Structured Concurrent Programming

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, 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.
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**

We look at only the third problem.
Overview of Orc

- Orchestration language.
  - Invoke services by calling sites
  - Manage time-outs, priorities, and failures

- A Program execution
  - calls sites,
  - publishes values.

- Simple
  - Language has only 3 combinators.
  - Semantics described by labeled transition system and traces.
  - Combinators are (monotonic and) continuous.
Structure of Orc Expression

- **Simple**: just a site call, $CNN(d)$
  Publishes the value returned by the site.

- **Composition** of two Orc expressions:

  do $f$ and $g$ in parallel  
  for all $x$ from $f$ do $g$  
  for some $x$ from $g$ do $f$

  $f \mid g$  
  $f > x > g$  
  $f$ where $x \in g$

  Symmetric composition  
  Piping  
  Asymmetric composition
Symmetric composition: \( f \mid g \)

\( CNN \mid BBC \): calls both \( CNN \) and \( BBC \) simultaneously.

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

- Evaluate \( f \) and \( g \) independently.
- Publish all values from both.
- No direct communication or interaction between \( f \) and \( g \). They may communicate only through sites.
Pipe: \texttt{f >x> g}

For all values published by \texttt{f} do \texttt{g}. Publish only the values from \texttt{g}.

- \texttt{CNN >x> Email(address,x)}
  
  Call \texttt{CNN}. Bind result (if any) to \texttt{x}. Call \texttt{Email(address,x)}.
  
  Publish the value, if any, returned by \texttt{Email}.

- \texttt{(CNN | BBC) >x> Email(address,x)}
  
  May call \texttt{Email} twice. Publishes up to two values from \texttt{Email}.  

Write $f \gg g$ for $f > x > g$ if $x$ unused in $g$.

Precedence:

$$f > x > g \mid h > y > u$$

$$(f > x > g) \mid (h > y > u)$$
Schematic of piping

Figure 1: Schematic of $f \succ x \succ g$
Asymmetric parallel composition:  \(( f \text{ where } x:\in g )\)

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

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

Binds \( x \) to the first value from \( CNN \mid BBC \).

- Evaluate \( f \) and \( g \) in parallel.
  Site calls that need \( x \) are suspended; other site calls proceed.
  \[ (M \mid N(x)) \text{ where } x:\in g \]

- When \( g \) returns a value, assign it to \( x \) and terminate \( g \).
  Resume suspended calls.

- Values published by \( f \) are the values of \(( f \text{ where } x:\in g )\).
Some Fundamental Sites

0: never responds.

let(x, y, ...) : 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(true).

Rtimer(t) : integer t, t ≥ 0, returns a signal t time units later.
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 $R\text{timer}$ respond immediately.

- Concurrent and distributed executions are derived from an expression.
**Expression Definition**

\[ MailOnce(a) \triangleq Email(a, m) \text{ where } m \in (CNN \mid BBC) \]

\[ MailLoop(a, t) \triangleq MailOnce(a) \gg Rtimer(t) \gg MailLoop(a, t) \]

- Expression is called like a procedure. May publish many values. *MailLoop* does not publish a value.
- Site calls are strict; expression calls non-strict.
Metronome

Publish a signal at every time unit.

\[ \text{Metronome} \triangle \text{Signal} \mid (\text{Rtimer}(1) \gg \text{Metronome}) \]

Publish \( n \) signals.

\[ \begin{align*}
BM(0) & \triangle 0 \\
BM(n) & \triangle \text{Signal} \mid (\text{Rtimer}(1) \gg BM(n - 1))
\end{align*} \]
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.

$$Metronome \Rightarrow Query \Rightarrow x \Rightarrow Accept(x)$$
Publish $M$’s response if it arrives before $t$, and 0 otherwise.

\[
\text{let}(z) \\
\text{where} \\
z \in \\
M \\
| \text{Rtimer}(t) \gg \text{let}(0)
\]
Fork-join parallelism

Call $M$ and $N$ in parallel.

Return their values as a tuple after both respond.

$$\text{let}(u, v)$$

where $u \in M$

where $v \in N$

This stands for:

$$(\text{let}(u, v)$$

where $u \in M)$$

where $v \in N$$
Recursive definition with time-out

Call a list of sites.

