CSC165 Week 10

Larry Zhang, November 11, 2014
today’s outline

➔ big-Ω proof

➔ big-O proofs for general functions

➔ introduction to computability
Recap of definitions

**upper bound**

A function $f(n)$ is in $O(n^2)$ iff

$$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \text{ such that } \forall n \in \mathbb{N}, n \geq B \implies f(n) \leq cn^2$$

**lower bound**

A function $f(n)$ is in $\Omega(n^2)$ iff

$$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \text{ such that } \forall n \in \mathbb{N}, n \geq B \implies f(n) \geq cn^2$$
Recap of a proof for big-O

\[7n^6 - 5n^4 + 2n^3 \in \mathcal{O}(6n^8 - 4n^5 + n^2)\]

call \( B = 1 \) (magic brk-pt)

assume \( n \geq 1 \)

\begin{align*}
6n^8 &- 4n^5 + n^2 \\
6n^8 &- 4n^5 \\
6n^8 &- 4n^8 = 2n^8
\end{align*}

\[9n^6 \leq \frac{9}{2} \cdot 2n^8\]

\begin{align*}
7n^6 &+ 2n^6 = 9n^6 \\
7n^6 &+ 2n^3 \\
7n^6 &- 5n^4 + 2n^3
\end{align*}

pick a \textbf{c large} enough to make the right side an upper bound
There exist $c \in \mathbb{R}^+$ and $B \in \mathbb{N}$ such that $\forall n \in \mathbb{N}, n \geq B \implies f(n) \geq cn^2$

**now a new proof**

Prove $n^2 + n \in \Omega(15n^2 + 3)$
Proof: \[ n^2 + n \in \Omega(15n^2 + 3) \]
takeaway

choose Magic Breakpoint $B = 1$, then we can assume $n \geq 1$

**under-estimation tricks**

$\rightarrow$ **remove a positive** term
- $3n^2 + 2n \geq 3n^2$

$\rightarrow$ **multiply a negative** term
- $5n^2 - n \geq 5n^2 - n \times n = 4n^2$

**over-estimation tricks**

$\rightarrow$ **remove a negative** term
- $3n^2 - 2n \leq 3n^2$

$\rightarrow$ **multiply a positive** term
- $5n^2 + n \leq 5n^2 + n \times n = 6n^2$

**simplify the function without changing the highest degree**
all statements we have proven so far

\[
3n^2 + 2n \in \mathcal{O}(n^2)
\]
\[
3n^2 + 2n + 5 \in \mathcal{O}(n^2)
\]
\[
7n^6 - 5n^4 + 2n^3 \in \mathcal{O}(6n^8 - 4n^5 + n^2)
\]
\[
n^3 \notin \mathcal{O}(3n^2)
\]
\[
2^n \notin \mathcal{O}(n^2)
\]
\[
n^2 + n \in \Omega(15n^2 + 3)
\]
general statements about big-Oh
a definition

\[ \mathcal{F} : \{ f : \mathbb{N} \rightarrow \mathbb{R}^{\geq 0} \} \]

The **set** of all functions that take a natural number as input and return a non-negative real number.
now prove

$$\forall f, g, h \in \mathcal{F}, (f \in \mathcal{O}(g) \land g \in \mathcal{O}(h)) \Rightarrow f \in \mathcal{O}(h)$$

Intuition:

If \(f\) grows no faster than \(g\), and \(g\) grows no faster than \(h\), then ...
thoughts

∀f, g, h ∈ F, (f ∈ O(g) ∧ g ∈ O(h)) ⇒ f ∈ O(h)

want to find \( B'' \), \( c'' \), so that \( f(n) \leq c'' h(n) \) beyond \( B'' \)

Beyond \( B'' \):

want \( f \leq c'' h \)
Proof: \( \forall f, g, h \in \mathcal{F}, (f \in \mathcal{O}(g) \land g \in \mathcal{O}(h)) \Rightarrow f \in \mathcal{O}(h) \)
another general statement
Prove $\forall f, g \in \mathcal{F}, f \in O(g) \Rightarrow g \in \Omega(f)$

Intuition:
if $f$ grows no faster than $g$,
then ...
**Prove** \( \forall f, g \in \mathcal{F}, f \in \mathcal{O}(g) \Rightarrow g \in \Omega(f) \)

**thoughts:**

Assume \( f \in \mathcal{O}(g) : n \geq B \Rightarrow f \leq cg \)

Want to pick \( B', c' \) \( g \in \Omega(f) : n \geq B' \Rightarrow g \geq c' f \)
Proof \( \forall f, g \in \mathcal{F}, f \in \mathcal{O}(g) \Rightarrow g \in \Omega(f) \)
yet another general statement
Prove: \( \forall f, g, h \in \mathcal{F}, (f \in \mathcal{O}(h) \land g \in \mathcal{O}(h)) \Rightarrow (f + g) \in \mathcal{O}(h) \)

thoughts:

Assume \( f \in \mathcal{O}(h) : n \geq B \Rightarrow f \leq ch \)

and \( g \in \mathcal{O}(h) : n \geq B' \Rightarrow g \leq c'h \)

