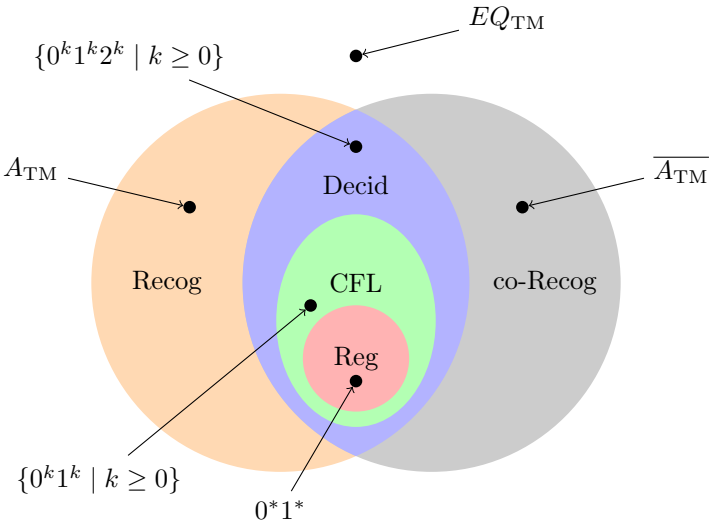
18.404/6.840 Recitation 6—Midterm Review

Abbas Zeitoun

October 9, 2020

Hierarchy of Languages



Proof Methods

Regular Languages

- To show regularity: DFA, NFA, GNFA, Reg-Ex, closure properties
- To show non-regularity: pumping lemma, closure properties

Context-Free Languages

- To show context-freeness: CFG, PDA, closure properties
- To show non-context-freeness: pumping lemma, closure properties

Decidable Languages

- To show decidability: Decider, reduction to a known decidable language, enumerator that enumerates the language in string order, closure properties.
- To show undecidability: Diagonalization (rare), reduction from a known undecidable language (special cases: reduction using computation history, recursion theorem), closure properties.

T-Recognizable Languages

- To show recognizability: Recognizer (single tape TM, multi-tape TM, non-deterministic TM), reduction to a known recognizable language, enumerator, closure properties.
- To show unrecognizability: Reduction from a known unrecognizable language, closure properties.

co-T-Recognizable Languages

- To show co-recognizability: Co-recognizer (recognizer of the complement of the language), reduction to a known co-recognizable language, closure properties.
- To show un-co-recognizability: Reduction from a known un-co-recognizable language, proving the complement of the language is unrecognizable, closure properties.

Closure Properties

Class	$\cup$	$\circ$	$*$	$\cap$	$\overline{\cdot}$	$\cdot^R$ (reversal)	$\cap$ REG
Regular	Yes[1]	Yes[2]	Yes[3]	Yes[4]	Yes[5]	Yes[6]	Yes[4]
Context-free	Yes[7]	Yes[8]	Yes[9]	No[10]	No[11]	Yes[12]	Yes[13]
Decidable	Yes[14]	Yes[15]	Yes[16]	Yes[17]	Yes[18]	Yes[19]	Yes[20]
Recognizable	Yes[21]	Yes[22]	Yes[23]	Yes[24]	No[25]	Yes[26]	Yes[27]
Co-recognizable	Yes[28]	Yes[29]	Yes[30]	Yes[31]	No[32]	Yes[33]	Yes[34]

Proof ideas:

1) Combine the two DFAs. Create a new start state with ε-transitions to the start states of the original DFAs.

2) Make the accept states of one DFA non-accepting, and add ε-transitions from those states to the start state of the other DFA. Make the start state of first DFA the start state of the new machine.

3) Create a new (accepting) start state with an ε-transition to the old start state. Add ε-transitions from each accept state to the new start state.

4) Create a new DFA whose states are pairs of states (q_i, r_j), where q_i is a state in the first DFA and r_j is a state in the second DFA. The new DFA only accepts when both states in the pair are accepting in their respective machines.

5) Flip the accept and reject states of the DFA.

6) Flip the arrows in the DFA. Create a new start state with ε-transitions to the old accepting states. Make the old start state the only accepting state.

