

---

# Computer Language Processing

## Quiz

Thursday, November 29, 2018

---

Read all questions and ask us for any clarifications as early as possible.

Unless specified, you do not need to justify your answers.

Write your answers using dark permanent pen directly on this exam sheet.

For multiple-choice questions, please circle only the letters corresponding to your choices, not the text of choices themselves.

We will not grade any additional sheets.

Exercise	Points	Points Achieved
1	20	
2	20	
3	20	
4	20	
5	20	
<b>Total</b>	100	

This page intentionally left blank.

## Exercise 1: Lexing (20 points)

### Question 1.1

Circle all of the regular expressions whose language is **disjoint** from the language of  $(a|b)^+c^*$ :

- A.  $a^*b^*c^*$
- B.  $c^+$
- C.  $(b|c)^+$
- D.  $(a|b)^+c^+a^+$

### Question 1.2

Consider the lexical analyzer with token classes defined by following regular expressions:

$\langle$	IMP	$:=$	$=>$ ,
	LESSEQ	$:=$	$<=$ ,
	LESS	$:=$	$<$ ,
	GREATER	$:=$	$>$ $\rangle$

Circle all the correct statements amongst the ones below, assuming the longest match rule is applied:

- A.  $<=>$  will be tokenized to LESS, IMP.
- B.  $<=<$  will be tokenized to LESSEQ, LESS.
- C.  $<==>$  will be tokenized to LESSEQ, IMP.
- D.  $<>$  will be tokenized to GREATER, LESS.

## Exercise 2: Grammars & Parsing (20 points)

Consider the following grammar:

$$\langle Object \rangle ::= \text{'{' } \langle ObjectRest \rangle}$$
$$\langle ObjectRest \rangle ::= \langle ObjectContent \rangle \text{'}' } \mid \text{'{' } \text{'}' }$$
$$\langle ObjectContent \rangle ::= \text{STRING } \text{'.' } \langle Value \rangle \langle ObjectContentRep \rangle$$
$$\langle ObjectContentRep \rangle ::= \varepsilon \mid \text{' , ' } \langle ObjectContent \rangle$$
$$\langle Array \rangle ::= \text{'[' } \langle ArrayRest \rangle$$
$$\langle ArrayRest \rangle ::= \langle ArrayContent \rangle \text{' ] ' } \mid \text{' ] ' }$$
$$\langle ArrayContent \rangle ::= \langle Value \rangle \langle ArrayContentRep \rangle$$
$$\langle ArrayContentRep \rangle ::= \varepsilon \mid \text{' , ' } \langle ArrayContent \rangle$$
$$\langle Value \rangle ::= \langle Object \rangle \mid \langle Array \rangle \mid \text{STRING } \mid \text{'true' } \mid \text{'false' }$$

Note that we do **not** have an end-of-file marker in the above grammar. Do **not** add one to the grammar. Do **not** assume such an implicit marker is present.

### Question 2.1

Which of the following statements are true? Circle all correct answers. Hint: be bold!

- A. **STRING is a terminal.**
- B.  $\langle Object \rangle$  is a terminal.
- C.  $\langle ArrayContent \rangle$  **is a non-terminal.**
- D. **'true' is a terminal.**

### Question 2.2

Which of the following statements are true? Circle the correct answers.

- A.  $\langle ObjectContent \rangle$  is nullable.
- B.  $\langle ArrayRest \rangle$  is nullable.
- C.  $\langle ArrayContentRep \rangle$  is nullable.
- D.  $\langle Value \rangle$  is nullable.

### Question 2.3

Which of the following statements are true? Circle the correct answers.

- A.  $\text{FIRST}(\langle ObjectContent \rangle)$  contains  $\langle , \rangle$ .
- B.  $\text{FIRST}(\langle Object \rangle)$  is disjoint from  $\text{FIRST}(\langle Array \rangle)$ .
- C.  $\text{FIRST}(\langle Value \rangle)$  is a subset of  $\text{FIRST}(\langle Array \rangle)$ .
- D.  $\text{FIRST}(\langle ArrayRest \rangle)$  contains  $\langle ' \rangle$ .

### Question 2.4

Which of the following statements are true? Circle the correct answers.

