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 **50**.

Question:	1	2	3	4	5	6	Total
Points:	10	5	10	10	5	10	50
Score:							

Page intentionally left blank.

Question 1. (5 points)

Consider the following language L over $\{a, b, c, d\}$:

$$\{a^l b^m c^n d^{l+m+n} \in \{a, b, c, d\}^* \mid l, m, n \geq 0\}$$

(i) $\boxed{\text{3 points}}$ Produce a context-free grammar G such that $L(G) = L$, i.e. that G generates the language L, and show a derivation of the word ab^2cd^4 in G.

(ii) $\boxed{\text{2 points}}$ Prove that L is not a regular language.

Solution: Context-free grammar:

$$\begin{aligned} A &::= aAd \mid B \\ B &::= bBd \mid C \\ C &::= cCd \mid \epsilon \end{aligned}$$

Derivation of $ab^2cd^4 = abbcdddd$:

$$\begin{aligned} A &\to aAd \\ &\to aBd \\ &\to abBdd \\ &\to abbBddd \\ &\to abbCddd \\ &\to abbcCdddd \\ &\to abbcdddd \\ &= ab^2cd^4 \end{aligned}$$

We prove that L is not a regular language by using the pumping lemma. To the contrary, assume that L is regular. Then, by the pumping lemma, there exists a pumping length p such that any word $w \in L$ with $|w| \geq p$ can be decomposed into three parts $w = xyz$ such that:

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

Choose a candidate word $w = a^p b^p d^{2p}$. Then, we can decompose w as $w = xyz$ with the conditions as above. Since $|xy| \leq p$, we know that xy must be in the first part of the word, and thus $x = a^n$ and $y = a^m$ for some $n \geq 0, m > 0, n + m \leq p$. However, $xy^iz = a^{p+m\cdot(i-1)} b^p d^{2p}$ is not in L for any $i \neq 1$. Thus, w cannot be pumped, and we have a contradiction.

Therefore, L is not a regular language.

Question 2. (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

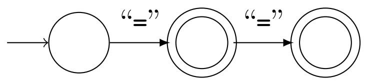
(xi) 0.6 points `BLOCKSKIP`: E

(xii) `WS`: $(\text{"\textbackslash s"} \mid \text{"\textbackslash t"} \mid \text{"\textbackslash n"})^+$

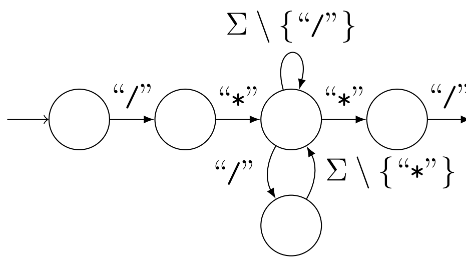
We use the following notation for regular expressions and sets, where e_1 and e_2 are regular expressions, and a, b, and c are characters in the alphabet:

1. e_1e_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** :

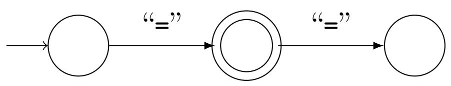


E :

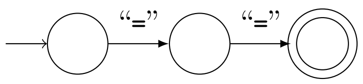


F : "`/*`" $(\Sigma \setminus \{$"`/`"$\})^*$ "`*/`"

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

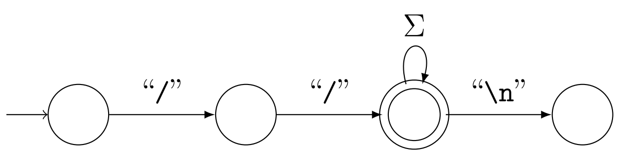


I :

