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

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
  - C ω: 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 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

$\mathbf{0}$: 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.

Composition Operators

do f and g in parallel	$f \mid g$	Symmetric composition
for all x from f do g	$f >x> g$	Sequencing
for some x from f do g	$f \text{ where } x {:\in}\ g$	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

Conventions

- 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.
- Convention: In examples, I write $u + v$ and $x \vee y$.
  Assume that a compiler converts these to $add(u, v)$ and $or(x, y)$.

Centralized Execution Model

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

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.

Notes on Sequencing

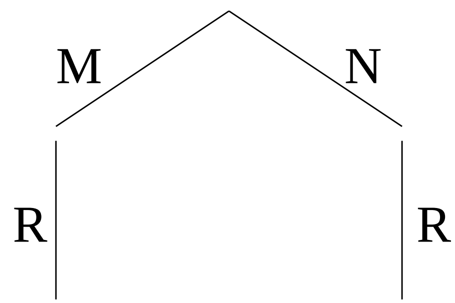
- $\gg$ is associative. $>x>$ is right associative.
- A fresh evaluation of g is started with each returned value x. Many copies of g may be executing, possibly, with f.
- If f publishes at most one value, $f >x> g$ is $f;g$.
- If f publishes no value, g is never evaluated in $f >x> g$.

Questions

- $M \;|\; M \stackrel{?}{=} M$
- $(M \;|\; N) \gg R \stackrel{?}{=} M \gg R \;|\; N \gg R$
- $M \gg (N \;|\; R) \stackrel{?}{=} M \gg N \;|\; M \gg R$
- $\mathit{if}\,(b) \gg M \;|\; \mathit{if}\,(\neg b) \gg M \stackrel{?}{=} M$

$$(M \mid N) \gg R \;=\; M \gg R \mid N \gg R$$

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

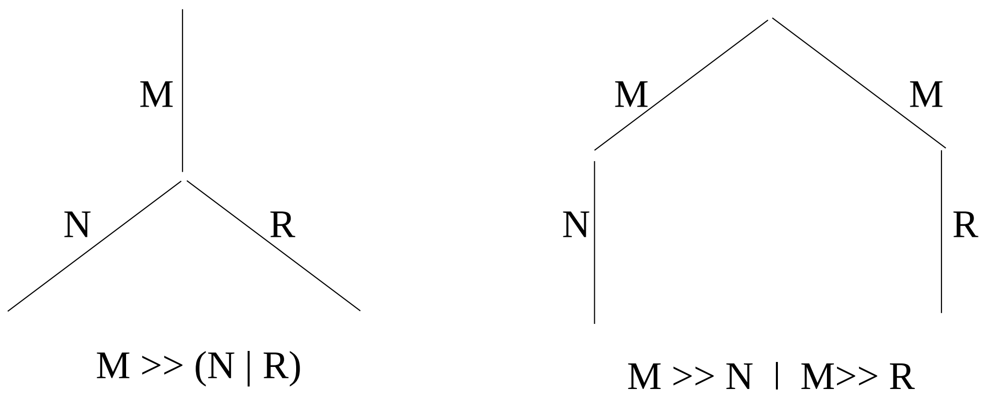


Thus, R may be called twice.

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

Double notification.

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



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

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

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

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 after both respond

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

Use

$$((let(u, v)\ \gg\ Notify$$

where

$$u{:}\in\ Email(address1, message))$$

where

$$v{:}\in\ Email(address2, message))$$

Adopt the notation:

$$(let(u, v)\ \gg\ Notify$$

where

$$u{:}\in\ Email(address1, message)$$

$$v{:}\in\ Email(address2, message))$$

Eager Evaluation in $(g \text{ where } x{:}\in\ f)$

- In $M \mid N(x) \text{ where } x{:}\in\ f$:

  g is evaluated, i.e., M is called even before x has a value.

  Any response from M will be published even before x has a value.

- In $M \gg N(x) \text{ where } x{:}\in\ f$:

  f is evaluated even before the value of x is needed.

Difference in evaluation strategies

- In $f \; {>}x{>} \; g$, g is not evaluated if f is silent.
- In $(g \text{ where } x{:}{\in} \; f)$, g may be partially evaluated if f is silent.
- Analogous to difference in values of quantified expressions over empty range.

Fundamental Site $\mathbf{0}$

$\mathbf{0}$ is a site which never responds.

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

$$(Email(address1, message) \gg \mathbf{0} \quad | \quad Notify)$$

Expression Definition

An expression is defined like a procedure.

