Tracking causal dependencies in Web services orchestrations defined in Orc

Matthieu Perrin Claude Jard Achour Mostefaoui

LINA, University of Nantes

NETYS 2015
Agadir, Morocco
May, 14th 2015

Introduction

Context

- Web services orchestrations
- Distributed languages (Orc)
- Analysis of QoS or non functional properties

Problem

- Orc has an operational semantics
- How to track:
    - causality (root cause analysis)
    - concurrency (data race detection)

Approach

- Online tracking of additional information
- Instrumentation of the semantics

Plan

The Orc Programming Language

Orc standard semantics

The instrumented semantics

A language for Web site orchestration

Philosophy :

- web sites and services already exist
- provide operators for orchestration
- Orc calculus: model of concurrent programming

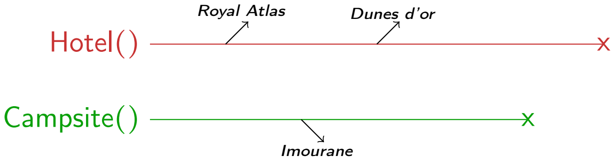
Orc sites

- look like functions
- publish 0, 1 or more values
- external
  - encapsulation of web services
  - standard library, constants and data types
  - control structures (conditional: ift, iff)
- internal
  - **def** Accomodation() = Hotel()|Campsite()

Operators

- $f|g$ (Parallel composition)
  - f and g are started in parallel
- $f\,;g$ (Otherwise operator)
- f >x> g (Sequential composition)
- f <x< g (Prunning)

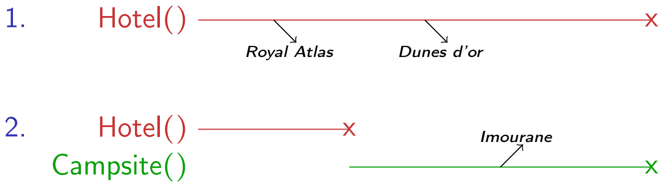
Example: Hotel()|Campsite()



Operators

- $f|g$ (Parallel composition)
- $f;g$ (Otherwise operator)
  - g is run if and only if f halts without publishing
- $f >x> g$ (Sequential composition)
- $f <x< g$ (Prunning)

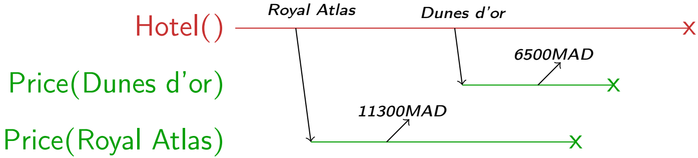
Example: Hotel(); Campsite()



Operators

- $f|g$ (Parallel composition)
- $f; g$ (Otherwise operator)
- $f >x> g$ (Sequential composition)
  - f is started alone first
  - a new instance of g started at each publication by f
- $f <x< g$ (Prunning)

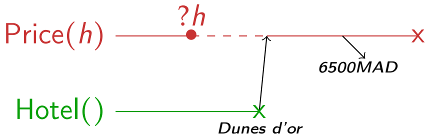
Example: Hotel() $>h>$ Price(h)



Operators

- $f|g$ (Parallel composition)
- $f;g$ (Otherwise operator)
- $f >x> g$ (Sequential composition)
- $f <x< g$ (Prunning)
  - f and g are started in parallel
  - f is paused when it needs to evaluate x
  - g is halted when it publishes a value
  - this value is bounded to x in f

Example: Price(h) $<h<$ Hotel()



Example: a travel agency

```
def find_best(agencies, destination) =

  def find_offers() =
    each(agencies) >agency> agency(destination) >offer>
    (offers.add((offer, agency)) |
    (best_offer.read() >o> compare(o, offer) >b>
    ift(b) >x> (best_agency.write(agency) | best_offer.write(offer))))        #

  def extend_best() =
    best_agency.read() >ba> best_offer.read() >bo> ba.exists(bo) >b>
    (ift(b) >x> ba.get_info(bo) | iff(b) >x> alarm("inconsistent"))           #

  def sort_offers() =
    offers.sort(); best_offer.read() = offers.first() >b>
    (ift(b) >x> offers | iff(b) >x> alarm("not best"))                        #

  ((t <t< (find_offers() | timer(2000))) >t>
  ((e_b, s_o) <e_b< extend_best() <s_o< sort_offers()))

  <offers< Stack()
  <best_offer< (Register() >r> r.write(null); r)
  <best_agency< Register()                                                    #
```

