

**Terms**

syntax  
semantics  
language  
lexeme

tokens  
sentence  
terminal  
non-terminal

inherited  
synthesized  
more...

**Topics**

syntax fit a Grammar?	Derivations	Parsing
Solving for Ambiguity	Left/Right recursion	<del>BNF</del> to EBNF
<del>Attribute Grammars</del>	<del>Turing Machine</del>	Logic
<del>Weakest Precondition</del>	Shift Reduce	Parse Trees
	Regular Expressions (Lexical and Syntax Analysis) <a href="https://regexcrossword.com/">https://regexcrossword.com/</a> (is great practice!!)	RE to DFSM
DFA to RegEx (or vise versa)	Changing precedence	Transition Tables
Scheme		

No notes, cheat sheets, cheating, etc...

2 days

Day 1 → theory

Day 2 → code

Day 1	Day 2
(on Paper)	(on ECampus)
mostly Theory	mostly Coding
Ambiguity	Changing Precedence
Parse Table	Derivations
Shift Reduce	Scheme
DFSM	Reg. Expressions
parse Trees	Allowed Dr. Racket/Compute
Grammar basics	

**What normally hurts grades**

You don't have homework!!

Don't study at all, take is light hearted

Topics that people lose the most points: DFA to RegEx, Shift Reduce, Left/Right recursion

Day 2: coding

**Scheme**

Built in: equal?, number?, integer?, string?, boolean?

LISTS: (length list) (null? list) (remove element '(list))

Modulo == % (modulo 5 2) = 1

Append( (value) (list (vals in list))) -> backwards like given 5 and then recursively calling rest of list will bring 0 1 2 3 4


Vs  
Cons-> (cons (element) (list (vals)))

## Precedence

NOTES: Lexical Analysis, Grammars and Parse Trees

## Derivations

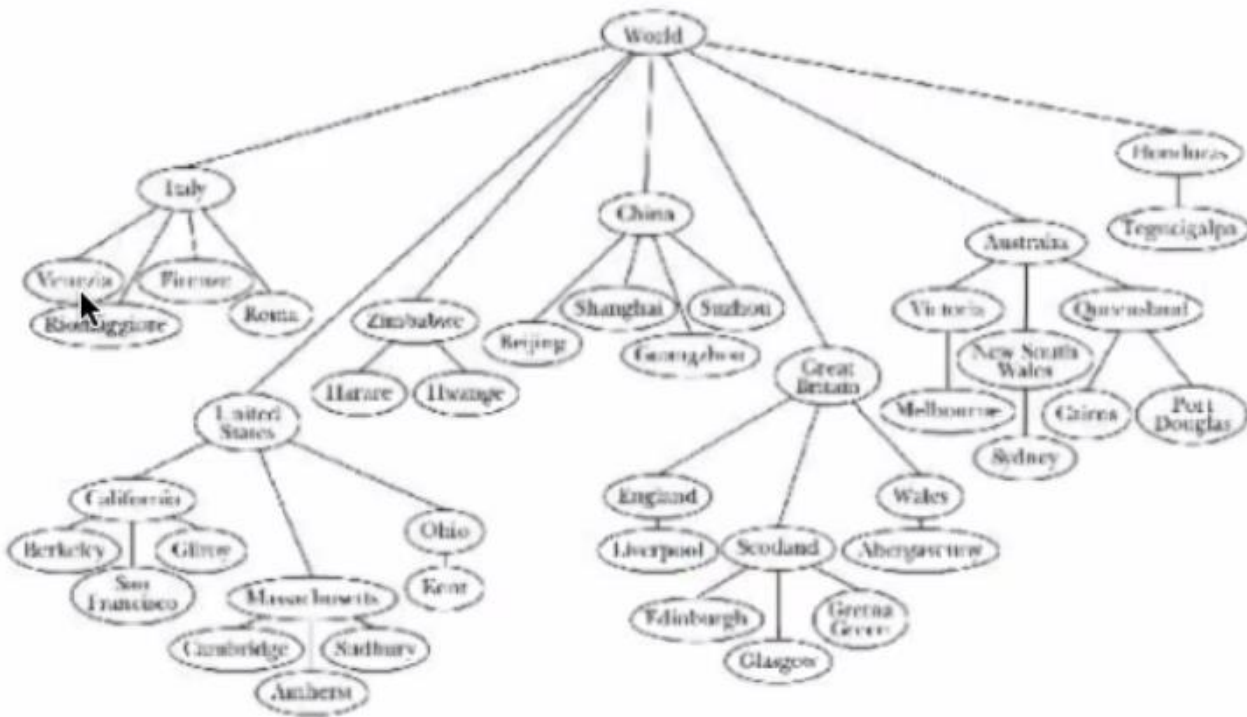
NOTES: Lexical Analysis

Derivation for A = B + C * A	
<pre>&lt;assign&gt; → &lt;id&gt; = &lt;expr&gt; &lt;expr&gt; → &lt;expr&gt; + &lt;term&gt;             &lt;term&gt; &lt;term&gt; → &lt;term&gt; * &lt;factor&gt;             &lt;factor&gt; &lt;factor&gt; → ( &lt;expr&gt; )             &lt;id&gt; &lt;id&gt; → A   B   C</pre>	
<p>(leftmost)</p> <pre>&lt;assign&gt; =&gt; &lt;id&gt; = &lt;expr&gt; =&gt; A = &lt;expr&gt; =&gt; A = &lt;expr&gt; + &lt;term&gt; =&gt; A = &lt;term&gt; + &lt;term&gt; =&gt; A = &lt;factor&gt; + &lt;term&gt; =&gt; A = &lt;id&gt; + &lt;term&gt; =&gt; A = B + &lt;term&gt; =&gt; A = B + &lt;term&gt; * &lt;factor&gt; =&gt; A = B + &lt;factor&gt; * &lt;factor&gt; =&gt; A = B + &lt;id&gt; * &lt;factor&gt; =&gt; A = B + C * &lt;factor&gt; =&gt; A = B + C * &lt;id&gt; =&gt; A = B + C * A</pre>	<p>(rightmost)</p> <pre>&lt;assign&gt; =&gt; &lt;id&gt; = &lt;expr&gt; =&gt; &lt;id&gt; = &lt;expr&gt; + &lt;term&gt; =&gt; &lt;id&gt; = &lt;expr&gt; + &lt;term&gt; * &lt;factor&gt; =&gt; &lt;id&gt; = &lt;expr&gt; + &lt;term&gt; * &lt;id&gt; =&gt; &lt;id&gt; = &lt;expr&gt; + &lt;term&gt; * A =&gt; &lt;id&gt; = &lt;expr&gt; + &lt;factor&gt; * A =&gt; &lt;id&gt; = &lt;expr&gt; + &lt;id&gt; * A =&gt; &lt;id&gt; = &lt;expr&gt; + C * A =&gt; &lt;id&gt; = &lt;term&gt; + C * A =&gt; &lt;id&gt; = &lt;factor&gt; + C * A =&gt; &lt;id&gt; = &lt;id&gt; + C * A =&gt; &lt;id&gt; = B + C * A =&gt; A = B + C * A</pre>

Left derivation:

Given Grammar	Leftmost Derivation w/ Target Syntax
<pre>&lt;sentence&gt; -&gt; &lt;subject&gt; &lt;predicate&gt; &lt;subject&gt; -&gt; &lt;article&gt; &lt;noun&gt; &lt;predicate&gt; -&gt; &lt;verb&gt; &lt;direct-object&gt; &lt;direct-object&gt; -&gt; &lt;article&gt; &lt;noun&gt; &lt;article&gt; -&gt; THE   A &lt;noun&gt; -&gt; MAN   DOG &lt;verb&gt; -&gt; BITES   PETS</pre>	<pre>A·DOG·PETS·A·DOG ¶ &lt;sentence&gt; -&gt; ¶ Answer: ¶ =&gt; &lt;subject&gt; &lt;predicate&gt; ¶ =&gt; &lt;article&gt; &lt;noun&gt; &lt;predicate&gt; ¶ =&gt; A &lt;noun&gt; &lt;predicate&gt; ¶ =&gt; A DOG &lt;predicate&gt; ¶ =&gt; A DOG &lt;verb&gt; &lt;direct-object&gt; ¶ =&gt; A DOG PETS &lt;direct-object&gt; ¶ =&gt; A DOG PETS &lt;article&gt; &lt;noun&gt; ¶ =&gt; A DOG PETS A &lt;noun&gt; ¶ =&gt; A DOG PETS A DOG</pre>

Lab 5: Trees ADT (L) Missing lab



For this problem:

1. Download the trees2.scm code by:
  - a. (in UNIX)  
`wget http://faculty.cse.tamu.edu/slupoli/notes/Scheme/code/trees/trees2.txt`
  - b. (in Windows)  
<http://faculty.cse.tamu.edu/slupoli/notes/Scheme/code/trees/trees2.txt>

Use the code and pic above for the tree.

## The old ways are not always the best

1. Using only the `cdr`/`car` function variations, display `"scotland"`.

Please have a Dr Racket window open when presenting your answer.