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

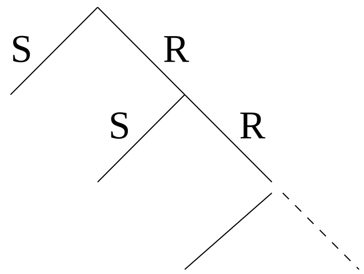
$$Ticker(a,t) \;\underline{\Delta}\; MailOnce(a) \gg Rtimer(t) \gg Ticker(a,t)$$

Note: $Ticker$ does not publish a value.

Metronome

Publish a signal at every time unit.

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



Publish n signals.

$$\begin{array}{lll} BM(0) & \underline{\Delta} & \mathbf{0} \\ BM(n) & \underline{\Delta} & Signal \ | \ Rtimer(1) \gg BM(n-1) \end{array}$$

Example of Expression call

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

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

Asynchronous Semantics

Call sites eventually.

In $M \mid N$, the sites may be called at arbitrary times.

In $M \mid Rtimer(1)$,

$Rtimer$ is any site that returns a signal after unit time.

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 $(g \text{ where } x :\in f)$, x gets the first value from f.

Fundamental sites get priority:

Process responses from fundamental sites before any other response.

Some Fundamental Sites

$\mathbf{0}$: 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.

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

- Time-out: return M's response if it arrives before t, return 0 after t.

$$let(z) \text{ where } z{:}\in \; M \;\mid\; Rtimer(t) \gg let(0)$$

Priority

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

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

- 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.

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

Opera Seat assignments

- An opera house has a list of patrons.
- It emails each patron a set of seats, from which the patron can choose.
- It contacts the patrons one at a time, and uses time-out if some patron does not respond.
- $Assign(x, m)$: x is a patron, m a seat. Assigns m to x.
- $Opera(ps, s)$: ps is a list of patrons (in decreasing order of priority), s a set of available seats, performs assignments.

Opera Seat assignments, contd.

$$
\begin{array}{lll}
Opera([], \{\}) & \underline{\Delta} & \mathbf{0} \ \{\text{All patrons and seats assigned}\} \\
Opera(ps, \{\}) & \underline{\Delta} & \mathbf{0} \ \{\text{all seats assigned to patrons}\} \\
Opera([], s) & \underline{\Delta} & \mathbf{0} \ \{\text{All patrons assigned seats}\}
\end{array}
$$

$$
\begin{array}{ll}
Opera(x : ps, s) & \underline{\Delta} \ \ Opera(ps, t) \\
& \text{where} \\
& \quad t :\in \ x(s) \ >m> \ Assign(x, m) \gg let(s - \{m\}) \\
& \qquad | \ \ Rtimer(1) \gg let(s)
\end{array}
$$

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 **do** $x{:=}\ S(x)$

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

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

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

Time-out

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

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

Fork-join parallelism

Call M and N in parallel.

Return their values as a tuple after both respond.

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

Return a signal after both respond.

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

Screen Refresh

Get: screen image, keyboard input, mouse position every time unit.

Call $Draw$ with this triple.

$$
\begin{array}{l}
\quad Metronome \\
\gg \; (\; let(i, k, m) \\
\qquad \text{where } i \;:\in\; Image \\
\qquad\qquad k \;:\in\; Keyboard \\
\qquad\qquad m :\in\; Mouse \\
\quad) \\
>x> \; Draw(x)
\end{array}
$$

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.

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

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:

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

Process Interrupt:

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

Parallel or

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

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

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

Return just one value.

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

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.

