Session-Typed Concurrent Programming

Stephanie Balzer
Carnegie Mellon University

July 20, 2018

1 Curry-Howard Correspondence

Linear Propositions vs Session Types

- $A \& B$ vs. External Choice (Polarity (-))
- $A \oplus B$ vs. Internal Choice (+)
- $A \multimap B$ vs. Channel Input (-)
- $A \otimes B$ vs. Channel Output (+)
- 1 vs. Termination (+)

1.1 Cut

Parallel composition / spawning a process

$$\frac{\Delta \vdash P :: (x : A) \quad \Delta', x : A \vdash Q :: (z : C)}{\Delta, \Delta' \vdash x \leftarrow P; Q :: (z : C)}\,(\text{Cut})$$

1.2 Identity

$$\frac{}{y : A \vdash \texttt{fwd}\ x\ y :: (x : A)}\,(\text{ID})$$

2 Multiset Rewriting Rules

$S \to T$
$S = \texttt{proc}(c_1, P_{c_1}), \ldots, \texttt{proc}(c_n, P_{c_n})$

```
(1):   proc(c, wait a, Q), proc(a, close a) → proc(c, Q)
(&):   proc(c, a.L(h), Q), proc(a, case a of seq(L => P))
     → proc(c, Q), proc(a, P(h))
(cut):   proc(a, x ← P, Q(x)) → proc(a, [b/x]Q), proc(b, P)
     with b fresh
(fwd):   proc(a, fwd a b) → a = b
```

3 Computer Example

– Using the language C0 –

Notation:

```
c: & seq(L:A)
! for output/send
? for input/receive
c: A ⊗ B is written as $c: <!A, B>
c: 1 is written as $c: <>
$c: ?choice{<A>L}
```

```
#use <conio> gives I/O
typedef <?choice queue> queue;
typedef <!choice queue_elem> queue_elem;
choice queue {
  <?int; queue> enq;
  <queue_elem> deq;
}
choice queue_elem {
  <> none;
  <!int; queue> some;
}
queue $q empty () {
  switch ($q) {
    case enq:
      int y = recv($q);

    case deq:

  }
}
```

4 Progress and Preservation

```
Ω ::= · | proc(a, P(a)), Ω'
```

4.1 Preservation

If $\models \Omega : \Delta$ and $\Omega \to \Omega'$, then $\models \Omega' : \Delta$

4.2 Progress

By using a tree as the representation for the channels we can never have a loop, meaning there will always be a process ready to execute.