2. Write a function count-nodes to return the number of non leaf nodes in the world tree. Result should be 17.

```
(display (car (caddr (car (cddddr (children world-tree2)))))) ; return scotland
(newline)
```

Q2

### Rightmost element in the tree

3. Write a function `rightmost` to return the rightmost element in the tree. Display the results.

`(rightmost world-tree2)` → `tegucigalpa`

`(rightmost (car (cdr world-tree2)))` → `roma`

Q3 function `rightmost` rightmost element in tree

```
(define (lastEl l)
  (cond ((null? (cdr l)) (car l))
        (else (lastEl (cdr l)))))

(define (rightmostelement node)
  (cond ((leaf? node) node)
        (else (rightmostelement (lastEl (children node) ))))
  )

(display (rightmostelement world-tree2))
(newline)
```

Welcome to [DrRacket](#), version 7.8 [3m].  
Language: `racket`, with `debugging`; memory limit: 128 MB.  
`(tegucigalpa)`

### Display leaf nodes

4. Write a function `leafDisplay` to return a **flattened** list of all the leaf nodes of a tree. Display the results for `world-tree2`

Q4 flattened list `leafDisplay`

```

(define (my-flatten lst)
  (cond ((null? lst) '())
        ((pair? lst)
         (append (my-flatten (car lst)) (my-flatten (cdr lst))))
        (else (list lst))))

(display (my-flatten world-tree2))
(newline)

```

### Display leaf nodes

- Write `replace`, a procedure that takes `city1`, `city2` and a tree as argument and returns a copy of the tree, with `city2` replacing `city1`. `city1` will be present in the `world-tree2`.

For example:

```
(display (replace 'italy 'hello world-tree2))
```

## Lab 3: Lists and Recursions

### QUESTION 1

10

- Create the file `SchemeFlipped1.scm`. Inside that file, create a method `listmaker`. This function should take an integer argument  $n$  and return a list with values ranging from 0 to  $n-1$ . If zero is entered, it should return an empty list. Use recursion.

```
(listmaker 5) -> (0 1 2 3 4)
```

Please have TWO Putty windows open when presenting your answer. One with the code and the second with the code running.

Instructor/TA signature

Q1

```
ssh tatiaris@compute.cs.tamu.edu
File Edit Options Buffers Tools Scheme Help
:=====
: File: SchemeFlipped1.scm
: Written by: Sunhee Kim and Rishabh Tatia
: Date: 9/12/20
: TAMU email: sunheek@tamu.edu tatiaris@tamu.edu
: Class: CSCE 314 (section 502)
: Description: This is the first program for lab 3
:=====

: given an int input n, returns a list of values ranging from 0 -> \
n-1
(define (listmaker n)
  (cond
    ((= n 0) '())
    (else (append (listmaker (- n 1)) (list (- n 1)))))
  )
)

(display (listmaker 5))
(println)
(display (listmaker 0))

-uu-:----F1 SchemeFlipped1.scm All L19 (Scheme) -----
Wrote /home/ugrads/t/tatiaris/csce_314/lab3/SchemeFlipped1.scm

ssh tatiaris@compute.cs.tamu.edu

[tatiaris]@compute ~/csce_314/lab3> (18:45:49 09/12/20)
:: gsi SchemeFlipped1.scm
(0 1 2 3 4)
()
[tatiaris]@compute ~/csce_314/lab3> (18:45:55 09/12/20)
::
```

2. Create the file SchemeFlipped2.scm using Emacs. Define a function `alternate first` that returns the alternate elements from a simple list, starting from the first element.

**alternate-first:** Returns the elements of the list in an alternate fashion from the first element onwards.  
For example  
(`alternate-first` '(2 3 4 5 6 9 11)) -> (2 4 6 11)

Q2



```
ssh tatiaris@compute.cs.tamu.edu
File Edit Options Buffers Tools Scheme Help
;=====
; File: SchemeFlipped2.scm
; Written by: Sunhee Kim and Rishabh Tatia
; Date: 9/12/20
; TAMU email: sunheek@tamu.edu tatiaris@tamu.edu
; Class: CSCE 314 (section 502)
; Description: This is the second program for lab 3
;=====

; given an list input l, returns a list of the alternating values in l
(define (alternate-first l)
  (cond
    ((< (length l) 3) (list (car l)))
    (else (cons (car l) (alternate-first (cddr l)))))
  )

(display (alternate-first '(2 3 4 5 6 9 11)))
(println)
(display (alternate-first '(2)))

-uuu:---F1 SchemeFlipped2.scm All L20 (Scheme) -----
Wrote /home/ugrads/t/tatiaris/csce_314/lab3/SchemeFlipped2.scm

ssh tatiaris@compute.cs.tamu.edu

[tatiaris]@compute ~/csce_314/lab3> (19:01:42 09/12/20)
:: gsi SchemeFlipped2.scm
(2 4 6 11)
(2)
[tatiaris]@compute ~/csce_314/lab3> (19:02:59 09/12/20)
:: □
```

3. Create the file SchemeFlipped3.scm using Emacs. Define a function alternate-second that returns the elements of the list in an alternate fashion from the second element onwards. This function should handle empty lists and should flatten nested lists.

(alternate-second (flatten '(1 (2 (7) 4) () 3 5))) -> (2 4 5)

Q3

```
ssh: tatiaris@compute.cs.tamu.edu
File Edit Options Buffers Tools Scheme Help
: File: SchemeFlipped3.scm
: Written by: Sunhee Kim and Rishabh Tatia
: Date: 9/12/20
: TAMU email: sunheek@tamu.edu tatiaris@tamu.edu
: Class: CSCE 314 (section 502)
: Description: This is the third program for lab 3
: =====

(define (flatten list)
  (cond ((null? list) '())
        ((list? (car list)) (append (flatten (car list)) (flatten (cdr list))))
        (else (cons (car list) (flatten (cdr list))))))

; given an list input l, returns a list of the alternating values in l \
; starting from the second value
(define (alternate-second l)
  (cond ((< (length l) 2) '())
        ((< (length l) 3) (list (cadr l)))
        (else (cons (cadr l) (alternate-second (cddr l))))))

(display (alternate-second (flatten '(1 (2 (7) 4) () 3 5))))
(println)
(display (alternate-second (flatten '(1 (2)))))
(println)
(display (alternate-second (flatten '(1))))

[tatiaris]@compute ~/csce_314/lab3> (19:19:32 09/12/20)
:: gsi SchemeFlipped3.scm
(2 4 5)
(2)
()
[tatiaris]@compute ~/csce_314/lab3> (19:19:33 09/12/20)
::
```

4. Create the file SchemeFlipped4.scm. Write a function that merges two lists that are sorted in increasing order while maintaining the sort. If the values are the same, put the value from the left list first.

Example:

(mergelist '(1 5 9 13 72) '(2 4 7 8 9)) -> (1 2 4 5 7 8 9 9 13 72)

Please have two Putty windows open when presenting your answer. One with the code and the second with the code running.

Q4 mergelist



```

File Edit Options Buffers Tools Scheme Help
:
: File: SchemeFlipped4.scm
: Written by: Sunhee Kim and Rishabh Tatia
: Date: 9/12/20
: TAMU email: sunheek@tamu.edu tatiaris@tamu.edu
: Class: CSCE 314 (section 502)
: Description: This is the fourth program for lab 3
:
:
: given 2 sorted lists l1 and l2, merges the two into one list
(define (mergelist l1 l2)
  (cond
    ((and (null? l1) (null? l2)) '())
    ((null? l1) l2)
    ((null? l2) l1)
    (else
     (cond
      ((<= (car l1) (car l2)) (cons (car l1) (mergelist (cdr l1) l2)))
      (else (cons (car l2) (mergelist l1 (cdr l2)))))
     )
    )
  )
)
)
[
(display (mergelist '(1 5 9 13 72) '(2 4 7 8 9)))
(println)
(display (mergelist '() '(2 4 7 8 9)))
(println)
(display (mergelist '(1 5 9 13 72) '()))
(println)
(display (mergelist '(2 3 4 5) '(4 5 6 7)))

-uuu:----F1 SchemeFlipped4.scm All L24 (Scheme) -----
Wrote /home/ugrads/t/tatiaris/csce_314/lab3/SchemeFlipped4.scm

ssh tatiaris@compute.cs.tamu.edu

[tatiaris]@compute ~/csce_314/lab3> (19:20:40 09/12/20)
:: gsi SchemeFlipped4.scm
(1 2 4 5 7 8 9 9 13 72)
(2 4 7 8 9)
(1 5 9 13 72)
(2 3 4 4 5 5 6 7)
[tatiaris]@compute ~/csce_314/lab3> (19:36:18 09/12/20)
::

```

5. Create the file SchemeFlipped5.scm. Inside that file, Write the function "intersection" that returns the intersection of two simple list parameters that represent sets. There should be no duplicates in the intersected list. Order does not matter. Don't forget to test if the list(s) are empty. We want TWO examples with TWO different results. **You are NOT allowed to use Scheme's built in "remove-duplicates" function or any of the "member" or "memv" functions.**

```
(intersection '(1 2 3 4) '(4 -1 2 5)) => '(2 4); no duplicates!! (BTW, order does not matter)
```

```
(intersection '(5 6 7 8) '(1 2 3)) => '()
```