7) Combine the two CFGs and add a new rule: $S \longrightarrow S_1 \mid S_2$ where S is the new start variable and S_1 and S_2 are the start variables of the original CFGs.

8) Combine the two CFGs and add a new rule: $S \longrightarrow S_1 S_2$ where S is the new start variable and S_1 and S_2 are the start variables of the original CFGs.

9) Add a new rule to the CFG: $S \longrightarrow S_1 S \mid \varepsilon$ where S is the new start variable and S_1 is the start variable of the original CFG.

10) Consider the counter-example:
Let $A = \{0^*1^k2^k \mid k \geq 0\}$ and let $B = \{0^k1^k2^* \mid k \geq 0\}$. A and B are both context-free, but $C = A \cap B = \{0^k1^k2^k \mid k \geq 0\}$ is not context-free.

11) Consider the counter-example:
Let $C = \{0^k1^k2^k \mid k \geq 0\}$. C is not context-free, but its complement $\overline{C}$ is: A string of the form $0^k1^k2^k$ must satisfy three conditions: a) the symbols are in the right order, b) the number of zeros is equal to the number of ones, and c) the number of ones is equal to the number of twos. The PDA recognizing $\overline{C}$ non-deterministically guesses which condition the input string breaks and verifies that that condition is indeed broken before accepting the input.

12) For each rule in the CFG, rewrite the right-hand side of that rule from right to left.

13) Construct a new PDA to recognize the intersection. The new PDA operates like the PDA recognizing the CFL but also uses its finite control to recognize the regular language in parallel. The modification of the finite control to allow for this is similar to 4).

14) Construct a TM that simulates both deciders on the input string and accepts whenever either of them accepts.

15) Construct a TM that tries all possible splits of the input string. For each split, simulate the first decider on the first part of the string and the second decider on the second part. Accept if both deciders accept.

16) Construct a TM that tries all possible splits of the input string (the number of such splits is very large but still finite). The TM runs the decider on each part of the split, accepting when the decider accepts all parts.

17) Construct a TM that simulates both deciders on the input string and accepts when both of them accept.

18) Flip the accept and reject decisions of the decider.

19) Flip the input string at the beginning, and run the decider on the flipped string.

20) Construct a TM that simulates both the decider and the DFA of the regular language and accepts when both of them accept.

21) Simulate the two recognizers in parallel and accept if either of them accepts.

22) Construct a TM that tries all possible splits of the input string. For all splits (in parallel), simulate (in parallel) the first recognizer on the first part of the string and the second recognizer on the second part. Accept if both recognizers accept for some split.

23) Construct a TM that tries all possible splits of the input string (the number of such splits is very large but still finite). For all possible splits (in parallel), the TM runs the recognizer on each part of the split (again in parallel), accepting when the recognizer accepts all parts of some split.

24) Simulate the two recognizers in parallel and accept if both of them accept.

25) A_{TM} is recognizable but $\overline{A_{TM}}$ is not.

26) Flip the input string at the beginning, and run the recognizer on the flipped string.

27) Construct a TM that simulates both the recognizer and the DFA of the regular language and accepts when both of them accept.

28) Simulate the two co-recognizers in parallel and accept if either of them accepts.

29) Construct a TM that tries all possible splits of the input string. For all splits (in parallel), simulate (in parallel) the first co-recognizer on the first part of the string and the second co-recognizer on the second part. Accept if both co-recognizers accept for some split.

30) Construct a TM that tries all possible splits of the input string (the number of such splits is very large but still finite). For all possible splits (in parallel), the TM runs the co-recognizer on each part of the split (again in parallel), accepting when the co-recognizer accepts all parts of some split.

31) Simulate the two co-recognizers in parallel and accept if both of them accept.

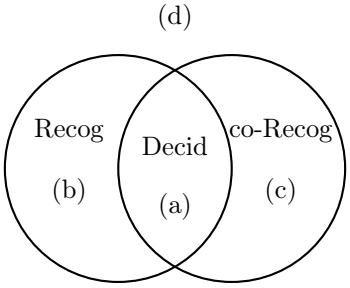
32) $\overline{A_{TM}}$ is co-recognizable but A_{TM} is not.

33) Flip the input string at the beginning, and run the co-recognizer on the flipped string.

