Concurrency with Pi

Lecture 27

CS 565

(slides adapted from Martin Abadi)

Concurrency

Concurrency comes in several flavors:

- Distributed processes
  - Code running at multiple sites
- Threads
  - A language mechanism for specifying interleaving computations; often run on a single processor
- Parallel (SIMD)
  - A single program but with simultaneous operations on multiple data

Different notions) different research communities

- Internet agents vs. high-performance physics calculations

Ways to Describe Concurrency

Sequential processes are modeled by λ-calculus

- inevitable: the most natural way to observe an algorithm is to examine its output for various inputs) functions

A concurrent system is naturally non-deterministic

- Interleaving of atomic actions from different processes

Concurrent processes can be observed in many ways

- When are two concurrent systems equivalent ?
- Intra-process behavior vs. inter-process behavior

Concurrency can be described in many ways

- Process creation: fork/wait, cobegin/coend, data parallelism
- Process communication: shared memory, message passing
- Process synchronization: monitors, semaphores, transactions

This lead to a variety of process calculi

Communication Primitives

As usual we focus on foundations.

- Focus on communication as fundamental concept.
- Not a foundation for everything.
  - Not much to say about parallel computing here.

Communication through shared variables.

- Found mostly in parallel computing and threads.

Communication through messages.

- synchronous or asynchronous
- static or dynamic communication topology
- first-order or high-order data
- local or distributed

Communication in Languages

Historically the treatment of communication is weak.

- I/O often not considered part of the language.
- Added only as an afterthought (Pascal)

Even “modern” languages have primitive language I/O facilities.

- Messages are rare.
- Higher-level remote procedure/method call is rare.

Focus on communication by message passing.

Calculi and Languages

Several calculi and languages rely on message-passing:

- Communicating Sequential Processes (CSP) (Hoare, 1978)
- Occam (Jones)
- Calculus of Communicating Systems (CCS) (Milner, 1980)
- The Pi calculus (Milner, 1989 and others)
- Pict (Pierce and Turner)
- Concurrent ML (Reppy)
- Java RMI

Sometimes messaging is built in higher-level primitives

- Remote procedure call
- Remote method invocation

The Pi Calculus

The pi calculus is a process algebra (a la CCS)

Constructs for concurrency.

Communication on channels

- channels are first-class

  channel names can be sent on channels

- access restrictions for channels

In λ-calculus everything is a function

In Pi calculus everything is a process

Communication in the Pi Calculus

Processes communicate on channels:

c<M> send message M on channel c.

c(x) receives x on channel c.

Sequencing:

c<M>.p sends message M on c, then does p.

c(x).p receives x on c, then does p with x.

Concurrency:

p | q is the parallel composition of p and q.

Replication:

! p creates an infinite number of replicas of p

Examples

For example we might define

```
Speaker   = air<M>
Phone     = air(x).wire<x>
ATT       = wire(x).fiber<x>
System    = Speaker | Phone | ATT
```

Communication between processes is modeled by reduction:

```
Speaker | Phone → wire<M>
wire<M> | ATT   → fiber<M>
```

Composing these reductions we get:

```
Speaker | Phone | ATT   →  fiber<M>
```

Channel Visibility

Anybody can monitor an unrestricted channel:

Consider that we define

```
WireTap = wire(x).wire<x>.NSA<x>
```

- Copies the messages from the wire to NSA
- Possible since the name "`wire`" is globally visible

Now

```
WireTap | wire<M> | ATT !
wire<M>.NSA<M> | ATT !
NSA<M> | fiber<M>                  OOPS !
```

Restriction

The restriction operator `(νc)p` makes a fresh channel `c` within process `p`.

- ν is the Greek letter “nu”
- The name “c” is local (bound) in `p`

Restricted channels cannot be monitored.

```
wire(x) … | (ν wire)(wire<M> | ATT) →
wire(x) … | fiber<M>
```

The scope of the name “`wire`” is restricted

There is no conflict with the global `wire`

Restriction and Scope

Restriction

- is a binding construct (like λ)
- is lexically scoped
- allocates a new object (a channel)

`(νc)p` is like `let c = new Channel() in p`

In particular, `c` can be sent outside its scope.

- But only if `p` decides so

First-Class Channels

A channel c can leave its scope of declaration

- via a message d<c> from within p

Allowing channels to be sent as messages means communication topology is dynamic.

- If channels are not sent as messages (or stored in the heap) then the communication topology is static.
- This differentiates Pi-calculus from CCS

Example of First-Class Channels

Consider:

```
MobilePhone = air(x).cell<x>
ATT1        = wire<cell>
ATT2        = wire(y).y(x).fiber(x)
```

in

```
(ν cell)( MobilePhone | ATT1) | ATT2
```

ATT1 is trying to pass c out of the static scope of the restriction ν cell

Scope Extrusion

A channel is a name.