Q5 intersection of two lists

```
ssh tatiaris@compute.cs.tamu.edu
File Edit Options Buffers Tools Scheme Help
=====
: File: SchemeFlipped5.scm
: Written by: Sunhee Kim and Rishabh Tatia
: Date: 9/12/20
: TAMU email: sunheek@tamu.edu tatiaris@tamu.edu
: Class: CSCE 314 (section 502)
: Description: This is the fifth program for lab 3
=====

; returns whether atm exists in the list l
(define (member atm l)
  (cond
    ((null? l) #f)
    ((eq? atm (car l)) #t)
    (else (member atm (cdr l)))
  )
)

; given a list l, returns a new list without duplicates
(define (remove_duplicates l)
  (cond
    ((null? l) '())
    ((member (car l) (cdr l)) (remove_duplicates (cdr l)))
    (else (cons (car l) (remove_duplicates (cdr l))))
  )
)

; given 2 lists l1 and l2, returns the intersection of the two
(define (intersection l1 l2)
  (cond
    ((or (null? l1) (null? l2)) '())
    (else
      (cond
        ((member (car l1) l2)
         (remove_duplicates (cons (car l1) (intersection (cdr l1) l2))))
        (else (remove_duplicates (intersection (cdr l1) l2)))
      )
    )
  )
)

(display (intersection '(1 2 3 4) '(4 -1 2 5)))
(println)
(display (intersection '(5 6 7 8) '(1 2 3)))
(println)
(display (intersection '() '(2 4 7 8 9)))
(println)
(display (intersection '(1 5 9 13 72) '()))
(println)
(display (intersection '(2 3 4 5 7 7 9 9 10) '(4 5 9 3 10 6 7)))

[tatiaris]@compute ~/csce_314/lab3> (20:08:06 09/12/20)
:: gsi SchemeFlipped5.scm
(2 4)
()
()
(3 4 5 7 9 10)
[tatiaris]@compute ~/csce_314/lab3> (20:08:13 09/12/20)
:: []
```

## Lab 2: Types and Scheme Conditions

1. Create the file SchemeFlipped1.scm using Emacs. Inside that file, create a "max2diff" method which takes three parameters, that will choose the largest 2 numbers out of 3 different numbers and subtract the second largest from the largest and return the result. To complete the process, have a `main()` to call the function. Indent your code!! Treat the `[js` as `]s`!! Once complete, go to the scheme interpreter to run the file you just created.

Please have TWO Putty windows open when presenting your answer. One window with the code and the second with the code running.

Q1 "max2diff" choose largest 2 numbers out of 3 and subtract the second largest from the largest

```
[tatiaris]@compute ~/csce_314/lab2> (19:21:16 09/04/20)
:: gsi SchemeFlipped5.scm
Glen
5/2
5000
10

[tatiaris]@compute ~/csce_314/lab2> (19:21:18 09/04/20)
:: 

File Edit Options Buffers Tools Scheme Help
: File: SchemeFlipped1
: Written by: Sunhee Kim and Rishabh Tatia
: Date: 09/04/20
: TAMU email: sunheek@tamu.edu, tatiaris@tamu.edu
: Class: CSCE 314 (section 502)
: Description: This is the first program for lab 2
:
; takes 3 number parameters and returns the difference between the largest 2 numbers
(define (max2diff x y z)
  (if (and (< x y) (< x z))
      (if (> (- z y) 0) (- z y)
          (- y z))
      (if (and (< y x) (< y z))
          (if (> (- z x) 0) (- z x)
              (- x z))
          (if (> (- x y) 0) (- x y)
              (- y x))
      )
  )
)

(define (main)
  ; testing different test cases
  (println (max2diff 3 6 10))
  (println (max2diff 6 3 10))
  (println (max2diff 10 6 3))
)

;call to main function
(main)
```

2. Create the file SchemeFlipped2.scm. Inside that file, create a method named "is-negative". The overall concept of the function is to return TRUE (#t) if a negative number (0 is considered positive) passed in, or not. The function needs to contain either an "if" or "cond" branch structure. We want THREE separate calls with -100, 0 and 100, to make sure it works.

Please have TWO Putty windows open when presenting your answer. One with the code and the second with the code running.

3. Create the file SchemeFlipped3.scm. Write a function that determines if a year is a leap year. Remember that a leap year is any year divisible by 4, except those divisible by 100. Except years divisible by 400 are leap years.

Please have two Putty windows open when presenting your answer. One with the code and the second with the code running.

4. Create the file SchemeFlipped4.scm using Emacs. Inside that file, create a "max2diff" method which takes three parameters, that will choose the largest 2 numbers out of 3 different numbers and subtract the second largest from the largest and return the result. To complete the process, have a (main) to call the function. Indent you code!! Treat the (is as (is!! Once complete, go to the scheme interpreter to run the file you just created.

```
ssh tataris@compute.cs.tamu.edu
[tataris]@compute ~/csce_314/lab2> (19:26:07 09/04/20)
:: gsi SchemeFlipped2.scm
#t
#f

[tataris]@compute ~/csce_314/lab2> (19:26:10 09/04/20)
::
```

```
File Edit Options Buffers Tools Scheme Help
: File: SchemeFlipped2
: Written by: Sunhee Kim and Rishabh Tatia
: Date: 09/04/20
: TAMU email: sunheek@tamu.edu, tataris@tamu.edu
: Class: CSCE 314 (section 502)
: Description: This is the second program for lab 2
:
: takes in a number parameter and returns whether the number is negative
(define (is-negative x)
  (if (= x 0) #f
      (if (< x 0) #t
          #f)
  )
)

(define (main)
  ; testing different cases
  (println (is-negative -100))
  (println (is-negative 0))
  (println (is-negative 100))
)

;call to main function
(main)
```

Q3

```
ssh tataris@compute.cs.tamu.edu
[tataris]@compute ~/csce_314/lab2> (19:27:47 09/04/20)
:: gsi SchemeFlipped3.scm
#t
#f
#t
#f

[tataris]@compute ~/csce_314/lab2> (19:27:53 09/04/20)
::
```

```
File Edit Options Buffers Tools Scheme Help
: File: SchemeFlipped3
: Written by: Sunhee Kim and Rishabh Tatia
: Date: 09/04/20
: TAMU email: sunheek@tamu.edu, tataris@tamu.edu
: Class: CSCE 314 (section 502)
: Description: This is the third program for lab 2
:
: takes in a number (year) as a parameter and returns whether it is a leap year
(define (isLeapYear y)
  (if (integer? (/ y 4))
      (if (integer? (/ y 400)) #t
          (if (integer? (/ y 100)) #f
              #t)
      )
      #f
  )
)

(define (main)
  ; testing different cases
  (println (isLeapYear 2000))
  (println (isLeapYear 2003))
  (println (isLeapYear 2012))
  (println (isLeapYear 2100))
)

;call to main function
(main)
```

Q4

```
[tatiaris@compute ~:/csce_314/lab2> (19:24:54 09/04/20)
:: gsi SchemeFlipped4.scm
4
4
4

[tatiaris@compute ~:/csce_314/lab2> (19:24:56 09/04/20)
::
```

```
File Edit Options Buffers Tools Scheme Help
: File: SchemeFlipped4
: Written by: Sunhee Kim and Rishabh Tatia
: Date: 09/04/20
: TAMU email: sunheek@tamu.edu, tatiaris@tamu.edu
: Class: CSCE 314 (section 502)
: Description: This is the fourth program for lab 2
:
: takes 3 number parameters and returns the difference between the largest 2 numbers
(define (max2diff x y z)
  (if (and (< x y) (< x z))
      (if (> (- z y) 0) (- z y)
          (- y z))
      (if (and (< y x) (< y z))
          (if (> (- z x) 0) (- z x)
              (- x z))
          (if (> (- x y) 0) (- x y)
              (- y x))
          )
      )
  )
)

(define (main)
  ; testing different cases
  (println (max2diff 3 6 10))
  (println (max2diff 6 3 10))
  (println (max2diff 10 6 3))
)

;call to main function
(main)
```

5. Create the file SchemeFlipped5.scm using Emacs. Write a function that takes four arguments. The first two arguments are functions; f1 and f2. The third and fourth arguments are values. If both values are equal, and both are equal to 1, then return f1(x) + f2(y). If either value is the letter "g", return "Glen". If x is greater than y, return f1(x) \* f2(y). If none of these conditions are true, return x \* y. YOU MAY ONLY USE THREE "if" keywords, and you may not use any COND statements.

Please have TWO Putty windows open when presenting your answer. One with the code and the second with the code running.

Q5

```
[tatiaris@compute ~:/csce_314/lab2> (19:21:16 09/04/20)
:: gsi SchemeFlipped5.scm
Glen
5/2
5000
10

[tatiaris@compute ~:/csce_314/lab2> (19:21:18 09/04/20)
::
```

```
File Edit Options Buffers Tools Scheme Help
: File: SchemeFlipped5
: Written by: Sunhee Kim and Rishabh Tatia
: Date: 09/04/20
: TAMU email: sunheek@tamu.edu, tatiaris@tamu.edu
: Class: CSCE 314 (section 502)
: Description: This is the fifth program for lab 2
:
: takes 4 arguments and evaluates the parameters to return a value
(define (problem5 f1 f2 x y)
  (if (or (equal? x "g") (equal? y "g")) "Glen"
      (if (= x y 1) (+ (f1 x) (f2 y))
          (if (> x y) (* (f1 x) (f2 y))
              (* x y))
      )
  )
)

; testing function 1 to be passed in as a parameter
(define (func1 n)
  (* n 2)
)

; testing function 2 to be passed in as a parameter
(define (func2 n)
  (/ n 2)
)

(define (main)
  ; testing each case
  (println (problem5 func1 func2 "g" 1))
  (println (problem5 func1 func2 1 1))
  (println (problem5 func1 func2 100 50))
  (println (problem5 func1 func2 2 5))
)

;call to main function
(main)
```

