A Language for Task Orchestration and its Semantic Properties

David Kitchin, William Cook and Jayadev Misra

Department of Computer Science
University of Texas at Austin

Email: {dkitchin,wcook,misra}@cs.utexas.edu
web: http://www.cs.utexas.edu/users/wcook/projects/orc/

Overview of Orc

- Orchestration language.
  - Invoke services by calling sites
  - Manage time-outs, priorities, and failures
- Structured concurrent programming. A Program execution
  - calls sites,
  - publishes values.
- Simple calculus, with only 3 combinators.
  - Semantics described by labeled transition system and traces
  - Easy to create and terminate processes
- Prototype implementation available.

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	$f > x > g$	Piping
for some x from g do f	f where $x :\in g$	Asymmetric composition

Symmetric composition: $f \mid g$

$CNN \mid BBC$: calls both CNN and BBC simultaneously.

Publishes values returned by both sites. (0, 1 or 2 values)

- Evaluate f and g independently.
- Publish all values from both.
- No direct communication or interaction between f and g. They may communicate only through sites.

Pipe: $f \;>x>\; g$

For all values published by f do g. Publish only the values from g.

- $CNN \;>x>\; Email(address, x)$

  Call CNN. Bind result (if any) to x. Call $Email(address, x)$.

  Publish the value, if any, returned by $Email$.

- $(CNN \mid BBC) \;>x>\; Email(address, x)$

  May call $Email$ twice. Publishes up to two values from $Email$.

Notation:
Write $f \gg g$ for $f \;>x>\; g$ if x unused in g.
Precedence: $f \;>x>\; g \mid h \;>y>\; u$
$(f \;>x>\; g) \mid (h \;>y>\; u)$

Schematic of piping

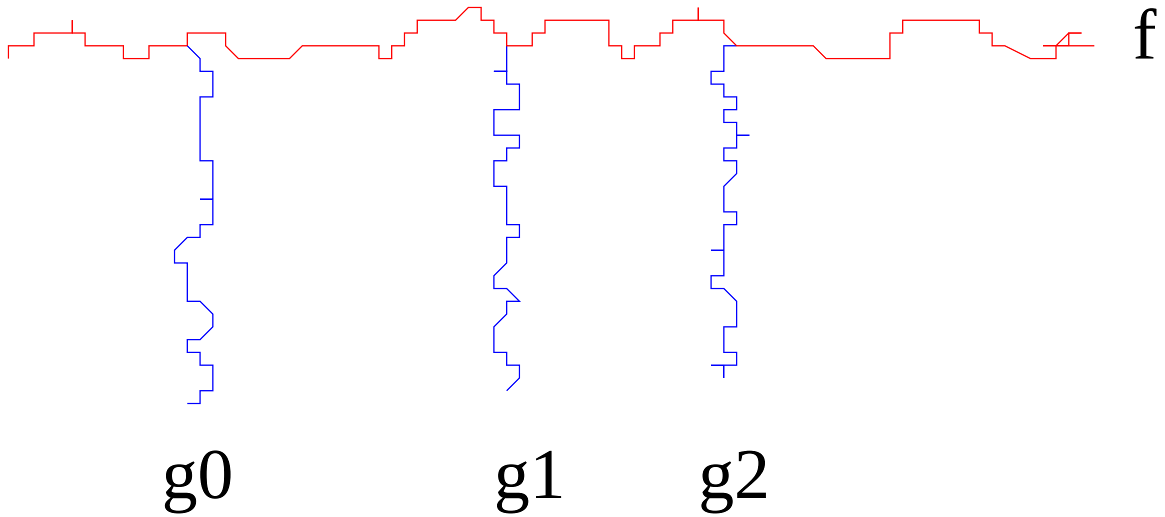


Figure 1: Schematic of $f > x > g$

Asymmetric parallel composition: $(f \text{ where } x{:}\in g)$

For some value published by g do f. Publish only the values from f.

$$Email(address, x) \text{ where } x{:}\in (CNN \mid BBC)$$

Binds x to the first value from $CNN \mid BBC$.