Count the number of responses received within 10 time units.

\[ \text{tally}([]) \triangleq \text{let}(0) \]
\[ \text{tally}(M : MS) \triangleq u + v \]

where

\[ u : \in (M \triangleright \text{let}(1)) \mid (\text{Rtimer}(10) \triangleright \text{let}(0)) \]
\[ v : \in \text{tally}(MS) \]
Barrier Synchronization in \( M \gg f \mid N \gg g \)

\( f \) and \( g \) start only after both \( M \) and \( N \) complete.

\[
\begin{align*}
\text{( let}(u, v) \\
\text{where} \quad u &\in M \\
v &\in N \text{)} \\
\gg (f | g)
\end{align*}
\]
In CCS/ Pi-Calculus: \( \alpha.P + \beta.Q \)

In Orc:

\[
\text{if}(b) \Rightarrow P | \text{if}(\neg b) \Rightarrow Q
\]

where

\[
b : \in (\text{Alpha} \Rightarrow \text{let}(true)) | (\text{Beta} \Rightarrow \text{let}(false))
\]

Orc does not permit non-deterministic internal choice.
- Publish $N$’s response asap, but no earlier than 1 unit from now.

$$\text{Delay} \triangleq (R\text{timer}(1) \triangleright \text{let}(u)) \text{ where } u \in N$$

- Call $M$, $N$ together.

  If $M$ responds within one unit, take its response.
  Else, pick the first response.

$$\text{let}(x) \text{ where } x \in (M \mid \text{Delay})$$
Evaluation of $f$ can not be directly interrupted.

Introduce two sites:

- `Interrupt.set`: to interrupt $f$
- `Interrupt.get`: responds after `Interrupt.set` has been called.

Instead of $f$, evaluate

$$let(z) \text{ where } z : \in (f \mid Interrupt.get)$$
Sites $M$ and $N$ return booleans. Compute their parallel or.

$\text{ift}(b) \triangleq \text{if}(b) \Rightarrow \text{let(true)}$: returns true if $b$ is true; silent otherwise.

\[
\begin{align*}
\text{ift}(x) & \mid \text{ift}(y) \mid \text{or}(x,y) \\
\text{where} \quad & \quad x \in M, \ y \in N
\end{align*}
\]

To return just one value:

\[
\begin{align*}
\text{let}(z) \\
\text{where} \\
\quad z \in \text{ift}(x) \mid \text{ift}(y) \mid \text{or}(x,y) \\
\quad x \in M \\
\quad y \in N
\end{align*}
\]
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.

$\text{threshold}(x)$ returns $x$ if $x < c$; silent otherwise.

$\text{Min}(x, y)$ returns the minimum of $x$ and $y$.

\begin{verbatim}
let(z)
where
  z :∈ \text{threshold}(x) | \text{threshold}(y) | \text{Min}(x, y)
  x :∈ A
  y :∈ B
\end{verbatim}
Sequential Computing

- \((S; T) \text{ is } (S \triangleright T)\)

- \textbf{if } b \textbf{ then } S \textbf{ else } T
  
ak is
  
  \[
  \text{if}(b) \triangleright S \mid \text{if}(\neg b) \triangleright T
  \]

- \textbf{while } B(x) \textbf{ do } x:= S(x)

  \[
  \text{loop}(x) \triangleq B(x) > b > (\text{if}(b) \triangleright S(x) > y > \text{loop}(y) \mid \text{if}(\neg b) \triangleright \text{let}(x))
  \]
Angelica vs. Demonic non-determinism

- For all $x$ from $f$ do $g$: implements angelic non-determinism. All paths of computation are explored.

- For some $x$ from $f$ do $g$: implements demonic non-determinism. Some selected path of computation is explored.
Backtracking: Eight queens

Figure 2: Backtrack Search for Eight queens
Eight queens; contd.

- **configuration**: placement of queens in the last $i$ rows. Represented by a list of $i$ values from 0..7

- **Valid configuration**: no queen captures another.

  $valid(z)$ returns $z$ if configuration $z$ is valid; silent otherwise.

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

  \[
  \begin{align*}
  & extend(z, 1) \triangleq valid(0:z) | valid(1:z) | \cdots | valid(7:z) \\
  & extend(z, n) \triangleq extend(z, 1) > y > extend(y, n - 1)
  \end{align*}
  \]

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

- Processes typically communicate via channels.
- For channel \( c \), treat \( c\cdot put \) and \( c\cdot get \) as site calls.
- In our examples, \( c\cdot get \) is blocking and \( c\cdot put \) 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 $e$:

$$P(c, e) \triangleq c.get \; > x > \; \text{Compute}(x) \; > y > \; e.put(y) \; \Rightarrow P(c, e)$$

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

$$\text{Net}(c, d, e) \triangleq P(c, e) \; | \; P(d, e)$$
Run a dialog with a child.

