Simulations in the Orc Programming Language

Jayadev Misra

Department of Computer Science
University of Texas at Austin

ICTAC 2011

Simulation as Concurrent Programming

- A simulation description is a real-time concurrent program.
- The concurrent program includes physical entities and their interactions.
- The concurrent program specifies the time interval for activities.

Features needed in the Concurrent Programming Language

- Describe entities and their interactions.
- Describe passage of time.
- Allow birth and death of entities.
- Allow programming novel interactions.
- Support hierarchical structure.

Orc

- Goal: Internet scripting language.
- Next: Component integration language.
- Next: A general purpose, structured “concurrent programming language”.
- A very late realization: A simulation language.

Internet Scripting

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

Orc Basics

- Site: Basic service or component.
- Concurrency combinators for integrating sites.
- Theory includes nothing other than the combinators.

No notion of data type, thread, process, channel, synchronization, parallelism $\cdots$

New concepts are programmed using the combinators.

Examples of Sites

- $+ \quad - \quad * \quad \&\& \quad || \quad =$ ...
- `Println`, `Random`, `Prompt`, `Email` ...
- Mutable Ref, Semaphore, Channel, ...
- `Timer`
- External Services: Google Search, MySpace, CNN, ...
- Any Java Class instance, Any Orc Program
- Factory sites; Sites that create sites: `Semaphore`, `Channel` ...
- Humans

  ...

Sites

- A site is called like a procedure with parameters.
- Site returns at most one value.
- The value is published.

Site calls are strict.

Overview of Orc

- Orc program has
  - a goal expression,
  - a set of definitions.
- The goal expression is executed. Its execution
  - calls sites,
  - publishes values.

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

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.

  Example: $CNN(d) \mid BBC(d)$

  calls both CNN and BBC simultaneously.
  Publishes values returned by both sites. (0, 1 or 2 values)

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 $Email$.

Notation: $f \gg g$ for $f \; >x> \; g$, if x unused in g.

Schematic of Sequential composition

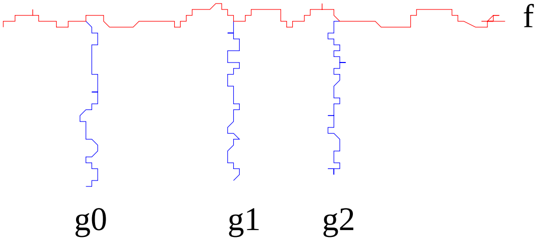

Figure: Schematic of f $>x>$ g

Pruning: $(f \ \texttt{<x<} \ g)$

For some value published by g do f.

- Evaluate f and g in parallel.
  - Site calls that need x are suspended.
  - see $(M() \mid N(x)) \ \texttt{<x<} \ g$
- When g returns a (first) value:
  - Bind the value to x.
  - Terminate g.
  - Resume suspended calls.
- Values published by f are the values of $(f \ \texttt{<x<} \ g)$.

Example of Pruning

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

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

Some Fundamental Sites

- $Ift(b)$, $Iff(b)$: boolean b.
  Returns a signal if b is true/false;
  remains silent otherwise.

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

- *stop* : never responds. Same as $Ift(false)$ or $Iff(true)$.

- *signal* : returns a signal immediately.
  Same as $Ift(true)$ or $Iff(false)$.

Expression Definition

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

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

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

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

Functional Core Language

- Data Types: Number, Boolean, String, with usual operators
- Conditional Expression: if E then F else G
- Data structures: Tuple and List
- Pattern Matching
- Function Definition; Closure

Variable Binding; Silent expression

$val\ \ x = \ 1 + 2$

$val\ \ y = \ x + x$

$val\ \ z = \ x/0$ - - expression is silent

$val\ \ u = \ \text{if } (0 < 5) \text{ then } 0 \text{ else } z$

Comingling with Orc expressions

Components of Orc expression could be functional.
Components of functional expression could be Orc.

$$(1 + 2) \mid (2 + 3)$$

$$(1 \mid 2) + (2 \mid 3)$$

Convention

Whenever expression F appears in context C where a single value is expected from F, convert it to $C[x]$ $<x<$ F.

