University of Nevada, Las Vegas Computer Science 456/656 Spring 2025

Answers to Practice Problems for the Final Examination May 14, 2025

Throughout, $\mathcal{P}$ means $\mathcal{P}$–TIME.

Despite my efforts, this list of problems may contain duplicates. Sorry about that!

1. Consult the lists of true false questions that are posted in Handouts.
2. True or False. T = true, F = false, and O = open, meaning that the answer is not known science at this time. In the questions below, $\mathcal{P}$ and $\mathcal{NP}$ denote $\mathcal{P}$-TIME and $\mathcal{NP}$-TIME, respectively.
   (i) **T** The union of any two regular languages is regular.
   (ii) **T** The intersection of two regular languages is regular.
   (iii) **T** The concatenation of two regular languages is regular.
   (iv) **T** The complement of any regular language is regular.
   (v) **T** The Kleene closure of any regular language is regular.
   (vi) **T** The union of two context-free languages is context-free.
   (vii) **F** The intersection of two context-free languages is context-free.
   (viii) **T** The concatenation of two context-free languages is context-free.
   (ix) **F** The complement of any context-free language is context-free.
   (x) **T** The Kleene closure of any context-free language is context-free.
   (xi) **T** The union of two $\mathcal{P}$ languages is $\mathcal{P}$.
   (xii) **T** The intersection of two $\mathcal{P}$ languages is $\mathcal{P}$.
   (xiii) **T** The concatenation of two $\mathcal{P}$ languages is $\mathcal{P}$.
   (xiv) **T** The complement of any $\mathcal{P}$ language is $\mathcal{P}$.
   (xv) **T** The Kleene closure of any $\mathcal{P}$ language is $\mathcal{P}$.
   (xvi) **T** The union of two $\mathcal{NP}$ languages is $\mathcal{NP}$.
   (xvii) **T** The intersection of two $\mathcal{NP}$ languages is $\mathcal{NP}$.
   (xviii) **T** The concatenation of two $\mathcal{NP}$ languages is $\mathcal{NP}$.
   (xix) **O** The complement of any $\mathcal{NP}$ language is $\mathcal{NP}$.
   (xx) **T** The Kleene closure of any $\mathcal{NP}$ language is $\mathcal{NP}$.
   (xxi) **T** The union of two decidable languages is decidable.

(xxii) **T** The intersection of two decidable languages is decidable.

(xxiii) **T** The concatenation of two decidable languages is decidable.

(xxiv) **T** The complement of any decidable language is decidable.

(xxv) **T** The Kleene closure of any decidable language is decidable.

(xxvi) **T** The union of two $\mathcal{RE}$ languages is $\mathcal{RE}$.

(xxvii) **T** The intersection of two $\mathcal{RE}$ languages is $\mathcal{RE}$.

(xxviii) **T** The concatenation of two $\mathcal{RE}$ languages is $\mathcal{RE}$.

(xxix) **F** The complement of any $\mathcal{RE}$ language is $\mathcal{RE}$.

(xxx) **T** The Kleene closure of any $\mathcal{RE}$ language is $\mathcal{RE}$.

(xxxi) **F** The union of two undecidable languages is undecidable.

(xxxii) **F** The intersection of two undecidable languages is undecidable.

(xxxiii) **T** The complement of any undecidable language is undecidable.

(xxxiv) **F** The Kleene closure of any undecidable language is undecidable.

(xxxv) **T** The union of any two $\mathcal{NC}$ languages is $\mathcal{NC}$.

(xxxvi) **T** The intersection of two $\mathcal{NC}$ languages is $\mathcal{NC}$.

(xxxvii) **T** The concatenation of $\mathcal{NC}$ languages is $\mathcal{NC}$.

(xxxviii) **T** The complement of any $\mathcal{NC}$ language is $\mathcal{NC}$.

(xxxix) **T** The Kleene closure of any $\mathcal{NC}$ language is $\mathcal{NC}$.

(xl) **O** $\mathcal{NC}. = \mathcal{P}$.

(xli) **F** $\mathcal{NC}. = \mathcal{P}$–SPACE.

(xlii) **O** $\mathcal{P} = \mathcal{NP}$.

(xliii) **O** $\mathcal{P} = \mathcal{P}$–SPACE.

(xliv) **F** $\mathcal{P}$ = EXP–TIME.

(xlv) **F** $\mathcal{P}$–SPACE = EXP–SPACE.

(xlvi) **T** There is a PDA that accepts the language of all palindromes over $\{a, b\}$.

(xlvii) **F** If every $w \in L$ can be proved to be in L, then L must be decidable.

(xlix) **T** The language $\{a^n b^n \mid n \geq 0\}$ is context-free.

(l) **F** The language $\{a^n b^n c^n \mid n \geq 0\}$ is context-free.

(li) **T** The language $\left\{a^i b^j c^k \mid j = i + k\right\}$ is context-free.

(lii) **T** The intersection of any regular language with any context-free language is context-free.

(liii) **T** If L is a context-free language over an alphabet with just one symbol, then L is regular.

(liv) **T** The set of strings that your high school algebra teacher would accept as legitimate expressions is a context-free language.

(lv) **T** Every language accepted by a non-deterministic machine is accepted by some deterministic machine.

(lvi) **T** The problem of whether a given string is generated by a given context-free grammar is decidable.

(lvii) **F** Every language generated by an unambiguous context-free grammar is accepted by some DPDA.

(lviii) **T** The language $\{a^n b^n c^n \mid n \geq 0\}$ is in the class $\mathcal{P}$-TIME.

(lix) **O** There exists a polynomial time algorithm which finds the factors of any positive integer, where the input is given as a binary numeral.

(lx) **T** Every $\mathcal{NP}$ language is decidable.

(lxi) **O** $\mathcal{NP} = \mathcal{P}$-SPACE

(lxii) **O** EXP-TIME = EXP-SPACE

(lxiii) **T** The traveling salesman problem (TSP) is $\mathcal{NP}$–complete.

(lxiv) **T** The knapsack problem is $\mathcal{NP}$–complete.

(lxv) **T** The language consisting of all satisfiable Boolean expressions is $\mathcal{NP}$-complete.

(lxvi) **T** The Boolean Circuit Problem is in $\mathcal{P}$.

(lxvii) **O** The Boolean Circuit Problem is in $\mathcal{NC}$.

