Prog & Alg I (COMS10002) Week 5 - Intro to Theory

Dr. Oliver Ray
Department of Computer Science
University of Bristol

Wednesday 29th October, 2014

Timetable for Weeks 5-8

	Mon	Tue	Wed	Thu	Fri
9		🟥			
10		🟥 LAB (Group 1)	🟩 LEC (Oliver)		
11		🟥			
12					
1					
2		🟥			
3	🟧 LEC (IAN)	🟥 LAB (Group 2)			
4		🟥		🟦 TUT (Group 2)	
5				🟦 TUT (Group 1)	

🟧	C-Programming
🟥	Coursework (or Lab Exam)
🟩	Theory
🟦	Worksheet

Key Topics for Theory Content

- Introduction to program correctness
  - Show that a program computes what is intended by logical specification, induction, loop invariants, …
- Introduction to program complexity
  - Show how its performance scales to larger inputs by asymptotic function approximation, Big-O, …
- Introduction to program transformation
  - Rewrite program equivalently but more efficiently by accumulator variables, tail recursion, …

Integer Exponentiation: Definition

- Recall the definition of integer exponentiation

$$x^n = \begin{cases} 1 & \mathit{if} \quad n = 0 \\ x.x^{n-1} & \mathit{if} \quad n > 0 \end{cases}$$

- Let's just check this is correct by trying some examples

e.g. 2^5 5^2 $(-2)^5$ $(-5)^2$ 1^2 1^1 1^0 0^2 0^1 0^0

- Thus, we can agree it is correct for all $(x,n) \in (Z \times N)/\{(0,0)\}$ (as mathematicians usually regard 0^0 as being *undefined*)

Integer Exponentiation: Code

- Recall the corresponding C-code (omitting types)

```
p(x,n) {
    if (n==0) return 1;
    else return x*p(x,n-1);
}
```

- And confirm it behaves as expected

e.g. 2^5 5^2 $(-2)^5$ $(-5)^2$ 1^2 1^1 1^0 0^2 0^1 $\cancel{0^0}$

- So, it works in these particular cases; but can we prove that it is *always* correct?

Observe the close fit of definition and code

$$x^n = \begin{cases} 1 & \text{if} \quad n = 0 \\ x.x^{n-1} & \text{if} \quad n > 0 \end{cases}$$

```
p(x,n) {
   if (n==0) return 1;
   else return x*p(x,n-1);
}
```

