Closure Properties of CFLs; Introducing TMs

CS154

Chris Pollett

Apr 9, 2007.

Outline

- Closure Properties of Context Free Languages
- Algorithms for CFLs
- Introducing Turing Machines

Closure Properties of CFL

- CFL are closed under union.
  - Proof idea: let G and H with start symbols S and T respectively, be two CFGs for the CFL's we want to take the union of. Make a new grammar with the same alphabet, with the union of the two grammars productions (after renaming) together with the new rules S´->S|T where S´ is the new start variable.
- CFLs are closed under intersection by regular languages.
  - Proof idea: use the cartesian product construction on a PDA for the CFL together with a DFA for the regular language.
- CFLs are not closed under intersection or complementation
  - Proof: The languages $\{a^n b^n c^m \mid n, m \geq 0\}$ and $\{a^m b^n c^n \mid n, m \geq 0\}$ are bothe context-free. Their intersection is $\{a^n b^n c^n \mid n \geq 0\}$ which is not by the pumping lemma. They are not closed under complementation since using deMorgan rules, intersection can be defined from union and complementation.

Algorithms for CFLs

- We have already given an algorithm (CYK) for checking if a string is in a CFLs.
- We have also given algorithms for checking for useless variables in grammars, as well as, for eliminating λ-rules from grammars.
- We now claim there is an algorithm, which given a grammar G written down formally as a 4-tuple, can decide whether or not L(G) is empty.
  - To do this, the algorithm check is the start variable is useless. If it is we know the language is empty; otherwise it is not.
- We also claim that there is an algorithm to check given G whether or not L(G) is infinite.
  - To do this, we first eliminate λ-rules, unit-productions, and useless symbols from G. We then construct a graph where (A,B) is an edge for two variables A,B in the graph iff A-->xBy for some production in G. If there is a cycle in this graph then C=>* uCv for some variable C in the original grammar. As there are no useless symbols in this grammar, we must have C=>*z, for some string z of terminals. Hence, also, S=>* sCt =>* szt, S=>* sCt =>* suCvt =>* suzvt, etc. Thus, one can argue there is a cycle in the graph iff G's grammar is infinite.

General Models of Computation

- So far we have looked at machines that either have bounded memory or access to memory limited to stack operations.
- We would like to consider models of computation which correspond to general purpose computers.
- In 1936, Alan Turing presented such a general model of a computer now called a Turing Machine.
- In this model, the machine has a finite control and a arbitrarily long tape of data consisting of squares able to hold one symbol. The machine also has a read head which can read one square at a time. Initially, this tape is blank except for the n first squares under and to the right of the tape head which have the input. The machine in one step is allowed to read what's under its tape head, write a new symbol, move left or right one square and change its state.
- The machine has special states which cause it to halt. The contents of the tape when these states are entered is the output of the machine.
- The first actual computer developed for code-breaking during World War II was actually partially based on his model.
- It turns out this model is actually equivalent to what can be done on modern computers.

Example

- Let B be the language {w#w | w is a string over 0,1}. This language is not context free.
- A Turing Machine M that could accept this language might operate as follows:
  On input w:
  - Zig-zag across the tape to the corresponding positions on either side of the # symbol to check whether these positions contain the same symbol. If they do not, or if no # is found do into the reject state. If they have the same symbol change the symbol to a new symbol X.
    When all the symbols on the left side of the # have been X'd out, check if there are any more symbols to the right of the #. If yes reject; if not accept.
  - By accept or reject, I mean we view the halting states as further partitioned into those which are accepting and those which are rejecting and we enter the appropriate kind of halting state.

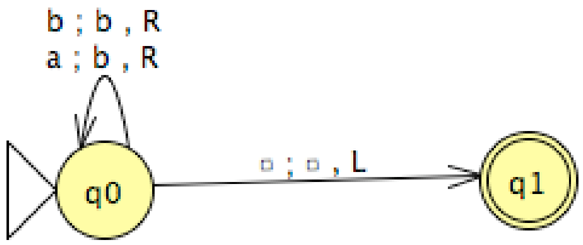
Formal Definition of Turing Machine

- A **Turing Machine** (TM)s a 6-tuple (Q, Σ, Γ, δ, q_0, H) where
  1. Q is a finite set of states
  2. Σ⊆Γ are respectively the input and tape alphabets. Γ contains a space symbol '_' not in Σ.
  3. δ:Q x Γ --> Q x Γ x {L, R} is the transition function. (JFLAP allows L,R,S where S is stay put)
  4. q_0 ∈ Q is the start state.
  5. H ⊆Q is the set of Halting states of the machine
- A TM receives its input w=w_1w_2 … w_n. on the n squares of its tape under and to the right of the tape head. The input is not allowed to contain blanks. The tape has arbitrarily many tape square to he right and to the left of the starting tape head location. Except for the input, the rest of the tape squares have the blank symbol. Once M starts, it follows the rules prescribed by the transition function. A rule

(q,a)-->(q´,b, L) says in state q reading an 'a' go to state q´, write a 'b' in the current square then move the tape head left. (q,a)-->(q´,b, R) would say the same thing except moves the tape head right.

- Computation continues until the machine enters a halt state.

Diagrams and Examples



- As you can see above, there is a diagrammatic notation for Turing Machines similar to that for PDA and DFAs.
- The states are drawn as circles; two concentric circles indicates a halt state (the book calls these final states).
- Each transition is labeled with a triple x;y, D; where x is supposed to be the symbol being read, y is the symbol to write, and D is the direction to move.
- For example, the above machine when started on a string over the alphabet {a,b} converts it to a string of the same length consisting of only b’s, then stops on the rightmost character of the output string.

Configurations, Yields

- To specify the state of a computation at a given time (i.e., a **configuration**) one needs to specify the tape contents, the head position, and the current state.
- To do this it suffices to consider only the non-blank squares.
- One can use the notation u q v to represent this information. Here u is a string that represents the contents of the tape to the left of the tape head, q is the current state and v is a string consisting of what is under the tape head followed by the non-blank symbols to the right of the tape head.
- For example, one might have $0011q_7 1100$. This says the tape contents are 00111100, the machine is in state q_7, and it is read the third 1 in this string.
- We say configuration C **yields** C´ if the TM can legally go from C to C' in one step. For instance, ua q bv yields u q´ acv if $\delta(q,b) = (q´, c, L)$.

Accept, Reject, Recognize, Decide

- The **start configuration** of M on input w is the configuration q_0 w.
- An **accepting configuration** is one in which the state of the configuration is q_{accept}, where q_{accept} is some state in H.
- A **rejecting configuration** is one in which the state of the configuration is q_{reject} where q_{reject} is some state in H.
- A Turing machine M **accepts** w if a there is a sequence of configurations C_1, C_2, .., C_k such that C_1 is the start configuration of M on w; for i between 1 and k-1, C_i yields C_{i+1} and C_k is an accept configuration.
- The collection of strings M accepts is denoted L(M) and is called the **language recognized by M**.
- A language is **Turing-recognizable** if some TM recognizes it.
- A language is called **decidable** if there is some TM which halts on all inputs which recognizes it.