Automata Final

tyang27

May 2019

1 Regular Languages

1.1 Deterministic Finite Automata

$M = (Q, \Sigma, \delta, q_0, F)$

- Q = set of states
- Σ = alphabet, set of symbols
- $\delta : Q \times \Sigma \to Q$ = transition function, maps states, symbols to new states
- $q_0 \in Q$ = start state
- $F \subseteq Q$ = set of final states

1.2 Nondeterministic Finite Automata

$M = (Q, \Sigma, \delta, q_0, F)$

- Q = states
- Σ = alphabet
- $\delta : Q \times \Sigma_\varepsilon \to P(Q)$ = transition function, maps states, symbols/varepsilon to set of new states
- $q_0 \in Q$ = start state
- $F \subseteq Q$ = set of final states

Modified delta transition allows us to 1) varepsilon jump to states, 2) be at multiple states at once, and 3) have dead states

1.3 Regular Expressions

Base cases:

- $a \in \Sigma$
- ε
- $\emptyset$

Operations:

- $R_1 \cup R_2$
- $R_1 \circ R_2$
- R_1^*

1.4 Converting between DFA/NFA/RE

1.4.1 DFA to NFA

Trivial

1.4.2 NFA to DFA

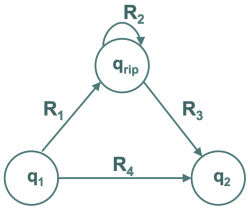
Subset construction

- $Q' = P(Q)$ - NFA states are subset of DFA states
- $\Sigma' = \Sigma$
- $\delta'(R, a) = \bigcup_{r \in R} E(\delta(r, a))$ if $R \in Q'$ - For each NFA state (a subset), we follow the DFA transition of each element in the set (a new subset). Apply varepsilon transition to the result of original transition.
- $q_0' = E(\{q_0\})$
- $F' = \{R \in Q' | \exists r \in R, r \in F\}$ - Final NFA states contain some original final state

1.4.3 NFA to RE

Ripping method using GNFA.

1. Add new start state that varepsilon jumps to original start state
2. Add new final state that all original final states varepsilon jump to
3. Replace commas with unions

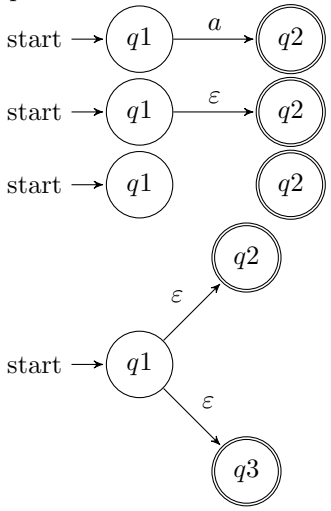


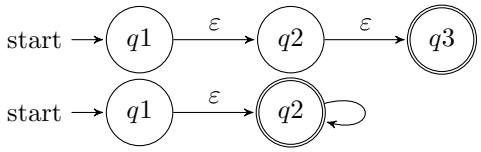
4. 

5. Rip rip rip! Enumerate paths that go through q_{rip} and link the two connections using the formula above.

1.4.4 RE to NFA

Use process similar to induction.





1.5 Closure

1.5.1 Union using NFA

- $Q = Q_1 \cup Q_2 \cup \{q_{new}\}$
- $\Sigma = \Sigma_1 = \Sigma_2$
- $\delta(r,a) = \begin{cases} \delta_1(r,a) & \text{if } r \in Q_1 \\ \delta_2(r,a) & \text{if } r \in Q_2 \\ \{q_{01}, q_{02}\} & \text{if } r = q_{new}, a = \varepsilon \\ \emptyset & \text{if } r = q_{new}, a \neq \varepsilon \end{cases}$
- $q_0 = q_{new}$
- $F = F_1 \cup F_2$

1.6 Pumping Lemma

1.6.1 Formal

If A is a regular language, then there exists an integer p where if s from A is any string of length at least p, then s may be divided into three pieces, $s = xyz$, such that:

- $\forall i \geq 0$, $xy^iz \in A$.
- $|y| > 0$
- $|xy| \leq p$

1.6.2 Informal

Given a regular language A, the pumping lemma holds. The pumping lemma says that if we have a long string (a string with length greater than the number of states), we know that there must be some loop in the machine, and we can partition the machine into three pieces, xyz, where y is the loop.

- If we pump the loop 0 or more times, it is still in the language.
- The loop exists, in the sense that it has at least one state.
- The leadup to the loop and the loop combined fit into the machine.

1.7 Proving nonregularity

FSOC, assume that A is regular. Then, the properties of pumping lemma apply, enumerate. Consider a long string $s = xyz$ (choose s) with an arbitrary partition up to p that satisfies pumping lemma. If we pump it more or less (choose i), then it is no longer in the language.

