Operational and Denotational Semantics of the Otherwise Combinator

David Kitchin

December 9, 2012

1 The Otherwise Combinator

The otherwise combinator, written f **;** g, is a fourth concurrency combinator that was not present in the original formulation of the Orc calculus. It is used to detect when an expression *halts*; a halted expression can undergo no more transitions. We allow site calls to halt rather than returning a value or waiting forever; this communicates to the orchestration that the site call will never return a value. Then, we can determine when an entire expression halts based on whether its subexpressions have halted and whether its free variables will ever be bound.

The otherwise combinator was introduced to take advantage of additional information that sites can convey about whether they will ever return a value. This is especially relevant when operating over finite data streams, and allows Orc to express certain manipulations of lists, channels, and other structures much more naturally and with greater modularity than could be achieved with the original three combinators.

$$\frac{f \xrightarrow{a} f' \qquad a \neq !v}{f \;\mathbf{;}\; g \xrightarrow{a} f' \;\mathbf{;}\; g} \text{ (StepOW)} \qquad \frac{}{\mathbf{stop} \;\mathbf{;}\; g \xrightarrow{\tau} g} \text{ (Otherwise)}$$

$$\frac{f \xrightarrow{!v} f'}{f \;\mathbf{;}\; g \xrightarrow{!v} f'} \text{ (Discard)} \qquad \frac{}{?k \xrightarrow{k?\bot} \mathbf{stop}} \text{ (SiteHalts)}$$

Figure 1: Simple Conservative Semantics of **;**

2 Operational Semantics

To support the otherwise combinator, we add new rules to the original operational semantics. No rules are removed; the new semantics is a superset of the old semantics. See Figure 1.

These four rules alone implement a conservative approximation of the otherwise combinator. We apply the Otherwise rule only when it is certain that the left expression will undergo no more transitions. However, these rules only allow us to conclude that a site call halts; if the left expression is more complex than just a site call, the Otherwise rule will never apply. For example, the expression (**stop** | **stop**) **;** g has no transitions.

This is too conservative. What we want is a set of rules which will determine when an entire expression is equivalent to **stop**; effectively, we wish to extend the notion of halting to entire expressions, rather than just site calls.

$$\frac{f \equiv \textbf{stop} \qquad g \equiv \textbf{stop}}{f \mid g \equiv \textbf{stop}} \quad (\textsc{Par0})$$

$$\frac{f \equiv \textbf{stop}}{f \;;\; g \xrightarrow{\tau} g} \quad (\textsc{Otherwise}) \qquad\qquad \frac{f \equiv \textbf{stop}}{f \;{>}x{>}\; g \equiv \textbf{stop}} \quad (\textsc{Seq0})$$

$$\frac{f \equiv \textbf{stop} \qquad g \equiv \textbf{stop}}{f \;{<}x{<}\; g \equiv \textbf{stop}} \quad (\textsc{Prune0})$$

Figure 2: Attempted Semantics of **;** using Structural Equivalence

The standard technique in this case is to use a structural equivalence relation. We provide a set of rules which inductively define the equivalence, and revise the Otherwise rule to make use of this structural equivalence. See Figure 2.

This approach is still slightly too conservative. The Prune0 rule waits for both sides to be equivalent to **stop**, but we can do better than this. If the right expression has halted, then we know that the variable x will never be bound; thus, all calls in the left expression which use x are blocked on it forever and should be considered equivalent to **stop**.

But now a problem arises with our use of structural equivalence. Consider the expression $M(x)$ **;** $N(0)$ $<x<$ **stop**. It is clear that $M(x)$ is equivalent to **stop**, so $N(0)$ should proceed, but we cannot derive this fact using the current structural equivalence rules: the premise $f \equiv$ **stop** depends only on f and does not take into account those combinators outside of f which bind variables that it mentions.

