CSE322 Theory of Computation

Lec 19

If L is decidable then L is (recursively) enumerable.

3.13 L is decidable iff L is string-order enumerable.

Short to long, among same length, use dictionary ordering.

<= Level 1: Suppose E enumerates L in the string order.

Case(1) L is finite. L is regular, and hence decidable.

Case(2) L is infinite.

We will construct decider D to decide L.

D(x) :

1. D runs E
2. For every w that E outputs,
3. if w=x, D goes to qaccept
4. if w>x, D goes to qrej.
5. if w≠x, D continues.

What if x comes after the last string in L?

E: w_1 w_2 ... w_k x

× × × ✓

Claim: If x is in L, D(x) halts and accepts.

Claim: If x is not in L, D(x) halts and rejects.

a
b
c
d
z
aa
...
az
ba
bb
bc
bf
bg
ct
atz
azz
...

3.13 L is decidable iff L is string-order enumerable.

=> Level 1: Suppose M decides L.

We will construct enumerator E which enumerates L in the string ord.

E() : // enumerator ignores input

1. ... for all strings w in the string order:
2. .... run M(w) // decider
3. If M accepts w : E outputs w

Claim: If $x \in L$, E would output x.

E never goes into an infinite loop

On line #3, E outputs x.

Claim:- If E outputs x, $x \in L$.

Claim:- All outputs of E are in the string order.

Closure Property: Union

* If L1 and L2 are decidable, then L1 U L2 is decidable.

Level 1: Proof by construction. M1 decides L1, M2 decides L2. Construct M to decide $L_1 \cup L_2$

Level 2: L1 is decidable => M1 decide L1. Similarly, let M2 decide L2.

Construct M to decide L = L1 U L2.

M(on input x):

1. ... Run $M_1(x)$
2. If M_1 accepts x, M goes to q_{acc}.
3. Else, run $M_2(x)$. Do what M_2 does. If M_2 accepts x, goto q_{acc}. If M_2 rejects x, goto q_{rej}.

~~Claim: M halts on all input.~~

Claim: If x is in L, M accepts x. ($L_1 \cup L_2$)

Claim: If x is not in L, M rejects x.

Closure Property: Union (solution)

* If L1 and L2 are decidable, then L1 U L2 is decidable.

L1 is decidable => M1 decide L1. Similarly, let M2 decide L2.

Construct M to decide L = L1 U L2.

M on input x:

1. Runs M1 on x. Since M1 is a decider, M1 must halt on all input.
2. If M1 accepts x, M accepts x. Otherwise, continue.
3. Runs M2 on x. Since M2 is a decider, M2 must halt on all input.
4. If M2 accepts x, M accepts x. Otherwise, M rejects x.

Claim: M halts on all input. (Proof: by construction of M)

Claim: If x is in L, M accepts x. (Proof: x in L => x in L1 or x in L2.

If x in L1, M1 accepts x. So M accepts x (in line 2).

If x is not in L1 then x must be in L2, then M2 accepts x and so does M (in line 4).)

Claim: If x is not in L, M rejects x. (Proof: x is not in L => x not in L1 and x not in L2.

[Use similar argument as above.] M will continue in line 2 and reject in line 4.)

Closure Property: Union

* If L1 and L2 are enumerable, then L1 U L2 is enumerable.
* If L1 and L2 are recognizable, then L1 U L2 is recognizable.

Make use of 2-tape TMs to run two TMs in parallel.

Exercise: Complete construction and proof.

M_1 recognizes L1
M_2 " L2

M to recognize $L_1 \cup L2$

$M(x)$: // NDTM
Non-deterministically run $M_1(x)$ or $M_2(x)$ and do as they do.

$M(x)$: // DTM
Run both $M_1(x)$ & $M_2(x)$, one step at a time.
Whenever any one accepts, M also accepts.

Closure Properties

* Decidable languages are closed under intersection.
* Recognizable languages are closed under intersection.
* Enumerable languages are closed under intersection.

* Recognizable languages are closed under concatenation.
* Decidable languages are closed under concatenation.
* Enumerable languages are closed under concatenation.

* Decidable languages are closed under complement.
* Recognizable languages are not (proof requires newer techniques, later).

Accept -DFA

A-DFA

A-DFA = { <B,w> : B is a DFA that accepts string w}

Thm: A-DFA is decidable.

Proof:

Level-1 :- Proof by constructing a TM that decides A-DFA.

