Structured Wide-Area Programming: Orc Programming Language

Jayadev Misra

Department of Computer Science
University of Texas at Austin

http://orc.csres.utexas.edu

Orc Language

- Data Types: Number, Boolean, String, with Java operators
- Conditional Expression: *if* E *then* F *else* G
- Data structures: Tuple, List, Record
- Pattern Matching; Clausal Definition
- Function Closure
- Comingling functional and Orc expressions

Data types

- Number: $5, \ -1, \ 2.71828, \ -2.71e-5$
- Boolean: *true*, *false*
- String: `"orc"`, `"ceci n'est pas une |"`

$1+2$	evaluates to	3
$0.4 = 2.0/5$	evaluates to	*true*
$3-5 :> 5-3$	evaluates to	*false*
true && (*false* || *true*)	evaluates to	*true*
$3/0$	is silent	
`"Try" + "Orc"`	evaluates to	`"TryOrc"`

Variable Binding

$val \;\; x = \;\; 1 + 2$

$val \;\; y = \;\; x + x$

$val \;\; z = \;\; x/0$ this expression is silent; other evaluations continue

$val \;\; u = \;\; if \;\; (0 <: 5) \; then \;\; 0 \; else \;\; z$

Conditional Expression

if	*true*	then	"blue"	else	"green"	—	is "blue"
if	"fish"	then	"yes"	else	"no"	—	is silent
if	*false*	then	4+5	else	4+true	—	is silent
if	*true*	then	0/5	else	5/0	—	is 0

Tuples

$(1+2,7)$	is	$(3,7)$
("true" + "false", $\mathit{true} \mid\mid \mathit{false}, \mathit{true}$ && false)	is	("truefalse", true, false)
(2/2, 2/1, 2/0)	is	silent

Lists

$[1, 2+3]$	is	$[1, 5]$
$[true \ \&\& \ \ true]$	is	$[true]$
$[\,]$	is	the empty list
$[5, 5+true, 5]$	is	silent

List Constructor is a colon :

$3{:}[5, 7] = [3, 5, 7]$

$3{:}[\,] = [3]$

Pattern Matching in val

(x,y) = (2+3,2*3)	binds	x to 5 and y to 6
[a,b] = ["one", "two"]	binds	a to "one", b to "two"
((a,b),c) = ((1, true), [2, false])	binds	a to 1, b to true, and c to [2, false]
(x,_,_) = (1,(2,2),[3,3,3])	binds	x to 1
[[_,x],[_,y]] = [[1,3],[2,4]]	binds	x to 3 and y to 4

Pattern Matching in Function Definition

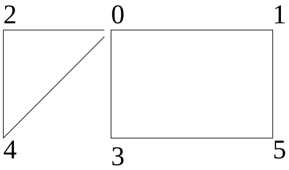
A function adds two pairs componentwise;
publishes the resulting pair.

$$def \;\; pairsum(a, b) = $$
$$a \;\; >(x, y)> \;\; b \;\; >(x', y')> \; (x + x', y + y')$$

or, even better,

$$def \;\; pairsum((x, y), (x', y')) = \; (x + x', y + y')$$

Clausal Definition, Pattern Matching
Example: Defining graph connectivity



An Undirected Graph

$$
\begin{aligned}
&\textit{def} \;\; conn(0) = [1, 2, 3, 4]\\
&\textit{def} \;\; conn(1) = [0, 5]\\
&\textit{def} \;\; conn(2) = [0, 4]\\
&\textit{def} \;\; conn(3) = [0, 5]\\
&\textit{def} \;\; conn(4) = [0, 2]\\
&\textit{def} \;\; conn(5) = [1, 3]
\end{aligned}
$$

$$
\begin{aligned}
&\textit{def} \;\; conn(i) =\\
&\quad\;\; i > 0 > [1, 2, 3, 4]\\
&\quad \mid i > 1 > [0, 5]\\
&\quad \mid i > 2 > [0, 4]\\
&\quad \mid i > 3 > [0, 5]\\
&\quad \mid i > 4 > [0, 2]\\
&\quad \mid i > 5 > [1, 3]
\end{aligned}
$$

