Unpublished working draft – Not for distribution – September 27, 2011 13:48
SVN Id: Orc-token-semantics.tex 295 2011-09-27 18:47:53Z jthywiss

Token Semantics of Orc

John A. Thywissen
The University of Texas at Austin
jthywiss@cs.utexas.edu

[TODO: Other authors here...]
The University of Texas at Austin
x@cs.utexas.edu

Abstract

For Orc, a language for orchestrating concurrent computations, a "token semantics" is presented. This semantic model preserves the original program text during evaluation, rather than reducing the given program to its evaluated value.

Categories and Subject Descriptors D.3.1 [*Programming Languages*]: Formal Definitions and Theory—Semantics, Orc; D.1.3 [*Programming Languages*]: Concurrent Programming—Distributed programming, Parallel programming; D.3.2 [*Programming Languages*]: Language Classifications—Concurrent, distributed, and parallel languages, Orc

General Terms Languages

Keywords structural operational semantics, tokens

1. Introduction

The Orc programming language [Kitchin et al. 2009] is a language for the purpose of orchestrating concurrent computations. The semantics of Orc have been presented in several forms [Kitchin et al. 2006, Wehrman et al. 2008]; however, some research questions need a semantic model that preserves the original program text during evaluation, rather than reducing the given program to its evaluated value.

Therefore, a "token semantics" of Orc is presented here. Threads of execution are represented as tokens which move through the program text, carrying intermediate values related to that thread of execution. A original program's original text is always available by simply erasing all tokens from an executing instance.

2. Orc Language, Compiler and Runtime System

The Orc language is complied from its concrete syntax into the Orc Intermediate Language (OIL), which is a serializable abstract syntax tree (AST) format. OIL is converted into the execution graph, which is what the Orc runtime engine interprets.

The semantic model presented here operates on an abstract syntax close to OIL (version 0.9.7), with the following simplifications: 1) use named variables, instead of OIL's de Bruijn indices; 2) disregard the "field" language feature; 3) disregard Orc/Java site distinction (the keywords `site` vs. `class`); 4) erase type information; 5) disregard the experimental `atomic` and `isolated` constructs.

3. Orc Token Semantics System

The system presented here is based on the structural operational semantics of Plotkin [2004], but instead of carrying environments on the left of a turnstile symbol, environments are carried in tokens. Also, instead of rewriting portions of a program's text to its evaluated value, the resulting value is carried in a token.

3.1 Object Language Symbols

Phrases of object language of the Orc token semantics system include the following kinds of syntactic symbols:

Abstract Lexeme Our abstract syntax of the source program is composed from the following symbols: literal values, identifiers, `stop`, $(,)$, $>$, $|$, $<$, `;`, $\stackrel{\text{def}}{=}$, and site declaration bodies.

For simplicity, assume all identifiers are unique – shadowing does not occur.

A "site declaration body" specifies the location of the interface to an Orc site. This is not elaborated further in this paper.

Value A value, which is either a literal value, a closure, or a value returned by a site call. (In Orc, expressible and denotable values are the same set.)

Environment A mapping from identifiers to values or $\bot$, which indicates a value that has not yet been computed.

Tag Set An ordered set of opaque values (tags) which are used to identify groups of tokens. Sub-group membership is indicated by superset relationships.

Token A structure used by the rewrite rules that carries state. The notation $\bullet^{v}_{\rho,\theta}$ represents a token with environment ρ, tag set θ, and result v.

Prototoken Another structure used by the rewrite rules that carries state. The notation $\odot_{\rho,\theta}$ represents a prototoken with environment ρ and tag set θ.

3.2 Metavariables and Abstract Grammar

The OIL-derived AST definition can be instantiated into the following grammar, which is used by the rewrite rules in following sections:

Expressions $d, d', e, e', l, l', r, r' \in \text{Exp} ::=$

c	– literal value
x	– variable
`stop`	– silent expression
$x(\overline{a}_n)$	– site or function call
$l > x > r$	– sequential
$l \mid r$	– parallel
$l < x < r$	– pruning
$l \,;\, r$	– otherwise
$x(\overline{x}_m) \stackrel{\text{def}}{=} d\ e$	– function definition and scope
$x(\overline{x}_m) \stackrel{\text{def}}{=} s\ e$	– site declaration and scope

| $\overline{\bullet}\ e$ – expression ready for evaluation
| $e\ \overline{\bullet}$ – evaluation complete
| $x\overline{\bullet}(\overline{a}_n)$ – call in progress

where $\overline{\bullet}$ is a possibly empty sequence of:
$\bullet^{v}_{\rho,\theta}$ – token with result
| $\bullet_{\rho,\theta}$ – token without result
| $\odot_{\rho,\theta}$ – prototoken

Literal values (Constants) $c \in \mathrm{Con}$ – integer literals, float literals, string literals, boolean literals, `signal`, and `null`.

