CSE105 Midterm 2 Review Doc

- The second exam for this class is on Wednesday May 23. The exam covers sections 1.4, 2.1, 2.2, 2.3, 3.1, 3.2, 3.3, 4.1 of Sipser.
- 5 x 8 inch index card

Topics: *Everything after midterm 1*

- DFA, NFA, Regular language and expressions
- **Pumping lemma**
- Finite/infinite
- **CFG, CFL Push-Down Automata**
- **Non-regular languages**
- **Turing Machine, high-level/implementation-level, recognizable/decidable**
- **Closure under operations of different languages**

1.4 Pumping Lemma

- Regular language ({w has equal number of 01's and 10's}) Vs Non-regular language ({w has equal language of 0s and 1s})

- **Pumping Lemma** (p. 78)
  If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, s = xyz, satisfying the following condition:
  - For each $i \geq 0$, $xy^iz \in A$,
  - $|y| > 0$, and
  - $|xy| \leq p$

- Proving pumping lemma: Assign p as the number of states in DFA. If all strings length < p, PL true for strings ≥ p. If s in A has length at least p, apply pigeonhole principle.

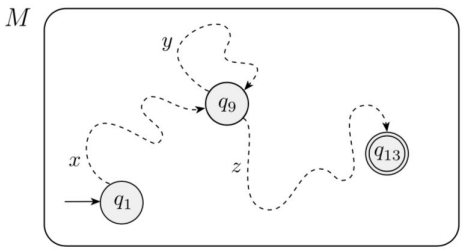


FIGURE **1.72**
Example showing how the strings x, y, and z affect M

- **Proving non-regularity (Pumping lemma can only be used to prove non-regularity)** Assume B is regular. Use PL to guarantee the pumping length p. Find string s where |s| ≥ p, but cannot be pumped. Demonstrate in *all cases* s cannot be pumped. Contradicion.

- Examples
  - B = $\{0^n1^n \mid n \geq 0\}$
    Assume L is regular. Let p be the pumping length. For string s = 0^p1^p, s should be able to be pumped. Let s = xyz. Since |xy| ≤ p, y should be all 0's. In this case, xy^iz for i > 1 doesn't belong to language L. Therefore, by contradiction L is not regular.
  - C = { w | w has equal number of 0s and 1s }
    C ∩ 0*1* = B. Regular language closed under intersection, but B is non-regular.
  - D = $\{0^i1^j \mid i > j\}$ s = $0^{p+1}1^p$ = xyz. Y consists of only 0's. S = xz not in D.

- **Find string that exhibits the ESSENCE of non-regularity.**

2.1 Context free grammar

- A context free grammar is a (V, ∑, R, S) where
  - V is a finites set called Variables
  - ∑ is disjoint finite set from V called Terminals
  - R is the finite set of rules
  - S is Start variable [only one]

- Derivation: sequence of substitutions to obtain a string.
  - uAv **yields** uwv. U **derives** v.

- GFG construction
  - CFLs are unions of simpler CFLs
  - Convert DFA to CFG
    - Make variable R_i for each state q_i of the DFA
    - Add the rule $R_i \rightarrow aR_j$ if $(q_i, a) = q_j$.
    - Add the rule $R_i \rightarrow$ empty string if q_i is an accept state.
    - Make R_0 where q_0 is the start state
  - Contain strings with two substrings are linked
    - $R \rightarrow uRv$
  - Recursive structure
    - Any time symbol *a* appears, an entire parenthesized expression appear recursively. Place variable symbol generating the structure in the location where structure may recursively appear.
- Ambiguity
  -

- If a grammar can generate a string in multiple ways. Different parse trees, but not different derivations (differ in the order they replace variable)
- **Leftmost derivation**
  - At every step, the leftmost remaining variable is the one replaced
- **A string is derived ambiguously**
  - If in some CFG it has n leftmost derivations, n ≥ 2

2.2 Pushdown Automata

- NondeterminIstic automata with a stack.
  - Stack: Last in first out; store **unlimited** amount of information
  - Deterministic and nondeterministic PDA are **NOT** the same.
  - NPDA recognizes the class of context free grammars.

- Formal Definition (Q, ∑, T, delta, q_0, F). all finite sets
  - Q - set of states
  - ∑ - alphabet
  - T - stack alphabet
  - Delta - Q X ∑ X T → P(Q X T) --- power set
  - q_0 - start state
  - F - set of accept states

- Transition function example
  - q_1 ---- (0, ε → 0) ----> q_2
  - meaning, when at q_1, read in 0, pop ε, and push 0 onto the stack, and go to q_2.

- **Equivalence** in power: A language is context-free **iff** some PDA recognizes it
  - If language is context free, PDA recognizes it.
  - If recognizes by a PDA, the language is context free.

- **Every regular language is context free.**

2.3 Non-context-Free Languages

- Non-CFL pumping lemma
  - There is a pumping length p, such that every string in this language has length ≥ p and can be divided into 5 pieces, s = uvxyz
    - For each i ≥ 0, uv^ixy^iz is in this CFL
    - |vy| > 0
    - |vxy| ≤ p
-

2.4 Deterministic Context-Free Languages

3.1 Turing Machines

- Mechanism
  - Input on the leftmost n squares, and rest are blank.
  - If move left off the left-hand end, stay there.
  - Halts
    - Accept
    - Reject
  - Never halts and keep looping

- Differences between finite automata and Turing Machine.
  - The tape is **infinite.**
  - Tape head can **read / write** symbols and **left / right**.
  - Once reach either accept or reject states, computation stops. Accept/reject takes effect **immediately**.

- Formal definition
  - Q - set of states
  - ∑ - alphabet **except the blank symbol** ␣
  - T - tape alphabet
  - q_0 - start state
  - q_{accept} - accept state
  - q_{reject} - reject state
  - **δ**
    - **Q X T → Q X T X {L, R}**
    - If **$\delta(q_0, a) = (q_1, b, L)$**, then it means we are in a certain state q_0, and the head is over a tape square of symbol a. **Replace** a with symbol b, move to the **left** afterwards, and go to state q_1.

- Configuration -- **changes occur in**
  - Uqv, where u is uv is current tape content, q is current state, tape head at the first symbol of v.
  - Start, accept, reject, halting configurations.
  - **ex. Group_hw5**
  - **decider : halt on every input**

- Turing-recognizable → some TM **recognizes** the language(accept and halt)
- Turing-decidable → TM is a **decider** and recognizes the language (either accept or reject, no loop)
- **All decidable languages are Turing-Recognizable**

- **Descriptions (3.3)**

- Usually only gives **high-level description**
    - No mentioning of tape, memory management, read/write head…
- Implementation level: mention **tape**, but not states
- Formal definition:
-
    - always a string.
    - **states** and transition functions
    - <>

3.2 Church - Turing Thesis: Variants of Turing Machines

- Robustness: all variations of Turing Machines are equivalent.