## Lab 4: Higher Order Functions



### Debugging in Dr. Racket

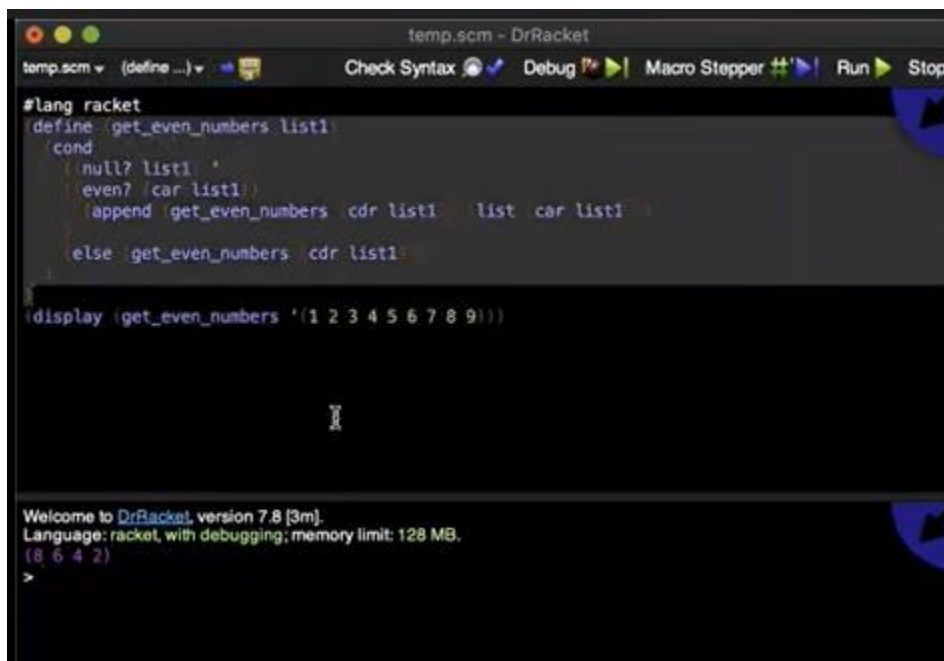
1. Consider the codes below for the function "get\_even\_numbers." This function, given a list of numbers, will return the even numbers in the list as a separate list. Type in the code and use tools in Dr. Racket to detect and debug any erroneous expression in the codes. **Show that your fixed code return correct results.**

```
(define (get_even_numbers list1 list2)
  (cond
    ((null? list1) list2)
    (else (even? (car list1)) (cons (car list1)
                                     (append_list (cdr list1) list2))))

(get_even_numbers '(1 2 3 4 5 6 7 8 9)) -> (8 6 4 2)
```

Debugging in Dr. Racket #2

Q1



The screenshot shows the Dr. Racket IDE interface. The top toolbar includes buttons for 'Check Syntax', 'Debug', 'Macro Stepper', 'Run', and 'Stop'. The code editor contains the following Racket code:

```
#lang racket
(define (get_even_numbers list1)
  (cond
    (null? list1) '
    (even? (car list1)
      (append (get_even_numbers (cdr list1)) list (car list1))
    )
    (else (get_even_numbers (cdr list1))
  )
)

(display (get_even_numbers '(1 2 3 4 5 6 7 8 9)))
```

The bottom status bar displays the message: "Welcome to DrRacket, version 7.8 [3m]. Language: racket, with debugging; memory limit: 128 MB." Below this, the output of the code is shown as "(8 6 4 2)" followed by a prompt ">".



## Debugging in Dr. Racket #2

2. Consider the codes for the function "performance\_review" below. Type in the code and use tools in Dr. Racket to detect and debug any erroneous expression in the code. Show that your fixed code returns correct results. (Tip: you can enter Debug mode and press Stop, in order to exit from runtime and reload the interactive mode of Dr. Racket)

```
(define (performance_review_score x)
  (cond
    [(< x 3) "Product could use some improvement"]
    [(and (< x 3) (>= x 0)) "Good Product"]
    [(and (< x 4) (>= x 3)) "Good Product"]
    [else "Excellent Product"]
  )
)

(define (main)
  (display (performance_review_score "0"))
  (newline)
  (display (performance_review_score "2.5"))
  (newline)
  (display (performance_review_score "5"))
  (newline)
)

(main)
```

Q2

```
#lang racket

(define (performance_review_score x)
  (cond
    [(and (string=? x "3") (string=? x "0")) "Product could use some improvement"]
    [(and (string=? x "4") (>= x "3")) "Good Product"]
    [else "Excellent Product"]
  )
)

(define (main)
  (display (performance_review_score "0"))
  (newline)
  (display (performance_review_score "2.5"))
  (newline)
  (display (performance_review_score "5"))
  (newline)
)

(main)
```

Welcome to DrRacket, version 7.8 [3m].  
Language: racket, with debugging; memory limit: 128 MB.  
Product could use some improvement  
Product could use some improvement  
Excellent Product  
>

3. Write a higher-order defined function **cube root** which applies a function to a list as values and produces a list results. The function **cube root** should apply the passed function to each element in the list and return a same-size list as a result. **You are NOT allowed to use the "filter" function.**

Example:

```
(cube root '(1 2 3)) ==>(1 1.25992105 1.44224957)
```

Instructor/TA signature

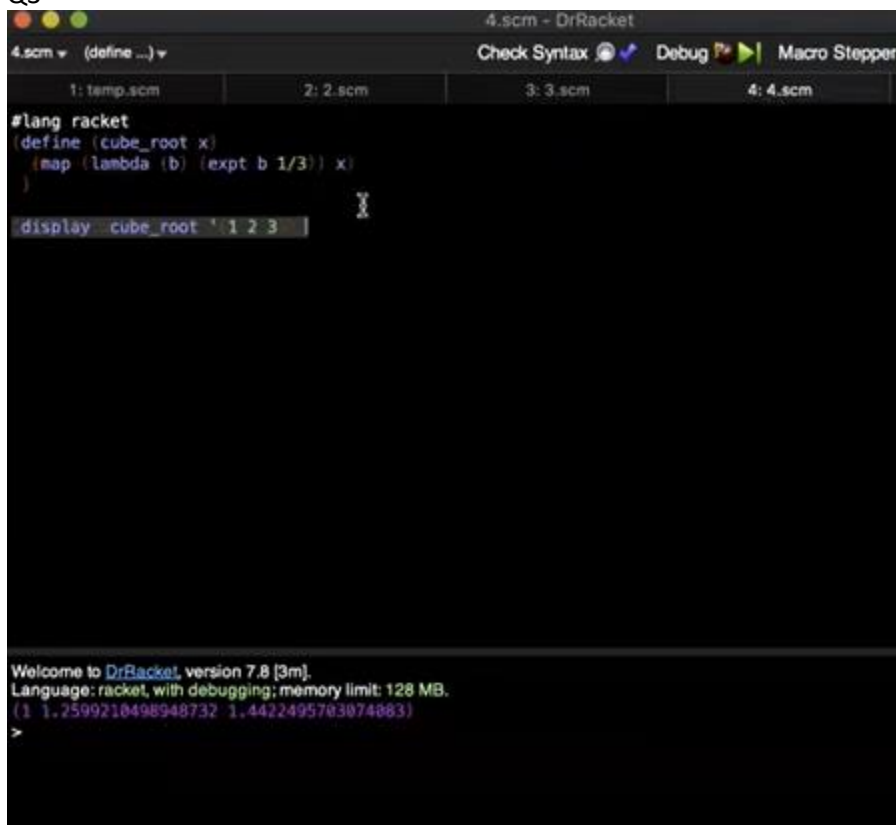
4. Write a higher order function called "method" that takes in two inputs – a method and another argument. This function then applies the method to each word of the argument.

```
(method square '(1 2 3 4)) -> (1 4 9 16)
```

```
(method first '(it is snowing)) -> (i i s)
```

Instructor/TA signature

Q3



The screenshot shows the DrRacket IDE with a file named 4.scm. The code defines a function `cube_root` using `define` and `map`. The function takes an argument `x` and maps a lambda function over it, which computes the cube root of each element using `expt`. The code also includes a `display` statement to show the result of `cube_root '1 2 3`. The bottom of the window shows the DrRacket welcome message and the output of the code: `(1 1.2599210498948732 1.4422495703074083)`.

```
#lang racket
(define (cube_root x)
  (map (lambda (b) (expt b 1/3)) x))

display (cube_root '1 2 3)
```

Welcome to DrRacket, version 7.8 [3m].  
Language: racket, with debugging; memory limit: 128 MB.  
(1 1.2599210498948732 1.4422495703074083)  
>

Q4

WORKING CODE::::

```

)
(define (method func x)
  (map func x)

  )
(define (char->symbol chr)
  (string->symbol (list->string (list chr))))

(define (symbol->list sym)
  (map char->symbol
    (string->list (symbol->string sym))))

(define (square x)
  (expt x 2)

  )

(define (first2 x)
  (first (symbol->list x))

  )

```

#lang racket

```

(define (method func arg)
  (map func arg)
  )

(define (square x)
  (* x x)
  )

(define (first s)
  >first s
  )

(display (method square '(1 2 3 4)))
(newline)
(display (method first '(it is snowing)))

```

Welcome to [DrRacket](#), version 7.8 [3m].  
 Language: racket, with debugging; memory limit: 128 MB.  
 (1 4 9 16)

5. The code below aims to remove all elements that are divisible by 'e' from the list 'L'. Type in the code and use tools in Dr. Racket to detect and debug any erroneous expression in the code. Show that your fixed code returns correct results

```
(define (indivisible e L)
```

```
(map (remove 0 ((map ((lambda (x y) (modulo x y))) L e))))
```

Instructor/TA signature

Q5 divisible by e errors type the code and use tools in Dr. Racket

```
#lang racket
(define (indivisible e L)
  map (remove 1 (map (lambda (x y) (modulo x y)) L e)))

(indivisible 3 '1 3 4 6 7 8)
```

Welcome to DrRacket, version 7.8 [3m].  
Language: racket, with debugging; memory limit: 128 MB.