Variables $x \in \mathrm{Var}$; $\overline{x}_m$ is a sequence of m (≥ 0) variables, separated by commas

Arguments $a_i \in \mathrm{Con} \cup \mathrm{Var}$; $\overline{a}_n$ is a sequence of n (≥ 0) arguments, separated by commas

Site declaration bodies $s \in \mathrm{SiteDecBod}$, references to sites' locations in the external-to-Orc environment, not detailed here

External values SiteRetVal. These external-to-Orc values are opaque values potentially returned by site calls. They are not detailed here.

Closures Clo, tuples of the form $\langle \theta, m \rangle$ or $\langle s, m \rangle$

Values $v, v' \in \mathrm{Val} = \mathrm{Con} \cup \mathrm{Clo} \cup \mathrm{SiteRetVal}$

Environments $\rho, \rho' : \mathrm{Var} \to \mathrm{Val} \cup \{\bot\}$

Cardinalities (of argument lists) $m, n \in \mathbb{N}$

Tag sets $\theta, \theta' \in \mathcal{P}(\mathrm{Tag})$

3.3 Derivation Rules

These rules operate on the abstract grammar above.

3.3.1 Values and stop

$$\frac{}{\bullet_{\rho,\theta}\ c \to c\ \bullet^{c}_{\rho,\theta}}\ \textsc{Literal}$$

$$\frac{x \in \rho \qquad \rho(x) \neq \bot}{\bullet_{\rho,\theta}\ x \to x\ \bullet^{\rho(x)}_{\rho,\theta}}\ \textsc{Variable}$$

$$\frac{}{\bullet_{\rho,\theta}\ \texttt{stop} \to \texttt{stop}}\ \textsc{Silent}$$

3.3.2 Site Calls

TODO: Variable lookup for args? (here and in func call)

$$\frac{\rho(x) = \langle s, m \rangle \qquad m = n \qquad a_{1\ldots n} \neq \bot}{\bullet_{\rho,\theta}\ x(\overline{a}_n) \xrightarrow{s!\langle \theta', \overline{a}_n \rangle} x\ \bullet_{\rho,\theta'}\ (\overline{a}_n)}\ \textsc{SiteCall-Issue}$$

where $\theta' = \mathrm{newTag}(\theta)$

$$\frac{\text{result } \langle \theta, v \rangle \text{ ready} \qquad v \neq \textbf{null}}{x\ \bullet_{\rho,\theta}\ (\overline{a}_n) \xrightarrow{\theta?\langle v \rangle} x(\overline{a}_n)\ \bullet^{v}_{\rho,\theta'}}\ \textsc{SiteCall-Return}$$

where $\theta' = \mathrm{popTag}(\theta)$

$$\frac{\text{result } \langle \theta, v \rangle \text{ ready} \qquad v = \textbf{null}}{x\ \bullet_{\rho,\theta}\ (\overline{a}_n) \xrightarrow{\theta?\langle v \rangle} x(\overline{a}_n)}\ \textsc{SiteCall-NullReturn}$$

3.3.3 Function Calls

$$\frac{\rho(x) = \langle \theta_0, m \rangle \qquad m = n}{\bullet_{\rho,\theta}\ x(\overline{a}_n) \xrightarrow{\theta_0!\langle \theta', \overline{a}_n \rangle} x \odot_{\rho,\theta'}\ (\overline{a}_n)}\ \textsc{FuncCall-Issue}$$

where $\theta' = \mathrm{newTag}(\theta)$

$$\frac{\text{result } \langle \theta', v \rangle \text{ ready}}{x \odot_{\rho,\theta}\ (\overline{a}_n) \xrightarrow{\theta'?\langle v \rangle} x \odot_{\rho,\theta}\ (\overline{a}_n)\ \bullet^{v}_{\rho,\theta'}}\ \textsc{FuncCall-Return}$$

where $\theta' = \mathrm{popTag}(\theta)$

3.3.4 Sequential Combinator

$$\frac{}{\bullet_{\rho,\theta}\ (l >x> r) \to (\bullet_{\rho,\theta}\ l) >x> r}\ \textsc{Sequential-Enter}$$

$$\frac{}{l\ \bullet^{v}_{\rho,\theta}\ >x> r \to l >x>\ \bullet_{\rho',\theta}\ r}\ \textsc{Sequential-PubL}$$

where $\rho' = \rho[x = v]$

$$\frac{}{l >x> (r\ \bullet^{v}_{\rho,\theta}) \to (l >x> r)\ \bullet^{v}_{\rho',\theta}}\ \textsc{Sequential-PubR}$$

where $\rho' = \rho \setminus \{x\}$

3.3.5 Parallel Combinator

$$\frac{}{\bullet_{\rho,\theta}\ (l \mid r) \to (\bullet_{\rho,\theta}\ l) \mid (\bullet_{\rho,\theta}\ r)}\ \textsc{Parallel-Enter}$$

$$\frac{}{(l\ \bullet^{v}_{\rho,\theta}) \mid r \to (l \mid r)\ \bullet^{v}_{\rho,\theta}}\ \textsc{Parallel-PubL}$$

$$\frac{}{l \mid (r\ \bullet^{v}_{\rho,\theta}) \to (l \mid r)\ \bullet^{v}_{\rho,\theta}}\ \textsc{Parallel-PubR}$$