Example: a travel agency

```
def find_best(agencies, destination) =
  def find_offers() =
    each(agencies) >agency> agency(destination) >offer>
    (offers.add((offer, agency)) |
    (best_offer.read() >o> compare(o, offer) >b>
    ift(b) >x> (best_agency.write(agency) | best_offer.write(offer))))        #
  def extend_best() =
    best_agency.read() >ba> best_offer.read() >bo> ba.exists(bo) >b>
    (ift(b) >x> ba.get_info(bo) | iff(b) >x> alarm("inconsistent"))          #
  def sort_offers() =
    offers.sort(); best_offer.read() = offers.first() >b>
    (ift(b) >x> offers | iff(b) >x> alarm("not best"))                       #
  ((t <t< (find_offers() | timer(2000))) >t>
  ((e_b, s_o) <e_b< extend_best() <s_o< sort_offers()))
  <offers< Stack()
  <best_offer< (Register() >r> r.write(null); r)
  <best_agency< Register()                                                   #
```

Example: a travel agency

```
def find_best(agencies, destination) =

  def find_offers() =
    each(agencies) >agency> agency(destination) >offer>
    (offers.add((offer, agency)) |
    (best_offer.read() >o> compare(o, offer) >b>
    ift(b) >x> (best_agency.write(agency) | best_offer.write(offer))))      #

  def extend_best() =
    best_agency.read() >ba> best_offer.read() >bo> ba.exists(bo) >b>
    (ift(b) >x> ba.get_info(bo) | iff(b) >x> alarm("inconsistent"))         #

  def sort_offers() =
    offers.sort(); best_offer.read() = offers.first() >b>
    (ift(b) >x> offers | iff(b) >x> alarm("not best"))                      #

  ((t <t< (find_offers() | timer(2000))) >t>
  ((e_b, s_o) <e_b< extend_best() <s_o< sort_offers()))

  <offers< Stack()
  <best_offer< (Register() >r> r.write(null); r)
  <best_agency< Register()                                                  #
```

Plan

The Orc Programming Language

Orc standard semantics

The instrumented semantics

Structural operational semantics

Rules of inference

$$\frac{\overbrace{f_1 \xrightarrow{l_1} f_1'}^{\text{premise } 1} \quad \dots \quad \overbrace{f_n \xrightarrow{l_n} f_n'}^{\text{premise } n}}{\underbrace{F(f_1, \dots, f_n) \xrightarrow{l} F'(f_1', \dots, f_n')}_{\text{conclusion}}}$$

- If the *premises* are possible, then the *conclusion* is possible
- *Axioms* are rules with no premise
- Define a transition system

Semantics of f

- $l_1 \dots l_n \in [\![f]\!]$ if there are $f_1, \dots, f_n$ such that $f \xrightarrow{l_1} f_1 \dots \xrightarrow{l_n} f_n$

The semantics of Orc

Semantics of the prunning operator

$$\text{(PruneLeft)}\ \frac{f \xrightarrow{l} f'}{f \text{ <x< } g \xrightarrow{l} f' \text{ <x< } g}\ l \neq \omega$$

$$\text{(PruneN)}\ \frac{g \xrightarrow{n} g'}{f \text{ <x< } g \xrightarrow{n} f \text{ <x< } g'}\ n \notin \{!v, \omega\}$$

$$\text{(PruneV)}\ \frac{g \xrightarrow{!v} g'}{f \text{ <x< } g \xrightarrow{h(!v)} [v/x]f}$$

$$\text{(PruneStop)}\ \frac{g \xrightarrow{\omega} \bot}{f \text{ <x< } g \xrightarrow{h(\omega)} [\mathbf{stop}/x]f}$$

Example: a travel agency

1. each([A1, A2])
2. timer(2000)
3. new_register()
4. new_register()
5. A1(D)
6. r.write(null)
7. best_offer.read()
8. new_stack()
9. offers.add(O1)
10. A2(D)
11. offers.add(O2)
12. compare(null, 01)
13. best_offer.read()
14. compare(null, 02)
15. ift(true)
16. ift(true)
17. best_offer.write(O2)
18. best_offer.write(O1)
19. best_agency.write(A1)
20. best_agency.write(A2)
21. best_agency.read()
22. best_offer.read()
23. A2.exists(O1)
24. iff(false)
25. ift(false)
26. alarm("inconsistent")
27. offers.sort()
28. best_offer.read()
29. offers.first()
30. =(O1, O2)
31. iff(false)
32. ift(false)
33. alarm("not best")

Plan

The Orc Programming Language

Orc standard semantics

The instrumented semantics

Labelled Asymmetric Event Structures (LAES)

Definition

$(E, L, \leq, \nearrow, \Lambda)$

- E: set of *events*
- L: set of *labels*
- $\leq \ \in E^2$: *causality* (partial order)
- $\nearrow \ \in E^2$: *weak causality*
- $\Lambda : E \mapsto L$: *labelling function*