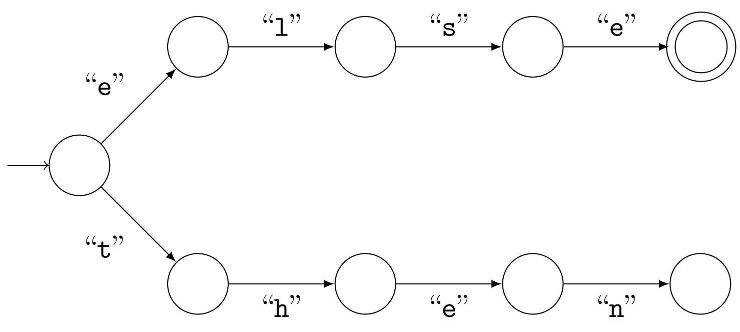


J : $(A \cup D)^*$ **K** : D^+ **L** : "`)`" **M** : "`(`"

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



P :



Question 3. (5 points)

Consider a context-free language L over alphabet Σ defined by some grammar G, with start symbol S. We define the language L' by the following grammar G':

$$R ::= RS \mid \epsilon$$

where R is the start symbol of G', and $L(G') = L'$.

(i) $\boxed{\text{1 point}}$ Express the grammar G' as a set of rules defining an inductive relation. You may assume that the inductive relation $S \subseteq \Sigma^*$ has been defined. Note that a set is a special case of an inductive relation, having one argument.

Solution:

$$\frac{w \in R}{w \in G'}\, G'_{start}$$

$$\frac{}{\epsilon \in R}\, R_\epsilon \qquad \frac{w \in R \qquad v \in S}{wv \in R}\, R_s$$

(ii) $\boxed{\text{4 points}}$ Use these inductive definitions to prove that $L' = L^*$. Use the fact that for any grammar G and word w, $w \in L(G) \iff w \in G$ where G is defined as an inductive relation.

Recall that $w \in L^*$ if and only if for some $n \geq 0$, there exist $w_1, \dots, w_n$ such that $w = w_1 \dots w_n$ and $w_i \in L$ for each $0 < i \leq n$.

Solution: Proving that $L' = L^*$. We must show that $w \in L'$ if and only if $w \in L^*$. We will interchangeably use $w \in L' \iff w \in G' \iff w \in R$, as well as $w \in L \iff w \in G \iff w \in S$.

1. $L' \subseteq L^*$: we need to show that if there is a derivation of $w \in R$, then there exist $w_1, \dots, w_n$ such that $w = w_1 \dots w_n$ and $w_i \in S$ for each $0 < i \leq n$.

   We prove this by induction on the depth of the derivation of $w \in R$.

   Base case: The proof is of depth 1. The only possible case is the proof ends with R_ϵ, so $w = \epsilon$:

   $$\frac{}{\epsilon \in R}\, R_\epsilon$$

   Choosing $n = 0$, we have $w = \epsilon \in L^*$ vacuously.

   Inductive case: The proof is of depth $k \geq 2$. The inductive hypothesis states that for any v, if there is a derivation of $v \in R$ of depth $< k$, then $v \in L^*$. The last step of the proof must be R_s, so we have:

   $$\frac{\dfrac{\dots}{v_1 \in R} \qquad \dfrac{\dots}{v_2 \in S}}{v_1 v_2 \in R}\, R_s$$

where $w = v_1 v_2$. Given that we have a derivation of $v_1 \in R$ with depth $< k$, from the inductive hypothesis, we know that there exists an $m \geq 0$ such that $v_1 = u_1 \dots u_m$ and there is a derivation of $u_i \in S$ for each $0 < i \leq m$. Thus, $w = u_1 \dots u_m v_2$. But, we also have a derivation for $v_2 \in S$. Finally, choose $n = m + 1$, and choose $w_1 = u_1, \dots, w_m = u_m$, and $w_{m+1} = v_2$. We have $w \in L^*$ again.

2. $L^* \subseteq L'$: we need to show that if there exist $w_1, \dots, w_n$ such that $w = w_1 \dots w_n$ and there is a derivation of $w_i \in L$ for each $0 < i \leq n$, then there is a derivation of $w \in L'$.

