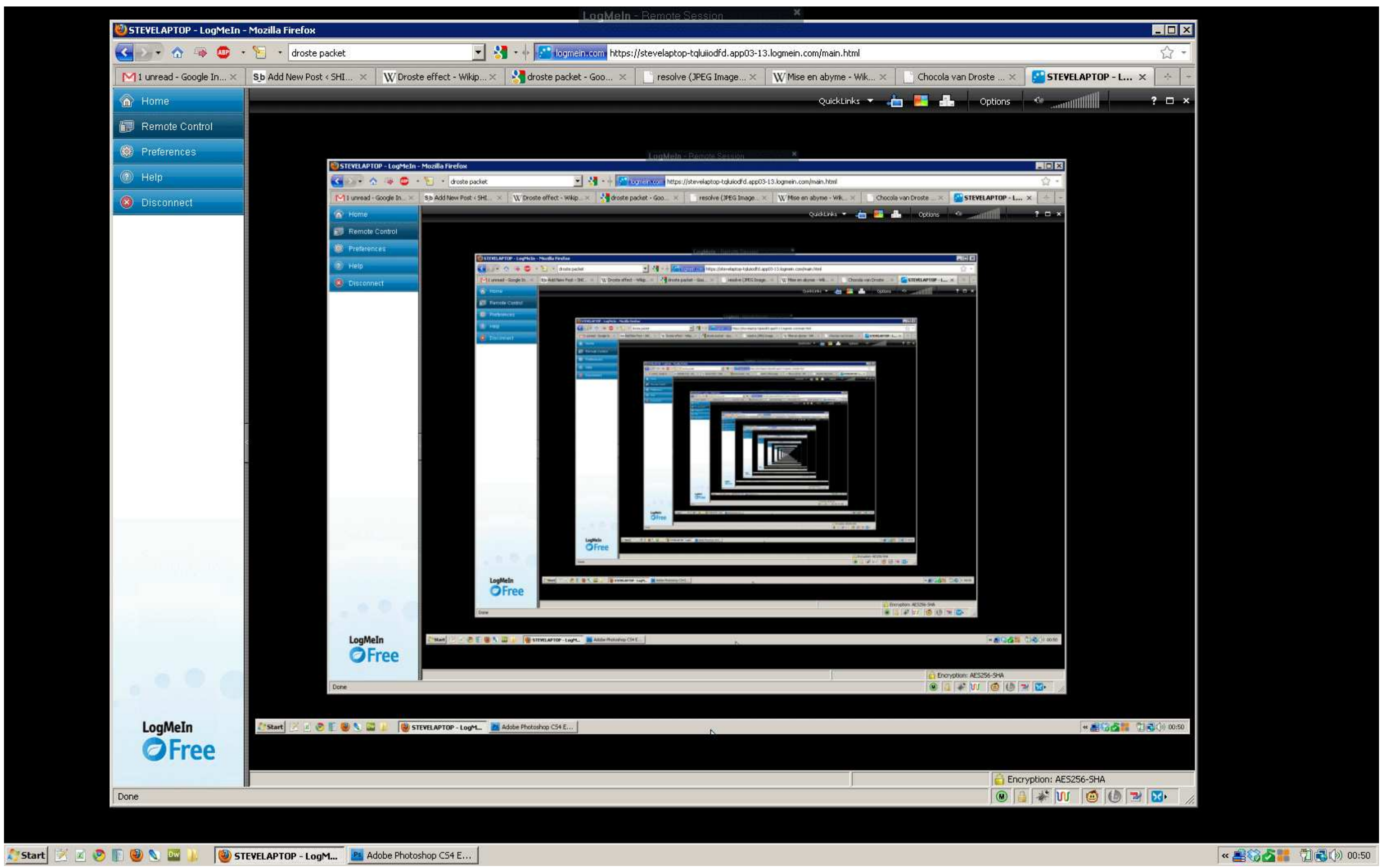
Foundations of Computer Science
Lecture 7

Recursion

Powerful but Dangerous
Recursion and Induction
Recursive Sets and Structures



Last Time

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

Today: Recursion

1. Recursive functions
   - Analysis using induction
   - Recurrences
   - Recursive programs
2. Recursive sets
   - Formal Definition of $\mathbb{N}$
   - The Finite Binary Strings Σ^*
3. Recursive structures
   - Rooted binary trees (RBT)

A Fantastic Recursion

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

A Fantastic Recursion

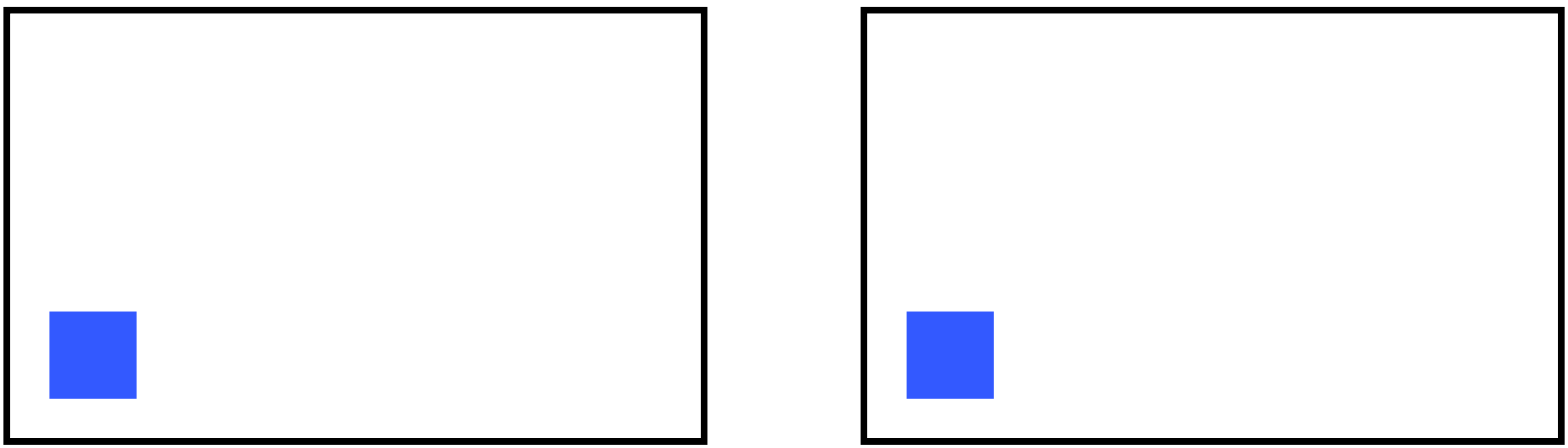
Online lecture tool "Demo": allows lecturer to see screen of remote student.

PROFESSOR

A Fantastic Recursion

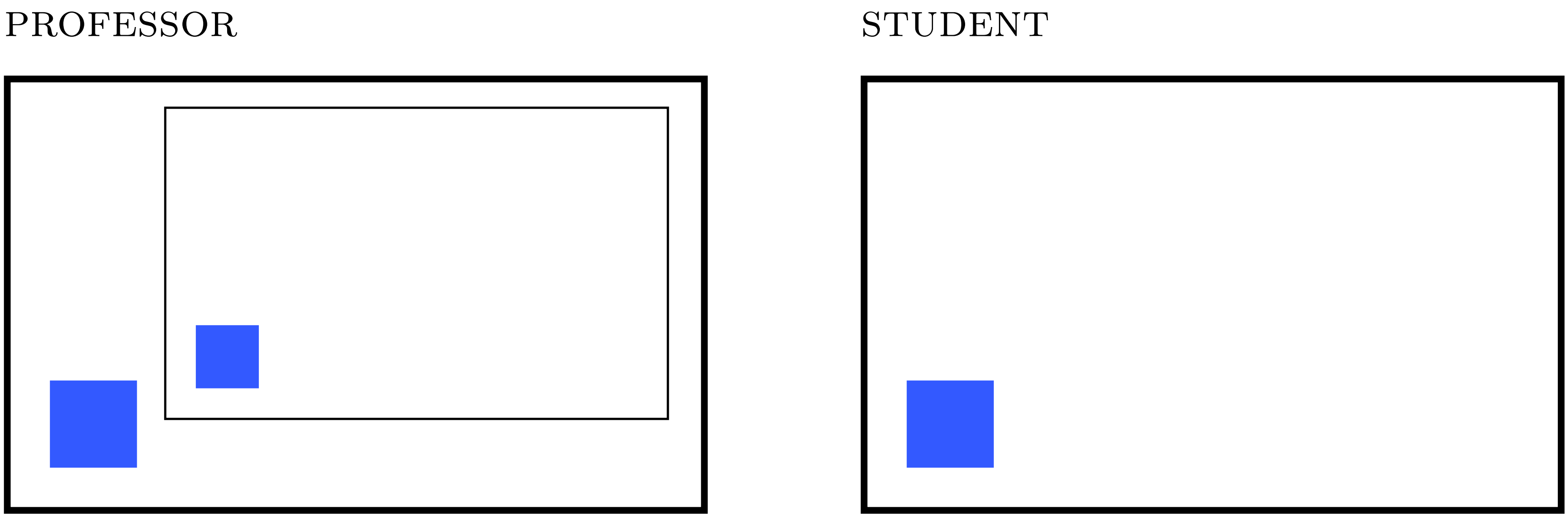
Online lecture tool “Demo”: allows lecturer to see screen of remote student.

