Foundations of Computer Science

A list application: making change

Dr. Robert Harle & Dr. Jeremy Yallop

2020–2021

An Application: Making Change

Till has **unlimited supply of coins**, for certain coin values

List of coins in till given in **descending order**

Larger coins preferred (tried first)

```
In[1]: let rec change till amt =
         match till, amt with
         | _, 0          -> []
         | [], _         -> raise (Failure "no more coins!")
         | c::till,amt -> if amt < c then change till amt
                          else c :: change (c::till) (amt - c)

Out[1]: val change : int list -> int -> int list = <fun>
```

The recursion terminates when amt $= 0$

Tries the **largest coin first**

The algorithm is **greedy**, and it **can fail**!

list of possible coin values

```
In[1]: let rec change till amt =
       match till, amt with
       | _, 0          -> []
       | [], _         -> raise (Failure "no more coins!")
       | c::till,amt -> if amt < c then change till amt
                        else c :: change (c::till) (amt - c)

Out[1]: val change : int list -> int -> int list = <fun>
```

The recursion terminates when amt $= 0$

Tries the **largest coin first**

The algorithm is **greedy**, and it **can fail**!

list of possible coin values

```
In[1]: let rec change till amt =
       match till, amt with
       | _, 0          -> []
       | [], _         -> raise (Failure "no more coins!")
       | c::till,amt -> if amt < c then change till amt
                        else c :: change (c::till) (amt - c)

Out[1]: val change : int list -> int -> int list = <fun>
```

The recursion terminates when amt $= 0$

Tries the **largest coin first**

The algorithm is **greedy**, and it **can fail**!

```
In[2]: let till = [50; 20; 10; 5; 2; 1]
Out[2]: val till : int list = [50; 20; 10; 5; 2; 1]
In[3]: change till 43
50  20 (amt=23)  20 (amt=3)  10  5  2 (amt=1)  1 (amt=0)
Out[3]: - : int list = [20; 20; 2; 1]
In[4]: let till = [5; 2]
Out[4]: val till : int list = [5; 2]
In[5]: change till 16
5 (amt=11)  5 (amt=6)  5 (amt=1)  2  ? amt≠0, till=[]
Out[5]: Exception: Failure "no more coins!"
```

```
In[2]: let till = [50; 20; 10; 5; 2; 1]
Out[2]: val till : int list = [50; 20; 10; 5; 2; 1]
In[3]: change till 43
~~50~~  20 (amt=23)  20 (amt=3)  ~~10~~  ~~5~~  2 (amt=1)  1 (amt=0)
Out[3]: - : int list = [20; 20; 2; 1]
In[4]: let till = [5; 2]
Out[4]: val till : int list = [5; 2]
In[5]: change till 16
5 (amt=11)  5 (amt=6)  5 (amt=1)  ~~2~~  ? amt≠0, till=[]
Out[5]: Exception: Failure "no more coins!"
```

```
 In[2]: let till = [50; 20; 10; 5; 2; 1]
Out[2]: val till : int list = [50; 20; 10; 5; 2; 1]
 In[3]: change till 43

50  20 (amt=23)  20 (amt=3)  10  5  2 (amt=1)  1 (amt=0)
Out[3]: - : int list = [20; 20; 2; 1]
 In[4]: let till = [5; 2]
Out[4]: val till : int list = [5; 2]
 In[5]: change till 16

5 (amt=11)  5 (amt=6)  5 (amt=1)  2  ? amt≠0, till=[]
Out[5]: Exception: Failure "no more coins!"
```

```
In[2]: let till = [50; 20; 10; 5; 2; 1]
Out[2]: val till : int list = [50; 20; 10; 5; 2; 1]
In[3]: change till 43
```

~~50~~ 20 (amt=23) 20 (amt=3) ~~10~~ ~~5~~ 2 (amt=1) 1 (amt=0)

```
Out[3]: - : int list = [20; 20; 2; 1]
In[4]: let till = [5; 2]
Out[4]: val till : int list = [5; 2]
In[5]: change till 16
```

5 (amt=11) 5 (amt=6) 5 (amt=1) ~~2~~ ? amt≠0, till=[]

```
Out[5]: Exception: Failure "no more coins!"
```

```
 In[2]: let till = [50; 20; 10; 5; 2; 1]
Out[2]: val till : int list = [50; 20; 10; 5; 2; 1]
 In[3]: change till 43
```

