Structured Concurrent Programming

William Cook, David Kitchin, Jayadev Misra, Adrian Quark, Ian Wehrman

Department of Computer Science
University of Texas at Austin

CS398T,
Sept. 26, 2008

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	$f < x < g$	Asymmetric composition

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	$f <x< g$	Asymmetric composition

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	$f <x< g$	Asymmetric composition

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	$f <x< g$	Asymmetric composition

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

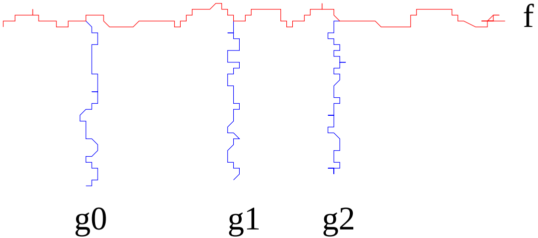


Figure: Schematic of f $>x>$ g

Asymmetric composition: $(f \ <x< \ g)$

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 Asymmetric composition

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

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

-

Time-out

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

$$z \; <z< \; (M() \; | \; (Rtimer(t) \gg 0))$$

Fork-join parallelism

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

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

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

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

$$
\begin{array}{l}
(u, v) \\
\qquad < u < M() \\
\qquad < v < N()
\end{array}
$$

Expression Definition

- output n signals -

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

- Publish a signal at every time unit.-

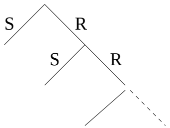
$def\ \ metronome() = signal\ \mid (Rtimer(1) \gg metronome())$

- 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.

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

or, even better,

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

Parallel or

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

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

To return just one value:

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

-

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.

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

Backtracking: Eight queens

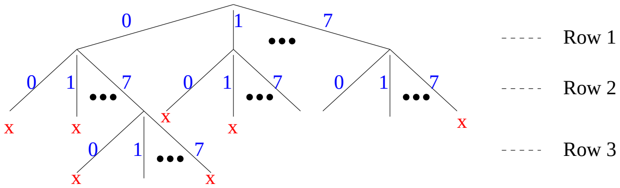


Figure: Backtrack Search for Eight queens

Eight queens; contd.

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

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

- 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

$$\mathit{def}\ \ insurance = apply \mid join \mid payment$$

$$\begin{aligned}\mathit{def}\ \ apply = \ & inApply.get \ >x>\ quote(x)\ >y>\ Email(x.addr, y) \gg \\ & apply\end{aligned}$$

$$\begin{aligned}\mathit{def}\ \ join = \ & inJoin.get \ >(id,p)>\ validate(id,p) \gg \\ & (\ add\_client(id,p) \gg Email(id.addr, welcome) \\ & \ \mid renew(id) \\ &) \gg \\ & join\end{aligned}$$

$$\begin{aligned}\mathit{def}\ \ payment = \ & inPayment.get \ >(id,p)>\ validate(id,p) \gg \\ & update\_client(id,p) \gg \\ & payment\end{aligned}$$

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