- Multi-tape Turing Machine
    - Convert to a single tape
    - Used to prove recognizable languages closed under union

- Non-deterministic turing machine
    - Proof idea: refer p. 178 - 180.
    - Also used to prove recognizable languages closed under union.

- Enumerators
    - A TM with an attached printer
    - Start with blank input
    - If does NOT halt, print **infinite strings**
    - **Can generate strings in any order, repetition**
    - A language is Turing-recognizable **iff** some enumerator enumerates it
        - Assume enumerates L, WTS L is Turing recognizable (subroutine)
        - Assume L is Turing recognizable, WTS some enumerates. (print out the strings M recognizes)

3.3. The definition of algorithm

- Each algorithm can be implemented by some TM

	Suppose M is a TM that recognizes L	Suppose D is a TM that decides L	Suppose E is E that enumerates L
If string w is in L then…	M accepts w	D accepts w	E prints w
If string w is not in L then…	M rejects w or loops on w	D rejects w	E never prints w

- <O> is the string that represents the object O
- Define **using high-level description** a Turing machine M_1 =
    - "On input <B, w>, where B is a DFA and w is a string:
    - Type check encoding to check input is valid type
    - **Simulates B** on w
    - If simulations ends in accept states of B, accept; otherwise, reject
-

4.1 Decidable Language

- Computation problem is **decidable** iff the language encoding the problem instances is decidable

- Encode objects of interest as strings.

- A_i, E_j, EQ_j are all computational problems. <DFA, w> member of A_{DFA}.

- **Decidable languages**
    - A_{DFA} = {<B, w> | B is a DFA that accepts input w}, A_{NFA}, A_{REX}; A_{CFG}
    - E_{DFA} = {<A> | A is a DFA and L(A) = ø}, E_{CFG}
    - EQ_{DFA} = {<A, B> | A and B are DFAs and L(A) = L(B)}

- Proving EQ_{DFA} as a decidable language
    - Symmetric difference: L(C) = (L(A) ∩ L'(B)) ∪ (L'(A) ∩ L(B)).
    - L(A) = L(B) iff L(C) = empty set.

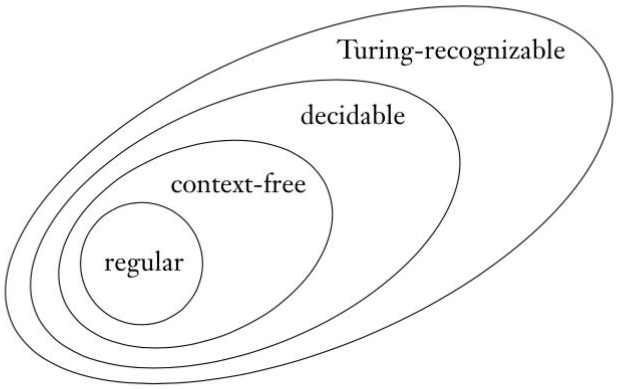


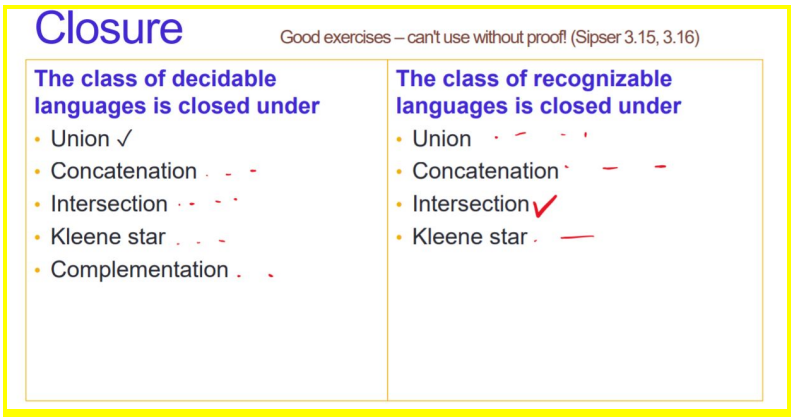
FIGURE **4.10**
The relationship among classes of languages

- All Turing-recognizable languages **countably infinite** languages.
- Set of subsets of Turing-recognizable languages = P(L) → uncountably infinite
- Non Turing-recognizable languages **uncountably** infinite.

Closure Claim

Closure properties of …

The class of regular languages is closed under	The class of context-free languages is closed under
• Union	• Union
• Concatenation	• Concatenation
• Star	• Star
• Complementation	• Reversal ✓
• Intersection	• FlipBits ✓
• Difference	**The class of context-free languages is not closed under**
• Reversal ✓	• Intersection
• FlipBits ✓	• Complementation
	• Difference



Lecture notes

- **4-18 Non-regular set**
  - Regular set
    - Finite language are all regular.
    - Non-regular sets must be infinite
  - Proving non-regularity
    - Subset of regular set can be non-regular. (*Prove?*)
  - Counting languages
    - Languages are in **Power set of {0, 1}***, uncountably infinite.
    - Regular languages ≤ regular expressions, **countably infinite.**

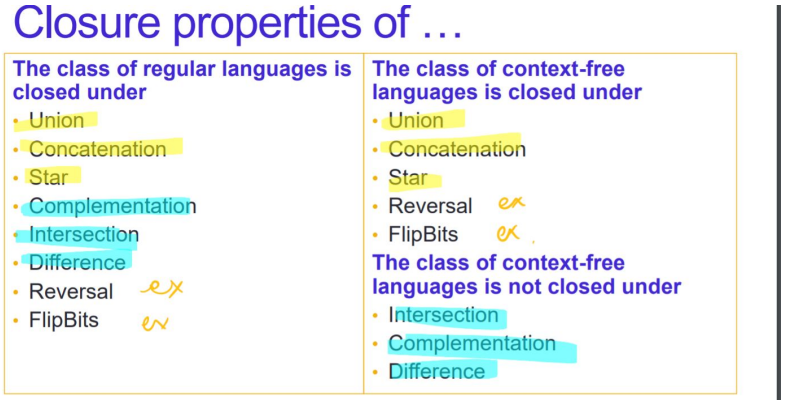
- **4-20 Pumping lemma Exercises**
  - {w w^R | w is a string over {0, 1} } s = 1^P001^P, i = 3
  - $\{a^nb^ma^n \mid m, n \geq 0\}$ s = $a^pb^pa^p$

- **4-27 Push-down Automata**
  - Read a, pop b, push c: a, b → c
  - At q_1, read in a, the top symbol of the stack is b, goes to q_2 and the new top item is c
  - Empty string **not** included in stack and input alphabet.
  - Informal description
    - How to push and pop, how to read.
    - Stack management
  - Design PDA L = { $a^ib^jc^k$ | i = j or i = k, with i, j, k ≥ 0 }