PROFESSOR STUDENT



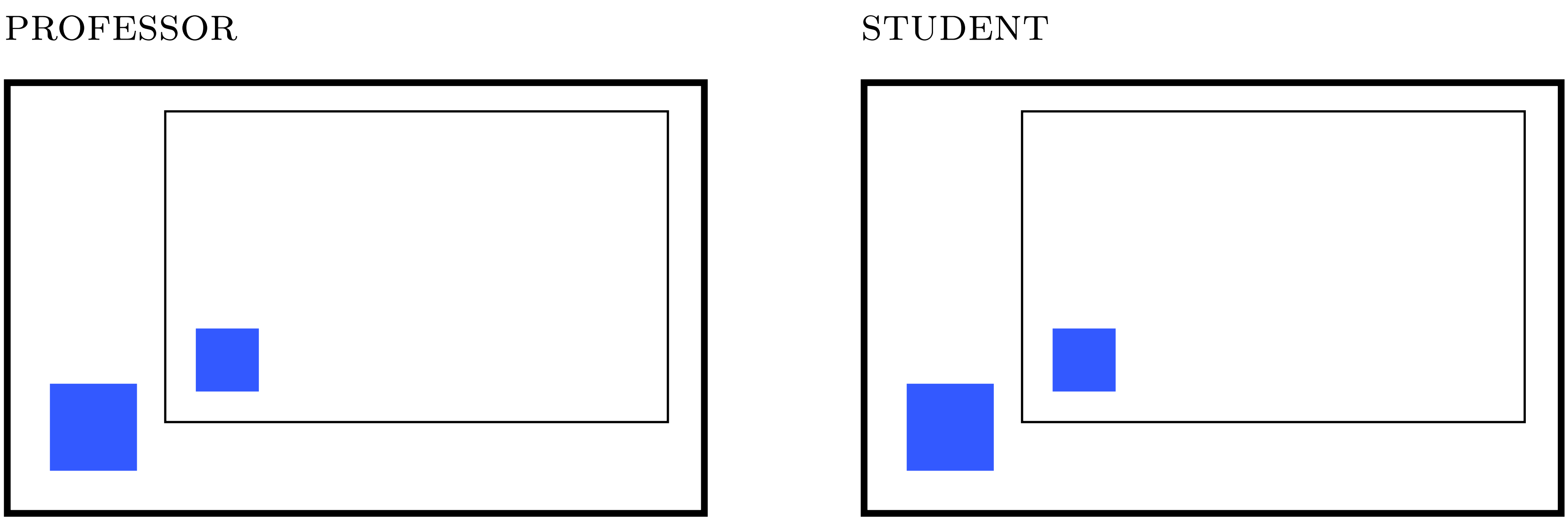
A Fantastic Recursion

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



A Fantastic Recursion

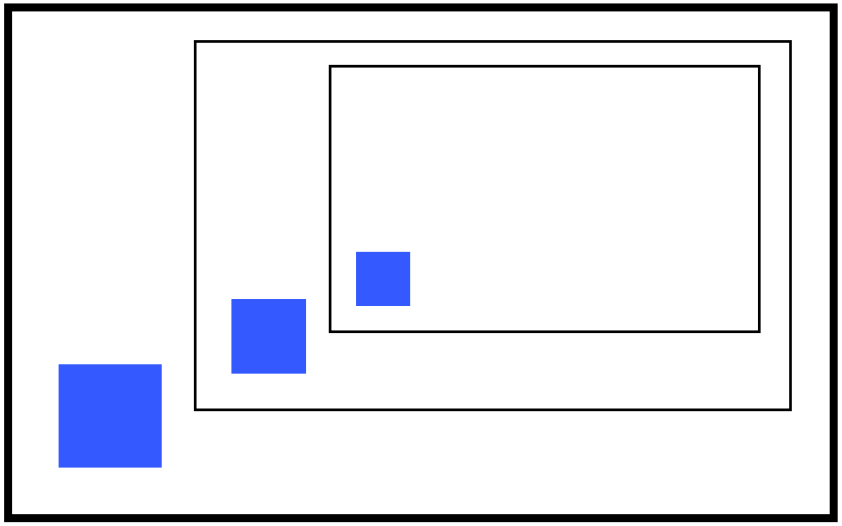
Online lecture tool "Demo": allows lecturer to see screen of remote student.



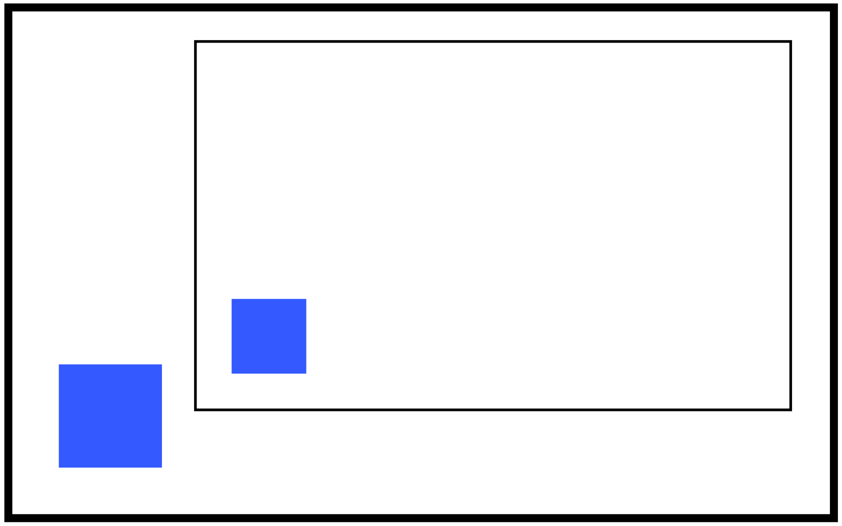
A Fantastic Recursion

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

PROFESSOR

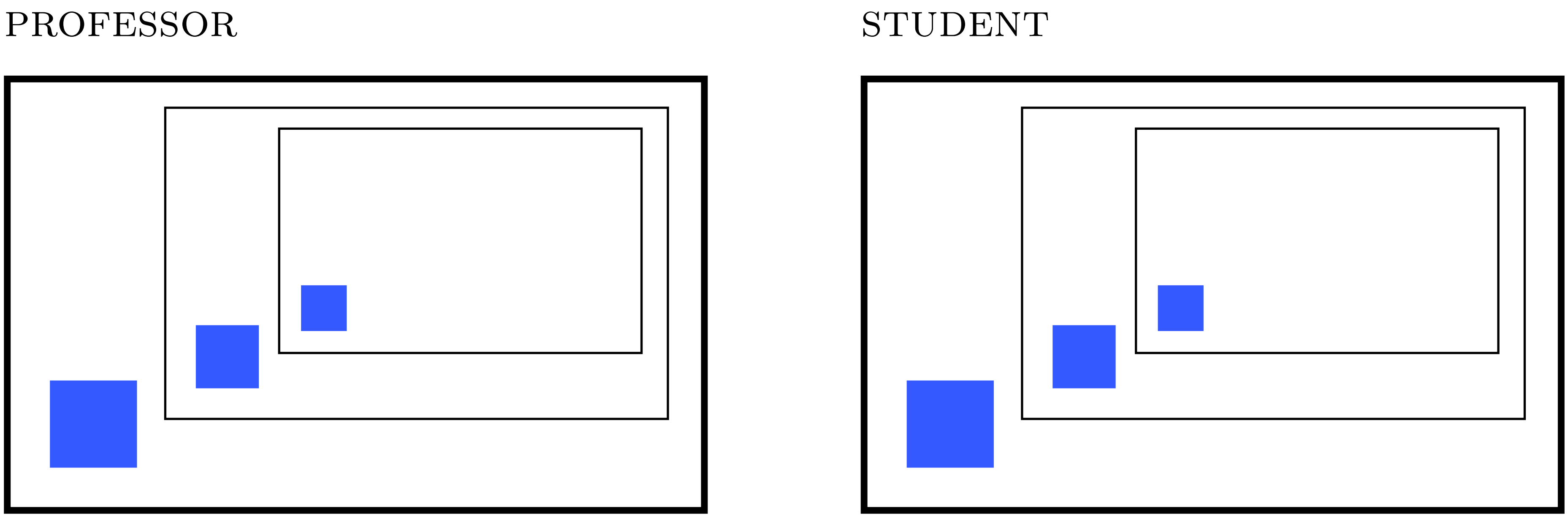


STUDENT



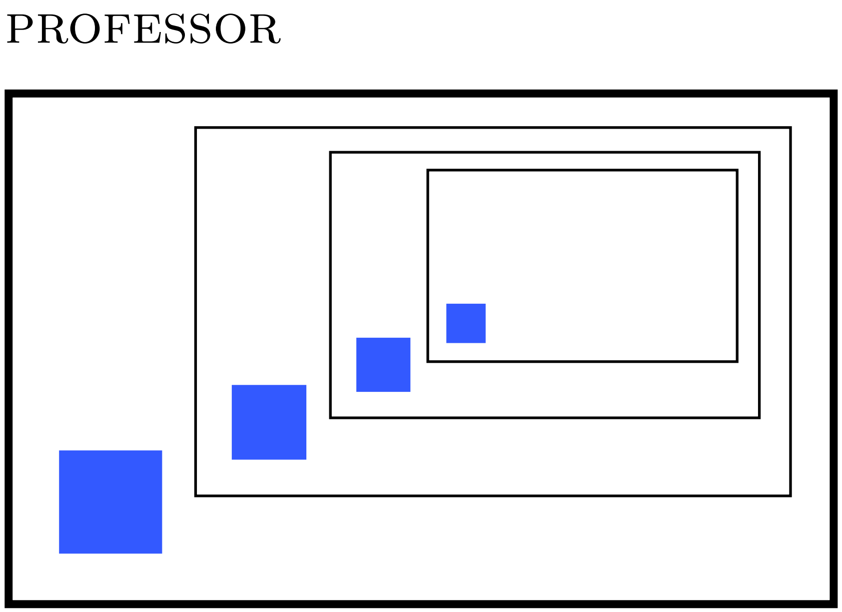
A Fantastic Recursion

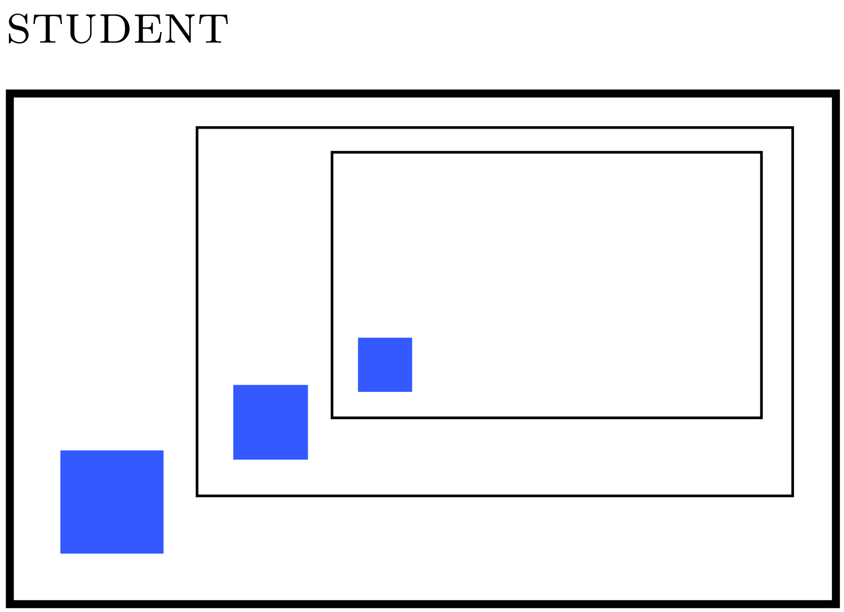
Online lecture tool "Demo": allows lecturer to see screen of remote student.



