Exceptions

Sound the alarm

15-150 M21

Lecture 0702
02 July 2021

aux-library/CPSIterate.sml

```
48 datatype result = Accept
49                 | Keep
50                 | Discard
51                 | Break of string
52
53 fun For (check : 'a -> result)
54         (L : 'a list)
55         (combine : 'a -> 'b -> 'b)
56         (base : 'b)
57         (success : 'a -> 'c)
58         (panic : string -> 'c)
59         (return : 'b -> 'c)
           : 'c
```

aux-library/CPSIterate.sml

```
61  =
62    let
63      fun run ([] : 'a list) (k:'b -> 'c) : 'c =
64            k base
65        | run (x::xs) k =
66            (case (check x) of
67               Accept => success x
68             |    Keep => run xs (k o (combine x))
69             | Discard => run xs k
70             |(Break s)=> panic s)
71    in
72      run L return
    end
```

Observation: Continuations allow us to "jump out" of a context

Calling success or panic breaks us out of run

```
61  =
62    let
63      fun run ([] : 'a list) (k:'b -> 'c) : 'c =
64            k base
65        | run (x::xs) k =
66            (case (check x) of
67               Accept => success x
68             |    Keep => run xs (k o (combine x))
69             | Discard => run xs k
70             |(Break s)=> panic s)
71    in
72      run L return
```

“Jumping out” is common enough to warrant special syntax/language features for it

0 Exn

exn is the type of exceptions

- SML has a built-in type called `exn`
- Some constructors of the type include
  - `Div : exn`
  - `Bind : exn`
  - `Match : exn`
  - `Fail : string -> exn`
  - `Empty : exn`
  - `Option : exn`
- Value of type `exn` are just like any other type:

0702.0 (exn.sml)

```
2 val 3 = case Div of
3             (Fail "s") => 1
4           | Bind => 2
5           | _ => 3
```

Note: This is not how we typically use values of type `exn`

exn is the extensible type

Under the hood, we can pretend `exn` is implemented something like

```
datatype exn= Div | Bind | Fail of string |...
```

But `exn` has a special feature: *extensibility*. The keyword `exception` declares a new *constructor* of type `exn`

0702.1 (exn.sml)

```
9  exception myExn1
10 exception myExn2 of int * string
```

0702.2 (exn.sml)

```
13 val 6 = case myExn2(6,"hhh") of
14           myExn1 => 4
15         | (myExn2 (n,"hhh")) => n
                                 | _ => 3
```

SML includes the keyword `raise`, which *raises* an exception.

0702.3 (exn.sml)

```
21 val k = raise myExn1
```

What is the type of `k`? Well, a raised exception has most general type `'a`, so it takes on whatever type it needs to.

0702.4 (exn.sml)

```
26 fun x () = if 3+(raise myExn1) = 5
27            then Int.toString(raise myExn1)
28            else raise myExn1
```

Useful for starter code

0702.5 (exn.sml)

```
32 fun findPartition A pL pR sc fc =
33   raise Fail "Unimplemented"
```

Extensionally Equivalent

0702.6 (exn.sml)

```
37 exception Negative
38 fun h_fact n =
39   case Int.compare(n,0) of
40        LESS => raise Negative
41     | EQUAL => 1
42     | GREATER => n * h_fact(n-1)
43
44 fun h_tfact n =
45   let
46     fun tfact 0 acc = acc
47       | tfact k acc = tfact (k-1) (k*acc)
48   in
49     if n<0 then raise Negative else tfact n 1
   end
```

NOT Extensionally Equivalent

0702.7 (exn.sml)

```
54 exception Neg of int
55 fun h_fact n =
56   case Int.compare(n,0) of
57        LESS => raise Negative
58     | EQUAL => 1
59     | GREATER => n * h_fact(n-1)
60
61 fun h_tfact n =
62   let
63     fun tfact 0 acc = acc
64       | tfact k acc = tfact (k-1) (k*acc)
65   in
66     if n<0 then raise Neg(Int.abs n) else tfact n 1
   end
```

Question:

Why *don't* we consider expressions which raise different exceptions to be extensionally equivalent?

If `e` is some expression with might raise exception `ex`, then we can "handle" the raised exception `ex` as follows

```
e handle ex => e'
```

Note:

- If `e` does not raise `ex`, then the whole expression has the same behavior as `e`
- If `e`, when evaluated, raises `ex`, then the whole expression has the same behavior as `e'`
- `e` and `e'` must have the same type
- If `e` raises an exception besides `ex`, that exception gets propagated

0702.8 (handling.sml)

```
3 fun safediv (m : int, n : int):int option=
4       SOME(m div n) handle Div => NONE
```

0702.9 (handling.sml)

```
8 fun safehd (L:'a list):'a option =
9   SOME(List.hd L) handle Empty => NONE
```

0702.10 (handling.sml)

```
13 exception NotDiv
14 exception OnlyDivBy2 of int
15 exception OnlyDivBy3 of int
16 fun sixdiv (n:int) : int =
17   case (n mod 2, n mod 3) of
18         (0,0) => n div 6
19     | (0,_) => raise (OnlyDivBy2 (n div 2))
20     | (_,0) => raise (OnlyDivBy3 (n div 3))
21     | _ => raise NotDiv
```

Pattern-matching exceptions

0702.11 (handling.sml)

```
24 fun printSixDivData n =
25 let val nStr = Int.toString n
26 in (nStr^" is divisible by 6: it is 6 times "^
27     (Int.toString(sixdiv n)) )
28
29   handle (OnlyDivBy2 d) =>
30     nStr^" is divisible by 2: it is 2 times "^
31     (Int.toString d)
32   | (OnlyDivBy3 d) =>
33     nStr^" is divisible by 3: it is 3 times "^
34     (Int.toString d)
35   | NotDiv =>
36     nStr^" is not divisible by 2,3, or 6."
nd
```

handle-nesting versus handle-casing

```
exception Bad1
exception Bad2
fun foo 1 = raise Bad1 | foo 2 = raise Bad2 |
  foo n = SOME n
```

```
(foo 1) handle Bad1 => foo 2
               | Bad2 => NONE
```

versus

```
(foo 1) handle Bad1 => foo 2
          handle Bad2 => NONE
```

Demonstration: Evaluation Traces with Exceptions

CPS treeSearch

0702.12 (handling.sml)

```
41 datatype 'a tree =
42     Empty | Node of 'a tree * 'a * 'a tree
43
44 fun search1 p Empty sc fc = fc ()
45   | search1 p (Node(L,x,R)) sc fc =
46     if p x then sc x else
47       search1 p L sc (fn () =>
48       search1 p R sc fc)
```

Use exception for failure continuation

0702.13 (handling.sml)

```
52 exception NotFound
53 fun search2 p Empty sc = raise NotFound
54   | search2 p (Node(L,x,R)) sc =
55       if p x then sc x else
56         search2 p L sc
57         handle NotFound => search2 p R sc
```

Use exceptions for both continuations

0702.14 (handling.sml)

```
61 fun search3 (p : 'a -> bool) T sc =
62   let
63     exception Found of 'a
64     fun look Empty = raise NotFound
65       | look (Node(L,x,R)) =
66           if p x then raise (Found x) else
67           (look L handle NotFound => look R)
68   in
69     look T handle (Found x) => sc x
70   end
```

5-minute break?

Demonstration:

`nQueens`

Summary

Next Time

Thank you!