34) Construct a TM that simulates both the co-recognizer and the DFA of the regular language and accepts when both of them accept.

Important Languages

	DFA	CFG	TM	LBA
Acceptance	A_{DFA} (a)[1]	A_{CFG} (a)[5]	A_{TM} (b)[9]	A_{LBA} (a)[13]
Emptiness	E_{DFA} (a)[2]	E_{CFG} (a)[6]	E_{TM} (c)[10]	E_{LBA} (c)[14]
Completeness	ALL_{DFA} (a)[3]	ALL_{CFG} (c)[7]	ALL_{TM} (d)[11]	ALL_{LBA} (c)[15]
Equality	EQ_{DFA} (a)[4]	EQ_{CFG} (c)[8]	EQ_{TM} (d)[12]	EQ_{LBA} (c)[16]



Proof Ideas:

1) Construct a decider: Simulate the DFA on the input string. Accept if the DFA accepts and reject otherwise.

2) Construct a decider: Check if any accept states are reachable from the start state. If yes, reject. Otherwise, accept.

3) Construct a decider: Check if any non-accepting states are reachable from the start state. If yes, reject. Otherwise, accept.

4) Construct a decider that checks if the symmetric difference of the two regular languages is empty: given two DFAs A and B, construct a DFA C that recognizes strings accepted by A or B but not both i.e. $L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$. Check if $L(C)$ is empty. Accept if yes. Reject otherwise.

5) Convert the CFG into Chomsky normal form. For an input string of length n, use the modified CFG to generate all derivations of length $2n - 1$ steps. Accept if any of the derivations generates the input string. Reject otherwise.

6) Mark all terminal symbols. Repeat until no new symbol is marked: for each rule in the CFG, if the right-hand side of the rule consists of only marked variables/terminal symbols, mark the variable on the left-hand side of the rule. At the end, check if the start variable is marked. If yes, reject. Otherwise, accept.

7) Undecidable: Reduce from A_{TM} to the equivalent language ALL_{PDA}. Given a TM M and a string w, construct a PDA that accepts all strings except accepting computation histories of M on w. If the PDA accepts all strings, then M rejects w. Otherwise, M accepts w.
Co-recognizable: Construct a TM that recognizes $\overline{ALL_{CFG}}$: Enumerate all strings and use the decider of A_{CFG} to check if the CFG can generate each string. If the CFG can't generate some string, accept.

8) Undecidable: Reduce from ALL_{CFG}. Assume that some decider D decides EQ_{CFG}. Then given a CFG G, we can decide if G generates all strings by running D on $\langle G, G_{all} \rangle$, where G_{all} is the grammar that generates Σ^*. But ALL_{CFG} is undecidable $\longrightarrow$ contradiction.
Co-recognizable: Construct a TM that recognizes $\overline{EQ_{CFG}}$: Enumerate all strings and use the decider of A_{CFG} to check if the two CFGs both generate or both can't generate each string. If the two CFGs differ in behavior on some string, accept.

9) Undecidable: Diagonalization.
Recognizable: Simulate the TM on its input string and return the same result.

10) Undecidable: Reduce from A_{TM}. Assume some decider D decides E_{TM}. Given a TM M and a string w, construct a TM M_w that rejects all strings $x \neq w$ and on w simulates M and returns the same result. Using D to check if $L(M_w) = \phi$ decides A_{TM}, leading to a contradiction.
Co-recognizable: Construct a TM that recognizes $\overline{E_{TM}}$: Given a TM M, enumerate all strings and check (through simulation) if M accepts any of them. If yes, accept. Note that simulating M on all strings in order might result in M looping on some string. So instead, simulate M on the first k strings for k steps for $k = 1, 2 \ldots$

11) Unrecognizable: Show that $\overline{A_{TM}} \leq_m ALL_{TM}$. Given a TM M and a string w, construct a TM M_w that accepts all strings except accepting computation histories of M on w.
$\langle M, w\rangle \in \overline{A_{TM}} \iff f(\langle M, w\rangle) = \langle M_w\rangle \in ALL_{TM}$
Un-co-recognizable: Show that the complement of the language is unrecognizable
i.e. $\overline{A_{TM}} \leq_m \overline{ALL_{TM}} \iff A_{TM} \leq_m ALL_{TM}$. Given a TM M and a string w, construct TM M_w that ignores its input and instead simulates M on w and returns the same result.
$\langle M, w\rangle \in A_{TM} \iff f(\langle M, w\rangle) = \langle M_w\rangle \in ALL_{TM}$