A Fantastic Recursion

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

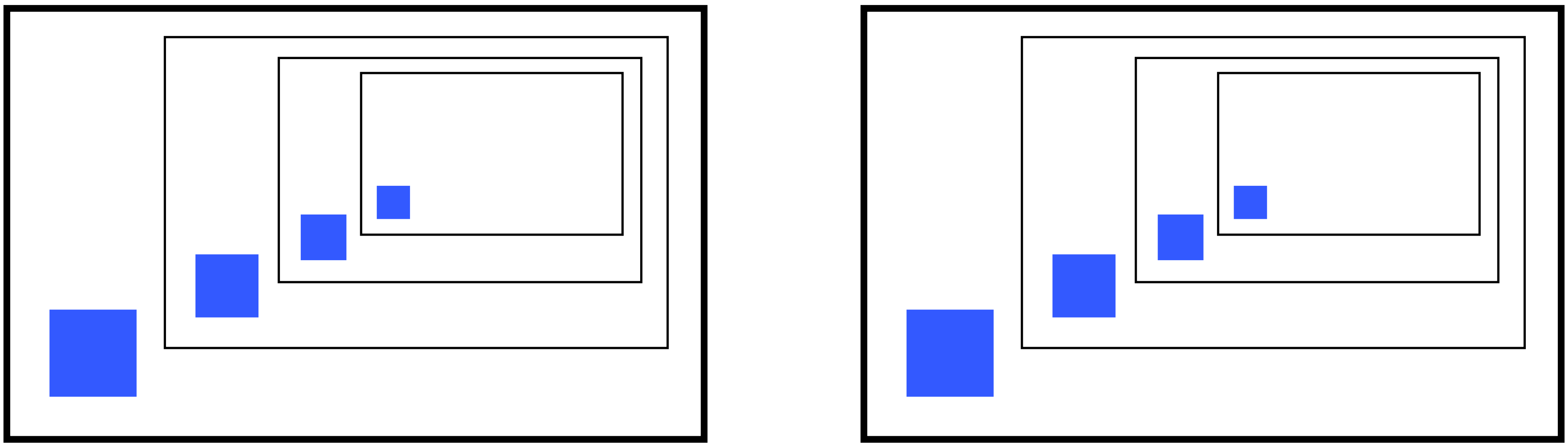




A Fantastic Recursion

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

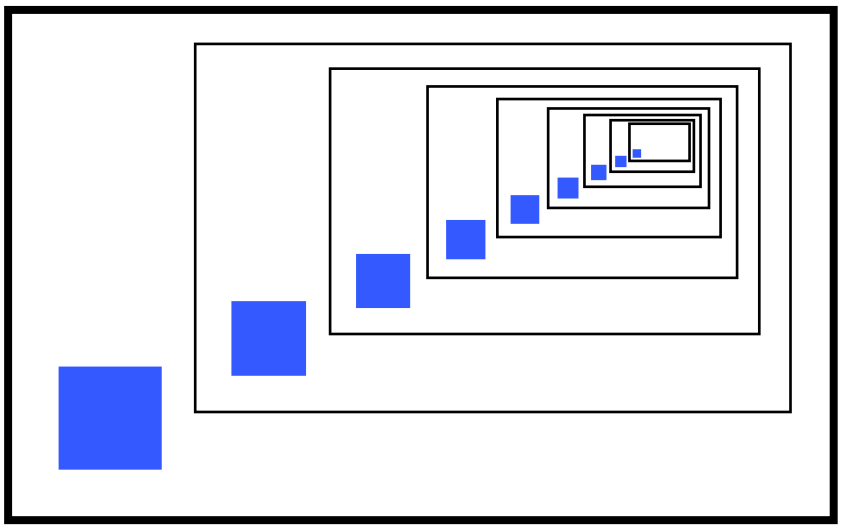
PROFESSOR STUDENT

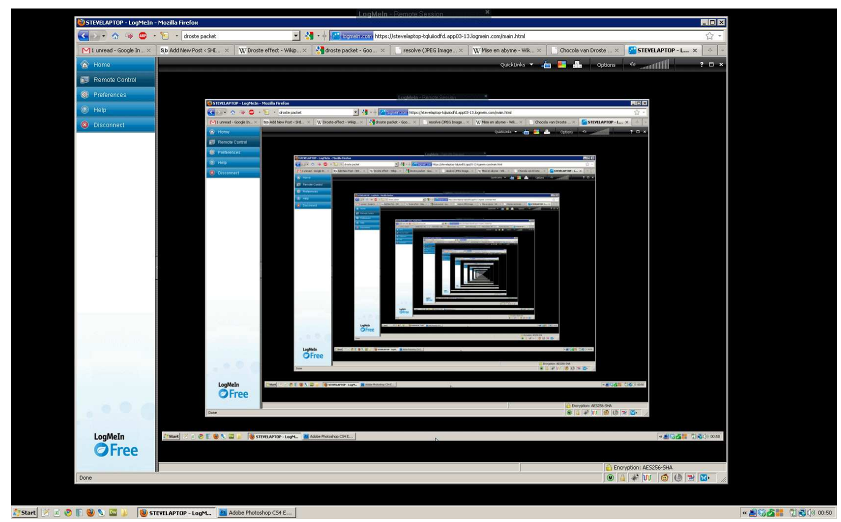


A Fantastic Recursion

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

PROFESSOR





HANG!, CRASH!, BANG!, reboot required */?%&# ☹@$#!

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*

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).

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 $f(2)$?

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 $f(2)$?

$$f(2) = f(1) + 3$$

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 $f(2)$?

$$f(2) = f(1) + 3 = f(0) + 4$$

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 $f(2)$?

$$f(2) = f(1) + 3 = f(0) + 4 = f(-1) + 3$$

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 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,

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

$f(2) = f(1) + 3$

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,

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

$f(2) = f(1) + 3 = f(0) + 4$

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)

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 $f(0)$.

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 $f(0)$.

Recursion and Induction

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

$\boxed{\text{f(0)}}$

Recursion and Induction

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

$\boxed{\text{f(0)}} \rightarrow f(1)$

Recursion and Induction

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

$\boxed{\text{f(0)}} \to f(1) \to f(2)$

Recursion and Induction

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

$$\boxed{\text{f(0)}} \to f(1) \to f(2) \to f(3) \to f(4) \to \cdots$$

Recursion and Induction

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

$\boxed{\text{f}(0)} \to f(1) \to f(2) \to f(3) \to f(4) \to \cdots$

Induction

$P(0)$ is T; $P(n) \to P(n+1)$

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

$$\boxed{P(0)} \to P(1) \to P(2) \to P(3) \to P(4) \to \cdots$$

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

Recursion

Recursion and Induction

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

$\boxed{\text{f(0)}} \to f(1) \to f(2) \to f(3) \to f(4) \to \cdots$

Induction

$P(0)$ is T; $P(n) \to P(n+1)$

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

$\boxed{P(0)} \to P(1) \to P(2) \to P(3) \to P(4) \to \cdots$

$P(n)$ is T for all $n \ge 0$.

Recursion

$f(0) = 0$; $f(\boldsymbol{n+1}) = f(n) + 2n + 1$

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

$\boxed{f(0)} \to f(1) \to f(2) \to f(3) \to f(4) \to \cdots$

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

Recursion and Induction

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

$\boxed{\text{f(0)}} \to f(1) \to f(2) \to f(3) \to f(4) \to \cdots$

Induction

$P(0)$ is T; $P(n) \to P(n+1)$

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

$\boxed{P(0)} \to P(1) \to P(2) \to P(3) \to P(4) \to \cdots$

$P(n)$ is T for all $n \ge 0$.

Recursion

$f(0) = 0$; $f(\boldsymbol{n+1}) = f(n) + 2n + 1$

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

$\boxed{f(0)} \to f(1) \to f(2) \to f(3) \to f(4) \to \cdots$

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

Example: More Base Cases

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

n	0	1	2	3	4	5	6	7	8
$f(n)$	1	✘							

Recursion and Induction

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

$\boxed{\text{f(0)}} \to f(1) \to f(2) \to f(3) \to f(4) \to \cdots$

Induction

$P(0)$ is T; $P(n) \to P(n+1)$

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

$\boxed{P(0)} \to P(1) \to P(2) \to P(3) \to P(4) \to \cdots$

$P(n)$ is T for all $n \ge 0$.

Recursion

$f(0) = 0$; $f(\mathbf{n+1}) = f(n) + 2n + 1$

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

$\boxed{f(0)} \to f(1) \to f(2) \to f(3) \to f(4) \to \cdots$

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

Example: More Base Cases

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

n	0	1	2	3	4	5	6	7	8
$f(n)$	1	✘	3	✘	5	✘	7	✘	9

Recursion and Induction

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

$\boxed{\text{f(0)}} \to f(1) \to f(2) \to f(3) \to f(4) \to \cdots$

Induction

$P(0)$ is T; $P(n) \to P(n+1)$

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

$\boxed{P(0)} \to P(1) \to P(2) \to P(3) \to P(4) \to \cdots$

$P(n)$ is T for all $n \ge 0$.

Recursion

$f(0) = 0$; $f(\boldsymbol{n+1}) = f(n) + 2n + 1$

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

$\boxed{f(0)} \to f(1) \to f(2) \to f(3) \to f(4) \to \cdots$

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

Example: More Base Cases

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

n	0	1	2	3	4	5	6	7	8
$f(n)$	1	✘	3	✘	5	✘	7	✘	9

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

$\boxed{\boldsymbol{f(0)}}$ $f(1)$ $f(2)$ $f(3)$ $f(4)$ $f(5)$ $f(6)$ $f(7)$ $f(8)$ $\cdots$

Practice. Exercise 7.4

Using Induction to Analyze a Recursion

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

Using Induction to Analyze a Recursion

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

n	0	1	2	3	4	5	6	7	8	$\cdots$
$f(n)$	0	1	4	9	16	25	36	49	64	$\cdots$

Using Induction to Analyze a Recursion

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

n	0	1	2	3	4	5	6	7	8	$\cdots$
$f(n)$	0	1	4	9	16	25	36	49	64	$\cdots$

Unfolding the Recursion

$$f(n) = f(n-1) + \; 2n - 1$$

Using Induction to Analyze a Recursion

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

n	0	1	2	3	4	5	6	7	8	$\cdots$
$f(n)$	0	1	4	9	16	25	36	49	64	$\cdots$

Unfolding the Recursion

$$\begin{aligned} f(n) &= f(n-1) + \ 2n - 1 \\ f(n-1) &= f(n-2) + \ 2n - 3 \end{aligned}$$

Using Induction to Analyze a Recursion

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

n	0	1	2	3	4	5	6	7	8	$\cdots$
$f(n)$	0	1	4	9	16	25	36	49	64	$\cdots$

Unfolding the Recursion

$$\begin{aligned} f(n) &= f(n-1) + \ 2n-1 \\ f(n-1) &= f(n-2) + \ 2n-3 \\ f(n-2) &= f(n-3) + \ 2n-5 \end{aligned}$$

Using Induction to Analyze a Recursion

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

n	0	1	2	3	4	5	6	7	8	$\cdots$
$f(n)$	0	1	4	9	16	25	36	49	64	$\cdots$

Unfolding the Recursion

$$\begin{aligned} f(n) &= f(n-1) + 2n-1 \\ f(n-1) &= f(n-2) + 2n-3 \\ f(n-2) &= f(n-3) + 2n-5 \\ &\vdots \\ f(2) &= f(1) + 3 \\ f(1) &= f(0) + 1 \end{aligned}$$

Using Induction to Analyze a Recursion

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

n	0	1	2	3	4	5	6	7	8	$\cdots$
$f(n)$	0	1	4	9	16	25	36	49	64	$\cdots$

Unfolding the Recursion

$$\begin{aligned}
f(n) &= f(n-1) + 2n-1 \\
f(n-1) &= f(n-2) + 2n-3 \\
f(n-2) &= f(n-3) + 2n-5 \\
&\vdots \\
f(2) &= f(1) + 3 \\
f(1) &= f(0) + 1
\end{aligned}$$

$+$

Using Induction to Analyze a Recursion

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

n	0	1	2	3	4	5	6	7	8	$\cdots$
$f(n)$	0	1	4	9	16	25	36	49	64	$\cdots$

Unfolding the Recursion

$$\begin{aligned}
f(n) &= \cancel{f(n-1)} + 2n - 1 \\
\cancel{f(n-1)} &= \cancel{f(n-2)} + 2n - 3 \\
\cancel{f(n-2)} &= \cancel{f(n-3)} + 2n - 5 \\
&\vdots \\
\cancel{f(2)} &= \cancel{f(1)} + 3 \\
\cancel{f(1)} &= \cancel{f(0)}^{\,0} + 1 \\
\hline
+ &
\end{aligned}$$

Using Induction to Analyze a Recursion

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

n	0	1	2	3	4	5	6	7	8	$\cdots$
$f(n)$	0	1	4	9	16	25	36	49	64	$\cdots$

Unfolding the Recursion

$$\begin{aligned} f(n) &= \cancel{f(n-1)} + 2n - 1 \\ \cancel{f(n-1)} &= \cancel{f(n-2)} + 2n - 3 \\ \cancel{f(n-2)} &= \cancel{f(n-3)} + 2n - 5 \\ &\vdots \\ \cancel{f(2)} &= \cancel{f(1)} + 3 \\ \cancel{f(1)} &= \cancel{f(0)}^{\,0} + 1 \\ \hline + \quad f(n) &= 1 + 3 + \cdots + 2n - 1 \end{aligned}$$