- **4-30 PDA Design**
  - If L is regular, then there is a PDA recognizes it.
  - Closed under **union, concatenation, kleene star**

- **5-2 Context-free Grammar**
  - Start variable, **one step application** of rule, string of terminals.
  - PDA, CFG equally expressive.
  - $\{0^i1^j \mid j \geq i \geq 0\}$ X → 0x1 | x1 | empty string | 1

- **5-4 Context-free language**
  - CFG examples:
    - At least 3 ones
    - Odd length
  - Closure under **union**
    - Assume V_1 V_2 disjoint. G = (V_1 U V_2 U $\{S_0\}$, .. R_1 U R_2 U $\{S_0 \rightarrow S_1 \mid S_2\}$, $S_0\}$
  - Closure under **concatenation**
    - Assume V_1 V_2 disjoint. G = (V_1 U V_2 U $\{S_0\}$, .. R_1 U R_2 U $\{S_0 \rightarrow S_1S_2\}$, $S_0\}$
  - $\{a^nb^m \mid n \neq m\}$
    - Union n < m and n > m
  - Closure:



  - Every regular language is a context-language
  - There are context-free languages that are not regular
    - **E.x $\{0^n1^n \mid n \geq 0\}$**
  - Countably infinite regular languages and context-free languages.

- **5-7 Turing Machines - Formal definition**
  - Unlimited input, memory, and time
  - Simulate DFA/NFA, PDA with Turing Machine
  - Turing Machines sometime **neither** accept or reject.

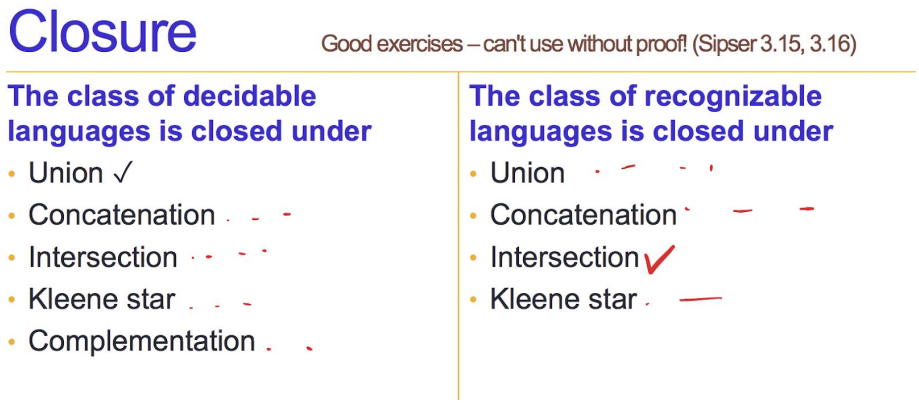
- **5-9 Turing Machines - Implementation-level description**
  - Recognize a language → halt and accept
  - Configuration
  - Implementation level description for L = { w # w | w in {0, 1}* }
    - How to move around on the tape

- State diagrams.
    - **Convention: Missing transitions are (q_{reject}, _, R)**

- **5-11 Turing Machines - High level description**
    -

Formal	Set of states, input alphabet, tape alphabet, transitions
Implementation	English description on how to move the tape, and change the content on the tape (**No states**)
High-Level	**Without implementation details. Algorithm description**

    -
    - Decider: halts on all input.
    - L(M) = L(M_1) intersects L(M_2). → assume M1 and 2 deciders, then M decider.
    - Prove in two ways.
    - Closure
        - Decidable languages closed under union



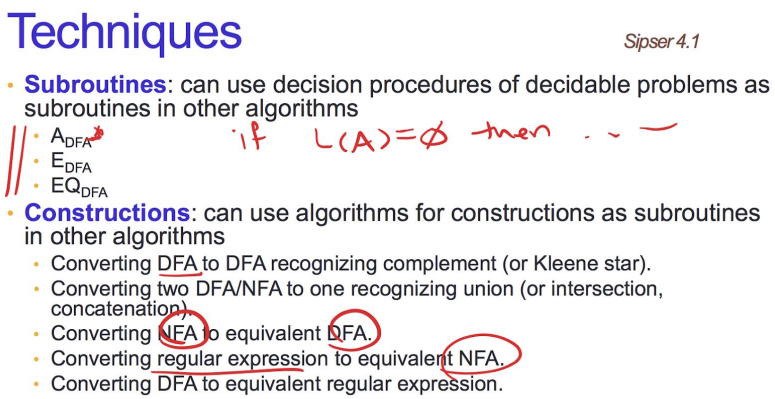
        -
        - * recognizable run on two machines **step by step**.

- **5-14 Church-Turing Thesis**
    - All variants of Turing Machines are **equally expressive**. AKA every language recognized by M1 is recognized by M2, and every language recognized by M2 is recognized by M1
        - E.x. Recognizable closed under **union**
    - Refer back in 3.1 variants of turing machines.
    - Church-Turing Thesis - each algorithm can be described by some Turing machine.
    - Enumerator
        - **Does not have input**
        - There is **no w**, undeclared variable

- **5-16 Decidable problems**

- Represent the computational problem as strings.
- Can simulate other Turing machines / algorithms as subroutine of program.
- Prove decidability: confirm strings in the language are accepted and not in languages are rejected.

- **5-18 Decidable problems example**
  - E_{DFA}
    - BFS in diagram to look for paths to F
    - WTS 1) L(M) = E_{DFA} (two way) and 2) M is decider.
  - E'_{DFA}
    - M_4
      - Loops if DFA A, L(A) = ø
      - Recognizes but not decidable
  - EQ_{EFA}
    - Using symmetric difference.
    - Check the if the result is empty set
    - Correctness proof
      - L(M) = EQ_{DFA} (two way)
      - M is a decider
  - Techniques:



  -

- **5-21 Decidable languages**
  - Decidable computational problems:

    A computational problem is **decidable** iff the language encoding the problem instances is decidable.

    In Sipser 4.1: The computational problems below
    A_{DFA}, A_{NFA}, A_{REX}, A_{CFG}
    E_{DFA}, E_{NFA}, E_{REX}, E_{CFG}
    EQ_{DFA}, EQ_{NFA}, EQ_{REX}
    are all decidable

  -

- Counting argument to prove undecidable
    - Turing recognizable are countable.
    - A_{TM} not decidable..

