Exceptions

ML provides an elegant exception handling mechanism

1. built-in exceptions
2. partial functions
3. user-defined exceptions
4. exception handling
5. further applications

Built-in Exceptions


```
- 5 div 0;
uncaught exception Div [divide by zero]
  raised at: ...

- chr(500);
uncaught exception Chr
  raised at: ...

- hd(nil: int list);
uncaught exception Empty
  raised at: ...
```

Exceptions for Non-Total Functions

Some functions are naturally undefined for some input.
Dealing with that can be awkward:

```
fun lookup_table x nil = NONE
  | lookup_table x ((x',v')::t)
      = if x = x' then SOME v'
                  else lookup_table x t;

(*   ''a -> (''a * 'b) list -> 'b option *)
```

Instead, one may explicitly raise an exception:

```
exception Empty_table;

fun lookup_table x nil = raise Empty_table
  | lookup_table x ((x',v')::t)
      = if x = x' then v'
                  else lookup_table x t;

(*   ''a -> (''a * 'b) list -> 'b  *)
```

Exceptions with Parameters


```
exception Empty_table of string;
fun lookup_table x nil
        = raise Empty_table(x)
  | lookup_table x ((x',v')::t)
        = if x = x' then v'
                    else lookup_table x t;

(* string -> (string * 'a) list -> 'a *)
```

Note: polymorphism of function has been lost

```
- lookup_table "mary" [("joe",12),("ed",7)];
uncaught exception Empty_table
```

```
- Empty_table;
val it = fn : string -> exn
```

```
- raise Empty_table;
Error: argument of raise is not an exception
```

Exception Handling

Wrap a handler around an exception returning expression:

```
fun lookup_table' x v
      = lookup_table x v
          handle Empty_table(s)
            => (print("Entry not found: ");
                print(x);
                print("\n");
                0);
(* string -> (string * int) list -> int *)
```

```
- lookup_table "ed" [("joe",12),("ed",7)];
val it = 7 : int
```

```
- lookup_table "mary" [("joe",12),("ed",7)];
Entry not found: mary
val it = 0 : int
```

Giving Change

Problem: given a set of coins (infinite supply of each denomination), produce

- exact change for a given amount
- involving minimal number of coins.

This may not always be possible

return 7c using 5c coins and 3c coins

but is always possible if we have 1c coins.

- We would like not to test all combinations

Greedy Strategy: return as many as possible from highest denomination, then as many as possible from second-highest denomination, etc.

- this is not always optimal:

  return 8c using 5c,4c,1c

- but for US coin set $\{25,10,5,1\}$ it is optimal (though not trivial to prove)

Greedy Implementation


```
fun change (coins,0) = []
|   change (c::coins,amount) =
      if amount < c
      then change(coins,amount)
        (* take largest coin possible *)
      else c::change(c::coins,amount-c)
```

```
change ([25,10,5,1],48);
val it = [25,10,10,1,1,1] : int list
```

```
change ([5,2],16);
uncaught exception Match
  [nonexhaustive match failure]
```

```
change ([5,4,1],8);
val it = [5,1,1,1] : int list
```

Exhaustive Search


```
(* expects: current_solution,coins,amount
   returns: list of solutions *)
fun FindAll (sol,_,0) = [sol]
|   FindAll (_,[],_) = []
|   FindAll(sol,c::coins,amount) =
      if amount < 0 then []
      else FindAll(c::sol,c::coins,amount-c)
        @  FindAll(sol,coins,amount)
fun change_exh(coins,amount) =
  FindAll([],coins,amount)
```

```
change_exh ([5,2],16);
val it = [[2,2,2,5,5],[2,2,2,2,2,2,2,2]]
  : int list list
```

```
change_exh([5,4,1],8);
val it = [[1,1,1,5],[4,4],[1,1,1,1,4],
  [1,1,1,1,1,1,1,1]] : int list list
```

```
change_exh ([25,10,5,1],48);
```

Exceptions for Backtracking


```
exception Failure
fun change1 (coins,0) = []
|   change1 ([],amount) = raise Failure
|   change1 (c::coins,amount) =
      if amount < c
      then change1(coins,amount)
      else (c :: change1(c::coins,amount-c)
            handle Failure =>
                   change1(coins,amount)
```

```
change1 ([5,2],16);
val it = [5,5,2,2,2] : int list
```

```
change1 ([25,10,5,1],48);
val it = [25,10,10,1,1,1] : int list
```

```
change1 ([5,4,1],8);
val it = [5,1,1,1] : int list
```