3.3.6 Pruning Combinator

$$\frac{}{\bullet_{\rho,\theta}\ (l <x< r) \to (\bullet_{\rho',\theta}\ l) <x< (\bullet_{\rho,\theta'}\ r)}\ \textsc{Pruning-Enter}$$

where $\theta' = \mathrm{newTag}(\theta) \quad \rho' = \rho[x = \bot_{\theta'}]$

$$\frac{}{(l\ \bullet^{v}_{\rho,\theta}) <x< r \to (l <x< r)\ \bullet^{v}_{\rho',\theta}}\ \textsc{Pruning-PubL}$$

where $\rho' = \rho \setminus \{x\}$

$$\frac{}{l <x< (r\ \bullet^{v}_{\rho,\theta}) \to l' <x< r'}\ \textsc{Pruning-PubR}$$

where $l' = \mathrm{substEnvValue}(l, \bot_\theta, v) \quad r' = \mathrm{eraseTokens}(\theta, r)$

3.3.7 Otherwise Combinator

$$\frac{}{\bullet_{\rho,\theta}\ (l\ ;\ r) \to (\bullet_{\rho,\theta'}\ l)\ ;\ (\odot_{\rho,\theta'}\ r)}\ \textsc{Otherwise-Enter}$$

where $\theta' = \mathrm{newTag}(\theta)$

$$\frac{\neg\, \mathrm{isLive}(\theta, l)}{l\ ;\ \odot_{\rho,\theta}\ r \to l\ ;\ \bullet_{\rho,\theta}\ r}\ \textsc{Otherwise-NoPub}$$

$$\frac{}{(l\ \bullet^{v}_{\rho,\theta})\ ;\ r \to (l\ ;\ r')\ \bullet^{v}_{\rho,\theta'}}\ \textsc{Otherwise-PubL}$$

where $r' = \mathrm{eraseTokens}(\theta, r) \quad \theta' = \mathrm{popTag}(\theta)$

$$\frac{}{l\ ;\ (r\ \bullet^{v}_{\rho,\theta}) \to (l\ ;\ r)\ \bullet^{v}_{\rho,\theta'}}\ \textsc{Otherwise-PubR}$$

where $\theta' = \mathrm{popTag}(\theta)$

3.3.8 Site and Function Definitions

$$\frac{}{\bullet_{\rho,\theta}\ (x(\overline{x}_m) \stackrel{\text{def}}{=} s)\ e \to (x(\overline{x}_m) \stackrel{\text{def}}{=} s)\ \bullet_{\rho',\theta}\ e} \quad \text{DEF-SITEDEF}$$

where $\rho' = \rho[x = \langle s, m\rangle]$

$$\frac{}{\bullet_{\rho,\theta}\ (x(\overline{x}_m) \stackrel{\text{def}}{=} d)\ e \to (x(\overline{x}_m) \stackrel{\text{def}}{=}\ \odot_{\rho,\theta'}\ d)\ \bullet_{\rho',\theta}\ e} \quad \text{DEF-FUNCDEF}$$

where $\theta' = \text{newTag}(\theta) \quad \rho' = \rho[x = \langle \theta', m\rangle]$

$$\frac{\langle \theta, \theta', \overline{a}_n\rangle \text{ sent}}{(x(\overline{x}_m) \stackrel{\text{def}}{=}\ \odot_{\rho,\theta}\ d)\ e \xrightarrow{\theta?\langle\theta',\overline{a}_n\rangle} (x(\overline{x}_m) \stackrel{\text{def}}{=}\ \odot_{\rho,\theta}\bullet_{\rho',\theta'}\ d)\ e} \quad \text{DEF-ENTERBODY}$$

where $\rho' = \rho[x_1 = a_1, \ldots, x_m = a_n]$

$$\frac{}{(x(\overline{x}_m) \stackrel{\text{def}}{=} d\ \bullet^{v}_{\rho,\theta})\ e \xrightarrow{\theta!\langle v\rangle} (x(\overline{x}_m) \stackrel{\text{def}}{=} d)\ e} \quad \text{DEF-PUBBODY}$$

$$\frac{}{((x(\overline{x}_m) \stackrel{\text{def}}{=} d)\ e\ \bullet^{v}_{\rho,\theta}) \to ((x(\overline{x}_m) \stackrel{\text{def}}{=} d)\ e)\ \bullet^{v}_{\rho',\theta}} \quad \text{DEF-PUBSCOPE}$$

where $\rho' = \rho \setminus \{x\}$

TODO: Are these next two rules desirable?

