Lecture 1: Introduction to Languages & Theory of Computation

CSCI 81
Spring, 2019
Kim Bruce
TA's:Gerard Bentley, Sarp Misoglu, Seana Huang, Danny Rosen, Alice Tan

Course web page: http://www.cs.pomona.edu/classes/cs101

New Course

- Wedding of
  - CS 81 (Logic & Theory of Computation)
  - CS 131 (Principles of Programming Languages
- 2/3 of 81 plus 1/3 of 131
  - Dropped logic from 81
  - Add language implementation issues from 131 tied to models of computation

Goals

- Describe and use formal systems to model real phenomena
- Recognize language classes (e.g., regular, cfl, decidable, r.e.), their gaps, and why important.
- Determine in which classes a language is contained.

Goals

- Understand the equivalence of generators and recognizers of languages (e.g. regular expressions and finite automata, cfgs and pdas, r.e. languages and Turing machines).
- Understand the use of formal specifications (e.g., context-free grammars) to derive algorithms to parse, type-check, and/or interpret languages, and be able to implement some of these in a functional language.

More Goals

- Understand the Church-Turing thesis and several models of universal computation.
- Understand and be able to show that a variety of interesting problems involving computation are undecidable.

Syllabus

- First time this course is offered
- More interesting combination of theory & practice — see where and how theory used
- You have different backgrounds:
  - CS 54/55, Math 55/103
- Let me know how course is going.
  - Expect to revise syllabus on the fly

Administrivia

- Prereqs:
  - CSC054 *or*
  - {CSC 052 and {CSC 055 or HMC/CMC Math 055 or Math 103}}.
- Corequisite: CSC 062/070

Reading/Homework

- Skim chapter ahead of lecture
  - Work practice problems after lecture
  - Thus skim section 5.4 and work problems 2.7, 4.4 before lecture this Thursday
- Homework due Wednesdays at midnight

Outline

- Finite State Machines & Regular Expressions
- Haskell Programming / Lexical Analysis
- Context-free grammars and parse trees
  - push-down automata
  - parsing programming languages
- Turing Machines & Undecidability
- Formal semantics and interpreters

Required Texts

- Rich: Automata, Computability, & Complexity
  - Advantages: Informal, readable
  - Disadvantage: Few proofs (except in index). Provide algorithms, but not show correct. We usually will!
- Any Haskell text, e.g.
  - *Learn you a Haskell for Greater Good* by Miran Lipovaca *or*
  - *Real World Haskell* by O'Sullivan, Stewart and Goerzen
  - *Both available for free on-line*

Slides

- Will generally be available before class
- Designed for class presentation, not for complete notes
- Will need to take notes (perhaps on slides)
- *No laptops or other electronics open in class*
  - If that is problem, come see me.

Homework

- Due Wednesday night
  - Turn in electronically
  - Use LaTeX
  - See LaTeX tutorials on "Links to useful info" page
- Daily homework:
  - Not to be turned in: do in groups. Should prepare you for real homework.
- Weekly homework
  - See Academic Honesty statement. Must write up on own. Document any collaborations.

Questions?

Modeling Computation

- Simple example: Vending machine taking only nickles, and quarters.
  - Only dispenses one item, that costs 30 cents
  - When amount deposited is at least 30 cents, dispenses item and correct change.
  - Want to model each possible state of machine as customer uses it.

General Model

- Results of actions depend on current state
  - What happens when hit return key in an application?

Alphabets & Strings

- An alphabet Σ is a finite non-empty set whose elements are called symbols or characters.
- A string (or word) over Σ is a finite sequence of elements of Σ.
  - The empty string is denoted ε.
- The length of a string is the # of letters in it.