   We prove this by induction on n.

   Base case: $n = 0$. This means that $w = \epsilon$. We can construct a proof for $w \in R$:

$$\frac{}{\epsilon \in R}\, R_\epsilon$$

   Inductive case: $n = k + 1$, there exist words $w_1, \dots, w_{k+1}$ such that $w = w_1 \dots w_{k+1}$ and there is a derivation of $w_i \in S$ for each $0 < i \leq k + 1$. The induction hypothesis states that for any word $v \in L^*$ with a decomposition of $\leq k$ words in L, there exists a derivation of $v \in R$. Consider the word $w_1 \dots w_k$. By the induction hypothesis, there exists a derivation of $w_1 \dots w_k \in R$. We can construct a proof for $w \in R$:

$$\frac{\dfrac{\dots}{w_1 \dots w_k \in R} \qquad \dfrac{\dots}{w_{k+1} \in S}}{w_1 \dots w_k w_{k+1} \in R}\, R_s$$

   Thus, the proof is complete, and $L' = L^*$.

Question 4. (5 points)

Consider the following grammar for a language consisting of variables, constructors, and match-case statements:

$$
\begin{aligned}
S &::= \mathit{Expr}\ \textbf{EOF} && (0)\\
\mathit{Expr} &::= \mathit{SimpleExpr}\ \mathit{Expr}' && (1)\\
\mathit{Expr}' &::= \epsilon \mid \mathit{Match} && (2)\\
\mathit{SimpleExpr} &::= \texttt{var} \mid \mathit{Cons} \mid (\mathit{Expr}) && (3)\\
\mathit{Cons} &::= \texttt{id}(\mathit{ExprList}) && (4)\\
\mathit{ExprList} &::= \epsilon \mid \mathit{NExprList} && (5)\\
\mathit{NExprList} &::= \mathit{Expr} \mid \mathit{Expr},\ \mathit{NExprList} && (6)\\
\mathit{Match} &::= \texttt{match}\ \mathit{CaseList} && (7)\\
\mathit{CaseList} &::= \mathit{Case} \mid \mathit{Case}\ \mathit{CaseList} && (8)\\
\mathit{Case} &::= \texttt{case}\ \mathit{SimpleExpr}\ \texttt{=>}\ \mathit{Expr} && (9)
\end{aligned}
$$

where `var`, `id`, `match`, `case`, `=>`, `(`, `)`, `,`, and **EOF** are all terminal tokens.

This question has four (4) subparts, one on each of the following pages.

(i) $\boxed{\text{1 point}}$ Compute nullable for each non-terminal in the grammar.

$$\begin{aligned}
\text{nullable}(S) &= false\\
\text{nullable}(Expr) &= false\\
\text{nullable}(Expr') &= true\\
\text{nullable}(SimpleExpr) &= false\\
\text{nullable}(Cons) &= false\\
\text{nullable}(ExprList) &= true\\
\text{nullable}(NExprList) &= false\\
\text{nullable}(Match) &= false\\
\text{nullable}(CaseList) &= false\\
\text{nullable}(Case) &= false
\end{aligned}$$

(ii) $\boxed{\text{1 point}}$ Compute the first($\cdot$) sets for each non-terminal in the grammar.

Constraints for first sets, labelled by which rule number they come from:

$$\begin{aligned}
\text{first}(S) &\subseteq \text{first}(\mathit{Expr}) && (0)\\
\text{first}(\mathit{SimpleExpr}) &\subseteq \text{first}(\mathit{Expr}) && (1)\\
\text{first}(\mathit{Match}) &\subseteq \text{first}(\mathit{Expr}') && (2)\\
\{(, \texttt{var}\} &\subseteq \text{first}(\mathit{SimpleExpr}) && (3)\\
\text{first}(\mathit{Cons}) &\subseteq \text{first}(\mathit{SimpleExpr}) && (3)\\
\{\texttt{id}\} &\subseteq \text{first}(\mathit{Cons}) && (4)\\
\text{first}(\mathit{NExprList}) &\subseteq \text{first}(\mathit{ExprList}) && (5)\\
\text{first}(\mathit{Expr}) &\subseteq \text{first}(\mathit{NExprList}) && (6)\\
\{\texttt{match}\} &\subseteq \text{first}(\mathit{Match}) && (7)\\
\text{first}(\mathit{Case}) &\subseteq \text{first}(\mathit{CaseList}) && (8)\\
\{\texttt{case}\} &\subseteq \text{first}(\mathit{Case}) && (9)
\end{aligned}$$