$1 + 2 \mid 2 + 3$ is $add(1, 2) \mid add(2, 3)$

$(1 \mid 2) + (2 \mid 3)$ is $(add(x, y)$ $<x<$ $(1 \mid 2))$ $<y<$ $(2 \mid 3)$

Implication:
Arguments of site calls are evaluated in parallel.
Site is called when all arguments have been evaluated.

Example: Fibonacci numbers

$$\mathit{def}\ \ H(0) = (1,1)$$
$$\mathit{def}\ \ H(n) = H(n-1)\ >(x,y)>\ (y, x+y)$$

$$\mathit{def}\ \ \mathit{Fib}(n) = H(n)\ >(x,\_)>\ x$$

{- Goal expression -}

$\mathit{Fib}(5)$

Some Typical Applications

- Adaptive Workflow (Business process management):
  Workflow lasting over months or years
  Security, Failure, Long-lived Data

- Extended 911:
  Using humans as components
  Components join and leave
  Real-time response

- Network simulation:
  Experiments with differing traffic and failure modes
  Animation

Some Typical Applications, contd.

- Grid Computations
- Music Composition
- Traffic simulation
- Computation Animation

Some Typical Applications, contd.

- Map-Reduce using a server farm
- Thread management in an operating system
- Mashups (Internet Scripting).
- Concurrent Programming on Android.

Timeout

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

$z \;{<}z{<}\; (M() \mid (Rwait(t) \gg 0))$, or

$val\;\; z = M() \mid (Rwait(t) \gg 0)$
z

Fork-join parallelism

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

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

or,

$$(M(), N())$$

Simple Parallel Auction

- A list of bidders in a sealed-bid, single-round auction.
- $b.ask()$ requests a bid from bidder b.
- Ask for bids from all bidders, then publish the highest bid.

$$
\begin{aligned}
&def \;\; auction([\,]) = 0 \\
&def \;\; auction(b : bs) = max(b.ask(), auction(bs))
\end{aligned}
$$

Notes:

- All bidders are called simultaneously.
- If some bidder fails, then the auction will never complete.

Parallel Auction with Timeout

- Take a bid to be 0 if no response is received from the bidder within 8 seconds.

$$
\begin{array}{l}
\mathit{def}\ \ \mathit{auction}([\,]) = 0 \\
\\
\mathit{def}\ \ \mathit{auction}(b : bs) = \\
\quad \mathit{max}(\\
\qquad b.\mathit{ask}() \mid (\mathit{Rwait}(8000) \gg 0), \\
\qquad \mathit{auction}(bs) \\
\quad)
\end{array}
$$

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.

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. Shortest path length to node i = time for i to receive its first ray.

Graph structure in function $Succ()$

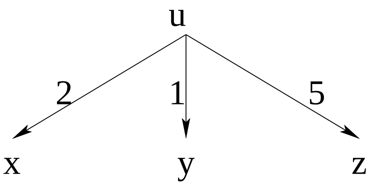

Figure: Graph Structure

$Succ(u)$ publishes $(x, 2)$, $(y, 1)$, $(z, 5)$.

Algorithm

$def\ \ eval(u, t) =$ record value t for u $\gg$
for every successor v with d = length of (u, v) :
wait for d time units $\gg$
$eval(v, t + d)$

$Goal$: $eval(source, 0)$ |
read the value recorded for the $sink$

Record path lengths for node u in FIFO channel u.

Algorithm(contd.)

$def \;\; eval(u,t) =$ record value t for u $\gg$
for every successor v with d = length of (u,v) :
wait for d time units $\gg$
$eval(v, t+d)$

$Goal:$ $eval(source, 0)$ $|$
read the value recorded for the $sink$

$def \;\; eval(u,t) =$ $u.put(t) \gg$
$Succ(u) \; >(v,d)>$
$Rwait(d) \gg$
$eval(v, t+d)$

{- $Goal:$ -} $eval(source, 0) \mid sink.get()$

Algorithm(contd.)

$$\mathit{def}\;\; \mathit{eval}(u,t) = \quad u.put(t) \gg Succ(u) >(v,d)> Rwait(d) \gg eval(v, t+d)$$

{- *Goal* :-} $\quad eval(source, 0) \mid sink.get()$

- Any call to $eval(u,t)$: Length of a path from source to u is t.
- First call to $eval(u,t)$: Length of the shortest path from source to u is t.
- *eval* does not publish.

