CS 320
Computer Language Processing
Midterm

April 4, 2025

Name:	Ada Lovelace	Room:	INM 200
Sciper:	111111	Seat:	314

1. The exam starts at 13:15 and ends at 15:45. You have **150 minutes** to complete the exam.
2. Place your CAMIPRO card on your desk.
3. Put all electronic devices in a bag away from the bench.
4. Write your final answers using a permanent pen (no pencils, no erasable pens).
5. This exam is 18 pages long, including this cover page. Check that you have all the pages.
6. The exercises are not ordered by how difficult they may be. If you are stuck on an exercise, you can skip it and come back to it later.
7. Answer all questions in the provided space. Do not submit additional sheets. Do not unstaple the given sheets. Material in the scratch area will not be graded.
8. Any multiple-choice questions have a **single correct answer**. Clearly circle the letter corresponding to your choice on the page itself.
9. The maximum number of points on the exam is **30**.

Question:	1	2	3	4	5	Total
Points:	5	5	5	10	5	30
Score:						

Page intentionally left blank.

Question 1. (5 points)

Consider the following language described in set-builder notation:

$$L = \{w \in \{a, b\}^* \mid \#a(w) = \#b(w)\}$$

i.e., the set of all strings over the alphabet $\{a, b\}$ with an equal number of a's and b's.

(i) $\boxed{\text{1.5 points}}$ Produce a context-free grammar G that generates the language L, i.e. $L(G) = L$.

(ii) $\boxed{\text{1.5 points}}$ If your grammar is ambiguous, show that it is by producing a word demonstrating the ambiguity with two distinct derivations or parse trees. If your grammar is unambiguous, argue why.

(iii) $\boxed{\text{2 points}}$ Prove that the language L cannot be represented by a non-deterministic finite automaton.

Question 2. (5 points)

Consider the grammar G, with start symbol R, representing the set of palindromes over the alphabet $\Sigma = \{a, b\}$:

$$R ::= aRa \mid bRb \mid a \mid b \mid \epsilon$$

From this, we construct the reflexive-transitive closure, the grammar G^* with start symbol S, representing $L(G)^*$, by adding the single rule

$$S ::= \epsilon \mid SR$$

(i) $\boxed{\text{1 points}}$ Express the grammar G^* as an inductive relation defined by its production rules.

(ii) $\boxed{\text{4 points}}$ Inductively prove that the relation G^* you defined contains *every* word $w \in \Sigma^*$, i.e., that G^* is universal.

Question 3. (5 points)

Consider a simple language with variables (alphanumeric strings starting with a letter), assignments (`=`), equality (`==`), conditionals (`if then else`), single-line comments (`// ...`), and block comments (`/* ... */`).

We wish to write a lexer for this language, by defining each of the following tokens (in the given priority order):

`IF, THEN, ELSE, EQ, ASSIGN, LPAREN, RPAREN, ID, INT, LINESKIP, BLOCKSKIP, WS`

The following example strings must be tokenized as given below. `\n` is a single character, representing a newline.

String	Token Stream
`x=1`	`ID ASSIGN INT`
`if then else`	`IF WS THEN WS ELSE`
`(x == 1)`	`LPAREN ID WS EQ WS INT RPAREN`
`===`	`EQ ASSIGN`
`if // comment if`	`IF WS LINESKIP`
`// comment \n if`	`LINESKIP WS IF`
`x ==/* comment */5`	`ID WS EQ BLOCKSKIP INT`
`/* comment // other */ 5`	`BLOCKSKIP WS INT`
`/* comment /* other */ */`	*Lexing error*

The lexer must enforce that block comments cannot be nested, i.e., a block comment start `/*` must not appear inside another block comment.

The lexing priority follows longest match rule and the priority of tokens as given. You may assume Σ is the total alphabet, A is the set of English letters (a-z, A-Z), and D is the set of digits (0-9).

For each of the tokens below, fill in the blank, choosing **one** regular expression or non-deterministic finite automaton (NFA) (**next page**) that represents the words to be matched by the token. `WS` (accepting spaces, tabs, and newlines) is defined.

(i) 0.4 points `IF:` ______________

(ii) 0.4 points `THEN:` ______________

(iii) 0.4 points `ELSE:` ______________

(iv) 0.4 points `EQ:` ______________

(v) 0.4 points `ASSIGN:` ______________

(vi) 0.4 points `LPAREN:` ______________

(vii) 0.4 points `RPAREN:` ______________

(viii) 0.6 points `ID:` ______________

(ix) 0.4 points `INT:` ______________

(x) 0.6 points `LINESKIP:` ______________

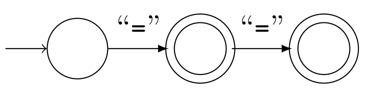
(xi) 0.6 points `BLOCKSKIP:` ______________