Which can be solved to get:

$$\begin{aligned}
\text{first}(S) &= \{(, \texttt{var}, \texttt{id}\}\\
\text{first}(\mathit{Expr}) &= \{(, \texttt{var}, \texttt{id}\}\\
\text{first}(\mathit{Expr}') &= \{\texttt{match}\}\\
\text{first}(\mathit{SimpleExpr}) &= \{(, \texttt{var}, \texttt{id}\}\\
\text{first}(\mathit{Cons}) &= \{\texttt{id}\}\\
\text{first}(\mathit{ExprList}) &= \{(, \texttt{var}, \texttt{id}\}\\
\text{first}(\mathit{NExprList}) &= \{(, \texttt{var}, \texttt{id}\}\\
\text{first}(\mathit{Match}) &= \{\texttt{match}\}\\
\text{first}(\mathit{CaseList}) &= \{\texttt{case}\}\\
\text{first}(\mathit{Case}) &= \{\texttt{case}\}
\end{aligned}$$

(iii) $\boxed{\text{1 point}}$ Compute the follow($\cdot$) sets for each non-terminal except S in the grammar.

Constraints for follow sets, labelled by which rule number they arise from:

$$
\begin{aligned}
\{\textbf{EOF}\} &\subseteq \text{follow}(\mathit{Expr}) && (0)\\
\text{first}(\mathit{Expr}') &\subseteq \text{follow}(\mathit{SimpleExpr}) && (1)\\
\text{follow}(\mathit{Expr}) &\subseteq \text{follow}(\mathit{SimpleExpr}) && (1)\\
\text{follow}(\mathit{Expr}) &\subseteq \text{follow}(\mathit{Expr}') && (1)\\
\text{follow}(\mathit{Expr}') &\subseteq \text{follow}(\mathit{Match}) && (2)\\
\text{follow}(\mathit{SimpleExpr}) &\subseteq \text{follow}(\mathit{Cons}) && (3)\\
\{)\} &\subseteq \text{follow}(\mathit{Expr}) && (3)\\
\{)\} &\subseteq \text{follow}(\mathit{ExprList}) && (4)\\
\text{follow}(\mathit{ExprList}) &\subseteq \text{follow}(\mathit{NExprList}) && (5)\\
\{,\} &\subseteq \text{follow}(\mathit{Expr}) && (6)\\
\text{follow}(\mathit{Match}) &\subseteq \text{follow}(\mathit{CaseList}) && (7)\\
\text{follow}(\mathit{CaseList}) &\subseteq \text{follow}(\mathit{Case}) && (8)\\
\text{first}(\mathit{CaseList}) &\subseteq \text{follow}(\mathit{Case}) && (8)\\
\{\texttt{=>}\} &\subseteq \text{follow}(\mathit{SimpleExpr}) && (9)\\
\text{follow}(\mathit{Case}) &\subseteq \text{follow}(\mathit{Expr}) && (9)
\end{aligned}
$$