~~50~~ 20 (amt=23) 20 (amt=3) ~~10~~ ~~5~~ 2 (amt=1) 1 (amt=0)

```
Out[3]: - : int list = [20; 20; 2; 1]
 In[4]: let till = [5; 2]
Out[4]: val till : int list = [5; 2]
 In[5]: change till 16
```

5 (amt=11) 5 (amt=6) 5 (amt=1) ~~2~~ ? amt≠0, till=[]

```
Out[5]: Exception: Failure "no more coins!"
```

```
 In[2]: let till = [50; 20; 10; 5; 2; 1]
Out[2]: val till : int list = [50; 20; 10; 5; 2; 1]
 In[3]: change till 43
```

~~50~~ 20 (amt=23) 20 (amt=3) ~~10~~ ~~5~~ 2 (amt=1) 1 (amt=0)

```
Out[3]: - : int list = [20; 20; 2; 1]
 In[4]: let till = [5; 2]
Out[4]: val till : int list = [5; 2]
 In[5]: change till 16
```

5 (amt=11) 5 (amt=6) 5 (amt=1) ~~2~~ ? amt≠0, till=[]

```
Out[5]: Exception: Failure "no more coins!"
```

```
 In[2]: let till = [50; 20; 10; 5; 2; 1]
Out[2]: val till : int list = [50; 20; 10; 5; 2; 1]
 In[3]: change till 43
```

~~50~~ 20 (amt=23) 20 (amt=3) ~~10~~ ~~5~~ 2 (amt=1) 1 (amt=0)

```
Out[3]: - : int list = [20; 20; 2; 1]
 In[4]: let till = [5; 2]
Out[4]: val till : int list = [5; 2]
 In[5]: change till 16
```

5 (amt=11) 5 (amt=6) 5 (amt=1) ~~2~~ ? amt≠0, till=[]

```
Out[5]: Exception: Failure "no more coins!"
```

```
 In[2]: let till = [50; 20; 10; 5; 2; 1]
Out[2]: val till : int list = [50; 20; 10; 5; 2; 1]
 In[3]: change till 43
```

~~50~~ 20 (amt=23) 20 (amt=3) ~~10~~ ~~5~~ 2 (amt=1) 1 (amt=0)

```
Out[3]: - : int list = [20; 20; 2; 1]
 In[4]: let till = [5; 2]
Out[4]: val till : int list = [5; 2]
 In[5]: change till 16
```

5 (amt=11) 5 (amt=6) 5 (amt=1) ~~2~~ ? amt≠0, till=[]

```
Out[5]: Exception: Failure "no more coins!"
```

```
 In[2]: let till = [50; 20; 10; 5; 2; 1]
Out[2]: val till : int list = [50; 20; 10; 5; 2; 1]
 In[3]: change till 43
```

~~50~~ 20 (amt=23) 20 (amt=3) ~~10~~ ~~5~~ 2 (amt=1) 1 (amt=0)

```
Out[3]: - : int list = [20; 20; 2; 1]
 In[4]: let till = [5; 2]
Out[4]: val till : int list = [5; 2]
 In[5]: change till 16
```

5 (amt=11) 5 (amt=6) 5 (amt=1) ~~2~~ ? amt≠0, till=[]

```
Out[5]: Exception: Failure "no more coins!"
```

```
let rec change till amt =
    match till, amt with
    | _, 0          -> []
    | [], _         -> raise (Failure "no more coins!")
    | c::till, amt -> if amt < c then change till amt
                      else c :: change (c::till) (amt - c)
```

? amt $\neq$ 0, till =[]

(**Disclaimer**: This is rather hard!)

```
In[6]: let rec change till amt =
         match till, amt with
           | _     , 0   -> [ [] ]
           | []    , _   -> []
           | c::till , amt ->
             if amt < c then change till amt
             else let rec allc = function
                    | [] -> []
                    | cs :: css -> (c::cs) :: allc css
                  in
                    allc (change (c::till) (amt - c)) @
                         change till amt

Out[6]: val change : int list -> int -> int list list = <fun>
```

(**Disclaimer**: This is rather hard!)

```
In[6]: let rec change till amt =
         match till, amt with
         | _       , 0   -> [ [] ]
         | []      , _   -> []
         | c::till , amt ->
           if amt < c then change till amt
           else let rec allc = function
                  | [] -> []
                  | cs :: css -> (c::cs) :: allc css
                in
                  allc (change (c::till) (amt - c)) @
                       change till amt

Out[6]: val change : int list -> int -> int list list = <fun>
```