...1a/Desktop/5.scm:3:24: arity mismatch;  
the expected number of arguments does not match the given number  
expected: 2  
given: 0

```
#lang racket
(define (indivisible e L)
  (remove 0 (map (lambda (x) (modulo x e))
                 L)
  )
)

(display (indivisible 3 '(1 2 3 5 6)))
```

```
Welcome to DrRacket, version 7.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
(1 2 2 0)
>
```

---

## Lab 1: Introduction to EMAC and Introduction to Scheme

## QUESTION 1

### Returning a Value

1. Create the file SchemeFlipped7.scm. Inside that file, create a method named "CircleArea". This function should take one parameter, the radius of a circle and return its area. Indent your code!! Call CircleArea twice in the main with two different radius 2 and 4, and the expected output is as follows:

```
(CircleArea 2) = 12.56  
(CircleArea 4) = 50.24
```

Please have TWO Putty windows open when presenting your answer. One with the code and the second with the code running.

### Returning a type

2. Create the file SchemeFlipped8.scm. Inside that file, create a function named "typeID" that accepts one value inside the function, it needs to ID the basic type passed in. Use the manual to help. The function should work with all basic types, BUT we want to see three calls to your new function use the following, you can display inside main:

- a. integer
- b. Boolean
- c. Float
- d. String

These calls all need to happen and produce the correct output to receive credit.

Please have TWO Putty windows open when presenting your answer. One with the code and the second with the code running.

Q1 create method named "CircleArea"



```
File Edit Options Buffers Tools Minibuf Help
; File: SchemeFlipped7
; Written by: Sunhee Kim, Rishabh Tatia
; Date: 8/28/20
; TAMU email: sunheekgtamu.edu, tatiarisgtamu.edu
; Class: CSCE 314 (section 502)
; Description: This is the program for the first Lab
;=====
```

```
(define (CircleArea radius)
  (* 3.14 radius radius)
)

(display (CircleArea 2))
(display "\n")
(display (CircleArea 4))
```

```
-UU-:----F1 SchemeFlipped7.scm All L15 (Scheme) -----
Beginning of history; no preceding item
```

```
[tatiaris]@compute ~/csce_314/lab1> (18:21:29 08/28/20)
:: gsi SchemeFlipped7.scm
12.56
50.24
```

## Q2 typeId

```
File Edit Options Buffers Tools Scheme Help
; File: SchemeFlipped8
; Written by: Sunhee Kim, Rishabh Tatia
; Date: 8/28/20
; TAMU email: sunheekgtamu.edu, tatiarisgtamu.edu
; Class: CSCE 314 (section 502)
; Description: This is the second program for the first Lab
;=====

; given a variable, returns a string containing the type of the variable
(define (typeID var)
  (cond ((integer? var) "Integer")
        ((boolean? var) "Boolean")
        ((number? var) "Float")
        ((string? var) "String")
  )
)

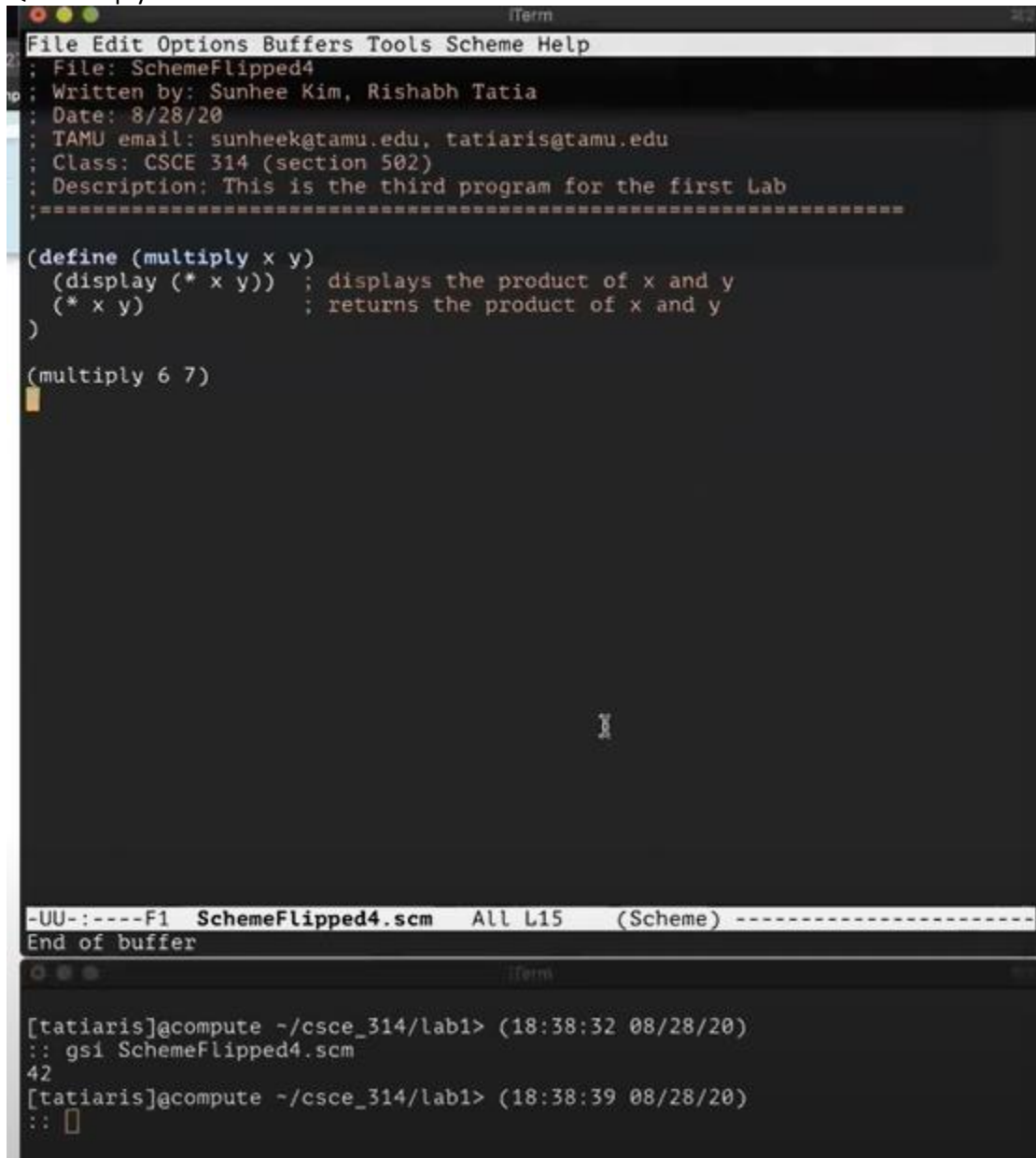
(println (typeID 4))
(println (typeID "string"))
(println (typeID 23.42))
(display (typeID #t))
```

### Creating a method to multiply 2 parameters

3. Create the file SchemeFlipped4.scm using Emacs. Inside that file, create a "multiply" method which takes two parameters, that will multiply both and display the result. To complete the process, have a `main()` to call the multiply function. Indent you code!! Treat the `()`s as `{}`s!! Once complete, go to the scheme interpreter to run the file you just created.

Please have TWO Putty windows open when presenting your answer. One window with the code and the second with the code running.

Q3 "multiply"



The image shows two terminal windows. The top window is an Emacs editor displaying a Scheme file named SchemeFlipped4.scm. The file contains a `define` block for a `multiply` function and a `(multiply 6 7)` call. The bottom window is a Scheme interpreter showing the execution of the file, resulting in the output `42`.

```
File Edit Options Buffers Tools Scheme Help
: File: SchemeFlipped4
: Written by: Sunhee Kim, Rishabh Tatia
: Date: 8/28/20
: TAMU email: sunheekgtamu.edu, tatiaris@tam.u.edu
: Class: CSCE 314 (section 502)
: Description: This is the third program for the first Lab
: =====
(define (multiply x y)
  (display (* x y)) ; displays the product of x and y
  (* x y)           ; returns the product of x and y
)
(multiply 6 7)
```

```
-UU-:----F1 SchemeFlipped4.scm All L15 (Scheme) -----
End of buffer

[tatiaris]@compute ~/csce_314/lab1> (18:38:32 08/28/20)
:: gsi SchemeFlipped4.scm
42
[tatiaris]@compute ~/csce_314/lab1> (18:38:39 08/28/20)
::
```

QUESTION 4:

The next question is 4, please ignore the 3.

