

A Timed Semantics of Orc

Ian Wehrman, David Kitchin, William R. Cook, and Jayadev Misra

{iwehrman,dkitchin,wcook,misra}@cs.utexas.edu
The University of Texas at Austin*

Abstract. *Orc* is a kernel language for structured concurrent programming. *Orc* provides three powerful combinators that define the structure of a concurrent computation. These combinators support sequential and concurrent execution with blocking and termination. In this paper, we define an operational semantics of *Orc* that allows reasoning about delays introduced explicitly by timers or implicitly by the network. We also present an equivalent denotational semantics of trace sets.

1 Introduction

Orc is a language for structured concurrent programming. Expressions in an *Orc* program are either primitive or a combination of two expressions. A primitive expression is a call to an existing service, a *site*, to perform its computations and return a result to the caller. There are only three combinators for *Orc* expressions, which allow sequential and concurrent executions of expressions, and concurrent execution with termination.

Time is introduced in *Orc* implicitly by delays resulting from remote service calls and explicitly by the site *Rtimer*, which waits a given amount of time when invoked before continuing execution. Previous accounts of the semantics of *Orc* [1–4] have not covered the semantics of time.

An operational semantics of *Orc* is given in Section 2 that includes time. The semantics shown here is based on an asynchronous semantics of *Orc* [1] in which the transition relation is extended to include the time when an event occurs. The semantics allows multiple events to occur at a single instant, and time can be considered either discrete or continuous.

We show that traces form a denotation in Section 3, which allows us reason about the operational behavior of an *Orc* expression compositionally. In particular, the denotational semantics shows that the trace set of a recursively defined expression can be computed as the limit of a sequence of trace sets.

Familiarity with *Orc* is assumed here; an overview of *Orc* with many examples can be found in a longer version of this paper submitted to

* The second and third authors are partially supported by NSF grant CCF-0448128.

$$\begin{array}{lcl}
\frac{[E(x) \triangle f] \in \mathcal{D}}{E(p) \xrightarrow{0,\tau} [p/x].f} & \text{(DEF)} & \frac{f \xrightarrow{t,a} f' \quad a \neq !m}{f > x > g \xrightarrow{t,a} f' > x > g} & \text{(SEQ1N)} \\
\frac{k \in \Sigma(M, m)}{M(m) \xrightarrow{0,\tau} ?k} & \text{(CALL)} & \frac{f \xrightarrow{t,!m} f'}{f > x > g \xrightarrow{t,\tau} (f' > x > g) \mid [m/x].g} & \text{(SEQ1V)} \\
\frac{(t, m) \in k}{?k \xrightarrow{t,!m} \mathbf{0}} & \text{(RETURN)} & \frac{f \xrightarrow{t,a} f'}{f < x < g \xrightarrow{t,a} f' < x < g^t} & \text{(ASYM1)} \\
\frac{f \xrightarrow{t,a} f'}{f \mid g \xrightarrow{t,a} f' \mid g^t} & \text{(SYM1)} & \frac{g \xrightarrow{t,!m} g'}{f < x < g \xrightarrow{t,\tau} [m/x].f^t} & \text{(ASYM2V)} \\
\frac{g \xrightarrow{t,a} g'}{f \mid g \xrightarrow{t,a} f^t \mid g'} & \text{(SYM2)} & \frac{g \xrightarrow{t,a} g' \quad a \neq !m}{f < x < g \xrightarrow{t,a} f^t < x < g'} & \text{(ASYM2N)}
\end{array}$$

Fig. 1. Timed Semantics of Orc

Theoretical Computer Science [5]. Details of the semantics and proofs of the results can be found in a technical report [6].

2 Timed Operational Semantics

The timed operational semantics of Orc, presented in Figure 1 as a labeled transition system, is based on the asynchronous operational semantics of Orc [1]. A transition $f \xrightarrow{t,a} f'$ means expression f may transition with event a to f' exactly t time units after evaluation starts.

Events are either *publication* events, written $!m$, or *internal* events, written τ . Publication events correspond to the communication of value m to the environment during a transition. Internal events correspond to state changes unobservable to the environment. We refer to both publication and internal events as *base* events.

The times in the transition relation are *relative* to the start of evaluation of the expression. Furthermore, $f \xrightarrow{t,a} f'$ specifies that no other events have occurred during the t units that have elapsed since the beginning of the evaluation of f . We take times to be non-negative reals.

Notation Henceforth, expressions are denoted by f, g, h ; variables by x, y ; events by a, b ; and times by t, s . Parameters, which are either variables or values, are denoted by p . Substitution application is denoted by $[m/x].f$.

Site Calls and Responses Sites are the fundamental units of computation in Orc, and can be thought of as either unreliable remote services, or as locally defined procedures with predictable behavior.

