Modeling Computation
CS 330: Discrete Structures

How do we model / describe computation?

and

What are the limits (if any) to these models / descriptions?

limits of computation

abstract machines / automata

states I/O transitions.

mathematical formula

functions

languages

"natural"

programming (context-free)

regular languages (regexps)

finite-state machines

turing machines

we can use a language to write down programs and express computations

— in CS, we mostly care about languages w/ well-defined syntax

syntax: rules that describe what valid sentences look like in the associated language.

the mechanism we might use to define or recognize a language says a lot about its complexity — and the complexity of the type of computation the language can be used to express!

one way to describe a formal language is w/ a grammar

a grammar $G = (V, T, S, P)$ consists of:

V, the vocabulary: a non-empty set of symbols.

$T \subset V$: the set of terminal symbols — cannot be rewritten as other symbols.

$V - T = N$: the set of non-terminal symbols

$S \in V$: the start symbol

P: a set of productions

each production is a "rewrite rule" of form LHS → RHS; LHS & RHS consist of symbols, and LHS contains at least one non-terminal.

e.g., use the following grammar to generate some sentences:

$V = \{(,), E\}$

$T = \{(,)\}$

$S = E$

$P = \{E \to EE, E \to (E), E \to \varepsilon\}$

epsilon: empty string.

1. $E \Rightarrow \varepsilon =$ " "
2. $E \Rightarrow (E) \Rightarrow ((E)) \Rightarrow ((\varepsilon)) \Rightarrow (())$
3. $E \Rightarrow EE \Rightarrow (E)E \Rightarrow (\varepsilon)E \Rightarrow ()(E) \Rightarrow ()(\varepsilon) = ()()$

the language generated by grammar $G = (V, T, S, P)$, denoted $L(G)$, is the set of all strings of terminals that can be derived from S:

$$L(G) = \{ w \in T^* \mid S \overset{*}{\Rightarrow} w \}$$

"Kleene star"

set of all finite-length strings over non-terminals

w is derivable from S

e.g., what is an interesting characteristic of the language generated by this grammar?

$V = \{0, 1, A, B\}$ $T = \{0, 1\}$

$S = A$

shorthand for $A \to 0A$, $A \to 1B$, $A \to \varepsilon$

$P = \{A \to 0A \mid 1B \mid \varepsilon,\ B \to 0B \mid 1A\}$

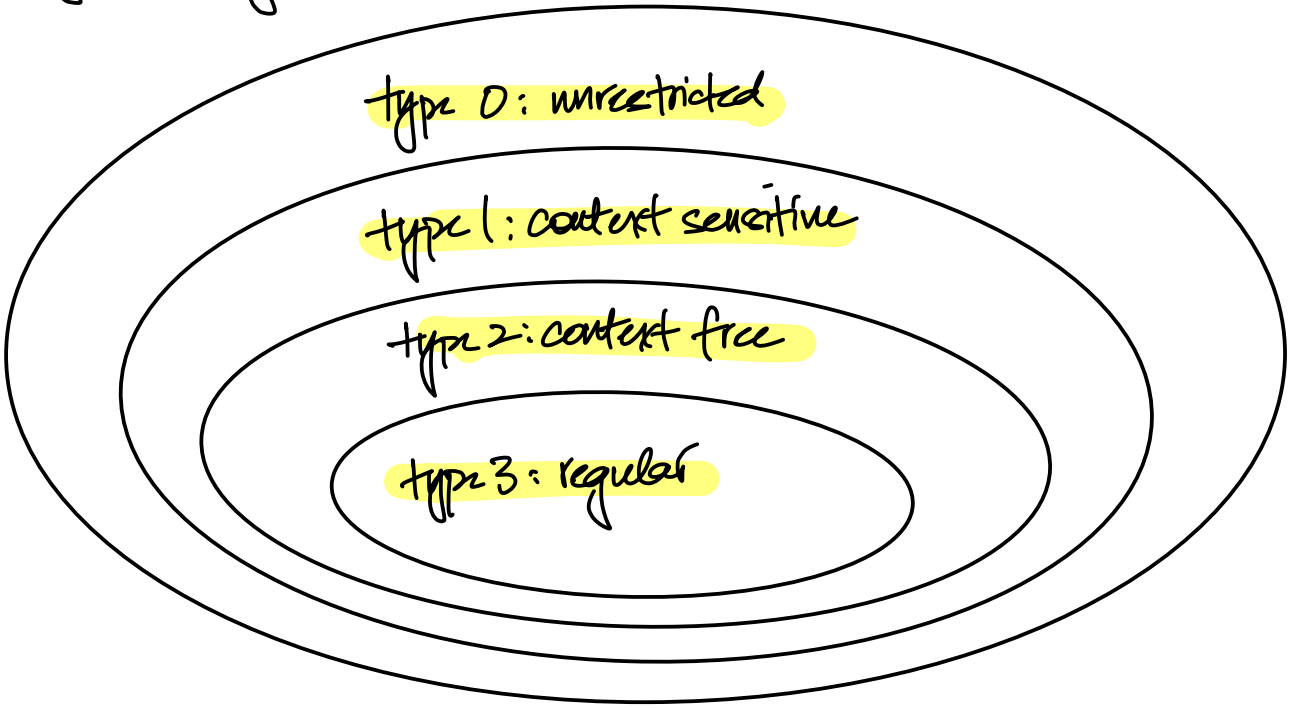
$A \Rightarrow 0A \Rightarrow 00A \Rightarrow 00$; $A \Rightarrow 1B \Rightarrow 10B \Rightarrow 101A \Rightarrow 101$;

