CS 241 Final - Practice Problems

Pierre-Louis Guidez & Edward Tan

Winter 2019

You are encouraged to attempt these problems prior to the review session. Questions marked with *Exercise* will not be solved during the session.

1 MIPS and Regular Languages

1. For the following statements, indicate whether it is true or false and give a short explanation.
   (a) For any MIPS assembly program, the number of entries in its symbol table is always less than or equal to the number of instructions.
   (b) The largest positive decimal value that can be expressed by a k-bit two's complement number is 2^{k-1}.
   (c) For any language A, if there exists a NFA M such that $\mathrm{L}(M) = A$, then there exist a CFG G such that $\mathrm{L}(G) = A$.
   (d) For any language B, if there exists a CFG G such that $\mathrm{L}(G) = B$, then there exist a NFA N such that $\mathrm{L}(N) = B$.
   (e) If a language C is regular, than $\overline{C}$ is also regular. ($\overline{C}$ denotes the complement of C, i.e. all words not in C)
2. *Exercise.* Give a regular expression for the following language: The string over $\Sigma = \{a, b\}$ such that no two consecutive letters are the same.

2 Context Free Grammars

1. Give an example of a language that is context-free, but not regular.
2. What does it mean for a context-free grammar to be ambiguous? Give an example of an ambiguous context-free grammar.
3. Perform both a leftmost and righmost derivation on the string *aaabbc* given the grammar specified by the following production rules:
   0. $S \to aABc$ 1. $A \to aA$ 2. $A \to \epsilon$ 3. $B \to bb$

3 Top Down Parsing

1. What does LL(1) parsing stands for? When is a grammar not LL(1)?
2. Build a predict table for the following grammar, then parse the string $\vdash deaccb \dashv$
   0. $S' \to \vdash S \dashv$
   1. $S \to aYb$
   2. $S \to XS$
   3. $Y \to ccY$
   4. $Y \to \epsilon$
   5. $X \to de$

4 Bottom Up Parsing

1. Given the following grammar, construct the LR(0) DFA for it. Is the grammar LR(0)? Why or why not?

   0. $S' \to \vdash X \dashv$
   1. $X \to XbAb$
   2. $X \to XaBa$
   3. $X \to \epsilon$
   4. $A \to Aa$
   5. $A \to \epsilon$
   6. $B \to Bb$
   7. $B \to \epsilon$

2. Given the above grammar and the shift-reduce table below, parse the string $\vdash ababaab \dashv$

0	$\vdash$	shift 3	3	a	reduce 3	5	a	shift 9	8	a	shift 2
1	a	reduce 1	3	b	reduce 3	5	b	shift 1	8	b	shift 6
1	b	reduce 1	3	$\dashv$	reduce 3	6	a	reduce 6	9	a	reduce 4
1	$\dashv$	reduce 1	3	X	shift 10	6	b	reduce 6	9	b	reduce 4
2	a	reduce 2	4	a	reduce 5	7	a	reduce 7	10	a	shift 7
2	b	reduce 2	4	b	reduce 5	7	b	reduce 7	10	b	shift 4
2	$\dashv$	reduce 2	4	A	shift 5	7	B	shift 8	10	$\dashv$	shift 11

5 Error Checking, Semantic Analysis, and Code Generation

1. Each of the following WLP4 programs contains one or more errors. For each program below, circle the first error that is detected and state whether it is a lexical (scanning), syntax (parsing), semantic, or run-time error.

(a)
```
int wain(int* a, int b){
  int c = a*b;
  println(a);
  return b(1);
}
```

(b)
```
int wain(int a, int b) {
  int c = 0;
  if (12345678910 !== a) {
    c = a;
  } else {}
  return c;
}
```

(c)
```
int foo(int x) {
  return x * 2;
}

int wain(int *a, int b) {
  int foo = 1;
  println(foo(*(a + 1)));
  return foo + b;
}
```

(d)
```
int wain(int* a, int b) {
  int* c = NULL;
  a = c;
  *&*a = b;
  return b;
}
```

(e)
```
int wain(int a, int b) {
  if (b > 0){
     return a;
  }
  return b;
}
```

2. The following MIPS program contains many errors. Circle each error, describe it, and state whether it is a lexical (scanning), syntax, semantic, or run-time error. For ease, consider each error seperately (errors do not compound).

```
.import print
.export wain
beq $2, $0, 3
lis $3
.word print:
lis $3
.word -1
jr $ 31
```

3. Suppose WLP4 now supports pointers to pointers. For example, the following fragment of WLP4 code would be valid:

```
int **c = NULL;
int ***d = NULL;
d = &c;
```

You are given variables a, b and c. The following table indicates the type of each variable. Determine the type of expressions composed by these variables and fill in each blank with $\tau \in$ {int, int*, int**, int***,...} if the expression is semantically valid, otherwise write "invalid".

typeof(a)	typeof(b)	typeof(c)	typeof(a+b)	typeof(b-c)	typeof(*(a-b+c))	typeof(&a - &b)
int	int	int*				
int	int*	int**				
int***	int***	int				

4. *Exercise.* Draw the stack as it appears when a function f has called a function g, and g has finished its prologue and is currently executing its body. Assume that the callee is responsible for saving as much as it can, and that the caller pushes arguments from right to left.