$$\frac{\neg\,\text{isLive}(\theta, d)}{x\ \odot_{\rho,\theta}\ (\overline{a}_n) \to x(\overline{a}_n)} \quad \text{FUNCCALL-CLEANDEAD}$$

where $d = x$'s def body

$$\frac{\neg\,\text{isLive}(\theta', e)}{(x(\overline{x}_m) \stackrel{\text{def}}{=}\ \odot_{\rho,\theta}\ d)\ e \to (x(\overline{x}_m) \stackrel{\text{def}}{=} d)\ e} \quad \text{DEF-CLEANDEAD}$$

where $\theta' = \text{popTag}(\theta)$

3.3.9 Congruence Rules

TODO: Use context redexes to eliminate the need for congruence rules

$$\frac{l \to l'}{l >x> r \to l' >x> r} \quad \text{SEQUENTIAL-CONGRUL}$$

$$\frac{r \to r'}{l >x> r \to l >x> r'} \quad \text{SEQUENTIAL-CONGRUR}$$

$$\frac{l \to l'}{l \mid r \to l' \mid r} \quad \text{PARALLEL-CONGRUL}$$

$$\frac{r \to r'}{l \mid r \to l \mid r'} \quad \text{PARALLEL-CONGRUR}$$

$$\frac{l \to l'}{l <x< r \to l' <x< r} \quad \text{PRUNING-CONGRUL}$$

$$\frac{r \to r'}{l <x< r \to l <x< r'} \quad \text{PRUNING-CONGRUR}$$

$$\frac{l \to l'}{l\ ;\ r \to l'\ ;\ r} \quad \text{OTHERWISE-CONGRL}$$

$$\frac{r \to r'}{l\ ;\ r \to l\ ;\ r'} \quad \text{OTHERWISE-CONGRR}$$

$$\frac{e \to e'}{(x(\overline{x}_m) \stackrel{\text{def}}{=} s)\ e \to (x(\overline{x}_m) \stackrel{\text{def}}{=} s)\ e'} \quad \text{DEF-SITECONGRU}$$

$$\frac{d \to d'}{(x(\overline{x}_m) \stackrel{\text{def}}{=} d)\ e \to (x(\overline{x}_m) \stackrel{\text{def}}{=} d')\ e} \quad \text{DEF-CONGRUBODY}$$

$$\frac{e \to e'}{(x(\overline{x}_m) \stackrel{\text{def}}{=} d)\ e \to (x(\overline{x}_m) \stackrel{\text{def}}{=} d)\ e'} \quad \text{DEF-CONGRUSCOPE}$$

$$\frac{e \to e'}{\bullet_{\rho,\theta}\ e \to \bullet_{\rho,\theta}\ e'} \quad \text{TOKEN-CONGRU}$$

$$\frac{}{\bullet^{v}_{\rho,\theta}\ (\bullet^{v'}_{\rho',\theta'}\ e) \to \bullet^{v'}_{\rho',\theta'}\ (\bullet^{v}_{\rho,\theta}\ e)} \quad \text{TOKEN-COMMUTE}$$

$$\frac{e \to e'}{\odot_{\rho,\theta}\ e \to \odot_{\rho,\theta}\ e'} \quad \text{PROTOTOKEN-CONGRU}$$

$$\frac{}{\odot_{\rho,\theta}\ (\odot_{\rho',\theta'}\ e) \to \odot_{\rho',\theta'}\ (\odot_{\rho,\theta}\ e)} \quad \text{PROTOTOKEN-COMMUTE}$$

$$\frac{}{\bullet^{v}_{\rho,\theta}\ (\odot_{\rho',\theta'}\ e) \to \odot_{\rho',\theta'}\ (\bullet^{v}_{\rho,\theta}\ e)} \quad \text{TOKENPROTO-COMMUTE}$$

$$\frac{}{\bullet^{v}_{\rho,\theta}\ (e\ \bullet^{v'}_{\rho',\theta'}) \to (\bullet^{v}_{\rho,\theta}\ e)\ \bullet^{v'}_{\rho',\theta'}} \quad \text{TOKEN-ASSOC}$$

3.4 Auxiliary Functions

$$\text{newTag}(\theta) = \{\text{new tag value}\} \cup \theta$$

$\text{popTag}(\theta) = \theta \setminus \theta_1$, where θ_1 is the most recently added element

$$\neg\,\text{isLive}(\theta, e) \Leftrightarrow \text{eraseTokens}(\theta, e) = e$$

TODO: Formalize the following two def'ns

$\text{eraseTokens}(\theta, e)$ = walk lexemes in e and erase all tokens $\bullet_{,\theta'}$ and all prototokens $\odot_{,\theta'}$, where $\theta' \supseteq \theta$. For all function calls $x\ \odot_{,\theta'}\ (\overline{a}_n)$ found in the expression e, apply eraseTokens to the function definition body.

$\text{substEnvValue}(e, \bot_\theta, v)$ = walk lexemes in e, updating tokens' environments, changing any variable bound to $\bot_\theta$ to be bound to v.

3.5 Program Launch and Results

In the following, e is specifically the entire program.

$$\frac{\text{run program signal}}{e \to \bullet_{\varnothing,\theta}\ e} \quad \text{PROGRAM-RUN}$$

where $\theta = \text{newTag}(\varnothing)$

$$\frac{}{e\ \bullet^{v}_{\rho,\theta} \xrightarrow{\text{publish } v} e} \quad \text{PROGRAM-PUBLISH}$$

3.6 Actions (Transition labels)

TODO: Need to rework the transition labels – currently, they are the not the conventional use of labels.

