Towards Verification of a Service Orchestration Language

Tan Tian Huat
Outline

- Background of Orc
- Motivation of Verifying Orc
- Overview of Orc Language
- Verification using PAT
- Future Works
Outline

- Background of Orc
- Motivation of Verifying Orc
- Overview of Orc Language
- Verification using PAT
- Future Works
Background of Orc

- Proposed by Jayadev Misra at University of Texas at Austin (UT Austin) in 2004.
- Orc is a service orchestration language, which can be used as:
  - Executable specification language
    - Web Service Orchestration
    - Workflow process [1]
  - General purpose programming language
Background of Orc

- Process calculus $\rightarrow$ Full programming language.
- Involve timing aspect.
- Has the structure and feel of a functional programming language, yet it handles many non-functional aspects effectively, including time-outs etc.
- A simulator is created in Java by UT Austin team.
Outline

- Background of Orc
- Motivation of Verifying Orc
- Overview of Orc Language
- Verification using PAT
- Future Works
Motivation of Verifying Orc

- The only related work is [5] which is done by our group
  - Translate Orc to Time-Automata.
  - Verify it using UPPAAL.

- Downside
  - The translation from Orc to Time-Automata takes time.
  - The translated model might be complicated.
  - Furthermore, Orc has evolved over time.

- Our new approach
  - Direct Verification of Orc.
Outline

- Background of Orc
- Motivation of Verifying Orc
- Overview of Orc Language
- Verification using PAT
- Future Works
Overview of Orc Language

- Site – Basic service or component

Category of Sites

- Internal Site:
  - +, -, *, &&, ||, <=
  - 1+1 → (+)(1,1)
  - Rtimer
    - Rtimer(5000)
- External Site: Google Search, MySpace, CNN, ...
  - Google ("Orc")
Overview of Orc Language

- Site call - two steps:
  - Invocation
  - Response with published value, or halt

- Published value can be:
  - Constant – string, boolean, number, list, and so on
  - Signal – A special value which carries no information

- Effects of calling sites:
  - Response
  - Halted
  - Pending – Neither response nor halted
Structure of Orc Expression

- Simple: just a site call, eg. $CNN(d)$
  - Publishes the value returned by the site.

- Composition of two Orc expressions:
  - $f$ and $g$ can be simple expression like $CNN(d)$, or composite expression like $CNN(d) \mid BBC(d)$, $x$ is a variable to be bounded.
  - $f \mid g$ Parallel Combinator
  - $f > x > g$ Sequential Combinator
  - $f < x < g$ Pruning Combinator
  - $f ; g$ Otherwise Combinator

- Orc is about the theory of combinators.
Parallel Combinator: $f \mid g$

- Evaluate $f$ and $g$ independently.
- Publish all values from both.
- No direct communication or interaction between $f$ and $g$.

Example: $CNN(d) \mid BBC(d)$

Calls both $CNN$ and $BBC$ simultaneously. Publishes values returned by both sites. (0, 1 or 2 values)
Sequential Combinator: \( f >\!\!x\!\!> g \)

For all values published by \( f \) do \( g \).
Publish only the values from \( g \).

- \( CNN(d) >\!\!x\!\!> Email(address, x) \)
  - Call \( CNN(d) \).
  - Bind result (if any) to \( x \).
  - Call \( Email(address, x) \).
  - Publish the value, if any, returned by \( Email \).

- \( (CNN(d) \mid BBC(d)) >\!\!x\!\!> Email(address, x) \)
  - May call \( Email \) twice.
  - Publishes up to two values from \( Email \).

Notation: \( f >> g \) for \( f >\!\!x\!\!> g \), if \( x \) is unused in \( g \).
Pruning Combinator: \((f < x < g)\)

For some values published by \(g\) do \(f\).

- Evaluate \(f\) and \(g\) in parallel.
  - Site calls in \(f\) that need \(x\) are suspended.
  - See \((M()) \mid N(x)) < x < g\)
- When \(g\) returns a (first) value:
  - Bind the value to \(x\).
  - Terminate \(g\).
  - Resume suspended calls in \(f\).
- Values published by \((f < x < g)\) are the values returned by \(f\).
- Example:
  
  \(Email(address, x) < x < (CNN(d) \mid BBC(d))\)
Otherwise Combinator \((f ; g)\)

- First executes \(f\)
  - If \(f\) stops and publishes any value, \(g\) is ignored. If \(f\) stops and publishes no value, then \(g\) executes. \(f\) stops if:
    - All site calls in the execution of \(f\) have either responded or halted.
    - \(f\) will never call any more sites.
    - \(f\) will never publish any more values.
  - Example:
    \((CNN(d) ; BBC(d)) > x > Email(a,x)\)
Functional subset of Orc

- Constants – true, false, 1, 2, 3, "abc"
  - 1 → let(1)

- Conditional – if true then 4 else 5
  - (if(b) >> let(4) | if(~b) >> let(5)) <b< let(true)

- Local variables – val a=1 a
  - let(a)<a<let(1)

- Functions – def A(x, y)=x+y
Example

include "search.inc"
def sum (x, y)= x + y
val a=1
if sum(a,1)=2
then
    Google("Orc Language")
else
    "impossible!"
Programming Idioms

- Fork-Join
- Parallel Or
- Timeout
- Priorities
- And so on…
Programming Idioms

- Fork-Join
- Parallel Or
- Timeout
- Priorities
- Non-deterministic choice
- And so on…
Timeout

result < result < (  
  Google("impatience") | (Rtimer(5000) >>="Search timed out.") )

- Search for the keyword “impatience" in Google.
- If the result is not returned within 5 seconds, "Search timed out." is published.
Operational Semantics of Orc [2]
Outline

- Background of Orc
- Motivation of Verifying Orc
- Overview of Orc Language
- Verification using PAT
- Future Works
PAT Architecture
Verification using PAT

- Support all combinators
- Local Site
  - Fundamental – Arithmetic and logic operator, If site.
  - Time – Rtimer
  - Other – Ref site, SynChannel site, and so on …
- Remote Site
- Approach
  - Parse the language into the model.
  - Applying abstraction (such as Process Counter Abstraction) on the model.
  - Generate the labeled transition system with operational semantics
  - After that, the pool of verification algorithms in PAT will be available for usage.
Challenges of Verifying Orc

- State explosion problem
  - Many normal operations such as declaration of variable, or application of function are designed to run in parallel.
    - val a=1+1
    - 1+1+a
    - def f(a,b)=1+1+a+b
    - f(1+1, 1+1)

- Solution:
  - Partial Order Reduction
Outline

- Background of Orc
- Motivation of Verifying Orc
- Overview of Orc Language
- Verification using PAT
- Future Works
Future Works

- Support verification of more libraries such as channel, semaphore, etc of Orc
- Refined the current state reduction approach
- Explore on more state reduction techniques
References


Thanks!