Foundations of Computer Science
Lecture 7

Recursion

Powerful but Dangerous
Recursion and Induction
Recursive Sets and Structures

Last Time

- With induction, it may be easier to prove a stronger claim.
- Leaping induction.
  - \( n^3 < 2^n \) for \( n \geq 10 \).
  - Postage.
- Strong induction.
  - Representation theorems: FTA, binary expansion.
  - Games: Nim with 2 equal piles.

Today: Recursion

- Recursive functions
  - Analysis using induction
  - Recurrences
  - Recursive programs

- Recursive sets
  - Formal Definition of \( \mathbb{N} \)
  - The Finite Binary Strings \( \Sigma^* \)

- Recursive structures
  - Rooted binary trees (RBT)

A Fantastic Recursion

Online lecture tool “Demo”: allows lecturer to see screen of remote student.

Online lecture tool “Demo”

HANG!, CRASH!, BANG!, reboot required

Creator: Malik Magdon-Ismail
Recursion: 4/16
Examples of Recursion: Self Reference

The tool shows the student’s screen, i.e my previous screen, which is what the tool showed, 

The tool shows what the tool showed. 

— self reference

**look-up (word):** Get definition, if a word \( x \) in the definition is unknown, **look-up (x).**

\[
f(n) = f(n-1) + 2n - 1.
\]

What is \( f(2) \)?

\[
f(2) = f(1) + 3 = f(0) + 4 = f(-1) + 3 = \cdots
\]

Recursion and Induction

\[
f(n) = \begin{cases} 
0 & n \leq 0; \\
 f(n-1) + 2n - 1 & n > 0.
\end{cases}
\]

**Induction**

\( P(0) \) is T; \( P(n) \Rightarrow P(n+1) \)

(you can conclude \( P(n+1) \) if \( P(n) \) is T)

\[ P(0) \Rightarrow P(1) \Rightarrow P(2) \Rightarrow P(3) \Rightarrow P(4) \Rightarrow \cdots \]

\( P(n) \) is T for all \( n \geq 0 \).

**Recursion**

\[
f(0) = 0; \ f(n+1) = f(n) + 2n + 1
\]

(we can compute \( f(n+1) \) if \( f(n) \) is known)

\[ f(0) \Rightarrow f(1) \Rightarrow f(2) \Rightarrow f(3) \Rightarrow f(4) \Rightarrow \cdots \]

We can compute \( f(n) \) for all \( n \geq 0 \).

**Example: More Base Cases**

\[
f(n) = \begin{cases} 
1 & n = 0; \\
 f(n-2) + 2 & n > 0.
\end{cases}
\]

How to fix \( f(n) \)? Hint: leaping induction.

\[ f(0) \Rightarrow f(1) \Rightarrow f(3) \Rightarrow f(5) \Rightarrow f(7) \Rightarrow f(8) \Rightarrow \cdots \]

Recursion Must Have Base Cases: **Partial Self Reference**

look-up (word) works if there are some known words to which everything reduces.

Similarly with recursive functions,

\[
f(n) = \begin{cases} 
0 & n \leq 0; \\
 f(n-1) + 2n - 1 & n > 0.
\end{cases}
\]

\[ f(2) = f(1) + 3 = f(0) + 4 = 0 + 4 = 4. \]

(ends at a base case)

Must have **base cases:**

In this case \( f(0) \).

Must make **recursive progress:**

To compute \( f(n) \) you must move closer to the base case \( f(0) \).

Using Induction to Analyze a Recursion

\[
f(n) = \begin{cases} 
0 & n \leq 0; \\
 f(n-1) + 2n - 1 & n > 0.
\end{cases}
\]

\[ f(n) = 1 + \lceil \log_2 n \rceil. \]

**Proof by induction that** \( f(n) = n^2 \).

\[
P(0): f(0) = 0^2 \]

[Base case] \( P(0) : f(0) = 0^2 \) (clearly T).

[Induction] Show \( P(n) \Rightarrow P(n+1) \) for \( n \geq 0 \).