Let $a!b$ mean "send", under the tag a, the value b. Sending can be modeled as adding the pair (a, b) to a "sent" multiset.

Let $a?x$ mean "receive" a value sent under the tag a, into the variable x. Receiving can be modeled as selecting a pair (a, b) in the "sent" multiset (matching on a), removing it from the multiset, and letting $x = b$ in the rest of the rule.

$s!\langle\theta, \overline{a}_n\rangle$
$\theta!\langle\theta', \overline{a}_n\rangle$
$\theta?\langle\theta', \overline{a}_n\rangle$

$\theta!\langle v\rangle$
$\theta?\langle v\rangle$
publish v is an event sent to the environment to indicate a result value of program execution, not further specified here.

4. Properties of the Semantics

4.1 Non-rewriting

Property. In all rules, the eraseTokens result of the lhs and rhs of every transition is the same. In other words, the program text is not changed by the rules when one disregards tokens and prototokens.

This property is proved for the rules by cases. The rules are presented again below, with the results of eraseTokens applied to the transitions. Most resultant rules are of the form $t \to t$, so it is an immediate conclusion that no rewrites occur for these rules. Many congruence rules are of the form $t \to t' \Rightarrow t \to t'$, so these rules cause no rewrites if no other rule causes a rewrite. These two patterns cover all the rules except PRUNING-PUBR and OTHERWISE-PUBL.

In PRUNING-PUBR, l has its tokens' environments updated to produce l'. Since l contains no tokens, this is a no-op, and $l = l'$.

In both PRUNING-PUBR and OTHERWISE-PUBL, r is rewritten to r', which is r with certain tokens erased. Since there are no tokens in r in this case, $r = r'$.

4.1.1 Values and stop

$$\frac{}{c \to c}\ \textsc{Literal}$$

$$\frac{}{x \to x}\ \textsc{Variable}$$

$$\frac{}{\texttt{stop} \to \texttt{stop}}\ \textsc{Silent}$$

4.1.2 Site Calls

$$\frac{a_{1\ldots n} \neq \bot}{x(\overline{a}_n) \xrightarrow{s!\langle\theta',\overline{a}_n\rangle} x(\overline{a}_n)}\ \textsc{SiteCall-Issue}$$

where $\theta' = \mathrm{newTag}(\theta)$

$$\frac{}{x(\overline{a}_n) \xrightarrow{\theta?\langle v\rangle} x(\overline{a}_n)}\ \textsc{SiteCall-Return}$$

$$\frac{}{x(\overline{a}_n) \xrightarrow{\theta?\langle v\rangle} x(\overline{a}_n)}\ \textsc{SiteCall-NullReturn}$$

4.1.3 Function Calls

$$\frac{}{x(\overline{a}_n) \xrightarrow{\theta_0!\langle\theta',\overline{a}_n\rangle} x(\overline{a}_n)}\ \textsc{FuncCall-Issue}$$

where $\theta' = \mathrm{newTag}(\theta)$

$$\frac{}{x(\overline{a}_n) \xrightarrow{\theta'?\langle v\rangle} x(\overline{a}_n)}\ \textsc{FuncCall-Return}$$

4.1.4 Sequential Combinator

$$\frac{}{l >x> r \to l >x> r}\ \textsc{Sequential-Enter}$$

$$\frac{}{l >x> r \to l >x> r}\ \textsc{Sequential-PubL}$$

$$\frac{}{l >x> r \to l >x> r}\ \textsc{Sequential-PubR}$$

4.1.5 Parallel Combinator

$$\frac{}{l \mid r \to l \mid r}\ \textsc{Parallel-Enter}$$

$$\frac{}{l \mid r \to l \mid r}\ \textsc{Parallel-PubL}$$

$$\frac{}{l \mid r \to l \mid r}\ \textsc{Parallel-PubR}$$

4.1.6 Pruning Combinator

$$\frac{}{l \mathrel{<}x\mathrel{<} r \to l \mathrel{<}x\mathrel{<} r} \text{ PRUNING-ENTER}$$

$$\frac{}{l \mathrel{<}x\mathrel{<} r \to l \mathrel{<}x\mathrel{<} r} \text{ PRUNING-PUBL}$$

$$\frac{}{l \mathrel{<}x\mathrel{<} r \to l' \mathrel{<}x\mathrel{<} r'} \text{ PRUNING-PUBR}$$

where $l' = \text{substEnvValue}(l, \bot_\theta, v) \quad r' = \text{eraseTokens}(\theta, r)$

