Foundations of Computer Science Appending & reversing lists

Dr. Robert Harle & Dr. Jeremy Yallop 2020–2021

In[1]: let rec append xs ys =

| [] -> vs

x::xs -> x :: append xs ys

Out[1]: val append : 'a list -> 'a list -> 'a list = <fun>

```
In[1]: let rec append xs ys =
    match xs with
    | [] -> ys
    | x::xs -> x :: append xs ys
```

```
In[1]: let rec append xs ys =
    match xs with
    | [] -> ys
    | x::xs -> x :: append xs ys
Out[1]: val append : 'a list -> 'a list -> 'a list = <fun>
```

```
In[1]: let rec append xs ys =
    match xs with
    | [] -> ys
    | x::xs -> x :: append xs ys
Out[1]: val append : 'a list -> 'a list -> 'a list = <fun>
```

append [1; 2; 3] [4]

```
In[1]: let rec append xs ys =
    match xs with
    | [] -> ys
    | x::xs -> x :: append xs ys
Out[1]: val append : 'a list -> 'a list -> 'a list = <fun>
```

append [1; 2; 3] [4] \Rightarrow 1 :: append [2;3] [4]

```
In[1]: let rec append xs ys =
    match xs with
    | [] -> ys
    | x::xs -> x :: append xs ys
Out[1]: val append : 'a list -> 'a list -> 'a list = <fun>
```

```
In[1]: let rec append xs ys =
    match xs with
    | [] -> ys
    | x::xs -> x :: append xs ys
Out[1]: val append : 'a list -> 'a list -> 'a list = <fun>
```

```
In[1]: let rec append xs ys =
    match xs with
    | [] -> ys
    | x::xs -> x :: append xs ys
Out[1]: val append : 'a list -> 'a list -> 'a list = <fun>
```

```
In[1]: let rec append xs ys =
    match xs with
    | [] -> ys
    | x::xs -> x :: append xs ys
Out[1]: val append : 'a list -> 'a list -> 'a list = <fun>
```

ã

```
In[1]: let rec append xs ys =
    match xs with
    | [] -> ys
    | x::xs -> x :: append xs ys
Out[1]: val append : 'a list -> 'a list -> 'a list = <fun>
```

What is the time and space complexity of this function?

In[2]: let rec nrev = function

| x::xs -> (nrev xs) @ [x]

Out[2]: val nrev : 'a list-> 'a list = <fun>

nrev [a; b; c]

nrev [a; b; c] \Rightarrow nrev [b; c] @ [a]

 $\begin{array}{rrrr} \mathsf{nrev} \ [\mathsf{a}; \ \mathsf{b}; \ \mathsf{c}] & \Rightarrow & \mathsf{nrev} \ [\mathsf{b}; \ \mathsf{c}] \ @ [\mathsf{a}] \\ & \Rightarrow & (\mathsf{nrev} \ [\mathsf{c}] \ @ [\mathsf{b}]) \ @ [\mathsf{a}] \end{array}$

```
\begin{array}{rrrr} \mathsf{nrev} \ [\mathsf{a}; \ \mathsf{b}; \ \mathsf{c}] & \Rightarrow & \mathsf{nrev} \ [\mathsf{b}; \ \mathsf{c}] \ @ \ [\mathsf{a}] \\ & \Rightarrow & (\mathsf{nrev} \ [\mathsf{c}] \ @ \ [\mathsf{b}]) \ @ \ [\mathsf{a}] \\ & \Rightarrow & ((\mathsf{nrev} \ [] \ @ \ [\mathsf{c}]) \ @ \ [\mathsf{b}]) \ @ \ [\mathsf{a}] \end{array}
```

$$\begin{array}{rrrr} \mathsf{nrev} \ [\mathsf{a}; \ \mathsf{b}; \ \mathsf{c}] & \Rightarrow & \mathsf{nrev} \ [\mathsf{b}; \ \mathsf{c}] \ @ [\mathsf{a}] \\ & \Rightarrow & (\mathsf{nrev} \ [\mathsf{c}] \ @ [\mathsf{b}]) \ @ [\mathsf{a}] \\ & \Rightarrow & ((\mathsf{nrev} \ [] \ @ [\mathsf{c}]) \ @ [\mathsf{b}]) \ @ [\mathsf{a}] \\ & \Rightarrow & (([] \ @ [\mathsf{c}]) \ @ [\mathsf{b}]) \ @ [\mathsf{a}] \end{array}$$