With

- $[e] = \{e' \in E | e' \leq e\}$ finite
- $e < e' \Rightarrow e \nearrow e'$
- $e \in E, \nearrow \cap [e]^2$ acyclic

Concepts

- causality $(e \leq e')$
  e always before e'
- weak causality $(e \nearrow e')$
  e never after e'
- preemption $(e' \rightsquigarrow e)$
  $e \nearrow e' \wedge \neg(e \leq e')$
- concurrency $(e || e')$
  $\neg(e \nearrow e' \vee e' \nearrow e)$
- conflict $(\#\{e_1, ..., e_n\})$
  $e_1 \nearrow ... \nearrow e_n \nearrow e_1$

Instrumented executions

Labels on transitions

$\sigma = \sigma_0 \ldots \sigma_n \in [\![f]\!]_i$
$\sigma_i = (k_i, l_i, c_i, a_i)$

- k_i: unique identifier
- l_i : label
- c_i : causes
- a_i : weak causes

LAES

$\overline{\overline{\sigma}} = (E, L, \leq, \nearrow, \Lambda)$

$E = \{k_0, \ldots, k_n\}$

$L = \{l_0, \ldots, l_n\} \quad \Lambda(k_i) = l_i$

$k_i \leq k_j \Leftrightarrow k_i \in c_j$

$k_i \nearrow k_j \Leftrightarrow k_i \in a_j$

The causal operator

$\langle f, c, a \rangle_L$

- f: a program
- c: its causes
- a: its weak causes
- L: a type of labels ($!v$, ω, l)

$$(\text{CauseYes})\ \frac{f \xrightarrow{k,l,c,a}_i f'}{\langle f, c', a' \rangle_L \xrightarrow{k,l,c \cup c', a \cup a' \cup c'}_i \langle f', c', a' \rangle_L}\ l \in L$$

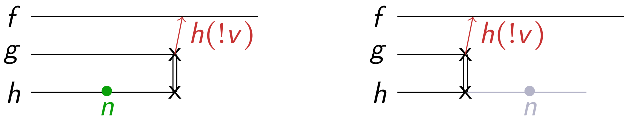
$$(\text{CauseNo})\ \frac{f \xrightarrow{k,l,c,a}_i f'}{\langle f, c', a' \rangle_L \xrightarrow{k,l,c,a}_i \langle f', c', a' \rangle_L}\ l \notin L$$

Instrumentation of rule PruneN

Standard semantics

$$(\text{PruneN})\ \frac{g \xrightarrow{n} g'}{f \mathbin{<x<} g \xrightarrow{n} f \mathbin{<x<} g'}\ n \notin \{!v, \omega\}$$

Preemption: $f \mathbin{<x<} (g|h)$



Instrumented semantics

$$(\text{PruneN})\ \frac{g \xrightarrow{k,n,c,a}_i g'}{f \mathbin{<x<} g \xrightarrow{k,n,c,a}_i f \mathbin{<x<} \langle g', \emptyset, \{k\}\rangle_{!v}}\ n \notin \{!v, \omega\}$$

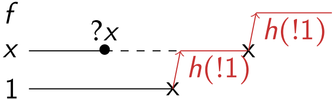
Instrumentation of rule PruneV

Standard semantics

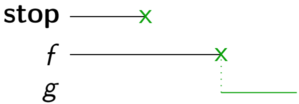
$$(\text{PruneV})\,\frac{g \xrightarrow{!v} g'}{f \,{<}x{<}\, g \xrightarrow{h(!v)} [v/x]f}$$