Using Induction to Analyze a Recursion

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

n	0	1	2	3	4	5	6	7	8	$\cdots$
$f(n)$	0	1	4	9	16	25	36	49	64	$\cdots$

Unfolding the Recursion

$$\begin{aligned} f(n) &= \cancel{f(n-1)} + 2n - 1 \\ \cancel{f(n-1)} &= \cancel{f(n-2)} + 2n - 3 \\ \cancel{f(n-2)} &= \cancel{f(n-3)} + 2n - 5 \\ &\vdots \\ \cancel{f(2)} &= \cancel{f(1)} + 3 \\ \cancel{f(1)} &= \cancelto{0}{f(0)} + 1 \\ \hline + \quad f(n) &= 1 + 3 + \cdots + 2n - 1 \end{aligned}$$

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

$P(n) : f(n) = n^2$

Using Induction to Analyze a Recursion

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

n	0	1	2	3	4	5	6	7	8	$\cdots$
$f(n)$	0	1	4	9	16	25	36	49	64	$\cdots$

Unfolding the Recursion

$$\begin{aligned} f(n) &= \cancel{f(n-1)} + 2n-1 \\ \cancel{f(n-1)} &= \cancel{f(n-2)} + 2n-3 \\ \cancel{f(n-2)} &= \cancel{f(n-3)} + 2n-5 \\ &\vdots \\ \cancel{f(2)} &= \cancel{f(1)} + 3 \\ \cancel{f(1)} &= \cancel{f(0)}^{\,0} + 1 \\ \hline + \quad f(n) &= 1 + 3 + \cdots + 2n-1 \end{aligned}$$

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

$P(n) : f(n) = n^2$

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

Using Induction to Analyze a Recursion

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

n	0	1	2	3	4	5	6	7	8	$\cdots$
$f(n)$	0	1	4	9	16	25	36	49	64	$\cdots$

Unfolding the Recursion

$$\begin{aligned}
f(n) &= \cancel{f(n-1)} + 2n-1 \\
\cancel{f(n-1)} &= \cancel{f(n-2)} + 2n-3 \\
\cancel{f(n-2)} &= \cancel{f(n-3)} + 2n-5 \\
&\vdots \\
\cancel{f(2)} &= \cancel{f(1)} + 3 \\
\cancel{f(1)} &= \cancel{f(0)}^{\,0} + 1 \\
\hline
+ \quad f(n) &= 1 + 3 + \cdots + 2n - 1
\end{aligned}$$

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

$P(n) : f(n) = n^2$

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

[**Induction**] Show $P(n) \to P(n+1)$ for $n \ge 0$.

Using Induction to Analyze a Recursion

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

n	0	1	2	3	4	5	6	7	8	$\cdots$
$f(n)$	0	1	4	9	16	25	36	49	64	$\cdots$

Unfolding the Recursion

$$\begin{aligned}
f(n) &= \cancel{f(n-1)} + 2n - 1 \\
\cancel{f(n-1)} &= \cancel{f(n-2)} + 2n - 3 \\
\cancel{f(n-2)} &= \cancel{f(n-3)} + 2n - 5 \\
&\vdots \\
\cancel{f(2)} &= \cancel{f(1)} + 3 \\
\cancel{f(1)} &= \cancel{f(0)}^{\,0} + 1 \\
\hline
+\quad f(n) &= 1 + 3 + \cdots + 2n - 1
\end{aligned}$$

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

$P(n) : f(n) = n^2$

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

[Induction] Show $P(n) \to P(n+1)$ for $n \ge 0$.

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

Using Induction to Analyze a Recursion

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

n	0	1	2	3	4	5	6	7	8	$\cdots$
$f(n)$	0	1	4	9	16	25	36	49	64	$\cdots$

Unfolding the Recursion

$$\begin{aligned} f(n) &= \cancel{f(n-1)} + 2n - 1 \\ \cancel{f(n-1)} &= \cancel{f(n-2)} + 2n - 3 \\ \cancel{f(n-2)} &= \cancel{f(n-3)} + 2n - 5 \\ &\vdots \\ \cancel{f(2)} &= \cancel{f(1)} + 3 \\ \cancel{f(1)} &= \cancel{f(0)}^{\,0} + 1 \\ \hline + \quad f(n) &= 1 + 3 + \cdots + 2n - 1 \end{aligned}$$

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

$P(n) : f(n) = n^2$

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

[Induction] Show $P(n) \to P(n+1)$ for $n \ge 0$.

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

$f(n+1)$

Using Induction to Analyze a Recursion

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

n	0	1	2	3	4	5	6	7	8	$\cdots$
$f(n)$	0	1	4	9	16	25	36	49	64	$\cdots$

Unfolding the Recursion

$$\begin{aligned}
f(n) &= \cancel{f(n-1)} + 2n-1 \\
\cancel{f(n-1)} &= \cancel{f(n-2)} + 2n-3 \\
\cancel{f(n-2)} &= \cancel{f(n-3)} + 2n-5 \\
&\vdots \\
\cancel{f(2)} &= \cancel{f(1)} + 3 \\
\cancel{f(1)} &= \cancel{f(0)}^{\,0} + 1 \\
\hline
+\quad f(n) &= 1 + 3 + \cdots + 2n-1
\end{aligned}$$

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

$P(n) : f(n) = n^2$

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

[Induction] Show $P(n) \to P(n+1)$ for $n \ge 0$.

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

$$f(n+1) = f(n) + 2(n+1) - 1 \qquad \text{(recursion)}$$

Using Induction to Analyze a Recursion

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

n	0	1	2	3	4	5	6	7	8	$\cdots$
$f(n)$	0	1	4	9	16	25	36	49	64	$\cdots$

Unfolding the Recursion

$$\begin{aligned}
f(n) &= \cancel{f(n-1)} + 2n-1 \\
\cancel{f(n-1)} &= \cancel{f(n-2)} + 2n-3 \\
\cancel{f(n-2)} &= \cancel{f(n-3)} + 2n-5 \\
&\vdots \\
\cancel{f(2)} &= \cancel{f(1)} + 3 \\
\cancel{f(1)} &= \cancel{f(0)}^{\,0} + 1 \\
\hline
+ \quad f(n) &= 1 + 3 + \cdots + 2n - 1
\end{aligned}$$

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

$P(n) : f(n) = n^2$

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

[Induction] Show $P(n) \to P(n+1)$ for $n \ge 0$.

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

$$\begin{aligned}
f(n+1) &= f(n) + 2(n+1) - 1 && \text{(recursion)} \\
&= n^2 + 2n + 1 && (f(n) = n^2)
\end{aligned}$$

Using Induction to Analyze a Recursion

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

n	0	1	2	3	4	5	6	7	8	$\cdots$
$f(n)$	0	1	4	9	16	25	36	49	64	$\cdots$

Unfolding the Recursion

$$\begin{aligned} f(n) &= \cancel{f(n-1)} + 2n-1 \\ \cancel{f(n-1)} &= \cancel{f(n-2)} + 2n-3 \\ \cancel{f(n-2)} &= \cancel{f(n-3)} + 2n-5 \\ &\vdots \\ \cancel{f(2)} &= \cancel{f(1)} + 3 \\ \cancel{f(1)} &= \cancel{f(0)}^{\,0} + 1 \\ \hline + \quad f(n) &= 1 + 3 + \cdots + 2n - 1 \end{aligned}$$

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

$P(n) : f(n) = n^2$

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

[Induction] Show $P(n) \to P(n+1)$ for $n \ge 0$.

Assume $P(n)$*:* $f(n) = n^2$.

$$\begin{aligned} f(n+1) &= f(n) + 2(n+1) - 1 && \text{(recursion)} \\ &= n^2 + 2n + 1 && (f(n) = n^2) \\ &= (n+1)^2 && (P(n+1) \text{ is T}) \end{aligned}$$

Using Induction to Analyze a Recursion

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

n	0	1	2	3	4	5	6	7	8	$\cdots$
$f(n)$	0	1	4	9	16	25	36	49	64	$\cdots$

Unfolding the Recursion

$$\begin{aligned}
f(n) &= \cancel{f(n-1)} + 2n-1 \\
\cancel{f(n-1)} &= \cancel{f(n-2)} + 2n-3 \\
\cancel{f(n-2)} &= \cancel{f(n-3)} + 2n-5 \\
&\vdots \\
\cancel{f(2)} &= \cancel{f(1)} + 3 \\
\cancel{f(1)} &= \cancel{f(0)}^{\,0} + 1 \\
\hline
+ \quad f(n) &= 1 + 3 + \cdots + 2n - 1
\end{aligned}$$

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

$P(n) : f(n) = n^2$

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

[Induction] Show $P(n) \rightarrow P(n+1)$ for $n \ge 0$.

Assume $P(n)$*:* $f(n) = n^2$.

$$\begin{aligned}
f(n+1) &= f(n) + 2(n+1) - 1 && \text{(recursion)} \\
&= n^2 + 2n + 1 && (f(n) = n^2) \\
&= (n+1)^2 && (P(n+1) \text{ is T})
\end{aligned}$$

So, $P(n+1)$ is T. ■

Using Induction to Analyze a Recursion

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

n	0	1	2	3	4	5	6	7	8	$\cdots$
$f(n)$	0	1	4	9	16	25	36	49	64	$\cdots$

Unfolding the Recursion

$$\begin{aligned} f(n) &= \cancel{f(n-1)} + 2n - 1 \\ \cancel{f(n-1)} &= \cancel{f(n-2)} + 2n - 3 \\ \cancel{f(n-2)} &= \cancel{f(n-3)} + 2n - 5 \\ &\vdots \\ \cancel{f(2)} &= \cancel{f(1)} + 3 \\ \cancel{f(1)} &= \cancel{f(0)}^{\,0} + 1 \\ \hline + \quad f(n) &= 1 + 3 + \cdots + 2n - 1 \end{aligned}$$

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

$P(n) : f(n) = n^2$

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

[Induction] Show $P(n) \to P(n+1)$ for $n \ge 0$.

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

$$\begin{aligned} f(n+1) &= f(n) + 2(n+1) - 1 && \text{(recursion)} \\ &= n^2 + 2n + 1 && (f(n) = n^2) \\ &= (n+1)^2 && (P(n+1) \text{ is T}) \end{aligned}$$

So, $P(n+1)$ is T. ∎

Hard Example: A halving recursion (see text)

$$f(n) = \begin{cases} 1 & n = 1; \\ f(\frac{n}{2}) + 1 & n > 1, \text{ even}; \\ f(n+1) & n > 1, \text{ odd}; \end{cases}$$

(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 7.5

Checklist for Analyzing Recursion

- Tinker. Draw the implication arrows. Is the function well defined?

Checklist for Analyzing Recursion

- Tinker. Draw the implication arrows. Is the function well defined?
- Tinker. Compute $f(n)$ for small values of n.

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.

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.

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.

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

Recurrences: Fibonacci Numbers

Growth rate of rabbits, Sanskrit poetry, family trees of bees, ...

$$F_1 = 1;\ F_2 = 1; \qquad F_n = F_{n-1} + F_{n-2} \quad \text{for } n > 2.$$

Recurrences: Fibonacci Numbers

Growth rate of rabbits, Sanskrit poetry, family trees of bees, ...

$$F_1 = 1;\ F_2 = 1; \qquad F_n = F_{n-1} + F_{n-2} \quad \text{for } n > 2.$$

$\boldsymbol{F_1}$	$\boldsymbol{F_2}$	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	$\cdots$
1	**1**	2	3	5	8	13	21	34	55	89	144	$\cdots$

Recurrences: Fibonacci Numbers

Growth rate of rabbits, Sanskrit poetry, family trees of bees, ...

$$F_1 = 1;\ F_2 = 1; \qquad F_n = F_{n-1} + F_{n-2} \quad \text{for } n > 2.$$

$\boldsymbol{F_1}$	$\boldsymbol{F_2}$	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	$\cdots$
1	**1**	2	3	5	8	13	21	34	55	89	144	$\cdots$

Let us prove $P(n) : F_n \leq 2^n$ by **strong induction**.