The (CALL) rule in Figure 1 gives the operational semantics of site calls. It specifies that expression $M(m)$ —an invocation of site M with value m —performs an internal event at relative time 0 (i.e., without delay) and transitions to an intermediate expression $?k$. We write $\Sigma(M, m)$ for the set of *handles* that correspond to expression $M(m)$. Each handle describes a possible behavior of site M when it is called with value m .

A handle specifies the relative times at which particular values could potentially be returned by a site call, and also the possibility of perpetual non-response. A handle is a set of pairs (t, m) , where t is a time and m is a value, indicating that m may be returned at time t as a response. Additionally, a handle may also include a distinguished element ω , which indicates non-response.

The (RETURN) rule describes the behavior of handles as a set of potential responses in time. If $(t, m) \in k$, then $?k$ may transition after t units with event $!m$ to $\mathbf{0}$, an expression with no observable transitions. If $\omega \in k$, then it is possible that the handle will never respond, in which case the call blocks indefinitely.

Local sites have predefined and predictable behavior. Consequently, we can define $\Sigma(M, m)$ completely for a local site M and any value m . For example, $\Sigma(\text{let}, m) = \{\{(0, m)\}\}$ and $\Sigma(\text{Rtimer}, t) = \{\{(0, \cdot)\}\}$, where \cdot denotes a signal. These definitions imply that $\text{let}(m)$ engages in $!m$ immediately and that $\text{Rtimer}(t)$ signals after exactly t time units.

Time-shifted Expressions A time-shifted expression, written f^t , is the expression that results from f after t units have elapsed *without occurrence of an event*. When it is not possible for t time units to elapse without f engaging in an event we write $f^t = \perp$, where \perp is an unreachable expression described later. The time-shifted expression f^t , for $t \geq 0$, is defined in Figure 2 based on the structure of f .

$$\begin{aligned}
 (f > x > g)^t &= f^t > x > g & (f \mid g)^t &= f^t \mid g^t \\
 (f < x < g)^t &= f^t < x < g^t & M(x)^t &= M(x) \\
 M(m)^t &= \begin{cases} M(m) & \text{if } t = 0 \\ \perp & \text{otherwise.} \end{cases} & E(p)^t &= \begin{cases} E(p) & \text{if } t = 0 \\ \perp & \text{otherwise.} \end{cases} \\
 ?k^t &= ?(\{(s, m) \mid (t + s, m) \in k\} \cup (k \cap \{\omega\}))
 \end{aligned}$$

Fig. 2. Definition of Time-shifted Expressions

The first three cases, for each of the combinators, are easy to justify informally. Expression $M(x)^t$, where x is a variable, is simply $M(x)$ because the site cannot be invoked until the parameter has a value. Expression $M(m)$, where m is a value, must be invoked at time 0; therefore, $M(m)^0 = M(m)$, whereas $M(m)^t = \perp$ for $t > 0$. The time-shifted handle $?k^t$ may publish m at time s iff $?k$ may publish m at $t+s$; and $?k^t$ includes ω iff $?k$ does. We take $?0$ to mean \perp .

In some cases, it is not possible for t units of time to elapse without occurrence of an event. For example, it is not possible for 1 unit to elapse without an event after the start of evaluation of $let(1)$ because the site call must occur without any delay and $let(1)^1 = \perp$. Any expression which has \perp as a constituent is defined to be \perp . Such an expression is *unreachable*. The transition $f \xrightarrow{t,a} \perp$ for a reachable expression f denotes that f does not engage in the given transition.

Combinator Rules We now describe the rules in Figure 1 that pertain to the three combinators. From $f \xrightarrow{t,a} f'$, we can infer with rule (SYM1) that $f \mid g \xrightarrow{t,a} f' \mid g^t$. Here, g is time-shifted to g^t because t time units have elapsed without an event by g . Note that g^t could be \perp ; in that case, the rule cannot be applied because the corresponding transition is not counted as part of an execution (see Section 2).

When f publishes a value $f \xrightarrow{t,!m} f'$, rule (SEQ1V) creates a new instance of the right side, $[m/x].g$, the expression in which all free occurrences of x in g are replaced by m . The publication $!m$ is hidden, and the entire expression performs a τ event. Note that f and all instances of g are executed in parallel. Because multiple events may occur at the same time instant, it is not guaranteed that the values published by the first instance will precede the values of later instances.

Asymmetric parallel composition is similar to parallel composition, except when g publishes a value m . In this case, rule (ASYM2V) terminates g and x is bound to m in f . One subtlety of these rules is that f may contain both active and blocked subprocesses – any site call that uses x is blocked until g publishes.

Substitution Events We introduce another kind of event, called a *substitution event*, to represent the binding of a value to a free variable in an expression. Substitution events have the form $(t, [m/x])$, where m is a value and x is a variable. The following transition rule introduces substitution events at the top level of expression derivations.

$$f \xrightarrow{t, [m/x]} [m/x].(f^t) \quad (\text{SUBST})$$

Introducing substitution events allows us to distinguish between $\mathbf{0}$ and $\text{let}(x)$. Both $\mathbf{0}$ and $\text{let}(x)$ have transitions due to (SUBST), e.g., with event $(0, [1/x])$. However, $[1/x].\mathbf{0}^0 = \mathbf{0}$ still has no observable transitions, while $[1/x].\text{let}(x)^0 = \text{let}(1)$ publishes 1.