3^5

$=3(3^4)$

$=3(3(3^3))$

$=3(3(3(3^2)))$

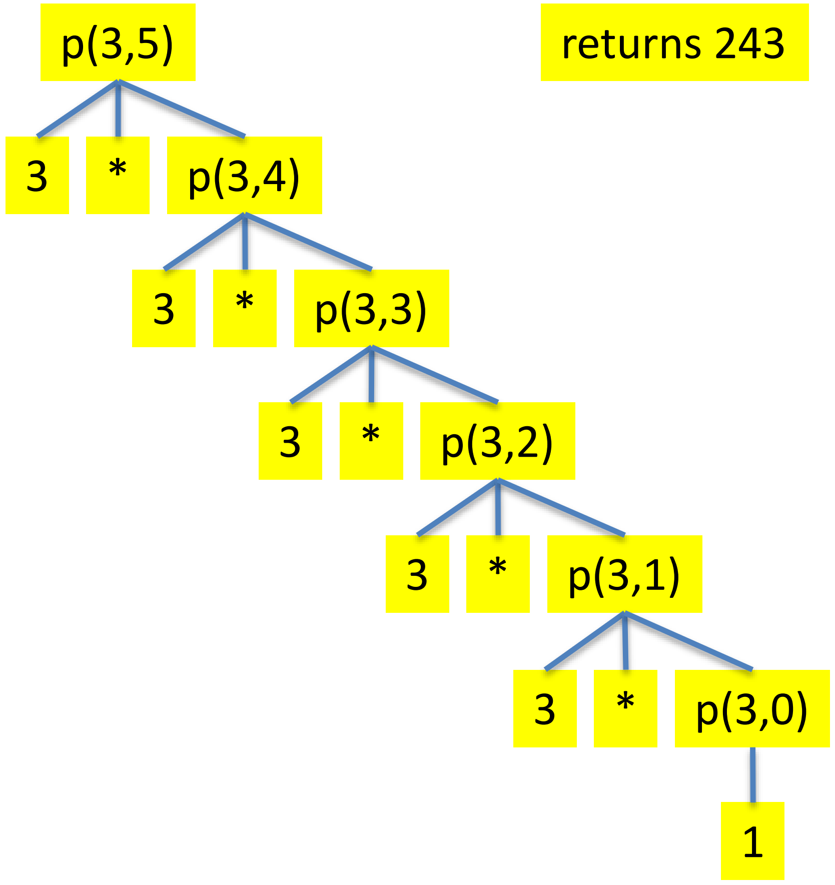
$=3(3(3(3(3^1))))$

$=3(3(3(3(3(3^0)))))$

$=3(3(3(3(3(1)))))$

$= 243$

returns 243



Now do a proof by induction

- Theorem

  p(x, n) returns x^n for all $(x, n) \in (\mathrm{Z} \times \mathrm{N}) / \{(0,0)\}$

- Proof

  by induction on n

  Base Case (n = 0)

➔ show p$(x,0)$ returns x^0 for all $x \in Z / \{0\}$ (next slide)

  Induction Step (n = k > 0)

  assume p$(x, k-1)$ returns x^{k-1} for some $\mathrm{k} > 0$ and all $x \in Z$

➔ show p(x, k) returns x^k (slide after next)

Base Case

$$x^n = \begin{cases} 1 & \text{if} \quad n = 0 \\ x.x^{n-1} & \text{if} \quad n > 0 \end{cases}$$

```
p(x,n) {
    if (n==0) return 1;
    else return x*p(x,n-1);
}
```

- We need to show that p(x,0) returns x^0

- But p(x,0) returns 1 by the if-case of its definition
- And $x^0=1$ by the base case of its definition
- Thus p(x,0) returns x^0

QED!

Induction Step

$$x^n = \begin{cases} 1 & \text{if} \quad n = 0 \\ x.x^{n-1} & \text{if} \quad n > 0 \end{cases}$$

```
p(x,n) {
    if (n==0) return 1;
    else return x*p(x,n-1);
}
```

- Assuming that p(x,k-1) returns x^{k-1} where k>0
- We need to show p(x,k) returns x^k

- But p(x,k) returns x*p(x,k-1) by the else-case of its definition
- Which equals $x.x^{k-1}$ by the inductive hypothesis
- And $x^k = x.x^{k-1}$ by the recursive case of its definition
- Thus p(x,k) returns x^k for all x

QED!

NB: Strong induction can be more convenient

- <u>Theorem</u>

  $\mathrm{p}(x, n)$ returns x^n

- <u>Proof</u>

  by induction on n

  <u>Base Case</u> (n = 0)

➔ show $\mathrm{p}(x,0)$ returns x^0

  <u>Induction Step</u> (n = k > 0)

  assume $\mathrm{p}(x, n)$ returns x^n for all $\mathrm{n} \in [0, \mathrm{k})$

➔ show $\mathrm{p}(x, k)$ returns x^k

i.e. Assume true for all n<k
(not just k-1)

Tail Recursion

- Recursive programs are easy to write and reason about but can be wasteful of stack resources (variable copies)
- Tail recursion is a special form of recursion which can be handled very efficiently by modern compilers
- Tail recursive calls must all occur at the end of a branch of the computation which simply returns the result of the recursive call (unmodified in any way)

```
f(x1,..,xn) {
    …
    return f(y1,..,yn);
    …
}
```

