Formal Theory of Communication Topology in Concurrent ML

Thomas Logan
July 8, 2018

1 Mathematical Artifacts

```
f(x) = x^2
```

```
1
      type thread_id
2
      val spawn: (unit -> unit) -> thread_id
3
4
      type 'a chan
5
      val channel : unit -> 'a chan
      val recv : 'a chan -> 'a
6
7
      val send : ('a chan * 'a) -> unit
1
2
      signature SERV = sig
3
        type serv
        val make : unit -> serv
4
5
        val call : serv * int -> int
6
7
8
      structure Serv : SERV = struct
9
         datatype serv = S of (int * int chan) chan
10
11
         fun make () = let
12
           val reqCh = channel ()
13
           fun loop state = let
14
             val (v, replCh) = recv reqCh
15
16
            send (replCh, state);
17
            loop v
18
           end
19
20
           spawn (fn () => loop 0);
21
          S reqCh
22
         end
23
24
        fun call (server, v) = let
25
          val S reqCh = server
26
           val replCh = channel ()
27
28
          send (reqCh, (v, replCh));
29
          recv replCh
30
         end
31
       end
32
1
2 type 'a event
3 val sync : 'a event -> 'a
4 val recvEvt : 'a chan -> 'a event
```

```
5 val sendEvt : 'a chan * 'a -> unit event
6 val choose: 'a event * 'a event -> 'a event
7
8 \text{ fun send (ch, v)} = \text{sync (sendEvt (ch, v))}
9 fun recv v = sync (recvEvt v)
11 val thenEvt: 'a event * ('a -> 'b event) -> 'b event
12
13
 2
      val server = Serv.make ()
4 \text{ val} = \text{spawn (fn () => Serv.call (server, 35))}
5 \text{ val} = \text{spawn (fn ()} =>
6
           Serv.call (server, 12);
 7 Serv.call (server, 13)
8)
9 \text{ val} = \text{spawn (fn ()} => \text{Serv.call (server, 81)})
10 val \_ = spawn (fn () => Serv.call (server, 44))
 1
2
 3
4
       structure Serv :> SERV = struct
5
     datatype serv = S of (int * int chan) chan
6
7
     fun make () = let
8
      val reqCh = FanIn.channel()
9
       fun loop state = let
10
         val (v, replCh) = FanIn.recv reqCh
11
12
         OneShot.send (replCh, state);
13
         loop v
14
       end
15
16
       spawn (fn () => loop 0);
17
       S reqCh
18
19
20
     fun call (server, v) = let
21
     val S reqCh = server
22
       val replCh = OneShot.channel ()
23
24
       FanIn.send (reqCh, (v, replCh));
25
       OneShot.recv replCh
26
     end
27 \, \, \mathrm{end}
28
```

```
1 let
2
    val w = 4
3
    val x = ref 1
4
    val y = ref 2
    val z = (!x + 1) + (!y + 2) + (w - 3)
5
6
    val w = 1
     in
     y := 0;
8
9
     (!y + 2) - (!x + 1) * (w - 3)
10
    end
11
2
    let
   val x = 1
3
4
   val y = 2
5
   val z = ref (4 * 73)
6 \quad val \quad x = 4
7 in
8 z := 1;
9 \times * !z
10 \text{ end}
11
1
   let
2
3
   val f = fn x \Rightarrow x 1
   val g = fn y => y + 2
4
5 val h = fn z => z + 3
6 in
7 	 (f g) + (f h)
8 end
9
10
           datatype 'a list = Nil | Cons 'a "'a list"
3
4 inductive sorted :: "('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a list \Rightarrow bool"
      where
     Nil : "sorted P Nil" |
5
     Single : "sorted P (Cons x Nil)" |
     Cons : "P x y \Longrightarrow sorted P (Cons y ys) \Longrightarrow sorted P (Cons x
        (Cons y ys))"
1
            datatype nat = Z \mid S nat
3 inductive lte :: "nat \Rightarrow nat \Rightarrow bool" where
```

```
4 Eq : "lte n n" |
    Lt : "lte n1 n2 \Longrightarrow lte n1 (S n2)"
5
6
7 theorem \"
    sorted lte (Cons (Z) (Cons (S Z) (Cons (S Z) (Cons (S (S (
      S Z))) Nil)))\"
9 apply (rule Cons)
10 apply (rule Lt)
   apply (rule Eq)
11
12 apply (rule Cons)
    apply (rule Eq)
13
14 apply (rule Cons)
15
    apply (rule Lt)
16
    apply (rule Lt)
   apply (rule Eq)
17
18 apply (rule Single)
19 done
1
2
           definition True :: bool where
3
4
    "True \equiv ((\lambdax::bool. x) = (\lambdax. x))"
6 definition False :: bool where
7 "False \equiv (\forallP. P)"
1
2 signature CHAN = sig
3 type 'a chan
   val channel: unit -> 'a chan
4
5 val send: 'a chan * 'a -> unit
6 val recv: 'a chan -> 'a
7 end
1
2
           structure ManyToManyChan : CHAN = struct
3
4
    type message_queue = 'a option ref queue
5
6
    datatype 'a chan_content =
7
      Send of (condition * 'a) queue |
      Recv of (condition * 'a option ref) queue |
8
9
      Inactive
10
11
    datatype 'a chan = Ch of 'a chan_content ref * mutex_lock
12
13
    fun channel () = Ch (ref Inactive, mutexLock ())
14
15
    fun send (Ch (contentRef, lock)) m =
16 acquire lock;
```

```
17
       (case !contentRef of
18
         Recv q =>
19
           let
20
             val (recvCond, mopRef) = dequeue q
21
22
             mopRef := Some m;
23
             if (isEmpty q) then contentRef := Inactive else ()
24
             release lock;
25
             signal recvCond;
26
             ()
27
           end
28
         Send q \Rightarrow
29
           let
30
             val sendCond = condition ()
31
           in
32
             enqueue (q, (sendCond, m));
33
             release lock;
34
             wait sendCond;
35
             ()
36
           end
37
         Inactive =>
38
           let
             val sendCond = condition ()
39
40
             contentRef := Send (queue [(sendCond, m)]);
41
42
             release lock;
43
             wait sendCond;
44
             ()
45
           end)
46
47
     fun recv (Ch (contentRef, lock)) =
48
       acquire lock;
49
       (case !contentRef of
         Send q =>
50
51
           let
52
             val (sendCond, m) = dequeue q
53
54
             if (isEmpty q) then contentRef := Inactive else ()
55
             release lock;
             signal sendCond;
56
57
             m
58
           end
59
         Recv q =>
60
           let
61
             val recvCond = condition ()
62
             val mopRef = ref None
63
           in
64
             enqueue (q, (recvCond, mopRef));
```

```
65
              release lock;
66
              wait recvCond;
67
              valOf (!mopRef) |
68
            \verb"end"
69
         Inactive =>
70
           let
71
              val recvCond = condition ()
72
              val mopRef = ref None
73
74
              contentRef := Recv (queue [(recvCond, mopRef)]);
75
              release lock;
76
              wait recvCond;
77
              valOf (!mopRef)
78
           end)
79
80 \ \mathtt{end}
```