4.1.7 Otherwise Combinator

$$\frac{}{l \mathbin{;} r \to l \mathbin{;} r} \text{ OTHERWISE-ENTER}$$

$$\frac{}{l \mathbin{;} r \to l \mathbin{;} r} \text{ OTHERWISE-NOPUB}$$

$$\frac{}{l \mathbin{;} r \to l \mathbin{;} r'} \text{ OTHERWISE-PUBL}$$

where $r' = \text{eraseTokens}(\theta, r)$

$$\frac{}{l \mathbin{;} r \to l \mathbin{;} r} \text{ OTHERWISE-PUBR}$$

4.1.8 Site and Function Definitions

$$\frac{}{(x(\overline{x}_m) \stackrel{\text{def}}{=} s)\ e \to (x(\overline{x}_m) \stackrel{\text{def}}{=} s)\ e} \text{ DEF-SITEDEF}$$

$$\frac{}{(x(\overline{x}_m) \stackrel{\text{def}}{=} d)\ e \to (x(\overline{x}_m) \stackrel{\text{def}}{=} d)\ e} \text{ DEF-FUNCDEF}$$

$$\frac{}{(x(\overline{x}_m) \stackrel{\text{def}}{=} d)\ e \xrightarrow{\theta?\langle \theta', \overline{a}_n \rangle} x(\overline{x}_m) \stackrel{\text{def}}{=} d)\ e} \text{ DEF-ENTERBODY}$$

$$\frac{}{(x(\overline{x}_m) \stackrel{\text{def}}{=} d)\ e \xrightarrow{\theta!\langle v \rangle} (x(\overline{x}_m) \stackrel{\text{def}}{=} d)\ e} \text{ DEF-PUBBODY}$$

4.1.9 Congruence Rules

$$\frac{l \to l'}{l \mathrel{>}x\mathrel{>} r \to l' \mathrel{>}x\mathrel{>} r} \text{ SEQUENTIAL-CONGRUL}$$

$$\frac{r \to r'}{l \mathrel{>}x\mathrel{>} r \to l \mathrel{>}x\mathrel{>} r'} \text{ SEQUENTIAL-CONGRUR}$$

$$\frac{l \to l'}{l \mid r \to l' \mid r} \text{ PARALLEL-CONGRUL}$$

$$\frac{r \to r'}{l \mid r \to l \mid r'} \text{ PARALLEL-CONGRUR}$$

$$\frac{l \to l'}{l \mathrel{<}x\mathrel{<} r \to l' \mathrel{<}x\mathrel{<} r} \text{ PRUNING-CONGRUL}$$

$$\frac{r \to r'}{l \mathrel{<}x\mathrel{<} r \to l \mathrel{<}x\mathrel{<} r'} \text{ PRUNING-CONGRUR}$$

$$\frac{l \to l'}{l \mathbin{;} r \to l' \mathbin{;} r} \text{ OTHERWISE-CONGRL}$$

$$\frac{r \to r'}{l \mathbin{;} r \to l \mathbin{;} r'} \text{ OTHERWISE-CONGRR}$$

$$\frac{e \to e'}{(x(\overline{x}_m) \stackrel{\text{def}}{=} s)\ e \to (x(\overline{x}_m) \stackrel{\text{def}}{=} s)\ e'} \text{ DEF-SITECONGRU}$$

$$\frac{d \to d'}{(x(\overline{x}_m) \stackrel{\text{def}}{=} d)\ e \to (x(\overline{x}_m) \stackrel{\text{def}}{=} d')\ e} \text{ DEF-CONGRUBODY}$$

$$\frac{e \to e'}{(x(\overline{x}_m) \stackrel{\text{def}}{=} d)\ e \to (x(\overline{x}_m) \stackrel{\text{def}}{=} d)\ e'} \text{ DEF-CONGRUSCOPE}$$

$$\frac{e \to e'}{e \to e'} \text{ TOKEN-CONGRU}$$

$$\frac{}{e \to e} \text{ TOKEN-COMMUTE}$$

$$\frac{e \to e'}{e \to e'} \text{ PROTOTOKEN-CONGRU}$$

$$\frac{}{e \to e} \text{ PROTOTOKEN-COMMUTE}$$

$$\frac{}{e \to e} \text{ TOKENPROTO-COMMUTE}$$

$$\frac{}{e \to e} \text{ TOKEN-ASSOC}$$

4.2 Entry-Exit Tag Set Equality

Property. For every expression type, the tag set of a token leaving is equal to the tag set of the corresponding token at entry to the expression. (I.e., non-leakage of tags.)

An inspection of the rules shows that tokens' tag sets are only modified in the rules for site/function calls/returns, the pruning combinator, and the otherwise combinator. These cases are examined below.

We use the identity

$$\text{popTag}(\text{newTag}(\theta)) = \theta$$

below.

4.2.1 Site Calls

Rule SITECALL-RETURN writes a token leaving a site call expression with a tag set of $\text{popTag}(\theta)$, where θ is the tag set of the token when it was waiting for the site call return. This tag set, by rule SITECALL-ISSUE is $\text{newTag}(\theta)$, where this θ is the tag set of the token entering the site call expression. So tokens leaving site calls have the tag set $\text{popTag}(\text{newTag}(\theta))$, i.e., unchanged from entry.

4.2.2 Function Calls

Analogously with site calls, rule FUNCCALL-RETURN writes a token leaving a function call expression with a tag set of $\text{popTag}(\theta)$, where θ is the tag set of the prototoken for the function call return. This tag set, by rule FUNCCALL-ISSUE is $\text{newTag}(\theta)$, where this θ is the tag set of the token entering the function call expression. So tokens leaving function calls have the tag set $\text{popTag}(\text{newTag}(\theta))$, i.e., unchanged from entry.