Executions and Traces The execution relation \Rightarrow is similar to the reflexive-transitive closure of the transition relation \rightarrow . However, we need to shift the times in forming the transitive closure. Given $f \xrightarrow{(s,a)} f'$ and $f' \xrightarrow{(t,b)} f''$, we can not claim that $f \xrightarrow{(s,a)(t,b)} f''$, because b occurs $s+t$ units after the evaluation of f starts. We define u_t as the sequence that results from increasing each time component of u by t . Define relation \Rightarrow as follows:

$$f \xRightarrow{\epsilon} f \quad \frac{f \xrightarrow{(t,a)} f'', f'' \xRightarrow{u} f'}{f \xrightarrow{(t,a)u_t} f'}$$

Call u an *execution* of f if $f \xRightarrow{u} f'$ for some $f' \neq \perp$. A *trace* \bar{u} is obtained from execution u by removing each internal event (t, τ) . The definition is lifted pointwise to sets: $\bar{U} = \{\bar{u} \mid u \in U\}$. The *trace set* of f is written $\langle\langle f \rangle\rangle$.

3 Denotational Semantics

We now discuss a denotational semantics of Orc in which the meaning of an expression f is a set of traces $\mu(f)$. The denotation of an expression is determined by the denotations of its subexpressions. For each combinator $*$, we define an overloaded function $U * V$, where U, V and $U * V$ are sets of traces. These functions are then used to define the denotations of Orc expressions. Each Orc combinator is related to its overloaded counterpart in that $\langle\langle f * g \rangle\rangle = \langle\langle f \rangle\rangle * \langle\langle g \rangle\rangle$.

A recursive procedure for construction of denotations $\mu(f)$ is described below. Since an expression may be recursively defined, the denotation is defined as the least upper bound of an infinite ascending chain of trace sets $\mu_i(f)$. Intuitively $\mu_i(f)$, is the trace set of f in which recursively defined subexpressions have been unfolded i times. Let A be the set of all finite sequences of substitution events at time 0, a subset of the denotation of every Orc expression. Define $\mu(f)$ as the union over all $\mu_i(f)$, defined in Figure 3.

$$\mu_0(f) = A$$

$$\mu_{i+1}(f) = \begin{cases} \langle\langle b \rangle\rangle & \text{if } f = b, \text{ a primitive expression} \\ \mu_{i+1}(g) * \mu_{i+1}(h) & \text{if } f = g * h \\ \mu_i([p/x].g) & \text{if } f = E(p) \text{ and } E(x) \triangle g \end{cases}$$

Fig. 3. Definition of the Denotation Function

We have proved that the operational and denotational semantics are equivalent: $\langle\langle f \rangle\rangle = \mu(f)$. This shows that we can reason about the behavior of an Orc program either operationally with the semantics from Section 2, or with μ , which is both compositional and inductive, and thus allows a full treatment of recursively defined expressions.

4 Conclusion

This article develops a timed semantics for Orc in order to provide a simple, well-defined interpretation of Orc in the presence of time. The semantics is shown both operationally and denotationally, where the denotations are traces of events labeled by the time at which they occur. Equivalence of the semantics allows us to reason about Orc programs both operationally and compositionally. The timed semantics enjoys the same properties and identities as the previous asynchronous semantics.

References

1. Kitchin, D., Cook, W.R., Misra, J.: A Language for Task Orchestration and Its Semantic Properties. In: CONCUR. (2006) 477–491
2. Misra, J., Cook, W.R.: Computation Orchestration: A Basis for Wide-Area Computing. *Journal of Software and Systems Modeling* **May** (2006)
3. Rosario, S., Benveniste, A., Haar, S., Jard, C.: Net Systems Semantics of Web Services Orchestrations Modeled in Orc. Technical Report PI 1780, IRISA (2006)
4. Hoare, T., Menzel, G., Misra, J.: A Tree Semantics of an Orchestration Language. In Broy, M., ed.: Proc. of the NATO Advanced Study Institute, Engineering Theories of Software Intensive Systems. NATO ASI Series, Marktoberdorf, Germany (2004)
5. Wehrman, I., Kitchin, D., Cook, W.R., Misra, J.: A Timed Semantics of Orc. Submitted to Theoretical Computer Science. Available at <http://www.cs.utexas.edu/users/iwehrman/pub/tcs07.pdf> (2007)
6. Wehrman, I., Kitchin, D., Cook, W.R., Misra, J.: Properties of the Timed Operational and Denotational Semantics of Orc. Technical Report TR-07-65, University of Texas at Austin, Department of Computer Sciences (2007)