which can be solved to get

$$
\begin{aligned}
\text{follow}(\mathit{Expr}) &= \{\texttt{case}, \text{“,”}, \text{“)”}, \textbf{EOF}\}\\
\text{follow}(\mathit{Expr}') &= \{\texttt{case}, \text{“,”}, \text{“)”}, \textbf{EOF}\}\\
\text{follow}(\mathit{SimpleExpr}) &= \{\texttt{match}, \texttt{case}, \texttt{=>}, \text{“,”}, \text{“)”}, \textbf{EOF}\}\\
\text{follow}(\mathit{Cons}) &= \{\texttt{match}, \texttt{case}, \texttt{=>}, \text{“,”}, \text{“)”}, \textbf{EOF}\}\\
\text{follow}(\mathit{ExprList}) &= \{)\}\\
\text{follow}(\mathit{NExprList}) &= \{)\}\\
\text{follow}(\mathit{Match}) &= \{\texttt{case}, \text{“,”}, \text{“)”}, \textbf{EOF}\}\\
\text{follow}(\mathit{CaseList}) &= \{\texttt{case}, \text{“,”}, \text{“)”}, \textbf{EOF}\}\\
\text{follow}(\mathit{Case}) &= \{\texttt{case}, \text{“,”}, \text{“)”}, \textbf{EOF}\}
\end{aligned}
$$

(iv) $\boxed{\text{2 points}}$ Construct the parsing table for this grammar. Use the production options in order as given to fill in the parse table. For example, with $Expr' ::= \epsilon \mid Match$, write 1 for choosing the rule $Expr' ::= \epsilon$, and 2 for choosing $Expr' ::= Match$. Show that the grammar is *not* LL(1) by marking every conflict in the table.

Non-terminal	`var`	`id`	`(`	`)`	`match`	`case`	`=>`	`,`	**EOF**
Expr	1	1	1						
Expr'				1	2	1		1	1
SimpleExpr	1	2	3						
Cons		1							
ExprList	2	2	2	1					
NExprList	**1, 2**	**1, 2**	**1, 2**						
Match					1				
CaseList						**1, 2**			
Case						1			

Question 5. (5 points)

For the grammar defined in Question 4, produce an equivalent LL(1) grammar. You do not need to show the parsing table or otherwise prove that the new grammar is LL(1).

Solution:

$$\begin{aligned}
S &::= \mathit{Expr}\ \textbf{EOF} \\
\mathit{Expr} &::= \mathit{SimpleExpr}\ \mathit{Expr}' \\
\mathit{Expr}' &::= \epsilon \mid \mathit{Match} \\
\mathit{SimpleExpr} &::= \texttt{var} \mid \mathit{Cons} \mid (\mathit{Expr}) \\
\mathit{Cons} &::= \texttt{word}(\mathit{ExprList}) \\
\mathit{Match} &::= \texttt{match}\ \mathit{CaseList}
\end{aligned}$$

Removing common prefixes from *NExprList*:

$$\begin{aligned}
\mathit{ExprList} &::= \epsilon \mid \mathit{NExprList} \\
\mathit{NExprList} &::= \mathit{Expr}\ \mathit{NExprList}' \\
\mathit{NExprList}' &::= \epsilon \mid\ ,\ \mathit{NExprList}
\end{aligned}$$

Removing common prefixes from *CaseList* and adjusting for *Match* priority:

$$\begin{aligned}
\mathit{CaseList} &::= \texttt{case}\ \mathit{SimpleExpr}\ \texttt{=>}\ \mathit{SimpleExpr}\ \mathit{MaybeMatch} \\
\mathit{MaybeMatch} &::= \epsilon \mid \mathit{CaseList} \mid \mathit{Match}
\end{aligned}$$

Question 6. (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 $\tau_1, \tau_2, \ldots, \tau_5$ ascribing subterms of t as shown:

$$((\mathtt{x} : \tau_1) \Rightarrow (\mathtt{y} : \tau_2) \Rightarrow (\mathtt{if}\ (\mathtt{fst}(x) : \tau_3)\ \mathtt{then}\ \mathtt{snd}(x)\ \mathtt{else}\ 1 + \mathtt{snd}(x)) : \tau_4) : \tau_5$$

