On the menu today…

- jFLAP demo
- Regular expressions
- Pumping lemma
- Turing Machines
- Sections 12.4 and 12.5 in the text

jFLAP Demo

- jFLAP: Useful tool for creating and testing abstract machines
  - Finite automata, Turing machines
- Use in homework 5 (optional) and homework 6
- Download from class website

Regular Expressions

- **Definition of a <u>Regular Expression</u>**
  - R is a regular expression iff
    R is a string over $V \cup \{ \lambda, \varnothing, (,), \cup, * \}$ and R is:
    1. Some symbol $a \in V$, <u>or</u>
    2. λ, <u>or</u>
    3. $\varnothing$, <u>or</u>
    4. $(R_1 \cup R_2)$ where R_1 and R_2 are regular exps., <u>or</u>
    5. $R_1R_2 = R_1 \circ R_2$ where R_1 and R_2 are reg. exps., <u>or</u>
    6. $R_1{}^*$ where R_1 is a regular expression.
- **Precedence**: Evaluate * first, then ^{o}, then $\cup$
  - E.g. $0 \cup 11^* = 0 \cup (1^\circ (1^*)) = \{0\} \cup \{1, 11, 111, \ldots\}$

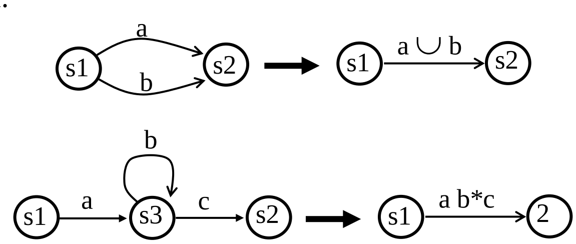
Regular languages (regular sets)

- A language is regular if it can be represented by a regular expression.
- Examples:
  L(R) = {w | w contains exactly two 0's}
  R = 1*01*01*
  L(R) = {w | w contains an even number of 0's}
  R = (1*01*01*)* 1*
  L(R) = {w | w is a valid identifier in C}
  R =
  $((A\cup\ldots Z)\cup(a\cup\ldots z)\cup \_)((A\cup\ldots Z)\cup(a\cup\ldots z)\cup(0\cup\ldots 9)\cup\_)^*$

L(R) = {w | w is a word in an Eminem song}

Regular Expressions and Finite Automata

- Kleene's theorem: A set is regular if and only if it is recognized by a finite state automaton.
- Proof: See Theorem 1 in Section 12.4 for proof.
  - ($\rightarrow$) Construct an NFA for each possible case in the definition: R = *a*, or R = λ, or R = $\varnothing$, or R = (R1 $\cup$ R2), or R = R1°R2, or R = R1*.
  - ($\leftarrow$) Main Idea:



A set is regular

⇔ it can be expressed using a regular expression

⇔ it can be recognized by a DFA

⇔ it can be recognized by an NFA

Some Applications of Regular Languages

- Pattern matching and searching:
  - E.g. In Unix:
    - `ls *.c`
    - `cp /myfriends/games/*.* /mydir/`
    - `grep 'Spock' *trek.txt`
- Compilers:
  - `id ::= letter (letter | digit)*`
  - `int ::= digit digit*`
  - `float ::= d d*.d*(λ|E d d*)`
  - The symbol | stands for "or" (= union)

Are there languages that are *not* regular?

- Is L = $\{0^n1^n \mid n \geq 0\}$ regular?
- Can you memorize the number of 0's encountered so far with a finite number of states?
- How do we prove L is not regular?

Beyond the Regular world…

- How do we prove a language is not regular?
- **Idea:** If a language violates a property obeyed by all regular languages, it cannot be regular!
  - **Pumping Lemma** for showing *non-regularity* of languages
  - See Example 6 in Sec. 12.4

I love ze pumping lemma!

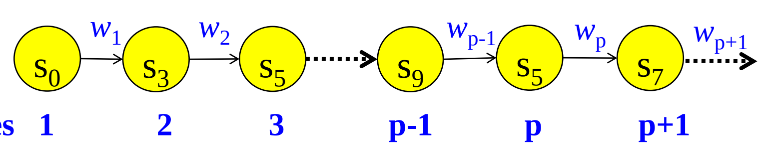
http://www.ipjnet.com/schwarzenegger2/pages/arnold_01.htm

The Pumping Lemma for Regular Languages

- **What is it?**
  - A statement ("lemma") that is true for all regular languages
- **Why is it useful?**
  - Can be used to show that certain languages are *not regular*
  - How? *By contradiction*: Assume the given language is regular and show that it does not satisfy the pumping lemma

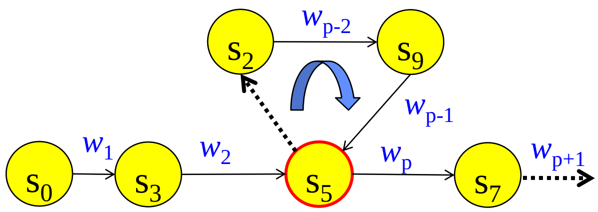
More about the Pumping Lemma

- **What is the idea behind it?**
  Any regular language L has a DFA M that recognizes it
  Suppose M has **p states** and accepts a string ***w*** of **length ≥ p**.



  - p transitions, p+1 states, i.e., a state must repeat within the first p symbols due to the pigeonhole principle
  - The sequence of states M goes through must contain a non-empty **cycle within the first p symbols**

More about the Pumping Lemma



- M goes through a non-empty **cycle within the first p symbols**
- Therefore, *all strings* that make M go through this cycle 0 or any number of times are also accepted by M and *should be in L*.