(lxviii) **T** 2-SAT is $\mathcal{P}$-TIME.

(lxix) **O** 3-SAT is $\mathcal{P}$-TIME.

(lxx) **T** Primality, using binary numerals, is $\mathcal{P}$-TIME.

(lxxi) **T** Every context-free language is in $\mathcal{P}$.

(lxxii) **T** Every context-free language is in $\mathcal{NC}$.

(lxxiii) **F** Every language generated by an unrestricted grammar is recursive.

(lxxiv) **F** The problem of whether two given context-free grammars generate the same language is decidable.

(lxxv) **F** The language of all fractions (using base 10 numeration) whose values are less than π is decidable.

(lxxvi) **T** There exists a polynomial time algorithm which finds the factors of any positive integer, where the input is given as a unary ("caveman") numeral.

(lxxvii) **T** For any two languages L_1 and L_2, if L_1 is undecidable and there is a recursive reduction of L_1 to L_2, then L_2 must be undecidable.

(lxxviii) **F** If P is a mathematical proposition that can be written using a string of length n, and P has a proof, then P must have a proof whose length is $O(2^{2^n})$.

(lxxix) **T** If L is any $\mathcal{NP}$ language, there must be a $\mathcal{P}$–TIME reduction of L to the partition problem.

(lxxx) **F** Every bounded function is recursive.

(lxxxi) **O** If L is $\mathcal{NP}$ and also co-$\mathcal{NP}$, then L must be $\mathcal{P}$.

(lxxxii) **T** If a language L is in $\mathcal{RE}$ and also in co-$\mathcal{RE}$, then L must be decidable.

(lxxxiii) **T** Every language is enumerable.

(lxxxiv) **F** If a language L is undecidable, then there can be no machine that enumerates L.

(lxxxv) **T** There is a non-recursive function which grows faster than any recursive function.

(lxxxvi) **T** There exists a machine that runs forever and outputs the string of decimal digits of π (the well-known ratio of the circumference of a circle to its diameter).

(lxxxvii) **O** **Rush Hour**, the puzzle sold in game stores everywhere, generalized to a board of arbitrary size, is $\mathcal{NP}$–complete.

(lxxxviii) **O** There is a polynomial time algorithm which determines whether any two regular expressions are equivalent.

(lxxxix) **O** If two regular expressions are equivalent, there is a polynomial time proof that they are equivalent.

(xc) **F** Every subset of a regular language is regular.

(xci) **F** Let L be the language over $\{a, b, c\}$ consisting of all strings which have more a's than b's and more b's than c's. There is some PDA that accepts L.

(xcii) **T** Every subset of any enumerable set is enumerable.

(xciii) **T** If L is a context-free language which contains the empty string, then $L \backslash \{\lambda\}$ must be context-free.

(xciv) **T** The computer language C++ has Turing power.

(xcv) **T** Let Σ be the binary alphabet. Every $w \in \Sigma^*$ which starts with 1 is a binary numeral for a positive integer. Let $Sq : \Sigma^* \to \Sigma^*$ be a function which maps the binary numeral for any integer n to the binary numeral for n^2. Then Sq is an $\mathcal{NC}$ function.

(xcvi) **T** If L is any $\mathcal{P}$-TIME language, there is an $\mathcal{NC}$ reduction of L to the Boolean circuit problem.

(xcvii) **T** If an abstract Pascal machine can perform a computation in polynomial time, there must be some Turing machine that can perform the same computation in polynomial time.

(xcviii) **T** The binary integer factorization problem is co-$\mathcal{NP}$.

(xcix) **F** Every subset of a regular language is decidable.

(ci) **O** The independent set problem is $\mathcal{P}$-TIME.

(cii) **T** If S is a set of positive integers whose set of binary numerals decidable, then $\sum_{n \in S} 2^{-n}$ must be a recursive real number.

(ciii) **T** Multiplication of matrices with binary numeral entries is $\mathcal{NC}$.

(civ) **T** Equivalence of regular expressions is decidable.

(cv) **T** Equivalence of context-free grammars is co–$\mathcal{RE}$.

(cvi) **T** The language consisting of all fractions whose values are less than the natural logarithm of 5.0 is recursive.

(cvii) **T** Every sliding block problem is $\mathcal{P}$–SPACE.

(cix) **F** There are uncountably many co–$\mathcal{RE}$ binary languages.

(cx) **T** If L is any $\mathcal{P}$–TIME language, there is an $\mathcal{NC}$ reduction of L to CVP, the Boolean circuit problem.

(cxi) **T** Every finite language is regular.

(cxii) **T** If L is a $\mathcal{P}$–TIME language, there is a Turing Machine which decides L in polynomial time.

(cxiii) **T** If anyone ever finds a polynomial time algorithm for any $\mathcal{NP}$–complete language, then we will know that $\mathcal{P} = \mathcal{NP}$.

(cxiv) **T** RSA encryption is believed to be secure because it is believed that the factorization problem for integers is very hard.

(cxv) **F** If S is a set of positive integers whose set of binary numerals is recursively enumerable, then $\sum_{n \in S} 2^{-n}$ must be a recursive real number. (Hint: This problem is very hard.)

(cxvi) **F** There is some PDA that accepts $\left\{w \in \{a, b, c\}^* : \#_a(w) > \#_b(w) > \#_c(w)\right\}$, that is, the language over $\{a, b, c\}$ consisting of all strings which have more a's than b's and more b's than c's.

(cxvii) **T** If L is $\mathcal{RE}$ and $w \in L$, there is a proof that $w \in L$.

(cxx) **T** The language $\{a^n b^n c^n \mid n \geq 0\}$ is in the class $\mathcal{NC}$.

(cxxi) **F** Every problem that can be mathematically defined has an algorithmic solution.

(cxxii) **O** $\mathcal{NC} = \mathcal{P}$.

(cxxiii) **T** The set of binary numerals for prime numbers is $\mathcal{P}$-TIME.

(cxxiv) **T** There is a mathematical proposition that is true but cannot be proved true.

(cxxvi) **T** If L is $\mathcal{NP}$, there is a polynomial time reduction of L to the subset sum problem.

(cxxvii) **T** The intersection of any two $\mathcal{NP}$ languages is $\mathcal{NP}$.

(cxxviii) **T** The intersection of any two co-$\mathcal{NP}$ languages is co-$\mathcal{NP}$.

