Structured Concurrent Programming

William Cook, David Kitchin, Jayadev Misra, Adrian Quark

Department of Computer Science
University of Texas at Austin

http://orc.csres.utexas.edu

Outline

Structured Concurrent Programming

- Structured Sequential Programming: Dijkstra circa 1968
- Structured Concurrent Programming:
  - Fundamental combinators for concurrency
  - A paradigm for constructing concurrent and distributed programs
  - Component Integration and Orchestration

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.
...

Overview of Orc, an Orchestration Theory

- Orc program has
  - a goal expression,
  - a set of definitions.
- A Program execution evaluates the goal. It
  - calls sites, to invoke services,
  - publishes values.
- Orc is simple
  - Language has only 3 combinators to form expressions.
  - Can handle time-outs, priorities, failures, synchronizations, $\cdots$

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	$f \mid g$	Symmetric composition
for all x from f do g	$f >x> g$	Sequential composition
for some x from g do f	$f <x< g$	Pruning

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	$f \mid g$	Symmetric composition
for all x from f do g	$f > x > g$	Sequential composition
for some x from g do f	$f < x < g$	Pruning

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	$f \mid g$	Symmetric composition
for all x from f do g	$f > x > g$	Sequential composition
for some x from g do f	$f < x < g$	Pruning

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	$f \mid g$	Symmetric composition
for all x from f do g	$f > x > g$	Sequential composition
for some x from g do f	$f < x < g$	Pruning

Sites

- External Services: Google spell checker, Google Search, MySpace, CNN, Discovery ...
- Any Java Class instance
- Library sites
  - $+ \; - \; * \; \&\& \; ||$ ...
  - `println`, `random`, `prompt`, `Mail`
  - `if`
  - `Rtimer`
  - `storage`, `semaphore`, `MakeChannel`

  ...

Symmetric composition: $f \mid g$

- Evaluate f and g independently.
- Publish all values from both.
- No direct communication or interaction between f and g. They can communicate only through sites.

Examples

- $CNN(d) \mid BBC(d)$: calls both CNN and BBC simultaneously. Publishes values returned by both sites. (0, 1 or 2 values)
- $WebServer() \mid MailServer() \mid LinuxServer()$
  A System Configuration

-

Sequential composition: $f \; >x> \; g$

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

- $CNN(d) \; >x> \; Email(address, x)$
  - Call $CNN(d)$.
  - Bind result (if any) to x.
  - Call $Email(address, x)$.
  - Publish the value, if any, returned by $Email$.

- $(CNN(d) \mid BBC(d)) \; >x> \; Email(address, x)$
  - May call $Email$ twice.
  - Publishes up to two values from $Email$.

-

Schematic of Sequential composition

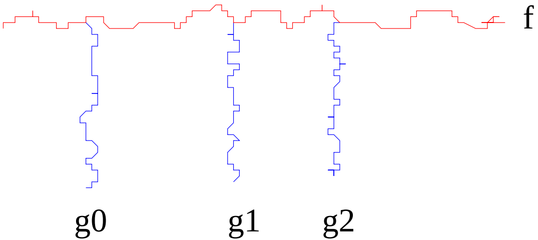


Figure: Schematic of $f\ {>}x{>}\ g$

Pruning: $(f \ <x< \ g)$

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

- Evaluate f and g in parallel.
  - Site calls that need x are suspended.
  - Other site calls proceed.
  - see $(M() \mid N(x)) \ <x< \ g$
- When g returns a value:
  - Assign it to x.
  - Terminate g.
  - Resume suspended calls.
- Values published by f are the values of $(f \ <x< \ g)$.

Example of Pruning

$$Email(address, x) \;\; {<}x{<} \; (CNN(d) \; | \; BBC(d))$$

Binds x to the first value from $CNN(d) \; | \; BBC(d)$.
Sends at most one email.

Some Fundamental Sites

- $if(b)$: boolean b,
  returns a signal if b is true; remains silent if b is false.
- $stop$: never responds. Same as $if(false)$.
- $Rtimer(t)$: integer t, $t \geq 0$, returns a signal t time units later.
- $signal()$ returns a signal immediately. Same as $if(true)$.

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.
- Round-based Execution.

