Orc Verification

Jayadev Misra

Department of Computer Science
University of Texas at Austin

http://orc.csres.utexas.edu

WG 2.3, St. Petersburg
June 5, 2013
Orc Verification has been a disaster

- Concurrency everywhere

- Non-determinacy
  As powerful as any other process calculus

- Real time
  Not just causal ordering among events but temporal ordering

- Basic orc has no mutable variables, but sites do

- Full functional programming (w/o monads) plus (active) Objects.
Subset Sum

Given integer  \( n \) and list of integers  \( xs \).

\( \text{parsum}(n, xs) \) publishes all sublists of  \( xs \) that sum to  \( n \).

parsum(5, [1, 2, 1, 2]) = [1, 2, 2], [2, 1, 2]
parsum(5, [1, 2, 1]) is silent

\[
\text{def } \text{parsum}(0, []) = []
\]

\[
\text{def } \text{parsum}(n, []) = \text{stop}
\]

\[
\text{def } \text{parsum}(n, x : xs) = \\
\quad \text{parsum}(n - x, xs) > ys > x : ys \\
\mid \text{parsum}(n, xs)
\]
Subset Sum (Contd.), Backtracking

Given integer \( n \) and list of integers \( xs \).

\( \text{seqsum}(n, xs) \) publishes the first sublist of \( xs \) that sums to \( n \).

“First” is smallest by index lexicographically.

\[
\text{seqsum}(5, [1, 2, 1, 2]) = [1, 2, 2]
\]

\( \text{seqsum}(5, [1, 2, 1]) \) is silent

\[
\text{def seqsum}(0, []) = []
\]

\[
\text{def seqsum}(n, []) = \text{stop}
\]

\[
\text{def seqsum}(n, x : xs) = \\
x : \text{seqsum}(n - x, xs) \\
; \text{seqsum}(n, xs)
\]
Subset Sum (Contd.), Concurrent Backtracking

Publish the first sublist of $xs$ that sums to $n$.

Run the searches concurrently.

```python
def parseqsum(0, []) = []
def parseqsum(n, []) = stop
def parseqsum(n, x : xs) =
  (p ; q)
  <p< x : parseqsum(n - x, xs)
  <q< parseqsum(n, xs)
```

Note: Neither search in the last clause may succeed.
Semantics

- Tree semantics with Hoare

- Operational semantics with Cook
  1. Traces
  2. Bisimulation can be applied to prove some identities.
  3. Denotational Semantics was difficult. But, it established that:

    Orc combinators are monotonic and continuous.

- But, operational semantics seems ineffective for program proving.

- I failed in applying axiomatic semantics.
A sequence of Verification Problems

- Basic Orc without mutable variables, real time
- add real time
- add mutable variables
- Full Orc language
Denotational semantics with composable proof theories

\[ [f \mid g] \triangleq [f] \mid [g] \]
\[ [f \gg x \gg g] \triangleq [f] \gg x \gg [g], \quad [f \gg g] \triangleq [f] \gg [g] \]
\[ [f \ll x \ll g] \triangleq [f] \ll x \ll [g], \quad [f \ll g] \triangleq [f] \ll [g] \]
\[ [f ; g] \triangleq [f] ; [g] \]
Simple Expressions

- 1 publishes just 1: \{1\}
- 1 | 2 publishes 1 and 2 in either order: \{1, 2\}
- 1 | 1 publishes 1 and 1: [1, 1]

Publications are unordered.
Simple Expressions

- $1$ publishes just $1$: $\{1\}$

- $1 | 2$ publishes $1$ and $2$ in either order: $\{1, 2\}$

- $1 | 1$ publishes $1$ and $1$: $[1, 1]$

Publications are unordered.
Simple Expressions

- 1 publishes just 1: \{1\}

- 1 | 2 publishes 1 and 2 in either order: \{1, 2\}

- 1 | 1 publishes 1 and 1: [1, 1]

Publications are unordered.
A possible denotation of expressions

- Represent an expression by a bag of values.

- $\mid$ combines two bags.

