

**Theory of Computation**  


---

**Lecture 01**  
**Introduction**

C SC 473 Automata, Grammars & Languages

---

---

---

---

---

---

---

---

**Fundamental Questions**

Theory of Computation seeks to answer fundamental questions about computing

- What is computation?
  - Ancient activity back as far as Babylonians, Egyptians
  - Not precisely settled until circa 1936
- What can be computed?
  - Different ways of computing (C, Lisp, ...) result in the same "effectively computable" functions from input to output?
- What cannot be computed?
  - Not  $\sqrt{2}$  but can get arbitrarily close
  - Are there precisely defined tasks ("problems") that cannot be carried out? Yes/No decisions that cannot be computed?
- What can be computed *efficiently*? (Computational Complexity)
  - Are there inherently difficult although computable problems?

C SC 473 Automata, Grammars & Languages 2

---

---

---

---

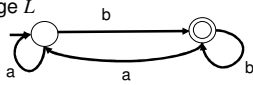
---

---

---

---

**Basic Concepts: Automata, Grammars & Languages**

- Language: a set of strings over some finite alphabet  $\Sigma$ 
  - Ex:  $L = \{TAA, TGA, TAG, \dots\}$  DNA codons
  - $\Sigma = \{A, G, C, T\}$
- Automaton (Machine): abstract (=simplified) model of a computing device. Used to "recognize" strings of a language  $L$ 
  - Ex: 

Finite Automaton  
(Finite State Machine)
- Grammar: finite set of string rewriting rules. Used to specify (derive) strings of a language
  - Ex:  $S \rightarrow +SS$  Context-Free Grammar (CFG)
  - $S \rightarrow x$

C SC 473 Automata, Grammars & Languages 3

---

---

---

---

---

---

---

---

### Languages

$L_1 = \{aa, ab, ba, bb\} \quad \Sigma = \{a, b\}$   
 $L_2 = \{\epsilon, a, aa, aaa, aaaa, \dots\} \quad \Sigma = \{a\}$   
 $L_3 = \{e : e \text{ is a well-formed arithmetic expression in } \mathbb{C}\}$   
 $\Sigma = \{0-9, a-z, A-Z, +, -, *, /, (, ), ., \cdot, \wedge, !, \dots\}$   
 $L_4 = \{p : p \text{ is a well-formed C program}\} \quad \Sigma = \{\text{ASCII}\}$   
 $L_5 = \{p : p \text{ is a w.-f. C program that halts for all inputs}\}$   
 $L_6 = \{(x, y) : x \text{ is a decimal integer and } y \text{ is its binary representation}\}$

C SC 473 Automata, Grammars & Languages 4

---

---

---

---

---

---

---

---

### Types of Machines

- Logic circuit
  - memoryless; values combined using gates

C SC 473 Automata, Grammars & Languages 5

---

---

---

---

---

---

---

---

### Types of Machines (cont.)

- Finite-state automaton (FSA)
  - bounded number of memory states
  - step: input, current state determines next state & output

C SC 473 Automata, Grammars & Languages 6

---

---

---

---

---

---

---

---

### Types of Machines (cont.)

- Pushdown Automaton (PDA)
  - finite control and a single *unbounded stack*

$L = \{ a^n b^n \# : n \geq 1 \}$

$\delta(q_2, a, \epsilon) = (q_2, A)$

- models finite program + one *unbounded stack of bounded registers*

C SC 473 Automata, Grammars & Languages 7

---

---

---

---

---

---

---

---

---

---

---

---

### Types of Machines (cont.)

- Random access machine (RAM)
  - finite program and an *unbounded, addressable* random access memory of "registers"
  - models general programs
    - unbounded # of bounded registers
    - Simple 1-addr instructions

Example:

```

R0 ← R0 + R1
L0 : JMPZ R1, L1
    INC R0
    DEC R1
    JMP L0
L1 : CONTINUE
    
```

C SC 473 Automata, Grammars & Languages 8

---

---

---

---

---

---

---

---

---

---

---

---

### Types of Machines (cont.)

- Turing Machine (TM)
  - finite control & tape of *bounded cells* unbounded in # to R
  - Input left adjusted on tape at start with blank cell terminating
  - current state, cell scanned determine next state & overprint symbol
  - control writes over symbol in cell and moves head 1 cell L or R
  - models simple "sequential" memory; no addressability
  - fixed amount of information (b bits) per cell

$\delta(q, X) = (p, Y, R)$

C SC 473 Automata, Grammars & Languages 9

---

---

---

---

---

---

---

---

---

---

---

---

### Theory of Computation

Study of languages and functions that can be described by computation that is finite in space and time