### Using Multiple Functions

3. Create the file SchemeFlipped6.scm. Inside that file, create a function named "CurvedGrade" that accepts an integer, and a lambda function. CurvedGrade should use the function passed in to modify the grade and return the value, the main function should display the returned value. The passed function must be a lambda function. Call CurvedGrade twice in main with two different grades and two different curving functions as follows:

Function 1: multiply the grade by 1.2

Grade 1: 80

Function 2: add five to the grade

Grade 2: 90

"curvedgrade"

Q4

```
File Edit Options Buffers Tools Scheme Help
: File: SchemeFlipped6
: Written by: Sunhee Kim, Rishabh Tatia
: Date: 8/28/20
: TAMU email: sunheek@tamu.edu, tatiaris@tamu.edu
: Class: CSCE 314 (section 502)
: Description: This is the program for the first Lab
: =====
: takes in a curve function and an integer and returns the modified value of\
the grade
(define (CurvedGrade lambda_function grade)
  (lambda_function grade)
)

(display
  (CurvedGrade
    (lambda (x) (* x 1.2)) ; multiplies the grade by 1.2
    80
  )
)

(display "\n")

(display
  (CurvedGrade
    (lambda (x) (+ x 5))
    90
  )
)

```

-UU-:\*\*\*-F1 SchemeFlipped6.scm All L29 (Scheme) -----  
End of buffer

[tatiaris]@compute ~/csce\_314/lab1> (18:55:56 08/28/20)  
:: gsi SchemeFlipped6.scm  
96.  
95  
[tatiaris]@compute ~/csce\_314/lab1> (18:55:58 08/28/20)  
::

Day 1:

1. Ambiguity – (Grammars and parse trees notes pg 21) ->

<https://www.youtube.com/watch?v=QTM-QTFxf8A&t=378s>

- a. where a sentence can be represented by more than one parse tree
  - i. NOT DERIVATION!!!
  - ii. this is bad!!!
  - iii. could be that
    - 1. left/right most derivations do not match!!
    - 2. several lefts don't match!!
- b. why does this matter??
  - i. mathematically seems the same
  - ii. just try programming this!!
- c. In proving ambiguity you are TRYING TO MISMATCH with the SAME TARGET Syntax
  - i. just make sure your syntax is correct first!!

Context Free Grammar vs Regular Grammars

- 1. for each of the following grammars, briefly describe the language
- 2. prove that the following grammar is ambiguous parse trees
- 3. The equation is given below create the left and rightmost derivation for the following grammar

1. For each of the following grammars, briefly describe the language it defines in a sentence or two. Assume that the *start* symbol is S for each and that any symbol found only on the right hand side of a production is a terminal symbol. We've done the first one for you as an example. Hint: if it's not obvious by inspection, try writing down sentences in the language until you can see the patterns emerging. If it is obvious by inspection, write down some sentences generated by the grammar to verify that they match your expectations and fit your English description. Please be as precise as possible in describing the language.

(Example) 0 points:

$\langle S \rangle \rightarrow \langle A \rangle$   
 $\langle A \rangle \rightarrow a \langle A \rangle \mid a$

Answer: This grammar defines the language consisting of strings N a's (where  $N \geq 0$ )

Main Problem:

$\langle S \rangle \rightarrow a \langle S \rangle b \mid ab$

2. Prove that the following grammar is ambiguous. Please use JFlap to draw both of your trees. Use the "Finite Automaton" option in JFlap to manually draw your tree. Name your parse trees syntaxA.png.

$S \rightarrow C \mid D$   
 $C \rightarrow cCd \mid cd$   
 $D \rightarrow cDd \mid \text{epsilon};$

3. The equation is given below. Create the left and rightmost **derivation (not parse tree)** for the following grammar and input string, and attach your answer to the file named above.

Input String: aabbba

Grammar:

$S \rightarrow aAS$   
 $S \rightarrow a$   
 $A \rightarrow SbA$   
 $A \rightarrow SS$   
 $A \rightarrow ba$

1. The grammar defines the language consisting of

1. The grammar defines the language consisting of strings N a's and b's lined up after each other, where  $N \geq 2$ .

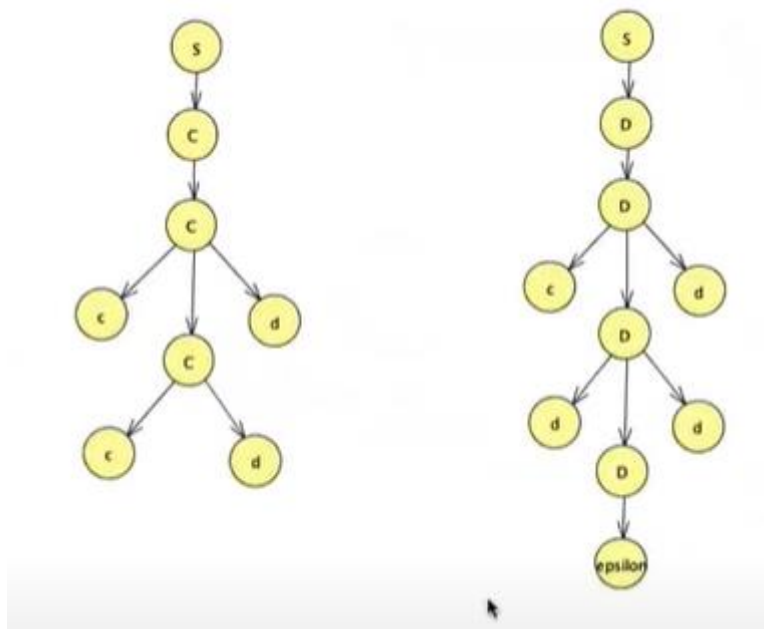
examples:

aaabbb

aabb

aaaaaaaaabbbbbbbbbbb

2.



3. (I've added spaces in between for readability of the substitutions, we understand that they shouldn't be there.)

Leftmost:

$S \rightarrow a A S$   
 $S \rightarrow a SbA S$   
 $S \rightarrow a ab A S$   
 $S \rightarrow a ab ba S$   
 $S \rightarrow aabbaa$

Rightmost:

$S \rightarrow a A S$   
 $S \rightarrow a A a$   
 $S \rightarrow a SbA a$   
 $S \rightarrow a Sb ba a$   
 $S \rightarrow aabbaa$

4. Give the strings abcd and accc, write a Context Free Grammar that can generate such strings and show how these strings can be derived from the grammar.

5. The following grammar tree is given in BNF:

$\langle id \rangle \rightarrow A | B | C$   
 $\langle expr \rangle \rightarrow \langle expr \rangle + \langle term \rangle$   
 $\quad | \langle term \rangle$   
 $\langle term \rangle \rightarrow \langle term \rangle * \langle factor \rangle$   
 $\quad | \langle factor \rangle$   
 $\langle factor \rangle \rightarrow ( \langle expr \rangle )$   
 $\quad | \langle id \rangle$

Rewrite grammar above to add ++ and -- unary operators

4. Give the strings abcd and accc, write context free grammar that can generate such strings and show how theses strings can be derived from the grammar
5. 5. The following grammar tree is given in BNF: rewrite grammar above to add ++ and -- unary operations



4. (Spaces added for readability of the substitutions)

Grammar:

$S \rightarrow C \mid CC$

$C \rightarrow a \mid b \mid c \mid d$

statement: abcd

$S \rightarrow C C$

$S \rightarrow CC C$

$S \rightarrow CC C C$

$S \rightarrow C C C C$

$S \rightarrow a C C C$

$S \rightarrow a b C C$

$S \rightarrow a b c C$

$S \rightarrow abcd$

statement: accc

$S \rightarrow C C$

$S \rightarrow CC C$

$S \rightarrow CC C C$

$S \rightarrow C C C C$

$S \rightarrow a C C C$

$S \rightarrow a c C C$

$S \rightarrow a c c C$

$S \rightarrow accc$

5.

$\langle id \rangle \rightarrow A \mid B \mid C$

$\langle expr \rangle \rightarrow \langle expr \rangle + \langle term \rangle$

$\mid \langle term \rangle$

$\langle term \rangle \rightarrow \langle term \rangle * \langle factor \rangle$

$\mid \langle factor \rangle$

$\langle factor \rangle \rightarrow (\langle expr \rangle)$

$\mid \langle id \rangle$

$\mid \langle id \rangle ++$

$\mid \langle id \rangle --$

$\mid ++\langle id \rangle$

$\mid --\langle id \rangle$

$\langle ++ \rangle$

statement: B + ++A

$\langle expr \rangle \Rightarrow \langle expr \rangle + \langle term \rangle$

$\langle expr \rangle \Rightarrow \langle term \rangle + \langle term \rangle$

$\langle expr \rangle \Rightarrow \langle factor \rangle + \langle term \rangle$

$\langle expr \rangle \Rightarrow \langle id \rangle + \langle term \rangle$

$\langle expr \rangle \Rightarrow B + \langle term \rangle$

$\langle expr \rangle \Rightarrow B + \langle factor \rangle$

$\langle expr \rangle \Rightarrow B + ++\langle id \rangle$

$\langle expr \rangle \Rightarrow B + ++A$

## 2. Parse Table

## QUESTION 1

1. Given the grammar

$$S \rightarrow S + S$$

$$S \rightarrow S * S$$

$$S \rightarrow id$$

Use a shift reduce parser for the input string:

id + id + id

2. Convert this grammar from left to right recursive for predictive parsing. You may assume that this grammar is unambiguous. Please use "X" as the new production symbol

$$E := E + a$$

$$| b$$

- Given the grammar use a shift reduce parser for the input string
- Convert this grammar from left to right recursive for predictive parsing new production symbol

Step	Parse Stack	Look Ahead	Uncanned	Parser Action
0	empty	id	+ id + id	just the start
1	id	+	id + id	P reducing id to S
2	S	+	id + id	Shift B
3	S+	id	+id	Shift A
4	S+id	+	id	P reducing id to S
5	S+S	+	id	Shift B
6	S+S+	id	S	Shift A
7	S+S+id	S		P reducing id to S
8	S+S+S	S		P reducing S+ S to S
9	S + S	S		P reducing S+ S to S
10	S	S		ACCEPT

Question 2:

$$E = E + a$$

$$| b$$

answer:

$$E = b X$$

$$X = + a X$$

$$| \epsilon$$

2. ....