- Bags may be infinite.

\[
\text{def } \text{nat}(i) = i \mid \text{nat}(i + 1)
\]

\[
\text{def } \text{nats}() = \text{nat}(0)
\]

\[
[\text{nats}()] = [0, 1, \cdots]
\]

- Computation may be infinite without any publication.

\[
\text{def } \text{unend}() = \text{signal } \gg \text{unend}()
\]

\[
[\text{unend}()] = []
\]
Bags are not enough

\[ \text{stop} \] = []

\[ \text{unend}() \] = []

But their behaviors are different:

\[ \text{stop} ; 3 \neq \text{unend}() ; 3 \]
Halting, Waiting

Associate a status, \( H \) for halting, \( W \) for waiting, to each bag.

\[
\begin{align*}
\text{[stop]} & = H[\ ] \\
\text{[unend]} & = W[\ ] \\
[1 \mid \text{unend}] & = W[1] \\
\text{[nats]} & = W[0, 1, \cdots]
\end{align*}
\]

Elementary term: A status and a bag. The status of an infinite bag is always \( W \).
Combining Elementary Terms with

\[ s[m] | s'[m'] = (s \cap s')[m \sqcup m'] \]

where \( H \cap s = s, \ W \cap s = W \)

\[
\begin{align*}
[1 \mid \text{true}] &= H[1] \mid H[\text{true}] = H[1, \text{true}] \\
[1 \mid \text{stop}] &= H[1] \mid H[] = H[1] \\
[1 \mid \text{unend}()] &= H[1] \mid W[] = W[1] \\
[nats()] \mid nats() &\neq [nats()]
\end{align*}
\]
A specification, \textit{spec}, is a set of terms, possibly infinitely many.

\[
\llbracket Random(3) \rrbracket = \{H[0], H[1], H[2]\}
\]

\[
\llbracket Random(3) \mid true \mid false \rrbracket = \{H[0, true, false], H[1, true, false], H[2, true, false]\}
\]

\[
\llbracket anynat() \rrbracket = \{H[i] \mid \text{natural } i\}
\]
Combining specs using \(\left\{ s_0, \ldots, s_i, \ldots \right\} \mid \left\{ t_0, \ldots, t_i, \ldots \right\} = \left\{ (s_0 \mid t_0), \ldots, (s_i \mid t_j), \ldots \right\} \) distributes over each argument set. Take Cartesian product.
Guarded Term

- $b \rightarrow s[m]$: the set of traces in which the bindings satisfy $b$ and the status and publications satisfy $s[m]$.

- Taking $|$ over guarded terms:
  
  \[
  b \rightarrow s[m] \mid b' \rightarrow s'[m'] = (b \land b') \rightarrow (s \land s')[m \sqcup m']
  \]

- Guards distribute over terms in a spec:
  
  \[
  b \rightarrow \{t_0, t_1 \cdots \} = \{b \rightarrow t_0, b \rightarrow t_1 \cdots \}
  \]
Parameters; Guarded terms

- $$[\text{not}(x)] = \{x = \text{true} \rightarrow H[\text{true}], \ x = \text{false} \rightarrow H[\text{false}]\}$$

- $$[x] = \{x = c \rightarrow H[c] \mid \text{for all } c\}$$

- $$[\text{Ift}(x)] = \{x = \text{true} \rightarrow H[\text{signal}], \ x \neq \text{true} \rightarrow H[\text{false}]\}$$

Often a parameter is known to remain unbound, denoted by $$\not\!x$$:

$$[x] = \{x = \not\!x \rightarrow H[\text{false}]\} \cup \{x = c \rightarrow H[c] \mid \text{for all } c\}$$
Example

\[
\[Ift(x)\] = \{ x = \text{true} \rightarrow H[\text{signal}], \ x \neq \text{true} \rightarrow H[] \}\n\]
\[
\[Iff(x)\] = \{ x = \text{false} \rightarrow H[\text{signal}], \ x \neq \text{false} \rightarrow H[] \}\n\]

\[
\[Ift(x) \mid Iff(x)\]
= \{ (x = \text{true} \land x = \text{false} \rightarrow \cdots) \\
, (x = \text{true} \land x \neq \text{false} \rightarrow H[\text{signal}]) \\
, (x \neq \text{true} \land x = \text{false} \rightarrow H[\text{signal}]) \\
, (x \neq \text{true} \land x \neq \text{false} \rightarrow H[]) \}\n\]

= \{ (x = \text{true} \lor x = \text{false} \rightarrow H[\text{signal}]) \\
, (x \neq \text{true} \land x \neq \text{false} \rightarrow H[]) \\}
Notation

Convention:

\[ s[\cdots f(x, y) \cdots] \triangleq \{ x = c \land y = c' \rightarrow s[\cdots f(c, c') \cdots] \mid \forall c, c' \}, \]

for any total function \( f \) that is strict in all its arguments.

