A Compositional Trace Semantics for Orc

Dimitrios Vardoulakis and Mitchell Wand
College of Computer and Information Science
Northeastern University

Overview of Orc

- Sites: autonomous computing units that do arithmetic, printing etc
- Operators to combine the executions of sites e.g. parallel composition
- Recursive declarations to express non-terminating processes

→ can encode many popular concurrent programming patterns

Overview of Orc

- The simplest Orc process is a site call:

```
Factorize(N)

Reddit(Oct'20)
```

Overview of Orc

- Symmetric Composition `(f | g)` : evaluate f and g in parallel, no interaction between them

```
Factorize(N) | Reddit(Oct’20)
```

Overview of Orc

- Sequencing `(f >x> g)` : evaluate f, when it publishes spawn a new instance of g in parallel

```
(Factorize(N) | Reddit(Oct'20)) >x> Print(x)
```

Overview of Orc

- Asymmetric Composition `(f where x:∈ g)`: evaluate f and g in parallel, when g publishes it sends the value to f and terminates

```
Print(x) where x:∈
(Factorize(N)|Reddit(Oct'20))
```

Overview of Orc

- Recursive Declarations `(E`$_i$`(x) = f`$_i$`)` : We can express processes that don't terminate

We define:

```
DOS(x) = Ping(x) | DOS(x)
```

And then call:

```
DOS(ip)
```

Syntax and Operational Semantics

Syntax

Program $\quad P ::= D_1, \ldots, D_k \ \ \mathtt{in} \ \ e$

Expression
$$
\begin{aligned}
e ::= {} & 0 \mid M(p) \mid \mathtt{let}(p) \\
\mid {} & E_i(p) \mid (e_1 | e_2) \\
\mid {} & e_1 > x > e_2 \\
\mid {} & e_1 \ \mathtt{where} \ x :\in e_2
\end{aligned}
$$

Actual Parameter $\quad p ::= x \mid v$

Declaration $\quad D_i ::= E_i(x) = e$

Operational Semantics

- Labeled transitions

$$\Delta, \Gamma \vdash \; f \; \xrightarrow{a} \; f'$$

Operational Semantics

$$(\mathrm{SITEC})\quad \frac{}{\Delta, \Gamma \vdash M(v) \xrightarrow{M_k(v)} ?k}\ k \text{ fresh}$$

$$(\mathrm{SITEC\text{-}VAR})\quad \frac{}{\Delta, \Gamma \vdash M(x) \xrightarrow{[v/x]} M(v)}\ \Gamma(x) = v$$

$$(\mathrm{SITERET})\quad \frac{}{\Delta, \Gamma \vdash ?k \xrightarrow{k?v} let(v)}$$

Operational Semantics

$$\text{(LET)} \quad \frac{}{\Delta, \Gamma \vdash \ let(v) \xrightarrow{!v} \mathbf{0}}$$

$$\text{(LET-VAR)} \quad \frac{}{\Delta, \Gamma \vdash \ let(x) \xrightarrow{[v/x]} let(v)} \ \Gamma(x) = v$$

Operational Semantics

$$\text{(DEF)} \qquad \frac{}{\Delta, \Gamma \vdash E_i(v) \xrightarrow{\tau} [v/x]f_i} \; (E_i(x) \triangleq f_i) \in \Delta$$

$$\text{(DEF-VAR)} \qquad \frac{}{\Delta, \Gamma \vdash E_i(x) \xrightarrow{[v/x]} E_i(v)} \; \begin{array}{c}(E_i(x) \triangleq f_i) \in \Delta, \\ \Gamma(x) = v\end{array}$$

Operational Semantics

(SYM-L)
$$\frac{\Delta, \Gamma \vdash f \xrightarrow{a} f'}{\Delta, \Gamma \vdash f \mid g \xrightarrow{a} f' \mid g}$$

(SYM-R)
$$\frac{\Delta, \Gamma \vdash g \xrightarrow{a} g'}{\Delta, \Gamma \vdash f \mid g \xrightarrow{a} f \mid g'}$$

Operational Semantics

(SEQ) $$\frac{\Delta, \Gamma \vdash \ f \ \xrightarrow{a} \ f'}{\Delta, \Gamma \vdash \ f \ {>}x{>} \ g \ \xrightarrow{a} \ f' \ {>}x{>} \ g} \ a \neq !v$$