Recurrences: Fibonacci Numbers

Growth rate of rabbits, Sanskrit poetry, family trees of bees, ...

$$F_1 = 1; \;\; F_2 = 1; \qquad F_n = F_{n-1} + F_{n-2} \quad \text{for } n > 2.$$

$\boldsymbol{F_1}$	$\boldsymbol{F_2}$	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	$\cdots$
1	**1**	2	3	5	8	13	21	34	55	89	144	$\cdots$

Let us prove $P(n) : F_n \le 2^n$ by **strong induction**.
Base Cases: $F_1 = 1 \le 2^1$ ✓ and $F_2 = 1 \le 2^2$ ✓ (why 2 base cases?)

Recurrences: Fibonacci Numbers

Growth rate of rabbits, Sanskrit poetry, family trees of bees, ...

$$F_1 = 1;\ F_2 = 1; \qquad F_n = F_{n-1} + F_{n-2} \quad \text{for } n > 2.$$

$\boldsymbol{F_1}$	$\boldsymbol{F_2}$	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	$\cdots$
1	**1**	2	3	5	8	13	21	34	55	89	144	$\cdots$

Let us prove $P(n) : F_n \le 2^n$ by **strong induction**.
Base Cases: $F_1 = 1 \le 2^1$ ✓ and $F_2 = 1 \le 2^2$ ✓ (why 2 base cases?)
Strong Induction: Prove $P(1) \wedge P(2) \wedge \cdots \wedge P(n) \rightarrow P(n+1)$ for $n \ge 2$.

Recurrences: Fibonacci Numbers

Growth rate of rabbits, Sanskrit poetry, family trees of bees, ...

$$F_1 = 1;\ F_2 = 1; \qquad F_n = F_{n-1} + F_{n-2} \quad \text{for } n > 2.$$

$\boldsymbol{F_1}$	$\boldsymbol{F_2}$	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	$\cdots$
1	**1**	2	3	5	8	13	21	34	55	89	144	$\cdots$

Let us prove $P(n) : F_n \le 2^n$ by **strong induction**.

Base Cases: $F_1 = 1 \le 2^1$ ✓ and $F_2 = 1 \le 2^2$ ✓ (why 2 base cases?)

Strong Induction: Prove $P(1) \wedge P(2) \wedge \cdots \wedge P(n) \rightarrow P(n+1)$ for $n \ge 2$.

Assume: $P(1) \wedge P(2) \wedge \cdots \wedge P(n)$: $F_i \le 2^i$ for $1 \le i \le n$.

Recurrences: Fibonacci Numbers

Growth rate of rabbits, Sanskrit poetry, family trees of bees, ...

$$F_1 = 1;\ F_2 = 1; \qquad F_n = F_{n-1} + F_{n-2} \quad \text{for } n > 2.$$

$\boldsymbol{F_1}$	$\boldsymbol{F_2}$	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	$\cdots$
1	**1**	2	3	5	8	13	21	34	55	89	144	$\cdots$

Let us prove $P(n) : F_n \le 2^n$ by **strong induction**.
Base Cases: $F_1 = 1 \le 2^1$ ✓ and $F_2 = 1 \le 2^2$ ✓ (why 2 base cases?)
Strong Induction: Prove $P(1) \wedge P(2) \wedge \cdots \wedge P(n) \rightarrow P(n+1)$ for $n \ge 2$.
Assume: $P(1) \wedge P(2) \wedge \cdots \wedge P(n)$: $F_i \le 2^i$ for $1 \le i \le n$.

F_{n+1}

Recurrences: Fibonacci Numbers

Growth rate of rabbits, Sanskrit poetry, family trees of bees, ...

$$F_1 = 1;\ F_2 = 1; \qquad F_n = F_{n-1} + F_{n-2} \quad \text{for } n > 2.$$

$\boldsymbol{F_1}$	$\boldsymbol{F_2}$	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	$\cdots$
1	**1**	2	3	5	8	13	21	34	55	89	144	$\cdots$

Let us prove $P(n) : F_n \le 2^n$ by **strong induction**.
Base Cases: $F_1 = 1 \le 2^1$ ✓ and $F_2 = 1 \le 2^2$ ✓ (why 2 base cases?)
Strong Induction: Prove $P(1) \wedge P(2) \wedge \cdots \wedge P(n) \to P(n+1)$ for $n \ge 2$.
Assume: $P(1) \wedge P(2) \wedge \cdots \wedge P(n)$: $F_i \le 2^i$ for $1 \le i \le n$.

$$F_{n+1} = F_n + F_{n-1}$$ (needs $n \ge 2$)

Recurrences: Fibonacci Numbers

Growth rate of rabbits, Sanskrit poetry, family trees of bees, ...

$$F_1 = 1; \ F_2 = 1; \qquad F_n = F_{n-1} + F_{n-2} \quad \text{for } n > 2.$$

$\boldsymbol{F_1}$	$\boldsymbol{F_2}$	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	$\cdots$
1	**1**	2	3	5	8	13	21	34	55	89	144	$\cdots$

Let us prove $P(n) : F_n \le 2^n$ by **strong induction**.
Base Cases: $F_1 = 1 \le 2^1$ ✓ and $F_2 = 1 \le 2^2$ ✓ (why 2 base cases?)
Strong Induction: Prove $P(1) \wedge P(2) \wedge \cdots \wedge P(n) \rightarrow P(n+1)$ for $n \ge 2$.
Assume: $P(1) \wedge P(2) \wedge \cdots \wedge P(n)$: $F_i \le 2^i$ for $1 \le i \le n$.

$$\begin{aligned} F_{n+1} &= F_n + F_{n-1} && \text{(needs } n \ge 2\text{)} \\ &\le 2^n + 2^{n-1} && \text{(strong indction hypothesis)} \end{aligned}$$

Recurrences: Fibonacci Numbers

Growth rate of rabbits, Sanskrit poetry, family trees of bees, ...

$$F_1 = 1; \; F_2 = 1; \qquad F_n = F_{n-1} + F_{n-2} \quad \text{for } n > 2.$$

$\boldsymbol{F_1}$	$\boldsymbol{F_2}$	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	$\cdots$
1	**1**	2	3	5	8	13	21	34	55	89	144	$\cdots$

Let us prove $P(n) : F_n \le 2^n$ by **strong induction**.
Base Cases: $F_1 = 1 \le 2^1$ ✓ and $F_2 = 1 \le 2^2$ ✓ (why 2 base cases?)
Strong Induction: Prove $P(1) \wedge P(2) \wedge \cdots \wedge P(n) \to P(n+1)$ for $n \ge 2$.
Assume: $P(1) \wedge P(2) \wedge \cdots \wedge P(n)$: $F_i \le 2^i$ for $1 \le i \le n$.

$$\begin{aligned} F_{n+1} &= F_n + F_{n-1} && \text{(needs } n \ge 2) \\ &\le 2^n + 2^{n-1} && \text{(strong indction hypothesis)} \\ &\le 2 \times 2^n = 2^{n+1} \end{aligned}$$

So, $F_{n+1} \le 2^{n+1}$, concluding the proof. ■

Recurrences: Fibonacci Numbers

Growth rate of rabbits, Sanskrit poetry, family trees of bees, ...

$$F_1 = 1;\ F_2 = 1; \qquad F_n = F_{n-1} + F_{n-2} \quad \text{for } n > 2.$$

$\boldsymbol{F_1}$	$\boldsymbol{F_2}$	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	$\cdots$
1	**1**	2	3	5	8	13	21	34	55	89	144	$\cdots$

Let us prove $P(n) : F_n \le 2^n$ by **strong induction**.
Base Cases: $F_1 = 1 \le 2^1$ ✓ and $F_2 = 1 \le 2^2$ ✓ (why 2 base cases?)
Strong Induction: Prove $P(1) \wedge P(2) \wedge \cdots \wedge P(n) \to P(n+1)$ for $n \ge 2$.
Assume: $P(1) \wedge P(2) \wedge \cdots \wedge P(n)$: $F_i \le 2^i$ for $1 \le i \le n$.

$$\begin{aligned} F_{n+1} &= F_n + F_{n-1} && \text{(needs } n \ge 2\text{)} \\ &\le 2^n + 2^{n-1} && \text{(strong indction hypothesis)} \\ &\le 2 \times 2^n = 2^{n+1} \end{aligned}$$

So, $F_{n+1} \le 2^{n+1}$, concluding the proof. ■

Practice. Prove $F_n \ge (\frac{3}{2})^n$ for $n \ge 11$.

Recursive Programs

```
out=Big(n)
 if(n==0) out=1;
 else out=2*Big(n-1);
```

Does this function compute 2^n?

Recursive Programs

Proving correctness: let's prove $\texttt{Big}(n) = 2^n$ for $n \geq 1$

```
out=Big(n)
 if(n==0) out=1;
 else out=2*Big(n-1);
```

Does this function compute 2^n?

Recursive Programs

Proving correctness: let's prove $\texttt{Big}(n) = 2^n$ for $n \geq 1$

```
out=Big(n)
 if(n==0) out=1;
 else out=2*Big(n-1);
```

Does this function compute 2^n?

Induction.

Recursive Programs

Proving correctness: let's prove $\texttt{Big}(n) = 2^n$ for $n \geq 1$

Induction.

When $n = 0$, $\texttt{Big}(0) = 1 = 2^0$ ✓

```
out=Big(n)
 if(n==0) out=1;
 else out=2*Big(n-1);
```

Does this function compute 2^n?

Recursive Programs

Proving correctness: let's prove $\texttt{Big}(n) = 2^n$ for $n \geq 1$

```
out=Big(n)
 if(n==0) out=1;
 else out=2*Big(n-1);
```

Does this function compute 2^n?

Induction.

When $n = 0$, $\texttt{Big}(0) = 1 = 2^0$ ✓

Assume $\texttt{Big}(n) = 2^n$ for $n \geq 0$

Recursive Programs

Proving correctness: let's prove $\texttt{Big}(n) = 2^n$ for $n \geq 1$

```
out=Big(n)
 if(n==0) out=1;
 else out=2*Big(n-1);
```

Does this function compute 2^n?

Induction.

When $n = 0$, $\texttt{Big}(0) = 1 = 2^0$ ✓

Assume $\texttt{Big}(n) = 2^n$ for $n \geq 0$

$$\texttt{Big}(n+1) = 2 \times \texttt{Big}(n) = 2 \times 2^n = 2^{n+1}.$$

Recursive Programs

```
out=Big(n)
 if(n==0) out=1;
 else out=2*Big(n-1);
```

Does this function compute 2^n?

Proving correctness: let's prove $\texttt{Big}(n) = 2^n$ for $n \geq 1$

Induction.

When $n = 0$, $\texttt{Big}(0) = 1 = 2^0$ ✓

Assume $\texttt{Big}(n) = 2^n$ for $n \geq 0$

$$\texttt{Big}(n+1) = 2 \times \texttt{Big}(n) = 2 \times 2^n = 2^{n+1}.$$

What is the runtime?

Let $T_n =$ runtime of `Big` for input n.

Recursive Programs

```
out=Big(n)
 if(n==0) out=1;
 else out=2*Big(n-1);
```

Does this function compute 2^n?

Proving correctness: let's prove $\texttt{Big}(n) = 2^n$ for $n \geq 1$

Induction.

