Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

CSI 3120, Fall 2016, midterm exam B
October 20, 2016, 08:30 – 09:50
Lecturer: Dr. Rafael Falcon

Last name(s) First name(s) Student number

....

Notes

1. This is a closed-book exam. **No electronic devices are allowed**. You can have one double-sided page of notes, US Letter size (8.5 by 11 inches); no magnifying glass, please. No books and no other papers are permitted; use this questionnaire for rough work, including the spare page. If you have a bilingual English-to-X dictionary, you can use it, but only under supervision.
2. Write comments and assumptions to get partial marks.
3. Please attach the cheat sheet to the exam when you hand it in.
4. Beware that poor handwriting could affect grades.
5. Do not remove the staple holding the examination pages together.
6. Write your answers in the space provided. Use the back of pages if necessary.
7. The spare page(s) are for your own use. They can be detached from the exam. Their content will not be graded.
8. Peeking at your neighbours' work will result in expulsion from the exam.

Problem	1	2	3	4	5	6	7	8	Bonus			
Maximum	2	3	2	6	10	8	5	6	2			
Grade										**Total**		/ 42

Problem 1 [2 points]

Consider the following predicate definition in Prolog:

```
guess1( L1, L2, N ) :-
  setof( p(X, Y),
         (member(X, L1), member(Y, L2), X < Y),
         L
       ),
  length(L, N).
```

Now, consider the execution of the query

```
?- guess1([7, 13, 17], [5, 11, 19], L).
```

The result of the query is

A	5	B	3
C	8	D	4

Answer
D: 4

Problem 2 [3 points]. Write a Scheme function named *elem* that takes 2 arguments: an integer and a list of atoms. Assume that the integer is never larger than the length of the list. Your function must return the atom appearing at the position specified by the integer. Example: `(elem 5 '(a b c d e f g h))` should return *e*. Please write the function using only elementary constructs such as car, cdr, etc.

```
(define (elem  n lst)
  (define (elem-check current-n current-lst)
     (if (= n current-n)
       (car current-lst)
       (elem-check (+ current-n 1) (cdr current-lst))))
  (elem-check 1 lst))

(print (elem 1 '(a b c d e f g h)))
(print (elem 2 '(a b c d e f g h)))
(print (elem 3 '(a b c d e f g h)))
(print (elem 4 '(a b c d e f g h)))
(print (elem 5 '(a b c d e f g h)))
(print (elem 6 '(a b c d e f g h)))
```

Problem 3 [2 points]

Consider the following grammar, in which <N> stands for any numerical constant and the start symbol of the grammar is <E>:

```
<E>  →  <E>  <E>  <OP>  |  <N>
<OP> →  +  |  *  |  -  |  /
```

The grammar does *not* generate one of the following instances of <E>. Which one?

A	6 9 4 3 + - 2 6 + *
B	6 5 4 + 8 2 / - 6 + *
C	6 5 6 2 - + 9 3 / - 7 3 - + *
D	6 9 4 + 6 + *

Answer

YES A (A is not a valid instance based on the provided grammar)

```
 6   9   4   3   +   -   2   6   +   *
<N> <N> <N> <N> <O> <O> <N> <N> <O> <O>
<N> <N>     <E>     <O>     <E>     <O>
<N>         <E>             <E>     <O>
<N>                 <E>
```

NOT B

```
 6   5   4   +   8   2   /   -   6   +   *
<N> <N> <N> <O> <N> <N> <O> <O> <N> <O> <O>
<N>     <E>         <E>     <O> <N> <O> <O>
<N>                 <E>         <N> <O> <O>
<N>                             <E>     <O>
                                <E>
```

Not C

```
6   5   6   2   -   +   9   3   /   -   7   3   -   +   *
<N> <N> <N> <N> <O> <O> <N> <N> <O> <O> <N> <N> <O> <O> <O>
<N> <N>     <E> <O>     <E> <O>     <E> <O> <O>
<N>         <E>         <E> <O>     <E> <O> <O>
<N>                     <E>         <E> <O> <O>
<N>                                 <E>     <O>
                                    <E>
```

Not D

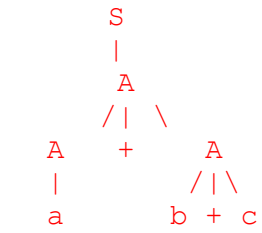
```
 6   9   4   +   6   +   *
<N> <N> <N> <O> <N> <O> <O>
<E> <E> <E> <O> <E> <O> <O>
<E>     <E>     <E> <O> <O>
<E>         <E>         <O>
            <E>
```

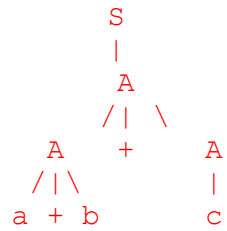
Problem 4 [6 points]

Prove that the following grammar is ambiguous:

```
<S> → <A>
<A> → <A> + <A> | <id>
<id> → a | b | c
```

Answer: The derivation of the sentence a + b + c generates two different parse trees; therefore, the grammar is ambiguous.





Problem 5 [10 points]

Consider the following grammar with <T> as the start symbol:

```
<T>  →  a <P> a <S>
<P>  →  <P> a <S>  |  a b
<S>  →  <S> b b  |  <S> b  |  b
```

1. **[4 points]** Find the First and Follow sets for <P> and for <S>.

