Using Concurrency for Structuring

Jayadev Misra

Department of Computer Science
University of Texas at Austin

http://orc.csres.utexas.edu

Why concurrency?

- To speed up things
- To model an inherently concurrent system
- To structure a system (e.g. operating systems)

Quick Intro to Orc; Parallel Composition

1

:: 1 — publishes 1

1 | 2

:: 1 — publishes both 1

:: 2 — and 2

Quick Intro to Orc; Sequential Composition

$1 \ {>}x{>}\ x + 3$

$:: \ 4$

$(1 \ | \ 2) \ {>}x{>}\ x$

$:: \ 1$

$:: \ 2$

$(1 \ | \ 2) \gg 3$

$:: \ 3$

$:: \ 3$

Quick Intro to Orc; Pruning

$x + 1 \;\; <x< \; 1$

$:: \;\; 1$

$x \;\; <x< \; (1 \;\; | \;\; 2)$

$:: \;\; 2$

$\text{val} \;\; x = (1 \;\; | \;\; 2)$

Example: Fibonacci numbers

def $H(0) = (1, 1)$

def $H(n) =$
  val $(x, y) = H(n-1)$
 $(y, x + y)$

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

– Goal expression
$Fib(5)$

Quick Intro to Orc; Otherwise Combinator

$1 \; ; \; 2$

$:: \; 1$

$\mathit{stop} \; ; \; 2$

$:: \; 2$

$1 \gg \mathit{stop} \; ; \; 2$

$:: \; 2$

Site

- An Orc program calls sites to carry out some of its work.
- Fundamental Site $\mathit{if}(b)$, where b is boolean:
  publish signal if b is true, silent otherwise.
- $\mathit{if}(\mathit{false}) = \mathit{stop}$

Subset Sum

Given is a list of positive integers xs and an integer n.

Enumerate all sublists of xs that add up to n.

Enumerate All Solutions to Subset Sum

$$\text{def } sums(0, \_) = [\,] \qquad \text{— } n = 0$$

$$\text{def } sums(\_, [\,]) = stop \qquad \text{— } n \neq 0 \text{ and } xs = [\,]$$

$$\begin{array}{l} \text{def } sums(n, x : xs) = \qquad \text{— } n \neq 0 \text{ and } xs \neq [\,] \\ \quad if(n > 0) \gg \\ \quad\quad (sums(n - x, xs) \;\; >ys> x : ys \;\mid\; sums(n, xs)) \end{array}$$

Completing the Program

$\text{def } \ \mathit{enum}(n, \mathit{xs}) = \mathit{sums}(n, \mathit{xs}) \ \ {>}\mathit{ys}{>} \ \mathit{Some}(\mathit{ys}) \ ; \mathit{None}()$

$\mathit{enum}(10, [2, 4, 1, 2, 3])$

$:: \ \ \mathit{Some}([2, 4, 1, 3])$
$:: \ \ \mathit{Some}([4, 1, 2, 3])$

Enumerate at most one solution

$\text{def } \; sums(0, \_) = [\,]$ — $n = 0$

$\text{def } \; sums(\_, [\,]) = stop$ — $n \neq 0$ and $xs = [\,]$

$\text{def } \; sums(n, x : xs) =$ — $n \neq 0$ and $xs \neq [\,]$

$$if(n > 0) \gg$$
$$(sums(n - x, xs) \; >ys> x : ys \; | \; sums(n, xs))$$

$\text{def } \; one(n, xs) = (Some(ys) \; <ys< sums(n, xs)) \; ; None()$

$one(10, [2, 4, 1, 2, 3])$

$:: \; Some([2, 4, 1, 3])$

The first lexicographic solution

$$\text{def } sum(0, \_) = [\,] \qquad — \; n = 0$$

$$\text{def } sum(\_, [\,]) = stop \qquad — \; n \neq 0 \text{ and } xs = [\,]$$

$$\text{def } sum(n, x : xs) = \qquad — \; n \neq 0 \text{ and } xs \neq [\,]$$
$$\quad if(n > 0) \gg$$
$$\qquad (x : sum(n - x, xs) \; ; \; sum(n, xs))$$

$$\text{def } first(n, xs) = Some(sum(n, xs)) \; ; \; None()$$

$$first(15, [2, 4, 1, 2, 3])$$

$$:: \; None()$$

Parsing using Recursive Descent

Consider the grammar:

$$\begin{array}{lcl} \mathit{expr} & ::= & \mathit{term} \mid \mathit{term} + \mathit{expr} \\ \mathit{term} & ::= & \mathit{factor} \mid \mathit{factor} * \mathit{term} \\ \mathit{factor} & ::= & \mathit{literal} \mid (\mathit{expr}) \\ \mathit{literal} & ::= & 3 \mid 5 \end{array}$$

Parsing strategy

For each non-terminal, say $expr$, define $expr(xs)$:
publish all suffixes of xs such that the prefix is a $term$.

$$\text{def } isexpr(xs) = expr(xs) \;\; >[]> true \;;\; false$$

To avoid multiple publications (in ambiguous grammars),

$$\begin{array}{l} \text{def } isexpr(xs) = \\ \quad \text{val } res = expr(xs) \;\; >[]> true \;;\; false \\ \quad\; res \end{array}$$

$$\begin{array}{l} isexpr \\ (["(","(","3","*","3",")",")","+","(","3","+","3",")"]) \end{array}$$

— ((3*3))+(3+3)

:: $true$

Function for each non-terminal

Given: $expr ::= term \mid term + expr$

Rewrite: $expr ::= term\ (\epsilon \mid + expr)$

$$\text{def } expr(xs) = term(xs) > ys > (ys \mid ys > \text{“}+\text{''} : zs > expr(zs))$$

$$\text{def } term(xs) = factor(xs) > ys > (ys \mid ys > \text{“}*\text{''} : zs > term(zs))$$

$$\text{def } factor(xs) = literal(xs) \mid xs > \text{“}(\text{''} : ys > expr(ys) > \text{“})\text{''} : zs > zs$$

$$\text{def } literal(n : xs) = n > \text{“}3\text{''} > xs \mid n > \text{“}5\text{''} > xs$$

$$\text{def } literal([\,]) = stop$$

Exception Handling; callback

- A client requests a service from a server.
- Typically, the server fulfills the request.
- Sometimes, server requests authentication.

Exception Handling Program

$\text{def } \ \mathit{request}() =$
$\quad \text{val } \ \mathit{exc} = \mathit{Buffer}()$ — returns a buffer site

$$\mathit{server.req}(\mathit{exc}) \ \ >v> \mathit{Some}(v)$$

$$\mid \ \mathit{exc.get}() \ \ >r> \mathit{exc.put}(\mathit{auth}(r)) \ \gg \mathit{stop}$$