- Grammar Theory
  - Context-free grammars
  - Right-linear grammars
  - Unrestricted grammars
  - Capabilities and limitations
  - *Application: programming language specification*
- Automata Theory
  - FA
  - PDA
  - Turing Machines
  - Capabilities and limitations
  - Characterizing "what is computable?"
  - *Application: parsing algorithms*

C SC 473 Automata, Grammars & Languages 10

---

---

---

---

---

---

---

---

### Theory of Computation (cont'd)

- Computational Complexity Theory
  - Inherent difficulty of "problems"
  - Time/space resources needed for computation
  - "Intractable" problems
  - Ranking of problems by difficulty (hierarchies)
  - *Application: algorithm design, algorithm improvement, analysis*

C SC 473 Automata, Grammars & Languages 11

---

---

---

---

---

---

---

---

### FSA Ex: Specifying/Recognizing C Identifiers

- Deterministic FA  $\Lambda=\{a,\dots,z,A,\dots,Z, \_ \}$   $\Delta=\{0,\dots,9\}$ 
  - State diagram (labeled digraph)
  - Regular Expression
 
$$(\_ + a + \dots + A + \dots) \cdot (\_ + a + \dots + A + \dots + 0 + \dots + 9)^*$$
  - Right-Linear Grammar
 
$$S \rightarrow aT \mid \dots \mid zT \quad T \rightarrow aT \mid \dots \mid zT$$

$$\quad \mid AT \mid \dots \mid ZT \quad \quad \mid AT \mid \dots \mid ZT$$

$$\quad \mid \_ T \quad \quad \quad \quad \quad \mid 0T \mid \dots \mid 9T \mid \_ T$$

C SC 473 Automata, Grammars & Languages 12

---

---

---

---

---

---

---

---

### FSA Ex: C Floating Constants

- "A floating constant consists of an integer part, a decimal point, a fraction part, an **e** or **E**, an optionally signed integer exponent (and an optional type suffix ...). The integer and fraction parts both consist of a sequence of digits. Either the integer part or the fraction part (not both) may be missing; either the decimal point or the **e** and the exponent (not both) may be missing. ..."
- B. W. Kernighan and D.M. Ritchie, *The C Programming Language*, Prentice-Hall, 1978
- (The type is determined by the suffix; **F** or **f** makes it a **float**, **L** or **l** makes it a long double; otherwise it is double.)

C SC 473 Automata, Grammars & Languages

13

---

---

---

---

---

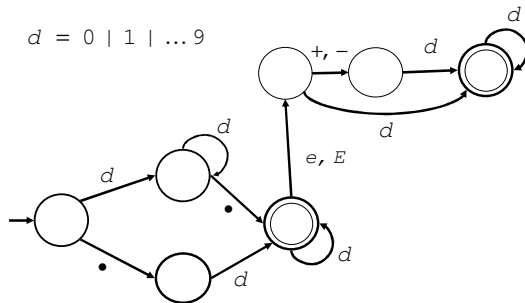
---

---

---

### FSA Ex: C Floats (cont'd)

$d = 0 | 1 | \dots | 9$



Note: type suffixes  
F, f, l, L omitted

C SC 473 Automata, Grammars & Languages

14

---

---

---

---

---

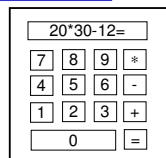
---

---

---

### CFG Ex: A Calculator Language

- Syntactic Classes
  - Numerals 3 40
  - Digits 0 1 9
  - Expressions 3\*9 40-3\*3
  - Commands 3\*9= 40-3\*3=



Note: no division & no decimal point

- Context-Free Grammar
    - $C \rightarrow E =$
    - $E \rightarrow N$
    - $E \rightarrow E + N$
    - $E \rightarrow E - N$
    - $E \rightarrow E * N$
    - $N \rightarrow ND$
    - $N \rightarrow D$
    - $D \rightarrow 0 \dots$
    - $D \rightarrow 9$
- terminals  $\Sigma = \{=, +, -, *, 0, \dots, 9\}$
- rules  $R$
- variables  $V = \{N, D, E, C\}$
- start variable = C
- grammar  $G = (V, \Sigma, R, C)$

C SC 473 Automata, Grammars & Languages

15

---

---

---

---

---

---

---

---

### Calculator Language (cont'd)

- Syntax Trees—exhibit “phrase structure”
- Numerals N
- Expressions E
- Commands C

Is this the parse you expected?

C SC 473 Automata, Grammars & Languages 16

---

---

---

---

---

---

---

---

---

---

### TM Ex: An “Algorithmically Unsolvable” Problem

- Q: Is there an algorithm for deciding if a given program  $P$  halts on a given input  $x$ ?