Practice

- Q3
    - 1. If current input symbol is b, reject.
    - 2. Read an a and cross off that a, keep reading other a's or crossed off symbols until reads a b, cross off that b. If sees any a's after seeing b, reject
    - 3. Scan all the way left to the right of the last crossed off a. If current symbol is a, go to step 2
    - 4. Accept if all symbols are crossed off

==========MIDTERM 2 后新内容分割线==========

ATM :

• Recognizable • Not decidable

Idea

If problem X is no harder than problem Y, and if Y is decidable then X must also be decidable.

If problem X is no harder than problem Y and if X is undecidable,then Y must also be undecidable

If problem X is no harder than problem Y and if Y is decidable then X must also be decidable

If problem X is no harder than problem Y and if X is undecidable then Y must also be undecidable

"Problem X is no harder than problem Y" means "Can convert questions about membership in X to questions about membership in Y

Problem A is mapping reducible to problem B means there is a computable function f: $\Sigma^* \rightarrow \Sigma^*$ such that for all strings x in Σ^* x is in A iff f(x) is in B

Diagonalization proof: A_{TM} not decidable

Sipser 4.11

Assume, towards a contradiction, that M_{ATM} decides A_{TM}

Define the TM D = "On input <M>:

1. Run M_{ATM} on <M, <M>>.
2. If M_{ATM} accepts, reject; if M_{ATM} rejects, accept."

G = "On input <M>, TM
1. Run M_{ATM} on <M, <M>>
2. If M_{ATM} accepts & rejects, reject. If rejects, accept."

Consider **running D on input <D>**. Because D is a decider:

- **either computation halts and accepts** …
- **or computation halts and rejects** …

The halting problem!

$HALT_{TM}$ = { | M is a TM and M halts on input w}
A_{TM} = { | M is a TM and w is in L(M)}

Reducing ATM to HALTTM
Desired function by cases:
• If x = and w is in L(M): map to in HALTTM
• If x = and w is not in L(M): map to not in HALTTM
• If x ≠ : map to some string not in HALTTM

Define computable function:
F = "On input x:
1. Type-check whether x=<M,W> for some TM M, and string w. If not, output constout .
2. Construct the following machine M' M'=
"On input x:
1. Run M on x.
2. If M accepts, accept.
3. If M rejects, enter a loop."
3. Output <M,W> "

$HALT_{TM}$ mapping reducible to A_{TM}.

Need $f: \Sigma^* \to \Sigma^*$ s.t.

$x \in HALT_{TM} \iff f(x) \in A_{TM}$

if computable

Given $ct \notin A_{TM}$.

f = "On input x.
1. Typecheck to see if $x = \langle M, w \rangle$, M TM, w string
   if not, output ct. Otherwise: $x = \langle M, w \rangle$.
2. Build M' = "On input y.
   1. Run M on y
   2. If M accepts, accept.
   If M rejects, accept"
3. Output $\langle M', w \rangle$."

Check: $x \in HALT_{TM}$ iff $f(x) \in A_{TM}$

Decidable	Undecidable but recognizable	Undecidable and unrecognizable
A_{DFA}	A_{TM}	A_{TM}^C
E_{DFA}	$HALT_{TM}$	$HALT_{TM}^C$
EQ_{DFA}		

Claim: A_{TM} is no harder than E_{TM}^C

In other words: we want to mapping reduce A_{TM} to E_{TM}^C

Define computable F : Σ* → Σ*

$const_{out} \notin E_{TM}$

Input string	Output string
<M,w> where M accepts w	<M'> where L(M') nonempty
<M,w> where M does not accept w	<M'> where L(M') empty
x not encoding any <M,w>	$const_{out}$

? Type check →

so $f(x) \in E_{TM}^C$

so $f(x) \notin E_{TM}^C$

Def of M' uses M, w as parameter

Claim: $HALT_{TM}$ is no harder than EQ_{TM}

In other words: we want to mapping reduce $HALT_{TM}$ to EQ_{TM}

Define computable F : $\Sigma^* \rightarrow \Sigma^*$

F = "On input x:

1. Type-check whether x= <M,w> for some TM M, and string w. If not, output $const_{out}$.
2. Construct the following machine M_1 = ($\{q_{acc}, q_{rej}\}, \Sigma, \Sigma \cup \{u\}, \delta, q_{acc}, q_{acc}, q_{rej}$)
   with $\delta((q, x)) = (q_{acc}, u, L)$ $\forall q \in Q, x \in T$.
   and
   M_2 = " On input y:
   1. If $y \neq w$, accept. Otherwise, goto 2.
   2. Run M on w. If M accepts, accept. If M rejects, accept. "
3. Output <M_1, M_2>"

ote: $L(M_1) = \Sigma^*$ $L(M_2) = \begin{cases} \Sigma^* & \text{if } M \text{ halts on } w \\ \Sigma^* - \{w\} & \text{if } M \text{ loops on } w \end{cases}$

Claim: $HALT_{TM}$ is no harder than EQ_{TM}

In other words: we want to mapping reduce $HALT_{TM}$ to EQ_{TM}

Define computable F : $\Sigma^* \rightarrow \Sigma^*$ $const_{out} \notin EQ_{TM}$

Input string	Output string
<M,w> where M halts on w	<M_1,M_2> where $L(M_1)=L(M_2)$
<M,w> where M loops on w	<M_1,M_2> where $L(M_1)\neq L(M_2)$
x not encoding any <M,w>	$const_{out}$

Type check →

Claim: $HALT_{TM}^C$ is no harder than EQ_{TM}

In other words: we want to mapping reduce $HALT_{TM}^C$ to EQ_{TM}

Define computable F : $\Sigma^* \rightarrow \Sigma^*$ Assume $const_{out} \notin EQ_{TM}$

Input string	Output string
<M,w> where M halts on w	<M_1,M_2> where $L(M_1)=L(M_2)=\{w\}$
<M,w> where M loops on w	<M_1,M_2> where $L(M_1)=\{w\}\neq L(M_2)=\{\}$
x not encoding any <M,w>	$const_{out}$

F = "On input x:
1. If $x \neq$ <M,w> for any TM M, string w, output $const_{out}$.
2. Let x = <M,w> and build
   M_1 = "On input y:
   1. If y = w, accept.
   Else reject."
   M_2 = "On input y:
   1. If $y \neq w$ reject.
   2. Run M on w. If M accepts w, accept.
   If M rejects w, accept."
3. Output <M_1, M_2>"

==========MIDTERM 1 内容==========

Vocabulary review

From CSE20, etc. Sipser p. 14

- { a,b,c,d,e } The **set** whose elements are a,b,c,d,e
- | ababab | = 6 The **length** of the string ababab is 6
- | { a,b,c,d,e } | = 5 The **size** of the set {a,b,c,d,e} is 5

New vocabulary

Sipser p. 14

- { a,b }* The set of finite strings over the symbols a, b
  - Includes empty string ε
  - Includes a, aa, aaa
  - Includes b, bb, bbb
  - Includes ab, ababab, aaaaaaabbb
  - Does **not** include infinite sequences of a's and b's
  - Has **infinitely many** different elements

