18.404 midterm review sessions

HALLO! — We'll offer problem-solving practice sessions,° in-person. These midterm review sessions are **completely optional**. It's easy to imagine a non-attendee earning an excellent midterm grade.

← Coordinates:
Mon 7:30-9pm in room 2-190 with Zed, Sarah
Tue 7:30-9pm in room 2-190 with Leo, Nathan

If you choose to work through these practice problems, you may want to play a kind of **Concept Bingo**. Notice whether and in what context you use the following concepts:°

← Your solutions to these practice problems will probably miss a few of these concepts. That's okay. These problems support multiple nice solutions; different solutions will use different concepts in this list.

AUTOMATA

- ☐ closure properties for regular languages and context-free languages
- ☐ pigeonhole principle on set of states (e.g. pumping lemma)
- ☐ equivalence of FAs and regular expressions
- ☐ equivalence of PDAs and CFGs

COMPUTABILITY, ABSTRACT PRINCIPLES

- ☐ closure properties for decidable languages and recognizable languages
- ☐ properties of mapping reductions (e.g. transitivity, complement)
- ☐ simulation of one TM variant by another

COMPUTABILITY, CONCRETE EXAMPLES

- ☐ decidability of $A_{\mathrm{LBA}}, A_{\mathrm{PDA}}, E_{\mathrm{PDA}}$
- ☐ undecidability of $A_{\mathrm{TM}}, ALL_{\mathrm{TM}}, E_{\mathrm{TM}}, ALL_{\mathrm{LBA}}, E_{\mathrm{LBA}}, ALL_{\mathrm{PDA}}$
- ☐ recognizability of $A_{\mathrm{TM}}, \overline{E_{\mathrm{TM}}}, \overline{ALL_{\mathrm{LBA}}}, \overline{E_{\mathrm{LBA}}}, \overline{ALL_{\mathrm{PDA}}}$

COMPUTABILITY, REDUCTION TECHNIQUES

- ☐ simulation for creating mapping reductions
- ☐ equivalence of enumerators and TM recognizers
- ☐ computation history method

What's above is an INcomplete list of concepts. Everything from class so far (excluding time complexity) is fair game.

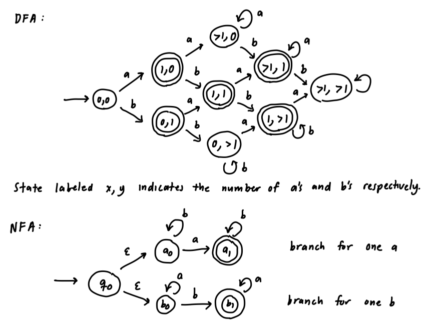
regular languages

showing regularity

Let $L = \{w$ containing exactly one `a` or exactly one `b`$\}$.
Show that L is regular by explicitly giving:

- ☐ a regular expression;
- ☐ an NFA or DFA.

Solution — This regular expression works: $(\mathtt{b}^\star\mathtt{a}\mathtt{b}^\star) \cup (\mathtt{a}^\star\mathtt{b}\mathtt{a}^\star)$. We can directly read off an NFA from that regular expression. This NFA nondeterministically guesses whether to check for unique `a`'s or unique `b`'s; the checks themselves each have three arrows, representing the three concatenated terms in the regexp's parentheses. See figure in margin. (You could also construct a DFA, but the NFA construction is simpler.) ■



pumping lemma question

Let $L = \{w$ contains only `0`'s and $|w| = k^2$ for some $k \in \mathbb{N}\}$. Show that L is not regular.

Solution — We use the pumping lemma.

Suppose L is regular and it has pumping length p. Then take $s = \mathtt{0}^{(p^2)} \in L$. By the pumping lemma, we can pump some substring of length $k \le p$ and it should stay in the language. But $\mathtt{0}^{p^2+k}$ for $k \le p$ is not square (since the next square after p^2 is $p^2 + 2p + 1 > p^2 + k$), so it is not in L. By contradiction, L is not regular. ■

regular languages with lots of 1s

Call a language **major** if it includes some string s with with strictly more `1`s than `0`s. ◦ For example, the language $\mathtt{0}^\star\mathtt{1}^\star$ is major; the language $(\mathtt{0} \cup \mathtt{10})^\star$ is not major.