**Forever:** child inputs an integer on channel $p$

Process outputs $true$ on channel $q$ iff the number is prime.

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

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

$$ Dialog(p, q) \triangleq \begin{align*} & p.get \quad > x > \\ & Prime?(x) \quad > b > \\ & q.put(b) \quad \Rightarrow \\ & Dialog(p, q) \end{align*} $$
Laws of Kleene Algebra

(Zero and \( \mid \) )
(Commutativity of \( \mid \) )
(Associativity of \( \mid \) )
(Idempotence of \( \mid \) )
(Associativity of \( \gg \) )
(Left zero of \( \gg \) )
(Right zero of \( \gg \) )
(Left unit of \( \gg \) )
(Right unit of \( \gg \) )
(Left Distributivity of \( \gg \) over \( \mid \) )
(Right Distributivity of \( \gg \) over \( \mid \) )

\[
\begin{align*}
    f \mid 0 &= f \\
    f \mid g &= g \mid f \\
    (f \mid g) \mid h &= f \mid (g \mid h) \\
    f \mid f &= f \\
    (f \gg g) \gg h &= f \gg (g \gg h) \\
    0 \gg f &= 0 \\
    f \gg 0 &= 0 \\
    Signal \gg f &= f \\
    f \gg x \gg let(x) &= f \\
    f \gg (g \mid h) &= (f \gg g) \mid (f \gg h) \\
    (f \mid g) \gg h &= (f \gg h) \mid (g \gg h)
\end{align*}
\]
Laws which do not hold

(Idempotence of \( | \)) \[ f | f = f \]
(Right zero of \( \gg \)) \[ f \gg 0 = 0 \]
(Left Distributivity of \( \gg \) over \( | \)) \[ f \gg (g | h) = (f \gg g) | (f \gg h) \]
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
\[
(f | g \text{ where } x \in h) = (f \text{ where } x \in h) | g
\]

(Distributivity over where) if \( g \) is \( y \)-free
\[
((f \text{ where } x \in g) \text{ where } y \in h)
= ((f \text{ where } y \in h) \text{ where } x \in g)
\]

(Elimination of where) if \( f \) is \( x \)-free, for site \( M \)
\[
(f \text{ where } x \in M) = f | M \gg 0
\]
Rules for Site Call

\[
\begin{align*}
\begin{array}{c}
k \text{ fresh} \\
\hline
M(v) \quad M_k(v) \quad ?k
\end{array}
\end{align*}
\]

\begin{align*}
?k & \overset{k?v}{\rightarrow} \text{let}(v) \\
\text{let}(v) & \overset{\downarrow v}{\rightarrow} 0
\end{align*}

(SITECALL)

(SITERET)

(LET)
Symmetric Composition

\[
\frac{f \xrightarrow{a} f'}{f \; | \; g \xrightarrow{a} f' \; | \; g} \quad (\text{SYM1})
\]

\[
\frac{g \xrightarrow{a} g'}{f \; | \; g \xrightarrow{a} f \; | \; g'} \quad (\text{SYM2})
\]
### Sequencing

\[
\begin{align*}
  f & \xrightarrow{a} f' & a \neq !v \\
  f & \xrightarrow{a} f' & f' \xrightarrow{a} f'' \\
  f & \xrightarrow{\tau} (f' \xrightarrow{a} f'') & [v/x].g
\end{align*}
\]

- (SEQ1N)
- (SEQ1V)
Asymmetric Composition

\[ f \xrightarrow{a} f' \]

\[ f \text{ where } x \in g \xrightarrow{a} f' \text{ where } x \in g \]

\[ g \xrightarrow{!v} g' \]

\[ f \text{ where } x \in g \xrightarrow{\tau} [v/x].f \]

\[ g \xrightarrow{a} g' \quad a \neq !v \]

\[ f \text{ where } x \in g \xrightarrow{a} f \text{ where } x \in g' \]
Expression Call

\[
\frac{\left[ E(x) \triangle f \right] \in D}{E(p) \xrightarrow{\tau} [p/x].f}
\] (DEF)
**Rules**

\[ k \text{ fresh} \]
\[
\frac{M(v) \quad M_k(v)}{M(v) \xrightarrow{?k} ?k}
\]

\[
?k \xrightarrow{k?v} \text{let}(v)
\]

\[
\text{let}(v) \quad !v \xrightarrow{} 0
\]

\[
f \xrightarrow{a} f' \quad f | g \xrightarrow{a} f' | g
\]

\[
g \xrightarrow{a} g' \quad f | g \xrightarrow{a} f | g'
\]