- **Alphabet** Nonempty set of symbols
- **String** over alphabet Σ, Element of Σ*
- **Language** over alphabet Σ, Subset of Σ*

- **1.1. Finite Automata**
  - Move from states to states, depending on the input received.
  - 5-tuple expression - **formal definition**
    - Q - a **finite** set of states
    - ∑ - a **finite** set of alphabet
    - **δ** - transition function
    - $q_0 \in Q$ - start state
    - $F \subseteq Q$ - set of accept states
  - Regular operations - closed
    - **Union** -

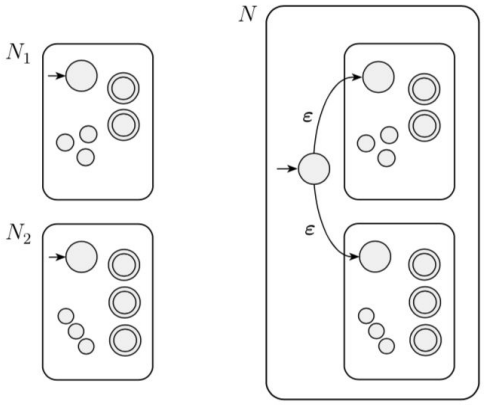
        - M_1 = (Q1, sigma, delta1, q1, F1); M_2 = (Q2, sigma, delta2, q2, F2).
        - M = (Q, sigma, epsilon, q0, F)
        - 1. Q = Q1 X Q2
        - 2. Delta((r1, r2), a) = (Delta1(r1, a), Delta2(r2, a))
        - 3. Q0 = (q1, q2)
        - 4. F = (r1, r2) where r1 belongs to F1 or r2 belongs to F2
    - **Concatenation** -
    - **Star** -
    - Proof by construction machines to recognize them.
- A language is called a regular language if some finite automaton recognizes it
- **Only has one unique next state**
- **Given the current state, we know what the next state will be**
- **The number of outgoing arrows must be |∑|**

- **1.2 Nondeterminism**
    - **DFA vs NFA**
        - **Every DFA is a NFA**
        - Every states of DFA has **exactly one transition arrow** for each symbol in the alphabet. NFA may have **n arrows** for each symbol, where **n ≥ 0**.
        - DFA has arrows only on alphabet but NFA might have arrow labeled with **ε**
        - **Every NFA can be converted to some DFA**
        - **Every DFA is a NFA**
    - NFA Computation
        - NFA splits to follow all possibilities in parallel, and if **any one of** the copies of machine is in an accept state at the end of input, NFA accepts.
        - When empty is string is encountered, one copy follow empty string arrow and one stay at current state.
    - **Formal definition of NFA**
        - Q is a finite set of states
        - ∑ is a finite alphabet
        - **δ:** Q X Sigma(empty string) -> P(Q)
        - q_0 belongs to Q: start state
        - F is subset of Q: set of accept states.
    - Equivalence of NFAs and DFAs
        - Two machines are equivalent if they **recognize the same language**
        - Every NFA has an equivalent DFA -> convert NFA to DFA
        - NFA has k states, then DFA has 2^k states (number of subsets **but not necessary**).
    - If a language is recognized by an NFA, then it is recognized by some DFA. Construct the DFA M as following
        - Q' = P(Q)

- δ'(R, a) = union of all sets of states original transition function takes to = E(δ(r, a)) ---> 包括empty string
- q_0' = E{q_0}
- F' = {R ∈ Q'| R contains an accept state of N}
- **Consider Ɛ arrows**
    - **We define E(R) to the collection of states that can be reached from members of R by going along Ɛ arrows**

- A language is regular if and only if some NFA recognizes it.
    - Two way
- **Given the current state, there could be multiple next states**

- **The class of regular languages is closed under the regular operations**
    - Union



- Concatenation

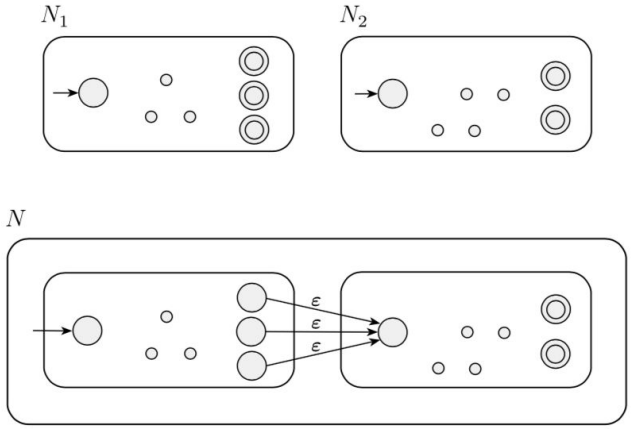


FIGURE 1.48
Construction of N to recognize $A_1 \circ A_2$

- Star operation

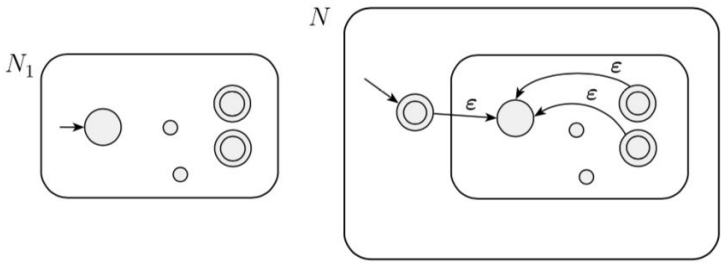


FIGURE **1.50**
Construction of N to recognize A^*

- **1.3. Regular Expressions**

  - Formal Definition: R is a regular expression if R is: (inductive definition)
    - a for some a in the alphabet
    - Empty string
    - Ø the language that doesn't contain any string
    - (R1 U R2)
    - (R1 0 R2)
    - (R1*)
    - Note: R+ has all strings that are 1 or more concatenation of strings from R.
  - Equivalence with Finite Automata
    - **A language if regular if and only if some regular expression describes it(exactly recognized by NFA, exactly recognized by DFA)**
      - Lemma 1: If language is described by a regular expression, it's regular. (Proof referred to textbook 67)
      - Lemma 2: if language is regular, it's described by a regular expression. (Proof refer to text. 68. GNFA??)

- **1.4. Non-regular Expression**

  - **Pumping Lemma**
    If A is a regular language, then there is a number p (the pumping length) where if s is any string in a of length at least p, then s may be divided into three pieces, s = xyz, satisfying the following conditions:
    1. For each i >= 0, x y^i z is an element of A
    2. |y| > 0, and
    3. |xy| ≤ p.
    - Proof idea: pigeonhole principle - sequence contains a repeated state.