- A: No. There is *no program* that works correctly for *all*  $P, x$
- For the proof, we will need a simple programming language‡: *NatC*—a simplified C
  - One data type:  $\text{nat} = \{0, 1, 2, \dots\}$ . All variables of type  $\text{nat}$
  - All programs have one  $\text{nat}$  input and one  $\text{nat}$  output

‡We will later on use Turing Machines to model a “simple programming language”. *NatC* is simpler to describe.

C SC 473 Automata, Grammars & Languages 17

---

---

---

---

---

---

---

---

---

---

### Unsolvable Problem (cont'd)

- Observations:
  - A standard C compiler can be modified to accept only *NatC* programs as “legal”
  - Every *NatC* program  $P$  computes a function from natural numbers to natural numbers.  $f_P : \text{nat} \rightarrow \text{nat}$
  - Note:  $f_P$  may not be defined for some inputs, i.e., it is a *partial function*

```

nat P(nat x)
{
    if (x=3)
        return(6);
    else {
        while(x=x) do x=x+1;
        return x;
    }
}
    
```

P does not halt for some inputs

C SC 473 Automata, Grammars & Languages 18

---

---

---

---

---

---

---

---

---

---

### Unsolvable Problem (cont'd)

- Enumeration
  - A systematic list of all *NatC* programs  $P_0, P_1, P_2, \dots$
  - For program  $P_i$   $i$  is called the program's *index*
  - **program**→**index**: write out program as bit sequence in ASCII; interpret the bit sequence as a binary integer—its index
    - A program is just a string of characters!
  - **index**→**program**: given  $i \geq 0$ , convert to binary. Divide into 8-bit blocks. If such division is impossible (e.g., 3 characters) or if some block is not an ASCII code, or if the string is not a legal program,  $P_i$  will be the default "junk" program `{nat x; read(x); while(x=x) do x=x+1; write(x)}` which is undefined ("diverges"  $\uparrow$ ) for every legal input.
  - Conclusions about enumeration  $P_0, P_1, P_2, \dots$ 
    - Given  $n$  can compute  $P_n$  with *NatC* program
    - Given  $P$  can compute index  $n$  such that  $P = P_n$  with *NatC*.

C SC 473 Automata, Grammars & Languages

19

---

---

---

---

---

---

---

---

---

---

### Unsolvable Problem (cont'd)

- **Unsolvability Result**: Does  $P_n$  halt on input  $n$  ? Question cannot be settled by an algorithm.
- **Theorem**: Define the function  $h: \text{nat} \rightarrow \text{nat}$  by
  - $h(x) = 1$  if  $P_x$  halts on input  $x$  then  $1$  else  $0$
 Then  $h$  is *not computable* by any *NatC* program.
 

*Proof*: Proof by contradiction. Suppose (contrary to what is to be proved) that  $h$  is computable by a program, called `halt`. `halt` has input variable  $x$ , and output variable  $y$ .

By assumption

$$f_{\text{halt}}(x) = 1 \text{ if } P_x \text{ halts on input } x \text{ then } 1 \text{ else } 0$$

C SC 473 Automata, Grammars & Languages

20

---

---

---

---

---

---

---

---

---

---

### Unsolvable Problem (cont'd)

- Modify `halt` to a *NatC* function `nat halt(nat x)`
- Construct the following *NatC* program:
 

```

nat diagonal(nat n)
{ nat y;
  if halt(n)=0
    y:=1;
  else {
    y:=1;
    while (y!=0) do y:=y+1;
  }
  return y;
}
            
```
- Consequences
  - If `halt` is a legal program, so is "**diagonal**"
  - Therefore, **diagonal** has some index  $e$  in the enumeration:
    - $P_e = \text{diagonal}$

C SC 473 Automata, Grammars & Languages

21

---

---

---

---

---

---

---

---

---

---

### Unsolvable Problem (cont'd)

- How does `diagonal` behave on its own index  $e$  ?
- $f_{\text{diagonal}}(e)=1 \Leftrightarrow f_{\text{halt}}(e)=0 \Leftrightarrow P_e$  does not halt on  $e \Leftrightarrow \text{diagonal}$  does not halt on  $e$
- $f_{\text{diagonal}}(e)=\text{undefined} \Leftrightarrow f_{\text{halt}}(e)=1 \Leftrightarrow P_e$  halts on  $e \Leftrightarrow \text{diagonal}$  halts on  $e$
- $\therefore$  `diagonal` halts on  $e \Leftrightarrow \text{diagonal}$  does not halt on  $e$
- Contradiction!
- $\therefore$  program `diagonal` cannot exist Q.E.D.
- The "Halting Problem" is unsolvable
  - Undecidable, recursively undecidable, algorithmically undecidable, unsolvable

---

---

---

---

---

---

---

---