← This concept seems to involve (unbounded) *counting*. Which machines that we discussed can count? For instance, can DFAs count?

- ☐ Is $L = \{\langle M\rangle \mid M$ is a DFA with a major language$\}$ decidable?

Solution — YES. A DFA M's language is major precisely when $\mathcal{L}(M) \cap F$ is non-empty, where $F = \{x \mid x$ has strictly more `1`s than `0`s$\}$. F is a CFL — we give a grammar in the margin. ◦ There is also a simple PDA that counts the number of `0`'s and 1's.

← The following grammar has language F:

$$S \to SS \mid T1 \qquad T \to 0T1 \mid 1T0 \mid TT \mid \epsilon$$

By induction on string-length, T generates all and only strings with an equal number of `0`'s and 1's. So by induction on string-length, S's language is F.

Using the construction showing CFLs are closed under intersection with regular languages, we can algorithmically construct a CFG whose language is $\mathcal{L}(M) \cap F$. Now, we can use our decider for E_{CFG}, and output the negation of its answer. ■

context-free languages

showing context-freeness

☐ Is $L = \{w\#x$ such that w^R is a substring of $x\}$ context-free?

SOLUTION — YES. Let's make a PDA. Push to the stack until you see #, then nondeterministically decide when to start popping; as you pop, compare to the input. If the stack symbol and input symbol are ever different, reject. If the stack is empty after seeing #, accept. ■

SOLUTION — YES. Let's make a CFG. We can describe L as containing all strings of the form $w\#aw^Rb$, i.e., palindromes with fluff added on either side of the right half. Without the a and b, this is just palindromes centered on #, which $P \to 0P0 \mid 1P1 \mid \#$ generates. We want to put fluff immediately after the # and as a suffix to the overall string. The rules to the right implement this idea. ■

$$S \to PF$$
$$P \to 1P1 \mid 0P0 \mid \#F$$
$$F \to F1 \mid F0 \mid \epsilon$$

closure properties

Suppose A is context free, B is not context free. Which of the following are possible:

☐ $B = A \cap C$ for C regular?

SOLUTION — NO, because CFLs are closed under intersection with regular languages (so B would have to be context-free). ■

☐ $B = A \cap C$ for C context free?

SOLUTION — YES, because CFLs are not closed under intersection (recall the example $A = a^n b^n c^*$, $C = a^* b^n c^n$). ■

☐ $B = A \cup C$ for C regular?

SOLUTION — NO, because a regular language is also CFLs and CFLs are closed under union. ■

☐ $B = A \cup C$ for C context free?

SOLUTION — NO, because CFLs are closed under union. ■

mapping reductions

the butterfly's head

Remember: A_{TM} and $\overline{A_{TM}}$ lie on opposite wings° of our butterfly diagram.

← This is a poetic way to say that A_{TM} is recognizable but its complement $\overline{A_{TM}}$ is not.

☐ True or false? $\overline{A_{TM}} \leq_m A_{TM}$.

SOLUTION — FALSE.° We can't m-reduce a non-recognizable language to a recognizable one. ■

← By complementing both sides of the $\leq_m$ sign, we get the corollary that $A_{TM} \leq_m \overline{A_{TM}}$ is also false.

Can we invent some problem (i.e., language) that's harder than both A_{TM} and $\overline{A_{TM}}$? Well, we could make the problem's solutions encode solutions both to A_{TM} and to $\overline{A_{TM}}$. So let $J = 0A_{TM} \cup 1\overline{A_{TM}}$. That's shorthand for:

$$J = \{b(\langle M \rangle, w) \text{ such that TM } M \text{ accepts } w \text{ if and only if bit } b \text{ equals } \texttt{0}\}$$

Then J sits atop both A_{TM} and its complement in the butterfly:° $A_{TM} \leq_m J$ and $\overline{A_{TM}} \leq_m J$. Moreover, J is on the axis of symmetry: $J \leq_m \overline{J}$.°

← ☐ [A] Prove this!