\[
[[E(x) \downarrow f]] \in D \quad E(p) \xrightarrow{\tau} [p/x].f
\]

\[
f \xrightarrow{a} f' \quad a \neq !v
\]

\[
f >x> g \xrightarrow{a} f' >x> g
\]

\[
f \xrightarrow{!v} f'
\]

\[
f >x> g \xrightarrow{\tau} (f' >x> g) \mid [v/x].g
\]

\[
f \xrightarrow{a} f' \quad f \xrightarrow{a} f' \quad f \xrightarrow{\tau} [v/x].f
\]

\[
g \xrightarrow{!v} g'
\]

\[
f \xrightarrow{a} f' \quad a \neq !v
\]

\[
f \xrightarrow{a} f' \quad f \xrightarrow{a} f \quad f \xrightarrow{a} f \quad f \xrightarrow{a} f
\]
Example

\[(\text{let}(x) \mid M(x)) > y > R(y)\] where \(x \in (N \mid S)\)

\[\frac{S_k}{\text{Call } S : S \xrightarrow{S_k} ?k ; N \mid S \xrightarrow{S_k} N \mid ?k}\]

\[(\text{let}(x) \mid M(x)) > y > R(y)\] where \(x \in (N \mid ?k)\)

\[\frac{N_l}{\text{Call } N}\]

\[(\text{let}(x) \mid M(x)) > y > R(y)\] where \(x \in (?l \mid ?k)\)

\[\frac{l?5}{\text{let}(5)} \xrightarrow{l?5} \text{let}(5) ; ?l \mid ?k \xrightarrow{l?5} \text{let}(5) \mid ?k}\]

\[(\text{let}(x) \mid M(x)) > y > R(y)\] where \(x \in (\text{let}(5) \mid ?k)\)
Example; contd.

\[ ((M(x) \mid let(x)) \succ y \succ R(y)) \text{ where } x : \in (let(5) \mid ?k) \]

\[ \xrightarrow{\tau} \{ \text{ let}(5) \xrightarrow{15} 0 ; \text{ let}(5) \mid ?k \xrightarrow{15} 0 \mid ?k \} \]

\[ (M(5) \mid let(5)) \succ y \succ R(y) \]

\[ \xrightarrow{\tau} \{ \text{ let}(5) \xrightarrow{15} 0 ; \text{ M}(5) \mid let(5) \xrightarrow{15} M(5) \mid 0 ; \]

\[ f \xrightarrow{uv} f' \text{ implies } f \succ y \succ g \xrightarrow{\tau} (f' \succ y \succ g) \mid [v/y].g \} \]

\[ ((M(5) \mid 0) \succ y \succ R(y)) \mid R(5) \]

\[ R_n(5) \xrightarrow{\{ \text{ call } R \text{ with argument (5)} \}} \]

\[ ((M(5) \mid 0) \succ y \succ R(y)) \mid ?n \]
Example; contd.

\[
((M(5) \mid 0) > y > R(y)) \mid ?n
\]
\[n\tau\{ ?n \xrightarrow{\tau} \text{let}(7) \}\]

\[
((M(5) \mid 0) > y > R(y)) \mid \text{let}(7)
\]
\[!\tau\{ f \mid \text{let}(7) \xrightarrow{!\tau} f \mid 0 \}\]

\[
((M(5) \mid 0) > y > R(y)) \mid 0
\]

The sequence of events: \( S_k \ N_i \ l?5 \ \tau \ \tau \ R_n(5) \ n?7 \ !7 \)

The sequence minus \( \tau \) events: \( S_k \ N_i \ l?5 \ R_n(5) \ n?7 \ !7 \)
## Executions and Traces

Define

\[ f \xrightarrow{\epsilon} f \]

\[ f \xrightarrow{a} f'', f'' \xrightarrow{s} f' \]

\[ f \xrightarrow{a \cdot s} f' \]

- Given \( f \xrightarrow{s} f' \), \( s \) is an execution of \( f \).

- A trace is an execution minus \( \tau \) events.