1.8 Other notes

- To show that reversed strings are closed under regular languages, we can just swap the arrows in a DFA, but cannot do so in NFA. This is because NFA has dead states.

2 Context Free Languages

2.1 Properties

- Ambiguous - string derived from grammar in fundamentally different ways. Show that either 1) give different parse trees or 2) different leftmost derivations.

2.2 Context Free Grammar

$G = (V, \Sigma, R, S)$

- V - variables, finite set of symbols, typically caps
- Σ - terminals, finite set of symbols, disjoint from variables ($V \cap \Sigma = \emptyset$), typically lowercase
- R - finite set of rules, e.g. $A \to B$
- $S \in V$ - start symbol

Start with S, derive by repeating rules until no more variables, only terminals.

2.2.1 Chomsky Normal Form

Equivalent to CFG. Basically, CFG with certain conditions. $2n - 1$ to derive a string of n. Conditions:

- $A \to BC$
- $A \to a$
- B, C cannot be S
- Only S can go to ε

2.3 Converting from RE to CFG

- $a = S \to a$
- $\varepsilon = S \to \varepsilon$
- $\emptyset = S \to S$
- $R_1 \cup R_2 = S \to S_1 | S_2$
- $R_1 \circ R_2 = S \to S_1 S_2$
- $R_1^* = S \to S_1 S$

2.4 Push Down Automata

$P = (Q, \Sigma, \Gamma, \delta, q_0, F)$

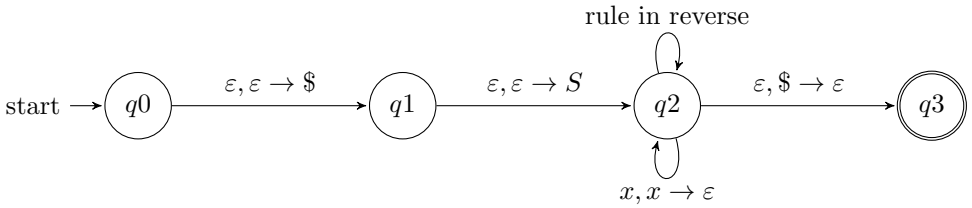
- Q = finite set of states
- Σ = alphabet, finite set of symbols
- Γ = stack alphabet, finite set of symbols
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to P(Q \times \Gamma_\varepsilon)$ = transition function, e.g. $a, b \to c$
- q_0 = start state
- $F \subseteq Q$ = set of final states

2.4.1 Useful transitions

- symbol a, pop $b \rightarrow$ push c = take arrow if symbol in string is a and top of stack is b. Pop b off stack and push c.
- $a, \varepsilon \rightarrow \varepsilon$ = consume symbol, ignore stack
- $\varepsilon, \varepsilon \rightarrow b$ = consume no symbol, push onto stack
- $\varepsilon, b \rightarrow \varepsilon$ = consume no symbol, pop off of stack
- $\varepsilon, \varepsilon \rightarrow \varepsilon$ = consume no symbol, ignore stack

2.5 Converting from CFG to PDA

- The general idea is that we place an empty marker on the stack and on top of that, the reverse of all possible generations. We put rules that consume the input $x, x \rightarrow \varepsilon$ to check if we can uncover the \$, meaning that it is a valid string.

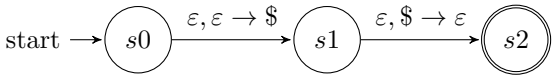


2.6 Converting from PDA to CFG

A bit too involved, but know that it is possible.

2.7 Other notes

- Stacks lack notion of emptiness, so often use \$ symbol emptiness



3 Context-free languages

3.1 Pumping lemma for context-free languages

If A is a context-free language, then $\exists p$, the pumping length. If s is a long string in A ($s \in A$ and $|s| > p$), then s may be divided into five pieces $s = uvxyz$ s.t.

- $\forall i \geq 0, uv^i xy^i z \in A$
- $|vy| > 0$ (note, v or y can be the empty string, but not both)
- $|vxy| \leq p$

If a string of terminals is longer than the number of rules in Chomsky Normal form (informally), we know that a variable is repeated, and we can replace the variable inside with the one outside recursively. Pumping once, we get $uvvxyyz$.

3.2 Showing nonCFLarity

Assume, FSOC, that A is a CFL. Let p be the pumping length given by PL for CFLs. Consider a string $s = uvxyz$ (e.g. $s = a^p b^p c^p \in A$) with at least length p. State some property about what v and y could be (e.g. we know that xyz cannot contain more than two different types of symbols). Pump up, so the number of occurrences of at least one of the symbols is not changing, meaning that the number of occurrences of one type of symbol is still p, while the number of occurences of the other symbol is now $p + k$ for some k.

