mid \cdots \mid check(7{:}z) \\
extend(z,n) & \underline{\Delta} & extend(z,1) \; >y> \; extend(y,n-1)
\end{array}
$$

- Solve the original problem by calling $extend([\,],8)$.

Processes

- Processes typically communicate via channels.
- For channel c, treat $c.put$ and $c.get$ as site calls.
- In our examples, $c.get$ is blocking and $c.put$ is non-blocking.
- Other kinds of channels can be programmed as sites.

Typical Iterative Process

Forever: Read x from channel c, compute with x, output result on e:

$$P(c,e) \underline{\Delta} \; c.get \; >x> \; Compute(x) \; >y> \; e.put(y) \; \gg \; P(c,e)$$

Process (network) to read from both c and d and write on e:

$$Net(c,d,e) \; \underline{\Delta} \; P(c,e) \mid P(d,e)$$

Multiplexor, from Hoare

- A mulitiplexor receives messages from several channels, $c[i]$, $0 \leq i \leq N$.
- It reproduces all messages on outgoing channel e.
- It stops reading from a channel after seeing an eos message.

Solution:

$$\begin{array}{lll} mux & \underline{\Delta} & P_0 \mid P_1 \mid \cdots \mid P_N \\ P_i & \underline{\Delta} & c[i].get \; >x> \; if\,(x \neq eos) \gg e.put(x) \gg P_i \end{array}$$

Example

Run a dialog with the client.

Forever: client inputs an integer on channel p

Process outputs *true* on channel q iff it is prime.

Sites: $c.get$ and $c.put$, for channel c.

$Prime?(x)$ returns *true* iff x is prime.

$$\begin{array}{ll} Dialog(p,q) \;\underline{\Delta} & \\ \quad p.get & >x> \\ \quad Prime?(x) & >b> \\ \quad q.put(b) & \gg \\ \quad Dialog(p,q) & \end{array}$$

Communications among Threads

$$\begin{array}{l} \quad CreateChannel \\ >c> \\ \quad f \;>x>\; c.put(x) \mid c.get \;>x>\; g \end{array}$$

Laws of Kleene Algebra

(Zero and $\|$)	$f \mid 0 = f$
(Commutativity of $\|$)	$f \mid g = g \mid f$
(Associativity of $\|$)	$(f \mid g) \mid h = f \mid (g \mid h)$
(Idempotence of $\|$)	$f \mid f = f$
(Associativity of $\gg$)	$(f \gg g) \gg h = f \gg (g \gg h)$
(Left zero of $\gg$)	$0 \gg f = 0$
(Right zero of $\gg$)	$f \gg 0 = 0$
(Left unit of $\gg$)	$Signal \gg f = f$
(Right unit of $\gg$)	$f > x > let(x) = f$
(Left Distributivity of $\gg$ over $\|$)	$f \gg (g \mid h) = (f \gg g) \mid (f \gg h)$
(Right Distributivity of $\gg$ over $\|$)	$(f \mid g) \gg h = (f \gg h \mid g \gg h)$

Laws which do not hold

(Idempotence of $\mid$)	$f \mid f = f$
(Right zero of $\gg$)	$f \gg 0 = 0$
(Left Distributivity of $\gg$ over $\mid$)	$f \gg (g \mid h) = (f \gg g) \mid (f \gg h)$

Additional Laws

(Distributivity over $\gg$) if g is x-free

$$(f \gg g \text{ where } x{:}\in h) = (f \text{ where } x{:}\in h) \gg g$$

(Distributivity over $|$) if g is x-free

$$(f \mid g \text{ where } x{:}\in h) = (f \text{ where } x{:}\in h) \mid g$$

(Distributivity over where) if g is y-free

$$\begin{aligned} & ((f \text{ where } x{:}\in g) \text{ where } y{:}\in h) \\ = \quad & ((f \text{ where } y{:}\in h) \text{ where } x{:}\in g) \end{aligned}$$