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

i. In every valid assignment, $\tau_1 = (\mathtt{Int}, \mathtt{Bool})$:
   A. True **B. False**

ii. In every valid assignment, $\tau_3 = \mathtt{Bool}$:
   A. True B. False

iii. There is a valid assignment where $\tau_2 = \mathtt{Int}$:
   A. True B. False

iv. In every valid assignment, $\tau_5 = (\tau_1 \Rightarrow \tau_2 \Rightarrow \tau_4)$:
   A. True B. False

(ii) $\boxed{\text{1 point}}$ 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. $((\mathtt{Bool}, \mathtt{Int}) \Rightarrow \mathtt{Bool}) \Rightarrow \mathtt{Int}$
B. $(\mathtt{Bool}, \mathtt{Int}) \Rightarrow \mathtt{Int} \Rightarrow \mathtt{Int}$
C. $(\mathtt{Int}, \mathtt{Bool}) \Rightarrow (\mathtt{Bool}, \mathtt{Int}) \Rightarrow \mathtt{Int}$
D. $(\mathtt{Int}, \mathtt{Bool}) \Rightarrow \mathtt{Bool} \Rightarrow \mathtt{Int}$

(iii) $\boxed{\text{2 points}}$ Consider the following term r:

$$\mathtt{x} \Rightarrow \mathtt{fst}(x)(\mathtt{snd}(x)) + \mathtt{snd}(x)$$

where we assign type variables to the subterms as follows:

$$\mathtt{x} : \tau_1 \quad \mathtt{fst(x)} : \tau_2 \quad \mathtt{snd(x)} : \tau_3$$
$$\mathtt{fst(x)(snd(x))} : \tau_4$$
$$\mathtt{fst}(x)(\mathtt{snd(x)}) + \mathtt{snd(x)} : \tau_5$$
$$\mathtt{x} \Rightarrow \mathtt{fst}(x)(\mathtt{snd(x)}) + \mathtt{snd(x)} : \tau_6$$

The initial unification constraints for type checking r are:

$$\tau_6 = \tau_1 \Rightarrow \tau_5$$
$$\tau_5 = \mathtt{Int}$$
$$\tau_4 = \mathtt{Int}$$
$$\tau_3 = \mathtt{Int}$$
$$\tau_2 = \tau_3 \Rightarrow \tau_4$$
$$\tau_1 = (\tau_3', \tau_3)$$
$$\tau_1 = (\tau_2, \tau_2')$$

for fresh type variables τ_2' and τ_3'. Note that $=$ has lower precedence than the type constructors $(\Rightarrow, (\cdot, \cdot))$, so $\tau_6 = \tau_1 \Rightarrow \tau_5$ is parsed as "τ_6 equals $\tau_1 \to \tau_5$".

Consider the following possible set of constraints at different unification steps (**this page and next**). The current set of unsolved constraints is listed *below the bar*. Whenever we substitute a type variable, we add the mapping to the list *above the bar*.

$$\textbf{Init}: \begin{array}{r} \emptyset \\ \hline \tau_6 = \tau_1 \Rightarrow \tau_5 \\ \tau_5 = \texttt{Int} \\ \tau_4 = \texttt{Int} \\ \tau_3 = \texttt{Int} \\ \tau_2 = \tau_3 \Rightarrow \tau_4 \\ \tau_1 = (\tau_3', \tau_3) \\ \tau_1 = (\tau_2, \tau_2') \end{array}$$

$$\textbf{1}: \begin{array}{r} \tau_1 = (\texttt{Int}, \texttt{Int}) \\ \tau_2 = \texttt{Int} \\ \tau_3 = \texttt{Int} \\ \tau_4 = \texttt{Int} \\ \tau_5 = \texttt{Int} \\ \tau_6 = (\texttt{Int}, \texttt{Int}) \Rightarrow \texttt{Int} \\ \tau_2' = \texttt{Int} \\ \tau_3' = \texttt{Int} \Rightarrow \texttt{Int} \\ \hline \emptyset \end{array}$$