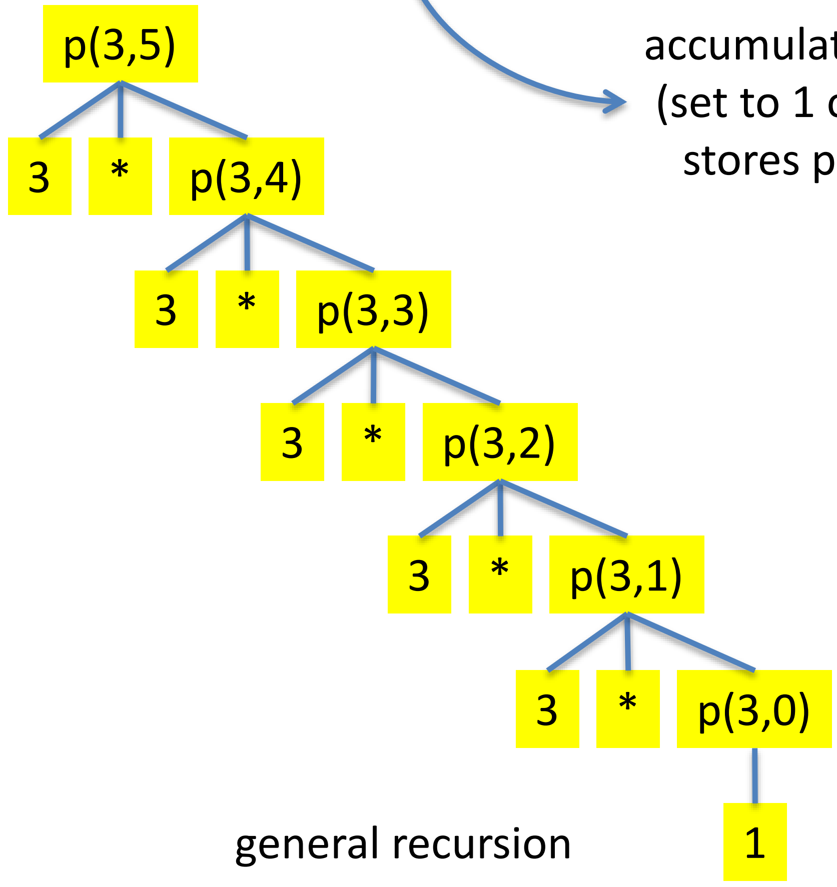
- Tail recursion is often achieved by adding accumulator variables into a function prototype (here it is easy but this is *not* always the case in practice!)

From General Recursion to Tail Recursion

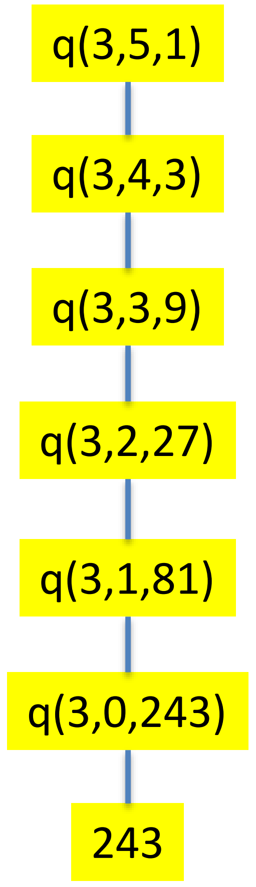
```
p(x,n) {
   if (n==0) return 1;
   else return x*p(x,n-1);
}
```

```
q(x,n,a) {
   if (n==0) return a;
   else return q(x,n-1,x*a);
}
```

accumulator variable a
(set to 1 on initial call)
stores partial result



general recursion



tail recursion

Exercise

- Prove (by induction) that

$$\mathrm{q}(x,n,a)\ \text{returns}\ \mathrm{a}x^{\mathrm{n}}\ \text{for all}\ a \in Z,\ (\mathrm{x},\mathrm{n}) \in (N \times Z)/\{(0,0)\}$$

- Hint: modify the previous proof!

Recursion and Iteration

- Recursion can always be transformed into iteration and vice versa (though this is sometimes difficult in practice)
- But tail recursion can be easily turned into iteration by
  - treating function parameters as variables
  - initialising accumulator variables to their intended start value
  - setting the loop condition as the negation of any base case tests
  - simulating the recursive parameter computations within the body of the loop
- So we can obtain an equivalent program that won’t run out of stack space when given large inputs!

From Tail Recursion to Iteration

```
q(x,n,a) {
    if (n==0) return a;
    else return q(x,n-1,x*a);
}
```

```
r(x,n) {
    int a=1;
    while (n!=0) {n--;a*=x;}
    return a;
}
```

$$\begin{tikzcd}
q(3,5,1) \arrow[d, no head] \\
q(3,4,3) \arrow[d, no head] \\
q(3,3,9) \arrow[d, no head] \\
q(3,2,27) \arrow[d, no head] \\
q(3,1,81) \arrow[d, no head] \\
q(3,0,243) \arrow[d, no head] \\
243
\end{tikzcd}$$

tail recursion

#	x	n	a
0	3	5	1
1	3	4	3
2	3	3	9
3	3	2	27
4	3	1	81
5	3	0	243

iteration