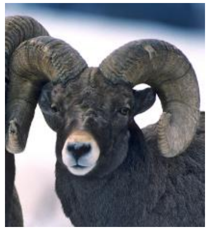
Lecture 9: Tail Recursion and Pairs

Kaleel Mahmood
Department of Computer Science and Engineering
University of Connecticut

Today's Problem: Rams Want Cake

- Problem: You are given a pack of rams and they are hungry!
- They want to make a special cake. If you do not make a cake for them, they will trample you.
- However, the rams only remember the instructions for baking the cake as they visit different stores to get the ingredients.
- Whenever they reach the last store they remember the complete recipe.

First way to solve the problem: Visit each store and leave a ram. (we need 3 rams)



I think we need eggs here but I don’t remember how many.

We need flour here but I forgot how much.

We need 1 cup of sugar here. Okay, now I remember we need 3 eggs and 2 cups of flour.

First way to solve the problem: Visit each store and leave a ram.

Go home and bake

I think we need eggs here but I don’t remember how many.

We need flour here but I forgot how much.

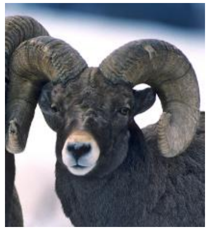
We need 1 cup of sugar here. Okay, now I remember we need 3 eggs and 2 cups of flour.

The Ram Cake Problem

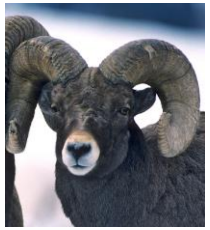
- What is the bottle neck? The recipe has 3 ingredients so you MUST have 3 rams. What if you have less? ***Is there a better way to do this?***

Solution: Don’t bake cakes for rams…
and get trampled…

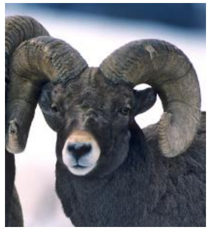
Only use 1 Ram



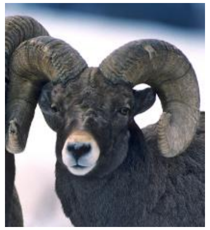
New Solution!



I think we need eggs here but I don’t remember how many. Take 1 dozen eggs.



We need flour here but I forgot how much. Take 1 lbs flour.

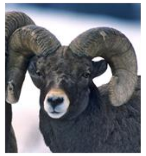


We need 1 cup of sugar here. Okay, now I remember we need 3 eggs and 2 cups of flour. We already have enough. Return home.

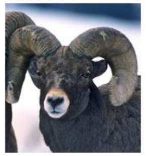
Now we can directly go home, no need to go back to the other stores to collect ingredients

Go home and bake

I think we need eggs here but I don't remember how many. Take 1 dozen eggs.



We need flour here but I forgot how much. Take 1 lbs flour.



We need 1 cup of sugar here. Okay, now I remember we need 3 eggs and 2 cups of flour. We already have enough. Return home.

What did we just do?

- We literally just showed you can save the number of rams needed in the cake problem by "accumulating" the ingredients as you go.
- We are going to discuss a very similar concept called tail recursion next. What are the computer science analogies here?

Ram (the animal) = RAM (the memory in your computer)

Leaving a ram at the store = Pending computation (takes up memory)

Accumulating ingredients = accumulating computations (to avoid using RAM)

RECURSION VS. ITERATION: “RECURSION”

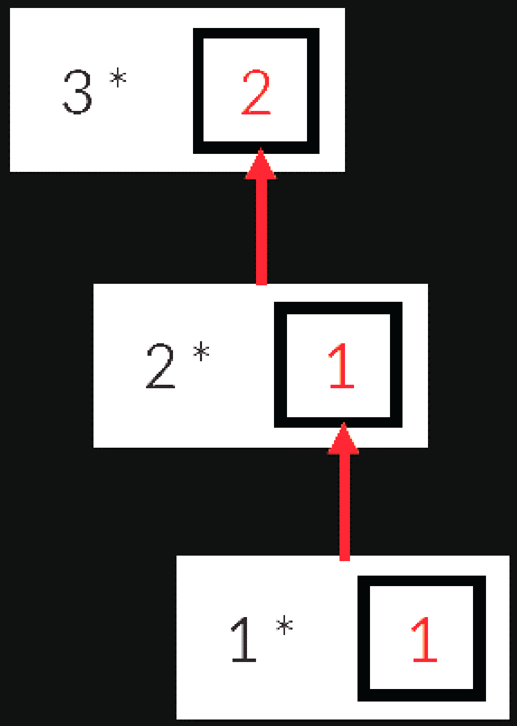
- Consider the familiar factorial function:

```
(define (factorial n)
  (if (= n 0)
      1
      (* n (factorial (- n 1)))))
```

- Let’s trace the evaluation of (fact 3). Note how the multiplications

$$(*3\ \square), (*2\ \square), \ldots$$

are pending while the recursive calls complete.