When $n = 0$, $\texttt{Big}(0) = 1 = 2^0$ ✓

Assume $\texttt{Big}(n) = 2^n$ for $n \geq 0$

$$\texttt{Big}(n+1) = 2 \times \texttt{Big}(n) = 2 \times 2^n = 2^{n+1}.$$

What is the runtime?

Let $T_n =$ runtime of `Big` for input n.

$$T_0 = 2$$

Recursive Programs

Proving correctness: let's prove $\texttt{Big}(n) = 2^n$ for $n \geq 1$

```
out=Big(n)
 if(n==0) out=1;
 else out=2*Big(n-1);
```

Induction.
When $n = 0$, $\texttt{Big}(0) = 1 = 2^0$ ✓
Assume $\texttt{Big}(n) = 2^n$ for $n \geq 0$

$$\texttt{Big}(n+1) = 2 \times \texttt{Big}(n) = 2 \times 2^n = 2^{n+1}.$$

Does this function compute 2^n?

What is the runtime?
Let $T_n =$ runtime of `Big` for input n.

$$\begin{aligned} T_0 &= 2 \\ T_n &= T_{n-1} + (\text{check } \texttt{n==0}) + (\text{multiply by 2}) + (\text{assign to } \texttt{out}) \end{aligned}$$

Recursive Programs

```
out=Big(n)
 if(n==0) out=1;
 else out=2*Big(n-1);
```

Does this function compute 2^n?

Proving correctness: let's prove $\texttt{Big}(n) = 2^n$ for $n \geq 1$

Induction.
When $n = 0$, $\texttt{Big}(0) = 1 = 2^0$ ✓
Assume $\texttt{Big}(n) = 2^n$ for $n \geq 0$

$$\texttt{Big}(n+1) = 2 \times \texttt{Big}(n) = 2 \times 2^n = 2^{n+1}.$$

What is the runtime?
Let $T_n =$ runtime of `Big` for input n.

$$\begin{aligned} T_0 &= 2 \\ T_n &= T_{n-1} + (\text{check } \texttt{n==0}) + (\text{multiply by } 2) + (\text{assign to } \texttt{out}) \\ &= T_{n-1} + 3 \end{aligned}$$

Recursive Programs

```
out=Big(n)
 if(n==0) out=1;
 else out=2*Big(n-1);
```

Does this function compute 2^n?

Proving correctness: let's prove $\texttt{Big}(n) = 2^n$ for $n \geq 1$

Induction.

When $n = 0$, $\texttt{Big}(0) = 1 = 2^0$ ✓

Assume $\texttt{Big}(n) = 2^n$ for $n \geq 0$

$$\texttt{Big}(n+1) = 2 \times \texttt{Big}(n) = 2 \times 2^n = 2^{n+1}.$$

What is the runtime?

Let $T_n =$ runtime of `Big` for input n.

$$\begin{aligned} T_0 &= 2 \\ T_n &= T_{n-1} + (\text{check } \texttt{n==0}) + (\text{multiply by } 2) + (\text{assign to } \texttt{out}) \\ &= T_{n-1} + 3 \end{aligned}$$

Exercise. Prove by induction that $T_n = 3n + 2$.

Recursive Sets: $\mathbb{N}$

Recursive definition of the natural numbers $\mathbb{N}$.

1. $1 \in \mathbb{N}$. **[basis]**

$$\mathbb{N} = \{1,$$

Recursive Sets: $\mathbb{N}$

Recursive definition of the natural numbers $\mathbb{N}$.

1. $1 \in \mathbb{N}$. **[basis]**
2. $x \in \mathbb{N} \rightarrow x + 1 \in \mathbb{N}$. **[constructor]**

$$\mathbb{N} = \{1, 2,$$

Recursive Sets: $\mathbb{N}$

Recursive definition of the natural numbers $\mathbb{N}$.

1. $1 \in \mathbb{N}$. **[basis]**
2. $x \in \mathbb{N} \rightarrow x + 1 \in \mathbb{N}$. **[constructor]**

$$\mathbb{N} = \{1, 2, 3,$$

Recursive Sets: $\mathbb{N}$

Recursive definition of the natural numbers $\mathbb{N}$.

1. $1 \in \mathbb{N}$. **[basis]**
2. $x \in \mathbb{N} \rightarrow x + 1 \in \mathbb{N}$. **[constructor]**

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

Recursive Sets: $\mathbb{N}$

Recursive definition of the natural numbers $\mathbb{N}$.

1. $1 \in \mathbb{N}$. **[basis]**
2. $x \in \mathbb{N} \rightarrow x + 1 \in \mathbb{N}$. **[constructor]**
3. 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.

Recursive Sets: $\mathbb{N}$

Recursive definition of the natural numbers $\mathbb{N}$.

1. $1 \in \mathbb{N}$. **[basis]**
2. $x \in \mathbb{N} \rightarrow x + 1 \in \mathbb{N}$. **[constructor]**
3. 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, Σ^*

Let ε be the *empty string* (similar to the empty set).

Recursive Sets: Finite Binary Strings, Σ^*

Let ε be the *empty string* (similar to the empty set).

Recursive definition of Σ^* (finite binary strings).

1. $\varepsilon \in \Sigma^*$. **[basis]**

Recursive Sets: Finite Binary Strings, Σ^*

Let ε be the *empty string* (similar to the empty set).

Recursive definition of Σ^* (finite binary strings).

1. $\varepsilon \in \Sigma^*$. **[basis]**
2. $x \in \Sigma^* \rightarrow x \bullet 0 \in \Sigma^*$ AND $x \bullet 1 \in \Sigma^*$. **[constructor]**

Recursive Sets: Finite Binary Strings, Σ^*

Let ε be the *empty string* (similar to the empty set).

Recursive definition of Σ^* (finite binary strings).

1. $\varepsilon \in \Sigma^*$. **[basis]**
2. $x \in \Sigma^* \rightarrow x \bullet 0 \in \Sigma^*$ AND $x \bullet 1 \in \Sigma^*$. **[constructor]**

Minimality is there by default: nothing else is in Σ^*.

Recursive Sets: Finite Binary Strings, Σ^*

Let ε be the *empty string* (similar to the empty set).

Recursive definition of Σ^* (finite binary strings).

1. $\varepsilon \in \Sigma^*$. **[basis]**
2. $x \in \Sigma^* \rightarrow x \bullet 0 \in \Sigma^*$ AND $x \bullet 1 \in \Sigma^*$. **[constructor]**

Minimality is there by default: nothing else is in Σ^*.

ε

Recursive Sets: Finite Binary Strings, Σ^*

Let ε be the *empty string* (similar to the empty set).

Recursive definition of Σ^* (finite binary strings).

1. $\varepsilon \in \Sigma^*$. **[basis]**
2. $x \in \Sigma^* \to x \bullet 0 \in \Sigma^*$ AND $x \bullet 1 \in \Sigma^*$. **[constructor]**

Minimality is there by default: nothing else is in Σ^*.

$$\varepsilon \to 0, 1$$

Recursive Sets: Finite Binary Strings, Σ^*

Let ε be the *empty string* (similar to the empty set).

Recursive definition of Σ^* (finite binary strings).

1. $\varepsilon \in \Sigma^*$. **[basis]**
2. $x \in \Sigma^* \rightarrow x \bullet 0 \in \Sigma^*$ AND $x \bullet 1 \in \Sigma^*$. **[constructor]**

Minimality is there by default: nothing else is in Σ^*.

$$\varepsilon \rightarrow 0, 1 \rightarrow 00, 01, 10, 11$$

Recursive Sets: Finite Binary Strings, Σ^*

Let ε be the *empty string* (similar to the empty set).

Recursive definition of Σ^* (finite binary strings).

1. $\varepsilon \in \Sigma^*$. **[basis]**
2. $x \in \Sigma^* \rightarrow x \bullet 0 \in \Sigma^*$ AND $x \bullet 1 \in \Sigma^*$. **[constructor]**

Minimality is there by default: nothing else is in Σ^*.

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

Recursive Sets: Finite Binary Strings, Σ^*

Let ε be the *empty string* (similar to the empty set).

Recursive definition of Σ^* (finite binary strings).
1. $\varepsilon \in \Sigma^*$. **[basis]**
2. $x \in \Sigma^* \to x\bullet 0 \in \Sigma^*$ AND $x\bullet 1 \in \Sigma^*$. **[constructor]**

Minimality is there by default: nothing else is in Σ^*.

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

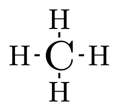
$$\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, \ldots\}$$

Practice. Exercise 7.12

Recursive Structures: Trees

Sir Aurthur Cayley discovered trees when modeling chemical hydrocarbons,

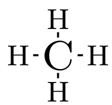
methane, CH_4



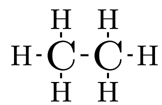
Recursive Structures: Trees

Sir Aurthur Cayley discovered trees when modeling chemical hydrocarbons,

methane, CH_4



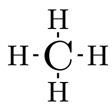
ethane, C_2H_6



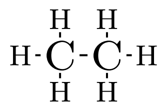
Recursive Structures: Trees

Sir Aurthur Cayley discovered trees when modeling chemical hydrocarbons,

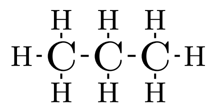
methane, CH_4



ethane, C_2H_6



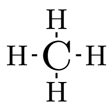
propane, C_3H_8



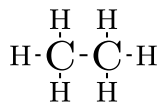
Recursive Structures: Trees

Sir Aurthur Cayley discovered trees when modeling chemical hydrocarbons,

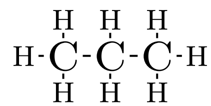
methane, CH_4



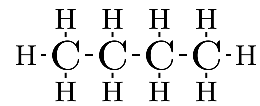
ethane, C_2H_6



propane, C_3H_8



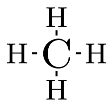
butane, C_4H_{10}



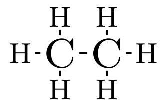
Recursive Structures: Trees

Sir Aurthur Cayley discovered trees when modeling chemical hydrocarbons,

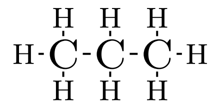
methane, CH_4



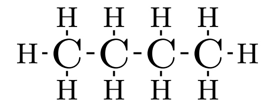
ethane, C_2H_6



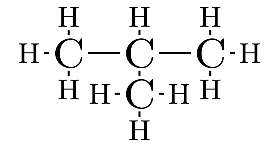
propane, C_3H_8



butane, C_4H_{10}



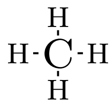
iso-butane, C_4H_{10}



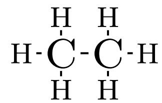
Recursive Structures: Trees

Sir Aurthur Cayley discovered trees when modeling chemical hydrocarbons,

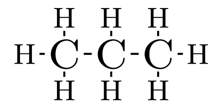
methane, CH_4



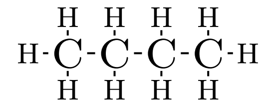
ethane, C_2H_6



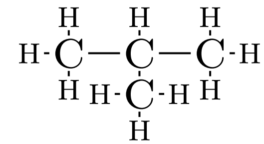
propane, C_3H_8



butane, C_4H_{10}



iso-butane, C_4H_{10}



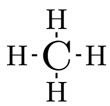
Trees have many uses in computer science

- Search trees.
- Game trees.
- Decision trees.
- Compression trees.
- Multi-processor trees.
- Parse trees.
- Expression trees.
- Ancestry trees.
- Organizational trees.
- ...

