1 Orc

Orc is all about "orchestration". It's basically the communciation, distribution skeleton of a program. Web scripting, workflow applications etc. Orc is the glue for building these orchestration tasks. Orc's key abstraction is a site, something you can call and that publishes results. Services are implemented as sites.

Orc is a simple language. It consists of

- Site calls $M(v)$
- Symmetric parallel composition $f|g$
- Sequential composition with respect to a variable x: $f > x > g$. f executes after g publishes a value, with x bound to that value. If g publishes more than one value, f is run multiple times, once for each value published.
- Asymmetric parallel composition with respect to x: $f < x < g$ Subcomputations of f that don't depend upon x execute in parallel with g, while computations dependent upon x block until g publishes a value which is in turn bound to x. f is run at most once.
- Definitions $D(x) =_{df} g$

Site calls Site calls perform a computation and publish at most one result.

Symmetric parallel composition To evaluate $f|g$, evaluate f and g in parallel. $f|g|$ publishes v iff f or g publishes v.

Sequential composition To evaluate $f > x > g$, begin by evaluating f. For each v published by f, evaluate $[v/x]g$ in parallel. $f > x > g$ publishes w iff some $[v/x]g$ publishes w.

Asymmetric parallel composition To evaluate $f < x < g$, evaluate f and g in parallel. f may block waiting for data from g (x). If g publishes v, kill g and continue evaluating $[v/x]f$.

Question 1. Is $< x <$ necessary? Can it be encoded using $|$ and $> x >$?

Comment 1. From Owen: Seems like it was inspired by Bash, I would really like to use a language like this to write in a command line script.

1.1 Examples

- $fork - join = (let(x, y) < x < M) < y < N$
- $sync = fork - join > x > (f|g)$
- $delay = (Rtimer(1) >> let(x)) < x < M$
- $priority = let(x) < x < (N|delay)$

2 Timed Trace semantics

Originally, Orc was given an asynchronous, then a "synchronous-but-untimed" semantics. Here we will use a "relative-time" semantics which describe delays from site calls.

$$(Rtimer(s) >> let(v))|(Rtimer(3) >> let(w))$$

The operational semantics are based on a labelled transition system $f \overset{t,a,f'}{\to}$ with time-event pairs t, a for labels.

$$f \overset{t,a,f'}{\to}$$

Expression f **m**ay engage in event a after t units of time, without engaging in other events, resulting in expression f'

2.1 Rules

Sites

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

Immediately publish value v and transition to an expression 0 that engages in no other events.

$$Rtimer(t) \overset{t,!}{\to} 0$$

Publish a signal after t time units.

Combinators

$$\frac{f \to^{t,a} f'}{f|g \to^{t,a} f'|g}$$

Works in asynchronous system, but NOT with time. Consider $Rtimer(8)|Rtimer(3)$.

To fix this, we introduce "Time Shifting", f^t. Evaluate for t time units without an event. For example, $Rtimer(5)^3 \equiv Rtimer(2)$. But, it may not alwats be possible: $Rtimer(5)^7 \equiv \perp$.

- $Rtimer(2)^5 \equiv \perp$

$$\frac{f \to^{t,a} f'}{f|g \to^{t,a} f'|g^t}$$

Only if g^t is not $\perp$.

Rest of the semantics similarly extend the asynchronous semantics.

$$\frac{[E(x) \triangleq f] \in \mathcal{D}}{E(p) \stackrel{0,\tau}{\to} [p/x].f}\text{Def} \qquad \frac{k \in \Sigma(M,m)}{M(m) \stackrel{0,\tau}{\to} ?k}\text{Call} \qquad \frac{(t,m) \in k}{?k \stackrel{t,!m}{\to} 0}\text{Return} \qquad \frac{f \stackrel{t,a}{\to} f'}{f|g \stackrel{t,a}{\to} f'|g^t}\text{Sym1}$$

$$\frac{f \stackrel{t,a}{\to} g'}{f|g \stackrel{t,a}{\to} f^t|g'}\text{Sym2} \qquad \frac{f \stackrel{t,a}{\to} f' \qquad a \neq !m}{f > x > g \stackrel{t,a}{\to} f' > x > g}\text{Seq1N}$$

$$\frac{f \stackrel{t,!m}{\to} f'}{f > x > g \stackrel{t,\tau}{\to} (f' > x > g)|[m/x].g}\text{Seq1V} \qquad \frac{f \stackrel{t,a}{\to} f'}{f < x < g \stackrel{t,a}{\to} f' < x < g^t}\text{Asym1}$$

$$\frac{f \stackrel{t,!m}{\to} g'}{f < x < g \stackrel{t,\tau}{\to} [m/x].f^t}\text{Asym2V} \qquad \frac{g \stackrel{t,a}{\to} g' \qquad a \neq !m}{f < x < g \stackrel{t,a}{\to} f^t < x < g'}\text{Asym2N}$$

2.2 Denotational Semantics

An **e**xecution is a finite sequence of time-event pairs that f engages in. A **t**race is an execution without internal events. $< f >=$ traces of f defined operationally.

Trace sets form a **d**enotation, $\mu(f)$.

Theorem 1. Operational and denotation semantics are equivalent: $< f >= \mu(f)$.

This allows for compositional reasoning.