$$\textbf{2}: \begin{array}{r} \tau_1 = (\tau_3', \texttt{Int}) \\ \tau_2 = \texttt{Int} \\ \tau_3 = \texttt{Int} \\ \tau_4 = \texttt{Int} \\ \tau_5 = \texttt{Int} \\ \hline \tau_6 = (\tau_3', \texttt{Int}) \Rightarrow \texttt{Int} \\ (\tau_3', \texttt{Int}) = (\texttt{Int}, \tau_2') \end{array}$$

$$\textbf{3}: \begin{array}{r} \tau_3 = \texttt{Int} \\ \tau_4 = \texttt{Int} \\ \tau_5 = \texttt{Int} \\ \hline \tau_6 = \tau_1 \Rightarrow \texttt{Int} \\ \tau_2 = \texttt{Int} \\ \tau_1 = (\tau_3', \texttt{Int}) \\ \tau_1 = (\tau_2, \tau_2') \end{array}$$

$$\textbf{4}: \begin{array}{r} \tau_1 = (\tau_3', \texttt{Int}) \\ \tau_2 = \texttt{Int} \Rightarrow \texttt{Int} \\ \tau_3 = \texttt{Int} \\ \tau_4 = \texttt{Int} \\ \tau_5 = \texttt{Int} \\ \hline \tau_6 = (\tau_3', \texttt{Int}) \Rightarrow \texttt{Int} \\ \tau_3' = \texttt{Int} \Rightarrow \texttt{Int} \\ \texttt{Int} = \tau_2' \end{array}$$

$$\textbf{5}: \begin{array}{r} \tau_1 = (\tau_3', \texttt{Int}) \\ \tau_2 = \texttt{Int} \Rightarrow \texttt{Int} \\ \tau_3 = \texttt{Int} \\ \tau_4 = \texttt{Int} \\ \tau_5 = \texttt{Int} \\ \hline \tau_6 = (\tau_3', \texttt{Int}) \Rightarrow \texttt{Int} \\ (\tau_3', \texttt{Int}) = (\texttt{Int} \Rightarrow \texttt{Int}, \tau_2') \end{array}$$

$$\textbf{6}: \begin{array}{r} \tau_3 = \texttt{Int} \\ \tau_4 = \texttt{Int} \\ \tau_5 = \texttt{Int} \\ \hline \tau_6 = \tau_1 \Rightarrow \texttt{Int} \\ \tau_2 = \texttt{Int} \Rightarrow \texttt{Int} \\ \tau_1 = (\tau_3', \texttt{Int}) \\ \tau_1 = (\tau_2, \tau_2') \end{array}$$

$$\boxed{\mathbf{7}: \begin{array}{r} \tau_1 = (\texttt{Int} \Rightarrow \texttt{Int}, \texttt{Int}) \\ \tau_2 = \texttt{Int} \Rightarrow \texttt{Int} \\ \tau_3 = \texttt{Int} \\ \tau_4 = \texttt{Int} \\ \tau_5 = \texttt{Int} \\ \tau_6 = (\texttt{Int} \Rightarrow \texttt{Int}, \texttt{Int}) \Rightarrow \texttt{Int} \\ \tau_2' = \texttt{Int} \\ \tau_3' = \texttt{Int} \Rightarrow \texttt{Int} \\ \hline \emptyset \end{array}}$$

$$\boxed{\mathbf{8}: \begin{array}{r} \tau_1 = (\texttt{Int} \Rightarrow \texttt{Int}, \texttt{Int}) \\ \tau_2 = \texttt{Int} \Rightarrow \texttt{Int} \\ \tau_3 = \texttt{Int} \\ \tau_4 = \texttt{Int} \\ \tau_5 = \texttt{Int} \\ \tau_2' = \texttt{Int} \\ \tau_3' = \texttt{Int} \Rightarrow \texttt{Int} \\ \hline \tau_6 = (\texttt{Int} \Rightarrow \texttt{Int}, \texttt{Int}) \Rightarrow \texttt{Int} \end{array}}$$