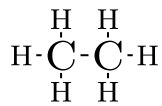
Recursive Structures: Trees

Sir Aurthur Cayley discovered trees when modeling chemical hydrocarbons,

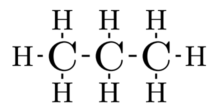
methane, CH_4



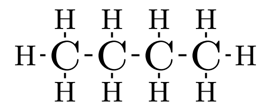
ethane, C_2H_6



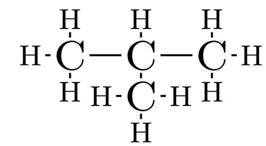
propane, C_3H_8



butane, C_4H_{10}



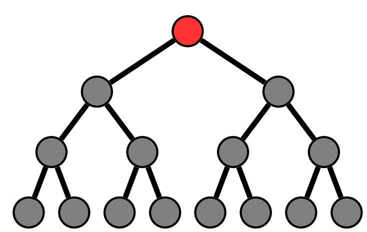
iso-butane, C_4H_{10}



Trees have many uses in computer science

- Search trees.
- Game trees.
- Decision trees.
- Compression trees.
- Multi-processor trees.
- Parse trees.
- Expression trees.
- Ancestry trees.
- Organizational trees.
- ...

Tree.

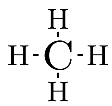


Not a tree.

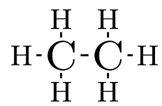
Recursive Structures: Trees

Sir Aurthur Cayley discovered trees when modeling chemical hydrocarbons,

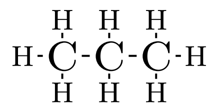
methane, CH_4



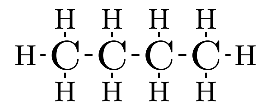
ethane, C_2H_6



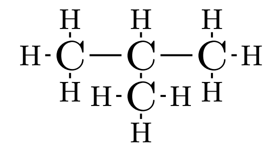
propane, C_3H_8



butane, C_4H_{10}



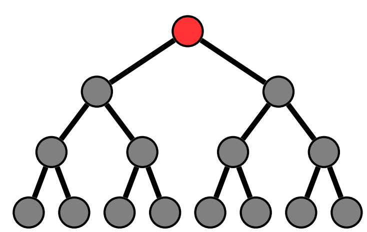
iso-butane, C_4H_{10}



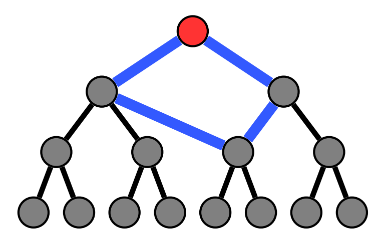
Trees have many uses in computer science

- Search trees.
- Game trees.
- Decision trees.
- Compression trees.
- Multi-processor trees.
- Parse trees.
- Expression trees.
- Ancestry trees.
- Organizational trees.
- ...

Tree.



Not a tree.



Rooted Binary Trees (RBT)

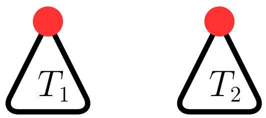
Recursive definition of Rooted Binary Trees (RBT).

1. The empty tree ε is an RBT.

Rooted Binary Trees (RBT)

Recursive definition of Rooted Binary Trees (RBT).

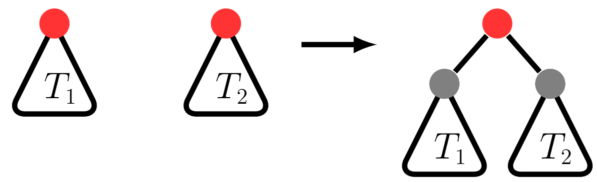
1. The empty tree ε is an RBT.
2. If T_1, T_2 are disjoint RBTs with roots r_1 and r_2, then linking r_1 and r_2 to a *new* root r gives a new RBT with root r.



Rooted Binary Trees (RBT)

Recursive definition of Rooted Binary Trees (RBT).

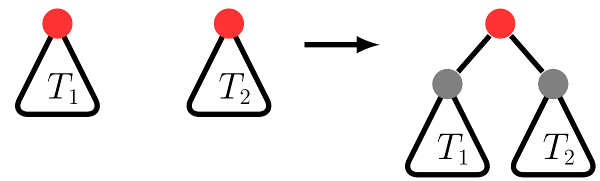
1. The empty tree ε is an RBT.
2. If T_1, T_2 are disjoint RBTs with roots r_1 and r_2, then linking r_1 and r_2 to a *new* root r gives a new RBT with root r.



Rooted Binary Trees (RBT)

Recursive definition of Rooted Binary Trees (RBT).

1. The empty tree ε is an RBT.
2. If T_1, T_2 are disjoint RBTs with roots r_1 and r_2, then linking r_1 and r_2 to a *new* root r gives a new RBT with root r.

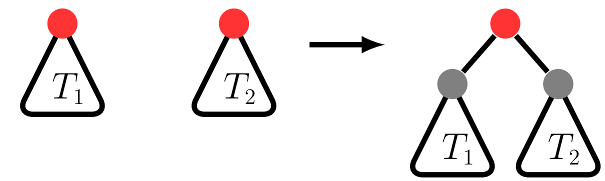


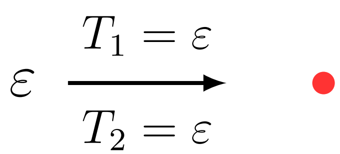
ε

Rooted Binary Trees (RBT)

Recursive definition of Rooted Binary Trees (RBT).

1. The empty tree ε is an RBT.
2. If T_1, T_2 are disjoint RBTs with roots r_1 and r_2, then linking r_1 and r_2 to a *new* root r gives a new RBT with root r.

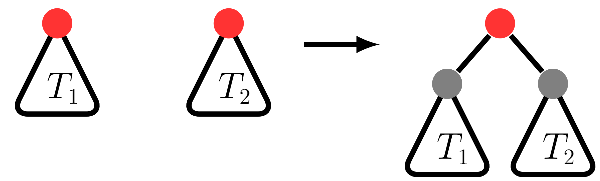


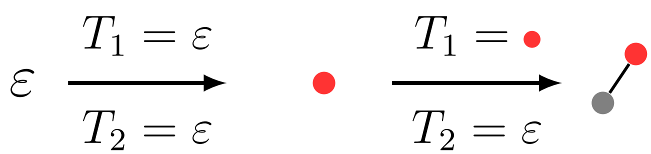


Rooted Binary Trees (RBT)

Recursive definition of Rooted Binary Trees (RBT).

1. The empty tree ε is an RBT.
2. If T_1, T_2 are disjoint RBTs with roots r_1 and r_2, then linking r_1 and r_2 to a *new* root r gives a new RBT with root r.

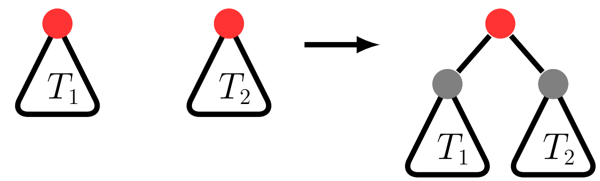


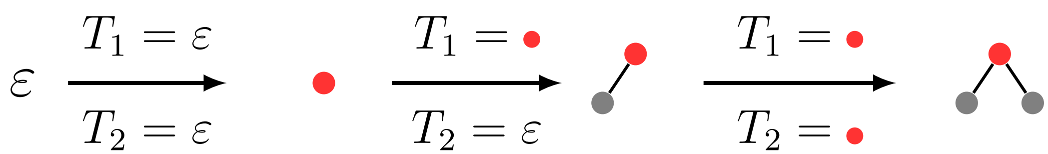


Rooted Binary Trees (RBT)

Recursive definition of Rooted Binary Trees (RBT).

1. The empty tree ε is an RBT.
2. If T_1, T_2 are disjoint RBTs with roots r_1 and r_2, then linking r_1 and r_2 to a *new* root r gives a new RBT with root r.

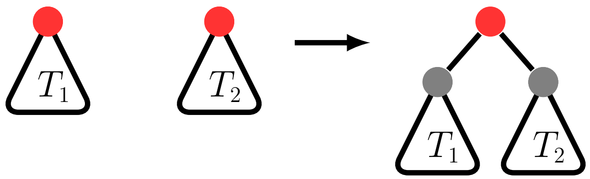


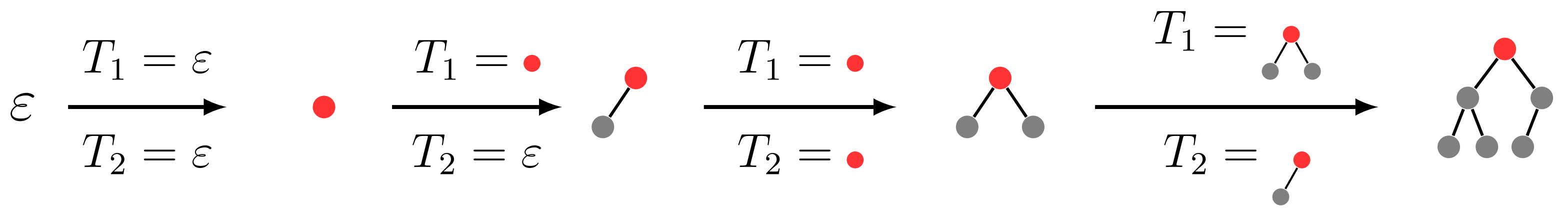


Rooted Binary Trees (RBT)

Recursive definition of Rooted Binary Trees (RBT).

1. The empty tree ε is an RBT.
2. If T_1, T_2 are disjoint RBTs with roots r_1 and r_2, then linking r_1 and r_2 to a *new* root r gives a new RBT with root r.

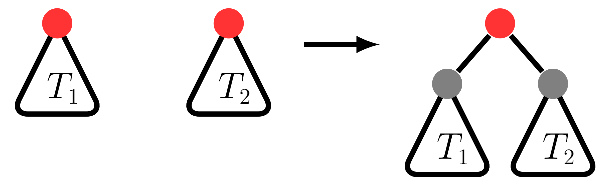


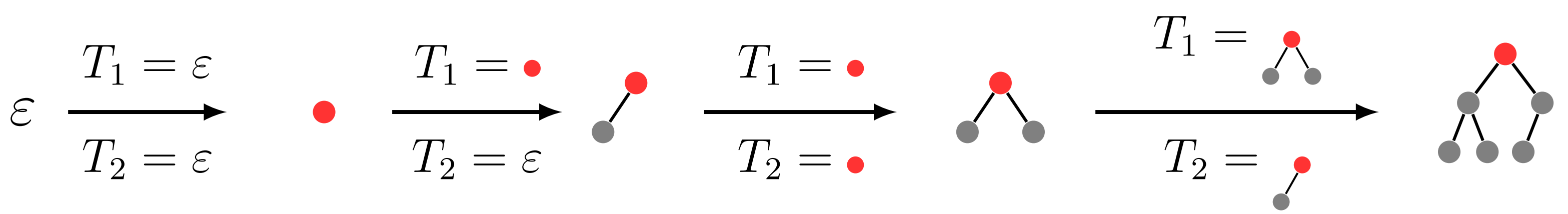


Rooted Binary Trees (RBT)

Recursive definition of Rooted Binary Trees (RBT).

1. The empty tree ε is an RBT.
2. If T_1, T_2 are disjoint RBTs with roots r_1 and r_2, then linking r_1 and r_2 to a *new* root r gives a new RBT with root r.

