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) [1.5 points] Produce a context-free grammar G that generates the language L, i.e. $L(G) = L$.

(ii) [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.

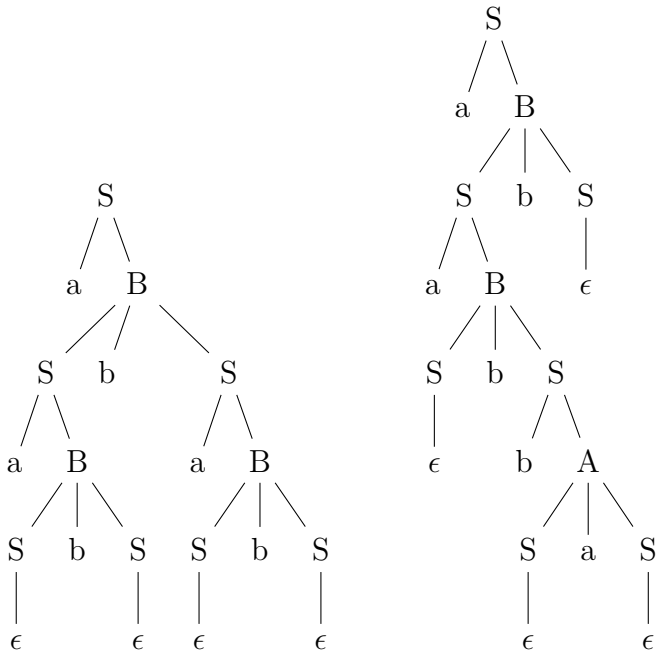
(iii) [2 points] Prove that the language L cannot be represented by a non-deterministic finite automaton.

Solution: [1.5 points] One possible grammar G is:

$$S ::= a\ B \mid b\ A \mid \epsilon$$
$$A ::= SaS$$
$$B ::= SbS$$

where S is the set of "balanced" strings, A is the set of strings with an extra a and B is the set of strings with an extra b.

[1.5 points] The grammar G above is ambiguous. Consider the string $aabbab$. It can be accepted by G with the two following distinct parse trees:



There is an LL(1), and thus deterministic and unambiguous, grammar for the language L as well:

$$S ::= a\ B\ S \mid b\ A\ S \mid \epsilon$$
$$A ::= a \mid b\ A\ A$$
$$B ::= b \mid a\ B\ B$$

where S is the set of "balanced" strings, A is the set of strings with an extra a and B is the set of strings with an extra b.

2 points The language L is *not* regular. We prove this using the pumping lemma. Assume that L is regular. Then, by the pumping lemma, there exists a pumping length p such that any string $w \in L$ with $|w| \geq p$ can be decomposed into three parts $w = xyz$ such that:

- $|xy| \leq p$,
- $|y| > 0$, and
- $xy^iz \in L$ for all $i \geq 0$.

Let $w = a^pb^p$. Then, we can decompose w into $x = a^k$, $y = a^m$ and $z = a^{p-k-m}b^p$ where $k + m \leq p$ and $m > 0$. Then, $xy^0z = a^ka^{p-k-m}b^p = a^{p-m}b^p$, must be in L. However, this string has less a's than b's, i.e. $\#a(xy^0z) \neq \#b(xy^0z)$. This is a contradiction, and the word w cannot be pumped. Thus, the language L cannot be regular, and thus 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) [1 points] Express the grammar G^* as an inductive relation defined by its production rules.

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

Solution: [1 points] G^* as an inductive relation:

$$\frac{}{\epsilon \in S}\, S_\epsilon \qquad \frac{w_1 \in S \qquad w_2 \in R}{w_1 w_2 \in S}\, S_R$$

$$\frac{}{a \in R}\, R_a \qquad \frac{}{b \in R}\, R_b \qquad \frac{}{\epsilon \in R}\, R_\epsilon$$

$$\frac{w \in R}{awa \in R}\, R_{aa} \qquad \frac{w \in R}{bwb \in R}\, R_{bb}$$

[4 points]

Essentially, the questions asks us to prove that every word $w \in \Sigma^*$ can be decomposed into a sequence of palindromes. While this can be done by discovering larger palindromes, one may also note that every single letter is a palindrome. We can thus induct on the length of the word w as follows:

Base case: $|w| = 0$. Thus, $w = \epsilon$. We have

$$\frac{}{\epsilon \in S}\, S_\epsilon$$

Inductive case: $|w| = n > 0$. The induction hypothesis states that for all $k < n$, any word $v \in \Sigma^*$ of length k is in S. Since $\Sigma = \{a, b\}$, w must end with either a or b. Choose a. The case for b is similar. Then, we can write $w = va$ for some $v \in \Sigma^*$ with length $n - 1 < n$. By the induction hypothesis, there is a proof of $v \in S$. We can then complete the proof for $w \in S$ as:

$$\frac{\dfrac{\cdots}{v \in S}\,\text{IH} \qquad \dfrac{}{a \in R}\, R_a}{va \in S}\, S_R$$

The induction proof is complete, and we have shown that for any word $w \in \Sigma^*$, there exists a proof of $w \in S$. Thus, $\Sigma^* \subseteq L(G^*)$, and it follows that $L(G^*) = \Sigma^*$. Hence, 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`: A

(ii) 0.4 points `THEN`: B

(iii) 0.4 points `ELSE`: P

(iv) 0.4 points `EQ`: I

(v) 0.4 points `ASSIGN`: H

(vi) 0.4 points `LPAREN`: M

(vii) 0.4 points `RPAREN`: L

(viii) 0.6 points `ID`: N

(ix) 0.4 points `INT`: K

(x) 0.6 points `LINESKIP`: G

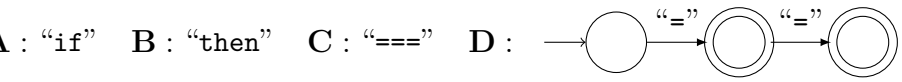
(xi) 0.6 points `BLOCKSKIP`: E

(xii) `WS`: (“`\s`” | “`\t`” | “`\n`”)$^+$

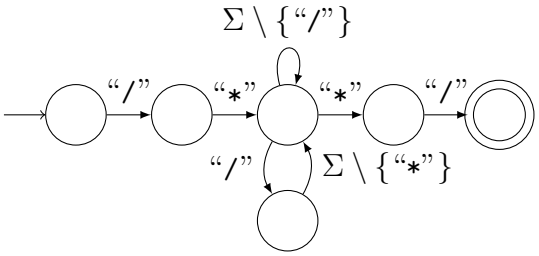
We use the following notation for regular expressions and sets, where e_1 and e_2 are regular expressions, and a, b, and c are characters in the alphabet:

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** :

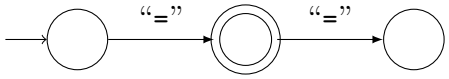


E :

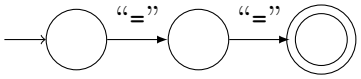


F : "`/*`" $(\Sigma \setminus \{$"`/`"$\})^*$ "`*/`"

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

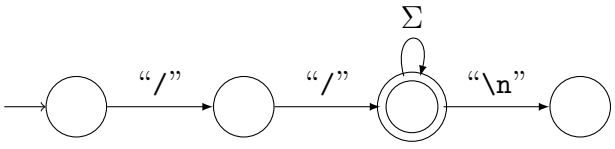


I :

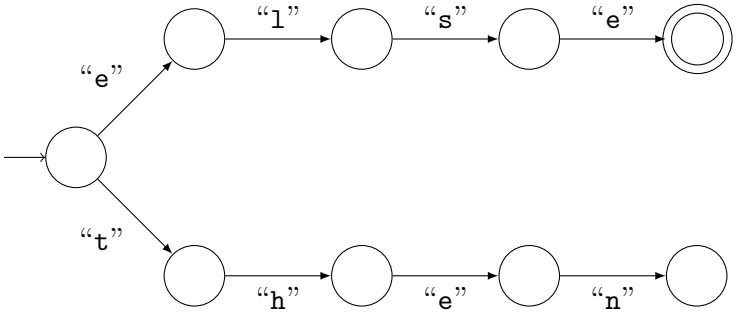


J : $(A \cup D)^*$ **K** : D^+ **L** : "`)`" **M** : "`(`"

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) [1 points] Is the given grammar LL(1)? Justify your answer.

(ii) [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) [1 points] Convert your grammar to be LL(1), if it is not already LL(1).

(iv) [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) [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.*

Solution:

(i) The grammar is not LL(1). F is left-recursive in multiple rules, and thus also shares a common prefix for multiple rules.

(ii) A possible grammar for CNF formulas is:

$$\begin{aligned} S &::= F\ \textbf{EOF} \\ F &::= C\ F' \\ F' &::= \wedge\ C\ F' \mid \epsilon \\ C &::= L \mid (L \vee L\ C') \\ C' &::= \vee\ L\ C' \mid \epsilon \\ L &::= atom \mid \neg atom \end{aligned}$$

(iii) The grammar is already LL(1).