$$\begin{array}{rcl} \mathsf{nrev} \ [\mathsf{a}; \ \mathsf{b}; \ \mathsf{c}] & \Rightarrow & \mathsf{nrev} \ [\mathsf{b}; \ \mathsf{c}] & @ [\mathsf{a}] \\ & \Rightarrow & (\mathsf{nrev} \ [\mathsf{c}] & @ [\mathsf{b}]) & @ [\mathsf{a}] \\ & \Rightarrow & ((\mathsf{nrev} \ [] & @ [\mathsf{c}]) & @ [\mathsf{b}]) & @ [\mathsf{a}] \\ & \Rightarrow & (([] & @ [\mathsf{c}]) & @ [\mathsf{b}]) & @ [\mathsf{a}] \\ & \Rightarrow & [\mathsf{c}; \ \mathsf{b}; \ \mathsf{a}] \end{array}$$

$$\begin{array}{rcl} \mathsf{nrev} \ [\mathsf{a}; \ \mathsf{b}; \ \mathsf{c}] & \Rightarrow & \mathsf{nrev} \ [\mathsf{b}; \ \mathsf{c}] & @ [\mathsf{a}] \\ & \Rightarrow & (\mathsf{nrev} \ [\mathsf{c}] & @ [\mathsf{b}]) & @ [\mathsf{a}] \\ & \Rightarrow & ((\mathsf{nrev} \ [] & @ [\mathsf{c}]) & @ [\mathsf{b}]) & @ [\mathsf{a}] \\ & \Rightarrow & (([] & @ [\mathsf{c}]) & @ [\mathsf{b}]) & @ [\mathsf{a}] \\ & \Rightarrow & [\mathsf{c}; \ \mathsf{b}; \ \mathsf{a}] \end{array}$$

What is the time and space complexity of this function?

$$\begin{array}{rcl} \mathsf{nrev} \ [\mathsf{a}; \ \mathsf{b}; \ \mathsf{c}] & \Rightarrow & \mathsf{nrev} \ [\mathsf{b}; \ \mathsf{c}] & @ [\mathsf{a}] \\ & \Rightarrow & (\mathsf{nrev} \ [\mathsf{c}] & @ [\mathsf{b}]) & @ [\mathsf{a}] \\ & \Rightarrow & ((\mathsf{nrev} \ [] & @ [\mathsf{c}]) & @ [\mathsf{b}]) & @ [\mathsf{a}] \\ & \Rightarrow & (([] & @ [\mathsf{c}]) & @ [\mathsf{b}]) & @ [\mathsf{a}] \\ & \Rightarrow & [\mathsf{c}; \ \mathsf{b}; \ \mathsf{a}] \end{array}$$

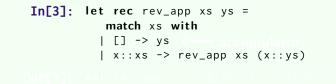
What is the **time and space complexity** of this function? **Recall**: append is O(n), and we have n(n + 1)/2 conses, which is $O(n^2)$

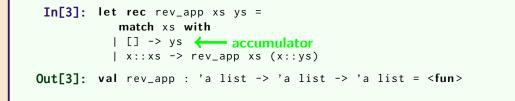
In[3]: let rec_rev_app xs ys =

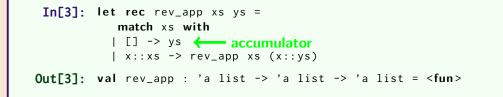
accumulator

x::xs -> rev_app xs (x::ys)

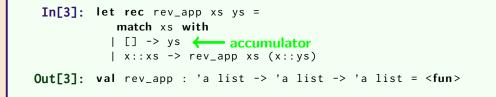
Out[3]: val rev_app : 'a list -> 'a list -> 'a list = <fun>







rev_app [a; b; c] []



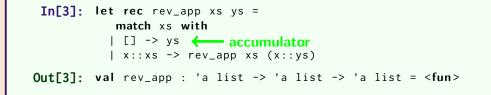
 $\mathsf{rev_app} \ [\mathsf{a}; \ \mathsf{b}; \ \mathsf{c}] \ \ [] \quad \Rightarrow \quad \mathsf{rev_app} \ [\mathsf{b}; \ \mathsf{c}] \ \ [\mathsf{a}]$



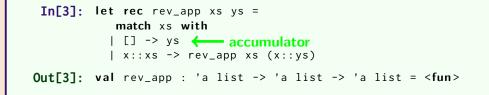
$$\begin{array}{rrrr} rev_app [a; b; c] [] \implies rev_app [b; c] [a] \\ \implies rev_app [c] [b; a] \end{array}$$



$$\begin{array}{rrrr} \mathsf{rev_app} \ [\mathsf{a}; \ \mathsf{b}; \ \mathsf{c}] & [] & \Rightarrow & \mathsf{rev_app} \ [\mathsf{b}; \ \mathsf{c}] \ [\mathsf{a}] \\ & \Rightarrow & \mathsf{rev_app} \ [\mathsf{c}] \ [\mathsf{b}; \ \mathsf{a}] \\ & \Rightarrow & \mathsf{rev_app} \ [] \ [\mathsf{c}; \ \mathsf{b}; \ \mathsf{a}] \end{array}$$



