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 |"

\[
\begin{align*}
1 + 2 & \quad \text{evaluates to} \quad 3 \\
0.4 &= 2.0/5 & \text{evaluates to} \quad \text{true} \\
3 - 5 &> 5 - 3 & \text{evaluates to} \quad \text{false} \\
true &\& (false \mid| true) & \text{evaluates to} \quad \text{true} \\
3/0 & & \text{is silent} \\
"Try" + "Orc" & \quad \text{evaluates to} \quad "TryOrc"
\end{align*}
\]
Variable Binding

\[
\begin{align*}
\text{val } x &= 1 + 2 \\
\text{val } y &= x + x \\
\text{val } z &= x/0 \quad \text{this expression is silent; other evaluations continue} \\
\text{val } u &= \text{if } (0 <: 5) \text{ then } 0 \text{ else } z
\end{align*}
\]
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
(1 + 2, 7) is (3, 7)

("true" + "false", true || false, 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.

\[
def\text{ pairsum}(a, b) = \\
a > (x, y) > b > (x', y') > (x + x', y + y')
\]

or, even better,

\[
def\text{ pairsum}((x, y), (x', y')) = (x + x', y + y')
\]
Clausal Definition, Pattern Matching
Example: Defining graph connectivity

An Undirected Graph

```
def conn(0) = [1, 2, 3, 4]
def conn(1) = [0, 5]
def conn(2) = [0, 4]
def conn(3) = [0, 5]
def conn(4) = [0, 2]
def conn(5) = [1, 3]
```

```
def conn(i) =
    if i > 0 then [1, 2, 3, 4]
    if i > 1 then [0, 5]
    if i > 2 then [0, 4]
    if i > 3 then [0, 5]
    if i > 4 then [0, 2]
    if i > 5 then [1, 3]
```
Clausal definition of a function
Example: Fibonacci numbers

```
def H(0) = (1, 1)
def H(n) = H(n - 1) > (x, y) > (y, x + y)

def Fib(n) = H(n) > (x, _) > x

{- Goal expression -}
Fib(5)
```
Closure: Functions as values

\begin{verbatim}
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.
\end{verbatim}
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 $\text{add}(1, 2)$
  All arithmetic and logical operators, tuples, lists are site calls.
  if-then-else is translated with calls to $\text{Ift}$, $\text{Iff}$ sites.

- $1 + (2 + 3)$ should become $\text{add}(1, \text{add}(2, 3))$
  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

\[
C[x] \ < x < \ F
\]

\[
1 + 2 | 2 + 3 \quad \text{is} \quad add(1, 2) | add(2, 3)
\]

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

\[
(1 | 2) + (2 | 3) \quad \text{is} \quad (add(x, y) < x < (1 | 2)) < y < (2 | 3)
\]

Invariably: \( F \) is a parameter in a site call.
Consequence of Deflation

- Translation of val:

  \[ val \ z = g \]

  \[ f \]

  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.

\[
\begin{align*}
def \text{toss}() &= \text{Random}(6) + 1 \quad \text{-- return random} \quad n, \quad 1 \leq n \leq 6 \\
def \text{exp}(0) &= 0 \\
def \text{exp}(n) &= \text{exp}(n - 1) \\
&\quad + (\text{if} \ \text{toss}() + \text{toss}() = 7 \ \text{then} \ 1 \ \text{else} \ 0)
\end{align*}
\]
Translation of the dice throw program

```python
def toss() = add(x, 1) <x< Random(6)
def exp(n) =
    ( Ift(b) ≫ 0
       | Iff(b) ≫
         ( add(x, y)
            <x< ( exp(m) <m< sub(n, 1) )
            <y< ( Ift(bb) ≫ 1 | Iff(bb) ≫ 0 )
            <bb< equals(p, 7)
            <p< add(q, r)
            <q< toss()
            <r< toss()
         )
     )
  ) <b< equals(n, 0)
```

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.

\[ z \mathrel{<z<} (M() \mid (R\text{wait}(t) \gg 0)), \text{ or} \]

\[ \text{val } z = M() \mid (R\text{wait}(t) \gg 0) \]

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

\[ \text{val } (z, b) = (M(), \text{true}) \mid (R\text{wait}(t), \text{false}) \]
\[ \text{if } b \text{ then } f(z) \text{ else } g \]
Fork-join parallelism

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

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

or,

$(M(), N())$
Simple definitions using \textit{Random}()

- Return a random boolean.
  \[
  \text{def } \text{rbool}(\) = (Random(2) = 0)
  \]

- Return a random real number between 0 and 1.
  \[
  \text{def } \text{frandom}(\) = Random(1001)/1000.0
  \]

- Return \textit{true} with probability \(p\), \textit{false} with \((1 - p)\)
  \[
  \text{def } \text{biasedBool}(p) = (Random(1000) <: p * 1000)
  \]
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.

\[
\begin{align*}
\text{def } & \text{ auction}([\ ])) = 0 \\
\text{def } & \text{ auction}(b : bs) = \max(b.ask(), \text{auction}(bs))
\end{align*}
\]

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.

\[
def \text{auction}([],) = 0
\]

\[
def \text{auction}(b : bs) =
\]
\[
max(
    \quad b.\text{ask}() | (R\text{wait}(8000) \gg 0),
    \quad \text{auction}(bs)
)
\]
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 $N$.
  \[
  \text{val } (u, \_ ) = (N(), Rwait(1))
  \]

- Call $M$, $N$ together.
  If $M$ responds within one unit, publish its response. Else, publish the first response.
  \[
  \text{val } x = M() \mid u
  \]
• Evaluation of $f$ cannot be directly interrupted.