3. use left-factoring to enable this grammar for predictive parsing you may assume this grammar is unambiguous. Please use the "|" symbol for multiple options

3. Use left-factoring to enable this grammar for predictive parsing. You may assume that this grammar is unambiguous. Please use the "|" symbol for multiple options.

$$S \rightarrow a a$$

$$| a b$$

$$| a$$

Question 3:

$S \rightarrow a a$   
 $\quad | a b$   
 $\quad | a$

answer:

$S \rightarrow a X$   
 $X \rightarrow a$   
 $X \rightarrow b$   
 $X \rightarrow \epsilon$

4. For the parse table below, prove the input string works. Show the stack, input stack, and the action. (Push, pop, etc)

Grammar

$A \rightarrow t B' D$   
 $\quad | v D'$   
 $B \rightarrow t B'$   
 $\quad | \epsilon$   
 $B' \rightarrow w B$   
 $\quad | u w B$   
 $D \rightarrow v D'$   
 $D' \rightarrow x B D'$   
 $\quad | \epsilon$

LL(1) Parse Table

	t	u	v	w	x
A	$t B' D$		$v D'$		
B	$t B'$		$\epsilon$		$\epsilon$
B'		$u w B$		$w B$	
D			$v D'$		
D'					$x B D'$

Input String

tuwvxtw

4 for the parse table below prove the input string works. Show the stack, input stack, and the action

Stack	Input	Action
\$	tuwvxtw	pop() Push(A)
A \$	tuwvxtw	pop() push(t B' D)
tB'D \$	tuwvxtw	pop() next++
B'D \$	uwvxtw	pop() push(uWB)
uwBD \$	uwvxtw	pop() next++
wBD \$	wvxtw	pop() next++
BD \$	vxtw	pop()
D \$	vxtw	pop() push(v D')
v D' \$	vxtw	pop() next++
D' \$	xtw	pop() push(x B)
x B D' \$	xtw	pop() next++
B D' \$	tw	pop() push(t B')
t B' D' \$	tw	pop() next++
B' D' \$	w	pop() push(w B)
w B D' \$	w	pop() next++
B D'	\$	DONE

5. Using the grammar below, show how the Shift-Reduce parser would work with the input string also given below. Please use the table given in the notes to be organized!! (Step, Parse Stack, Look ahead, Unscanned, Action) The input string SHOULD be correct.

Given grammar:

$E \rightarrow E + T$

$| T$

$T \rightarrow T * F$

$| F$

$F \rightarrow ( E )$

$| id$

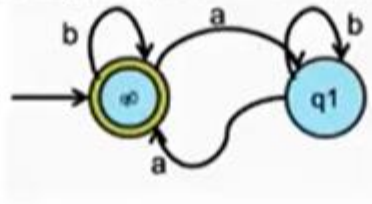
Given input:

id + id \* id

Step	Parse Stack	Look Ahead	Unscanned	Parser Action
0	empty	id	+ id * id	Shift A
1	id	+	id * id	P reducing id to F
2	F	+	id * id	P reducing F to T
3	T	+	id * id	P reducing T to E
4	E	+	id * id	Shift B
5	E +	id	* id	Shift A
6	E + id	*	id	P reducing id to F
7	E + F	*	id	P reducing F to T
8	E + T	*	id	Shift B
9	E + T *	id	\$	Shift A
10	E + T * id	\$		P reducing id to F
11	E + T * F	\$		P reducing T * F to T
12	E + T	\$		P reducing E + T to E
13	E	\$		ACCEPT

**DFSM** Deterministic Finite State Machine notes are Lexical Analysis:

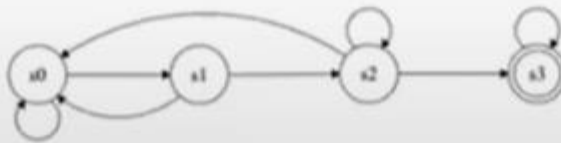
1. Convert the following DFA to a regular expression which represents the DFA.



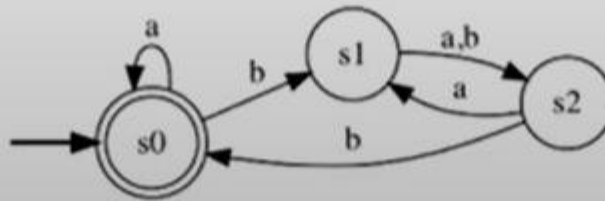
Please write the regular expression here.

3. Using **JFlap** create a DFA for the given regex expression,  $a?(b|c)^*$ . Run two input strings on this DFA
- > bbbb
  - > abcbbc
3. Using **JFlap** create a DFA for the given regex expression,  $x^*(yz?)^+$ . Run three input strings on this DFA
- > xxxyyy
  - > yzyzyyz
  - > xyzxyzxyz

4. You have been given a DFA with the alphabet  $\Sigma = \{a, b, c\}$ . The DFA doesn't have any transition labels on the edges. Label the DFA correctly so that it accepts any string with "aab" as a substring. There should not be any redundancy. (s0 is the start state)



- 5.. Give the table of transitions for the given DFA



## Entire List of Reg. Expression Symbols

	Description
<b>^</b>	Put a <u>circumflex</u> at the start of an expression to match the beginning of a line.
<b>\$</b>	Put a <u>dollar sign</u> at the end of an expression to match the end of a line.
<b>.</b>	Put a <u>period</u> anywhere in an expression to match any character.
<b>*</b>	Put an <u>asterisk</u> after an expression to match zero or more occurrences of that expression.
<b>+</b>	Put a <u>plus sign</u> after an expression to match one or more occurrences of that expression.
<b>—</b>	Put an <u>underscore</u> to match a comma(,), left brace ({), right brace (}), the beginning of the input string, the end of the input string, or a space.
<b>?</b>	Put a <u>question mark</u> after an expression to match zero occurrences or one.
<b>[ ]</b>	Put characters inside <u>square brackets</u> to match any one of the bracketed characters but no others.
<b>[ ^ ]</b>	Put a leading <u>circumflex</u> inside <u>square brackets</u> with one or more characters to match any character except those inside the brackets.
<b>[ - ]</b>	Put a <u>hyphen</u> inside <u>square brackets</u> between characters to designate a range of characters.
<b>&lt;</b>	Put a <u>left angle bracket</u> at the start of an expression to match the beginning of a word.
<b>&gt;</b>	Put a <u>right angle bracket</u> at the end of an expression to match the end of a word.



<code>\b</code>	Use <u>backslash b</u> to match the backspace character (# 8).
<code>\t</code>	Use <u>backslash t</u> to match the tab character (# 9).
<code>\n</code>	Use <u>backslash n</u> to match the new-line character (# 10).
<code>\f</code>	Use <u>backslash f</u> to match the form-feed character (# 12).
<code>\r</code>	Use <u>backslash r</u> to match the carriage-return character (# 13).
<code>\x00</code>	Use <u>backslash x</u> with a hexadecimal code of \x00 to \xFF to match the corresponding character.
<code>\</code>	Use a <u>backslash</u> to make a regular-expression symbol a literal character.
<code> </code>	Use a <u>vertical bar</u> between expressions to match either expression. Use up to nine vertical bars, separating up to ten expressions, any of which are to be found in a line. NOTE: Spaces before and after the vertical bar are significant. For example, "near   far" represents a regular-expression search for "near " or " far", not "near" or "far".
<code>&amp;</code>	Use an <u>ampersand</u> between expressions to match both expressions. Use up to nine angstroms, joining up to ten expressions, all of which are to be found in a line. NOTE: Spaces before and after the angstrom are significant. Thus, "near & far" is not the same as "near&far", which is probably what you want.

1.