$$threshold(x) \;\;|\;\; threshold(y) \;\;|\;\; Min(x, y)$$

where

$x :\in A$

$y :\in B$

Backtracking: Eight queens

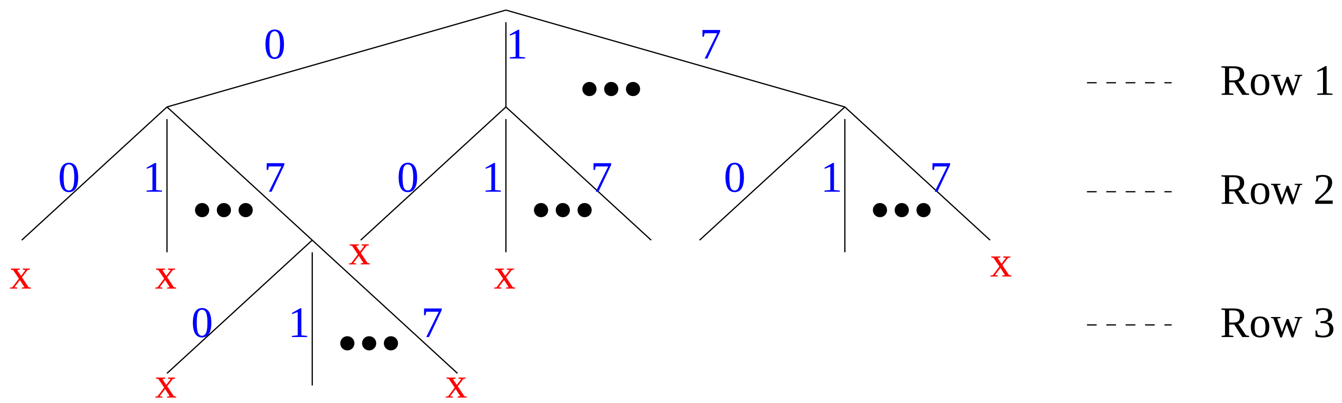


Figure 1: Backtrack Search for Eight queens

Eight queens; contd.

- configuration: placement of queens in the last i rows.
- Represent a configuration by a list of integers j, $0 \leq j \leq 7$.
- 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 x by placing n additional queens:

  x is a valid configuration, $1 \le n$ and $|x| + n \le 8$

- Define $extend(x, n)$

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

- 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.
- $c.get$ is blocking. In our examples, $c.put$ 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:

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

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.

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

Push, Pull

- $f \;{>}x{>}\; g$ may run many g in parallel, one for each publication of f.

- Run one copy of g at any time (iterate g with values from f):
  - f may publish at arbitrary speed.
  - Start an iteration only on completion of an iteration.
  - Assume g publishes at most one value.

- f writes to channel c.
  Read a new value from c only after processing the previous value.

$$(f \;{>}y{>}\; c.put(y) \gg \mathbf{0}) \mid Rg$$
$$Rg \;\underline{\Delta}\; c.get \;{>}x{>}\; g \gg Rg$$

Mutual Exclusion

- Process i writes a site name on channel c_i.
  $Multiplexor_i$ collects inputs from c_i, writes on e.

$$Multiplexor_i \;\underline{\Delta}\; c_i.get \;>x>\; e.put(x) \;\gg\; Multiplexor_i$$

- $Arbiter$ picks an item g from e; grants resource by calling g.
  g responds after the process completes the resource usage.

$$Arbiter \;\underline{\Delta}\; e.get \;>g>\; g \;\gg\; Arbiter$$

- $Mutex$ coordinates all the activities.

$$Mutex \;\underline{\Delta}\; (\;|\; i :: Multiplexor_i) \;|\; Arbiter$$

Dining Philosophers

A philosopher's life is depicted by

$$
\begin{aligned}
&P_i \;\underline{\Delta}\\
&\qquad (let(x,y) \gg Eat \gg Fork_i.put \gg Fork_{i'}.put\\
&\qquad\quad \text{where}\;\; x{:}\in\; Fork_i.get\\
&\qquad\qquad\qquad\; y{:}\in\; Fork_{i'}.get\\
&\qquad)\\
&\quad \gg P_i
\end{aligned}
$$

where $Fork_i$ and $Fork_i'$ are sites, returning signals.

Represent the ensemble of N philosophers by

$$DP \;\underline{\Delta}\; (\;|\; i{:}\; 0 \le i < N{:}\; P_i)$$

Synchronized Communication: Byzantine Protocol

- Process i sends values to j over channel c_{ij}.

$$Send_i(v) \quad \underline{\Delta} \quad (\ \mid j{::}\ c_{ij}.put(v) \gg \mathbf{0})$$
$$Read_i \quad \underline{\Delta} \quad (let(X) \text{ where } (\forall j{::}\ X_j{:}{\in}\ c_{ji}.get))$$

- $Round_i(v,n)$: For process i to run n rounds with initial value v.

$$Round_i(v,0) \quad \underline{\Delta} \quad let(v)$$
$$Round_i(v,n) \quad \underline{\Delta}$$
$$(Send_i(v) \ \mid \ Read_i) \ >X> \ Compute_i(X) \ >u> \ Round_i(u,n-1)$$

- $Byz(V,n)$: All processes run n rounds; initial value vector is V.

$$Byz(V,n) \ \underline{\Delta} \ (\ \mid i{::}\ Round_i(V_i,n))$$