$A \Rightarrow 1B \Rightarrow 11A \Rightarrow 111B \Rightarrow 1110B \Rightarrow 11100B \Rightarrow 111001A \Rightarrow 111001$

even # of 1s! "even parity"

Noam Chomsky created a classification system for formal grammars in 1956.

Chomsky hierarchy:



type 0 (unrestricted): no restrictions on grammar

type 1 (context-sensitive): productions have form $\alpha A \beta \to \alpha \gamma \beta$, where A is a non-terminal and α, β, γ are strings of terminals/non-terminals (γ non-empty).

len(LHS) ≤ len(RHS), and replacement of A up γ depends on context

type 2 (context-free): productions have form $A \to \alpha$, where A is a non-terminal and α is a string of terminals/non-terminals.

LHS has no context

type 3 (regular): [right regular]: productions have form $A \to aB$ or $A \to a$;
[left regular]: productions have form $A \to Ba$ or $A \to a$, where A, B are non-terminals, and a is a terminal symbol (or ε)

e.g., what types of grammars are the following?

G_1 : $V = \{(,), E\}$ $T = \{(,)\}$ $S = E$

$P = \{ E \to EE \mid (E) \mid \varepsilon \}$

string of non-terms (not regular)

no context.

G_1 is context free

G_2 : $V = \{0, 1, A, B\}$ $T = \{0, 1\}$ $S = A$

$P = \{A \to 0A \mid 1B \mid \varepsilon, B \to 0B \mid 1A\}$

form $A \to aB \mid a \mid \varepsilon$

G_2 is right regular

G_3 : $V = \{0, 1, 2, A, B, C\}$ $T = \{0, 1, 2\}$ $S = A$

$P = \{A \to 0BA2 \mid 012, B0 \to 0B, B1 \to 11\}$

G_3 is context-sensitive

LHS has context, and len(LHS) $\leq$ len(RHS)

e.g., what language is generated by G_3?

G_3: $V = \{0, 1, 2, A, B, C\}$ $T = \{0, 1, 2\}$ $S = A$

$P = \{A \to 0BA2 \mid 012,\ B0 \to 0B,\ B1 \to 11\}$

ex1: $A \Rightarrow 012$

ex2: $A \Rightarrow 0BA2 \Rightarrow 0B0BA22 \Rightarrow 0B0B01222$

$\Rightarrow 00BB01222 \Rightarrow 00B0B1222 \Rightarrow 000BB1222$

$\Rightarrow 000B11222 \Rightarrow 00011122$

$L(G_3) = \{0^n 1^n 2^n \mid n \geq 1\}$

0^n: string of n consecutive 0's

As an alternative to using a generating grammar to specify a language, we can design an automaton (abstract machine) that recognizes / accepts only those strings in a language.