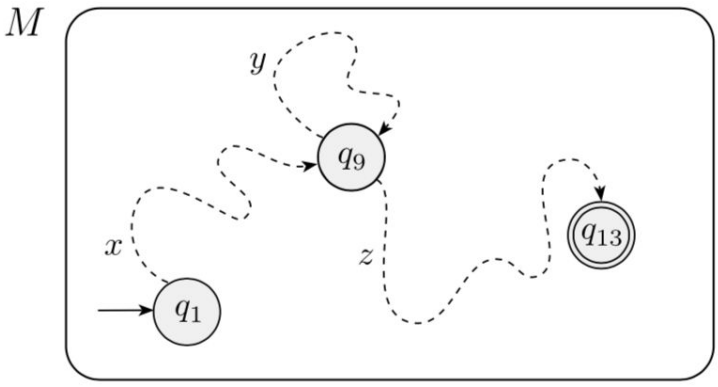


FIGURE 1.72
Example showing how the strings x, y, and z affect M

- Proving non-regularity
  - Assume B is regular and use pumping lemma. Find a string s in B that has length p or greater in B can be pumped. Then demonstrate that s cannot be pumped by considering all ways of dividing s into x, y, z.

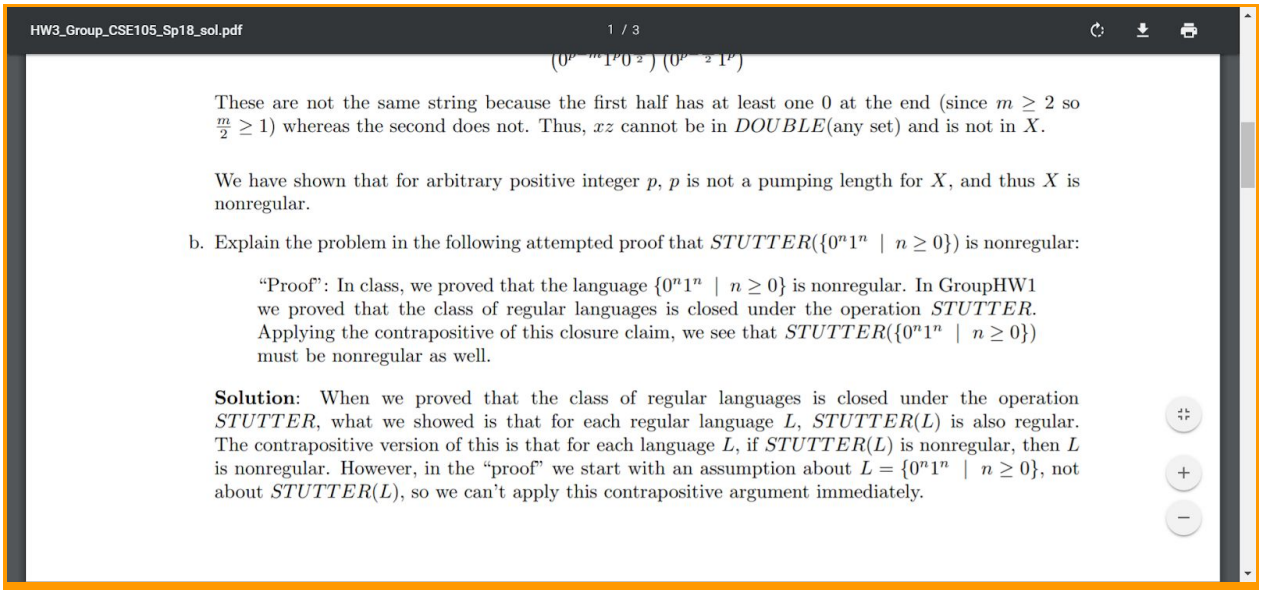
那些什么countably many 和 uncountably 什么什么的要考吗？
考吧·

✓ Which of the following sets are countably infinite? (select all that apply) 1/1

- [ ] The set of all languages over {0,1}
- [x] The set of all regular languages over {0,1} ✓
- [x] The set of all strings over {0,1} ✓
- [ ] The set {0,1}
- [x] The set of all DFAs over {0,1} (whose states are labelled by integers) ✓
- [x] The set of all regular expressions over {0,1} ✓

==========MIDTERM 1 之后内容==========

Prove 2 langanguage is the same: string w in L2 can be in L1, string w2 in L1 can be in L2
特别注意怎么prove stutter



$(0^{p-m}1^p0^2)\,(0^{p-2}1^p)$

These are not the same string because the first half has at least one 0 at the end (since $m \geq 2$ so $\frac{m}{2} \geq 1$) whereas the second does not. Thus, xz cannot be in $DOUBLE$(any set) and is not in X.

We have shown that for arbitrary positive integer p, p is not a pumping length for X, and thus X is nonregular.

b. Explain the problem in the following attempted proof that $STUTTER(\{0^n1^n \mid n \geq 0\})$ is nonregular:

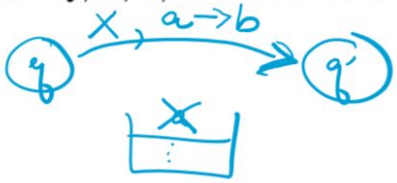
> "Proof": In class, we proved that the language $\{0^n1^n \mid n \geq 0\}$ is nonregular. In GroupHW1 we proved that the class of regular languages is closed under the operation $STUTTER$. Applying the contrapositive of this closure claim, we see that $STUTTER(\{0^n1^n \mid n \geq 0\})$ must be nonregular as well.

Solution: When we proved that the class of regular languages is closed under the operation $STUTTER$, what we showed is that for each regular language L, $STUTTER(L)$ is also regular. The contrapositive version of this is that for each language L, if $STUTTER(L)$ is nonregular, then L is nonregular. However, in the "proof" we start with an assumption about $L = \{0^n1^n \mid n \geq 0\}$, not about $STUTTER(L)$, so we can't apply this contrapositive argument immediately.

Formal definition of PDA

Sipser Def 2.13 p. 113

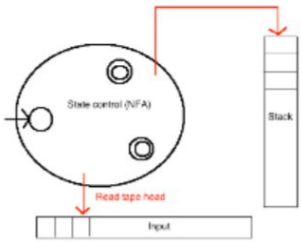
A PDA is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where Q, Σ, Γ, F are all finite sets and



1. Q is the set of states
2. Σ is the input alphabet
3. Γ is the stack alphabet
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function
5. $q_0 \in Q$ is the start state
6. $F \subseteq Q$ is the set of accept states.

PS: 如果是一个什么都不push进去的PDA， STACK ALPHABET还是要把占位符写上，随便写一个，但是要写

- Automata
  - String is read by the machine one character at a time, from left to right.
  - Determine if computation is successful by checking if entire string was read, and if land at an accept state.



