Midterm 2 Solutions

1. [20 points] Consider the language that consist of possibly empty lists of the identifier x enclosed by parentheses and separated by commas. The language includes { `()` `(x)` `(x,x)` `(x,x,x)` } etc.
(a) Write a short grammar suitable for use with your LALR parser generator, including augments, so that you accept only these sentences. The sentence `()` should produce an AST that looks like `(ExpList)`. The sentence `(x,x,x)` should look like `(ExpList x x x)`.

Many acceptable answers; one was the following:

```
(defparameter g1
  '(
    (S --> |(| L |)|          #'(lambda (o l c) (cons 'ExpList l)))
    (L -->                    #'(lambda () nil))
    (L --> x O                #'(lambda (x o) (cons 'x o)))
    (O -->                    #'(lambda () nil))
    (O --> \, x O             #'(lambda (c x o) (cons 'x o)))))
```

(b) Stephen claims this language can be recognized by a DFA with 5 states. Is he right? If so, draw the DFA. If not, explain why this cannot be.

Yes. The DFA looks like this:

```
State 1 (start state)
-- Transition on ( to State 2
State 2
-- Transition on x to State 3
-- Transition on ) to State 5
State 3
-- Transition on , to State 4
-- Transition on ) to State 5
State 4
-- Transition on x to State 3
State 5 (final state)
-- No outgoing transitions
```

2. [8 points] In assignment 3 you made extensive use of the program

```
(defun aug(&rest r)(if (null (cdr r)) (car r) r))
```

a. What program calls aug?

lalr-parser calls aug. More precisely: reduce-cat, a local function within lalr-parser, calls aug.

b. When is it called?

It is called when a reduction is performed.

c. What does r get bound to?

r is bound to a list of the results of parsing the constituents of the rhs of the rule.

d. What does `(aug '(1 2 3))` return?

(1 2 3)

3. [16 points] Henry considers implementing the following modifications to the Tiger programming language. For each proposed modification, identify the component(s) of the compiler that Henry will need to change. Select from the lexical analyzer (L), parser (P), type-checker (T), or interpreter (I). It is possible that no change is necessary at all (N). Sometimes more than one may be appropriate or possible. Unless it is utterly clear to you that there is only one possible answer that we are after, we recommend that you write your answer here and also explain on the reverse of this page why you said so.

1. Forbid the type definition let type foo= {}
P. Forbid in grammar. Or T/I: make typechecker/interpreter cause an error here.
2. Make the expression () identical to VOID.
N.
3. Make the expression () identical to nil.
P. Parser's augment for () should return nil. Or, have T/I to return nil on empty explist.
4. Allow arithmetic on Boolean results like `(1>0)+(2<3)`.
N.
5. Make the statement `(a<b<c)` mean the same as `(a<b) & (b<c)`.
P. Have our parser mangle the AST similarly to a&b.
6. Introduce a type Boolean with two values true and false, such that comparisons result in one of these values rather than 1 or 0.
L/P/T/I: We can make true and false reserved words in the lexer, and modify the parser to require them in boolean expressions. This would also require modifications to the typechecker and interpreter to do logic only on booleans. This is the preferred answer.
Alternatively, we can make the modifications only to the typechecker/ interpreter and leave true/false as non-reserved tokens.
7. Allow a recursive `x.next:=x` given that x is of `type z= { val=int, next=z }`.
N.
8. Addition of a procedure `shell()` that takes no argument and opens up an interactive shell window, returning an integer "return code" when the `shell` command is done.
T/I. We add this function to the initial environment in the typechecker as well as implementing the function in the interpreter.

4. [10 points] Lee is building an implementation of Tiger (not necessarily built on top of Lisp). Lee needs to consider storage management, and needs your answers to the following questions in complete English sentences.

1. At what point in a Tiger computation would a garbage collection likely be started? Hint: you ordinarily do a garbage collection when you need more memory but don't seem to have enough free. Even bigger hint: What kinds of statements in Tiger allocate heap space?
Garbage collection would start when we try to allocate an array or record and do not have enough memory.
2. What kinds of statements in Tiger allocate space on a stack?
Variable declarations in let statements, function calls, and for loops all allocate space on the stack.
3. What kinds of statements in Tiger de-allocate space?
Nothing deallocates memory on the heap, but return from a function as well as the "end" of a for/let all deallocate stack space.
4. If you wish to provide explicit heap memory management, what changes would you have to make to Tiger?
We would need to implement many free()-like statements, one for each type, to implement explicit memory management.
5. Any discussion of garbage collection refers to the roots needed to start the operation. In a Tiger implementation using garbage collection, what are the roots?
Live identifiers in accessible environments are roots of the garbage collection in tiger.