Pumping Lemma

- Let L be a regular language and let p = "pumping length" = no. of states of a DFA accepting L
- Then, any string *w* in L of length ≥ p can be expressed as *w* = *xyz* where:
  - *y* is not empty (*y* is the cycle)
  - $|xy| \leq p$ (cycle occurs within p state transitions), and
  - any "pumped" string xy^iz is also in L for all $i \geq 0$ (go through the cycle 0 or more times)

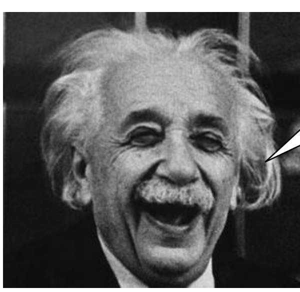
More details in Example 6 in Sec. 12.4 in text

Using The Pumping Lemma

- $L = \{0^n1^n \mid n \geq 0\}$ is not regular
- Proof by contradiction:
  1. Assume L is regular and let p be the pumping length given by the pumping lemma.
  2. Consider $w = 0^p1^p$ which is in L and has length $\geq$ p.
  3. Since $w = xyz$ and $|xy| \leq p$ and y is not empty, $y = 0^k$ for some $k > 0$.
  4. Then, $xy^2z = 0^{p-k}\,0^{2k}1^p = 0^p0^k1^p$ which is not in L.

  This contradicts the pumping lemma. Therefore, L is not regular.

Good news! Pumping lemma won't be in homeworks and final exam.



Just understand the basic idea and go over Example 6 in Section 12.4 in the text

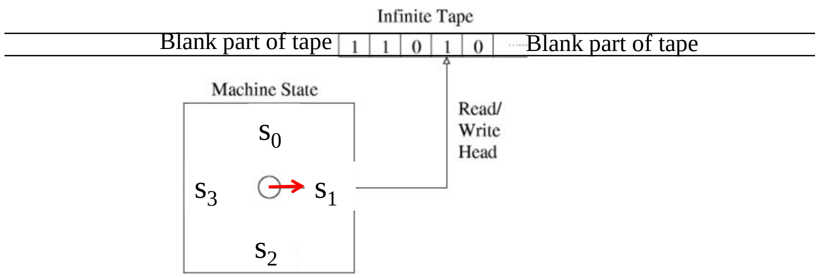
I’ll be back with da pumpin’ lemma.

If $\{0^n1^n \mid n \geq 0\}$ is not Regular, what is it?

Irregular??

Enter…Turing Machines

Turing Machines



Just like a DFA except with:

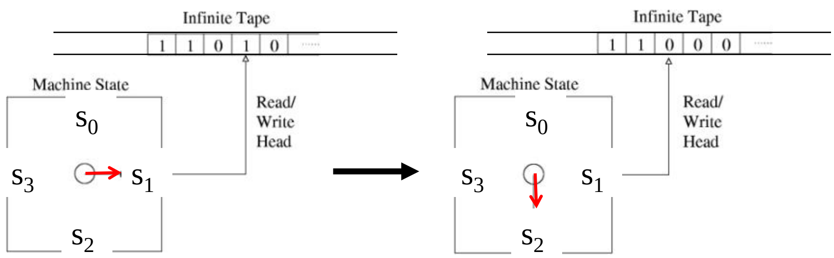
- Infinite "tape" memory (or scratchpad) on which you receive your input and on which you can do your calculations
- You can read one symbol at a time from a cell on the tape, write one symbol, then move the read/write pointer or head left (L) or right (R)

Who was Turing?

- Alan Turing (1912-1954): one of the most brilliant mathematicians of the 20^{th} century (one of the "founding fathers" of computing)
- Click on "Theory Hall of Fame" link on class web under "Lectures"
- Introduced the Turing machine as a formal model of what it means to compute and solve a problem (i.e. an "algorithm")
  - Paper: `On computable numbers, with an application to the Entscheidungsproblem, Proc. London Math. Soc. 42 (1936).`

How do Turing Machines compute?

- f(current state, symbol under the head) = (next state, symbol to write over current symbol, direction of head movement)



- 5-tuple representation: **(s_1, 1, s_2, 0, L)** (R = right, L = left)
- Turing machine "program" = set of such 5-tuples

Turing Machine (TM) Definition

- TM T = (S, V, I, f, s_0, F)
  - NOTE: We will use F and V in our definition of TMs; the textbook does not. Using V makes the input alphabet clear and distinct from tape alphabet I. Using F makes the final/accepting states clear.
- S, s_0, F are as in DFA definition
- Input strings are over an alphabet $V \subseteq I$.
  - TM can use other symbols in I as markers, etc. for computing.
  - Blank symbol □ is always in I (and not in V).
- f maps (state1, symbol1) to (state2, symbol2, direction)
  - f need not be defined for every (state,symbol) input
    - f is a "partial function"
  - If f not defined for a particular (state,symbol), TM halts.

Turing Machine (TM) Details

- Input string to TM given on tape
  - TM always starts on leftmost nonblank symbol
  - If no input, then can start on any cell of the tape
- TM can halt in two types of states:
  - TM halts and accepts the input iff it enters a final state in F
  - TM halts and rejects the input when it halts in any other state (when f is not defined for a (state, symbol) pair)
- TM *recognizes* a string *w* iff it halts in a final state for *w*
  - TM can reject *w* by halting in any non-final state or by looping forever!

Next Class:
Unsolvable problems…