$$\begin{array}{rrrr} \mathsf{rev_app} \ [\mathsf{a}; \ \mathsf{b}; \ \mathsf{c}] \ \ [] & \Rightarrow & \mathsf{rev_app} \ [\mathsf{b}; \ \mathsf{c}] \ \ [\mathsf{a}] \\ & \Rightarrow & \mathsf{rev_app} \ [\mathsf{c}] \ \ [\mathsf{b}; \ \mathsf{a}] \\ & \Rightarrow & \mathsf{rev_app} \ [] \ \ [\mathsf{c}; \ \mathsf{b}; \ \mathsf{a}] \\ & \Rightarrow & \ [\mathsf{c}; \ \mathsf{b}; \ \mathsf{a}] \end{array}$$



$$\begin{array}{rrrr} rev_app [a; b; c] [] \implies rev_app [b; c] [a] \\ \Rightarrow rev_app [c] [b; a] \\ \Rightarrow rev_app [] [c; b; a] \\ \Rightarrow [c; b; a] \end{array}$$

What is the time complexity of this function?

An **interface** to rev_app:

In[4]: let revies = reviepp vs []
Out[4]: val revies 1 list -> 'a list = vfun>
In[5]: rev [].2.3]
Out[5]: rev [].2.3]

An **interface** to rev_app:

In[4]: let rev xs = rev_app xs []
Out[4]: val rev a list = list = sfun>
In[5]: rev list = [3: 2: 1]

An **interface** to rev_app:

In[4]: let rev xs = rev_app xs []
Out[4]: val rev : 'a list -> 'a list = <fun>
In[5]: new list = list

An **interface** to rev_app:

In[4]: let rev xs = rev_app xs []
Out[4]: val rev : 'a list -> 'a list = <fun>
In[5]: rev [1;2;3]
Out[5]

An **interface** to rev_app:

In[4]: let rev xs = rev_app xs []
Out[4]: val rev : 'a list -> 'a list = <fun>
In[5]: rev [1;2;3]
Out[5]: - : int list = [3; 2; 1]

In[6]:	
In[7]:	
In[8]:	

```
In[6]: let a = [2]
In[7]:
In[8]:
```

```
In[6]: let a = [2]
Out[6]: val a : int list = [2]
 In[7]:
 In[8]:
```

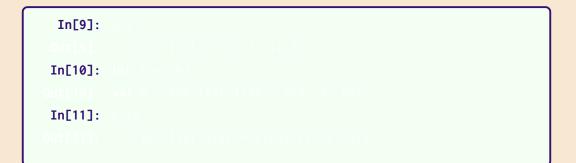
```
In[6]: let a = [2]
Out[6]: val a : int list = [2]
 In[7]: let b = [3; 4; 5]
 In[8]:
```

```
In[6]: let a = [2]
Out[6]: val a : int list = [2]
In[7]: let b = [3; 4; 5]
Out[7]: val b : int list = [3; 4; 5]
 In[8]:
```

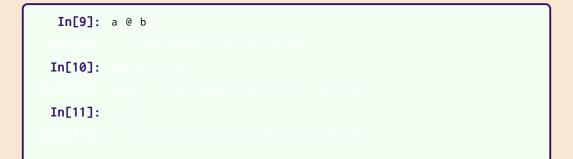
```
In[6]: let a = [2]
Out[6]: val a : int list = [2]
In[7]: let b = [3; 4; 5]
Out[7]: val b : int list = [3; 4; 5]
In[8]: a::b (* Q: what does this return? *)
```

```
In[6]: let a = [2]
Out[6]: val a : int list = [2]
 In[7]: let b = [3; 4; 5]
Out[7]: val b : int list = [3; 4; 5]
 In[8]: a::b (* Q: what does this return? *)
   Out: Line 1, characters 5-6:
        1 | a :: b
        Error: This expression has type int list
               but an expression was expected of type int list list
               Type int is not compatible with type int list
```

Question 3b: How to concatenate a and b?



Question 3b: How to concatenate a and b?



Question 3b: How to concatenate a and b?

```
In[9]: a @ b
Out[9]: - : int list = [2; 3; 4; 5]
In[10]: iet b = [b]
Out[10]: val b : int list list = [E3; 4; 5]]
In[11]: accb
Out[11]: - : int list list = [12]; E3, 4, 5]]
```

Question 3b: How to concatenate a and b?

```
In[9]: a @ b
Out[9]: - : int list = [2; 3; 4; 5]
In[10]: let b = [b]
Out[10]: val b = int list list = 113 4: 5]]
In[11]: a.b
Out[11]: - int list list = [12]: 14.4.5]]
```

Question 3b: How to concatenate a and b?