(cxxix) **F** The intersection of any two co-$\mathcal{RE}$ languages is co-$\mathcal{RE}$.

(cxxx) **T** Multiplication of matrices with binary numeral entries is $\mathcal{NC}$.

(cxxxi) **T** Every recursively enumerable language is generated by an unrestricted (general) grammar.

(cxxxii) **T** Equivalence of context-free grammars is co–$\mathcal{RE}$.

(cxxxiii) **F** The language of all true mathematical statements is recursively enumerable.

(cxxxiv) **T** The language of all **provably** true mathematical statements is recursively enumerable.

(cxxxv) **T** There are uncountably many undecidable languages over the binary alphabet.

(cxxxvii) **T** The language of all $\langle G_1 \rangle \langle G_2 \rangle$ such that G_1 and G_2 are CF grammars which are **not** equivalent is $\mathcal{RE}$.

(cxxxviii) **T** A real number x is recursive if and only if the set of fractions whose values are greater than x is recursive (decidable).

(cxxxix) **F** For any real number x, there exists a machine that runs forever and outputs the string of decimal digits of x.

(cxl) **T** If a Boolean expression is satisfiable, there is a $\mathcal{P}$–TIME proof that it's satisfiable.

(cxli) **O** If there is a solution to a given instance of any sliding block problem, there must be a solution of polynomial length.

(cxlii) **T** If the Boolean circut problem (CVP) is $\mathcal{NC}$, then $\mathcal{P} = \mathcal{NC}$.

3. Give a definition of each term.

   (i) Accept. (That is, what does it mean for a machine to accept a language.) "M accepts L" means that, if M is given input w, then M will halt in an accepting state if and only if $w \in L$.

   (ii) Decide. (That is, what does it mean for a machine to decide a language.) "M decides L" means that, if M is given input w, if $w \in L$, M will halt in an accepting state, while if $w \notin L$, M will halt, but not in an accepting state.

(iii) If Σ is an ordered alphabet and $u, v \in \Sigma^*$, we say $u < v$ in the canonical order if either $|u| < |v|$ or u is earlier than v in alphabetical order.

(iv) Give the verifier-certificate definition of the class $\mathcal{NP}$.

A language L is $\mathcal{NP}$ if and only if there exists an integer k and a machine V (called the verifier) such that:

i. for every string $w \in L$ there exists a string c, called the certificate of w, such that V accepts the string w, c in $\mathcal{P}$–TIME.

ii. For every string $w \notin L$ and any string c, V does not accept the string w, c.

State the pumping lemma for regular languages.

For any regular language L There exists a positive number p such that
For any $w \in L$ of length at least p
There exist strings x, y, z such that
1. $w = xyz$,
2. $|xy| \leq p$,
3. y is not the empty string,
4. For any $i \geq 0$ $xy^iz \in L$.

(v) State the pumping lemma for context-free languages.

For any context-free language L There exists a positive number p such that
For any $w \in L$ of length at least p
There exist strings u, v, x, y, z such that
1. $w = uvxyz$,
2. $|vxy| \leq p$,
3. v and y is not both the empty string,
4. For any $i \geq 0$ $uv^ixy^iz \in L$.

(vi) What is the importance nowdays of $\mathcal{NC}$?

Parallel computing is more common, and it is important to know which algorithms can be efficiently parallelized.

(vii) State the Church-Turing thesis.

Every machine is equivalent to some Turing machine.

4. Which class of languages does each of these machine classes accept?

(i) Deterministic finite automata.

Regular languages.

(ii) Non-deterministic finite automata.

Regular languages.

(iii) Push-down automata.

Context-free languages.

(iv) Turing Machines.

Recursively Enumerable languages.

5. The LALR parser given for this grammar:
   1. $E \to E -_2 E_3$
   2. $E \to E *_4 E_5$
   3. $E \to x_6$

   contains errors, meaning that it might parse a string in a manner that would be considered incorrect by your programming instructor. Find those errors and correct them.

	x	$-$	$*$	\$	E
0	$s6$				1
1		$s2$	$s4$	*halt*	
2	$s6$				3
3		$r1$	$r1$	$r1$	
4	$s6$				5
5		$s2$	$s4$	$r2$	
6		$r3$	$r3$	$r3$	

   In the "$-$" column, the entry in row 5 should be r2 instead of s2. In the "$*$" column, the entry in row 3 should be s4 instead of r1, and the entry in row 5 should be r2 instead of s4.

6. Every language, or problem, falls into exactly one of these categories. For each of the languages, write a letter indicating the correct category.

   A Known to be $\mathcal{NC}$.
   B Known to be $\mathcal{P}$–TIME, but not known to be $\mathcal{NC}$.
   C Known to be $\mathcal{NP}$, but not known to be $\mathcal{P}$–TIME and not known to be $\mathcal{NP}$–complete.
   D Known to be $\mathcal{NP}$–complete.
   E Known to be $\mathcal{P}$–SPACE but not known to be $\mathcal{NP}$
   F Known to be EXP–TIME but not nown to be $\mathcal{P}$–SPACE.
   G Known to be EXP–SPACE but not nown to be EXP–TIME.
   H Known to be decidable, but not nown to be EXP–SPACE.
   I $\mathcal{RE}$ but not decidable.
   K co-$\mathcal{RE}$ but not decidable.
   L Neither $\mathcal{RE}$ nor co-$\mathcal{RE}$.

   (i) **I** All C++ programs which halt with no input.

   (ii) **A** All base 10 numerals for perfect squares.

(iii) **E** All configurations of RUSH HOUR from which it's possible to win.

(iv) **D** All satisfiable Boolean expressions.

(v) **B** All binary numerals for composite integers. (Composite means not prime.)

(vi) **E** The furniture mover's problem.

(vii) **G** The set of all positions of Chinese GO, on a board of any size, from which white can win.

(viii) **K** All C++ programs which do not halt if given themselves as input.

(ix) **A** The Dyck language.

(x) **D** The Jigsaw problem. (That is, given a finite set of two-dimensional pieces, can they be assembled into a rectangle, with no overlap and no spaces.)

(xi) **B** Factorization of binary numerals.

(xii) **D** SAT.

(xiii) **D** 3-SAT.

(xiv) **A** 2-SAT.

(xv) **D** The Independent Set problem.