Clausal definition of a function
Example: Fibonacci numbers

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

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

{- Goal expression -}
$\textit{Fib}(5)$

Closure: Functions as values

$val\ \ minmax = \ (min, max)$

========================

$def\ \ apply2((f,g),(x,y)) = \ (f(x,y), g(x,y))$

$apply2(minmax, (2,1))$ publishes (1, 2)

========================

$def\ \ pmap(f, []) = \ []$

$def\ \ pmap(f, x : xs) = \ f(x) : pmap(f, xs)$

$pmap(lambda(i) = i * i, [2,3,5])$ publishes [4, 9, 25]

========================

$def\ \ repeat(f) = \ f() \gg repeat(f)$

$def\ \ pr() = \ Println(3)$

$repeat(pr)$ prints 3 forever.

Comingling functional and Orc expressions

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

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

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

Translating Programs to Orc Calculus

- All programs are translated to Orc calculus.
- $1 + 2$ becomes $add(1, 2)$
  All arithmetic and logical operators, tuples, lists are site calls.
  if-then-else is translated with calls to Ift, Iff sites.
- $1 + (2 + 3)$ should become $add(1, add(2, 3))$
  But this is not legal Orc! Site calls can not be nested.
- What is the meaning of $(1 \mid 2) + (2 \mid 3)$?

Deflation

Given $C[F]$ where a single value is expected from expression F, convert $C[F]$ to

$$C[x] \;\; {<}x{<} \; F$$

$$1 + 2 \mid 2 + 3 \quad \text{is} \quad add(1, 2) \mid add(2, 3)$$

$$1 + (2 + 3) \quad \text{is} \quad add(1, x) \;\; {<}x{<} \; add(2, 3)$$

$$(1 \mid 2) + (2 \mid 3) \quad \text{is} \quad (add(x, y) \;\; {<}x{<} \; (1 \mid 2)) \;\; {<}y{<} \; (2 \mid 3)$$

Invariably: F is a parameter in a site call.

Consequence of Deflation

- Translation of val:

  $$\mathit{val}\ \ z = \ g$$
  $$f$$

  becomes

  $$f\ \ <z<\ g$$

- All arguments of function(site) calls are evaluated concurrently.

  $M(f,g)$ becomes
  $(M(x,y))\ \ <x<\ f)\ \ <y<\ g$

Implicit Concurrency Example

- An experiment tosses two dice.
  Experiment is a success if and only if sum of the two dice thrown is 7.
- $exp(n)$ runs n experiments and reports the number of successes.

$$\mathit{def}\ \ toss() = Random(6) + 1$$ -- return random n, $1 \leq n \leq 6$

$$\begin{aligned}
&\mathit{def}\ \ exp(0) = 0 \\
&\mathit{def}\ \ exp(n) = exp(n-1) \\
&\qquad\qquad + (\mathit{if}\ \ toss() + toss() = 7\ \mathit{then}\ \ 1\ \mathit{else}\ \ 0)
\end{aligned}$$

Translation of the dice throw program

$$
\begin{array}{l}
\textit{def}\ \ toss() = add(x,1)\ \ {<}x{<}\ Random(6)\\
\textit{def}\ \ exp(n) =\\
\quad (\ Ift(b)\ \gg 0\\
\quad \mid Iff(b)\ \gg\\
\qquad (\ add(x,y)\\
\qquad\quad {<}x{<}\ (\ exp(m)\ \ {<}m{<}\ sub(n,1)\ )\\
\qquad\quad {<}y{<}\ (\ Ift(bb)\ \gg 1 \mid Iff(bb)\ \gg 0\ )\\
\qquad\qquad {<}bb{<}\ equals(p,7)\\
\qquad\qquad\quad {<}p{<}\ add(q,r)\\
\qquad\qquad\qquad {<}q{<}\ toss()\\
\qquad\qquad\qquad {<}r{<}\ toss()\\
\qquad)\\
\\
\quad)\ \ {<}b{<}\ equals(n,0)
\end{array}
$$

Note: $2n$ parallel calls to $toss()$.