- A.  $\text{FOLLOW}(\langle Object \rangle)$  is empty.
- B.  $\text{FOLLOW}(\langle ArrayContent \rangle)$  is equal to  $\{ \langle ' \rangle \}$ .
- C.  $\text{FOLLOW}(\langle ArrayRest \rangle)$  contains  $\langle , \rangle$ .
- D.  $\text{FOLLOW}(\langle Value \rangle)$  contains  $\langle \{ \rangle$ .

### Question 2.5

Which of the following statements are true? Circle the correct answers.

- A. The grammar is ambiguous.
- B. The grammar is LL(1).
- C. The grammar is in Chomsky normal form.
- D. The language defined by the grammar is non-regular.

### Exercise 3: $L^*$ Membership-Checking Algorithm (20 points)

Let  $L \subseteq A^*$  be a language given by a (terminating) algorithm  $\text{fL} : A^* \rightarrow \{\text{true}, \text{false}\}$  such that, for all  $w \in A^*$ ,

$$(\text{fL}(w) = \text{true}) \iff (w \in L)$$

In other words,  $\text{fL}$  checks whether a word belongs to  $L$ . (We do not have any other information about the language  $L$ .)

Let  $S = L^*$ .

#### Your goal

Write an algorithm  $\text{fS}$  using Scala-like notation to check if word belongs to  $S$ , that is,  $\text{fS} : A^* \rightarrow \{\text{true}, \text{false}\}$  such that

$$(\text{fS}(w) = \text{true}) \iff (w \in S)$$

Your algorithm can inspect the word  $w$  as well as apply the function  $\text{fL}$  on arbitrary words. For full points, your solutions should only invoke  $\text{fL}$  a polynomial (in the size of the input word) number of times. You may use any auxiliary data structures from Scala library. For your convenience, a small API for **List** and **Array** follows.

#### API

##### Methods of **List**[A]

- `def apply(n: Int): A`: Returns the  $n^{\text{th}}$  element of **this** list.
- `def take(n: Int): List[A]`: Returns first  $n$  elements of **this** list.
- `def drop(n: Int): List[A]`: Returns a copy of **this** list with the first  $n$  elements dropped.
- `def splitAt(n: Int): (List[A], List[A])`: Returns the pair `(this.take(n), this.drop(n))`.

##### Methods of **Array**[A]

- `def apply(n: Int): A`: Returns the  $n^{\text{th}}$  element of **this** array.
- `def update(n: Int, x: A): Unit`: Updates the  $n^{\text{th}}$  element of **this** array. Modifies the array.

##### Methods of the **Array** companion object

- `Array.tabulate[A](n: Int)(f: Int => A)`: Returns a new one-dimensional array of size  $n$ , initially populated by  $f$ .
- `Array.tabulate[A](n1: Int, n2: Int)(f: (Int, Int) => A): Array[Array[A]]`: Returns a new two-dimensional array of size  $n1 \times n2$ , initially populated by  $f$ .
- `Array.ofDim[A](n: Int): Array[A]`: Returns a new one-dimensional array of size  $n$ .
- `Array.ofDim[A](n1: Int, n2: Int): Array[Array[A]]`: Returns a new two-dimensional array of size  $n1 \times n2$ .

```
def fS[A](fL: List[A] => Boolean, w: List[A]): Boolean =
```

This page intentionally left blank.



## Exercise 4: Type Checking and Inference (20 points)

Consider the following typing rules for a simple language with integers, pairs and functions:

$$\frac{n \text{ is an integer literal}}{\Gamma \vdash n : \mathbf{Int}}$$

$$\frac{\Gamma \vdash e_1 : \mathbf{Int} \quad \Gamma \vdash e_2 : \mathbf{Int}}{\Gamma \vdash e_1 + e_2 : \mathbf{Int}}$$

$$\frac{\Gamma \vdash e_1 : \mathbf{Int} \quad \Gamma \vdash e_2 : \mathbf{Int}}{\Gamma \vdash e_1 * e_2 : \mathbf{Int}}$$

$$\frac{\Gamma \vdash e_1 : T_1 \quad \Gamma \vdash e_2 : T_2}{\Gamma \vdash (e_1, e_2) : (T_1, T_2)}$$