5. [9 points]

a. How does the Tiger interpreter discussed in class deal with logical "and" and "or" expressions such as `A&B|C` ?

The parser translates logical and/or operators to if statements, which are then interpreted as ordinary if statements by the interpreter. The interpreter never sees and/or – only if statements.

b. Typechecking comparison (such as `>`) is more elaborate in Tiger than typechecking addition `+`. Explain why.

Ordering comparison operators (such as >) work on string or int. Equality comparison operators (= and <>) operate on any type, provided that both operands are of the same type. Arithmetic operators only work on int.

c. What do you know about the type of x give than the expression `if x=nil then 1 else 2` passes a semantic check?

x is a record type.

6. [6 points] Explain in complete English sentences what steps are necessary in the Tiger interpreter to interpret a Tiger expression "call" such as `F(3,a)`.

- 3 and a are evaluated by recursive calls to the interpreter.
- The formal parameters, say x,y of F are bound to 3, a respectively and added into the environment of F.
- The body of F is interpreted in that environment

7. [16 points] For each of the following Tiger programs, determine where the processing will detect an error. If possible, indicate a location in the program text.
Mark the location L if it will fail in Lexical Analysis.
Mark the location P if it will fail in Parsing.
Mark the location T if it will fail Type Checking.
If there are no errors, write OK.
Give some explanation as to why the failure occurs. If you think there are several errors such that if you correct the first one, there are still others, mention them all. If you are not sure what will happen, explain that too, and maybe we can give you some partial credit.

```
--01---
let type t1={a:int,b:t1}
    type t2={c:t1}
    var x:t2:=nil
  in x.c.b.a end
```
No errors (OK) - program fails at run-time only.
```
--02----
let type t1=int
    var x:t1
  in x:= 3 end
```
P, line 2: var x:t1 is not a legal variable declaration; it lacks an initializer.
```
--03---
let var x:=5 in x:=3+4^5 end
```
L, line 1: ^ is not a legal token or part of a legal token in Tiger.
```
--04---
let type a= {x:int, y:int}
    type b= {x:int, y:int}
    type c=a
    var i:a :=(x=1,y=2}
    var j:b :=nil
    var k:c :=nil
  in b:=j; k:=j end
```
P, line 4: (is unmatched.
T, line 7: b:=j is illegal because b is a type, not a variable.
T, line 7: k:=j is illegal because k and j are different types.
```
--05---
let
  type intlist ={hd:int, tl:intlist}
  type b=c
  type c=b
    in 1234 end
```
T, line 3/4: b and c are illegal mutually recursive types.
```
--06---
let
  type intarray = array of intarray
  var M:intarray := array[10] of M
    in 5676 end
```
T, line 1: intarray is an illegal recursive type.
P, line 2: illegal syntax for array initialization.
```
--07---
let function f(a:int):int=3*a*a end
  in print(f(2)); "done" end
```
P, line 1: end token without matching let/in.
T, line 2: print is of type string -> void, but f returns int.
```
--08---
3+4*(("big"<"little")>0)-9)
```
P, line 1: Unmatched). There is no "T" error in this program.

8. [4 points] What is the amortized cost (that is, the cost per reclaimed cell) of a 2-space copying garbage collector process in which, at time *t* a garbage collection is begun, there are *R* live nodes and a total of *H* cells in the heap. a. Explain any additional symbols you introduce using complete English sentences. b. What happens if *R* = *H*/2?

a) At time t, there are R live nodes and H total cells in the heap. A copying GC divides the heap in two equal sections, so the time to perform the GC is: (*R* * c) / (*H*/2 - *R*)
where c is the cost of copying a single cell from one section of the heap to the other.
b) If *R*=*H*/2, we see that the cost of a GC approaches infinity; in other words, the GC fails to free any memory and must either ask the OS for more memory or exit with an error.

9. [11 points] In the Leopard programming language, a language similar in some respects to Tiger, the following program is legal.

```
let
  function q(a,b)= let in a:=3;b:=7 end
  type ar=array of int
  var m:=ar [6] of 0
  var i:=2
in
  q(i, m[i+2]);
  for i:=0 to 5 do print(chr(m[i]+ ord("0")))
end
```

a. If Leopard has call-by-reference parameter passing, what are the six values in the array m when they are printed?
m[4] is 7; other elements are 0
b. If Leopard has call-by-name parameter passing, what are the six values in the array m?
m[5] is 7; other elements are 0
c. If Leopard has the same parameter passing as Tiger, what are the six values in the array m?
All six elements of m are 0
d. Why did we not write `print(m[i])`?
print() requires a string argument, not an int.
e. What prevents this program from being legal Tiger?
q(a,b) is not properly type declared. The declaration should be q(a:int,b:int):int or something similar.