— deciding whether we can / how hard it is to recognize strings of a given language is a major area of theoretical CS

(also: are languages recognized by different automata the same?)

languages (and the grammars that generate them) can also be classified by the automata capable of recognizing them:

finite state automata can recognize regular languages

pushdown automata can recognize context-free languages

linear-bounded automata can recognize context-sensitive languages

Turing machines can recognize languages generated by any formal grammar

a (deterministic) finite-state automaton (DFA) $M = (S, I, f, s_0, F)$ consists of:

S : a finite set of states

I : a finite set of input symbols

f : the transition function $f: S \times I \to S$

assigns a "next" state to every combination of state & input

$s_0 \in S$: the start state

$F \subseteq S$: the final/accepting states

We can represent DFAs using state-transition diagrams:

$S: \{s_0, s_1, s_2, s_3\}$

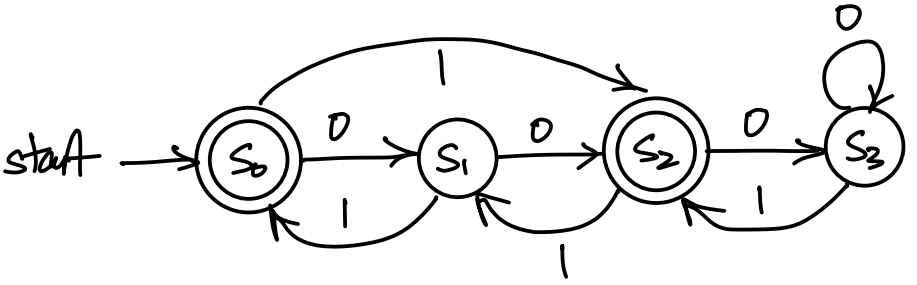
$I: \{0, 1\}$

$s_0 = s_0$

$F = \{s_0, s_2\}$

$f =$

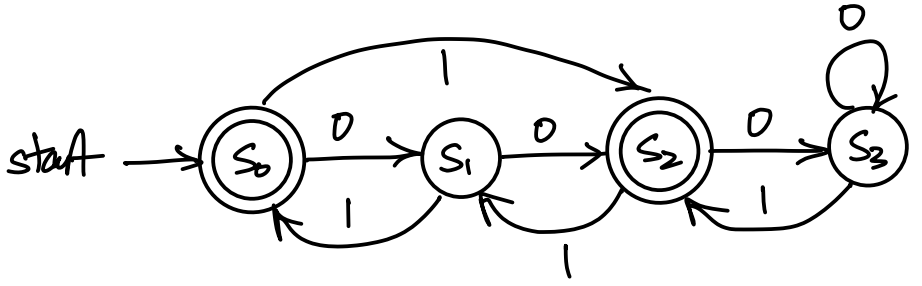
	0	1
s_0	s_1	s_2
s_1	s_2	s_0
s_2	s_3	s_1
s_3	s_3	s_2



a DFA $M = (S, I, f, s_0, F)$ can be run on an input string $i_1 i_2 \ldots i_n$, where $i_k \in I$ by starting at state s_0 and moving through subsequent states $t_1, t_2, \ldots, t_n$ where $t_j = f(t_{j-1}, i_j)$

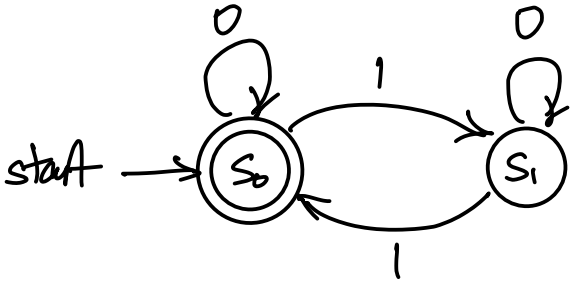
- an input string is accepted by the DFA if $t_n \in F$
- the language recognized by the DFA — $L(M)$ — is the set of all strings accepted by M
- two DFAs are equivalent if they recognize the same language.

e.g., does this DFA accept each of the following strings?



