CSE 105 Final Review

This review doc summarizes everything from the lecture slides, practice problems and homework. Created by M. and Yilin, feel free to collaborate.

Midterm 1 Topics

- Finite automata: DFA, NFA, Regex
- Closure claim
- Formal definitions
- Proving non-regularity

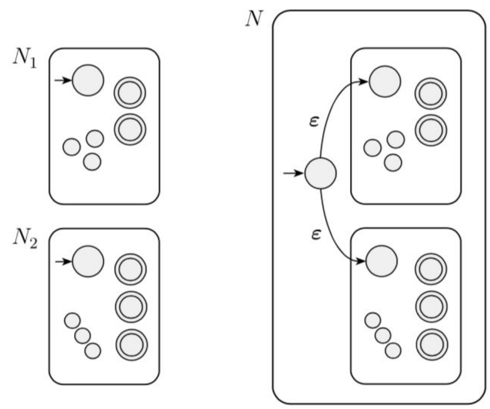
Summary

- **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 ε arrows**
        - **We define E(R) to the collection of states that can be reached from members of R by going along ε 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

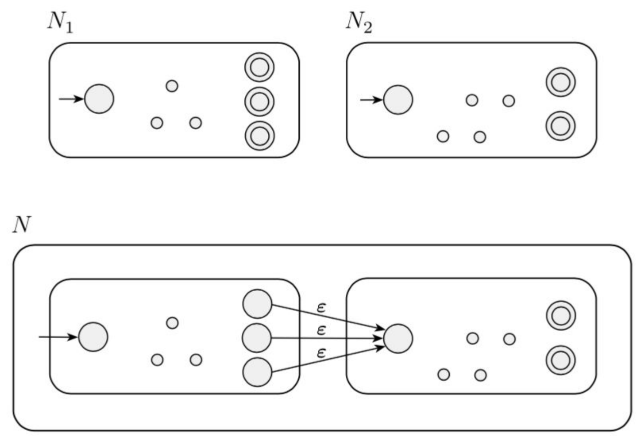


FIGURE 1.48
Construction of N to recognize $A_1 \circ A_2$

- Star operation

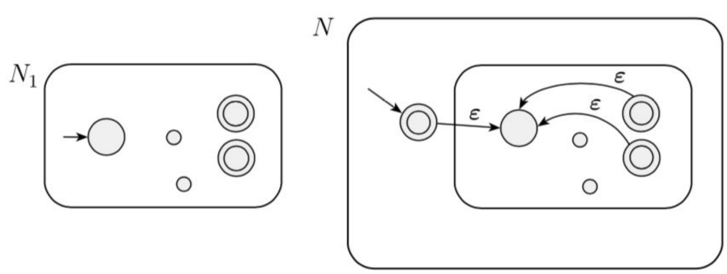


FIGURE 1.50
Construction of N to recognize A^*

- **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.

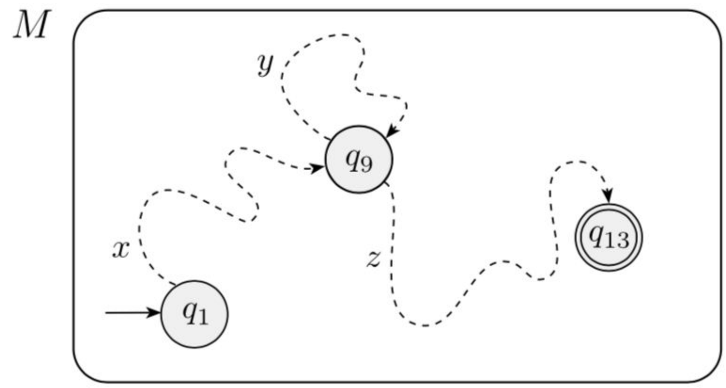


FIGURE 1.72
Example showing how the strings x, y, and z affect M

- 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.

Midterm 2 Topics:

- 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.

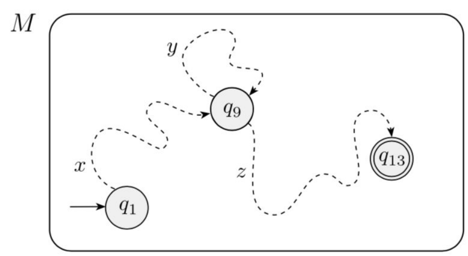


FIGURE 1.72
Example showing how the strings x, y, and z affect M

- **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, $n \geq 2$

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, $\sum$, T, delta, q_0, F). all finite sets
  - Q - set of states
  - $\sum$ - alphabet
  - T - stack alphabet
  - Delta - $Q \times \sum \times T \rightarrow P(Q \times T)$ --- power set
  - q_0 - start state
  - F - set of accept states

- Transition function example
  - q_1 ---- $(0, \varepsilon \rightarrow 0)$ ----> 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 $i \geq 0$, uv^ixy^iz is in this CFL
        - $|vy| > 0$
        - $|vxy| \leq p$
-

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.