(xvi) **D** The Subset Sum Problem.

(xvii) **D** The block sorting problem.

(xviii) **E** The sliding block problem.

(xix) **D** The Hamiltonian cyle problem.

(xx) **D** The traveling salesman problem.

(xxi) **C** The graph isomorphism problem.

(xxii) **D** The 3-coloring problem.

(xxiii) **A** The 2-coloring problem.

(xxiv) **K** The set of binary numerals for Busy Beaver numbers.

7. Prove that every context-sensitive language is decidable. The way to do this is to start with an arbitrary non-contracting grammar, and then design a program which decides whether any given string is generated by that grammar. (If the string has length n, the running time of your program could be very long, maybe an exponentially bounded function of n.) You do not have to actually write the program, not even pseudocode; just explain how you would do it.

   Let L be a context-senstive language.

   L is generated by some non-contracting grammar G, with start symbol S, variables V and terminals Σ. For each positive integer n Let $\mathcal{G}_n = (\mathcal{V}_n, \mathcal{A}_n)$ be the directed graph where $\mathcal{V}_n$ is the set of strings over $(V + \Sigma)$ of length at most n, $\mathcal{A}_n$ is the set of all pairs (u, v) such that $u, v \in \mathcal{V}_n$ and $u \Rightarrow v$. If $w \in L$ has length n, there is a derivation of w which is a directed path from S to w. Since G is non-contracting, that path lies entirely within $\mathcal{G}_n$. Thus w of length n is in L if and only if it is reachable from S. The reachability problem in a directed graph is decidable by DFS, hence L is decidable.

8. Prove that every decidable language is enumerated in canonical order by some machine.

   Let $L \subseteq \Sigma^*$ be decidable, where Σ is an alphabet. Let $w_1, w_2, \ldots$ be the canonical order enumeration of Σ^*. The following program enumerates L in canonical order.
   For all i from 1 to ∞
   If $w_i \in L$ write w_i

9. Prove that every language that is enumerated in canonical order by some machine is decided by some other machine. Let L be a language which is enumerated in canonical order by some machine. Let $w_1, w_2, \ldots$ be that canonical order enumeration. The following program decides L.
   Read w
   For all i from 1 to ∞
   If($w = w_i$)
   Halt and accept.
   If ($w < w_i$) in canonical order
   Halt and reject.

10. Prove that eny language accepted by any machine can be enumerated by some other machine. Let M be a machine which accepts a language L over an alphabet Σ, and let $w_1, w_2 \ldots$ be the canonical enumerattion of Σ^*. The following program enumerates L.
    For all t from 1 to ∞
    For all i from 1 to t
    If M accepts w_i within t steps
    Write w_i

    If $w \in L$, then $w = w_i$ for some i. Let T be the number of steps it takes for M to accept w. Then w is written by the program during the t^{th} iteration of the outer loop when $t \geq \max(T, i)$. On the hand, if $w \notin L$, it will never be accepted by M and hence never written.

11. Prove that any language which is enumerated by some machine is accepted by some other machine.

    Let $w_1, w_2 \ldots$ be a recursive enumeration of a language L. The following program accepts L.
    Read w
    For all i from 1 to ∞
    If($w = w_i$)
    Accept w

12. Prove that the halting problem is undecidable.

    Proof: By contradiction. Assume the halting problem is decidable. That is, there is a machine H such that, for any machine M and any string w $H(\langle M \rangle, w) = 1$ if M halts with input w, and $H(\langle M \rangle, w) = 0$ otherwise. Let Q be the following program.
    Read a machine description $\langle M \rangle$
    If $H(\langle M \rangle, \langle M \rangle)$ enter an infinite loop
    Else halt

We now investigate whether Q will halt with input $\langle Q \rangle$. That is, whether $H(\langle Q \rangle, \langle Q \rangle) = 1$. But in that case, according to its code, Q will enter an infinite loop, contradicting the fact that $H(\langle Q \rangle, \langle Q \rangle) = 1$. On the other hand, if Q does not halt with input $\langle Q \rangle$, we have $H(\langle Q \rangle, \langle Q \rangle) = 0$, which following its code, causes Q to enter an infinite loop with input $\langle Q \rangle$, contradiction. We conclude that the halting problem is undecidable. ∎

13. Prove that the grammar given in Problem 5 is ambiguous by giving two different leftmost derivations for some string. (If you simply give two different parse trees, you have not answered the question.)

    The start symbol of that grammar is E. The give two leftmost derivations for $x - x * x$.

    $E \Rightarrow E - E \Rightarrow x - E \Rightarrow x - E * E \Rightarrow x - x * E \Rightarrow x - x * x$

    $E \Rightarrow E * E \Rightarrow E - E * E \Rightarrow x - E * E \Rightarrow x - x * E \Rightarrow x - x * x$

14. Use the pumping lemma to prove that the Dyck language is not regular.

    Proof: By contradiction. Suppose the Dyck language, L, is regular. Then the pumping lemma holds for L. Pick a pumping length p. Let $w = a^p b^p$ which is a member of L of length at least p. (We use a,b for left and right parentheses.) Using the pumping lemma, we pick strings x, y, z such that
    1. $w = xyz$
    2. $|xy| \leq p$
    3. $|y| > 0$
    4. For any $i \geq 0$, $xy^i z \in L$. Let $i = 0$. By 2., xy lies entirely within a^p, and thus $y = p^j$ for $j > 0$. Let $u = xy^0 z \in L$. By the pumping lemma, $u \in L$. But $u = a^{p-j} b^p$, and thus $\#_a(u) < \#_b(u)$, contradiction. Thus, w is not regular. ∎

15. Give a polynomial time reduction of 3-SAT to to the independent set problem.

    Let E be a Boolean expression in 3-CNF form. Let k be the number of clauses of E. We construct $R(E) = G$ to be a graph which has an independent set of order k if and only if E is satisfiable.

    Let $E = C_1 * C_2 * \cdots * C_k$ where $C_i = (t_{i,1} + t_{i,2} + t_{i,3}$, where each $t_{i,j}$ is a literal, meaning either a variable or the negation of a variable. Let $G = (V, E)$ where $V = \{t_{i,j}\}$ for $i = 1, 2, 3$ and $j = 1, 2, \ldots k$ and E is the set of all pairs $V_{i,j}, v_{i',j'}$ such that either $i = i'$ or $t_{i,j}$ is the negation of $t_{i',j'}$. Then E is satisfiable if and only if G has an inpdendent set of k vertices.

