Welcome to 01418333 Formal Language and Automata Theory

Why Study Automata?

How Could That Be?

- Regular expressions are used in many systems.
  - E.g., UNIX a.*b.
  - E.g., DTD's describe XML tags with a RE format like person (name, addr, child*).
- Finite automata model protocols, electronic circuits.
  - Theory is used in *model-checking*.

2

How? – (2)

CFG ✓

grammar ขอ, หมาย ความ

if (___)
if a >b

- ☐ Context-free <u>grammars</u> are used to describe the syntax of essentially every programming language.
  - ☐ Not to forget their important role in describing natural languages.
- ☐ And DTD's taken as a whole, are really CFG's.

How? – (3)

☐ When developing solutions to real problems, we often confront the limitations of what software can do.

    ☐ *Undecidable* things – no program whatever can do it.

    ☐ *Intractable* things – there are programs, but no fast programs.

Course Outline

□ Regular Languages and their descriptors:

□ Finite automata, nondeterministic finite automata, regular expressions.

□ Algorithms to decide questions about regular languages, e.g., is it empty?

□ Closure properties of regular languages.

(handwritten annotations)

DFA In → ◯ →

NFA In → ◯ ⇉

ปกติจะออกทางเดียว

FA

เส้ง
รับ Input ลก ต้อง/ได้ตัวจริงแน่น
FA < DFA
        NFA
NFA
รับ Input เท่า ใดบ่างแต
output

Course Outline – (2)

CFL

☐ Context-free languages and their descriptors:

CFG

    ☐ Context-free grammars, pushdown automata.

    ☐ Decision and closure properties.

Course Outline – (3)

☐ Recursive and recursively enumerable languages.

☐ Turing machines, decidability of problems.

☐ The limit of what can be computed.

☐ Intractable problems.

☐ Problems that (appear to) require exponential time.

☐ NP-completeness and beyond.

Text

- Hopcroft, Motwani, Ullman, *Automata Theory, Languages, and Computation* 3rd Edition.
- Course covers essentially the entire book.

Finite Automata

Motivation

An Example

The slides are created by Jeffrey D. Ullman
http://infolab.stanford.edu/~ullman/ialc/spr10/spr10.html#LECTURE

Informal Explanation

- Finite automata are finite collections of states with transition rules that take you from one state to another.

- Original application was sequential switching circuits, where the "state" was the settings of internal bits.

- Today, several kinds of software can be modeled by FA.
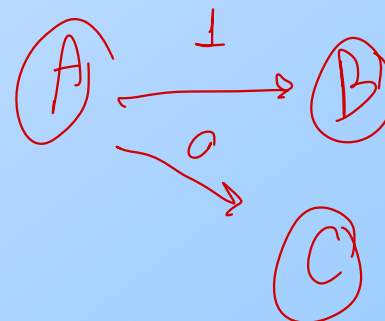
Representing FA

- Simplest representation is often a graph.
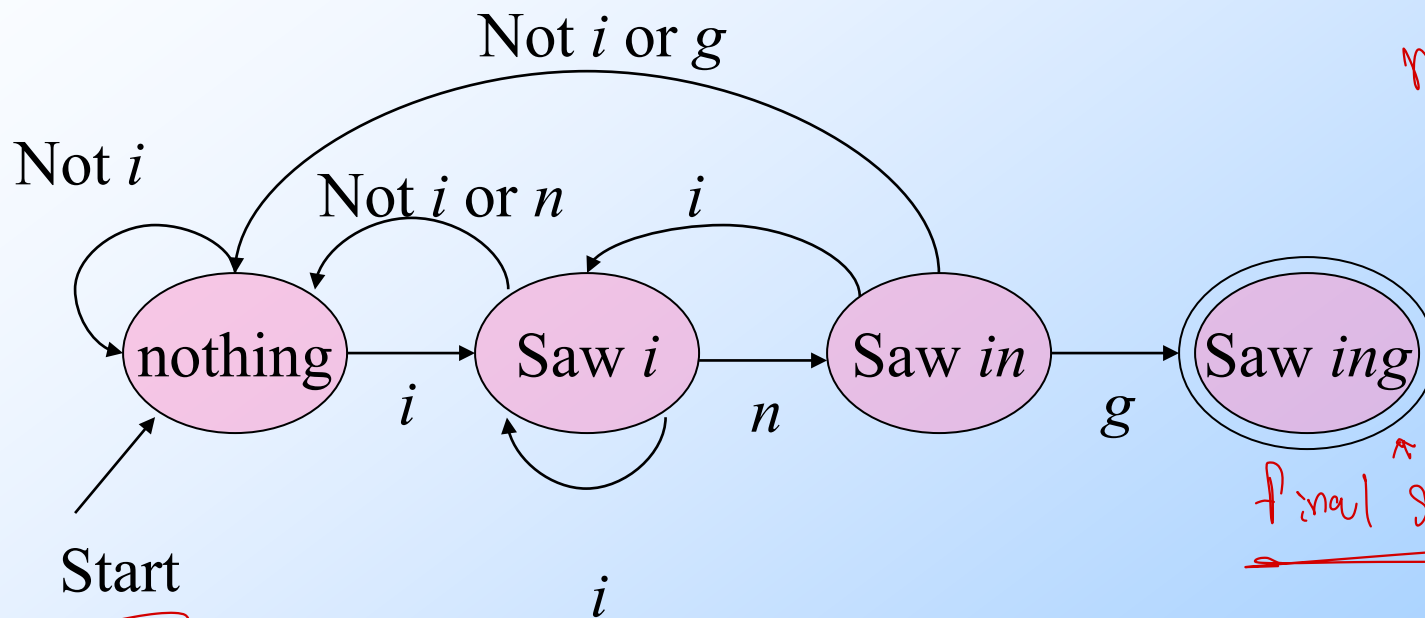 - Nodes = states.
  - เส้นเชื่อม Arcs indicate state transitions.
  - Labels on arcs tell what causes the transition. input

$A \longrightarrow B$

11

Example: Recognizing Strings Ending in "ing"

Not *i* or *g*

Not *i*

Not *i* or *n*

i

nothing → Saw *i* → Saw *in* → Saw *ing*

i

n

g

i

Start

Handwritten annotations:

Being ✓

accept ✓

reject ✗

final state

play → reject

ingby → reject

An

ภาค State หละ เส้น ใส่ ครบ. ทดอีกนิดก็ได.

12

Automata to Code

- In C/C++, make a piece of code for each state. This code:
    1. Reads the next input.
    2. Decides on the next state.
    3. Jumps to the beginning of the code for that state.

Example: Automata to Code

```
2: /* i seen */
   c = getNextInput();
   if (c == 'n') goto 3;
   else if (c == 'i') goto 2;
   else goto 1;
3: /* "in" seen */
   . . .
```

Automata to Code – Thoughts

- How would you do this in Java, which has no goto?
- You don't really write code like this.
- Rather, a code generator takes a "regular expression" describing the pattern(s) you are looking for.
  - Example: `.*ing` works in grep.

Example: Protocol for Sending Data

timeout

data in

Ready → Sending

Start

ack