- The set of executions of \( f \) (and traces) are prefix-closed.
Laws, using strong bisimulation

- \( f \mid 0 \sim f \)
- \( f \mid g \sim g \mid f \)
- \( f \mid (g \mid h) \sim (f \mid g) \mid h \)
- \( f \rhd x \rhd (g \rhd y \rhd h) \sim (f \rhd x \rhd g) \rhd y \rhd h \), if \( h \) is \( x \)-free.
- \( 0 \rhd x \rhd f \sim 0 \)
- \( (f \mid g) \rhd x \rhd h \sim f \rhd x \rhd h \mid g \rhd x \rhd h \)
- \( (f \mid g) \) where \( x : \in h \sim (f \) where \( x : \in h) \mid g \), if \( g \) is \( x \)-free.
- \( (f \rhd y \rhd g) \) where \( x : \in h \sim (f \) where \( x : \in h) \rhd y \rhd g \), if \( g \) is \( x \)-free.
- \( (f \) where \( x : \in g) \) where \( y : \in h \sim (f \) where \( y : \in h) \) where \( x : \in g \), if \( g \) is \( y \)-free, \( h \) is \( x \)-free.
Relation $\sim$ is an equality

Given $f \sim g$, show

1. $f \mid h \sim g \mid h$
   $h \mid f \sim h \mid g$

2. $f \succ x \succ h \sim g \succ x \succ h$
   $h \succ x \succ f \sim h \succ x \succ g$

3. $f$ where $x: \in h \sim g$ where $x: \in h$
   $h$ where $x: \in f \sim h$ where $x: \in g$
Treatment of Free Variables

Closed expression: No free variable.
Open expression: Has free variable.

- Law $f \sim g$ holds only if both $f$ and $g$ are closed.
  
  Otherwise: $let(x) \sim 0$
  
  But $let(1) > x > 0 \neq let(1) > x > let(x)$

- Then we can’t show $let(x) \; | \; let(y) \sim let(y) \; | \; let(x)$
Substitution Event

\[ f \xrightarrow{[v/x]} [v/x].f \quad \text{(SUBST)} \]

- Now, \( \text{let}(x) \xrightarrow{[1/x]} \text{let}(1) \).
  
  So, \( \text{let}(x) \neq 0 \)

- Earlier rules apply to base events only.

From \( f \xrightarrow{[v/x]} [v/x].f \), we can not conclude:

\[ f \parallel g \xrightarrow{[v/x]} [v/x].f \parallel g \]
Traces as Denotations

Define Orc combinators over trace sets, $S$ and $T$. Define:

$$S \mid T, \quad S \triangleright x \triangleright T, \quad S \text{ where } x \in T.$$ 

Notation: $\langle f \rangle$ is the set of traces of $f$.

Theorem

$$\langle f \mid g \rangle = \langle f \rangle \mid \langle g \rangle$$
$$\langle f \triangleright x \triangleright g \rangle = \langle f \rangle \triangleright x \triangleright \langle g \rangle$$
$$\langle f \text{ where } x \in g \rangle = \langle f \rangle \text{ where } x \in \langle g \rangle$$
Expressions are equal if their trace sets are equal

Define: \( f \equiv g \) if \( \langle f \rangle = \langle g \rangle \).

**Theorem** (Combinators preserve \( \equiv \))

Given \( f \equiv g \) and any combinator \( *: f * h \equiv g * h, \ h * f \equiv h * g \)

Specifically, given \( f \equiv g \)

1. \( f \mid h \equiv g \mid h \)
   \( h \mid f \equiv h \mid g \)

2. \( f \triangleright x \triangleright h \equiv g \triangleright x \triangleright h \)
   \( h \triangleright x \triangleright f \equiv h \triangleright x \triangleright g \)

3. \( f \text{ where } x : \in h \equiv g \text{ where } x : \in h \)
   \( h \text{ where } x : \in f \equiv h \text{ where } x : \in g \)
Monotonicity, Continuity

- Define: $f \sqsubseteq g$ if $\langle f \rangle \subseteq \langle g \rangle$.

Theorem (Monotonicity) Given $f \sqsubseteq g$ and any combinator $*$

$$f * h \sqsubseteq g * h, \ h * f \sqsubseteq h * g$$

- Chain $f : f_0 \sqsubseteq f_1, \cdots f_i \sqsubseteq f_{i+1}, \cdots$.

Theorem: $\bigsqcup (f_i * h) \cong (\bigsqcup f) * h$.

Theorem: $\bigsqcup (h * f_i) \cong h * (\bigsqcup f)$.
Least Fixed Point

\[ M \triangleq S \mid R \gg M \]

\[ M_0 \cong 0 \]
\[ M_{i+1} \cong S \mid R \gg M_i, \quad i \geq 0 \]

\( M \) is the least upper bound of the chain \( M_0 \sqsubseteq M_1 \sqsubseteq \cdots \)
Weak Bisimulation

\[
\begin{align*}
\text{signal} & \gg f \quad \models \quad f \\
\text{f} & \gg x \gg \text{let}(x) \quad \models \quad f
\end{align*}
\]