$$\frac{\Gamma \vdash e : (T_1, T_2)}{\Gamma \vdash \text{fst}(e) : T_1}$$

$$\frac{\Gamma \vdash e : (T_1, T_2)}{\Gamma \vdash \text{snd}(e) : T_2}$$

$$\frac{\Gamma \oplus \{(x, T_1)\} \vdash e : T_2}{\Gamma \vdash x \Rightarrow e : T_1 \Rightarrow T_2}$$

$$\frac{\Gamma \vdash e_1 : T_1 \Rightarrow T_2 \quad \Gamma \vdash e_2 : T_1}{\Gamma \vdash e_1(e_2) : T_2}$$

$$\frac{(x, T) \in \Gamma}{\Gamma \vdash x : T}$$

### Question 4.1

Consider the following type derivation, with type variables  $\mathbf{T}_1, \dots, \mathbf{T}_5$ , where  $\Gamma_0 = \emptyset$  and  $\Gamma = \{(x, \mathbf{T}_2)\}$ :

$$\frac{\frac{\frac{(x, \mathbf{T}_2) \in \Gamma}{\Gamma \vdash x : \mathbf{T}_2}}{\Gamma \vdash \text{fst}(x) : \mathbf{T}_4} \quad \frac{\frac{(x, \mathbf{T}_2) \in \Gamma}{\Gamma \vdash x : \mathbf{T}_2}}{\Gamma \vdash \text{snd}(x) : \mathbf{T}_5}}{\Gamma \vdash \text{fst}(x)(\text{snd}(x)) : \mathbf{T}_3} \quad \frac{}{\Gamma_0 \vdash x \Rightarrow \text{fst}(x)(\text{snd}(x)) : \mathbf{T}_1}$$

Circle all the correct answers:

- A. There are no assignments of  $\mathbf{T}_1, \dots, \mathbf{T}_5$  such that the resulting derivation is valid.
- B. In all valid derivations,  $\mathbf{T}_3$  is equal to  $\mathbf{T}_5$ .
- C. There does **not** exist valid derivations where  $\mathbf{T}_1$  is  $\mathbf{Int}$ .
- D. In all valid derivations,  $\mathbf{T}_2$  is equal to  $(\mathbf{T}_4, \mathbf{T}_5)$ .
- E. In all valid derivations,  $\mathbf{T}_3$  is equal to  $\mathbf{T}_2 \Rightarrow \mathbf{T}_1$ .

### Question 4.2

For which of the following expressions does type inference using unification succeed? Circle the correct answers.

- A.  $y \Rightarrow (x \Rightarrow (x, y))$
- B.  $x \Rightarrow (y \Rightarrow (x(y) + y(x)))$
- C.  $f \Rightarrow (x \Rightarrow f(f(x)))$
- D.  $f \Rightarrow (f(x \Rightarrow 4) + f(5))$

This page intentionally left blank.

## Exercise 5: Designing a Type System (20 points)

Consider an expression language with a halving operator on even numbers. We are designing an operational semantics and a type system that ensures that we never half an odd number.

$\langle Expr \rangle ::= \text{half}(\langle Expr \rangle) \mid \langle Expr \rangle + \langle Expr \rangle \mid \text{INTEGER}$

The values of our language are all integers. We denote values by  $n$  and  $k$ .

### Question 5.1

In this first question, we will design the operational semantics of our language. Semantics should define rule that apply to as many expressions as possible subject to the following constraints:

- Our operational semantics should not permit halving unless the value of an integer constant is even
- It should only perform evaluation of operands from left to right

Circle a **minimal** set of operational semantics rules that describe this behavior.

$$\frac{e \rightsquigarrow e'}{\text{half}(e) \rightsquigarrow e'} \quad (\text{A})$$

$$\frac{n \text{ is a value} \quad n = 2k}{\text{half}(n) \rightsquigarrow k} \quad (\text{B})$$

$$\frac{n \text{ is a value}}{\text{half}(n) \rightsquigarrow \lfloor \frac{n}{2} \rfloor} \quad (\text{C})$$

$$\frac{\text{half}(e) \rightsquigarrow \text{half}(e')}{\text{half}(e) \rightsquigarrow e'} \quad (\text{D})$$

$$\frac{e \rightsquigarrow e'}{\text{half}(e) \rightsquigarrow \text{half}(e')} \quad (\text{E})$$

$$\frac{e' \rightsquigarrow \text{half}(e)}{\text{half}(e) \rightsquigarrow e'} \quad (\text{F})$$

