Session-Typed Concurrent Programming

Stephanie Balzer
Carnegie Mellon University

July 18, 2018

1 Introduction

1.1 Road Map

- Message-Passing Concurrent Programming
- Session Types
- Connnection between Linear Logic and Session Types
- Manifest Sharing

1.2 Learning Objectives

- How to do message-passing concurrenty
- What session types are about
- Benefits of linear logic to programming
- How to accommodate sharing in a logically motivated way

2 Message-Passing Concurrency

Processes that compute by exchanging messages along channels.
Consider processes $P_1, \ldots, P_5$ where
$P_1 \longleftrightarrow_a P_3$
$P_1 \longleftrightarrow_a P_4$
$P_3 \longleftrightarrow_a P_4$
$P_2 \longleftrightarrow_b P_3$
$P_2 \longleftrightarrow_c P_5$
$P_4 \longleftrightarrow_d P_5$
This means there are n-arry channels (i.e.: channel a with P_1, P_3, P_4
These channels give us non-determinism
Formal under paring: process calculus (π-calculus (universal))

2.1 Example 1

2.1.1 Message-Passing Queue

Have a client and a queue: `client` $\longleftrightarrow_q$ `queue`
Can represent the queue as nodes designated by characters:
`'O'` $\longleftrightarrow_{n_1}$ `'P'`
`'P'` $\longleftrightarrow_{n_2}$ `'L'`
`'L'` $\longleftrightarrow_{n_3}$ `empty`
Can dequeue from `'O'` and enqueue at `empty`
$\downarrow$ `enq 'S'` updates `'L'` $\longleftrightarrow_{n_3}$ `'S'` and creates `'S'` $\longleftrightarrow_{n_4}$ `empty`
$\downarrow$ `deq` removes `'O'` $\longleftrightarrow_{n_1}$ `'P'` and updates the client's channel

2.1.2 Passing Channels

Can add a channel to a process as an offshoot of the queue.
This is what is referred to as mobility in π-calculus.
A.k.a: Higher order channels.

3 Session Types

Types for Protocols of Message Exchange

```
A := ?[T].A'  |  ![T].A'
     |  &{l_1.A_1 ,..., l_n.A_n}  "external choice"
     |  ⊕{l_1.A_1 ,..., l_n.A_n}  "internal choice"
     |  end  |  X  |  μX.A'

T := A  |  Int  |  Char  |  ...

queue := &{enq:  ?[Char].queue,
           deq:  ⊕{none:  end,  some:  ![Char].queue}}
```

3.1 Example 1 revisited

Now the channel and the queue are both of type queue:

```
q: queue
q: &{enq:... ,  deq:...}
*Send 'enq' along q
q: ?[Char].queue
*Send 'S' along q
q: queue
```

Observation: type of channel/process changes with message exchange.

This can be problematic for preservation:
What if two clients use the same queue?

```
q: queue
*Client 1: Send 'enq' along q
q: ?[Char].queue
*Client 2: Send 'deq' along q
***Oops!
```

3.2 Preservation in Session Types

AKA: "Session Fidelity"

Guarantees that expectation of client matches the one provided assuming they did initially.
Can use linearity and resource management to achieve this.
Treat channels as resources and base session types on linear logic.