12) Unrecognizable: Show that $\overline{A_{TM}} \leq_m EQ_{TM}$. Given a TM M and an input string w, construct the TM M_w that ignores its input and instead simulates M on w and returns the same result. Then $\langle M, w\rangle \in \overline{A_{TM}} \iff f(\langle M, w\rangle) = \langle M_w, M_{reject}\rangle \in EQ_{TM}$
where M_{reject} is the TM that rejects all input strings.
Un-co-recognizable: Show that $\overline{A_{TM}} \leq_m \overline{EQ_{TM}} \iff A_{TM} \leq_m EQ_{TM}$. Given a TM M and an input string w, construct the TM M_w that ignores its input and instead simulates M on w and returns the same result. Then $\langle M, w\rangle \in A_{TM} \iff f(\langle M, w\rangle) = \langle M_w, M_{accept}\rangle \in EQ_{TM}$
where M_{accept} is the TM that accepts all input strings.

13) Simulate the LBA on its input string for $n|Q||\Gamma|^n$ steps (where n is the length of the input string, Q is the set of states of the LBA, and Γ is the tape alphabet of the LBA). If the LBA accepts within that time, accept. Otherwise, reject.

14) Undecidable: Reduce from A_{TM}. Assume that some decider D decides E_{LBA}. Then given a TM M and an input string w, construct an LBA N that only accepts strings that are accepting computation histories of M on w. Using D to check if $L(N) = \phi$ would then decide A_{TM}, which is a contradiction.
Co-recognizable: Construct a TM that recognizes $\overline{E_{LBA}}$: Enumerate all strings and use the decider of A_{LBA} to check if the LBA accepts each string. If the LBA accepts some string, accept.

15) Undecidable: Reduce from A_{TM}. Assume that some decider D decides ALL_{LBA}. Then given a TM M and an input string w, construct an LBA M_w that accepts all strings except accepting computation histories of M on w. D can be used to check if $L(M_w) = \Sigma^*$, and by extension, to decide A_{TM}. But A_{TM} is undecidable $\longrightarrow$ contradiction.
Co-recognizable: Construct a TM that recognizes $\overline{ALL_{LBA}}$: Enumerate all strings and use the decider of A_{LBA} to check if the LBA accepts each string. If the LBA doesn't accept some string, accept.

16) Undecidable: Reduce from A_{TM}. Assume that some decider D decides EQ_{LBA}. Then given a TM M and an input string w, construct an LBA M_w that accepts all strings except accepting computation histories of M on w. D can be used to check if $L(M_w) = L(M_{all})$, where M_{all} is the LBA that accepts all its input strings. This means that D can be used to decide A_{TM}. But A_{TM} is undecidable $\longrightarrow$ contradiction.
Co-recognizable: Construct a TM that recognizes $\overline{EQ_{LBA}}$: Enumerate all strings and use the decider of A_{LBA} to check if the two LBAs return the same result on each string. If the two LBAs differ in behavior on some string, accept.

Other Useful Concepts

- **Mapping Reductions:** $A \leq_m B \iff \overline{A} \leq_m \overline{B}$ (using the same reduction). If $A \leq_m B$ then:
  - If B is decidable/recognizable/co-recognizable, then A is also decidable/recognizable/co-recognizable.
  - If A is undecidable/unrecognizable/un-co-recognizable, then B is also undecidable/unrecognizable/un-co-recognizable.
- **Recursion Theorem:** A TM can obtain a string representation of itself.
- **Counting Configurations:** If some machine can only have a finite number of configurations, we can use that number to determine if the machine is looping.
- Simulating a machine on the first k strings for k steps where k = 1, 2, ...: Useful when we want to check if *some* string is accepted/rejected by the machine, and we're worried about the machine looping on one of those strings during its operation.