Want to pick \( B'', c'' \) \( (f + g) \in \mathcal{O}(h) : n \geq B'' \Rightarrow (f + g) \leq c''h \)
yet another one, trickier
\[ \forall f, g \in \mathcal{F}, f \in \mathcal{O}(g) \Rightarrow f \in \mathcal{O}(g \cdot g) \]

\[ f \in \mathcal{O}(n) \Rightarrow f \in \mathcal{O}(n^2) \]

\[ f \in \mathcal{O}(n^3) \Rightarrow f \in \mathcal{O}(n^6) \]

\[ f \in \mathcal{O}\left(\frac{1}{n}\right) \Rightarrow f \in \mathcal{O}\left(\frac{1}{n^2}\right) \]

so \( g = \frac{1}{n} \) gives the counterexample \( \mathcal{F} : \{ f : \mathbb{N} \to \mathbb{R}^{\geq 0}\} \)

more precisely \( g = \frac{1}{n + 1} \) \# so that \( n \) can be 0
now we want to show

\[ \frac{1}{n+1} \notin \mathcal{O}\left( \frac{1}{(n+1)^2}\right) \]

which by definition is

\[ \forall c \in \mathbb{R}^+, \forall B \in \mathbb{N}, \exists n \in \mathbb{N}, \ n \geq B \land \frac{1}{n+1} > \frac{c}{(n+1)^2} \]

pick \( n \) wisely
Disproof: \( \forall f, g \in \mathcal{F}, f \in \mathcal{O}(g) \Rightarrow f \in \mathcal{O}(g \cdot g) \)
Summary of Chapter 4

→ **definition** of big-Oh, big-Omega

→ big-Oh proofs for **polynomials** (standard procedure with over/underestimates)

→ big-Oh proofs for **non-polynomials** (need to use limits and L’Hopital’s rule)

→ proofs for **general** big-Oh statements (pick $B$ and $c$ based on known $B$’s and $c$’s)
Chapter 5
Introduction to computability
computers solve problems using algorithms, in a systematic, repeatable way

however, there are some problems that are not easy to solve using an algorithm

what questions do you think an algorithm cannot answer?

Some questions really look like easy for computers to answer, but not really.
a python function $f(n)$ that may or may not halt

```python
def f(n):
    if n % 2 == 0:
        while True:
            pass
    else:
        return
```

Now devise an algorithm $\text{halt}(f, n)$ that predicts whether this function $f$ with input $i$ eventually halts, i.e., will it ever stop?

```python
def halt(f, n):
    '''return True iff $f(n)$ halts'''
    
    ????
```
another function f(n) that may or may not halt

```python
def f(n):
    while n > 1:
        if n % 2 == 0:
            n = n / 2
        else:
            n = 3n + 1
    return "i is 1"
```
what we are sure about

It is impossible to write one \texttt{halt(f, n)} that works for all functions \texttt{f}.

\begin{verbatim}
def halt(f, n):
    """return True if f(n) halts, return false otherwise""
    ...
\end{verbatim}
a naive thought of writing \( \text{halt}(f, n) \)

Why don’t we just implement \( \text{halt}(f, n) \) by calling \( f(n) \) and see if it halts?
Proof: nobody can write $\text{halt}(f, n)$

thoughts:
suppose we could write it, ...
Proof: nobody can write $\text{halt}(f, n)$

```python
def clever(f):
    while halt(f, f):
        pass
    return 42
```

Now consider:
```
clever(clever)
```

Does it halt?
**terminology**

A function $f$ is **well-defined** iff we can tell **what** $f(x)$ is for every $x$ in some domain.

A function $f$ is **computable** iff it is well-defined and we can tell **how** to compute $f(x)$ for every $x$ in the domain.

Otherwise, $f(x)$ is **non-computable**.

$\text{halt}(f, n)$ is well-defined and uncomputable.
what we learn to do in CSC165

Given any function, decide whether it is computable not not.

how we do it

use reductions
Reductions

If function $f(n)$ can be implemented by extending another function $g(n)$, then we say $f$ reduces to $g$

```python
def f(n):
    return g(2n)
```

g computable        f computable

f non-computable    g non-computable
f reduces to g

\[ g \text{ computable} \implies f \text{ computable} \]
\[ f \text{ non-computable} \implies g \text{ non-computable} \]

To prove a function **computable**

\[ \rightarrow \text{ show that this function reduces to a function } g \text{ that is computable} \]

To prove a function **non-computable**

\[ \rightarrow \text{ show that this function can be reduced from a function } f \text{ that is non-computable} \]
def initialized(f, v):
    '''return whether variable v is guaranteed to be initialized before its first use in f'''
    ...
    return True/False

def f1(x):
    return x + 1
print y

initialized(f1, y)

def f2(x):
    return x + y + 1

initialized(f2, y)
def initialized(f, v):
    '''return whether variable v is guaranteed to be initialized before its first use in f'''
    ...
    return True/False

now prove: `initialized(f, v)` is non-computable
all we need to show: \ textrm{\textbf{halt}}(f, n) \textbf{reduces to} \textrm{\textbf{initialized}}(f, v)

in other words, implement \textrm{\textbf{halt}}(f, n) using \textrm{\textbf{initialized}}(f, v)

\begin{verbatim}
def halt(f, n):
    def initialized(g, v):
        ...implementation of initialized...

        # code that scan code of f, and figure out
        # a variable name that doesn’t occur in f
        # then store it in as variable v

    def f_prime(x):
        # ignore arg x, call f with fixed arg n
        # (the one passed in to halt)
        f(n)
        print(v)

    return not initialized(f_prime, v)
\end{verbatim}

If \( f(n) \) \textbf{halts},
then,

If \( f(n) \) \textbf{does not halt},
then,
summary

→ Fact: \texttt{halt(f, v)} is non-computable

→ use reductions to prove other functions being non-computable