(**Disclaimer**: This is rather hard!)

```
In[6]: let rec change till amt =
         match till, amt with
         | _       , 0   -> [ [] ]
         | []      , _   -> []
         | c::till , amt ->
           if amt < c then change till amt
           else let rec allc = function
                  | [] -> []
                  | cs :: css -> (c::cs) :: allc css
                in
                  allc (change (c::till) (amt - c)) @
                       change till amt

Out[6]: val change : int list -> int -> int list list = <fun>
```

(**Disclaimer**: This is rather hard!)

```
In[6]: let rec change till amt =
         match till, amt with
         | _       , 0   -> [ [] ]   <-- success
         | []      , _   -> []
         | c::till , amt ->
           if amt < c then change till amt
           else let rec allc = function
                  | [] -> []
                  | cs :: css -> (c::cs) :: allc css
                in
                  allc (change (c::till) (amt - c)) @
                       change till amt

Out[6]: val change : int list -> int -> int list list = <fun>
```

(**Disclaimer**: This is rather hard!)

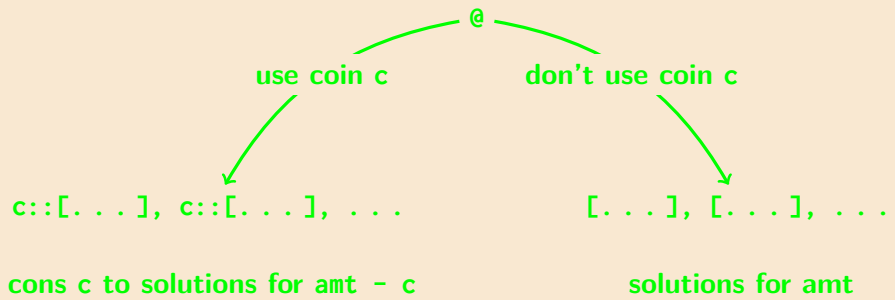
```
In[6]: let rec change till amt =
         match till, amt with
         | _       , 0   -> [ [] ]   <-- success
         | []      , _   -> []       <-- failure
         | c::till , amt ->
           if amt < c then change till amt
           else let rec allc = function
                  | [] -> []
                  | cs :: css -> (c::cs) :: allc css
                in
                  allc (change (c::till) (amt - c)) @
                       change till amt

Out[6]: val change : int list -> int -> int list list = <fun>
```

(**Disclaimer**: This is rather hard!)

```
In[6]: let rec change till amt =
         match till, amt with
         | _       , 0   -> [ [] ]   <-- success
         | []      , _   -> []       <-- failure
         | c::till , amt ->
           if amt < c then change till amt
           else let rec allc = function
                  | [] -> []
                  | cs :: css -> (c::cs) :: allc css
                in
                allc (change (c::till) (amt - c)) @
                     change till amt

Out[6]: val change : int list -> int -> int list list = <fun>
```

generates all possible solutions

```
let rec allc = function
| [] -> []
| cs :: css -> (c::cs) :: allc css
in
allc (change (c::till) (amt - c)) @ change till amt
```



```
In[7]: let till = [5; 3; 2];;
       change till 6

Out[7]: - : int list list = [[3; 3]; [2; 2; 2]]

In[8]: let till = [5; 2];;
       change till 10

Out[8]: - : int list list =
        [[2; 2; 2; 5; 5]; [2; 2; 2; 2; 2; 2; 2; 2]]
```

```
In[7]: let till = [5; 3; 2];;
       change till 6

Out[7]: - : int list list = [[3; 3]; [2; 2; 2]]

In[8]: let till = [5; 2];;
       change till 10

Out[8]: - : int list list =
        [[2; 2; 2; 2; 2]; [2; 2; 2; 2; 2; 2; 2; 2]]
```

ALL Ways of Making Change

```
In[7]: let till = [5; 3; 2];;
       change till 6

Out[7]: - : int list list = [[3; 3]; [2; 2; 2]]

In[8]: let till = [5; 2];;
       change till 10

Out[8]: - : int list list =
        [[2; 2; 2; 2; 2]; [2; 2; 2; 2; 2; 2; 2; 2]]
```

```
In[7]: let till = [5; 3; 2];;
       change till 6

Out[7]: - : int list list = [[3; 3]; [2; 2; 2]]

In[8]: let till = [5; 2];;
       change till 16

Out[8]: - : int list list =
        [[2; 2; 2; 5; 5]; [2; 2; 2; 2; 2; 2; 2; 2]]
```

```
In[7]: let till = [5; 3; 2];;
       change till 6

Out[7]: - : int list list = [[3; 3]; [2; 2; 2]]

In[8]: let till = [5; 2];;
       change till 16

Out[8]: - : int list list =
        [[2; 2; 2; 5; 5]; [2; 2; 2; 2; 2; 2; 2; 2]]
```

```
In[9]: let rec change till amt chg chgs =
         match till, amt with
         | _      , 0   -> chg::chgs
         | []     , _   -> chgs
         | c::till, amt ->
           if amt < 0 then chgs
           else change (c::till) (amt - c) (c::chg)
                       (change till amt chg chgs)

Out[9]: val change : int list -> int -> int list ->
                     int list list -> int list list
                   = <fun>
```