(iv) nullable can be computed as:

$$\begin{aligned} \text{nullable}(S) &= false \\ \text{nullable}(F) &= false \\ \text{nullable}(F') &= true \\ \text{nullable}(C) &= false \\ \text{nullable}(C') &= true \\ \text{nullable}(L) &= false \end{aligned}$$

For first, the constraints are:

$$\begin{aligned} \text{first}(F) &\subseteq \text{first}(S) \\ \text{first}(C) &\subseteq \text{first}(F) \\ \{\wedge\} &\subseteq \text{first}(F') \\ \text{first}(L) &\subseteq \text{first}(C) \\ \{\text{“(”}\} &\subseteq \text{first}(C) \\ \{\vee\} &\subseteq \text{first}(C') \\ \{atom, \neg\} &\subseteq \text{first}(L) \end{aligned}$$

which can be solved to get:

$$\begin{aligned} \text{first}(S) &= \{atom, \neg, \text{“(”}\} \\ \text{first}(F) &= \{atom, \neg, \text{“(”}\} \\ \text{first}(F') &= \{\wedge\} \\ \text{first}(C) &= \{atom, \neg, \text{“(”}\} \\ \text{first}(C') &= \{\vee\} \\ \text{first}(L) &= \{atom, \neg\} \end{aligned}$$

For follow, the constraints are:

$$\begin{aligned}
\{\textbf{EOF}\} &\subseteq \text{follow}(F) \\
\text{first}(F') &\subseteq \text{follow}(C) \\
\text{follow}(F) &\subseteq \text{follow}(C) \\
\text{follow}(F) &\subseteq \text{follow}(F') \\
\text{first}(F') &\subseteq \text{follow}(C) \\
\text{follow}(F') &\subseteq \text{follow}(C) \\
\text{follow}(C) &\subseteq \text{follow}(L) \\
\text{first}(C') &\subseteq \text{follow}(L) \\
\{\vee, \text{“)”}\} &\subseteq \text{follow}(L) \\
\{\text{“)”}\} &\subseteq \text{follow}(C') \\
\text{first}(C') &\subseteq \text{follow}(L)
\end{aligned}$$

which can be solved to get:

$$\begin{aligned}
\text{follow}(S) &= \{\} \\
\text{follow}(F) &= \{\textbf{EOF}\} \\
\text{follow}(F') &= \{\textbf{EOF}\} \\
\text{follow}(C) &= \{\wedge, \textbf{EOF}\} \\
\text{follow}(C') &= \{\text{“)”}\} \\
\text{follow}(L) &= \{\vee, \text{“)”}, \textbf{EOF}\}
\end{aligned}$$

We can finally write the parsing table:

State	*atom*	$\neg$	$\wedge$	$\vee$	(	)	**EOF**
S	1	1			1		
F	1	1			1		
F'			1				2
C	1	1			1		
C'				1		2	
L	1	2					

There are no conflicts in the parsing table, additionally verifying that the grammar is LL(1).

(v) Parsing the string $(a \vee \neg c) \wedge b$:

Stack	**Input**	**Next rule**
S	$(a \vee \neg c) \wedge b$	$S ::= F$ **EOF**
F **EOF**	$(a \vee \neg c) \wedge b$	$F ::= C\ F'$
$C\ F'$ **EOF**	$(a \vee \neg c) \wedge b$	$C ::= (L \vee L\ C')$
$(L \vee L\ C')\ F'$ **EOF**	$(a \vee \neg c) \wedge b$	match "("
$L \vee L\ C')\ F'$ **EOF**	$a \vee \neg c) \wedge b$	$L ::= atom$
$atom \vee L\ C')\ F'$ **EOF**	$a \vee \neg c) \wedge b$	match $atom$
$\vee L\ C')\ F'$ **EOF**	$\vee \neg c) \wedge b$	match $\vee$
$L\ C')\ F'$ **EOF**	$\neg c) \wedge b$	$L ::= \neg atom$
$\neg atom\ C')\ F'$ **EOF**	$\neg c) \wedge b$	match $\neg$
$atom\ C')\ F'$ **EOF**	$c) \wedge b$	match $atom$
$C')\ F'$ **EOF**	$) \wedge b$	$C' ::= \epsilon$
$)\ F'$ **EOF**	$) \wedge b$	match ")"
F' **EOF**	$\wedge b$	$F' ::= \wedge\ C\ F'$
$\wedge\ C\ F'$ **EOF**	$\wedge b$	match $\wedge$
$C\ F'$ **EOF**	$\wedge b$	match $C ::= L$
$L\ F'$ **EOF**	b	$L ::= atom$
$atom\ F'$ **EOF**	b	match $atom$
F' **EOF**	**EOF**	$F' ::= \epsilon$
EOF	**EOF**	match **EOF**

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.