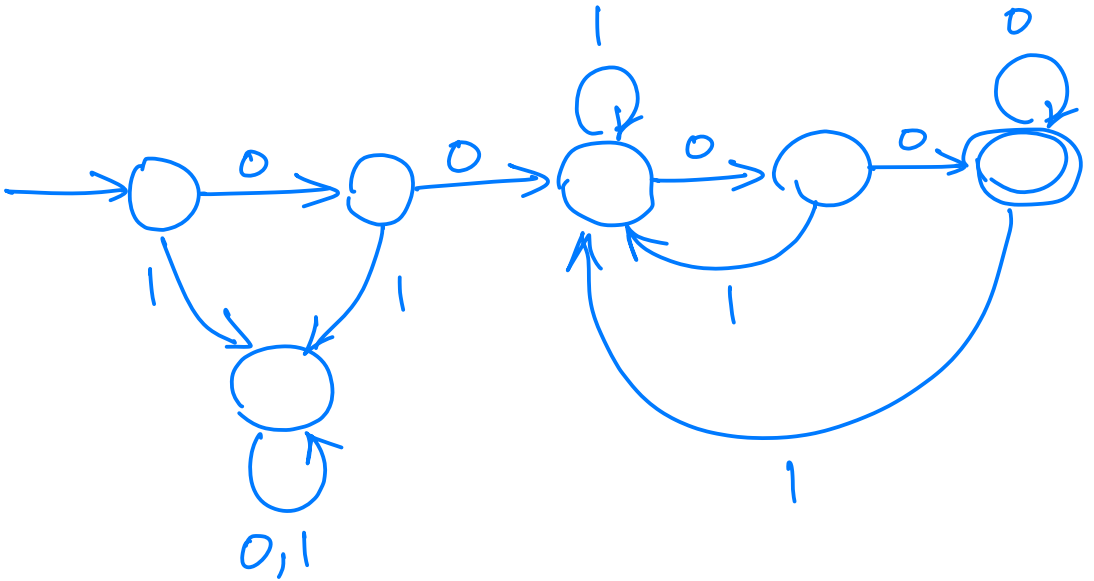
1. 01
2. 0001
3. 11101
4. 000000000
5. 1000001

e.g., what language is recognized by this DFA?



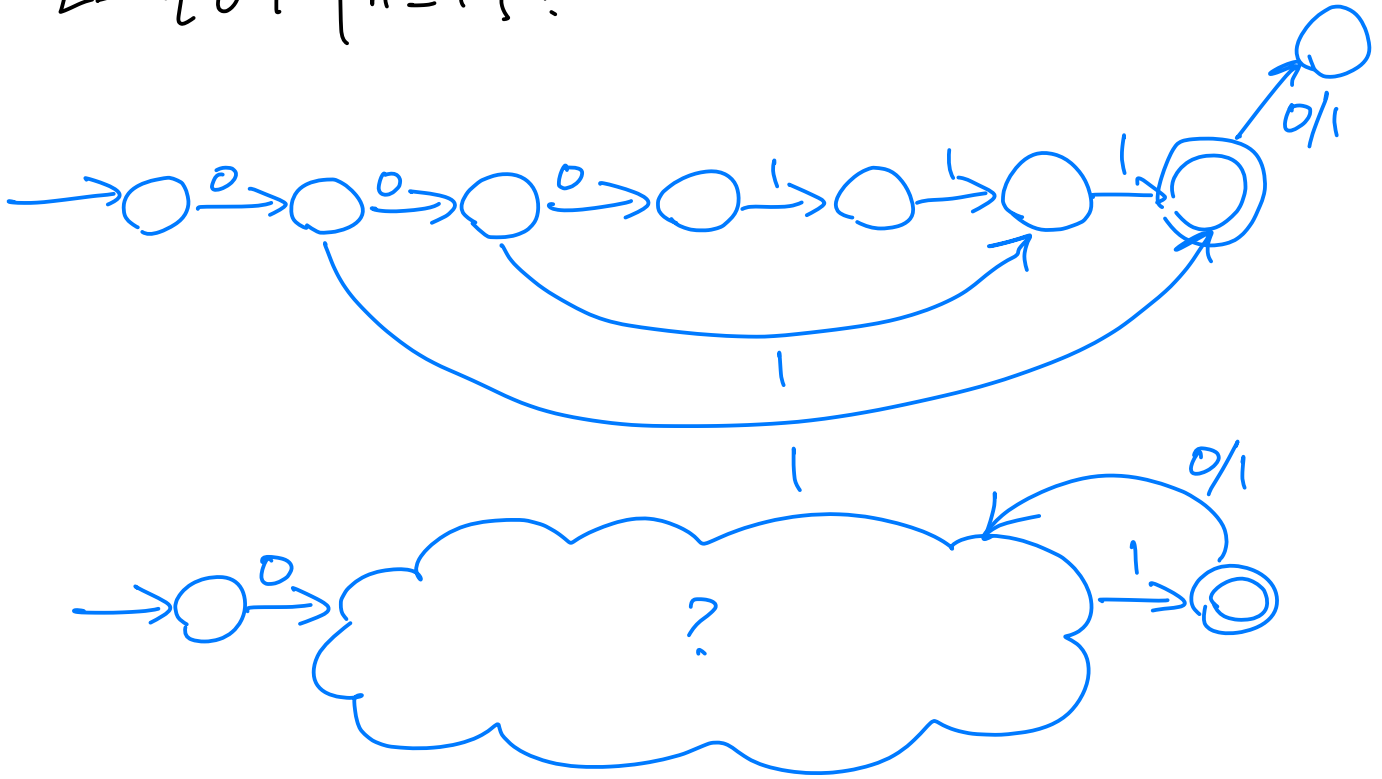
L(M) = set of all bit strings w/ even parity (even # of 1's)