- \( \llbracket \text{choose}(x, y) \rrbracket = \{ H[x], H[y] \} \)
- \( \llbracket \text{parallel}_\lor(x, y) \rrbracket \)
  \[ = \{(x = \text{true} \rightarrow H[\text{true}]),
     (y = \text{true} \rightarrow H[\text{true}]),
     H[x \lor y]\} \]
- \( (x = \text{true} \rightarrow H[\text{true}]), (y = \text{true} \rightarrow H[\text{true}]) \)
  is not the same as
  \( (x = \text{true} \lor y = \text{true} \rightarrow H[\text{true}]) \)

The first line is satisfied even if just one of \( x \) and \( y \) is bound.
(1 \mid 2) \gg (10 \mid 20) : \text{ Execute the rhs for every publication of lhs}

Should have the spec $H[10, 20, 10, 20]$, constructed from $H[1, 2]$ and $H[10, 20]$. 
Sequential Composition; contd.

In \( s[m] \gg q \), for every value in \( m \) one instance of a program with spec \( q \) is executed. All such programs are executed in parallel.

**Tentative Rule:**
\[
\begin{align*}
 s[m] \gg q &= \Vert q \mid c \in m \\
 H[1, 2] \gg H[10, 20] &= \{\text{Tentative Rule}\} \\
\end{align*}
\]

For \( W[1, 2] \gg H[10, 20] \), the result should be \( W[10, 20, 10, 20] \)

**Exact Rule:**
\[
\begin{align*}
 s[m] \gg q &= s[] \mid [q \mid c \in m] \\
p \gg q &= \bigcup_{t \in p} (t \gg q)
\end{align*}
\]
Sequential Composition with value passing

\[ [(1 \mid 2) \triangleright x \triangleright (10 + x \mid 20 - x)] = H[11, 12, 18, 19] \]

Rule:

\[ s[m] \triangleright x \triangleright q = s[\cdot] \mid [(x \mapsto c)q \mid c \in m] \]
\[ p \triangleright x \triangleright q = \bigcup_{t \in p}(t \triangleright x \triangleright q) \]

\[ H[1, 2] \triangleright x \triangleright H[10 + x, 20 - x] \]
\[ = H[\cdot] \mid (x \mapsto 1)H[10 + x, 20 - x] \mid (x \mapsto 2)H[10 + x, 20 - x] \]
\[ = H[\cdot] \mid H[10 + 1, 20 - 1] \mid H[10 + 2, 20 - 2] \]
\[ = H[11, 19, 12, 18] \]

Exercise: \[ [nats() \triangleright x \triangleright x \times x] \]
Pruning

\[(x < x < (1 \mid 2)) = \{H[1], H[2]\}\]

Rule:

\[p < x < s[m] = \bigcup_{(c \in m)} ((x \mapsto c)p)\]

\[p < x < q = \bigcup_{(t \in q)} (p < x < t)\]

\[\llbracket i < i < \text{nats()} \rrbracket = \]

\[H[i] < i < W[0, 1, \cdots] = \]

\[\bigcup_{(c \in [0,1,\cdots])} ((i \mapsto c)H[i]) = \]

\[\{H[0], H[1], \cdots\}\]
Recursive Definition

\[
\text{def } \text{nat}(i) = i \mid \text{nat}(i + 1)
\]

\[
\text{def } \text{nats}() = \text{nat}(0)
\]
Ordering over terms

- $t \leq t$

- $(b \rightarrow W[m]) \leq (b' \rightarrow s[m'])$ if $b' \Rightarrow b$, $m \sqsubseteq m'$

$W[\cdot]$ is the smallest term.
Prefix closure; Spec Ordering

Define:

- \( t^* = \{ s \mid s \leq t \} \)
- \( p^* = \bigcup_{t \in p} (t^*) \)
- \( p \leq q \triangleq p^* \subseteq q^* \)
- \( p \equiv q \triangleq (p \leq q) \land (q \leq p) \)

So, \((p \equiv q) = (p^* = q^*)\)
Monotonicity, Continuity

- Every combinator is monotonic in each argument.

- Every combinator is continuous in each argument. Take the lub of a chain of specs to be the union of their closures.
Extensions

- Real time needs a surprisingly simple extension.
- Yet to be done: Mutable sites, Orc language