Introduction to Theory of Computation

Introduction to Theory of Computation
CSC-4890

Division of Computer Science and Engineering, LSU
Fall 2015

- **Instructor:** Konstantin (Costas) Busch
  - Page: http://www.csc.lsu.edu/~busch/courses/theorycomp/fall2015/
  - Extra Slides
    - Other models of computation

Other Models of Computation

Models of computation:

- Turing Machines
- Recursive Functions
- Post Systems
- Rewriting Systems

Church's Thesis:

All models of computation are equivalent

Turing's Thesis:

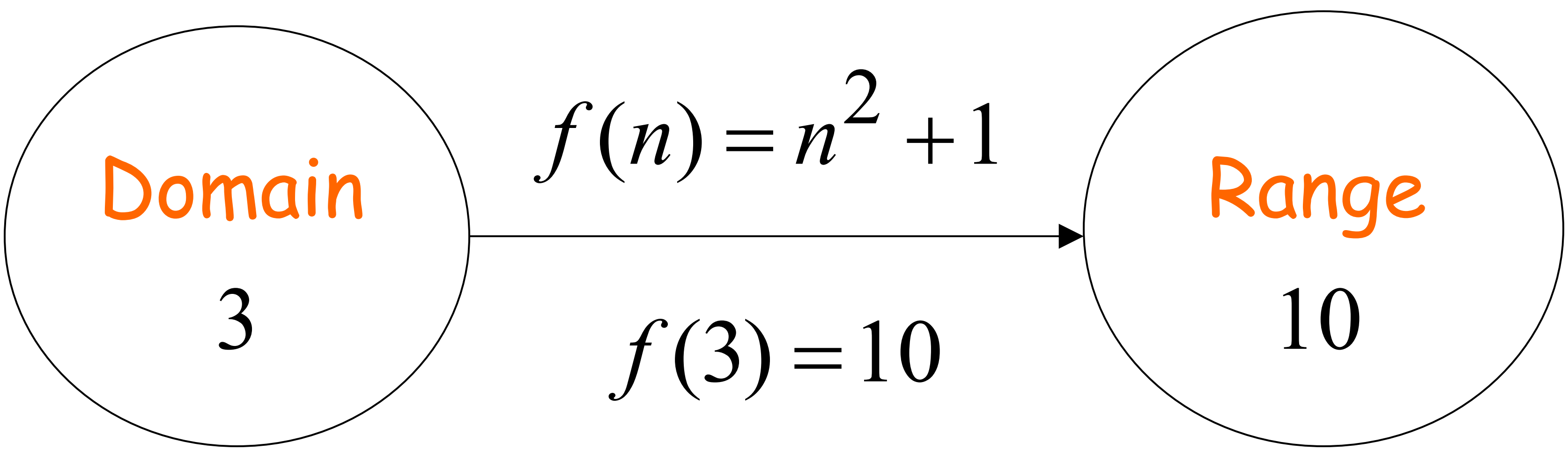
A computation is mechanical if and only if it can be performed by a Turing Machine

Church's and Turing's Thesis are similar:

Church-Turing Thesis

Recursive Functions

An example function:



We need a way to define functions

We need a set of basic functions

Basic Primitive Recursive Functions

Zero function: $z(x) = 0$

Successor function: $s(x) = x + 1$

Projection functions: $p_1(x_1, x_2) = x_1$

$$p_2(x_1, x_2) = x_2$$

Building complicated functions:

Composition: $$f(x,y)=h(g_1(x,y),g_2(x,y))$$

Primitive Recursion:

$$f(x,0)=g_1(x)$$

$$f(x,y+1)=h(g_2(x,y),f(x,y))$$

Any function built from
the basic primitive recursive functions
is called:

Primitive Recursive Function

A Primitive Recursive Function: $add(x, y)$

$$add(x,0) = x$$

(projection)

$$add(x, y+1) = s(add(x, y))$$

(successor function)

$$
\begin{aligned}
add(3,2) &= s(add(3,1)) \\
&= s(s(add(3,0))) \\
&= s(s(3)) \\
&= s(4) \\
&= 5
\end{aligned}
$$