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

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

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

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

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

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

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

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

(i) [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:

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

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

i. [0.5 points] In every valid assignment, $\tau_l = \tau_r$:
A. True B. False

ii. [0.5 points] In every valid assignment, $\tau_l = \mathtt{Bool}$:
A. True **B. False**

iii. [0.5 points] There is a valid assignment where $\tau_z = \mathtt{Int}$:
A. True **B. False**

iv. [0.5 points] There is a valid assignment where $\tau_x = (\mathtt{Int}, \mathtt{Bool})$:
A. True **B. False**

(ii) [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. $(\mathtt{Bool} \Rightarrow \mathtt{Bool}) \Rightarrow \mathtt{Int} \Rightarrow \mathtt{Bool}$
B. $((\mathtt{Int}, \mathtt{Int}) \Rightarrow \mathtt{Int}) \Rightarrow \mathtt{Bool}$
C. $(\mathtt{Bool} \Rightarrow (\mathtt{Bool}, \mathtt{Bool})) \Rightarrow \mathtt{Bool}$
D. $(\mathtt{Bool}, \mathtt{Int}) \Rightarrow \mathtt{Bool}$

(iii) [2 points] Consider the complete labelling of types of subterms of the term t below:

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

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.

Solution: Initial set of constraints:

$$\tau = \tau_z \Rightarrow \tau_{eq}$$
$$\tau_{eq} = \texttt{Bool}$$
$$\tau_l = \tau_r$$
$$\tau_l = (\tau_{funx}, \tau_z)$$
$$\tau_{funx} = \tau_x \Rightarrow \tau_{fx}$$
$$\tau_x = (\tau_{fx}, \tau_{sx})$$
$$\tau_r = (\tau_z, \tau_{funy})$$
$$\tau_{funy} = \tau_y \Rightarrow \tau_{sy}$$
$$\tau_y = (\tau_{fy}, \tau_{sy})$$

for fresh type variables τ_{sx} and τ_{fy}.

Step 1: substituting τ_{eq} and τ_l. Equations above the lines are constraints we have substituted or solved. Equations below the line are constraints we have not yet solved.

$$\tau_l = \tau_r$$
$$\tau_{eq} = \texttt{Bool}$$

$$\tau = \tau_z \Rightarrow \texttt{Bool}$$
$$\tau_r = (\tau_{funx}, \tau_z)$$
$$\tau_{funx} = \tau_x \Rightarrow \tau_{fx}$$
$$\tau_x = (\tau_{fx}, \tau_{sx})$$
$$\tau_r = (\tau_z, \tau_{funy})$$
$$\tau_{funy} = \tau_y \Rightarrow \tau_{sy}$$
$$\tau_y = (\tau_{fy}, \tau_{sy})$$

Step 2: substituting τ_r, τ_x, and τ_y.

$$\tau_l = (\tau_{funx}, \tau_z)$$
$$\tau_{eq} = \texttt{Bool}$$
$$\tau_r = (\tau_{funx}, \tau_z)$$
$$\tau_x = (\tau_{fx}, \tau_{sx})$$
$$\tau_y = (\tau_{fy}, \tau_{sy})$$

$$\tau = \tau_z \Rightarrow \texttt{Bool}$$
$$\tau_{funx} = (\tau_{fx}, \tau_{sx}) \Rightarrow \tau_{fx}$$
$$(\tau_{funx}, \tau_z) = (\tau_z, \tau_{funy})$$
$$\tau_{funy} = (\tau_{fy}, \tau_{sy}) \Rightarrow \tau_{sy}$$

Step 3: unification on $(\tau_{funx}, \tau_z) = (\tau_z, \tau_{funy})$:

$$\tau_l = (\tau_{funx}, \tau_z)$$
$$\tau_{eq} = \texttt{Bool}$$
$$\tau_r = (\tau_{funx}, \tau_z)$$
$$\tau_x = (\tau_{fx}, \tau_{sx})$$
$$\tau_y = (\tau_{fy}, \tau_{sy})$$

$$\tau = \tau_z \Rightarrow \texttt{Bool}$$
$$\tau_{funx} = (\tau_{fx}, \tau_{sx}) \Rightarrow \tau_{fx}$$
$$\tau_{funx} = \tau_z$$
$$\tau_z = \tau_{funy}$$
$$\tau_{funy} = (\tau_{fy}, \tau_{sy}) \Rightarrow \tau_{sy}$$