Choice: Execute either f or g

$if\ (true\ |\ false)\ then\ f\ else\ g$

Timeout

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

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

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

Variation:
Execute $f(z)$ in case there is no timeout,
g in case of timeout.

$$val\ \ (z, b) = \ (M(), true) \mid (Rwait(t), false)$$
$$if\ b\ then\ \ f(z)\ else\ \ g$$

Fork-join parallelism

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

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

or,

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

Simple definitions using $Random()$

- Return a random boolean.

  $$def\ \ rbool() = (Random(2) = 0)$$

- Return a random real number between 0 and 1.

  $$def\ \ frandom() = Random(1001)/1000.0$$

- Return *true* with probability p, *false* with $(1 - p)$

  $$def\ \ biasedBool(p) = (Random(1000) <: p * 1000)$$

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.

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

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.

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

Barrier Synchronization in $M() \gg f \mid N() \gg g$

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

$$(M(), N()) \gg (f \mid g)$$

Priority

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

  $$val\ \ (u, \_) = (N(), Rwait(1))$$

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

  $$val\ \ x = M() \mid u$$

Interrupt f

- Evaluation of f can not be directly interrupted.
- Introduce two sites:
  - *Interrupt.set*: to interrupt f
  - *Interrupt.get*: responds only after *Interrupt.set* has been called.

  - *Interrupt.set* is similar to *release* on a semaphore;
    Interrupt.get is similar to *acquire* on a semaphore.
- Instead of f, evaluate

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

Parallel or

Expressions f and g return single booleans. Compute the parallel or.

$$val\ \ x = f$$
$$val\ \ y = g$$

$$Ift(x) \gg true \ | \ Ift(y) \gg true \ | \ (x \,||\, y)$$

Parallel or; contd.

Compute the parallel or and return just one value:

$$
\begin{array}{l}
val\ \ x = f \\
val\ \ y = g \\
val\ \ z = Ift(x) \gg true \mid Ift(y) \gg true \mid (x \,||\, y) \\
\quad z
\end{array}
$$

But this continues execution of g if f first returns true.

$$
\begin{array}{l}
val\ \ z = \\
\quad val\ \ x = f \\
\quad val\ \ y = g \\
\\
\quad Ift(x) \gg true \mid Ift(y) \gg true \mid (x \,||\, y) \\
z
\end{array}
$$

Airline quotes: Application of Parallel or

- Contact airlines A and B.
- Return any quote if it is below $300 as soon as it is available, otherwise return the minimum quote.
- $threshold(x)$ returns x if $x < 300$; silent otherwise.
  $Min(x,y)$ returns the minimum of x and y.

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

Sites

- Sites are first-class values.
  A site may be a parameter in site call.
  A site may return a site as a value.

  $M() \;>(x,y)>\; x(y)$ -- x,y are sites

- Sites may have methods.

  $Channel() \;>ch>\; ch.put(3)$

- Translation of method call $ch.put(3)$:

  $ch(\text{“}put''\text{)} \;>x>\; x(3)$

Some Useful Library Sites

`Ref(n)`	Mutable reference with initial value *n*
`Cell()`	Write-once reference
`Array(n)`	Array of size *n* of Refs
`Semaphore(n)`	Semaphore with initial value *n*
`Channel()`	Unbounded (asynchronous) channel

$$Ref(3) \;\; >r> \; r.write(5) \;\; \gg \;\; r.read()$$

$$Cell() \;\; >r> \; (r.write(5) \mid r.read())$$

$$Array(3) \;\; >a> \; a(0).write(true) \;\; \gg \;\; a(1).read()$$

$$Semaphore(1) \;\; >s> \; s.acquire() \;\; \gg \;\; Println(0) \;\; \gg \;\; s.release()$$

$$Channel() \;\; >ch> \; (ch.get() \mid ch.put(3) \;\; \gg \;\; stop \;)$$

Simple Swap

Convention:

$a?$	is	$a.read()$
$b := x$	is	$b.write(x)$

Take two references as arguments,
Exchange their values, and return a signal.

$$def \;\; swap(a, b) = \; (a?, b?) \;\; >(x, y)> \; (a := y, \; b := x) \; \gg \; signal$$

Note: a and b could be identical Refs.

Update linked list

Given is a one-way linked list.
Its first item is called first.
Now add value v as the first item.

$$Ref() \;>r>$$
$$r := (v, first) \gg$$
$$first := r$$

or,

$$Ref((v, first)) \;>r>$$
$$first := r$$

Memoization

For function f (with no arguments) cache its value after the first call.

res:	stores the cached value.
s:	semaphore value is 0 if the function value has been cached.

$$
\begin{array}{l}
val\ \ res = \ Ref() \\
val\ \ s = \ Semaphore(1) \\
def\ \ memo() = \\
\quad val\ \ z = \ res? \mid \ s.acquire() \gg res := f() \gg stop \\
\quad z
\end{array}
$$

Note: Concurrent calls handled correctly.

Array Permutation

- Randomly permute the elements of an array in place.
- $randomize(i)$ permutes the first i elements of arry a and publishes a signal.

$$
\begin{array}{l}
def\ \ permute(a) = \\
\quad def\ \ randomize(0) = \ signal \\
\quad def\ \ randomize(i) = \ Random(i)\ \ >j> \\
\qquad\qquad\qquad\qquad swap(a(i-1), a(j))\ \gg \\
\qquad\qquad\qquad\qquad randomize(i-1) \\
\\
\quad randomize(a.length?())
\end{array}
$$

Example: Return Array of 0-valued Semaphores

$$
\begin{array}{l}
\textit{def}\ \ \textit{semArray}(n) = \\
\quad \textit{val}\ \ a = \textit{Array}(n) \\
\quad \textit{def}\ \ \textit{populate}(0) = \textit{signal} \\
\quad \textit{def}\ \ \textit{populate}(i) = a(i-1) := \textit{Semaphore}(0) \gg \textit{populate}(i-1) \\
\\
\quad \textit{populate}(n) \gg a
\end{array}
$$

Usage: $\textit{semArray}(5)\ \ {>}a{>}\ a(1)?.\textit{release}()$

Library function: *Table*

- $Table(n, f)$, where $n > 0$ and f a function closure.
  Creates function g, where $g(i) = f(i)$, $0 \leq i < n$.
  An array of function values pre-computed and reused.
- All values of g are computed at instantiation.
- Allows creating arrays of structures.
- Function f may be supplied as: $lambda(i) = h(i)$

Examples:

- $val\ \ g = \ Table(5, lambda(\_) = Channel()\ )$
- $val\ \ h = \ Table(5, lambda(i) = 2 * i)$
- $val\ \ s = \ Table(5, lambda(\_) = Semaphore(0)\ )$

Memoize Fibonacci Computation

Cache $mfib(i)$ using $s(i)$ and $res(i)$.

$val\ N = 200$ -- Largest call argument
$val\ s = Table(N, lambda(\_) = Semaphore(1))$
$val\ res = Table(N, lambda(\_) = Ref())$

$def\ mfib(0) = 0$
$def\ mfib(1) = 1$
$def\ mfib(i) =$
$\quad val\ z = res(i)?$
$\qquad | \ s(i).acquire() \gg res(i) := mfib(i-1) + mfib(i-2) \gg stop$
$\quad z$