(xii) `WS:` (“\s” | “\t” | “\n”)$^+$

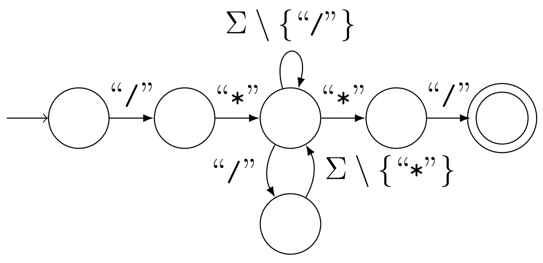
We use the following notation for regular expressions and sets, where e_1 and e_2 are regular expressions, and a, b, and c are characters in the alphabet:

1. $e_1 e_2$ is the concatenation of e_1 and e_2.
2. $e_1 \mid e_2$ is the union or disjunction of e_1 and e_2.
3. e^* represents zero or more repetitions of e. e^+ is one or more repetitions of e.
4. Characters in the alphabet are written with quotes for clarity, e.g. "a". A string of characters, e.g. "abc" represents the concatenation of characters, i.e., "a" "b" "c".
5. A finite set of characters represents the disjunction of its elements, e.g. $\{a, b, c\} =$ "a" | "b" | "c".
6. The binary operations union ($\cup$), intersection ($\cap$), and set difference ($\setminus$) are interpreted as usual on sets of characters.

A : "`if`" **B** : "`then`" **C** : "`===`" **D** :

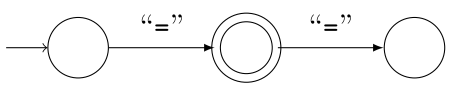


E :

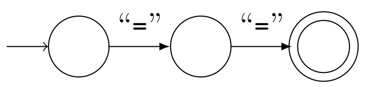


F : "`/*`" $(\Sigma \setminus \{$"`/`"$\})^*$ "`*/`"

G : "`//`" $(\Sigma \setminus \{$"`\n`"$\})^*$ **H** :

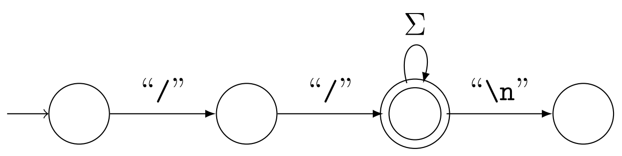


I :

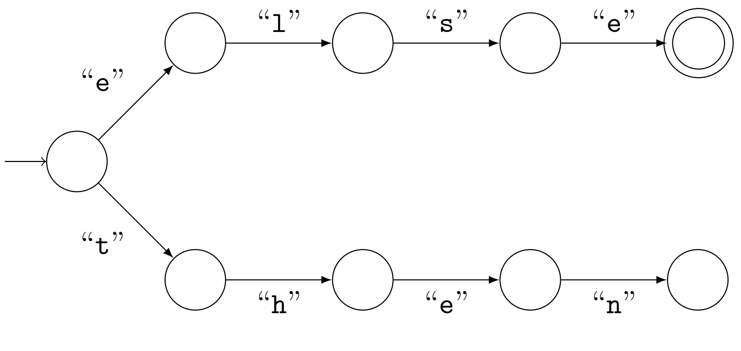


J : $(A \cup D)^*$ **K** : D^+ **L** : ")" **M** : "("

N : $A(A \cup D)^*$ **O** :



P :



Question 4. (10 points)

Consider the following grammar that accepts well-formed propositional formulas over the alphabet $A = \{\wedge, \vee, \neg,), (, atom\}$

$$S ::= F \ \textbf{EOF}$$
$$F ::= F \wedge F \mid F \vee F \mid \neg F \mid (F) \mid atom$$

(i) $\boxed{\text{1 points}}$ Is the given grammar LL(1)? Justify your answer.

(ii) $\boxed{\text{2 points}}$ We use the term *literal* to refer to either an atomic predicate *atom*, or its negation $\neg atom$. A *clause* is a single literal l or a disjunction of literals $(l_1 \vee l_2 \vee \ldots \vee l_m)$. A formula is said to be in *conjunctive normal form* (CNF) if it is a conjunction of clauses, i.e. $C_1 \wedge C_2 \wedge \ldots \wedge C_n$, where each C_i is a clause.

Modify the grammar so it accepts only formulas that are in *conjunctive normal form* (CNF).

For example, the following formulas are in CNF, where a and b are atomic predicates:

$$a, \quad a \wedge b, \quad (a \vee b), \quad \neg a \wedge \neg b, \quad (a \vee \neg c) \wedge b, \quad (\neg a \vee \neg b \vee c) \wedge (a \vee \neg b)$$

(iii) $\boxed{\text{1 points}}$ Convert your grammar to be LL(1), if it is not already LL(1).

(iv) $\boxed{\text{4 points}}$ Draw the LL(1) parsing table for your grammar. Compute and show the nullable, first, and follow sets for non-terminals in your grammar as intermediate steps.

(v) $\boxed{\text{2 points}}$ Use the LL(1) parsing table to parse, or attempt to parse till error, the following string, showing intermediate state of the stack, and the chosen production at each step:

$$(a \vee \neg c) \wedge b$$

Use the provided space on the ***next three pages*** *for your answer.*

Question 5. (5 points)

Consider the following type system for a programming language. The language contains integers, Booleans, functions, and pairs.