← ☐ [B] And prove this, too!

SOLUTION — [A]: Mapping reduce by prepending `0` (resp., `1`). ■

SOLUTION — [B]: Mapping reduce by inverting the input's first bit. Note that this inversion preserves well-formedness of strings. ■

☐ Finally, is it true that $\overline{E_{LBA}} \leq_m J$?

SOLUTION — YES, it's true. Since $\overline{E_{LBA}}$ is recognizable, it mapping reduces to A_{TM}. So: $\overline{E_{LBA}} \leq_m A_{TM} \leq_m J$. ■

ahh! shh!

Consider these two languages:°

← You'll notice that AH and SH seem dual. But we've learned to tread with care when making such comparisons.

☐ What goes wrong with the following attempt to reduce $AH \leq_m \overline{SH}$?

Reduction attempt: *map $\langle M \rangle$ to $\langle \tilde{M} \rangle$ where $\tilde{M}$ simulates M and loops forever if $\tilde{M}$ halts, else halt. Then $\langle M \rangle \in AH$ if and only if $\langle \tilde{M} \rangle \notin SH$.*

SOLUTION — We can't flesh out that pseudocode into code: there is no way to implement the conditional "if $\tilde{M}$ halts". Or, if we test this condition by naive simulation, M might loop in which case we never reach the code after the simulation that asks us to halt. ■

$$AH = \{\langle M \rangle \text{ such that } M \text{ is a TM that halts on } \textit{all} \text{ inputs}\}$$

$$SH = \{\langle M \rangle \text{ such that } M \text{ is a TM that halts on } \textit{some} \text{ input}\}$$

☐ Is AH recognizable? How about its complement?

SOLUTION — NEITHER AH nor $\overline{AH}$ is recognizable, since we can mapping reduce both $\overline{A_{TM}}$ and A_{TM} to AH. Given $(\langle M \rangle, w)$, make M' that on input x simulates M on w for $|x|$ steps, then loops forever iff M has by then accepted. Then M' always halts precisely when M does not accept w, so we've verified $\overline{A_{TM}} \leq AH$. Given $(\langle M \rangle, w)$, make $\tilde{M}$ that on input x simulates M on w, then loops forever unless M accepts w. Then $\tilde{M}$ always halts precisely when M accepts w, so we've verified $A_{TM} \leq AH$.
■

☐ Is SH Turing-recognizable? How about its complement?

SOLUTION — ONLY SH, not its complement, is Turing-recognizable. The following TM recognizes SH: Given input $\langle M \rangle$, for each number-string pair (k, s), simulate M on s for k steps and accept if M has halted within these k steps. To show that $\overline{SH}$ is not Turing-recognizable, we mapping reduce $\overline{A_{TM}} \leq_m \overline{SH}$. Given $(\langle M \rangle, w)$, make $\tilde{M}$ that on input x simulates M on w, then loops forever unless M accepts w.° Then $\tilde{M}$ sometimes halts precisely when M accepts w, so we've verified $\overline{A_{TM}} \leq_m \overline{SH}$. ■

← Observe: this is the same construction and argument as in the AH problem.

a mapping reduction edge case

☐ True or false? Whenever A is decidable and B is regular: $A \leq_m B$.

Solution — NO, due to the cases $B = \{\}$ and $B = \Sigma^*$. A mapping function must, by definition, map strings in A to strings in B, and strings not in A map to strings not in B; but this is impossible when, for example, there are strings in A but *there are no strings in* B! ■

all together now

double the tapes is double the fun

For this problem, we define a *2-tape DFA*. A 2-tape DFA has a pair of inputs (x, y), presented on two (finite) tapes, with a read-only, left-to-right head on each tape. The transition function is a mapping $Q \times (\Sigma \times \Sigma) \to Q \times \{\texttt{Head 1}, \texttt{Head 2}\}$, to indicate the new state and which head to step right. The machine accepts if the machine is in an accepting state when one of the heads moves off the end of its tape.