e.g., construct a DFA for the language of bit strings that begin w/ two 0's and end w/ two 0's.



e.g., can you construct a ~~DFA~~ that recognise the language

$L = \{ 0^n 1^n \mid n \geq 1 \}$?



pumping lemma: for any language L that can be recognized by a DFA, we can find some integer p s.t. for any string $t \in L$, where $len(t) \geq p$ we can break t into substrings uvw where:

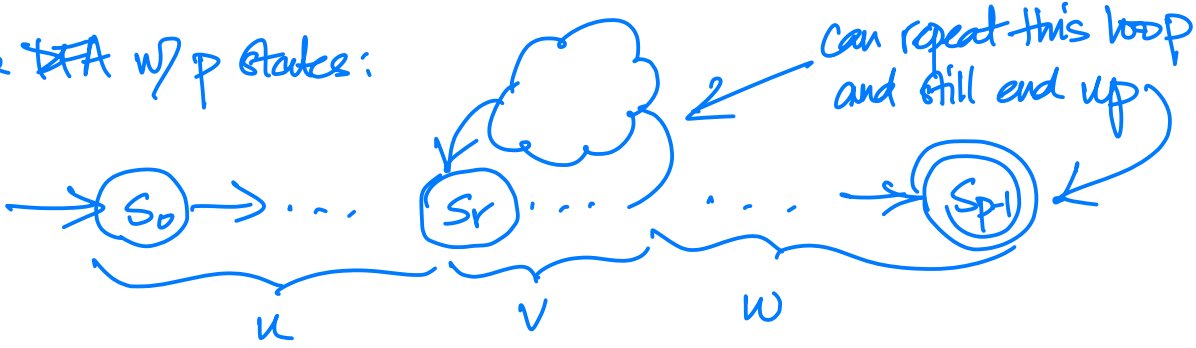
- $len(uv) \leq p$
- $len(v) \geq 1$
- $\forall n \geq 0, uv^n w \in L$

i.e., in a sufficiently long string, we can repeat some middle portion as many times as we want to get more strings in L

prove the pumping lemma.

- must show that for accepted string $x = uvw$ of total length p, v can be repeated to generate more accepted strings

- consider a ~~DFA~~ w/ p states:



- must be at least one repeated state! call it s_r
  - $s_0 \xrightarrow{*} s_r$ consumes u, $s_r \xrightarrow{*} s_r$ consumes v, and $s_r \xrightarrow{*} s_{p-1}$ consumes w

e.g., can you construct a ~~DFA~~ that recognise the language

$$L = \{ 0^n 1^n \mid n \geq 1 \} ?$$

by pumping lemma, if we have a sufficiently long string in L, we can repeat some middle portion as many times as we want to get more strings in L.

e.g. $\underbrace{000\ldots000111\ldots111}$

no middle portions can be arbitrarily repeated !

L is <u>not</u> recognizable by a ~~DFA~~. L is <u>not</u> a regular language.

to recognize more complex languages we need automatons w/ more memory + more freedom to read/write that memory

- pushdown automata are like DFAs, but can use a stack to help decide what transitions to take
- Turing machines can access an infinite amount of memory in arbitrary order to determine the progression through its states
  - most general model of computation!

That's all, folks!