Pairs $(\cdot, \cdot)$ and functions $\cdot \Rightarrow \cdot$ are distinct binary type constructors.

$$\frac{n \text{ is an integer value}}{\Gamma \vdash n : \texttt{Int}}\,\text{Int}$$

$$\frac{\Gamma \vdash e_1 : \texttt{Int} \qquad \Gamma \vdash e_2 : \texttt{Int}}{\Gamma \vdash e_1 + e_2 : \texttt{Int}}\,\text{Add}$$

$$\frac{b \text{ is a Boolean value}}{\Gamma \vdash b : \texttt{Bool}}\,\text{Bool}$$

$$\frac{\Gamma \vdash e_1 : \texttt{Bool} \qquad \Gamma \vdash e_2 : \tau \qquad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : \tau}\,\text{If}$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 == e_2 : \texttt{Bool}}\,\text{Eq} \qquad \frac{\Gamma \vdash e_1 : \texttt{Int} \qquad \Gamma \vdash e_2 : \texttt{Int}}{\Gamma \vdash e_1 < e_2 : \texttt{Bool}}\,\text{Lt}$$

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash (x : \tau_1) \Rightarrow e : \tau_1 \Rightarrow \tau_2}\,\text{Fun} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \Rightarrow \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2}\,\text{App}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : (\tau_1, \tau_2)}\,\text{Pair}$$

$$\frac{\Gamma \vdash e : (\tau_1, \tau_2)}{\Gamma \vdash \texttt{fst}(e) : \tau_1}\,\text{Fst} \qquad \frac{\Gamma \vdash e : (\tau_1, \tau_2)}{\Gamma \vdash \texttt{snd}(e) : \tau_2}\,\text{Snd}$$

(i) $\boxed{\text{2 points}}$ Consider the term t below, with type variables $\tau_z, \tau_x, \tau_y, \tau_l, \tau_r, \tau_{fx}$, and τ_{sy} ascribing subterms of t as shown:

$$(\texttt{z} : \tau_z) \Rightarrow (((\texttt{x} : \tau_x) \Rightarrow \texttt{fst}(\texttt{x}) : \tau_{fx}, \texttt{z}) : \tau_l) == ((\texttt{z}, (\texttt{y} : \tau_y) \Rightarrow \texttt{snd}(\texttt{y}) : \tau_{sy}) : \tau_r)$$

Which of the following statements are true about assignments to the type variables such that t is well-typed?

i. $\boxed{\text{0.5 points}}$ In every valid assignment, $\tau_l = \tau_r$:
A. True B. False

ii. $\boxed{\text{0.5 points}}$ In every valid assignment, $\tau_l = \texttt{Bool}$:
A. True B. False

iii. $\boxed{\text{0.5 points}}$ There is a valid assignment where $\tau_z = \texttt{Int}$:
A. True B. False

iv. $\boxed{\text{0.5 points}}$ There is a valid assignment where $\tau_x = (\texttt{Int}, \texttt{Bool})$:
A. True B. False

(ii) $\boxed{\text{1 points}}$ Which of the following types τ given below apply to the term t above, i.e. there is a derivation of $\vdash t : \tau$?

A. $(\texttt{Bool} \Rightarrow \texttt{Bool}) \Rightarrow \texttt{Int} \Rightarrow \texttt{Bool}$

B. $((\texttt{Int}, \texttt{Int}) \Rightarrow \texttt{Int}) \Rightarrow \texttt{Bool}$

C. $(\texttt{Bool} \Rightarrow (\texttt{Bool}, \texttt{Bool})) \Rightarrow \texttt{Bool}$

D. $(\texttt{Bool}, \texttt{Int}) \Rightarrow \texttt{Bool}$

(iii) $\boxed{\text{2 points}}$ Consider the complete labelling of types of subterms of the term t below:

$$t : \tau$$
$$z : \tau_z$$
$$(\texttt{x} \Rightarrow \texttt{fst}(\texttt{x}), \texttt{z}) == (\texttt{z}, \texttt{y} \Rightarrow \texttt{snd}(\texttt{y})) : \tau_{eq}$$
$$(\texttt{x} \Rightarrow \texttt{fst}(\texttt{x}), \texttt{z}) : \tau_l$$
$$\texttt{x} \Rightarrow \texttt{fst}(\texttt{x}) : \tau_{funx}$$
$$\texttt{x} : \tau_x$$
$$\texttt{fst}(\texttt{x}) : \tau_{fx}$$
$$(\texttt{z}, \texttt{y} \Rightarrow \texttt{snd}(\texttt{y})) : \tau_r$$
$$\texttt{y} \Rightarrow \texttt{snd}(\texttt{y}) : \tau_{funy}$$
$$\texttt{y} : \tau_y$$
$$\texttt{snd}(\texttt{y}) : \tau_{sy}$$

Given this labelling,

1. Compute the set of initial type checking constraints for the term t.
2. Compute a general solution to the set of constraints to infer a general type for t, using type unification.

Use the provided space on the **next three pages** for your answer.

Scratch area

End