- Evaluate f and g in parallel.
  Site calls that need x are suspended; other site calls proceed.
  $(M \mid N(x)) \text{ where } x{:}\in g$
- When g returns a value, assign it to x and terminate g.
  Resume suspended calls.
- Values published by f are the values of $(f \text{ where } x{:}\in g)$.

Some Fundamental Sites

0: never responds.

$let(x, y, \cdots)$: returns a tuple of its argument values.

$if(b)$: boolean b,
returns a signal if b is true; remains silent if b is false.

$Signal$ returns a signal immediately. Same as $if(\textit{true})$.

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

Expression Definition

$MailOnce(a)\ \underline{\Delta}$
$\quad Email(a,m)\ \text{where } m :\in (CNN \mid BBC)$

$MailLoop(a,t)\ \underline{\Delta}$
$\quad MailOnce(a) \gg Rtimer(t) \gg MailLoop(a,t)$

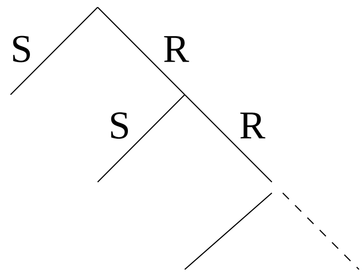
- Expression is called like a procedure.
  May publish many values. $MailLoop$ does not publish a value.

- Site calls are strict; expression calls non-strict.

Metronome

Publish a signal at every time unit.

$$Metronome \ \underline{\Delta}\ Signal \mid Rtimer(1) \gg Metronome$$



Publish n signals.

$$\begin{array}{lll} BM(0) & \underline{\Delta} & \mathbf{0} \\ BM(n) & \underline{\Delta} & Signal \mid Rtimer(1) \gg BM(n-1) \end{array}$$

Recursive definition with time-out

Call a list of sites.

Count the number of responses received within 10 time units.

$$
\begin{array}{lll}
tally([\,]) & \underline{\Delta} & let(0) \\
tally(M : MS) & \underline{\Delta} & \\
 & & u + v \\
 & & \quad \text{where} \\
 & & \qquad u{:}\in\; M \;\gg\; let(1) \mid Rtimer(10) \;\gg\; let(0) \\
 & & \qquad v{:}\in\; tally(MS)
\end{array}
$$

Time-out

Return $(x,$ ***true***$)$ if M returns x before t, and $(-,$ ***false***$)$ otherwise.

$$
\begin{array}{l}
let(z,b) \\
\quad \text{where} \\
\qquad (z,b) :\in \\
\qquad\qquad M \;\; > x > \; let(x, \textit{\textbf{true}}) \\
\qquad\quad\; \mid \; Rtimer(t) \;\; > x > \; let(x, \textit{\textbf{false}})
\end{array}
$$

Fork-join parallelism

Call M and N in parallel.

Return their values as a tuple after both respond.

$let(u, v)$
where $u :\in M$
$v :\in N$

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

f and g start only after both M and N complete.

$$
\begin{array}{ll}
(& let(u, v) \\
 & \text{where } u {:\in} M \\
 & \qquad\quad\; v {:\in} N\,) \\
\gg & (f \mid g)
\end{array}
$$

Arbitration

In CCS: $\alpha.P + \beta.Q$

In Orc:

$$if(b) \gg P \mid if(\neg b) \gg Q$$

where

$$b{:}\in\ Alpha \gg let(\textsf{true}) \mid Beta \gg let(\textsf{false})$$

Orc does not permit non-deterministic internal choice.

Priority

- Publish N's response asap, but no earlier than 1 unit from now.

$$Delay \;\underline{\Delta}\; (Rtimer(1) \gg let(u)) \text{ where } u :\in N$$

- Call M, N together.

  If M responds within one unit, take its response.