- First-class names must be usable even outside their original scope.

The pi calculus allows restrictions to move:

```
((ν c)p) | q = (ν c)(p|q)
```
if `c` not free in `q`

Renaming is needed in general:

```
((νc)p)|q =
((νd)[d/c]p)|q =
(νd)([d/c]p| q)
```
where `d` is fresh (not in p or q)

Example, Continued

```
(ν cell)(MobilePhone | ATT1) | ATT2 =
(ν cell)(MobilePhone | ATT1  | ATT2) →
(ν cell)(MobilePhone | cell(x).fiber<x>)
```

Scope extrusion distinguishes the pi calculus from other process calculi.

Syntax of the Pi Calculus

There are many versions of the Pi calculus

A basic version:

```
p,q ::=
        0         nil process
        x<y>.p    sending
        x(y).p    receiving
        p | q     parallel composition
        !p        replication
        (ν x)p    restriction
```

Note that only variables can be channels and messages

Operational Semantics

One basic rule of computation

$$\frac{}{x\langle y\rangle.p \mid x(z).q \;\rightarrow\; p \mid [y/z]q}$$

- Synchronous communication between a sender and a receiver
- Both the sender and the receiver proceed afterwards

Rules for identifying senders and receivers

$$\frac{p \rightarrow p'}{p \mid q \rightarrow p' \mid q} \qquad \frac{p \rightarrow p'}{(\nu x)p \rightarrow (\nu x)p'}$$

$$\frac{p \equiv p' \quad p' \rightarrow q' \quad q' \equiv q}{p \rightarrow q}$$

Structural Congruence

$$\frac{}{p \equiv p} \qquad \frac{q \equiv p}{p \equiv q} \qquad \frac{p \equiv q \quad q \equiv r}{p \equiv r}$$

$$\frac{p \equiv p'}{p \mid q \equiv p' \mid q} \qquad \frac{p \equiv p'}{(\nu x)p \equiv (\nu x)p'}$$

$$\begin{array}{rcll}
!p & \equiv & p \mid !p & \\
p \mid \mathtt{nil} & \equiv & p & \\
p \mid q & \equiv & q \mid p & \\
(\nu x)(\nu y)p & \equiv & (\nu y)(\nu x)p & \\
(\nu x)\mathtt{nil} & \equiv & \mathtt{nil} & \\
(\nu x)(p \mid q) & \equiv & (\nu x)p \mid q & \quad x \text{ not free in } q
\end{array}$$

Theory of Pi Calculus

Notes

- The Pi calculus does not have the Church-Rosser property

  Recall:

  ```
  WireTap|wire<M>|ATT →* NSA<M>|fiber<M>
  ```

  But also:

  ```
  WireTap|wire<M>|ATT →* WireTap| fiber<M>
  ```

- This captures the non-deterministic nature of concurrency

For Pi-calculus there are

- Type systems
- Equivalences and logics
- Expressiveness results, through encodings of numbers, lists, procedures, objects

Pi Calculus Applications

A number of languages are based on Pi calculus in whole or in part.

- e.g., Pict (Pierce and Turner)

Specification and verification.

- mobile phone protocols
- security protocols

The Pi Calculus and Security

The channels of the Pi calculus have nice built-in properties, such as:

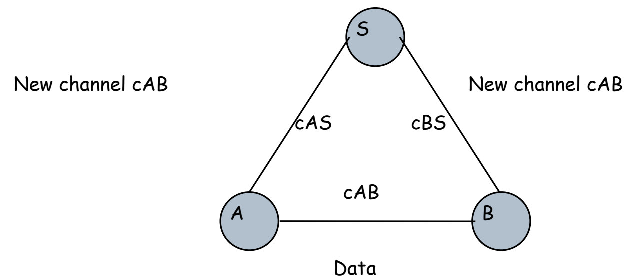
- integrity
- confidentiality (with ν)
- exactly-once semantics
- mobility (channels as first-class values)

These properties are useful in high-level descriptions of security protocols

More detailed descriptions are possible in the spi calculus (= pi calculus + cryptography)

A Typical Security Protocol

Establishment and use of a secret channel:



A and B are two clients

S is an authentication server

cAS and cBS are private channels with the server

cAB is a new channel for the clients

The Security Protocol in Pi Calculus

This protocol is described as follows:

```
A(M)      = (νcAB) cAS<cAB>.cAB <M>
S         = !(cAS(x).cBS<x> | cBS(x).cAS<x>)
B         = cBS(x).x(y) ...
System(M)= (νcAS)(νcBS) A(M) | S | B
```

- where ... represents what B does with the message it receives

Some Security Properties

An authenticity property

- For all N, if B receives N then A sent N to B

A secrecy property

- An outsider cannot tell System(M) apart from System(N), unless B reveals some part of A's message

Both of these properties can be formalized and proved in the Pi calculus

The secrecy property can be treated via a simple type system