Languages

- Σ^k is set of strings of length k
  - $\Sigma^0 = \{\varepsilon\}$
  - $\Sigma^{k+1} = \{xa \mid x \in \Sigma^k, a \in \Sigma\}$
  - Set of all strings over Σ is written $\Sigma^* = \cup_{k\geq 0} \Sigma^k$
- A language is a set of strings over Σ. *Examples*:
  - strings over {0,1} with more 1's than 0's.
  - strings over {0,1} that do not contain three 0's in a row
  - $\{0^n1^n0^n \mid n \geq 0\}$
  - Set of legal Java programs

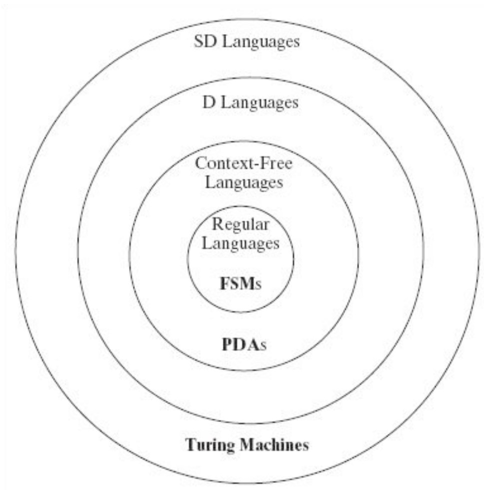
String Operations

- Concatenation of s, t is written s ‖ t or just st.
  - ε is identity: $\varepsilon s = s = s\,\varepsilon$
- String replication:
  - $w^0 = \varepsilon$
  - $w^{i+1} = w^i\, w$
- Reversal of w, w^R, has inductive definition:
  - $\varepsilon^R = \varepsilon$
  - if a is letter, $(ua)^R = a(u^R)$

Language Classes

Languages	**Machines Accepting**
Regular	Finite State Machine
Context-free	Pushdown Automata
Decidable	Turing Machines*
Semi-Decidable	Turing Machines

Language Classes



Regular Languages

- Can be characterized in 3 ways:
  - Languages accepted by finite state machines
  - Languages represented by regular expressions
  - Language generated by regular grammars

Deterministic Finite State Machine

- A FSM (or DFSM) is a quintuple $(K, \Sigma, \delta, s, A)$
  - K is a finite set of states
  - Σ is a finite input alphabet
  - $s \in K$ is the start state
  - $A \subseteq K$ is set of accepting (or final) states
  - $\delta: K \times \Sigma \rightarrow K$ is transition function
- Simple model of real computer
  - finite memory

Example

State Machines

- Responses depend on states
  - Car radio or remote control for TV
    - On FM mode, next takes to next station
    - On CD mode, next takes to next track
  - See "State Pattern" for OO languages
- Could also design "transducers"
  - Take action when entering or leaving state
- Lexical scanners for programming languages
  - Recognize identifiers / numbers

Other Real Examples

- Physical examples:
  - Combination lock on a safe
  - Vending machine
  - Traffic light
  - Elevator
- More esoteric:
  - Network protocol states

Terminology

- Configuration: $(q,w) \in K \times \Sigma^*$
  - snapshot of current state of a computation
  - q is current state, w is input still to be read
- Initial configuration is (s,w) where s is start state and w is input.

Computations

- Single step of M uses δ to process next character:
  - $(q_1,cw) \vdash_M (q_2,w)$ iff $\delta(q_1,c) = q_2$
- $\vdash_M^*$ is reflexive, transitive closure
  - $(q_1,u) \vdash_M^* (q_2,w)$ means get from first to second in 0 or more steps

Defining Language

- M *accepts* string w iff
  there is $q \in A$ s.t. $(s,w) \vdash_M^* (q,\varepsilon)$
- M *rejects* string w iff
  there is $q \notin A$ s.t. $(s,w) \vdash_M^* (q,\varepsilon)$
- $L(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \}$
- L is regular if it is $L(M)$ for some finite state machine M

Examples

- $L_0 = \{w \in \Sigma^* \mid w \text{ contains aab as a substring}\}$
- $L_1 = \{w \in \Sigma^* \mid w \text{ contains at least two b's}\}$