4.2.3 Pruning Combinator

Tokens leaving a pruning combinator expression (rule PRUNING-PUBL) have the tag set of the token leaving the left-hand side expression l. Rule PRUNING-ENTER places a token entering l with the tag set of the token entering the combinator expression. So, assuming evaluation of the expression l preserves this tag set, tokens leaving a pruning combinator expression have the tag set of the entering token. (Note that the tag sets of tokens in the right-hand side expression r do not leave the expression r.)

4.2.4 Otherwise Combinator

A token leaves an otherwise combinator expression by execution of rule OTHERWISE-PUBL or OTHERWISE-PUBR. Both rules write a token leaving an otherwise combinator expression with a tag set of $\mathrm{popTag}(\theta)$, where θ is the tag set of the token leaving a sub-expression (l or r). Rule OTHERWISE-ENTER places tokens entering the l and r sub-expressions with tag sets $\mathrm{newTag}(\theta)$, where this θ is the tag set of the token entering the otherwise combinator expression. So, assuming evaluation of the sub-expressions l and r preserve this tag set, tokens leaving otherwise combinator expressions have the tag set $\mathrm{popTag}(\mathrm{newTag}(\theta))$, i.e., unchanged from entry.

5. Orc Algebraic Identities

Most of these identities have been published previously [Misra 2005, Hoare et al. 2005, Kitchin et al. 2006, Misra and Cook 2007, Cook and Misra 2008, Wehrman et al. 2008]; however, there are a few additional identities listed below.

5.1 Associativity

$$(f >x> g) >y> h = f >x> (g >y> h), \quad \text{if } x \notin \mathrm{FV}(h) \qquad \text{(Seq-Assoc)}$$

$$(f \mid g) \mid h = f \mid (g \mid h) \qquad \text{(Par-Assoc)}$$

$$(f \,;\, g) \,;\, h = f \,;\, (g \,;\, h) \qquad \text{(Other-Assoc)}$$

The pruning combinator is not associative.

5.2 Commutativity

$$f \mid g = g \mid f \qquad \text{(Par-Commut)}$$

The sequential, pruning, and otherwise combinators are not commutative.

5.3 Identity

$$\texttt{signal} \gg f = f \qquad \text{(Seq-L-Ident)}$$

$$f >x> x = f \qquad \text{(Seq-R-Ident)}$$

$$\texttt{stop} \mid f = f \qquad \text{(Par-L-Ident)}$$

$$f \mid \texttt{stop} = f \qquad \text{(Par-R-Ident)}$$

$$f \ll \texttt{signal} = f \qquad \text{(Prun-R-Ident1)}$$

$$f \ll \texttt{stop} = f \qquad \text{(Prun-R-Ident2)}$$

$$\texttt{stop} \,;\, f = f \qquad \text{(Other-L-Ident)}$$

$$f \,;\, \texttt{stop} = f \qquad \text{(Other-R-Ident)}$$

The pruning combinator has no left identity element.

5.4 Zero

$$\texttt{stop} >x> g = \texttt{stop} \qquad \text{(Seq-L-Zero)}$$

The parallel and otherwise combinators have no zero, and the pruning combinator has no right zero. The sequential combinator has a right zero and the pruning a right only for purely functional subexpressions (see below).

5.5 Distribution

$$(f \mid g) >x> h = (f >x> h) \mid (g >x> h) \qquad \text{(Seq-R-Dist-Par)}$$

5.6 Contraction

$$f >x> g = f \gg g, \quad \text{if } x \notin \mathrm{FV}(g) \qquad \text{(Seq-Contract)}$$

$$f <x< g = f \ll g, \quad \text{if } x \notin \mathrm{FV}(f) \qquad \text{(Prune-Contract)}$$

5.7 Telescoping (suggested by David)

$$f <x< g = f <x< (y <y< g) \qquad \text{(Prun-Tele)}$$

The sequential combinator can be telescoped using Seq-R-Ident.

5.8 Moving Subexpressions in to/out of Pruning's lhs Scope

$$(f <x< g) >y> h = (f >y> h) <x< g, \quad \text{if } x \notin \mathrm{FV}(h) \qquad \text{(Seq-Prun-Assoc)}$$

$$f \mid (g <x< h) = (f \mid g) <x< h, \quad \text{if } x \notin \mathrm{FV}(f) \qquad \text{(Par-Prun-Assoc)}$$

5.9 Identities using Meta-data

Several proposed Orc extensions intend to make meta-data available about certain types of expressions, primarily site calls. This additional information permits the use of additional identities, shown below.