Two vectors of causality

$(x \,{<}x{<}\, 1) \,{>}y{>}\, f$



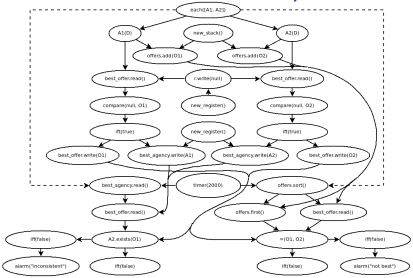
$(\textbf{stop} \,{<}x{<}\, f\,); g$



Instrumented semantics

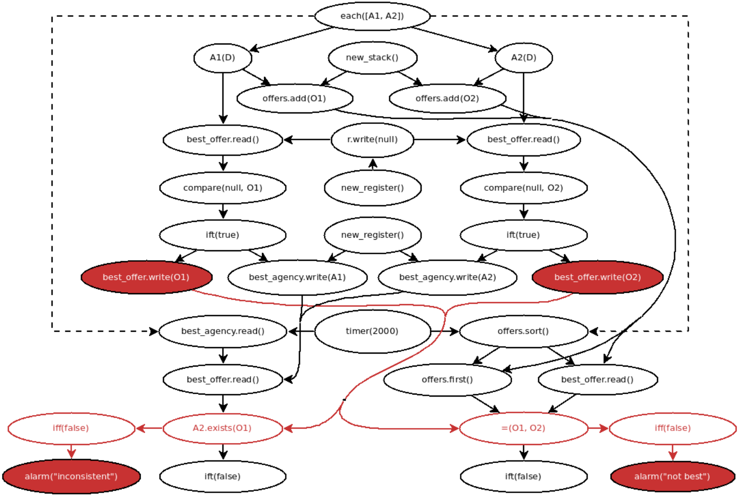
$$(\text{PruneV})\,\frac{g \xrightarrow{k,!v,c,a}_i g'}{f \,{<}x{<}\, g \xrightarrow{k,h(!v),c,a}_i \langle [\langle v, c \cup \{k\}, a\rangle_l / x] f, c \cup \{k\}, a\rangle_\omega}$$

Example: a travel agency (instrumented execution)



(1, each([A1, A2]), ∅, ∅)
(2, timer(2000), ∅, ∅)
(3, new_register(), ∅, ∅)
(4, new_register(), ∅, ∅)
(5, A1(D), {1}, {1})
(6, r.write(null), {4}, {4})
(7, best_offer.read(), {1,4-6}, {1,4-6})
(8, new_stack(), ∅, ∅)
(9, offers.add(O1), {1,5,8}, {1,5,8})
(10, A2(D), {1}, {1})
(11, offers.add(O2), {1,8,10}, {1,8,10})
(12, compare(null, 01), {1,4-7}, {1,4-7})
(13, best_offer.read(), {1,4,6,10}, {1,4,6,10})
(14, compare(null, 02), {1,4,6,10,13}, {1,4,6,10,13})
(15, ift(true), {1,4-7,12}, {1,4-7,12})
(16, ift(true), {1,4,6,10,13,14}, {1,4,6,10,13,14})
(17, best_offer.write(O2), {1,4,6,10,13,14,16}, {1,4,6,10,13,14,16})
(18, best_offer.write(O1), {1,4-7,12,15}, {1,4-7,12,15})
(19, best_agency.write(A1), {1,3-7,12,15}, {1,3-7,12,15})
(20, best_agency.write(A2), {1,3,4,6,10,13,14,16}, {1,3,4,6,10,13,14,16})
(21, best_agency.read(), {2}, {2,1})
(22, best_offer.read(), {1-7,10,12-16,19-21}, {1-7,10,12-16,19-21})
(23, A2.exists(O1), {1-7,10,12-19,22}, {1-7,10,12-19,22})
(24, iff(false), {1-7,10,12-19,22,23}, {1-7,10,12-19,22,23})
(25, ift(false), {1-7,10,12-19,22,23}, {1-7,10,12-19,22,23})
(26, alarm("inconsistent"), {1-7,10,12-19,22-24}, {1-7,10,12-19,22-24})
(27, offers.sort(), {2}, {2,1})
(28, best_offer.read(), {1,2,5,9-11,27}, {1,2,5,9-11,27})
(29, offers.first(), {1,2,5,9-11,27}, {1,2,5,9-11,27})
(30, =(O1, O2), {1,2,4-7,9-18,27-29}, {1,2,4-7,9-18,27-29})
(31, iff(false), {1,2,4-7,9-18,27-30}, {1,2,4-7,9-18,27-30})
(32, ift(false), {1,2,4-7,9-18,27-30}, {1,2,4-7,9-18,27-30})
(33, alarm("not best"), {1,2,4-7,9-18,27-31}, {1,2,4-7,9-18,27-31})

Example: a travel agency (LAES)



⟶ is a cause of

- - - -→ is preempted by

Properties

Instrumentation

We only add information on the existing executions:

$$\forall f, (\llbracket f \rrbracket_i)|_I = \{\sigma_1.I...\sigma_n.I | \sigma \in \llbracket f \rrbracket_i\} = \llbracket f \rrbracket.$$

Correctness

Only correct behaviors can be infered from an execution:

$$\forall f, \forall \sigma \in \llbracket f \rrbracket_i, \mathrm{Lin}(\overline{\overline{\sigma}}) \subset \llbracket f \rrbracket.$$

Linearization $\Lambda(e_0)...\Lambda(e_n) \in \mathrm{Lin}(\overline{\overline{\sigma}})$:

- left closed for causality
- respects weak-causality

Conclusion

Problem

- Tracking causality and concurrency in Orc orchestrations

Contribution

- Instrumentation of the standard semantics

Future work

- Extend the approach to other languages (BPEL)
- Track conflicts