1. regular expression: (b\*(ab\*a)?)\*

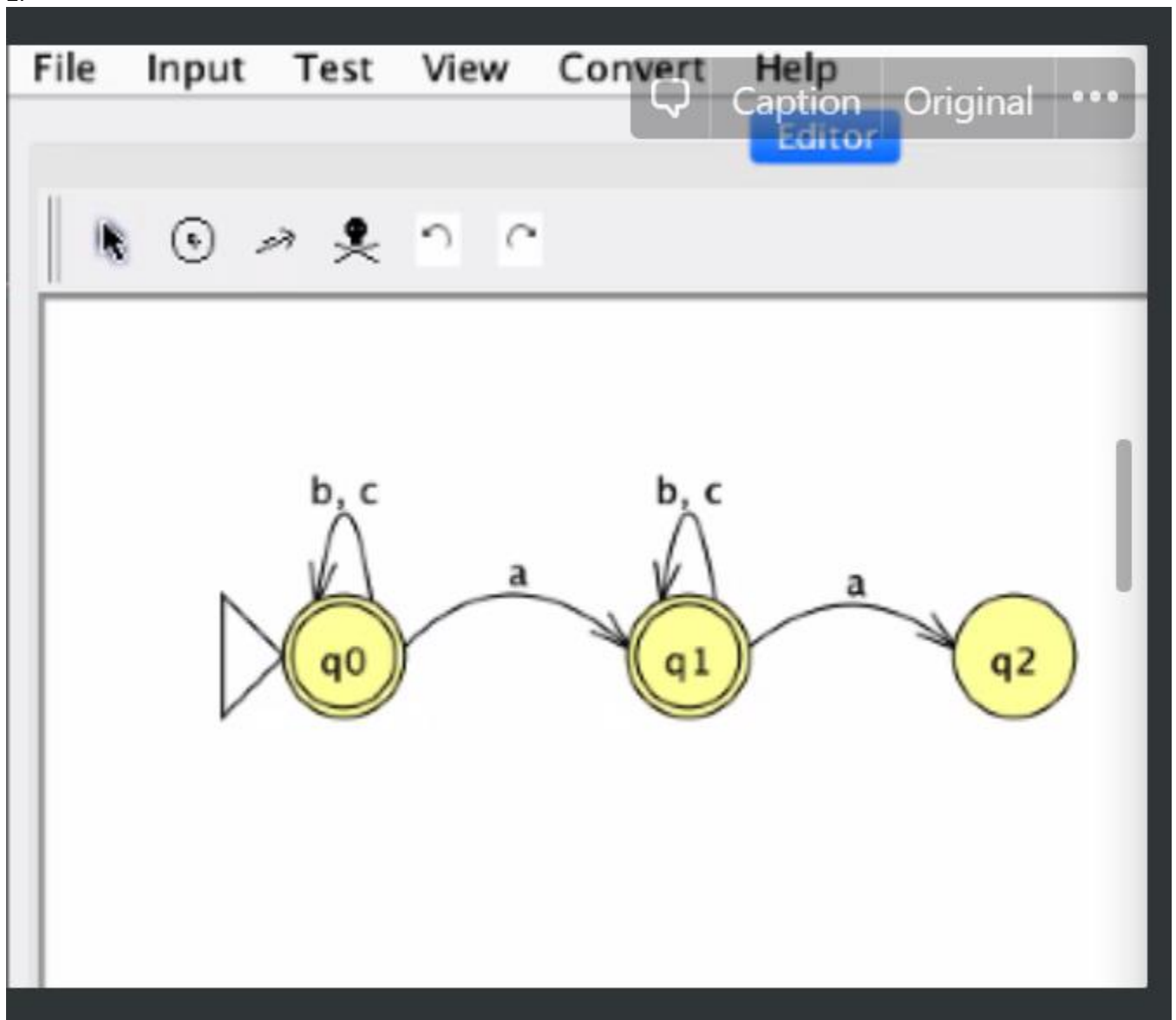
5.

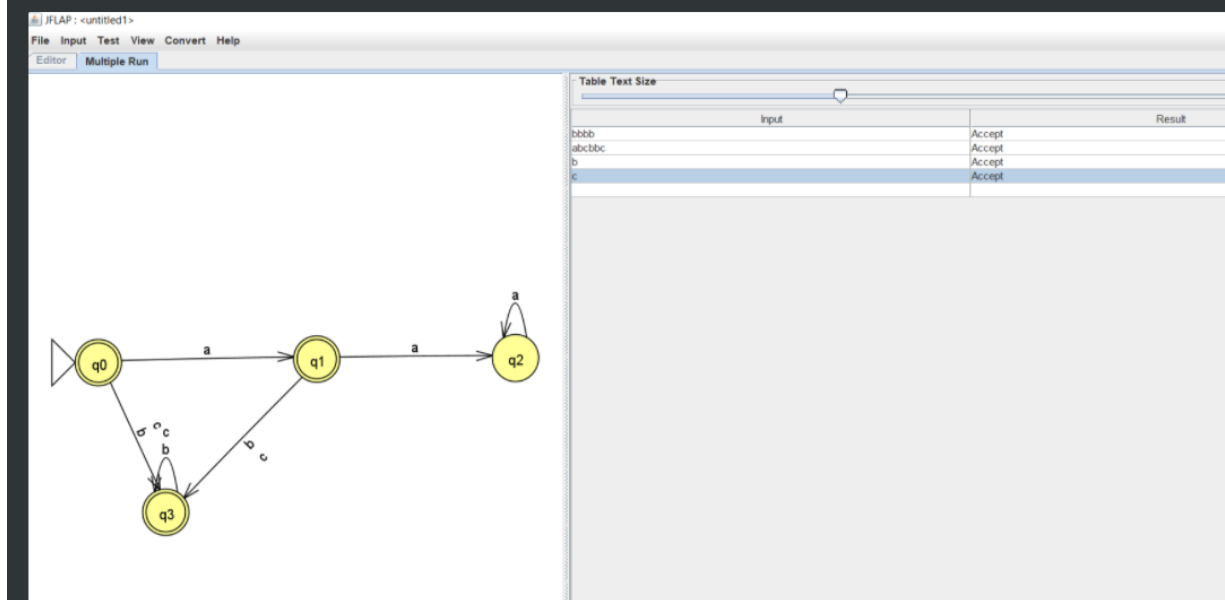
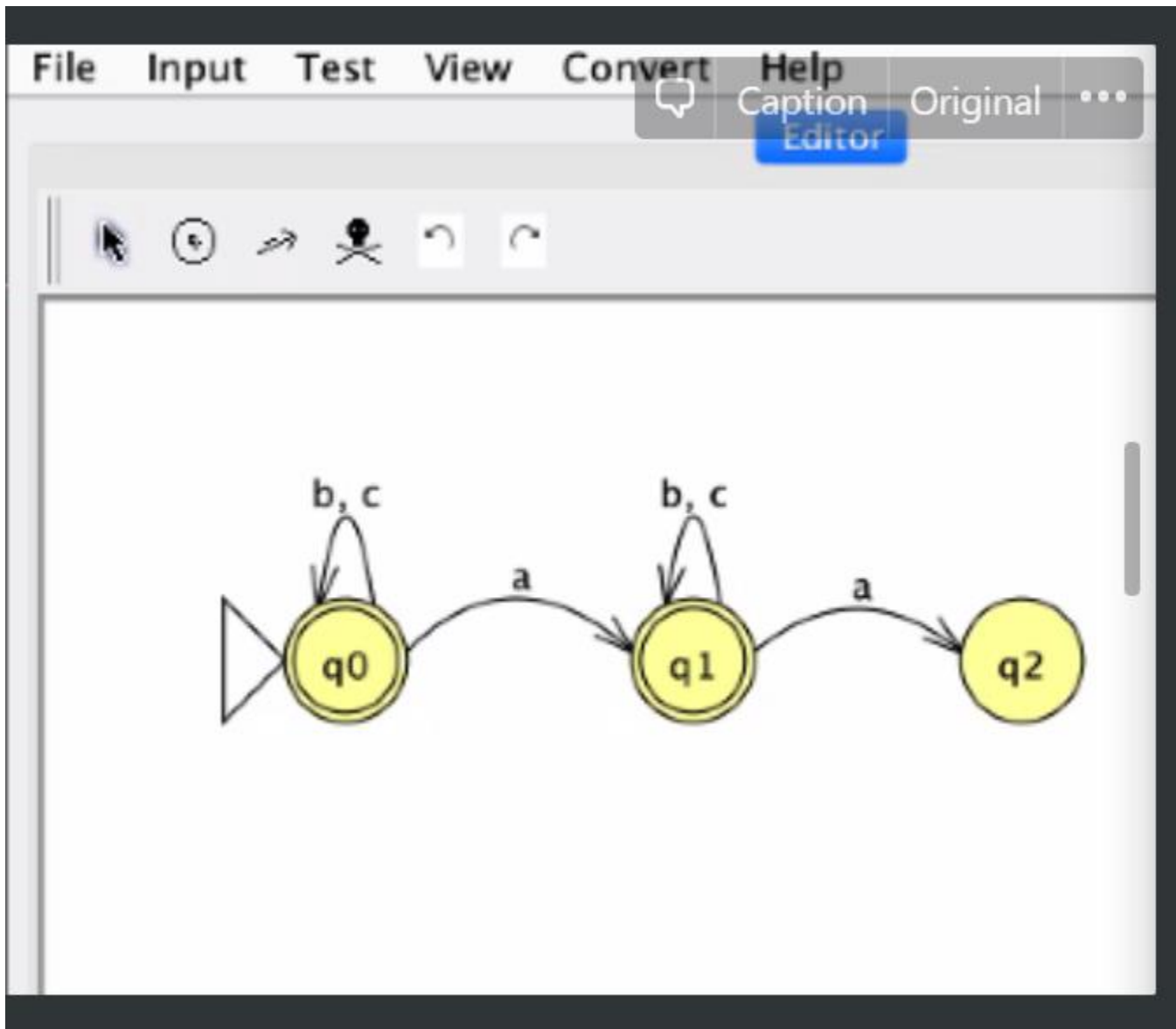
997  
110

inputs

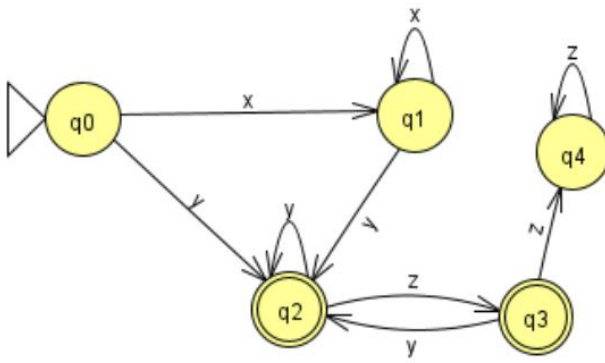
s			
t		a	b
a	0	0	1
t	1	2	2
e	2	1	0
s			

2.





3.



4.

Input	
	Reject
xxxyyy	Accept
yzzyyz	Accept
xyzxyzxyz	Reject