16. Give a polynomial time reduction of the subset sum problem to the partition problem.

    An instance of the subset sum problem is an ordered pair (X, K) where X is a lst of numbers $x_1, x_2, \ldots x_n$ and K is a number, which has a *solution* if there exists a sublist of X whose sum is K. An instance of the partition problem is a list of numbers $Y = y_1, y_2, \ldots y_m$ which has a sublist whose sum equals half the sum of Y. Given an instance (X, K) of the subset sum problem we construct an instance $R(X, K)$ of the partion problem which has a solution if and only if (X, K) has a solution.

17. I have repeatedly stated in class that no language that has parentheses can be regular. For that to be true, there must be parenthetical strings of arbitrary nesting depth. (If you don't know what nesting depth is, look it up.) Some programming languages have limitations on nesting depth. For example, I have read that ABAP has maximum nesting depth of 256. (Who would ever want to go that far?)

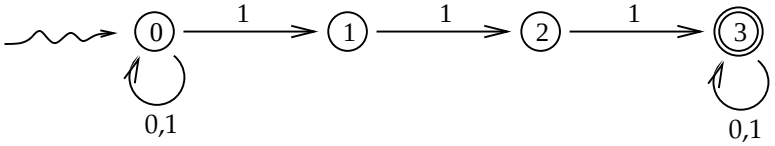
The Dyck language is generated by the following context-free grammar. (As usual, to make grading easier, I use a and b for left and right parentheses.)

1. $S \to aSbS$
2. $S \to \lambda$

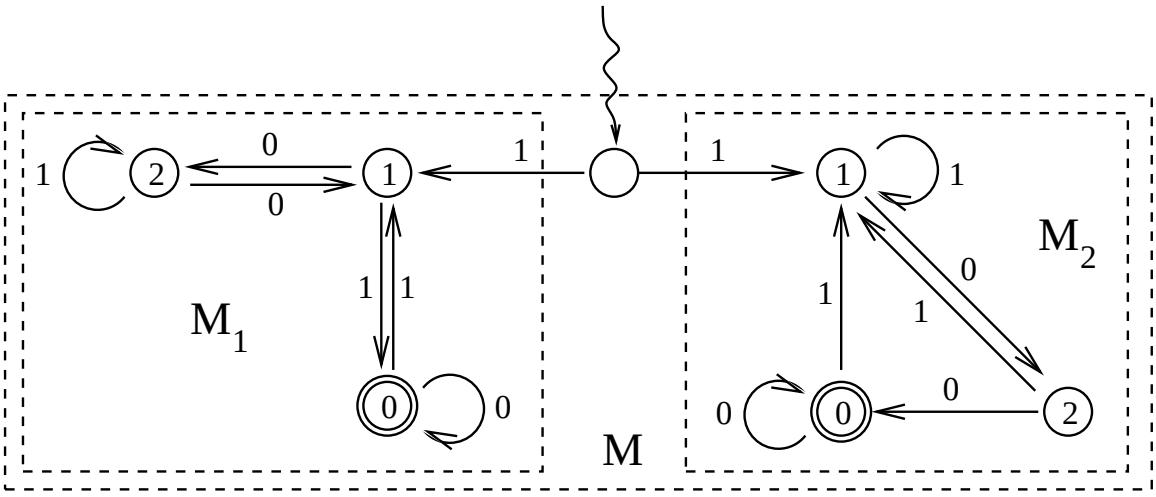
Prove that the subset of the Dyck language consisting of strings whose nesting depth is no more than 50 is regular.

Proof: That language cannot have any string whose length is longer than 100. Thus, the language is finite, hence regular. ∎

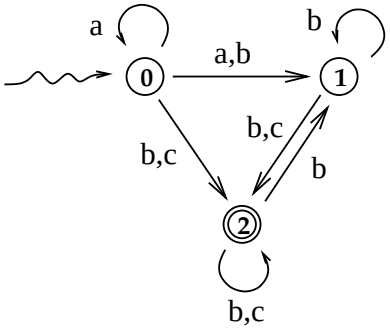
18. Find an NFA with at most 4 states which accepts the language of binary strings which contain the substring 111.

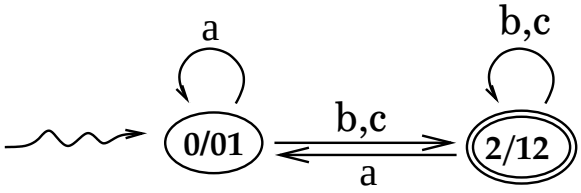


19. Let L be the language of all binary numerals for positive multiples of either 3 or 4, where leading zeros are not permitted. That is, $L = \{11, 100, 110, 1000, 1001, 1100, \ldots\}$. Find an NFA with 8 states which accepts L. (There is also a DFA with 12 states which accepts L.)

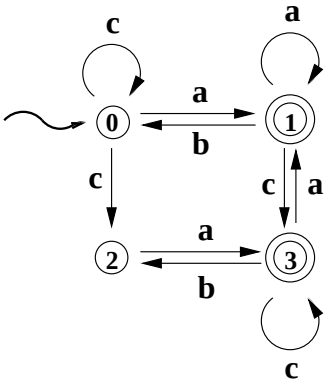


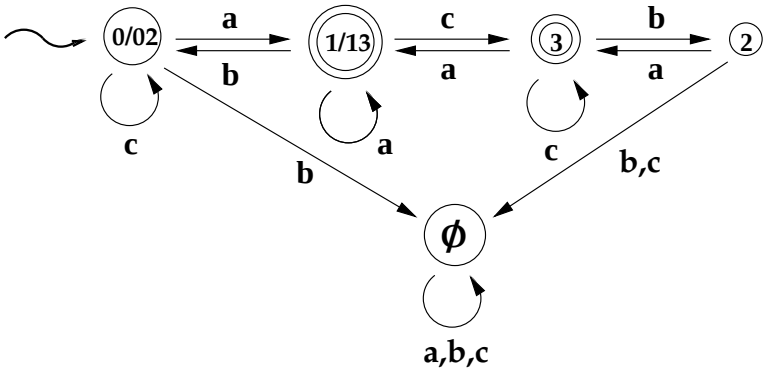
20. Construct a minimal DFA equivalent to the NFA shown below.