(SEQ-P) $$\frac{\Delta, \Gamma \vdash \ f \ \xrightarrow{!v} \ f'}{\Delta, \Gamma \vdash \ f \ {>}x{>} \ g \ \xrightarrow{\tau} \ (f' \ {>}x{>} \ g) \mid [v/x]g}$$

Operational Semantics

(ASYM-L) $$\frac{\Delta, \Gamma \vdash \ f \ \xrightarrow{a} \ f'}{\Delta, \Gamma \vdash \ f \ \mathbf{where}\ x :\in g \ \xrightarrow{a} \ f' \ \mathbf{where}\ x :\in g} \ a \neq [v/x]$$

(ASYM-R) $$\frac{\Delta, \Gamma \vdash \ g \ \xrightarrow{a} \ g'}{\Delta, \Gamma \vdash \ f \ \mathbf{where}\ x :\in g \ \xrightarrow{a} \ f \ \mathbf{where}\ x :\in g'} \ a \neq !v$$

(ASYM-P) $$\frac{\Delta, \Gamma \vdash \ g \ \xrightarrow{!v} \ g'}{\Delta, \Gamma \vdash \ f \ \mathbf{where}\ x :\in g \ \xrightarrow{\tau} \ [v/x]f}$$

More examples: Fork-join Parallelism

- Launch two threads and wait for both to complete before you proceed

(let(x,y) where x:∈ M(v_1)) where y:∈ N(v_2)

More examples: Parallel-or

- Call M and N in parallel, if one replies “true” don’t wait for the other to reply

```
((let(z) where z:∈ ift(x)|ift(y)|or(x,y)
  where x:∈ M(v1))
  where y:∈ N(v2))
```

Denotational Semantics

Denotational Semantics

- Trace Semantics
- Based on complete partial orders

$$[\![f]\!] : [Fenv \longrightarrow [Env \longrightarrow P]]$$

Denotational Semantics

- `([2/x] !2)` is a possible trace of `let(x)`
- `(!3 !5) and (!5 !3)` are possible traces of `(let(3) | let(5))`
- `(let(x) | let(7)) where x:∈ let(2)`

  In a trace of `(let(x) | let(7))`, how do we know if `let(7)` publishes before `let(2)` sends a value?

Denotational Semantics

- Receive events show what part of the trace is independent of some variable!

e.g. `([2/x] !2 !7)` and `(!7 [2/x] !2)` are possible traces of `(let(x) | let(7))`

`!7` is independent of `[2/x]` because it can happen before `[2/x]`

Then, `(τ !2 !7)` and `(!7 τ !2)` are possible traces of `(let(x)|let(7)) where x:∈ let(2)` but not `(!7 !2 τ)`

Denotational Semantics

We do not need this information in `(let(2) >x> (let(x) | let(7))` because when the right hand side is launched, `x` always has a value

Then, do we put the receive event in the traces of `let(x)` or not?

Two kinds of bindings for variables in the environment

Denotational Semantics

$$
\begin{aligned}
[\![\mathbf{0}]\!] &= \lambda\varphi.\lambda\rho.\{\varepsilon\} \\
[\![let(v)]\!] &= \lambda\varphi.\lambda\rho.\{!v\}_{\mathrm{p}} \\
[\![let(x)]\!] &= \lambda\varphi.\lambda\rho.\textbf{case } \rho(x) \textbf{ of } \begin{array}[t]{l} \mathrm{Absent}.\{\varepsilon\} \\ \flat v.\{!v\}_{\mathrm{p}} \\ \natural v.\{[v/x]\ !v\}_{\mathrm{p}} \end{array}
\end{aligned}
$$

Denotational Semantics

$$[\![M(v)]\!] = \lambda\varphi.\lambda\rho.\{\, M_k(v)\ k?w\ !w \mid k \text{ fresh}\,,\ w \in \mathit{Val} \}_{\mathrm{p}}$$

$$\begin{aligned}
[\![M(x)]\!] = \lambda\varphi.\lambda\rho.\textbf{case}\ \rho(x)\ \textbf{of}\ \ & \text{Absent}.\{\varepsilon\} \\
& \flat v.\{\, M_k(v)\ k?w\ !w \mid k \text{ fresh}\,,\ w \in \mathit{Val} \}_{\mathrm{p}} \\
& \natural v.\{\, [v/x]\ M_k(v)\ k?w\ !w \mid k \text{ fresh}\,,\ w \in \mathit{Val} \}_{\mathrm{p}}
\end{aligned}$$

$$[\![?k]\!] = \lambda\varphi.\lambda\rho.\{\, k?w\ !w \mid w \in \mathit{Val} \}_{\mathrm{p}}$$