Time-out

Publish M's response if it arrives before time t,
Otherwise, publish 0.

$$z \ <z< \ (M() \ \mid (Rtimer(t) \ \gg 0))$$

Fork-join parallelism

Call M and N in parallel.
Return their values as a tuple after both respond.

$$((u, v) \\ \quad <u< M()) \\ \quad\quad <v< N()$$

Notational Convention: $<u<$ is left-associative.

$(u, v) \;\; <u< M() \;\; <v< N()$, or

$$(u, v) \\ \quad <u< M() \\ \quad <v< N()$$

Expression Definition

$$\mathit{def}\ \ \mathit{MailOnce}(a) = \\ \quad \mathit{Email}(a, m)\ \ {<}m{<}\ (\mathit{CNN}(d)\ \mid \mathit{BBC}(d))$$

$$\mathit{def}\ \ \mathit{MailLoop}(a, d) = \\ \quad \mathit{MailOnce}(a)\ \gg \mathit{Rtimer}(d)\ \gg \mathit{MailLoop}(a, d)$$

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

Expression Definition

- output n signals -

$def \;\; signals(n) = if(n > 0) \gg (signal \mid signals(n-1))$

- Publish a signal at every time unit.-

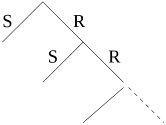
$def \;\; metronome() = signal \mid (Rtimer(1) \gg metronome())$

- Publish a signal every t time units.-

$def \;\; tmetronome(t) = signal \mid (Rtimer(t) \gg tmetronome(t))$

- Publish natural numbers from i every t time units.-

$def \;\; gen(i, t) = i \mid Rtimer(t) \gg gen(i+1, t)$



Recursive definition with time-out

Call a list of sites.
Count the number of responses received within 10 time units.

$$
\begin{array}{l}
def\ \ tally([\,]) = 0 \\
def\ \ tally(M : MS) = u + v \\
\qquad\qquad\qquad\qquad <u<\ (M() \gg 1) \mid (Rtimer(10) \gg 0) \\
\qquad\qquad\qquad\qquad <v<\ tally(MS)
\end{array}
$$

or, even better,

$$
\begin{array}{l}
def\ \ tally([\,]) = 0 \\
def\ \ tally(M : MS) = (M() \gg 1 \mid Rtimer(10) \gg 0) + tally(MS)
\end{array}
$$

Barrier Synchronization in $M \gg f \quad | \quad N \gg g$

f and g start only after both M and N complete.
Rendezvous of CSP or CCS; M and N are complementary actions.

$$
\begin{array}{l}
\quad ((u, v) \\
\qquad <u< M() \\
\qquad <v< N()) \\
\gg (f \mid g)
\end{array}
$$

Priority

- Publish N's response asap, but no earlier than 1 unit from now.
  Apply fork-join between $Rtimer(1)$ and N.

  $$def \;\; Delay() = (Rtimer(1) \gg u) \; <u< N()$$

- Call M, N together.
  If M responds within one unit, publish its response.
  Else, publish the first response.

  $$x \; <x< (M() \; | \; Delay())$$

Interrupt f

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

$$z \ {<}z{<}\ (f \ \mid \ Interrupt.get())$$

Parallel or

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

$$
\begin{array}{l}
if(x) \gg true \mid if(y) \gg true \mid or(x,y) \\
\quad <x< M() \\
\quad <y< N()
\end{array}
$$

To return just one value:

$$
\begin{array}{l}
z \\
\quad <z< if(x) \gg true \mid if(y) \gg true \mid or(x,y) \\
\quad <x< M() \\
\quad <y< N()
\end{array}
$$

-

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

$$
\begin{array}{l}
z \\
\qquad < z < \mathit{threshold}(x) \mid \mathit{threshold}(y) \mid \mathit{Min}(x,y) \\
\qquad < x < A() \\
\qquad < y < B()
\end{array}
$$

Backtracking: Eight queens

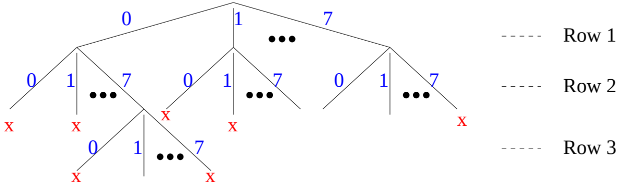


Figure: Backtrack Search for Eight queens

Eight queens; contd.

$$def\ \ extend(z,1) = valid(0{:}z)\ \mid valid(1{:}z)\ \mid \cdots\ \mid valid(7{:}z)$$

$$def\ \ extend(z,n) = extend(z,1)\ \ {>}y{>}\ extend(y, n-1)$$

- z: partial placement of queens (list of values from $0..7$)
- $extend(z,n)$ publishes all valid extensions of z with n additional queens.
- $valid(z)$ returns z if z is valid; silent otherwise.
- Solve the original problem by calling $extend([\,], 8)$.

