Concurrency: Theory, Languages and Programming

– From CCS to PiLib –

Session 5 – November 19, 2003

Martin Odersky

EPFL-LAMP

Pilib

- Pilib is a library, which allows one to use CCS primitives in a Scala program.
- CCS constructs are modelled as Scala functions.
- Their implementation is based on Java's threads.
- Pilib's functions are implemented in two modules:
  - *concurrency* for general thread management.
  - *pilib* for CCS actions and sums.

An Example

Here is a two-place buffer implementation using Pilib.

```
import concurrency;   // make available Pilib functions
import pilib;         // without qualification.

module bufferExample with {
  def Buffer[a](in: Chan[a], out: Chan[a]): unit = {
    def B0: unit = { val x = in.read; B1(x) }
    def B1(x: a): unit = choice {
          out(x) * (B0) +
          in * (y => B2(x, y))
       }
    def B2(x: a, y: a): unit = { out.write(x); B1(y) }
    B0  // initial state
  }
}
```

Explanations

- *Chan* is the type of CCS names (or: channels).
- *Chan* takes a *type parameter* *a*, which determines the type of values that can be read from and written to the channel.
- The *Buffer* process is modelled by a recrusive Scala function, nested functions *B0*, *B1*, *B2*.
- Each nested function represents a buffer state (0 = empty, 1 = half full, 2 = full).

A Buffer Client

```
val random = new java.util.Random();

def Producer(n: int, l: Chan[String]): unit = {
  sleep(1 + random.nextInt(1000));
  l.write("object " + n);
  System.out.println("Producer gave " + n);
  Producer(n + 1, l)
}

def Consumer(r: Chan[String]): unit = {
  sleep(1 + random.nextInt(1000));
  val a = r.read;
  System.out.println("Consumer took " + a);
  Consumer(r)
}
```

```
def main(args: Array[String]): unit = {
  val in = new Chan[String];
  val out = new Chan[String];
  spawn < Producer(0, in) | Consumer(out) | Buffer(in, out) >
}
```

Covered CCS Syntax

Action prefix	π	$::=$	$x(y)$	receive y along x
		$\mid$	$\bar{x}\langle y \rangle$	send y along x
Guarded process	G	$::=$	$\pi.P$	
Process	P	$::=$	$\sum_i G_i$	summation
		$\mid$	$P_1 \mid P_2$	composition
		$\mid$	$\nu a.P$	restriction
		$\mid$	$A\langle x_1, ..., x_n \rangle$	agent
Agent definition	D	$::=$	$A(x_1, ..., x_n) = P$	
Term	t	$::=$	$D_1, ..., D_n \vdash P$	

From CCS to Pilib

Guarded process

$$\llbracket x(y).P \rrbracket \quad = \quad x \text{ * } (y \text{ => } \llbracket P \rrbracket)$$

$$\llbracket \bar{x}\langle v \rangle.P \rrbracket \quad = \quad x(v) \text{ * } (\llbracket P \rrbracket)$$

Process

$$\llbracket \pi_1.P_1 + \ldots + \pi_n.P_n \rrbracket \quad = \quad \text{choice (} \llbracket \pi_1.P_1 \rrbracket \text{ + ... + } \llbracket \pi_n.P_n \rrbracket \text{)}$$

$$\llbracket P_1 \mid \ \ldots \ \mid P_n \rrbracket \quad = \quad \text{spawn} < \llbracket P_1 \rrbracket \mid \text{...} \mid \llbracket P_n \rrbracket >$$

$$\llbracket \nu a.P \rrbracket \quad = \quad \text{\{ val } a \text{ = new Chan[T]; } \llbracket P \rrbracket \text{ \}}$$

$$\llbracket A\langle x_1, ..., x_n \rangle \rrbracket \quad = \quad A(x_1, ..., x_n)$$

Agent definition

$$\llbracket A(x_1, ..., x_n) = P \rrbracket \quad = \quad \text{def } A(x_1, ..., x_n)\text{: unit = } \llbracket P \rrbracket$$