In[9]:	a@b	
Out[9]:	- : int	list = [2; 3; 4; 5]
In[10]:	let b =	: [b]
Out[10]:	val b :	int list list = [[3; 4; 5]]
In[11]:		

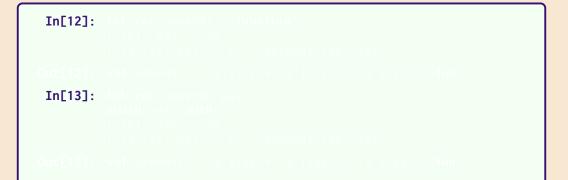
Question 3b: How to concatenate a and b?

```
In[9]: a @ b
Out[9]: - : int list = [2; 3; 4; 5]
In[10]: let b = [b]
Out[10]: val b : int list list = [[3; 4; 5]]
In[11]: a::b
Out[11]: - : int list list = [[2]: [3]: 4; 5]
```

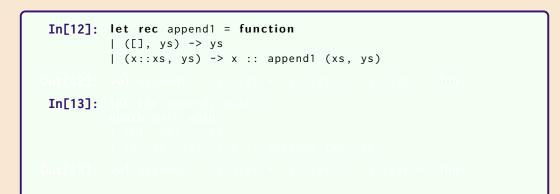
Question 3b: How to concatenate a and b?

In[9]:	a @ b
Out[9]:	- : int list = [2; 3; 4; 5]
In[10]:	let b = [b]
Out[10]:	val b : int list list = [[3; 4; 5]]
In[11]:	a : : b
Out[11]:	- : int list list = [[2]; [3, 4, 5]]

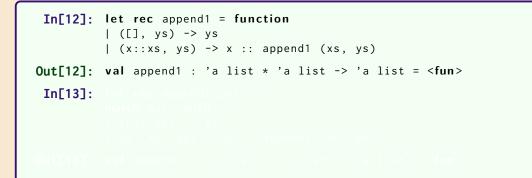
A Note on Notation: match vs function



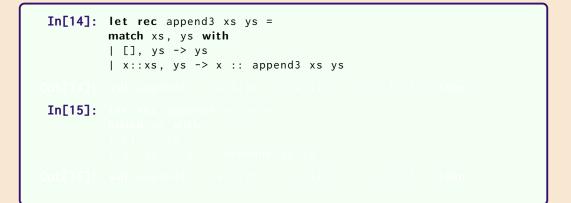
A Note on Notation: match vs function



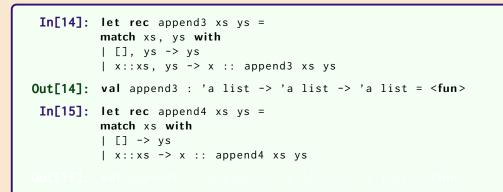
A Note on Notation: match vs function







In[14]:	<pre>let rec append3 xs ys = match xs, ys with [], ys -> ys x::xs, ys -> x :: append3 xs ys</pre>
Out[14]:	<pre>val append3 : 'a list -> 'a list -> 'a list = <fun></fun></pre>
In[15]:	



```
In[14]: let rec append3 xs ys =
    match xs, ys with
    | [], ys -> ys
    | x::xs, ys -> x :: append3 xs ys
Out[14]: val append3 : 'a list -> 'a list -> 'a list = <fun>
    In[15]: let rec append4 xs ys =
    match xs with
    | [] -> ys
    | x::xs -> x :: append4 xs ys
Out[15]: val append4 : 'a list -> 'a list -> 'a list = <fun>
```

Character constants

'A' '"'

String constants

"A" "B" "Oh, no!"

In[16]: String length 'abcde'

Out[16]: - : int = 5

In[17]: "0h," ^ " no!" (* concatenation *)

Out[17]: - : string = "Oh, no!'

Character constants

'A' '"'

String constants

"A" "B" "Oh, no!"

In[16]: String.length "abcde"

Out[16]: - : int = 5

In[17]: "0h," ^ " no!" (* concatenation *)

Out[17]: - : string = "Oh, no!"

Character constants

'A' '"'

String constants

"A" "B" "Oh, no!"

```
In[16]: String.length "abcde"
Out[16]: - : int = 5
In[17]: Out[17]: Ou
```

Character constants

'A' '"'

String constants

"A" "B" "Oh, no!"

Character constants

'A' '"'

String constants

"A" "B" "Oh, no!"

```
In[16]: String.length "abcde"
Out[16]: - : int = 5
In[17]: "Oh," ^ " no!" (* concatenation *)
Out[17]: - : string = "Oh, no!"
```