Network of Services: Insurance Company

$$def \;\; insurance() = apply() \mid join() \mid payment()$$

$$def \;\; apply() = inApply.get() \; >x> quote(x) \; >y> Email(x.addr, y) \gg apply()$$

$$def \;\; join() = inJoin.get() \; >(id,p)> validate(id,p) \gg (\; add\_client(id,p) \gg Email(id.addr, welcome) \mid renew(id) \;) \gg join()$$

$$def \;\; payment() = inPayment.get() \; >(id,p)> validate(id,p) \gg update\_client(id,p) \gg payment()$$

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:

$$def\ \ P(c,e) = \ c.get \ \ {>}x{>} \ Compute(x) \ \ {>}y{>} \ e.put(y) \ \ \gg P(c,e)$$

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

$$def\ \ Net(c,d,e) = \ P(c,e) \ \mid P(d,e)$$

Interaction: Run a dialog

Prompt the user to input an integer.
Print *true* iff the number is prime. Loop forever.
Site $Prime?(x)$ returns *true* iff x is prime.

$$
\begin{array}{l}
def\ \ Dialog() = \\
\quad Prompt(\text{" input an integer "}) \quad >x> \\
\quad Prime?(x) \quad >b> \\
\quad println(b) \quad \gg \\
\quad Dialog()
\end{array}
$$

Laws of Kleene Algebra

(Zero and $\mid$)	$f \mid 0 = f$
(Commutativity of $\mid$)	$f \mid g = g \mid f$
(Associativity of $\mid$)	$(f \mid g) \mid h = f \mid (g \mid h)$
(Idempotence of $\mid$)	$f \mid f = f$
(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 $\mid$)	$f \gg (g \mid h) = (f \gg g) \mid (f \gg h)$
(Right Distributivity of $\gg$ over $\mid$)	$(f \mid g) \gg h = (f \gg h \mid g \gg h)$

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

$$((f \gg g) < x < h) = (f < x < h) \gg g$$

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

$$((f \mid g) < x < h) = (f < x < h) \mid g$$

(Distributivity over $<<$) if g is y-free

$$\begin{aligned} & ((f < x < g) < y < h) \\ = \quad & ((f < y < h) < x < g) \end{aligned}$$

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

$$(f < x < M) = f \mid (M \gg 0)$$

Shortest path problem

- Directed graph; non-negative weights on edges.
- Find shortest path from source to sink.

We calculate just the length of the shortest path.

Algorithm with Lights and Mirrors

- Source node sends rays of light to each neighbor.
- Edge weight is the time for the ray to traverse the edge.
- When a node receives its first ray, sends rays to all neighbors. Ignores subsequent rays.
- Shortest path length = time for sink to receive its first ray.

Expressions and Sites needed for Shortest path

$succ(u)$: Publish all (v, d), where edge (u, v) has weight d.

$write(u, t)$: Write value t for node u. If already written, block.

$read(u)$: Return value for node u. If unwritten, block.

First Algorithm

$$
\begin{aligned}
\mathit{def}\;\; eval1(u,t) = \quad & write(u,t) \gg \\
& Succ(u) \;>(v,d)> \\
& Rtimer(d) \gg \\
& eval1(v, t+d)
\end{aligned}
$$

$$\mathit{Goal}: \quad eval1(source, 0) \;|\; read(sink)$$

First call to $eval1(u,t)$:

- The relative time in the evaluation is t.
- Length of the shortest path from source to u is t.
- $eval1$ does not publish.

Research Agenda

Establish Orc as a fundamental paradigm of concurrent and distributed computing.

- Transaction Processing
- Virtual Time and Simulation
- Distributed Implementation
- Verification
- High assurance workflow and Security
- Adaptive workflow
- Large system design using component integration
- Analysis tools