Banner number: Name:

Midterm Exam

CSCI 3136: Principles of Programming Languages

February 27, 2019

Question 1.1		Question 2.1		Question 3.1		$\sum$
Question 1.2		Question 2.2		Question 3.2		
$\sum$		$\sum$		$\sum$		

Instructions:

- Provide your answer in the box after each question. If you absolutely need extra space, use the backs of the pages; but try to avoid it. Keep your answers short and to the point.
- You are not allowed to use a cheat sheet.
- Make sure your answers are clear and legible. If I can't decipher an answer or follow your train of thought with reasonable effort, you'll receive 0 marks for your answer.
- Read every question carefully before answering.
- Do not forget to write your banner number and name on the top of this page.
- This exam has 7 pages, including this title page. Notify me immediately if your copy has fewer than 7 pages.

1 Program translation and regular languages

Question 1.1 **10 marks**

Recall the general workflow for compiling a program to machine code. The front-end creates an intermediate representation of the program from the source code using lexical analysis, syntactic analysis, and semantic analysis. The back-end optimizes the intermediate representation and translates it into executable machine code.

(a) What is the advantage of splitting the translation process into front-end and back-end?

The analysis of the source code done by the front-end is independent of the target architecture for which the code is to be compiled whereas the code generation and optimization steps performed by the back-end are fairly independent of the programming language. The split into separate phases allows us to mix and match front-ends and back-ends to obtain compilers for different programming languages on different architectures without significant duplication of effort.

(b) Why do many compilers implement lexical analysis and syntactic analysis of the program text as separate phases of the front-end?

Syntactic analysis is more costly than identifying lexical units. Thus, we want to perform it on as compact a representation of the code as can be produced cheaply from the program text. A stream of lexical tokens is such a representation. It is much more compact than a character stream and, using a DFA to implement the lexical analysis, it is extremely efficient to carry out.

Question 1.2 **10 marks**

(a) Formally define what a formal language is.

A formal language is a set of strings over a given alphabet.

(b) Formally state what structural conditions a language $\mathcal{L}$ must satisfy to be regular. (I am looking for a structural definition, not for a "definition by proxy" that states that a language is regular if it can be described by a regular expression or decided by a DFA or NFA.)

A language $\mathcal{L}$ is regular if it satisfies one of the following conditions:

- *It is the empty language.*
- *It contains exactly one string: the empty string ε.*
- *It contains exactly one string σ and this string has a single character from the underlying alphabet.*
- *It is the union of two regular languages $\mathcal{L}_1$ and $\mathcal{L}_2$: $\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2$.*
- *It is the concatenation of two regular languages $\mathcal{L}_1$ and $\mathcal{L}_2$: $\mathcal{L} = \{\sigma_1\sigma_2 \mid \sigma_1 \in \mathcal{L}_1, \sigma_2 \in \mathcal{L}_2\}$.*
- *It is the Kleene star of a regular language $\mathcal{L}_1$: $\mathcal{L} = \mathcal{L}_1^0 \cup \mathcal{L}_1^1 \cup \cdots$, where $\mathcal{L}_1^i = \{\sigma_1 \cdots \sigma_i \mid \sigma_1, \ldots, \sigma_i \in \mathcal{L}_1\}$.*

2 Scheme and Prolog

Question 2.1 **10 marks**

(a) We discussed in class that purely functional languages cannot support loops, yet the following Scheme code looks rather loop-like.

```
(define (fibonacci n)
  (let loop ([x 0] [y 1] [i 0])
    (if (= i n)
      y
      (loop y (+ x y) (+ i 1)))))
```

Is this actually a loop? If so, explain why the code is nevertheless purely functional. If not, provide equivalent code into which the Scheme interpreter translates this function and which demonstrates that this is not in fact a loop.

```
(define (fibonacci n)
  (define (loop x y z)
    (if (= i n)
      y
      (loop y (+ x y) (+ i 1))))
  (loop 0 1 0))
```

(b) The following is a simple implementation of Scheme's `map` function.

```
(define (map f lst)
  (if (null? lst)
      '()
      (cons (f (car lst)) (map f (cdr lst)))))
```

Provide an implementation of this function that uses constant stack space and runs in linear time.

```
(define (map f lst)
  (define go (in out)
    (if (null? in)
      (reverse out)
      (go (cdr in) (cons (f (car in)) out))))
  (go lst '()))
```

Question 2.2 **10 marks**

Consider the following Prolog program:

```
p(X,Y) :- q(X,Y).
p(3,6).

q(X,Y) :- a(X), b(Y).
q(4,7).

a(1). a(2).
b(4). b(5).
```