Another Primitive Recursive Function:

$$mult(x, y)$$

$$mult(x,0) = 0$$

$$mult(x, y+1) = add(x, mult(x, y))$$

Theorem:

The set of primitive recursive functions is countable

Proof:

Each primitive recursive function can be encoded as a string

Enumerate all strings in proper order

Check if a string is a function

Theorem

there is a function that
is not primitive recursive

Proof:

Enumerate the primitive recursive functions

$$f_1, f_2, f_3, \ldots$$

Define function $g(i) = f_i(i) + 1$

g differs from every f_i

g is not primitive recursive

END OF PROOF

A specific function that is <u>not</u>
Primitive Recursive:

Ackermann's function:

$$A(0, y) = y + 1$$

$$A(x, 0) = A(x-1, 1)$$

$$A(x, y+1) = A(x-1, A(x, y))$$

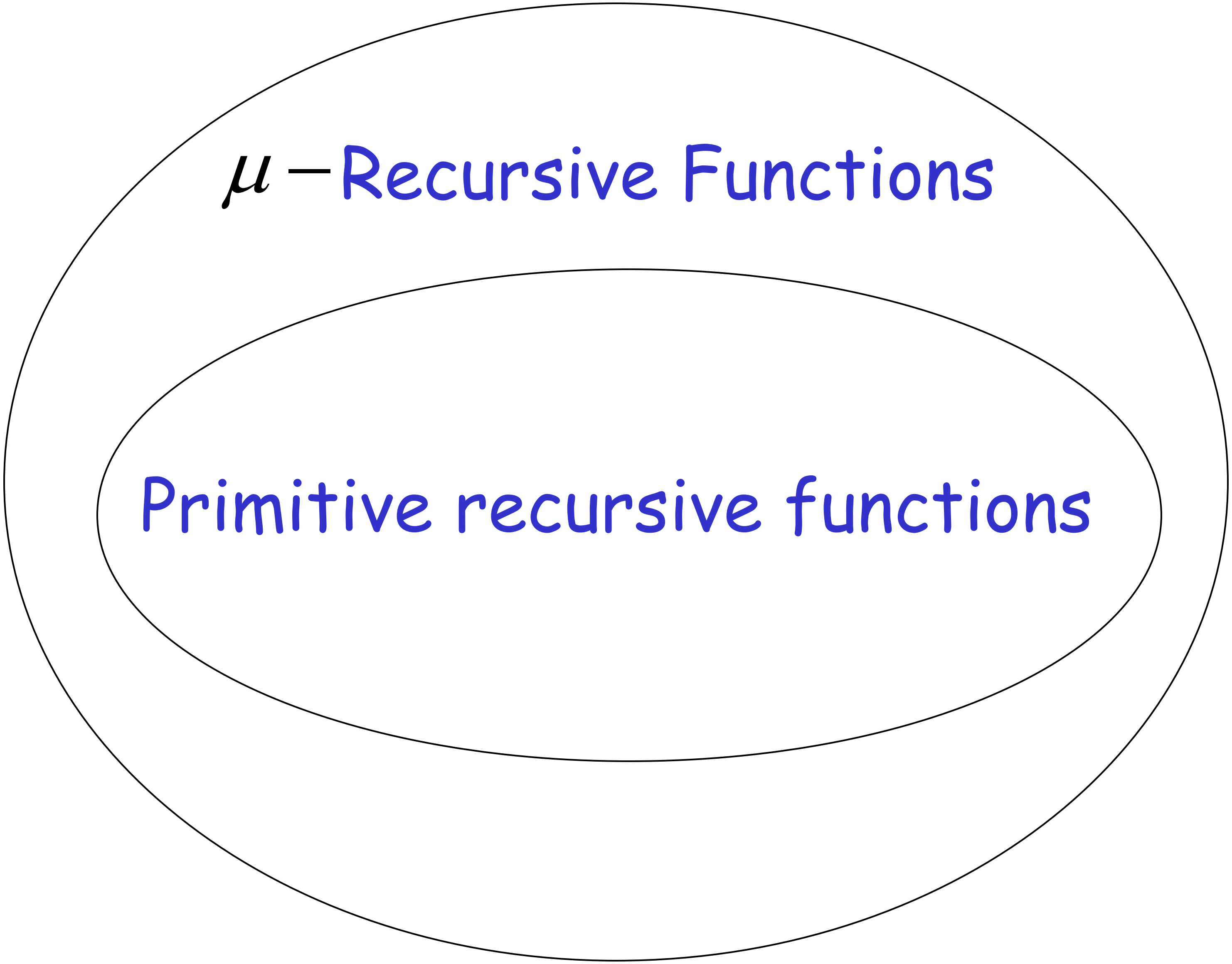
Grows very fast,
faster than any primitive recursive function