$$\boxed{\mathbf{9}: \begin{array}{r} \tau_1 = (\tau_3', \texttt{Int}) \\ \tau_2 = \texttt{Int} \\ \tau_3 = \texttt{Int} \\ \tau_4 = \texttt{Int} \\ \tau_5 = \texttt{Int} \\ \hline \tau_6 = (\tau_3', \texttt{Int}) \Rightarrow \texttt{Int} \\ \tau_3' = \texttt{Int} \\ \texttt{Int} = \tau_2' \end{array}}$$

$$\boxed{\mathbf{10}: \begin{array}{r} \tau_2 = \texttt{Int} \\ \tau_3 = \texttt{Int} \\ \tau_4 = \texttt{Int} \\ \tau_5 = \texttt{Int} \\ \hline \tau_6 = \tau_1 \Rightarrow \texttt{Int} \\ \tau_1 = (\tau_3', \texttt{Int}) \\ \tau_1 = (\texttt{Int}, \tau_2') \end{array}}$$

$$\boxed{\mathbf{11}: \begin{array}{r} \tau_1 = (\texttt{Int}, \texttt{Int}) \\ \tau_2 = \texttt{Int} \\ \tau_3 = \texttt{Int} \\ \tau_4 = \texttt{Int} \\ \tau_5 = \texttt{Int} \\ \tau_2' = \texttt{Int} \\ \tau_3' = \texttt{Int} \\ \hline \tau_6 = (\texttt{Int}, \texttt{Int}) \Rightarrow \texttt{Int} \end{array}}$$

$$\boxed{\mathbf{12}: \begin{array}{r} \tau_1 = (\tau_3', \texttt{Int}) \\ \tau_2 = \texttt{Int} \\ \tau_3 = \texttt{Int} \\ \tau_4 = \texttt{Int} \\ \tau_5 = \texttt{Int} \\ \hline \tau_6 = (\tau_3', \texttt{Int}) \Rightarrow \texttt{Int} \\ \texttt{Int} = \texttt{Int} \\ \tau_3' = \tau_2' \end{array}}$$

$$\boxed{\mathbf{13}: \begin{array}{r} \tau_2 = \texttt{Int} \Rightarrow \texttt{Int} \\ \tau_3 = \texttt{Int} \\ \tau_4 = \texttt{Int} \\ \tau_5 = \texttt{Int} \\ \hline \tau_6 = \tau_1 \Rightarrow \texttt{Int} \\ \tau_1 = (\tau_3', \texttt{Int}) \\ \tau_1 = (\texttt{Int} \Rightarrow \texttt{Int}, \tau_2') \end{array}}$$

$$\boxed{\mathbf{14}: \begin{array}{r} \tau_1 = (\tau_3', \texttt{Int}) \\ \tau_2 = \texttt{Int} \Rightarrow \texttt{Int} \\ \tau_3 = \texttt{Int} \\ \tau_4 = \texttt{Int} \\ \tau_5 = \texttt{Int} \\ \hline \tau_6 = (\tau_3', \texttt{Int}) \Rightarrow \texttt{Int} \\ \texttt{Int} = \texttt{Int} \\ \tau_3' = \tau_2' \end{array}}$$

Circle an order of unification steps that leads to a correct and complete type inference for r, i.e. ending with assignment of all type variables.

A. **Init, 6, 13, 5, 14, 11, 1**
B. **Init, 3, 13, 5, 4, 8, 7**
C. **Init, 6, 10, 12, 11, 1**
D. **Init, 6, 10, 9, 4, 8, 7**
E. **Init, 3, 10, 12, 11, 1**
F. **Init, 3, 10, 5, 4, 8, 1**
G. **Init, 6, 13, 5, 4, 8, 7**
H. **Init, 3, 10, 9, 11, 7**

Scratch area

End