  Else, pick the first response.

$$let(x) \text{ where } x :\in (M \mid Delay)$$

Parallel or

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

$ift(b) = if(b) \gg let(\textbf{true})$: returns **true** if b is **true**; silent otherwise.

$$ift(x) \mid ift(y) \mid or(x,y)$$
where
$x :\in M,\ y :\in N$

To return just one value:

$$let(z)$$
where
$z :\in ift(x) \mid ift(y) \mid or(x,y)$
$x :\in M$
$y :\in N$

Formal Syntax

f, g, h	$\in$	$Expr$	::=	$M(\overline{p})$	Site call
				$\parallel E(\overline{p})$	Expression call
				$\parallel f \ >x> \ g$	Sequential composition
				$\parallel f \mid g$	Symmetric composition
				$\parallel f \ \text{where}\ x{:}{\in}\ g$	Asymmetric composition
p	$\in$	$Actual$	::=	$x \parallel v \parallel M$	
		$Defn.$	::=	$E(x) \ \underline{\Delta} \ f$	

Transitions, Events

$f \xrightarrow{a} f'$: f may engage in event a and transit to f'.

Base events

$BaseEvent$	::=	$!v$	publish
		$\parallel \tau$	internal event
		$\parallel M_k(v)$	Site call with handle k
		$\parallel k?v$	Response

Response is outside the control of Orc.

Rules for Site Call

$$\frac{k \text{ fresh}}{M(v) \overset{M_k(v)}{\longrightarrow} ?k}$$

$$?k \overset{k?v}{\longrightarrow} let(v)$$

$$let(v) \overset{!v}{\longrightarrow} 0$$

Symmetric Composition

$$\frac{f \xrightarrow{a} f'}{f \mid g \xrightarrow{a} f' \mid g}$$

$$\frac{g \xrightarrow{a} g'}{f \mid g \xrightarrow{a} f \mid g'}$$

Piping

$$\frac{f \xrightarrow{a} f' \qquad a \neq !v}{f \;>x>\; g \xrightarrow{a} f' \;>x>\; g}$$

$$\frac{f \xrightarrow{!v} f'}{f \;>x>\; g \xrightarrow{\tau} (f' \;>x>\; g) \mid [v/x].g}$$

Asymmetric Composition

$$\frac{f \xrightarrow{a} f'}{f \text{ where } x{:}{\in}\ g \xrightarrow{a} f' \text{ where } x{:}{\in}\ g}$$

$$\frac{g \xrightarrow{!v} g'}{f \text{ where } x{:}{\in}\ g \xrightarrow{\tau} [v/x].f}$$

$$\frac{g \xrightarrow{a} g' \qquad a \neq\ !v}{f \text{ where } x{:}{\in}\ g \xrightarrow{a} f \text{ where } x{:}{\in}\ g'}$$

Expression Call

$$\frac{[[\, E(x) \;\; \underline{\Delta} \;\; f \,]] \in D}{E(p) \;\stackrel{\tau}{\rightarrow}\; [p/x].f}$$

Rules

$$\frac{k \text{ fresh}}{M(v) \overset{M_k(v)}{\to} ?k}$$

$$?k \overset{k?v}{\to} let(v)$$

$$let(v) \overset{!v}{\to} 0$$

$$\frac{f \overset{a}{\to} f'}{f \mid g \overset{a}{\to} f' \mid g}$$

$$\frac{g \overset{a}{\to} g'}{f \mid g \overset{a}{\to} f \mid g'}$$

$$\frac{[[\, E(x) \;\underline{\Delta}\; f \,]] \in D}{E(p) \overset{\tau}{\to} [p/x].f}$$

$$\frac{f \overset{a}{\to} f' \qquad a \neq !v}{f \;>x>\; g \overset{a}{\to} f' \;>x>\; g}$$

$$\frac{f \overset{!v}{\to} f'}{f \;>x>\; g \overset{\tau}{\to} (f' \;>x>\; g) \mid [v/x].g}$$

$$\frac{f \overset{a}{\to} f'}{f \text{ where } x{:}\in g \overset{a}{\to} f' \text{ where } x{:}\in g}$$

$$\frac{g \overset{!v}{\to} g'}{f \text{ where } x{:}\in g \overset{\tau}{\to} [v/x].f}$$

$$\frac{g \overset{a}{\to} g' \qquad a \neq !v}{f \text{ where } x{:}\in g \overset{a}{\to} f \text{ where } x{:}\in g'}$$