Level-2 :- Construct 3-tape TM M-A-DFA that on input w does following:

M-A-DFA (<B, w>) :

0. M rejects if input is not of form <encoding of DFA, input on DFA alph.
1. M copies w to second tape and writes q0 of DFA on 3rd tape.
2. M start simulating DFA with second head on first symbol of input.
3. In each step, M scans its input tape (B portion) and finds the next state and updates it on 3rd tape. It moves the second head to right.
4. When second head reaches blank, M checks if third tape contains a state which is listed in the accept states of DFA. If yes, M accepts otherwise M rejects.

A-NFA = { <B,w> : NFA B accepts w }

M-A-NFA(<B,w>):

0. - - - -

1. Use the subset method to construct a DFA D s.t. L(D) = L(B). // D accepts w iff B accepts w

2. Run M-A-DFA(<D,w>) where ~~M-A-DFA~~ is the decider for A-DFA.

Regular Languages are Decidable

Suppose L is a regular language. Therefore, there exists a DFA D to recognize L.

Construct ML to decide L using DFA. D

ML(x): runs M-A-DFA(<D,w>)

$$E\text{-}DFA = \{ \langle A \rangle : A \text{ is DFA and } L(A)=\{\} \}$$

M-E-DFA (⟨A⟩) :

0. Input validation.
1. ... Check if final state is reachable from initial state
   or
   use the pumping lemma based method from homework.
   → run DFA on all strings of length $\leq |Q|$

$$NON\text{-}E\text{-}DFA = \{ \langle DFA\ A \rangle : L(A) \neq \{\} \}$$

Runs decider for E-DFA after switching accept & reject states.

Is NON-E-DFA recognizable? Decidable?

ALL-DFA = { <A> : A is a DFA and L(A) = everything }

M-ALL-DFA (<A>) :

0. Input validation.

1. ... Construct A' which is a copy of A & final states are swapped. // $L(A') = \overline{L(A)}$.

   $L(A) = \Sigma^*$ iff $L(A') = \emptyset$

2. Run M-E-DFA(A'), & do what it does.

Proof of correctness:

LEVEL-1: We will prove that

(a) M-ALL-DFA always halts

(b) If M-ALL-DFA accepts w then w is in ALL-DFA

(c) If w is in ALL-DFA then M-ALL-DFA accepts w

$L(C)=L(D)$ iff $L(A)=\emptyset$

EQ-DFA = { ⟨C,D⟩ : C,D are DFAs & L(C)=L(D) }

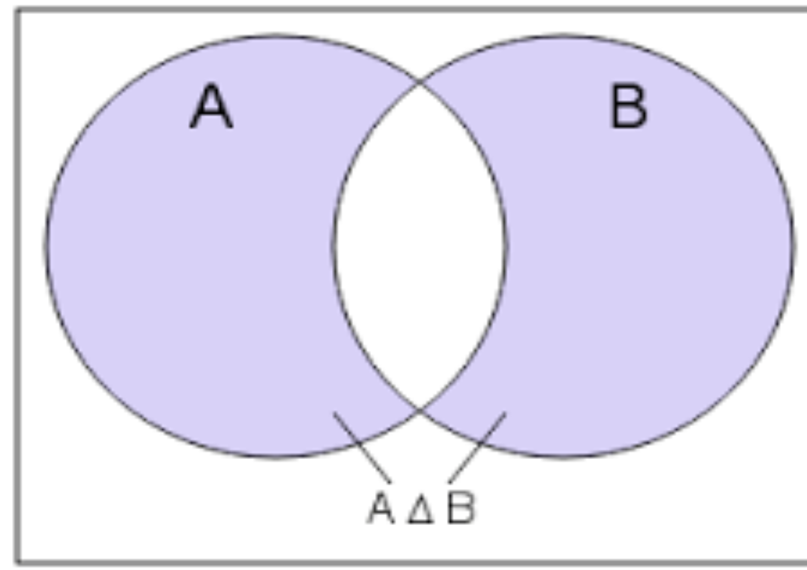
M-EQ-DFA (⟨C,D⟩) :

0. Input validation.
1. ... Construct DFA A s.t.

$L(A) = (L(C) - L(D)) \cup (L(D) - L(C))$

$= (L(C) \cap \overline{L(D)}) \cup (L(D) \cap \overline{L(C)})$

2. Run M-E-DFA(A).



Another proof for: E-DFA = {⟨A⟩ : L(A)={} }

M(⟨A⟩):

1. Construct DFA E which does not accept anything.
2. Run M-EQ-DFA(⟨A,E⟩).
3. M accepts iff M-EQ-DFA accepts.

Incorrect proof.

Why ?!