White Board Example 1

Example 1:

CSE 1729
Lecture 9

factorial (5) : 1R

$n \neq 0$

(* n factorial (4)) 2R

↓

$n \neq 0$ (* n factorial (3)) 3R

↓

$n \neq 0$ (* n factorial (2)) 4R

↓

$n \neq 0$ (* n factorial (1)) 5R

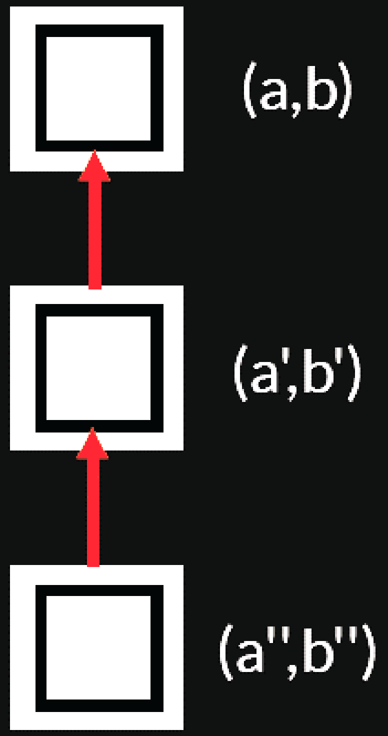
↓

$n \neq 0$ (* n factorial (0)) 6R

RECURSION VS. ITERATION: “ITERATION”

- Consider the `sqrt-converge` function we defined for extracting square roots:

```
(define (sqrt-converge a b)
  (let ((avg (/ (+ a b) 2)))
    (if (< (abs (- a b)) .000001) a
        (if (> (square avg) x)
            (sqrt-converge a avg)
```



- Note that a call to `(sqrt-converge a b)` typically generates a call to `(sqrt-converge a’ b’)`.
- In fact, the *result* of `(sqrt-converge a b)` is simply the the *result* of `(sqrt-converge a’ b’)` *without further processing or pending operations*.
- This is called tail recursion.

THIS RESULTS IN...

- New definition: function that computes a factorial *and* multiplies by a second "accumulator" argument.

```
(define (fact-accumulate n a)
  (if (= n 0) a
      (fact-accumulate (- n 1)
                       (* n a))))
```

Returns: `(factorial of n) x (a)`

White Board Example 2

Example 2:

CSE 1729
Lecture 9

fact-acc ($n=5$ $a=1$)   Want to get $5! = 5*4*3*2*1$

$n \neq 0$

↓

fast-acc ($n=4$, $a=1*5$)

$n \neq 0$

↓

fast-acc ($n=3$, $a=1*5*4$)

$n \neq 0$

↓

fast-acc ($n=2$, $a=1*5*4*3$)

$n \neq 0$

↓

fast-acc ($n=1$, $a=1*5*4*3*2$)

$n \neq 0$

↓

fast-acc ($n=0$, $a=1*5*4*3*2*1$)

$n=0$ return $a=5*4*3*2*1$

WRAPPING THIS TO CONCEAL THE INTERNAL MACHINERY

- New definition: function that computes a factorial *and* multiplies by a second "accumulator" argument.

```
(define (fact-tr n)
  (define (fact-accumulate m a)
    (if (= m 0) a
        (fact-accumulate (- m 1)
                         (* m a))))
  (fact-accumulate n 1))
```

Nothing Pending

- Now this is tail recursive.
- Why is ***accumulate*** an appropriate name for the second argument?

Scheme vs Python (Again)

Which has better memory performance in terms of function calls?