- Regular expressions and ??
  - Derive all strings in the language by following rules for required patterns.

Context-free grammar

Sipser Def 2.2, page 102

(V, Σ, R, S)

Variables: finite set of (usually upper case) variables **V**

Terminals: finite set of alphabet symbols **Σ** $V \cap \Sigma = \emptyset$

Rules/Productions: finite set of allowed transformations **R**

$A \to u$ $A \in V, u \in (V \cup \Sigma)^*$

Start variable: origination of each derivation **S**

with the rules

G = ({S}, {0}, R, S)

R = {S → 0S, S → 0}

Theorem 2.20: A language is context-free if and only if some nondeterministic PDA recognizes it.

A derivation is leftmost if at every step, the leftmost remaining variable is the one replaced.

CFL closed under union:

C. $G = (V1 \cup V2 \cup \{S0\}, \Sigma, R_1 \cup R_2 \cup \{S0 \rightarrow S1, S0 \rightarrow S2\}, S0)$

Summary :
A language L is Regular :
iff it is described by some regular expression
iff it is recognized by some DFA
iff it is recognized by some NFA
Context-free:
iff it is recognized by some PDA
iff it is generated by some CFG

• Fact: Every regular language is a context-free language.
Fact: There are context-free languages that are nonregular.
Fact: There are countably infinitely many regular languages.
Fact: There are countably infinitely many context-free languages.
Most languages are not context-free languages!

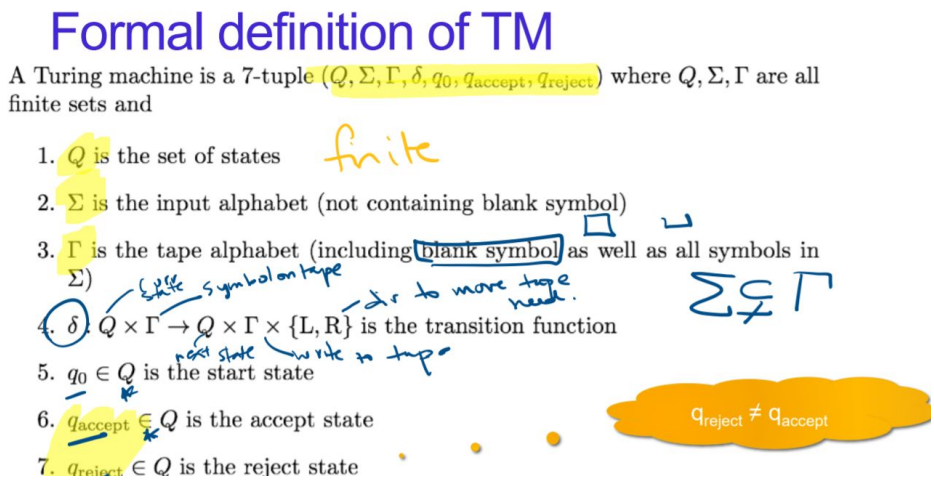
Examples of non-context-free languages

- $\{ a^n b^n c^n \mid 0 \le n \}$ Sipser Ex 2.36
- $\{ a^i b^j c^k \mid 0 \le i \le j \le k \}$ Sipser Ex 2.37
- $\{ w\,w \mid w \text{ is in } \{0,1\}^* \}$ Sipser Ex 2.38

Closure properties of …

The class of regular languages is closed under	The class of context-free languages is closed under
• Union	• Union
• Concatenation	• Concatenation
• Star	• Star
• Complementation	• Reversal
• Intersection	• FlipBits
• Difference	**The class of context-free languages is not closed under**
• Reversal	• Intersection
• FlipBits	• Complementation
	• Difference

TURING MACHINE:



Turing machine如果在左边的头被要求左走的话会一直读头的东西，图灵机往右无限长

Turing machine can: halt and accept/reject, loop

Describing TMs:
Formal definition: set of states, input alphabet, tape alphabet, transition function, state state, accept state, reject state. (提到states)

Implementation-level definition: English prose to describe Turing machine head movements relative to contents of tape.(提到tape)

High-level desciption: Description of algorithm, without implementation details of machine. As part of this description, can "call" and run another TM as a subroutine.(提到另一个machine as subroutine)

Language of a turing machine:
L(M) = { w | computation of M on w halts after entering the accept state}
L is recognized by Turing machine M if L(M) = L.
M recognizes L if M is a Turing machine and L(M) = L

• M is a decider if it is a Turing machine and halts on all inputs.

• L is decided by Turing machine M if M is a decider and L(M) = L.
M decides L if M is a decider and L(M) = L.

Decide 前提条件是有一个decider (always halt)

建造新的图灵机来搞各种操作：（union, star,concatination….):

几个例子

Closure

Theorem: The class of decidable languages over fixed alphabet Σ is closed under union.

Proof: Let L_1 and L_2 be languages and suppose M_1 and M_2 are TMs deciding these languages. Construct the TM M as "On input w,

1. Run M_1 on input w. If M_1 accepts w, accept. Otherwise, go to 2.
2. Run M_2 on input w. If M_2 accepts w, accept. Otherwise, reject."

Correctness of construction:

WTS $L(M) = L_1 \cup L_2$ and M is a decider.

Where do we use decidability?

b.

Let L_1 and L_2 be two decidable Languages. M_1 and M_2 be the Turing machines that decides L_1 and L_2 respectively.

There is a Turing machine M' such that, it decides concatenation of L_1 and L_2 i.e., $L(M') = L_1 \circ L_2$.

The description of M' is as follows:

M' = on input *w*:

1. Split *w* into two parts w_1, w_2 such that $w = w_1 w_2$
2. Run M_1 on w_1. If M_1 rejected then ***reject***.
3. Else run M_2 on w_2. If M_2 rejected then ***reject***.
4. Else ***accept***

Try each possible cut of *w*. If first part is accepted by M_1 and the second part is accepted by M_2 then *w* is accepted by M'. Else, *w* does not belong to the concatenation of languages and is rejected.

Therefore, $L(M') = L_1 o L_2$. The decidable languages are closed under concatenation.

c.

Let *L* be a Turing decidable Language and *M* be the Turing machine that decides *L*.

There is a Turing machine M' such that, it decides star of *L* i.e., $L(M')=L^*$.

The description of M' is as follows:

$M' =$ On input *w*:

1. Split *w* into *n* parts such that

$w = w_1 w_2 ... w_n$ in different ways.

2. Run *M* on w_i for $i = 1, 2, ...n$.

If *M* accepts each of these strings w_i, ***accept***.