Answer

	First	Follow
T	a	$
P	a	a
S	b	$, a, b

2. **[6 points]** Eliminate direct left recursion in this grammar.

Answer
The rule is
A -> A α1 |… | A αm | β1 | β2 |…| βn

Becomes
A -> β1 A' | β2 A' | … | βn A'
A' -> α1 A' | α2 A' | … | αm A' | ε

<T> → a <P> a <S>
<P> → a b <P'>
<P'> → a <S> <P'> | ε
<S> → b <S'>
<S'> → b b <S'> | b <S'> | ε

Problem 6 [8 points]

Given the code snippet below:

```
if (a == b)
      b = 2 * a + 1;
else
      a = a + 1;
     b = a ^ 2;
{b > 1}
```

a) **[4 points]** What is the weakest precondition to satisfy the above post-condition?

Computing the precondition for the THEN block
{??} b = 2 * a + 1; {b > 1}
2 * a + 1 > 1
a > 0

Computing the precondition for the ELSE block
{??} a = a + 1; {??}
{??} b = a ^ 2; {b > 1}
a ^ 2 > 1
a > 1 || a < -1
a = a + 1 {a > 1 || a < -1}
a + 1 > 1 || a + 1 < -1
a > 0 || a < -2

To satisfy both the THEN and the ELSE clauses, the weakest precondition P for the IF statement is a > 0.

b) **[4 points]** Prove the statement logically follows from the precondition in (a) using the IF/ELSE axiomatic semantic rule.

Now we need to prove two things:

- That we have the THEN clause with precondition {B and P} and postcondition {b > 1}.
- That we have the ELSE clause with precondition {not B and P} and postcondition {b > 1}.

From {a == b and a > 0} we can logically derive {a > 0} which coincides with the precondition for the THEN clause. The postcondition follows from the precondition.

From {a != b and a > 0} we can logically derive {a > 0}, from which we can logically infer {a > 0 || a < -2}, which is the precondition for the ELSE clause. The postcondition follows from the precondition.

Problem 7 [5 points]

Write True (T) or False (F) in front of the following statements. Justify your rationale for the False (F) statements.

_____ In operational semantics, the meaning of a program is determined by the execution of its listing on a virtual machine.

Answer: True
That's the exact definition of operational semantics

_____ Syntactic ambiguity of language structures is detrimental to the lexical analyzer.

Answer: False
The lexical analysis phase is all about translating the source code into more easily processable tokens, whereas the parsing phase, what we focused on most is best served with unambiguity.

_____ Algol 68 was characterized by its extreme use of the orthogonality concept; this resulted in its widespread adoption and booming popularity.

Answer: False
Yes Algol 68 was extreme in orthogonality, but that resulted in not being used and not being popular at all

_____ SLR and LALR are two flavors of bottom-up parsing strategies.

Answer: True
Yes, SLR and LALR are two flavours of Shift-Reduce parsing, which is a Bottom-up approach. The third being Canonical LR parsing.

_____ The quote operator (') in Scheme tells the interpreter to evaluate list arguments from right to left.

Answer: False
The quote operator supresses the evaluation of its argument

Problem 8 [6 points]

Explain what it means for a language to be reliable. Please illustrate your response with two examples.

Answer

(The solution should speak to the high-level characteristics of language reliability and then examples should accurately reflect the described characteristics. What follows is an example from my personal view.)

Reliability in a language is all about safety for the programmer and can come in many forms:

- Static Analysis to identify issues in the code (e.g. type checking, null pointers, etc.) before it is even run. For example, Java ensures through Type Checking that your code is properly sending the appropriate data types to function calls. Conversely, a language like Ruby, which isn't typed relies on additional automated testing to ensure that the provided data types are shaped appropriately (known as duck-typing).

- Ability to handle errors and exceptions. Java provides both Checked (you must explicitly handle them) and Unchecked (if uncaught they will bubble up) exception mechanisms that allow the developer to recover from exceptional cases in both a very specific or very broad sense depending on the context. A language like Erlang offers a different approach whereby running processes are OK to crash as a supervision tree is available to properly manage and restart dependencies in the appropriate order.

- Stability of the runtime environment. The JVM is a mature platform and allows a functional language like Clojure to execute in a stable environment. A language like Erlang supports hot code swapping whereby you can update the system without having to restart it, thus making it much easier to have zero downtime upgrades.

- Syntactic characteristics to avoid unintentional errors such as if (x = 1) versus if (x == 1), having fewer data types (e.g., just ints and doubles not small ints, big ints, unsigned ints, etc.)

- Unambiguous names most often related to scope and visibility rules. For example, the original Lisp implementation was dynamically scoped versus Scheme which was lexically scoped (and much easier to reason about, thus making it much safer for the programmer)

Bonus [2 points]: We briefly discussed Lisp in class and studied Scheme in more detail. Mention two other functional programming languages not studied in class.

Common Lisp, Haskell, ML, Scala, Elixir, Clojure, Erlang, F#, R, Python, SequenceL, D

Some of the examples above are not fully functional programming languages but are accepted as answers.

Spare page 1

Spare page 2