Drawbacks of this algorithm

- Running time proportional to shortest path length.
- Executions of *Succ*, *put* and *get* should take no time.

Virtual Timer

Methods:

$Vwait(t)$	Returns a signal after t virtual time units.
$Vtime()$	Returns the current value of the virtual timer.

Virtual timer Properties

- Virtual timer value is monotonic.
- $Vwait(t)$ consumes exactly t units of virtual time.
- A step is started as soon as possible in virtual time.
- Virtual timer is advanced only if there can be no other activity.

Implementing virtual timer

Data structures:

- n: current value of $Vtime()$, initially $n = 0$.
- q: queue of calls to $Vwait()$ whose responses are pending.

At run time:

- A call to $Vtime()$ immediately responds with n.
- A call to $Vwait(t)$ is assigned rank $n + t$ and queued.
- Progress: If the program is stuck, then:

  remove the item with the lowest rank r from q,
  set $n := r$,
  respond with a signal to the corresponding call to $Vwait()$.

Simulation: Bank

- Bank with two tellers and one queue for customers.
- Customers generated by a *source* process.
- When free, a teller serves the first customer in the queue.
- Service times vary for customers.
- Determine
  - Average wait time for a customer.
  - Queue length distribution.
  - Average idle time for a teller.

Structure of bounded simulation

Run the simulation for *simtime*.
Below, $Bank()$ never publishes .

$$val\ z = Bank() \mid Vwait(simtime)$$

$$z \gg Stats()$$

Description of Bank

$$
\begin{aligned}
\textit{def Bank}() &= (\textit{Customers}() \mid \textit{Teller}() \mid \textit{Teller}()) \gg \textit{stop} \\
\textit{def Customers}() &= \textit{Source}() \; >c> \; \textit{enter}(c) \\
\textit{def Teller}() &= \textit{next}() \; >c> \\
&\quad \textit{Vwait}(c.\textit{ServTime}) \gg \\
&\quad \textit{Teller}() \\
\textit{def enter}(c) &= q.\textit{put}(c) \\
\textit{def next}() &= q.\textit{get}()
\end{aligned}
$$

Fast Food Restaurant

- Restaurant with one cashier, two cooking stations and one queue for customers.
- Customers generated by a *source* process.
- When free, cashier serves the first customer in the queue.
- Cashier service times vary for customers.
- Cashier places the order in another queue for the cooking stations.
- Each order has 3 parts: main entree, side dish, drink
- A cooking station processes parts of an order in parallel.

Goal Expression for Restaurant Simulation

$$val\ \ z = \ Restaurant()(\mid Vwait(simtime))$$

$$z \ \gg \ Stats()$$

Description of Restaurant

```
def Restaurant()   =  (Customers() | Cashier() | Cook() | Cook()) ≫ stop

def Customers()    =  Source() >c> enter(c)

def Cashier()      =  next() >c>
                      Vwait(c.ringupTime) ≫
                      orders.put(c.order) ≫
                      Cashier()

def Cook()         =  orders.get() >order>
                      (
                        prepTime(order.entree) >t> Vwait(t),
                        prepTime(order.side)   >t> Vwait(t),
                        prepTime(order.drink)  >t> Vwait(t)
                      )  ≫  Cook()

def enter(c)       =  q.put(c)
def next()         =  q.get()
```

Collecting Statistics: waiting time

Change

$$\begin{array}{lcl} \mathit{def}\ \mathit{enter}(c) & = & q.put(c) \\ \mathit{def}\ \mathit{next}() & = & q.get() \end{array}$$

to

$$\begin{array}{lcl} \mathit{def}\ \mathit{enter}(c) & = & Vtime()\ >s>\ q.put(c, s) \\ & & \\ \mathit{def}\ \mathit{next}() & = & q.get()\ >(c,t)> \\ & & \quad Vtime()\ >s> \\ & & \quad reportWait(s - t)\ \gg \\ & & \quad c \end{array}$$