3.3 Closure

- Regular operators: union, concatenation, kleene star, intersection, complement
- Context free operators: union, concatenation, kleene star
- Turing decidable: union, intersection, concatenation, complement, and kleene star
- Turing recognizable: union, intersection, concatenation, and kleene star

4 Turing Machines

4.1 Church Turing Thesis

- Algorithms and turing machines are basically equivalent.

4.2 Showing Turing Decidable

- Guaranteed to halt on accept/reject.
- By given, want, construction, correctness. Make sure that our machine terminates.

4.3 Showing Turing Recognizable

- Guaranteed to accept, but may loop.
- By given, want, construction, correctness.
- Make sure that inner loops terminates in finite steps, but outer loop can be infinite. E.g. use shortlex ordering up to a certain amount of steps.

4.4 Co-Turing Recognizable

- The complement is Turing Recognizable.
- If Co-TR and TR, then TD.

5 Undecidable languages

- $A_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M(w) \text{ accepts}\}$, TR, but not co-TR
- $HALT_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M(w) \text{ halts}\}$
- $EMPTY_{TM} = \{\langle M\rangle | M \text{ is a TM and } L(M) = \emptyset\}$

6 Reduction

- $A \leq_m B$: If B is decidable, then A is decidable. If A is undecidable, then B is also undecidable.
- $A_{TM} \leq_m HALT_{TM}$: FSOC, assume that the halting problem is decidable by D_{HALT}. Then we can use that to create A_{TM}. Run D_{HALT} on M_{IN}. If it halts, reject. Otherwise, run M_{IN} on w_{IN}. However, we know that A_{TM} is undecidable, so $HALT_{TM}$ must also be undecidable.
- $EMPTY_{TM} \leq_m A_{TM}$: FSOC, say that the emptiness problem is decidable by D_{EMPTY}. Then we can use that to create A_{TM}. Construct a new machine such that if the string is not w_{IN}, reject. Otherwise, run $M_{IN}(w_{IN})$ and output what it outputs. Then, if the emptiness decider accepts, we know that the machine does not accept the string or the string is not correct. If it rejects, there must be a single item in it, so we know that set is not empty, so it must accept a string.

- $ALL_{CFG} \leq_m EQUAL_{CFG}$: FSOC, say that the emptiness problem is decidable by D_{EQUAL}. Then, we can use that to create $EQUAL_{CFG}$. We construct a CFG that accepts everything. If we run D_{EQUAL} on our constructed CFG and G_{IN}, and it accepts, we know that G_{IN} accepts everything.
- $ALL_{CFG} \leq_m \{\langle G_1, G_2\rangle |$ are CFGs and $L(G_1) = \overline{L(G_2)}\}$: Similar to above. Instead, we construct a CFG that accepts nothing as our G_2, and if the decider accepts, we know that it accepts everything
- $A_{TM} \leq_m REG_{TM}$: Create a machine that runs $M(w)$. If it accepts, accept nonregular strings of the form 0^n1^n, otherwise reject everything. Then, if we run REG_{TM}, deciding regularity also tells us if w was accepted, since we know that if $M(w)$ accepts, the language is nonregular, and if $M(w)$ rejects, we would have an empty set, which we know is regular.
- The way that you want to think about this is, if A reduces to B, FSOC, say that we have a decider D for B. How can we create a machine that produces a language that could be used on D to answer A?

6.1 Template for $A \leq_m B$

- Say that we know that A is unTD
- FSOC, say that we have a TM decider D that decides B.
- Want a construction for TM L that answers A s.t. if input is in A, L accepts, else L rejects
- Construct inner TM I where on input x, may or may not ignore x or w to create a set that is compliant with B and answers A. For A_{TM}, we can create a set based on $I(w)$'s output, since we never even run the machine. Run D on I.
- Correctness by looking at the properties of the language produced and seeing that if an input is in A, then our machine I will accept. If an input is not in A, then our machine I will reject.
- However, D does not exist, so we have reached a contradiction.
- Synonymous for TR

7 Mapping Reduction

- $A \leq_m B$, $f : \Sigma^* \to \Sigma^*$ s.t. $w \in A$ iff $f(w) \in B$
- If $A \leq_m B$ and B is TD/TR, then A is TD/TR (B is a harder problem and you can solve B)
- If $A \leq_m B$ and A is not TD/TR, then B is not TD/TR
- Similar in spirit to the normal reduction template, just output your input instead of pretending there's a decider and running it.

8 Complexity and P and NP

- Every multitape $T(n)$ has an equivalent $O(T(n)^2)$ single tape.
- Every $T(n)$ nondeterministic has an equivalent $O(2^{T(n)})$
- P: polynomial in deterministic single tape
- NP: polynomial in nondeterministic single tape, or verifiable in polynomial time