3. All cuts have been tried without success then ***reject***.

When *w* is cut into different substrings such that every string is accepted by *M*, then *w* belongs to the star of *L* and thus M' accepts *w* after finite number of steps, else *w* will be rejected. Since, there are finitely many possible cuts of *w*, M' will halt after finitely many steps.

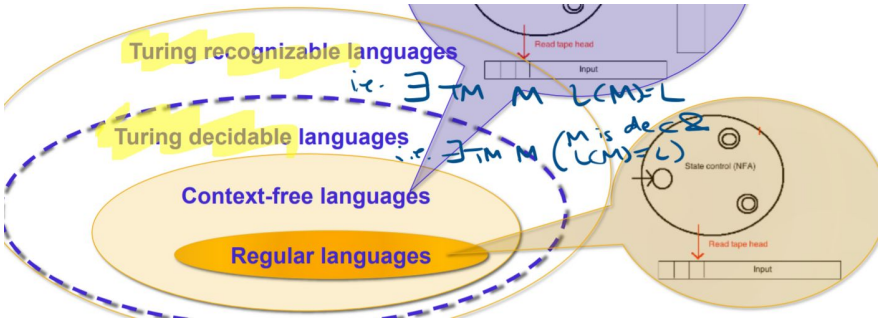
Therefore, $L(M') = L^*$. The decidable languages are closed under star.

Closure

Good exercises – can't use without proof! (Sipser 3.15, 3.16)

The class of decidable languages is closed under	The class of recognizable languages is closed under
• Union ✓	• Union
• Concatenation	• Concatenation
• Intersection	• Intersection ✓
• Kleene star	• Kleene star
• Complementation	

Turing machine, regular language的各种关系图



Meaning of equally expressive:
Model 1 is equally expressive as Model 2 iff every language recognized by a Model 1 machine is recognizable by a Model 2 machine and every language recognized by a Model 2 machine is recognizable by a Model 1 machine.
=> Thus, we can have equally expressive multitape turing machines(because it is equally expressive as turing machine)

Example: union

Claim: The class of recognizable languages over fixed alphabet Σ is closed under union.

Proof idea: Let M_1 and M_2 be TMs. Construct the two tape TM M = "On input w,

1. Copy w to second tape.
2. Simultaneously simulate the computations of M_1 and M_2 on w by using the two tapes.
3. If either computations halts and accept, accept."

Nondetermistic TMs:

Nondeterministic TMs

Sipser p. 178

At any point in the computation, machine may proceed according to several possibilities.

Transition function

$$Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L,R\})$$

Nondeterministic machine accepts w iff there is a computation path that halts and accepts

Sketch of proof of equivalence:

To simulate nondeterministic machine: Use 3 tapes to do breadth-first search of computation tree: "read-only" input tape, simulation tape, tape tracking nondeterministic braching.

Example: union

Claim: The class of recognizable languages over fixed alphabet Σ is closed under union.

Proof idea: Let M_1 and M_2 be TMs. Construct the nondeterministic TM M = "On input w,

1. Nondeterministically choose M_i to be either M_1 or M_2
2. Run M_i on w. If it accepts, accept; if it rejects, reject."

Enumerator:

Produce language as output rather than recognize input

At any point, machine may "send" a string to printer.

L(E) = { w | E eventually, in finite time, prints w}

	Suppose M is TM that recognizes L	Suppose D is TM that decides L	Suppose E is enumerator that enumerates L
If string w is in L then …	M accepts w	D accepts w	E prints w (in finite time)
If string w is not in L then …	M rejects w OR M loops on w	D rejects w	E never prints w.

Computational problems

Sipser p. 194

A computational problem is **decidable** iff the language encoding the problem instances is decidable

- Does a specific DFA accept a given string? encoded by
  { representations of DFAs M and strings w such that w is in L(M) }
- Is the language generated by a specific CFG empty? encoded by
  { representations of CFGs G such that L(G) = Ø }
- Is a Turing machine a decider? encoded by
  { representations of Turing machines M such that M always halts }

Decidability of Computational problems:

Proving decidability

Claim: E_{DFA} is decidable
Proof: WTS that { <A> | A is a DFA over Σ, L(A) is empty } is decidable.

Step 1: construction
Idea: breadth-first search in state diagram to look for paths to F
Define TM M_2 by: M_2 = "On input <A>: A DFA:

1. ~~Check whether input is a valid encoding of a DFA, if not, reject.~~ type check
2. Mark the start state of A.
3. Repeat until no new states get marked:
   i. Loop over states of A and mark any unmarked state that has an **incoming** edge from a marked state.
4. If no final state of A is marked, *accept*; otherwise, *reject*."

Decider? Yes b/c A has finitely many state

Techniques

Sipser 4.1

- **Subroutines**: can use decision procedures of decidable problems as subroutines in other algorithms
  - A_{DFA}
  - E_{DFA}
  - EQ_{DFA}
- **Constructions**: can use algorithms for constructions as subroutines in other algorithms
  - Converting DFA to DFA recognizing complement (or Kleene star).
  - Converting two DFA/NFA to one recognizing union (or intersection, concatenation).
  - Converting NFA to equivalent DFA.
  - Converting regular expression to equivalent NFA.
  - Converting DFA to equivalent regular expression.

a. True / False: L has pumping length 10.

True: In general, if a language has pumping length c then it has pumping length d for all $d \geq c$ because the definition of pumping length is that "all strings in the language of length greater than or equal to this pumping length can be pumped in the language". Since the collection of strings in the language of length greater than or equal to d is a subset of the collection of strings in the language of length greater than or equal to c, satisfying the definition of c being a pumping length guarantees that d is a pumping length too.

2. (**10 points**). We say that a string s in language L **can be pumped in** L to mean that there exist x, y, z such that $s = xyz$, $|y| > 0$, and for each $i \geq 0$, $xy^iz \in L$. *Notice that we don't have item 3 from Theorem 1.70 because we have not fixed a p.* For each of the strings and languages below, determine if the string can be pumped in the language. If so, give an example of x, y, z that work for this string.

a. $s = 110010$, $L = \{110010\}$. The string s **cannot be pumped.** (L is finite but if s was able to be pumped then infinitely many strings would be in L.)

b. $s = 111000$, $L = L(1^*0^*)$. The string s **can be pumped**. For example, $x = \varepsilon$, $y = 111$, $z = 000$ would work. Also $x = 111$, $y = 0$, $z = 00$ would work.

c. $s = 100$, $L = L(100 \cup 0^*1^*)$. The string s **cannot be pumped.** (Because no matter how we choose x, y, z with y nonempty, infinitely many of the strings xy^iz will have a least one 1 before at least one 0. There is only one such string in L.)

d. $s = \varepsilon$, $L = \{0, 1\}^*$. The string s **cannot be pumped.** (Because no nonempty string y is a substring of the empty string.)