Step 4: substituting τ_z:

$$\tau_l = (\tau_{funx}, \tau_{funy})$$
$$\tau_{eq} = \texttt{Bool}$$
$$\tau_r = (\tau_{funx}, \tau_{funy})$$
$$\tau_x = (\tau_{fx}, \tau_{sx})$$
$$\tau_y = (\tau_{fy}, \tau_{sy})$$
$$\tau_z = \tau_{funy}$$

$$\tau = \tau_z \Rightarrow \texttt{Bool}$$
$$\tau_{funx} = (\tau_{fx}, \tau_{sx}) \Rightarrow \tau_{fx}$$
$$\tau_{funx} = (\tau_{fy}, \tau_{sy}) \Rightarrow \tau_{sy}$$
$$\tau_{funy} = (\tau_{fy}, \tau_{sy}) \Rightarrow \tau_{sy}$$

Step 5: substituting τ_{funx} and τ_{funy}:

$$\tau_l = (\tau_{funy}, \tau_{funy})$$
$$\tau_{eq} = \texttt{Bool}$$
$$\tau_r = (\tau_{funy}, \tau_{funy})$$
$$\tau_x = (\tau_{fx}, \tau_{sx})$$
$$\tau_y = (\tau_{fy}, \tau_{sy})$$
$$\tau_z = (\tau_{fy}, \tau_{sy}) \Rightarrow \tau_{sy}$$
$$\tau_{funx} = (\tau_{fy}, \tau_{sy}) \Rightarrow \tau_{sy}$$
$$\tau_{funy} = (\tau_{fy}, \tau_{sy}) \Rightarrow \tau_{sy}$$

$$\tau = ((\tau_{fy}, \tau_{sy}) \Rightarrow \tau_{sy}) \Rightarrow \texttt{Bool}$$
$$(\tau_{fx}, \tau_{sx}) \Rightarrow \tau_{fx} = (\tau_{fy}, \tau_{sy}) \Rightarrow \tau_{sy}$$

Step 6: unification on remaining function and pair constraint:

$$\tau_l = (\tau_{funy}, \tau_{funy})$$
$$\tau_{eq} = \texttt{Bool}$$
$$\tau_r = (\tau_{funy}, \tau_{funy})$$
$$\tau_x = (\tau_{fx}, \tau_{sx})$$
$$\tau_y = (\tau_{fy}, \tau_{sy})$$
$$\tau_z = (\tau_{fy}, \tau_{sy}) \Rightarrow \tau_{sy}$$
$$\tau_{funx} = (\tau_{fy}, \tau_{sy}) \Rightarrow \tau_{sy}$$
$$\tau_{funy} = (\tau_{fy}, \tau_{sy}) \Rightarrow \tau_{sy}$$

$$\tau = ((\tau_{fy}, \tau_{sy}) \Rightarrow \tau_{sy}) \Rightarrow \texttt{Bool}$$
$$\tau_{fx} = \tau_{fy}$$
$$\tau_{sx} = \tau_{sy}$$
$$\tau_{fx} = \tau_{sy}$$

Step 7: substituting remaining equalities:

$$\tau_l = (\tau_{funy}, \tau_{funy})$$
$$\tau_{eq} = \texttt{Bool}$$
$$\tau_r = (\tau_{funy}, \tau_{funy})$$
$$\tau_x = (\tau_{fy}, \tau_{fy})$$
$$\tau_y = (\tau_{fy}, \tau_{fy})$$
$$\tau_z = (\tau_{fy}, \tau_{fy}) \Rightarrow \tau_{fy}$$
$$\tau_{funx} = (\tau_{fy}, \tau_{fy}) \Rightarrow \tau_{fy}$$
$$\tau_{funy} = (\tau_{fy}, \tau_{fy}) \Rightarrow \tau_{fy}$$
$$\tau = ((\tau_{fy}, \tau_{fy}) \Rightarrow \tau_{fy}) \Rightarrow \texttt{Bool}$$

No constraints left

Giving the final general type $\tau = ((\tau_{fy}, \tau_{fy}) \Rightarrow \tau_{fy}) \Rightarrow \texttt{Bool}$, for every possible value of the variable τ_{fy}.

Scratch area

End