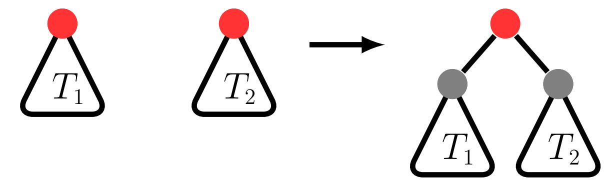


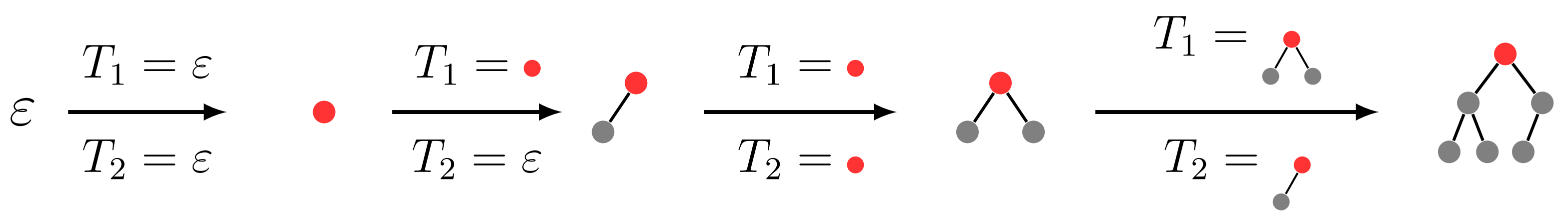


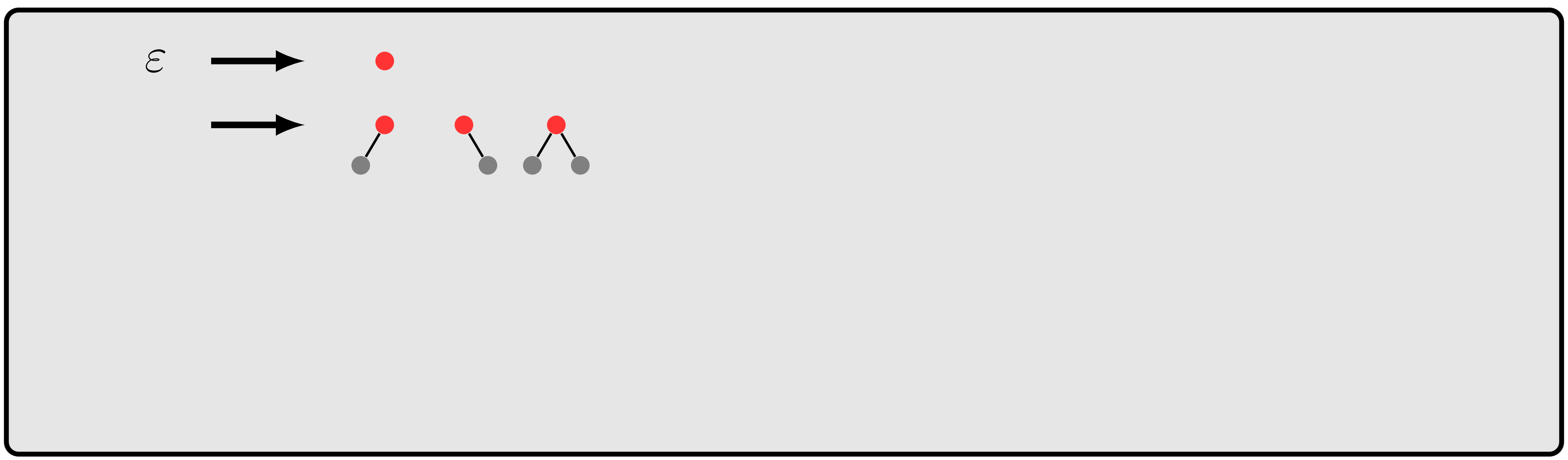
Rooted Binary Trees (RBT)

Recursive definition of Rooted Binary Trees (RBT).

1. The empty tree ε is an RBT.
2. If T_1, T_2 are disjoint RBTs with roots r_1 and r_2, then linking r_1 and r_2 to a *new* root r gives a new RBT with root r.



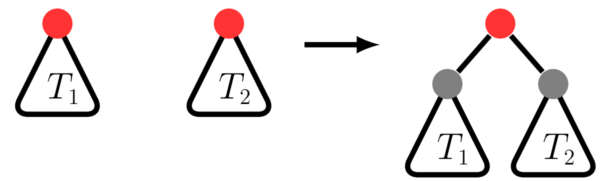


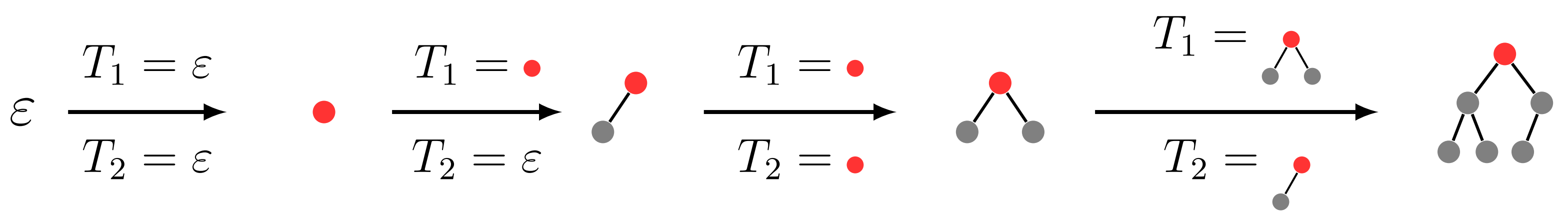


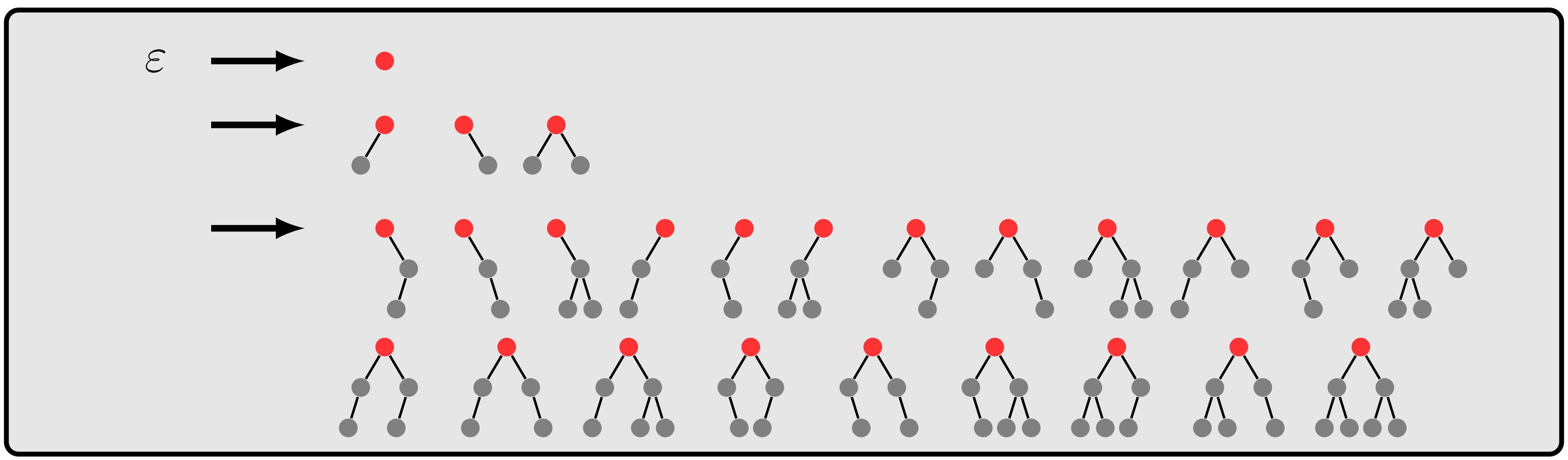
Rooted Binary Trees (RBT)

Recursive definition of Rooted Binary Trees (RBT).

1. The empty tree ε is an RBT.
2. If T_1, T_2 are disjoint RBTs with roots r_1 and r_2, then linking r_1 and r_2 to a *new* root r gives a new RBT with root r.

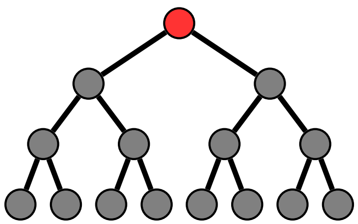




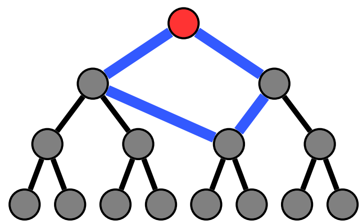


Trees Are Important: Food for Thought

- Tree.



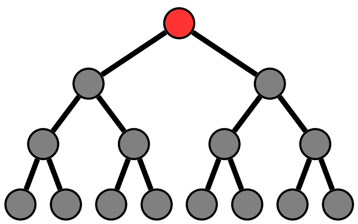
Not a tree.



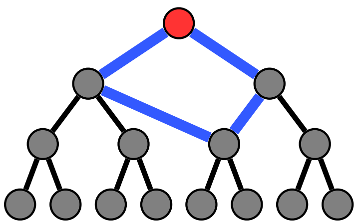
Do we *know* the right structure is not a tree?

Trees Are Important: Food for Thought

- Tree.



Not a tree.

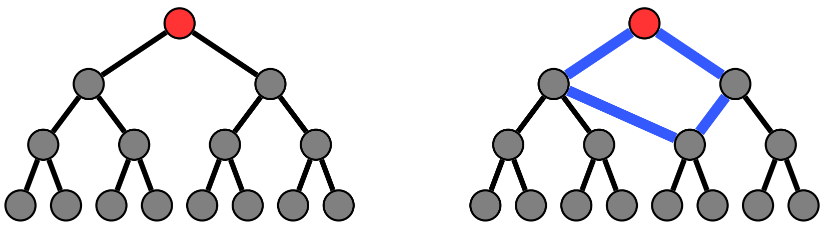


Do we *know* the right structure is not a tree?

Are we *sure* it can't be derived?

Trees Are Important: Food for Thought

- Tree. Not a tree.



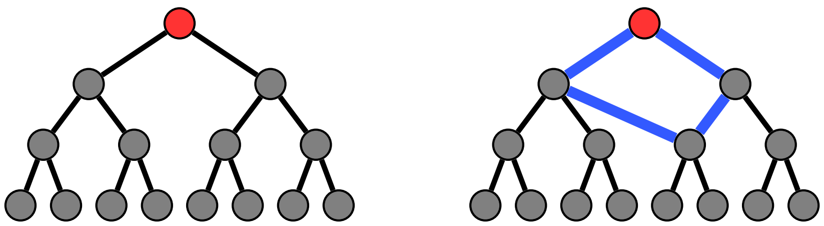
Do we *know* the right structure is not a tree?

Are we *sure* it can't be derived?

- Is there only one way to derive a tree?

Trees Are Important: Food for Thought

- Tree. Not a tree.



Do we *know* the right structure is not a tree?

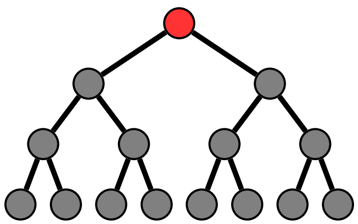
Are we *sure* it can't be derived?

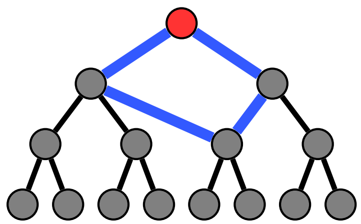
- Is there only one way to derive a tree?

- 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.

Trees Are Important: Food for Thought

- Tree. Not a tree.





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?