Denotational Semantics

$$[\![E_i(v)]\!] = \lambda\varphi.\lambda\rho.\{\,\tau\,t \mid t \in \varphi_i(v)\}_{\mathrm{p}}$$

$$[\![E_i(x)]\!] = \lambda\varphi.\lambda\rho.\mathbf{case}\ \rho(x)\ \mathbf{of}\ \ \mathrm{Absent}.\{\varepsilon\}$$

$$v.\{\,\tau\,t \mid t \in \varphi_i(v)\}_{\mathrm{p}}$$

$$v.\{\,[v/x]\,\tau\,t \mid t \in \varphi_i(v)\}_{\mathrm{p}}$$

Denotational Semantics

$$[\![h \mid g]\!] = \lambda\varphi.\lambda\rho.\ [\![h]\!]\varphi\rho \parallel [\![g]\!]\varphi\rho$$

Merge:

$$t_1 \parallel t_2 \triangleq \begin{cases} \{t_1\} & t_2 = \varepsilon \\ \{t_2\} & t_1 = \varepsilon \\ a(t_1' \parallel t_2) \cup b(t_1 \parallel t_2') & t_1 = at_1' \text{ and } t_2 = bt_2' \end{cases}$$

Merge trace-sets:

$$T_1 \parallel T_2 \triangleq \bigcup_{t_1 \in T_1, t_2 \in T_2} t_1 \parallel t_2$$

Denotational Semantics

$$[\![h >x> g]\!] = \lambda\varphi.\lambda\rho. \bigcup_{s \in [\![h]\!]\varphi\rho} s \gg \lambda v.[\![g]\!]\varphi\rho[x = \flat v]$$

Sequencing combinator:

$$s \gg F = \begin{cases} \{s\} & \text{no publ. in } s \\ s_1\, \tau\, ((s_2 \gg F) \,\|\, F(v)) & s \equiv s_1 ! v s_2 \text{ , no publ. in } s_1 \end{cases}$$

Denotational Semantics

$$[\![h \ \mathbf{where} \ x :\in g]\!] = \lambda\varphi.\lambda\rho.\ \left(\bigcup_{v \in Val} [\![h]\!]\varphi\rho[x = \natural v]\right) <_x [\![g]\!]\varphi\rho$$

Asymmetric combinator:

$$t_1 <_x t_2 = \begin{cases} t_1 \parallel t_2 & \text{no recv. for } x \text{ in } t_1 \text{ , no publ. in } t_2 \\ t_1 \parallel t_{21}\tau & \text{no recv. for } x \text{ in } t_1 \text{ , } t_2 \equiv t_{21}!v\, t_{22} \text{ , no publ. in } t_{21} \\ (t_{11} \parallel t_{21}\tau)(t_{12}\backslash[v/x]) & t_1 \equiv t_{11}[v/x]t_{12} \text{ , no recv. for } x \text{ in } t_{11} \text{ ,} \\ & t_2 \equiv t_{21}!v\, t_{22} \text{ , no publ. in } t_{21} \\ \{\varepsilon\} & \text{otherwise} \end{cases}$$

Asymmetric combinator for trace-sets:

$$T_1 <_x T_2 = \bigcup_{t_1 \in T_1, t_2 \in T_2} t_1 <_x t_2$$

Semantic Properties

- Continuity of the meaning functions
- Prefix-closure of the trace sets
- Adequacy:

$$t \in [\![f]\!][\![\Delta]\!]\rho \quad \textit{iff} \quad \exists f'.\, \Delta, \Gamma \vdash \sigma f \xrightarrow{t}{}^{*} f'$$

What to remember about Orc

- Abstracts computation away, focuses on communication
- Small but expressive
- Interesting theoretical properties

Related Work

- Kitchin, Cook and Misra. “A language for task orchestration and its semantic properties”, CONCUR (2006)
- van der Aalst et al. “Workflow Patterns”, Distributed and Parallel Databases 14(1): 5-51 (2003)
- Cook, Patwardhan and Misra. “Workflow patterns in Orc”, COORD (2006)
- Misra and Cook. “Computation Orchestration: a basis for wide-area computing”, Software and Systems Modeling, 6(1): 83-110 (2007)

A peek at what follows

- Proved useful congruences between Orc processes using strong bisimulation
- Created semantics insensitive to internal events, we can equate more processes

More info:

http://www.ccs.neu.edu/home/dimvar

PARADISE CITY A.C.
HOTEL
EWELER
THE END