Here, we say an expression is *idempotent* iff its effects are equivalent whether evaluated once or more than once. An expression is *single-valued partial* iff it always publishes exactly one or zero values. A large number of expressions are both idempotent and single-valued partial, so it may be useful to study such expressions' properties. We abbreviate single-valued idempotent partial as *s.i.p.*

5.10 Identities for S.I.P. Subexpressions

$$f \gg f = f, \quad \text{if } f \text{ is s.i.p.} \qquad \text{(Seq-Idem)}$$

$$f \ll f = f, \quad \text{if } f \text{ is s.i.p.} \qquad \text{(Prun-Idem)}$$

$$f \,;\, f = f, \quad \text{if } f \text{ is s.i.p.} \qquad \text{(Other-Idem)}$$

$$(f \,;\, g) \ll f = f \,;\, g, \quad \text{if } f \text{ is s.i.p.} \qquad \text{(Name?)}$$

$$f \,;\, (f \mid g) = f \,;\, g, \quad \text{if } f \text{ is s.i.p.} \qquad \text{(Name?)}$$

$$(f \gg g) \,;\, f = f \gg g, \quad \text{if } f \text{ is s.i.p.} \qquad \text{(Name?)}$$

$$(f \mid g) \,;\, f = f \mid g, \quad \text{if } f \text{ is s.i.p.} \qquad \text{(Name?)}$$

$$(f \ll g) \,;\, f = f \ll g, \quad \text{if } f \text{ is s.i.p.} \qquad \text{(Name?)}$$

5.11 Monoids over s.i.p. Subexpressions

Since the parallel and otherwise combinators are associative and have an identity element:

$\langle E,\ \mid \rangle$ forms a (commutative) monoid, where E is the set of s.i.p. Orc expressions.

$\langle E,\ ; \rangle$ forms an idempotent monoid (but not a semilattice), where E is the set of s.i.p. Orc expressions.

5.12 "Semi"-absorption for S.I.P. Subexpressions

The following identities are similar to absorption laws; however, the identities don't hold in both left and right forms.

$$f \gg (f \,;\, g) = f \,;\, (f \gg g) = f, \quad \text{if } f \text{ is s.i.p.} \qquad \text{(Name?)}$$

$$f \ll (f \,;\, g) = f \,;\, (f \ll g) = f, \quad \text{if } f \text{ is s.i.p.} \qquad \text{(Name?)}$$

5.13 Identities for Purely Functional Subexpressions

An expression is *purely functional* iff its has no effects beyond publishing zero or more values and halting.

$$f >x> \texttt{stop} = \texttt{stop}, \quad \text{if } f \text{ is purely functional} \qquad \text{(Seq-R-Zero)}$$

$$\texttt{stop} <x< f = \texttt{stop}, \quad \text{if } f \text{ is purely functional} \qquad \text{(Prun-L-Zero)}$$

Acknowledgments

[TODO:If not co-authors] Prof. William Cook, Prof. Jayadev Misra, David Kitchin, and Adrian Quark for reviewing drafts and making many valuable suggestions.

References

COOK, W. R. AND MISRA, J. 2008. Structured interacting computations. In *Software-Intensive Systems and New Computing Paradigms*, M. Wirsing, J.-P. Banâtre, M. Hölzl, and A. Rauschmayer, Eds. Lecture Notes in Computer Science, vol. 5380. Springer, 139–145.

HOARE, T., MENZEL, G., AND MISRA, J. 2005. A tree semantics of an orchestration language. In *Engineering Theories of Software Intensive Systems* (Marktoberdorf, Germany, 3–15 Aug 2004), M. Broy, D. Harel, T. Hoare, and J. Grünbauer, Eds. NATO Science Series II: Mathematics, Physics and Chemistry, vol. 195. Springer, 331–350.

KITCHIN, D., COOK, W. R., AND MISRA, J. 2006. A language for task orchestration and its semantic properties. In *CONCUR 2006 – Concurrency Theory* (Bonn, Germany, 27–30 Aug 2006), C. Baier and H. Hermanns, Eds. Lecture Notes in Computer Science, vol. 4137. Springer, 477–491.

KITCHIN, D., QUARK, A., COOK, W. R., AND MISRA, J. 2009. The Orc programming language. In *Proceedings of FMOODS/FORTE 2009* (Lisbon, Portugal, 9–11 Jun 2009), D. Lee, A. Lopes, and A. Poetzsch-Heffter, Eds. Lecture Notes in Computer Science, vol. 5522. Springer, 1–25.

MISRA, J. 2005. Computation orchestration: A basis for wide-area computing. In *Engineering Theories of Software Intensive Systems* (Marktoberdorf, Germany, 3–15 Aug 2004), M. Broy, D. Harel, T. Hoare, and J. Grünbauer, Eds. NATO Science Series II: Mathematics, Physics and Chemistry, vol. 195. Springer, 285–330.

MISRA, J. AND COOK, W. R. 2007. Computation orchestration: A basis for wide-area computing. *Software and Systems Modeling 6,* 1 (23 Mar), 83–110.

PLOTKIN, G. D. 2004. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming 60–61*, 17–139.

WEHRMAN, I., KITCHIN, D., COOK, W. R., AND MISRA, J. 2008. A timed semantics of Orc. *Theoretical Computer Science 402,* 2–3 (Aug), 234–248.