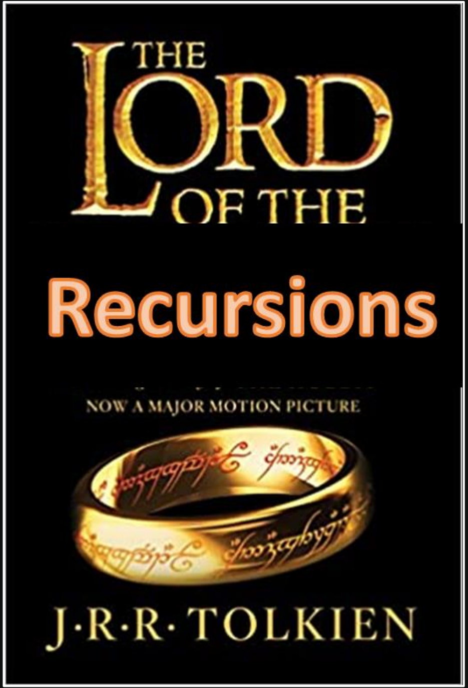
```scheme
(define (fact-tr n)
  (define (fact-accumulate m a)
    (if (= m 0) a
[redacted] (fact-accumulate (- m 1)
                                (* m a))))
  (fact-accumulate n 1))
```

```python
 #Define the factorial functio
def factorial(n):
    x = 1 #start with inital
    for i in range(n, 1, -1):
        x = x * i #This is sa
    return x
```

Structured Data In Scheme (Pairs and Lists)

OUR STORY THUS FAR...

- ...has focused on two "data-types:" numbers and functions.
  - (In fact, numeric data types are rather more complicated than you might think at first:
  - recall the difference between 4 and 4.0.)
- However, we often want to construct and manipulate more complicated *structured* data objects:
  - pairs of objects,
  - lists of objects,
  - trees, graphs, expressions, ...



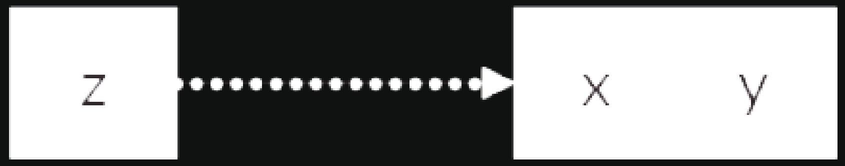
PAIRS

- Scheme has built-in support for *pairs* of objects. To maintain pairs, we require:
  - **A method for `cons`tructing a pair from two objects:**
    - In Scheme, this is the `cons` function. It takes two arguments and returns a pair containing the two values.
  - **A method of extracting the first (resp. second) object from a pair:**
    - In Scheme, these are two chimerically named functions: `car` and `cdr`.
    - Given a pair p, `(car p)` returns the first object in p; `(cdr p)` returns the second.

PAIRS

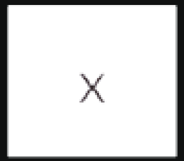
- Construction

```
(define z (cons x y))
```



- Access

```
(car z)
```




```
(cdr z)
```

Simple Pair Example in Scheme

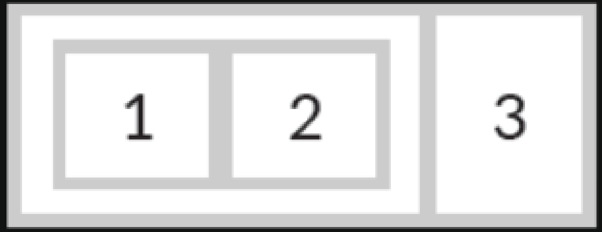
```scheme
(define z (cons 2 3))

(car z)

(cdr z)
```

```
Welcome to DrRacket, version 8.3 [cs].
Language: R5RS; memory limit: 128 MB.
2
3
>
```

EXAMPLES; NOTATION

```
> (cons 1 2)
(1 . 2)
> (define p (cons 1 2))
> (car p)
1
> (cdr p)
2
> (define q (cons p 3))
> (car q)
(1 . 2)
> (cdr q)
3
> (car (car q))
1
> (cdr (car q))
2
>
```

- Note that the interpreter denotes the pair containing the two objects a and b as: (a . b).
- Note that a coordinate of a pair can be... *another pair*! A natural diagram to represent this situation:



A COMPLEX NUMBER DATATYPE

- Recall that a complex number can be written $a + bi$, where i is $\sqrt{-1}$.
- To express a complex, we need to maintain two numbers
  - the real part and the complex part.
- We'll use Scheme pairs to represent complexes.
  - The first coordinate will hold the real part;
  - the second coordinate will hold the complex part.
- Thus:
  - construct a new complex number

```
(define (make-complex a b) (cons a b))
```

  - Extract the real part of a complex

```
(define (real-coeff c) (car c))
```

  - Extract the imaginary part of a complex

```
(define (imag-coeff c) (cdr c))
```

OPERATING ON COMPLEXES

- Adding complexes

```
(define (add-complex c d)
  (make-complex (+ (real-coeff c) (real-coeff d))
                (+ (imag-coeff c) (imag-coeff d))))
```

- Multiplying

$$(a_1 + b_1 i)(a_2 + b_2 i) = (a_1 a_2 - b_1 b_2) + (a_1 b_2 + b_1 a_2)i$$

```
(define (mult-complex c d)
  (make-complex (- (* (real-coeff c) (real-coeff d))
                   (* (imag-coeff c) (imag-coeff d)))
                (+ (* (real-coeff c) (imag-coeff d))
                   (* (imag-coeff c) (real-coeff d)))))
```

OTHER BASIC OPERATIONS

- Conjugate

```
(define (conjugate c)
  (make-complex (real-coeff c)
                (* -1 (imag-coeff c))))
```

- Modulus (length): two natural definitions:

```
(define (modulus c)
  (sqrt (real-coeff (mult-complex c (conjugate c)))))
```

or

```
(define (modulus-alt c)
  (define (square x) (* x x))
  (sqrt (+ (square (real-coeff c))
           (square (imag-coeff c)))))
```

RATIONAL NUMBERS ARE PAIRS

- A natural way to maintain a rational number is as a pair

```
(define (make-rat a b)
  (cons a b))

(define (denom r) (cdr r))
(define (numer r) (car r))
```

- Then, to multiply two rationals:

```
(define (mult-rat r s)
  (make-rat (* (numer r) (numer s))
            (* (denom r) (denom s))))
```

RATIONAL ADDITION, REDUCED FORM

- To add, we implement the familiar rule:

$$\frac{a}{b} + \frac{c}{d} = \frac{ad+bc}{bd}$$

- Thus:

```
(define (add-rat r s)
  (make-rat (+ (* (numer r) (denom s))
               (* (numer s) (denom r)))
            (* (denom r) (denom s))))
```

- Note that this implementation does not simplify fractions into reduced form.

REDUCING A FRACTION

- Note that

$$\frac{a}{b} = \frac{a/\alpha}{b/\alpha} \text{ if } \alpha \text{ divides } a \text{ and } b$$

- And hence we can always reduce a fraction by the rule:

$$\frac{a}{b} \rightsquigarrow \frac{a/gcd(a,b)}{b/gcd(a,b)}$$

- We could make a simplify function, or just redefine `make-rat`, so that all rationals are automatically in reduced form:

```
(define (make-rat a b)
  (let ((d (gcd a b)))
    (cons (/ a d)  (/ b d))))
```

EXAMPLES

- Using this new, automatically reducing package:

```
1 > (define r (make-rat 2 6))
```

Figure Sources

- http://wp.nathabblog.com/wp-content/uploads/2014/07/Img50224_HenryHoldsworth_Web.jpg
- https://www.supermarketnews.com/sites/supermarketnews.com/files/styles/article_featured_retina/public/Big_Y_Foods_supermarket-Walpole_MA.png?itok=EWukDJBM
- https://media.nbcconnecticut.com/2020/12/Stop-and-Shop-generic.jpg?quality=85&strip=all&resize=850%2C478
- https://fastly.4sqi.net/img/general/600x600/19395527_KJo7YA5PbiGkxYA8psPKgzyS59-o7ZugBUiPUeEVF9s.jpg
- https://pbs.twimg.com/media/ESdKGZgWkAA1aRl.jpg
- https://upload.wikimedia.org/wikipedia/commons/d/db/Swissbit_2GB_PC2-5300U-555.jpg
- https://thescranline.com/wp-content/uploads/2021/03/Chocolate-Cake.jpg
- https://cdn.britannica.com/92/80592-050-86EF29F3/Mouflon-ram.jpg
- https://images-na.ssl-images-amazon.com/images/I/51kfFS5-fnL._SX332_BO1,204,203,200_.jpg