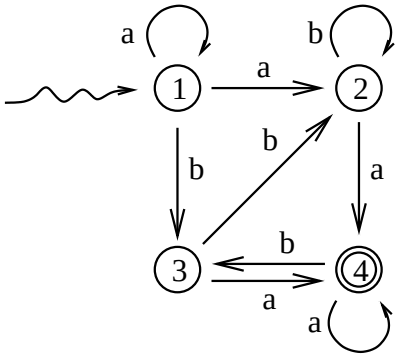


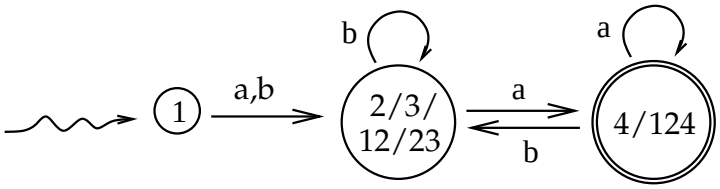


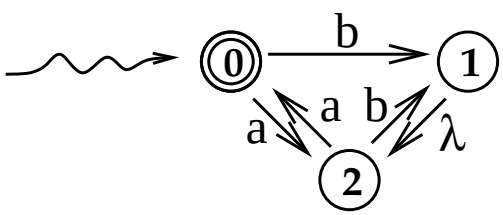
Find the minimal DFA equivalent to each of the following NFA.

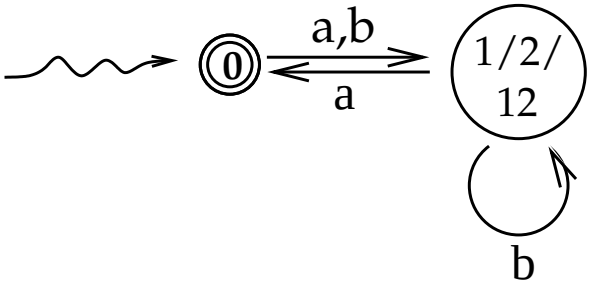




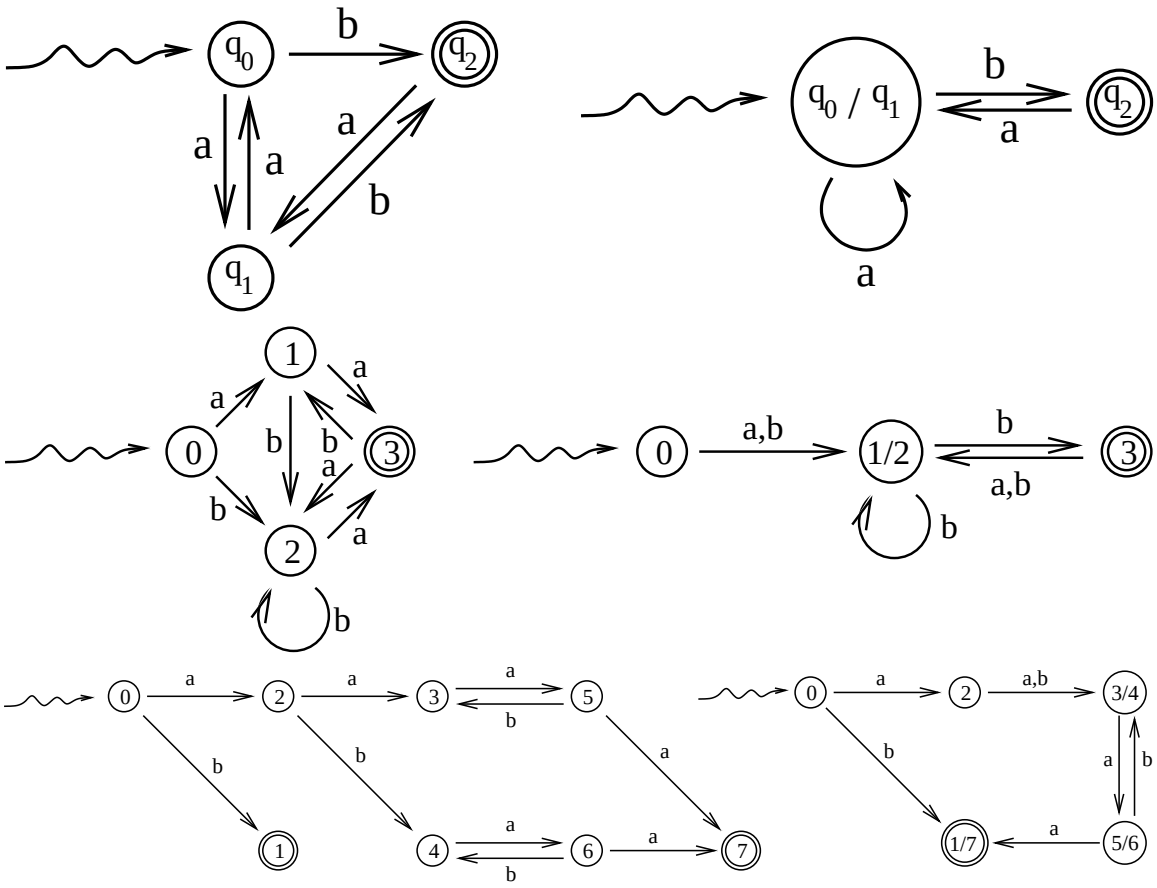








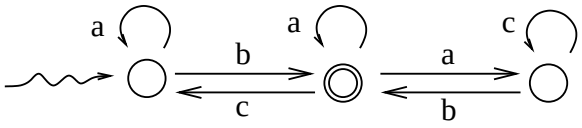
21. Minimize each DFA:



22. Find an NFA which accepts the language generated by this grammar.

$S \to aA|cS|cC$
$A \to aA|bS|cB|\lambda$
$B \to aA|cB|bC|\lambda$
$C \to aB$

23. Give a regular expression which describes the language accepted by this NFA.



$a^*b(a + ca^*b) + ac^*b)^*$

24. Find a DFA equivalent to the NFA shown in Figure 1.

25. Give a regular grammar for the language accepted by the machine in Figure 1.

26. Find a minimal DFA equivalent to the NFA shown in Figure 1.

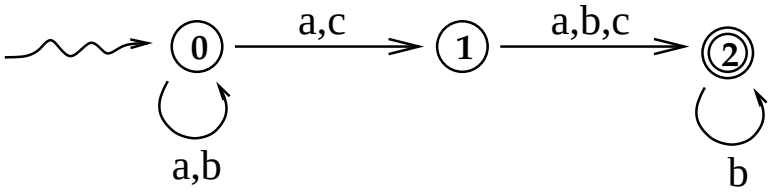


Figure 1: NFA for problems 26 and 27

27. Give a regular grammar with **no more than three variables** for the language accepted by the machine in Figure 1.

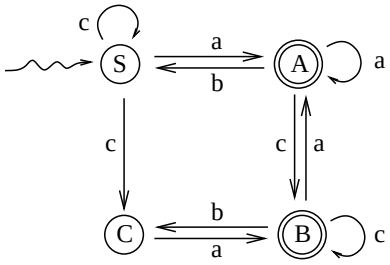
$S \to aS|bS|aA|cA$
$A \to aB|bB|cB$
$B \to bB|\lambda$

28. Give a regular expression for the language accepted by the machine in Figure 1

$(a+b)^*(a+c)(a+b+c)b^*$

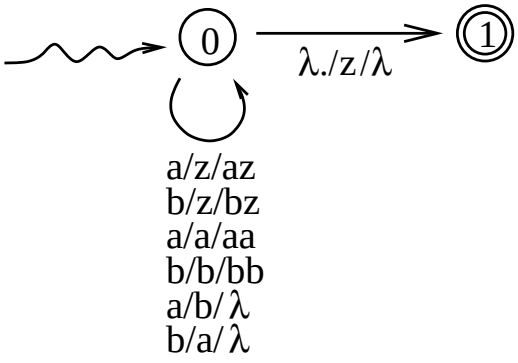
29. Illustrate an NFA which accepts the language generated by this grammar.

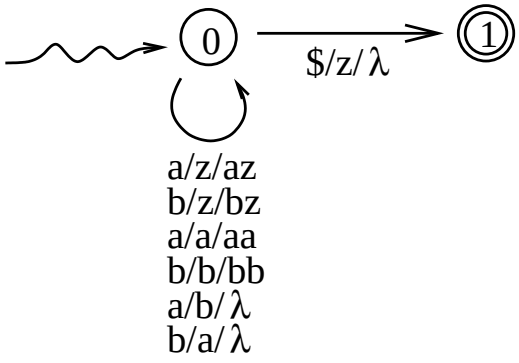
$S \to aA|cS|cC$
$A \to aA|bS|cB|\lambda$
$B \to aA|cB|bC|\lambda$
$C \to aB$



30. Let $L = \{w \in \{a,b\}^* : \#_a(w) = \#_b(w)\}$, which is generated by the following context-free grammar.
    1. $S \to aSbS$
    2. $S \to bSaS$
    3. $S \to \lambda$

    Draw a DPDA which accepts L. (Recall that the input to that DPDA must be of the form $w\$$, where $w \in L$ and \$ is the end-of-file symbol.)





The first figure shows a PDA which is not deterministic. The second figure shows a DPDA.

31. We know that context-free languages are exactly those which are accepted by push-down automata. We now define a new class of machines, which we call "limited push-down automata." An LPDA is exactly the same as a PDA, but with the restriction that the stack is never allowed to be larger than some given constant. What is the class of languages accepted by limited push-down automata? Think!

    Regular langages.

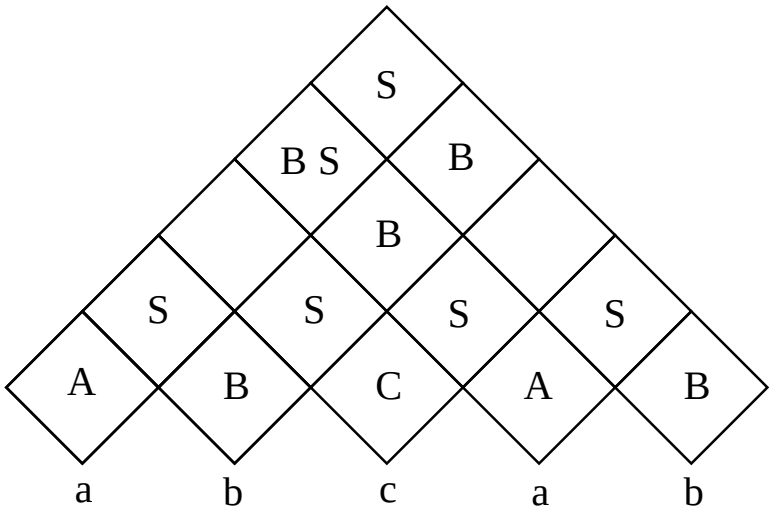
32. What is the class of languages decided by 2-PDA? A 2-PDA is the same as a PDA, except that it has two statcks instead of just one.

    Recursively enumerable languages.

33. Use the CYK algorithm to decide whether *abcab* is generated by the CNF grammar. The start symbol is S.

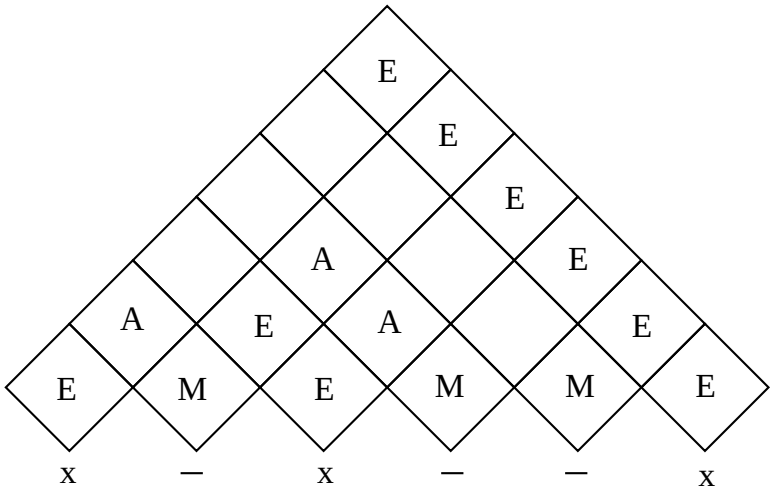
    $S \to AB \,|\, BC \,|CA$
    $A \to a$
    $B \to SA \,|\, SS \,|\, b$
    $C \to c$

    by filling in the matrix.