Example

$$((M(x) \mid let(x)) \ {>}y{>} \ R(y)) \text{ where } x{:}{\in} \ (N \mid S)$$

$$\stackrel{S_k}{\to} \{\text{Call } S: \ S \stackrel{S_k}{\to} ?k; \ N \mid S \stackrel{S_k}{\to} N \mid ?k\}$$

$$((M(x) \mid let(x)) \ {>}y{>} \ R(y)) \text{ where } x{:}{\in} \ (N \mid ?k)$$

$$\stackrel{N_l}{\to} \{\text{Call } N\}$$

$$((M(x) \mid let(x)) \ {>}y{>} \ R(y)) \text{ where } x{:}{\in} \ (?l \mid ?k)$$

$$\stackrel{l?5}{\to} \{ \ ?l \stackrel{l?5}{\to} let(5); \ ?l \mid ?k \stackrel{l?5}{\to} let(5) \mid ?k\}$$

$$((M(x) \mid let(x)) \ {>}y{>} \ R(y)) \text{ where } x{:}{\in} \ (let(5) \mid ?k)$$

Example

$$((M(x) \mid let(x)) \; {>}y{>} \; R(y)) \text{ where } x{:}{\in} \; (let(5) \mid ?k)$$

$$\stackrel{\tau}{\rightarrow} \{ \; let(5) \stackrel{!5}{\rightarrow} 0; \;\; let(5) \mid ?k \stackrel{!5}{\rightarrow} 0 \mid ?k \}$$

$$(M(5) \mid let(5)) \; {>}y{>} \; R(y)$$

$$\stackrel{\tau}{\rightarrow} \{ \; let(5) \stackrel{!5}{\rightarrow} 0; \;\; M(5) \mid let(5) \stackrel{!5}{\rightarrow} M(5) \mid 0;$$

$$f \stackrel{!v}{\rightarrow} f' \text{ implies } f \; {>}y{>} \; g \stackrel{\tau}{\rightarrow} (f' \; {>}y{>} \; g) \mid [v/y].g \}$$

$$((M(5) \mid 0) \; {>}y{>} \; R(y)) \mid R(5)$$

$\stackrel{R_n(5)}{\longrightarrow}$ {call R with argument (5)}

$$((M(5) \mid 0) \; {>}y{>} \; R(y)) \mid ?n$$

Example

$$((M(5) \mid 0) \;{>}y{>}\; R(y)) \mid ?n$$
$$\stackrel{n?7}{\rightarrow} \{\; ?n \;\stackrel{n?7}{\rightarrow}\; let(7) \}$$

$$((M(5) \mid 0) \;{>}y{>}\; R(y)) \mid let(7)$$

$$\stackrel{!7}{\rightarrow} \{\; f \mid let(7) \;\stackrel{!7}{\rightarrow}\; f \mid 0 \}$$

$$((M(5) \mid 0) \;{>}y{>}\; R(y)) \mid 0$$

The sequence of events: $S_k \quad N_l \quad l?5 \quad \tau \quad \tau \quad R_n(5) \quad n?7 \quad !7$

The sequence minus τ events: $S_k \quad N_l \quad l?5 \quad R_n(5) \quad n?7 \quad !7$

Executions and Traces

Define $f \stackrel{\epsilon}{\Rightarrow} f$ $\qquad \dfrac{f \stackrel{a}{\rightarrow} f'', \; f'' \stackrel{s}{\Rightarrow} f'}{f \stackrel{as}{\Rightarrow} f'}$

- Given $f \stackrel{s}{\Rightarrow} f'$, s is an execution of f.
- A trace is an execution minus τ events.
- The set of executions of f (and traces) are prefix-closed.

Laws, using strong bisimulation

- $f \mid 0 \sim f$
- $f \mid g \sim g \mid f$
- $f \mid (g \mid h) \sim (f \mid g) \mid h$
- $f > x > (g > y > h) \sim (f > x > g) > y > h$, if h is x-free.
- $0 > x > f \sim 0$
- $(f \mid g) > x > h \sim f > x > h \mid g > x > h$
- $(f \mid g)$ where $x :\in h \sim (f$ where $x :\in h) \mid g$, if g is x-free.
- $(f > y > g)$ where $x :\in h \sim (f$ where $x :\in h) > y > g$, if g is x-free.
- $(f$ where $x :\in g)$ where $y :\in h \sim (f$ where $y :\in h)$ where $x :\in g$,
  if g is y-free,
  h is x-free.