Assume \( P(n) : f(n) = n^2 \).

\[
f(n+1) = f(n) + 2(n+1) - 1 = n^2 + 2n + 1 \quad (f(n) = n^2)
\]

\[ = (n+1)^2 \quad (P(n+1) \text{ is T}) \]

So, \( P(n+1) \) is T.

**Hard Example: A halving recursion** (see text)

\[
f(n) = \begin{cases} 
1 & n = 1; \\
 f(\lceil \log_2 n \rceil + 1) & n > 1, \text{ even}; \\
 f(n+1) & n > 1, \text{ odd}.
\end{cases}
\]

(Looks esoteric? Often, you halve a problem (if it is even) or pad it by one to make it even, and then halve it.)

Prove \( f(n) = 1 + \lceil \log_2 n \rceil. \)

**Practice. Exercise 7.4**
Checklist for Analyzing Recursion

- Tinker. Draw the implication arrows. Is the function well defined?
- Tinker. Compute \( f(n) \) for small values of \( n \).
- Make a guess for \( f(n) \). "Unfolding" the recursion can be helpful here.
- Prove your conjecture for \( f(n) \) by induction.
  - The type of induction to use will often be related to the type of recursion.
  - In the induction step, use the recursion to relate the claim for \( n+1 \) to lower values.

Practice. Exercise 7.6

Recursive Programs

Proving correctness: let’s prove \( \text{Big}(n) = 2^n \) for \( n \geq 1 \)

**Induction.**

When \( n = 0 \), \( \text{Big}(0) = 1 = 2^0 \) ✓
Assume \( \text{Big}(n) = 2^n \) for \( n \geq 0 \)

\[
\text{Big}(n + 1) = 2 \times \text{Big}(n) = 2 \times 2^n = 2^{n+1}.
\]

What is the runtime?

Let \( T_n = \) runtime of \( \text{Big} \) for input \( n \).

\[
T_0 = 2 \\
T_n = T_{n-1} + (\text{check } n=0) + \text{(multiply by 2)} + \text{(assign to } \text{out}) \\
= T_{n-1} + 3
\]

Exercise. Prove by induction that \( T_n = 3n + 2 \)

Recursive Sets: \( \mathbb{N} \)

**Recursive definition of the natural numbers \( \mathbb{N} \).**

- \( 1 \in \mathbb{N} \). [basis]
- \( x \in \mathbb{N} \rightarrow x + 1 \in \mathbb{N} \). [constructor]
- Nothing else is in \( \mathbb{N} \). [minimality]

\[
\mathbb{N} = \{1, 2, 3, 4, \ldots \}
\]

Technically, by bullet 3, we mean that \( \mathbb{N} \) is the smallest set satisfying bullets 1 and 2.

**Pop Quiz.** Is \( \mathbb{R} \) a set that satisfies bullets 1 and 2 alone? Is it the smallest?
Recursive Sets: Finite Binary Strings, $\Sigma^*$

Let $\varepsilon$ be the empty string (similar to the empty set).

**Recursive definition of $\Sigma^*$ (finite binary strings).**
- $\varepsilon \in \Sigma^*$. [basis]
- $x \in \Sigma^* \rightarrow x \cdot 0 \in \Sigma^*$ and $x \cdot 1 \in \Sigma^*$. [constructor]

Minimality is there by default: nothing else is in $\Sigma^*$.

$$\varepsilon \rightarrow 0, 1 \rightarrow 00, 01, 10, 11 \rightarrow 000, 001, 010, 011, 100, 101, 110, 111 \rightarrow \cdots$$

$\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, \ldots\}$

*Practice. Exercise 7.12*

---

**Trees Are Important: Food for Thought**

Do we know the right structure is not a tree? Are we sure it can’t be derived?

Is there only one way to derive a tree?

Trees are more general than just RBT and have many interesting properties.
- A tree is a connected graph with $n$ nodes and $n - 1$ edges.
- A tree is a connected graph with no cycles.
- A tree is a graph in which any two nodes are connected by exactly one path.

Can we be sure every RBT has these properties?