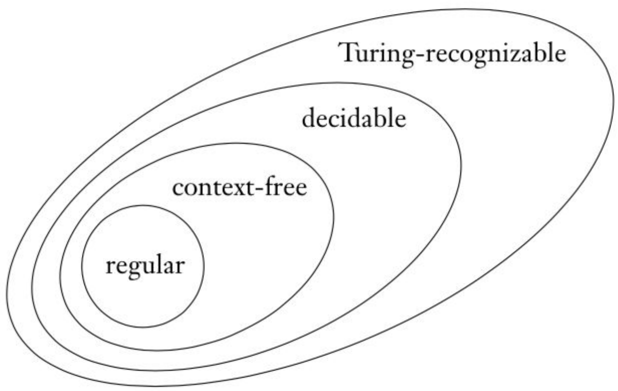


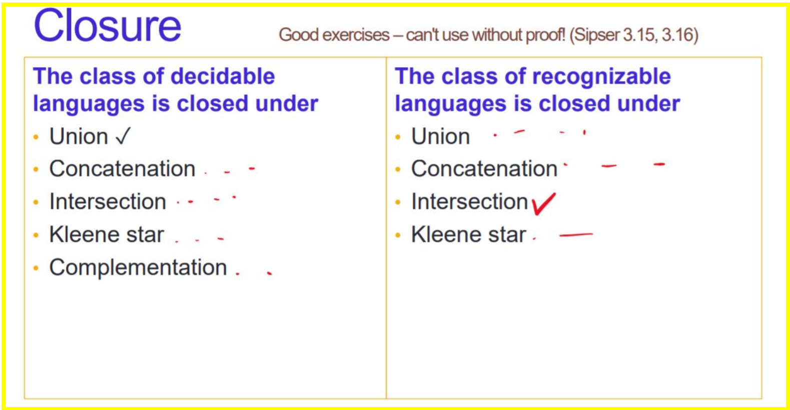
FIGURE 4.10
The relationship among classes of languages

- All Turing-recognizable languages **countably infinite** languages.
- Set of subsets of Turing-recognizable languages = P(L) → uncountably infinite
- Non Turing-recognizable languages **uncountably** infinite.
- Every language over a finite (or even countable) alphabet is countable. Assuming your Turing machine alphabet is finite, any language it can possibly accept is countable.
-
-

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 ex
• Intersection	• FlipBits ex
• Difference	**The class of context-free languages is not closed under**
• Reversal ex	• Intersection
• FlipBits ex	• 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.**

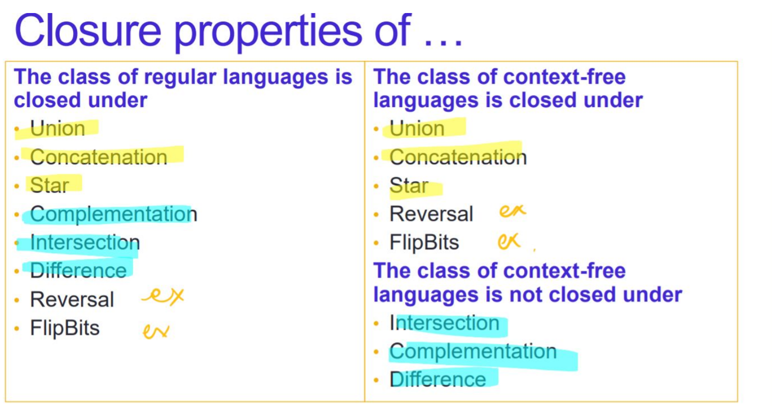
- **4-20 Pumping lemma Exercises**
    - $\{w w^R \mid w$ is a string over $\{0, 1\}\}$ $s = 1^P001^P$, $i = 3$
    - $\{a^nb^ma^n \mid m, n \geq 0\}$ $s = a^pb^pa^p$

- **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 $L = \{ a^ib^jc^k \mid i = j \text{ or } i = k, \text{ with } i, j, k \geq 0 \}$

- **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.
    - $\{0^i1^j \mid j \geq i \geq 0\}$ $X \rightarrow 0x1 \mid x1 \mid$ empty string $\mid 1$

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

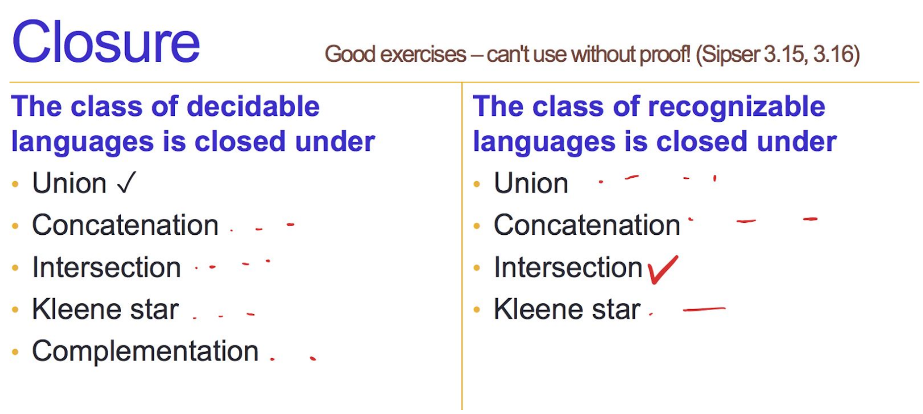
- **5-9 Turing Machines - Implementation-level description**
    - Recognize a language $\rightarrow$ halt and accept
    - Configuration
    - Implementation level description for $L = \{ w \# w \mid w \text{ in } \{0, 1\}^* \}$
        - How to move around on the tape

- 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:

Techniques

Sipser 4.1

- **Subroutines**: can use decision procedures of decidable problems as subroutines in other algorithms
    - A_{DFA}*
    - E_{DFA}
    - EQ_{DFA}

if L(A)=∅ then ... —

- **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.

-

- **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} ex, A_{REX}, A_{CFG}
E_{DFA}, E_{NFA} ex, E_{REX}, E_{CFG}
EQ_{DFA}, EQ_{NFA} ex, EQ_{REX} ~~EQ_{CFG}~~

are all decidable

-

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