Relation $\sim$ is an equality

Given $f \sim g$, show

1. $f \mid h \sim g \mid h$
   $h \mid f \sim h \mid g$

2. $f > x > h \sim g > x > h$
   $h > x > f \sim h > x > g$

3. $f \text{ where } x{:}\in h \sim g \text{ where } x{:}\in h$
   $h \text{ where } x{:}\in f \sim h \text{ where } x{:}\in g$

Treatment of Free Variables

Closed expression: No free variable.
Open expression: Has free variable.

- Law $f \sim g$ holds only if both f and g are closed.

  Otherwise: $let(x) \sim 0$

  But $let(1) >x> 0 \neq let(1) >x> let(x)$

- Then we can't show $let(x) \mid let(y) \sim let(y) \mid let(x)$

Substitution Event

$$f \stackrel{[v/x]}{\longrightarrow} [v/x].f \qquad \text{(SUBST)}$$

- Now, $let(x) \stackrel{[1/x]}{\longrightarrow} let(1)$.

  So, $let(x) \neq 0$

- Earlier rules apply to base events only.

  From $f \stackrel{[v/x]}{\longrightarrow} [v/x].f$, we can not conclude:

  $f \mid g \stackrel{[v/x]}{\longrightarrow} [v/x].f \mid g$

Traces as Denotations

Define Orc combinators over trace sets, S and T. Define:

$$S \mid T, \quad S \;{>}x{>}\; T, \quad S \text{ where } x{:}{\in}\; T.$$

Notation: $\langle f \rangle$ is the set of traces of f.

Theorem

$$\begin{array}{lcl} \langle f \mid g \rangle & = & \langle f \rangle \mid \langle g \rangle \\ \langle f \;{>}x{>}\; g \rangle & = & \langle f \rangle \;{>}x{>}\; \langle g \rangle \\ \langle f \text{ where } x{:}{\in}\; g \rangle & = & \langle f \rangle \text{ where } x{:}{\in}\; \langle g \rangle \end{array}$$

Expressions are equal if their trace sets are equal

Define: $f \cong g$ if $\langle f \rangle = \langle g \rangle$.

Theorem (Combinators preserve $\cong$)

Given $f \cong g$ and any combinator $*$: $f * h \cong g * h$, $h * f \cong h * g$

Specifically, given $f \cong g$

1. $f \mid h \cong g \mid h$
   $h \mid f \cong h \mid g$

2. $f > x > h \cong g > x > h$
   $h > x > f \cong h > x > g$

3. $f \text{ where } x :\in h \cong g \text{ where } x :\in h$
   $h \text{ where } x :\in f \cong h \text{ where } x :\in g$

Monotonicity, Continuity

- Define: $f \sqsubseteq g$ if $\langle f \rangle \subseteq \langle g \rangle$.

  Theorem (Monotonicity) Given $f \sqsubseteq g$ and any combinator $*$

  $$f * h \sqsubseteq g * h, \quad h * f \sqsubseteq h * g$$

- Chain f: $f_0 \sqsubseteq f_1, \cdots f_i \sqsubseteq f_{i+1}, \cdots$.

  Theorem: $\sqcup(f_i * h) \cong (\sqcup f) * h$.

  Theorem: $\sqcup(h * f_i) \cong h * (\sqcup f)$.

Least Fixed Point

$$M \;\underline{\Delta}\; S \mid R \gg M$$

$$M_0 \cong 0$$
$$M_{i+1} \cong S \mid R \gg M_i\,, \quad i \geq 0$$

M is the least upper bound of the chain $M_0 \sqsubseteq M_1 \sqsubseteq \cdots$

Weak Bisimulation

$$signal \gg f \quad \cong \quad f$$
$$f \;>x>\; let(x) \quad \cong \quad f$$

Theoretical Justification for Simplicity of Orc

- Simple trace semantics.
- Monotonicity, continuity of the combinators.
- Least fixed point characterizations of recursive definitions.
- Enjoys properties of functional programs, yet highly non-detrministic.

Extensions

- Time
- Synchrony
- Immediate sites