34. Use the CYK algorithm to decide whether $x - x - -x$ is generated by the CNF grammar below, by filling in the matrix. The start symbol is E.

    $E \to ME$
    $A \to EM$
    $E \to AE$
    $M \to -$
    $E \to x$



35. What complexity class contains all sliding block problems?

    $\mathcal{P}$–SPACE

36. Label each of the following sets as countable or uncountable. Write **C** or **U**.

    (i) **C** The set of integers.

(ii) **C** The set of rational numbers.

(iii) **U** The set of real numbers.

(iv) **U** The set of binary languages.

(v) **C** The set of co-$\mathcal{RE}$ binary languages.

(vi) **U** The set of undecidable binary languages.

(vii) **U** The set of functions from integers to integers.

(viii) **C** The set of recursive real numbers.

(ix) **C** The set of $\mathcal{P}$–SPACE languages over the binary alphabet.

(x) **U** The set of functions from the integers to the binary alphabet $\{0, 1\}$.

(xi) **C** The set of functions from the binary alphabet $\{0, 1\}$ to the integers.

37. Which class of languages does each of these machine classes accept?

(i) Deterministic finite automata. **Regular languages**

(ii) Non-deterministic finite automata. **Regular languages**

(iii) Push-down automata. **Context-free languages**

(iv) Turing Machines. **Recursively enumerable languages**

38. The grammar below is an ambiguous CF grammar with start symbol E, and is parsed by the LALR parser whose ACTION and GOTO tables are shown here. The ACTION table is missing actions for the second column, when the next input symbol is the "minus" sign. Fill it in. Remember the C++ precedence of operators. (Hint: the column has seven different actions: s2, s4, r1, r2, r3, r4, and r5, some more than once, and has no blank spaces.)

1. $E \to E -_2 E_3$
2. $E \to -_4 E_5$
3. $E \to E *_6 E_7$
4. $E \to (_8 E_9)_{10}$
5. $E \to x_{11}$

	x	$-$	$*$	(	)	\$	S
0	$s11$	$s4$		$s8$			1
1		$s2$	$s6$	$s2$		halt	
2	$s11$	$s4$		$s8$			3
3		$r1$	$s6$	$r1$	$r1$	$r1$	
4	$s11$	$s4$		$s8$			5
5		$r2$	$r2$		$r2$	$r2$	
6	$s11$	$s4$		$s8$			7
7		$r3$	$r3$		$r3$	$r3$	
8	$s11$	$s2$		$s8$			9
9		$s4$	$s6$		$s6$		
10		$r4$	$r4$	$r4$	$r4$	$r4$	
11		$r5$	$r5$		$r5$	$r5$	

40. Consider the CF grammar below. The ACTION and GOTO tables are given below, except that six actions are mising, indicated by question marks. Fill in the missing actions (below the question marks). The actions of your table must be consistent with the precedence of operators in C++.

1. $E \to E -_2 E_3$
2. $E \to -_4 E_5$
3. $E \to E *_6 E_7$
4. $E \to x_8$

	x	$-$	$*$	\$	E
0	$s8$	$s4$			1
1		$s2$	$s6$	HALT	
2	$s8$	$s4$			3
3		? $r1$	? $s1$	$r1$	
4	$s8$	$s4$			5
5		? $r2$	? $r2$	$r2$	
6	$s8$	$s4$			7
7		? $r3$	? $r3$	$r3$	
8	$s8$	$r4$	$r4$	$r4$	

41. When there is an "else" after two "if"s, which "if" does the "else" pair with? Here is CF grammar, G_3, which isolates this problem. The start symbol S is the only variable. The symbol i represents "if(condition)", e represents "else," w represents "while(condition)" and a represents any other statement, such as an assignment statement.

1. $S \to i_2 S_3$
2. $S \to i_2 S_3 e_4 S_5$
3. $S \to w_6 S_7$
4. $S \to a_8$

	ACTION					GOTO
	a	i	e	w	\$	S
0	s8	s2		s6		1
1					HALT	
2	s8	s2		s6		3
3			s4		r1	
4	s8	s2		s6		5
5			r2		r2	
6	s8	s2		s6		7
7			r3		r3	
8			r4		r4	

8. Which entry, or entries, solve the dangling else problem?

Row 3, column "e"

9. Walk through the actions of the LALR parser for the input string *iwiaewa*.

stack	input	output	action
$\$_0$	$iwiaewa\$$		
$\$_0 i_2$	$wiaewa\$$		$s2$
$\$_0 i_2 w_6$	$iaewa\$$		$s2$
$\$_0 i_2 w_6 i_2$	$aewa\$$		$s2$
$\$_0 i_2 w_6 i_2 a_8$	$ewa\$$		$s2$
$\$_0 i_2 w_6 i_2 S_3$	$ewa\$$	4	$r4$
$\$_0 i_2 w_6 i_2 S_3 e_4$	$wa\$$	4	$s4$
$\$_0 i_2 w_6 i_2 S_3 e_4 w_6$	$a\$$	4	$s6$
$\$_0 i_2 w_6 i_2 S_3 e_4 w_6 a_8$	$\$$	4	$s8$
$\$_0 i_2 w_6 i_2 S_3 e_4 w_6 S_7$	$\$$	44	$r4$
$\$_0 i_2 w_6 i_2 S_3 e_4 S_5$	$\$$	443	$r3$
$\$_0 i_2 w_6 S_7$	$\$$	4432	$r2$
$\$_0 i_2 S_3$	$\$$	44343	$r3$
$\$_0 S_1$	$\$$	443231	$r1$
halt			

42. (i) Give a context-sensitive grammar for $\{a^n b^n c^n : n > 0\}$

    1. $S \to abc$
    2. $S \to aSBc$
    3. $cB \to Bc$
    4. $bB \to bb$

    (ii) Using that grammar, give a derivation of the string *aaabbbccc*.

    $S \Rightarrow aSbc$
    $\Rightarrow aaSBcBc$
    $\to aaabcBcBc$
    $\to aaabBccBc$
    $\to aaabBcBcc$
    $\to aaabBBccc$
    $\to aaabbBccc$
    $\to aaabbbccc$

43. Work Exercise 1 on the handout reglrNC.pdf.