We'll be interested in showing that the following language is decidable:

$$\text{ALL}_{\text{2-tape DFA}} = \{\langle M \rangle | M \text{ is a 2-tape DFA that accepts every input pair } (x, y)\}.$$

☐ Given a 2-tape DFA M, show how to construct a PDA P with alphabet $\Sigma \cup \{\#\}$, such that $L(P)$ is empty if and only if M accepts every (x, y).

Solution — Consider a PDA that pushes symbols from the input onto the stack until it sees a #. Then, it treats the stack and the remaining portion of the input as the two tapes of the 2-tape DFA, simulates M on that pair of inputs, and outputs the reverse of its answer. ◦ Observe that, if this PDA is given input $x^{\mathcal{R}}\#y$ for $x, y \in \Sigma^*$, it will accept if and only if M rejects (x, y). On inputs with zero or more than one copies of #, we will just have the PDA always reject.

← Moving the first head right corresponds to popping the top input off of the stack, while moving the second head right corresponds to the PDA moving right on its input.

This PDA will accept some input if and only if there is some (x, y) that M rejects, so $L(A)$ will be empty if and only if M accepts every input. ■

☐ Using the above, conclude that $\text{ALL}_{\text{2-tape DFA}}$ is dedidable.

Solution — The above gives us a reduction showing $\text{ALL}_{\text{2-tape DFA}} \leq_m E_{\text{PDA}}$. Recall that E_{PDA} is equivalent to E_{CFG}, which we've shown is decidable. So, $\text{ALL}_{\text{2-tape DFA}}$ is decidable. ■

once more, with nondeterminism!

We'll now make a slight modification to the setup: we instead consider 2-tape *NFA*s. The picture is the same as before, except now the transition function is nondeterministic — i.e. it is of the form $Q \times (\Sigma \times \Sigma) \to \mathcal{P}(Q \times \{\texttt{Head 1}, \texttt{Head 2}\})$. As before, we define the ALL problem:

$$\text{ALL}_{\text{2-tape NFA}} = \{\langle M \rangle | \ M \text{ is a 2-tape NFA that accepts every input pair } (x, y)\}.$$

Interestingly, this modification of the model actually changes the story entirely.

☐ Show that $\mathrm{ALL}_{\text{2-tape NFA}}$ is undecidable.

Solution — We use the computation history method. Fixing an arbitrary Turing machine T and input w, our goal will be to design a 2-tape NFA that rejects if and only if its input (x, y) is a well-formed encoding of an accepting computation history of T on w.

The encoding we consider will make use of special symbols # (as a delimiter between states) and \$ (as an end-of-tape marker). We want our 2-NFA to reject if its input is

$$x = y = C_1\#C_2\#\ldots\#C_t\$$$

for C_1, C_2, ..., C_t an accepting computation history of T on w. To do so, we will have our machine nondeterministically choose one of the following options:

- Move the two heads right in sync, and accept if x and y disagree on any index.
- Accept if the input is badly formed in the sense of including a \$ in the middle or not including a \$ at the end (this can be checked with just one head).
- Accept if the C_1 isn't the starting configuration of T on w, or if C_t doesn't have T's accepting state (this can be checked with just one head).
- Move the two heads in sync until non-deterministically choosing, on some #, to instead leave the first head in place and move the second head to the next #. Moving them in sync again, now the first head is reading C_i and the second is reading C_{i+1} — accept if they find an offset where the configuration transition is invalid.

If $x = y = C_1\#C_2\#\ldots\#C_t\$$ for C_1, C_2, ..., C_t an accepting computation history of T on w, then none of those 4 conditions will hold, and so the 2-tape NFA will reject in every nondeterministic branch, and thus will reject overall. On the other hand, if there exists no accepting computation history of T on w, every possible input to this 2-tape NFA will satisfy at least one of the 4 conditions above, and will thus be accepted. So, this 2-tape NFA accepts all strings iff T rejects w. This reduction shows $\overline{A_{TM}} \leq_m \mathrm{ALL}_{\text{2-tape NFA}}$, so $\mathrm{ALL}_{\text{2-tape NFA}}$ is undecidable. ■