We've added **another accumulating parameter**!

Repeatedly improving simple code is called **stepwise refinement**.

```
In[9]: let rec change till amt chg chgs =
         match till, amt with
         | _       , 0   -> chg::chgs
         | []      , _   -> chgs
         | c::till , amt ->
           if amt < 0 then chgs
           else change (c::till) (amt - c) (c::chg)
                       (change till amt chg chgs)

Out[9]: val change : int list -> int -> int list ->
                     int list list -> int list list
                   = <fun>
```

We've added **another accumulating parameter**!

Repeatedly improving simple code is called **stepwise refinement**.

accumulators

```
In[9]: let rec change till amt chg chgs =
         match till, amt with
         | _       , 0   -> chg::chgs
         | []      , _   -> chgs
         | c::till , amt ->
           if amt < 0 then chgs
           else change (c::till) (amt - c) (c::chg)
                  (change till amt chg chgs)

Out[9]: val change : int list -> int -> int list ->
                     int list list -> int list list
                = <fun>
```

We've added **another accumulating parameter**!

Repeatedly improving simple code is called **stepwise refinement**.

accumulators

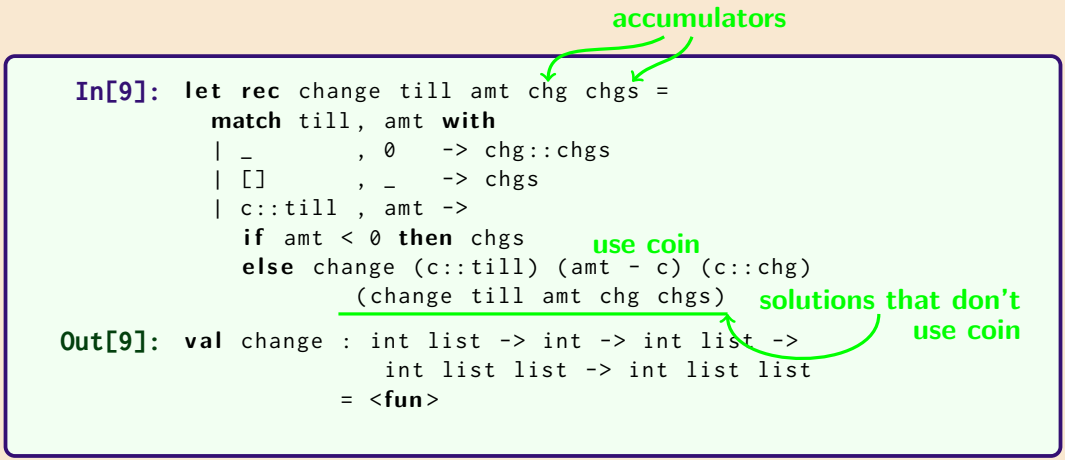
```
In[9]: let rec change till amt chg chgs =
         match till, amt with
         | _       , 0   -> chg::chgs
         | []      , _   -> chgs
         | c::till , amt ->
           if amt < 0 then chgs
           else change (c::till) (amt - c) (c::chg)
                  (change till amt chg chgs)

Out[9]: val change : int list -> int -> int list ->
                     int list list -> int list list
                = <fun>
```

use coin

We've added **another accumulating parameter**!

Repeatedly improving simple code is called **stepwise refinement**.

ALL Ways of Making Change — Faster!



We've added **another accumulating parameter**!

Repeatedly improving simple code is called **stepwise refinement**.

```
In[10]: change [5;3;2] 6 [] []
Out[10]: - : int list list = [[3; 3]; [2; 2; 2]]
```

```
In[10]: change [5;3;2] 6 [] []
Out[10]: - : int list list = [[3; 3]; [2; 2; 2]]
```

```
In[10]: change [5;3;2] 6 [] []
Out[10]: - : int list list = [[3; 3]; [2; 2; 2]]
```