$\mu-$Recursive Functions

$$\mu y(g(x,y)) = \text{smallest } y \text{ such that } g(x,y) = 0$$

Accerman's function is a

$\mu-$Recursive Function


$\mu-$Recursive Functions
Primitive recursive functions

Post Systems

- Have Axioms

- Have Productions

Very similar with unrestricted grammars

Example: Unary Addition

Axiom: $1+1=11$

Productions:

$$V_1+V_2=V_3 \quad \rightarrow \quad V_1 1+V_2=V_3 1$$

$$V_1+V_2=V_3 \quad \rightarrow \quad V_1+V_2 1=V_3 1$$

A production:

$$V_1 + V_2 = V_3 \quad \rightarrow \quad V_1 1 + V_2 = V_3 1$$

$$1+1=11 \Rightarrow 11+1=111 \Rightarrow 11+11=1111$$

$$V_1 + V_2 = V_3 \quad \rightarrow \quad V_1 + V_2 1 = V_3 1$$

Post systems are good for proving mathematical statements from a set of Axioms

Theorem:

A language is recursively enumerable if and only if a Post system generates it

Rewriting Systems

They convert one string to another

- Matrix Grammars
- Markov Algorithms
- Lindenmayer-Systems

Very similar to unrestricted grammars

Matrix Grammars

Example:

$$P_1: \quad S \rightarrow S_1 S_2$$

$$P_2: \quad S_1 \rightarrow aS_1, \; S_2 \rightarrow bS_2c$$

$$P_3: \quad S_1 \rightarrow \lambda, \; S_2 \rightarrow \lambda$$

Derivation:

$$S \Rightarrow S_1S_2 \Rightarrow aS_1bS_2c \Rightarrow aaS_1bbS_2cc \Rightarrow aabbcc$$

A set of productions is applied simultaneously

$$P_1: \quad S \to S_1 S_2$$

$$P_2: \quad S_1 \to aS_1, \;\; S_2 \to bS_2c$$

$$P_3: \quad S_1 \to \lambda, \;\; S_2 \to \lambda$$

$$L = \{a^n b^n c^n : n \geq 0\}$$

Theorem:

A language is recursively enumerable
if and only if
a Matrix grammar generates it

Markov Algorithms

Grammars that produce λ

Example:

$$ab \rightarrow S$$

$$aSb \rightarrow S$$

$$S \rightarrow .\lambda$$

Derivation:

$$aaabbb \Rightarrow aaSbb \Rightarrow aSb \Rightarrow S \Rightarrow \lambda$$

$$ab \to S$$

$$aSb \to S$$

$$S \to .\lambda$$

$$L = \{a^n b^n : n \geq 0\}$$

In general: $L = \{w : \; w \overset{*}{\Rightarrow} \lambda\}$

Theorem:

A language is recursively enumerable
if and only if
a Markov algorithm generates it

Lindenmayer-Systems

They are parallel rewriting systems

Example: $a \to aa$

Derivation: $a \Rightarrow aa \Rightarrow aaaa \Rightarrow aaaaaaaa$

$$L = \{a^{2^n} : n \geq 0\}$$

Lindenmayer-Systems are not general
As recursively enumerable languages

Extended Lindenmayer-Systems: $(x, a, y) \rightarrow u$

context

Theorem:

A language is recursively enumerable
if and only if an
Extended Lindenmayer-System generates it