(a) Provide the output of the query `?- findall((X,Y), p(X,Y), List).`

```
List = [(1,4), (1,5), (2,4), (2,5), (4,7), (3,6)].
```

(b) Provide the output of the same query after changing the definition of q/2 to

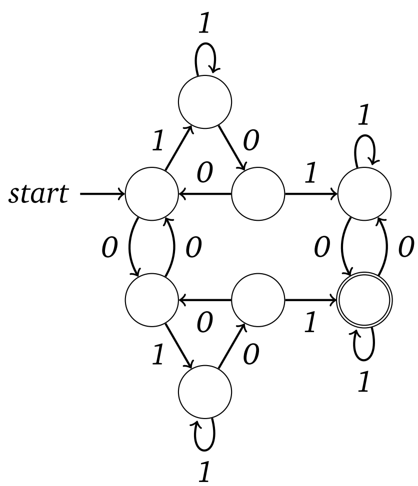
```
q(X,Y) :- a(X), !, b(Y).
q(4,7).
```

```
List = [(1,4), (1,5), (3,6)].
```

3 Finite automata and regular expressions

Question 3.1 **10 marks**

Provide a DFA that decides the language of all binary strings that contain an even number of 0s and contain the pattern 101. Provide the DFA in graphical form.

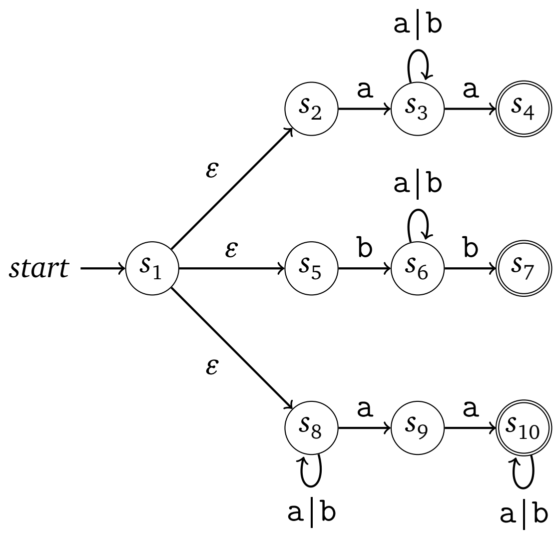


Question 3.2 **10 marks**

Provide a regular expression that describes the language of all strings over the alphabet $\{\texttt{a}, \texttt{b}\}$ that have the same first and last letter or contain two consecutive as. Also provide an NFA that decides this language and convert it to a DFA. Provide the NFA in graphical form and the DFA in tabular form.

Regular expression: `a.*a|b.*b|.*aa.*`

NFA:



DFA:

State	a	b
$\rightarrow \{s_1, s_2, s_5, s_8\}$	$\{s_3, s_8, s_9\}$	$\{s_6, s_8\}$
$\{s_3, s_8, s_9\}$	$\{s_3, s_4, s_8, s_9, s_{10}\}$	$\{s_3, s_8\}$
$\{s_6, s_8\}$	$\{s_6, s_8, s_9\}$	$\{s_6, s_7, s_8\}$
$* \{s_3, s_4, s_8, s_9, s_{10}\}$	$\{s_3, s_4, s_8, s_9, s_{10}\}$	$\{s_3, s_8, s_{10}\}$
$\{s_3, s_8\}$	$\{s_3, s_4, s_8, s_9\}$	$\{s_3, s_8\}$
$\{s_6, s_8, s_9\}$	$\{s_6, s_8, s_9, s_{10}\}$	$\{s_6, s_7, s_8\}$
$* \{s_6, s_7, s_8\}$	$\{s_6, s_8, s_9\}$	$\{s_6, s_7, s_8\}$
$* \{s_3, s_8, s_{10}\}$	$\{s_3, s_4, s_8, s_9, s_{10}\}$	$\{s_3, s_8, s_{10}\}$
$* \{s_3, s_4, s_8, s_9\}$	$\{s_3, s_4, s_8, s_9, s_{10}\}$	$\{s_3, s_8\}$
$* \{s_6, s_8, s_9, s_{10}\}$	$\{s_6, s_8, s_9, s_{10}\}$	$\{s_6, s_7, s_8, s_{10}\}$
$* \{s_6, s_7, s_8, s_{10}\}$	$\{s_6, s_8, s_9, s_{10}\}$	$\{s_6, s_7, s_8, s_{10}\}$