(Elimination of where) if f is x-free, for site M

$$(f \text{ where } x{:}\in M) = f \mid M \gg 0$$

Rules for Site Call

$$\frac{u \text{ fresh}}{M(c) \overset{M\langle c,u\rangle}{\hookrightarrow} ?u} \qquad \text{(SITECALL)}$$

$$?u \overset{u?c}{\hookrightarrow} let(c) \qquad \text{(SITERET)}$$

$$let(c) \overset{\dagger c}{\hookrightarrow} 0 \qquad \text{(LET)}$$

Symmetric Composition

$$\frac{f \overset{l}{\hookrightarrow} f'}{f \mid g \overset{l}{\hookrightarrow} f' \mid g} \qquad (\textsc{Sym1})$$

$$\frac{g \overset{l}{\hookrightarrow} g'}{f \mid g \overset{l}{\hookrightarrow} f \mid g'} \qquad (\textsc{Sym2})$$

Sequencing

$$\frac{f \overset{l}{\hookrightarrow} f' \qquad l \neq \dagger c}{f \;>x>\; g \overset{l}{\hookrightarrow} f' \;>x>\; g} \qquad \text{(SEQ1N)}$$

$$\frac{f \overset{\dagger c}{\hookrightarrow} f'}{f \;>x>\; g \overset{\tau}{\hookrightarrow} (f' \;>x>\; g) \mid [c/x]g} \qquad \text{(SEQ1V)}$$

Asymmetric Composition

$$\frac{f \overset{l}{\hookrightarrow} f' \qquad l \neq \dagger c}{g \text{ where } x{:}\in f \overset{l}{\hookrightarrow} g \text{ where } x{:}\in f'} \qquad \text{(ASYM1N)}$$

$$\frac{f \overset{\dagger c}{\hookrightarrow} f'}{g \text{ where } x{:}\in f \overset{\tau}{\hookrightarrow} [c/x]g} \qquad \text{(ASYM1V)}$$

$$\frac{g \overset{l}{\hookrightarrow} g'}{g \text{ where } x{:}\in f \overset{l}{\hookrightarrow} g' \text{ where } x{:}\in f} \qquad \text{(ASYM2)}$$

Expression Call

$$\frac{[\![\, E(q) \;\underline{\Delta}\; f \,]\!] \in D}{E(p) \overset{\tau}{\hookrightarrow} [p/q]f} \qquad \text{(DEF)}$$

Rules

$$\frac{u \text{ fresh}}{M(c) \overset{M\langle c,u\rangle}{\hookrightarrow} ?u}$$

$$?u \overset{u?c}{\hookrightarrow} let(c)$$

$$let(c) \overset{\dagger c}{\hookrightarrow} 0$$

$$\frac{f \overset{l}{\hookrightarrow} f'}{f \mid g \overset{l}{\hookrightarrow} f' \mid g}$$

$$\frac{g \overset{l}{\hookrightarrow} g'}{f \mid g \overset{l}{\hookrightarrow} f \mid g'}$$

$$\frac{[[\, E(q) \ \underline{\Delta} \ f \,]] \in D}{E(p) \overset{\tau}{\hookrightarrow} [p/q]f}$$

$$\frac{f \overset{l}{\hookrightarrow} f' \qquad l \neq \dagger c}{f \ {>}x{>}\ g \overset{l}{\hookrightarrow} f' \ {>}x{>}\ g}$$

$$\frac{f \overset{\dagger c}{\hookrightarrow} f'}{f \ {>}x{>}\ g \overset{\tau}{\hookrightarrow} (f' \ {>}x{>}\ g) \mid [c/x]g}$$

$$\frac{f \overset{l}{\hookrightarrow} f' \qquad l \neq \dagger c}{g \text{ where } x{:}\in f \overset{l}{\hookrightarrow} g \text{ where } x{:}\in f'}$$

$$\frac{f \overset{\dagger c}{\hookrightarrow} f'}{g \text{ where } x{:}\in f \overset{\tau}{\hookrightarrow} [c/x]g}$$

$$\frac{g \overset{l}{\hookrightarrow} g'}{g \text{ where } x{:}\in f \overset{l}{\hookrightarrow} g' \text{ where } x{:}\in f}$$