$$\frac{n_1 \text{ is a value} \quad n_2 \text{ is a value} \quad k = n_1 + n_2 \quad n_1 \text{ is odd} \quad n_2 \text{ is odd}}{n_1 + n_2 \rightsquigarrow k} \quad (\text{G})$$

$$\frac{e \rightsquigarrow e' \quad n \text{ is a value}}{n + e \rightsquigarrow n + e'} \quad (\text{H})$$

$$\frac{e_2 \rightsquigarrow e'_2}{e_1 + e_2 \rightsquigarrow e_1 + e'_2} \quad (\text{I})$$

$$\frac{n_1 \text{ is a value} \quad n_2 \text{ is a value} \quad k = n_1 + n_2 \quad n_1 \text{ is even} \quad n_2 \text{ is even}}{n_1 + n_2 \rightsquigarrow k} \quad (\text{J})$$

$$\frac{n_1 \text{ is a value} \quad n_2 \text{ is a value} \quad k = n_1 + n_2}{n_1 + n_2 \rightsquigarrow k} \quad (\text{K})$$

$$\frac{e_1 \rightsquigarrow e'_1}{e_1 + e_2 \rightsquigarrow e'_1 + e_2} \quad (\text{L})$$

## Question 5.2

In this second part, we will design a type system for our language. The following expressions should type check:

$4 + 5$   
 $\text{half}(2 + 4)$   
 $\text{half}(2) + 2$   
 $\text{half}(\text{half}(2) + \text{half}(2))$

Circle a subset of the following rules that form a *sound* type system for our language: if a program type checks, then it must evaluate to a constant using the rules of operational semantics in the previous part.

Make sure that the above expressions can be typed. Do not include rules that are redundant with other rules you circled: if removing a rule does not decrease the set of programs that type check, then remove the rule.

$$\frac{\Gamma \vdash n : \text{Even}}{\Gamma \vdash \text{half}(n) : \text{Even}} \quad (\text{A})$$

$$\frac{\Gamma \vdash n : \text{Integer}}{\Gamma \vdash \text{half}(n) : \text{Integer}} \quad (\text{B})$$

$$\frac{\Gamma \vdash n : \text{Even}}{\Gamma \vdash \text{half}(n) : \text{Integer}} \quad (\text{C})$$

$$\frac{n \text{ is an integer literal} \quad n \text{ is even}}{\Gamma \vdash n : \text{Even}} \quad (\text{D})$$

$$\frac{n \text{ is an integer literal} \quad n \text{ is odd}}{\Gamma \vdash n : \text{Integer}} \quad (\text{E})$$

$$\frac{\Gamma \vdash n : \text{Even}}{\Gamma \vdash n : \text{Integer}} \quad (\text{F})$$

$$\frac{\Gamma \vdash n : \text{Integer}}{\Gamma \vdash n : \text{Even}} \quad (\text{G})$$

$$\frac{}{\Gamma \vdash e_1 + e_2 : \text{Integer}} \quad (\text{H})$$

$$\frac{\Gamma \vdash e_1 : \text{Integer} \quad \Gamma \vdash e_2 : \text{Integer}}{\Gamma \vdash e_1 + e_2 : \text{Integer}} \quad (\text{I})$$

$$\frac{\Gamma \vdash e_1 : \text{Integer} \quad \Gamma \vdash e_2 : \text{Even}}{\Gamma \vdash e_1 + e_2 : \text{Even}} \quad (\text{J})$$

$$\frac{\Gamma \vdash e_1 : \text{Integer} \quad \Gamma \vdash e_2 : \text{Even}}{\Gamma \vdash e_1 + e_2 : \text{Integer}} \quad (\text{K})$$

$$\frac{\Gamma \vdash e_1 : \text{Even} \quad \Gamma \vdash e_2 : \text{Even}}{\Gamma \vdash e_1 + e_2 : \text{Even}} \quad (\text{L})$$

$$\frac{\Gamma \vdash e_1 : \text{Integer} \quad \Gamma \vdash e_2 : \text{Integer}}{\Gamma \vdash e_1 + e_2 : \text{Even}} \quad (\text{M})$$

$$\frac{\Gamma \vdash e : \text{Integer}}{\Gamma \vdash e + e : \text{Even}} \quad (\text{N})$$