5. In this question, we add **for loops** to WLP4, with some restrictions for the purpose of simplicity. A WLP4 for loop consists of at most 1 assignment as the initialization, exactly 1 condition, and at most 1 assignment as the after-thought. Also, any variables being initialized during initialization have to be previously declared, like in any other WLP4 program. Here is one valid example:

```
int a = 0;
...
for (a = 5 ; a < 8 ; a = a + 1){
    println(a);
}
```

This fragment of code will print 5, 6, 7 on separate lines. Now, to support this feature, complete each of the following sub-problems:

(a) List the changes we need to make to the scanner in order to add this feature.

(b) Give the production rules required by the parser to make this change. Indicate any new terminals/non-terminals you introduce. However, you may not change the existing WLP4 grammar.

(c) Describe any new required semantic rules.

(d) In pseudocode, write the code generation function to support this feature.

6 Optimization and Memory Management

1. *Exercise.* What are Constant Folding and Propagation, Dead code Elimination, and Register Allocation? How do they optimize produced assembly code?
2. List 2 advantages and 2 disadvantages of using implicit (automatic) memory management.
3. You are given with a heap with variable-sized allocations. Assume that each integer takes exactly 1 word, and the size of our heap is 40 words.

   Now, consider the following C++ code segment that consists of a sequence of allocations and deallocations:

```
int *a = new int [5];
int *b = new int [5];
int *c = new int [10];
int *d = new int [5];
int *e = new int [15];

delete a [];
delete b [];
delete d [];
a = new int [3];
b = new int [7];
d = new int [5];
```

   Name the 2 strategies you have learned to allocate memory, and for each strategy, indicate whether they would succeed or fail.

7 Linking, Loading, and MERL

1. *Exercise.* The following stages describes the process of taking a high-level language program and running it on a computer. Place these stages in the right order: Executing, Loading, Compiling, Linking, Assembling. List all the possible output between the stages, if there are any.
2. Let M_1 and M_2 be two MERL files and M_3 be the MERL file after merging M_1 and M_2. For the following statements, indicate whether it is true or false, explain your choice.
   (a) M_3 may contain some ESR entries.
   (b) The number of ESD entries in M_3 is equal to the number of ESD entries in M_1 plus the number of ESD entries in M_2.
   (c) The number of REL entries in M_3 is equal to the number of REL entries in M_1 plus the number of REL entries in M_2.
   (d) The number of ESD entries in M_2 must equal to the number of ESR entries in M_1 in order to link them.

3. Suppose we have two merl files m1.merl and m2.merl. We have linked them together, in that order, to produce m.merl (*linker* m1.merl m2.merl > m.merl). We have used the provided CS241 tool *printmerl* to see the contents either merl file. We have called *printmerl* on m.merl but some parts are missing. Your task is to fill all the blanks according to what the linking algorithm would have produced. For non-relocation entries in the footer, also write the symbol encoded by the entry in the third column.

```
$ printmerl < m1.merl
cookie   10000002
length         a0
clen           58
0000000c     1814
00000010  3c4e822
00000014 afdffffc
00000018     f814
0000001c        0
00000020 8fff0000
00000024 8fff0000
00000028 8fff0000
0000002c  3dff022
00000030   600009
00000034     f814
00000038        0
0000003c 8fff0000
00000040 8fff0000
00000044 8fff0000
00000048  3dff020
0000004c 8fdffffc
00000050  3e00008
00000054        4
ESR            1c def
ESR            38 def
ESD            54 ghi
```

```
$ printmerl < m2.merl
cookie   10000002
length         50
clen           18
0000000c       14
00000010  3e00008
00000014        0
REL             c
ESR            14 ghi
ESD             c def
```

```
$ printmerl < m.merl
cookie   10000002
length         b4
clen           __
0000000c     1814
00000010  3c4e822
00000014 afdffffc
00000018     f814
0000001c        __
00000020 8fff0000
00000024 8fff0000
00000028 8fff0000
0000002c  3dff022
00000030   600009
00000034     f814
00000038        __
0000003c 8fff0000
00000040 8fff0000
00000044 8fff0000
00000048  3dff020
0000004c 8fdffffc
00000050  3e00008
00000054        4
00000058        __
0000005c  3e00008
00000060        __
---             --
---             --
---             --
---             --
---             --
---             --
```

4. Write the MERL footer and header for the following MIPS program. Your answer must be written using a sequence of hexadecimal values and you cannot add labels to the code.

```
        .import g
        .export f
f:
        sw $31, -4($30)
        sub $30, $30, $4
loop:
        slt $3, $1, $2
        beq $3, $0, end
        lis $3
        .word g
        jalr $3
        lis $13
        .word loop
        jr $13
end:
        add $30, $30, $4
        lw $31, -4($30)
        jr $31
```