$$\frac{f \xrightarrow{a} f' \qquad a \neq !v}{f \; ; \; g \xrightarrow{a} f' \; ; \; g} \; (\textsc{StepOW}) \qquad \frac{}{f \mid \textbf{stop} \xrightarrow{\tau} f} \; (\textsc{Left0})$$

$$\frac{}{\textbf{stop} \mid f \xrightarrow{\tau} f} \; (\textsc{Right0})$$

$$\frac{f \xrightarrow{!v} f'}{f \; ; \; g \xrightarrow{!v} f'} \; (\textsc{Discard}) \qquad \frac{}{\textbf{stop} >x> g \xrightarrow{\tau} \textbf{stop}} \; (\textsc{Seq0})$$

$$\frac{}{\textbf{stop} \; ; \; g \xrightarrow{\tau} g} \; (\textsc{Otherwise}) \qquad \frac{}{f <x< \textbf{stop} \xrightarrow{\tau} [\perp/x]f} \; (\textsc{Prune0})$$

$$\frac{}{?k \xrightarrow{k?\perp} \textbf{stop}} \; (\textsc{SiteHalts}) \qquad \frac{v \neq \lambda... \qquad \perp \in \overline{p}}{v(\overline{p}) \xrightarrow{\tau} \textbf{stop}} \; (\textsc{Call0})$$

Figure 3: Final Operational Semantics of **;**

In order to resolve this problem, we instead give rules that convert an expression to **stop** by steps, rather than inductively determining an expression's equivalence to **stop**. The Otherwise rule returns to its original form; in order for the right expression to proceed, the left expression must be reduced to **stop**. The structural equivalence rules become transition rules instead.

To handle the pruning combinator, we add a new formal parameter $\perp$, which is neither a value nor a variable; it is a placeholder for a variable which will never be bound. When the right expression in a pruning combinator is reduced to **stop**, we then substitute $\perp$ for every occurrence of the variable x in the left expression. If $\perp$ occurs as a parameter to a call, and the callee is not a lambda (i.e. the call is strict), then that call is reduced to **stop**, since a strict call with an unbound parameter will never make progress. Making $\perp$ a formal parameter correctly preserves the semantics of non-strict calls, since the substitution used in those cases (i.e. $[p/x]$) simply substitutes $\perp$ for the appropriate occurrences of the argument variable that will never be bound.

The final set of rules is given in Figure 3.

3 Algebraic Laws

The **;** combinator obeys some basic algebraic laws. Interestingly, its unit and zero are actually the reverse of the $\gg$ combinator. Its left and right unit is **stop** (just like **|**), but its left zero is actually **`signal`**.

(Left zero of **;**)	**`signal`** $;\, f \;=$ **`signal`**
(Left unit of **;**)	**stop** $;\, f \;=\; f$
(Right unit of **;**)	$f\,;$ **stop** $\;=\; f$
(Associativity of **;**)	$(f \;;\; g) \;;\; h \;=\; f \;;\; (g \;;\; h)$
(Distributivity of **;** over $\gg$)	$(f \;;\; g) >x> h \;=\; (f >x> h) \;;\; (g >x> h)$ if f is *not* silent[1]
(Commutativity of **;** with $\ll$)	$(f \;;\; g) <x< h \;=\; (f <x< h) \;;\; g$, if g is x-free
(Conditional conversion)	$(\mathit{if}(x) \gg$ `true`$)\;;$ `false` $=\; \mathit{let}(x)$

[1]Thanks go to Gerard Nicolas for pointing out this necessary side condition.

4 Denotational Semantics

Using sets of executions as denotations, the denotational semantics of the otherwise combinator is straightforward.

Let $S.T$ be the concatenation of two sets of traces, defined by $s \in S \wedge t \in T \Rightarrow st \in S.T$.

Given a set of traces T, let $\mathit{silent}(T)$ be the greatest subset of T whose traces contain no publication events.

Given a set of traces T, let $\mathit{terminal}(T)$ be the greatest subset of T whose traces are not prefixes of any other trace in T. More precisely, $t \in \mathit{terminal}(T) \Rightarrow \neg\exists u :: (u \neq \epsilon \wedge tu \in T)$.

Then the denotation of the otherwise combinator is defined as follows:

$$[\![f \ ; \ g]\!] \quad = \quad ([\![f]\!] - H) \cup H.[\![g]\!] \quad \text{where } H = \mathit{silent}(\mathit{terminal}([\![f]\!]))$$