Lecture 17
Concurrent Programming

29th October 2002

We've now looked at basic communication primitives and their use in some simple examples. Today we look at some additional notations for programming with communications: Original CSP and Modern CSP/CCS. First, discussion of the project and the homework:

Project

Who is doing it in Java? In C/C++ pthreads? How is it coming? What are the sticking points?

Homework

7.6 - Readers/writers – what did you discover about implementing the database using a server process? Did you achieve concurrency between readers? How?

7.7 - Time server process - what problems can you foresee in your implementation? Did you give the ticks priority over client requests?

7.18 - Sieve of Eratosthenes - using asynchronous message passing. What were the issues that arose in this problem?

7.2 - Sorting network using partition - what kind of sort does it create?

Original CSP

Introduced in late 1970's by Anthony Hoare (he of monitors fame). Hoare introduced CSP in an attempt to have a framework for answering the question "what does a particular process *do*?" That is "what is the formal meaning of processes?" Also, Hoare was concerned with the questions of what it meant to compose programs from multiple processes. As you've noticed the PL is helpful in this regard but isn't easy to use.

Original CSP was quite influential in language design in the late 70's and had strong influence on Ada - the language that USDoD mandated for all defense-related software development over the last two decades. Original CSP also started a flurry of work on the semantics and theory of concurrent programming which eventually led to Modern CSP, Milner's Calculus of Communicating Systems (CCS). This line of research continues to expand to incorporate new theories of mobile agents and security.

Original CSP featured:

- synchronous send and receives
- communication directed to named ports of named processes
- a sequential language resembling Dijkstra's guarded command language
- selective communication in the form of guarded communication statements
- a static collection of processes

Example: bounded buffer

```
process Producer { ... }
process Consumer { ... }
process Copy {
   char buffer[n];
   int front = 0, rear = 0, count = 0;
   do
      count<n: Producer?buffer[rear] -> count += 1; rear = (rear+1) mod n
   []
      count>0: Consumer!buffer[front] -> count -= 1; front = (front+1) mod n
   od
}
```

Modern CSP

Modern CSP (and CCS) have evolved to languages focussed purely on communications with notations for controlling naming such as name visibility and renaming. It uses named channels rather than the process/port naming of original CSP. Interestingly, modern CSP is not so much a programming language as a design language for talking about communications. For example, communication is no longer embedded in a sequential programming language - rather sequencing is expressed as evolution of processes.

Modern CSP has been extended with notations for talking about real-time communication.

Language definition (I actually present CCS):

- We have an infinite set of names, a, b, c, ... (think of them as names for communication channels)
- and a corresponding set of co-names: a, b, c, ...
- together the names and co-names constitute the set of *labels.*
- We can express *agents* (think of an agent as a process in a particular state) thusly
  - a - an agent that sends on channel a
  - a - an agent that receives on channel a
  - $P.Q$ - an agent that first does what agent P does and then what agent Q does
  - $P + Q$ - an agent that either performs what P does or peforms what Q does (choice)
  - $P \mid Q$ - concurrent agents P and Q
  - $A \equiv P$ (recursive) definition
  - $P \backslash L$ - restriction of names
  - $P[f]$ - renaming (f is a function from labels to labels)

Example:

```
LIGHT≡red.green.yellow.LIGHT
CONTROLLER≡red.green.yellow.CONTROLLER
(LIGHT | CONTROLLER)\ φ  // the light and controller communicate
                         // only with one another
```

Does this set of definitions remind you of something else you've encountered in your CS career?

Each process has a communication *language* (in the formal-languages sense) - a set of strings formed from symbols in its label set that it can possibly communicate. The combining rules for processes allow processes to restrict the communications that other processes can perform; ultimately we end up with a communication language for the entire system. The theories of CSP and CCS attempt to allow us to reason with and about these languages.

The meaning of communication: in CCS communication can occur when a sender and a receiver both offer their actions on a channel at the same time. The book talks about an extension that allows passing values as well.

You probably think that CCS seems rather clumsy as a way to program, but it turns out that people have had considerable success with both modern CSP and CCS as vehicles for specifying complex systems of parallel processes and automatically analyzing their behaviors for consistency with what is expected or desired, including analysis of real-time behavior (with extensions to the languages).