• Introduce two sites:
  • $\text{Interrupt.set}$: to interrupt $f$
  • $\text{Interrupt.get}$: responds only after $\text{Interrupt.set}$ has been called.

    • $\text{Interrupt.set}$ is similar to $\text{release}$ on a semaphore;
      $\text{Interrupt.get}$ is similar to $\text{acquire}$ on a semaphore.

• Instead of $f$, evaluate

\[
z < z < (f | \text{Interrupt.get}())
\]
Parallel or

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

\[
\text{val } x = f \\
\text{val } y = g \\
\text{Ift}(x) \gg true \mid \text{Ift}(y) \gg true \mid (x \mid\mid y)
\]
Parallel or; contd.

Compute the parallel or and return just one value:

\[
\begin{align*}
val \ x &= f \\
val \ y &= g \\
val \ z &= \text{Ift}(x) \gg true \mid \text{Ift}(y) \gg true \mid (x \mid\mid y) \\
\end{align*}
\]

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

\[
\begin{align*}
val \ z &= \\
val \ x &= f \\
val \ y &= g \\
\text{Ift}(x) \gg true \mid \text{Ift}(y) \gg true \mid (x \mid\mid y) \\
\end{align*}
\]
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.

- $\text{threshold}(x)$ returns $x$ if $x < 300$; silent otherwise. $\text{Min}(x, y)$ returns the minimum of $x$ and $y$.

\[
\text{val } z = \\
\text{val } x = A() \\
\text{val } y = B()
\]

\[\text{threshold}(x) \text{ | threshold}(y) \text{ | Min}(x, y) \]

$z$
Sites

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

  \[ M() > (x, y) > x(y) \quad -- \quad x, y \text{ are sites} \]

• Sites may have methods.

  \[ Channel() > ch > ch.put(3) \]

• Translation of method call \( ch.put(3) \):

  \[ ch("put") > x > x(3) \]
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

```plaintext
Ref(3) >r> r.write(5) \implies r.read()

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

Array(3) >a> a(0).write(true) \implies a(1).read()

Semaphore(1) >s> s.acquire() \implies Println(0) \implies s.release()

Channel() >ch> (ch.get() \mid ch.put(3) \implies stop )
```
Simple Swap

Convention:

\[
\begin{align*}
  a? & \quad \text{is} \quad a\text{.read}(()) \\
  b := x & \quad \text{is} \quad b\text{.write}(x)
\end{align*}
\]

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

\[
\text{def } \text{swap}(a, b) = (a?, b?) > (x, y) > (a := y, b := x) \Rightarrow \text{signal}
\]

Note: \( a \) and \( b \) could be identical Refs.
Given is a one-way linked list. Its first item is called \texttt{first}. Now add value \( v \) as the first item.

\[
\begin{align*}
\text{Ref}() & > r > \\
r & := (v, \texttt{first}) \implies \\
\texttt{first} & := r
\end{align*}
\]

or,

\[
\begin{align*}
\text{Ref}((v, \texttt{first})) & > r > \\
\texttt{first} & := r
\end{align*}
\]
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.

```plaintext
val res = Ref()
val s = Semaphore(1)
def memo() =
    val z = res? | s.acquire() => res := f() => stop
    z
```

Note: Concurrent calls handled correctly.
Array Permutation

- Randomly permute the elements of an array in place.
- \textit{randomize}(i) permutes the first \(i\) elements of array \(a\) and publishes a signal.

```python
def permute(a):
    def randomize(0) = signal
    def randomize(i) = Random(i >j>
        swap(a(i - 1), a(j)) ≫
        randomize(i - 1)
    randomize(a.length?())
```
Example: Return Array of 0-valued Semaphores

\[
\text{def } \text{semArray}(n) = \\
\text{val } a = \text{Array}(n) \\
\text{def } \text{populate}(0) = \text{signal} \\
\text{def } \text{populate}(i) = a(i - 1) := \text{Semaphore}(0) \gg \text{populate}(i - 1)
\]

\[
\text{populate}(n) \gg a
\]

Usage: \text{semArray}(5) \gg a(1)?.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: \( \text{lambda}(i) = h(i) \)

Examples:

- \( \text{val } g = \text{Table}(5, \text{lambda}(_)) = \text{Channel}() \ )
- \( \text{val } h = \text{Table}(5, \text{lambda}(i) = 2 * i) \)
- \( \text{val } s = \text{Table}(5, \text{lambda}(_)) = \text{Semaphore}(0) \)
Memoize Fibonacci Computation

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

```plaintext
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) =
val z = res(i)?
  | s(i).acquire() ⇒ res(i) := mfib(i - 1) + mfib(i - 2) ⇒ stop
  z
```