CS 5860
Thur.
Nov 3, 2011

A Note on Recursion and Introduction to Event Logic

PLAN

1. The fix constructor (also called Y-combinator) and recursive function definition

   the $3x+1$ function as an example

   recursion as $f = \lambda(x. F(x,f))$

   recursion as $fix(\lambda(f. \lambda(x. F(x,f))))$

2. In the final week we will mention type theory and discuss partial types (bar types) as an approach to partial correctness, a key topic as we saw last time. (A good research topic is to use partial types to reexpress Manna's theory.)

3. Review Peterson's Mutual Exclusion Algorithm

4. Message passing version of Peterson, by Wu and Rahli.

5. A Logical Model of Asynchronous Distributed Computing

   Events, causal-order, event orderings, etc.

2

10/17/2011

Computability in All Types

Here is how Computational Type Theory (CTT) defines recursive functions. Consider the 3x+1 function with natural number inputs.

```
f(x) = if x=0 then 1
        else if even(x) then f(x/2)
             else f(3x+1)
             fi
        fi
```

Alternative Syntax

```
f = function(x. if x=0 then 1
      else if even(x) then f(x/2)
           else f(3x+1))
```

Using Lambda Notation

```
f = λ(x. if x=0 then 1
       else if even(x) then f(x/2)
            else f(3x+1))
```

Here is a related term with function input f

```
λ(f. λ(x. if x=0 then 1
          else if even(x) then f(x/2)
               else f(3x+1)))
```

The recursive function is computed using this term.

Defining Recursive Functions in CTT

```
fix(λ(f. λ(x. if x=0 then 1
              else if even(x) then f(x/2)
                   else f(3x+1)
                   fi
              fi)))
```

Recursion in General

f(x) = F(f,x) is a recursive definition, also f = λ(x.F(f,x)) is another expression of it, and the CTT definition is:

fix(λ(f. λ(x. F(f,x)))

which reduces in one step to:

λ(x.F(fix(λ(f. λ(x. F(f,x)))),x))

by substituting the fix term for f in λ(x.F(f,x)) .

Non-terminating Computations

CTT defines all general recursive functions, hence non-terminating ones such as this

fix(λ(x.x))

which in one reduction step reduces to itself!

This system of computation is a simple functional programming language. In CTT it is essentially the programming language also used in the metatheory, ML. Later we add non-functional features as well.

CS 5860

Thur Nov 3, 2011

Concurrent Processes – Peterson's Algorithm

Recall the shared memory computing model

A critical section (cs) is an execution of a process P_i during which it has exclusive access to (a portion of) the state, e.g. P_i needs to write to memory without interference from other processes.

We saw that two processes that share the variables Q1, Q2 and Turn can provide mutually exclusive access to the critical portion of the state. Here is the code.

P1

```
Q1, Q2 : Bool
Turn : {0,1}
{ to enter cs }
Q1 := true ; Turn := 1
wait until (~Q2 ∨ Turn = 2)
    enter cs
    exit cs
    Q1 := false
end
```

P2

```
Q1, Q2 : Bool
Turn : {0,1}
{ to enter cs }
Q2 := true; Turn := 2
wait until (~Q1 ∨ Turn = 1)
    enter cs
    exit cs
    Q2 := false
end
```

Thur. Nov 3, 2011

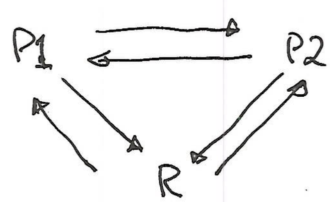
Peterson's Algorithm Properties

We can fairly easily see these properties of the algorithm.

Correctness : only one process is in the critical section "at a time," if P_i is in its CS then P_j for $j \neq i$ is not.

Liveness : if a process requests entry to CS, it will eventually get access.

Now we want to consider a message passing distributed version of the algorithm where the processes P1, P2 want to have mutually exclusive access to a resource at a separate location from those of P1, P2. The picture is



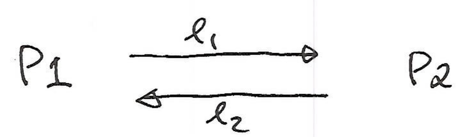
The arrows are communication channels, and communication is asynchronous. There is no global clock and no fixed time for a communication to complete. The processes can run at different speeds. We do assume that the communication channels are reliable, sometimes that they are FIFO, i.e. messages arrive on a channel in the order sent. (We can relax these assumptions by adding a process on the channel that drops, duplicates, and reorders messages if we want those assumptions.

CS 5860

Thur Nov 3, 2011

Messaging Passing Version of Peterson

Jason Wu and Vincent Rahli developed a distributed version of Peterson's algorithm based on token passing. Their algorithm is essentially the pseudo-code below. They have implemented it in Event ML.



One process starts with a token, and we can think of the value of a Boolean variable token? indicating whether the process has it. Each process executes this code to enter CS and to respond to requests for the token.

enter CS

    if token? then enter CS (busy := true; CS; busy := false)

    else (request_token; await token? then enter CS)

token request received

    if busy then (await ~ busy then send_token)

    else send_token

Consider

- Correctness
- Liveness

See notes <u>Peterson's Algorithm in a Distributed Environment</u> by Wu and Rahli.