Sample midterm questions and study suggestions

Describe in English the strings matched by the following regexps:

1. 0 | [1-9] [0-9]*
2. [a-hj-z]*i?[a-hj-z]*
3. [a-zA-Z] [a-zA-Z_]*

Write regular expressions corresponding to the following descriptions:

1. words that have exactly 5 characters
2. lines that consist of exactly 2 space-separated words (spaces can go before, after, or in between). The beginning of a line is matched by the regexp ^ and the end of a line by $.
3. Scary C assignments within if statements. C assignments use =, C comparisons use ==, C assignments are expressions so can be used in if statements, but that is almost always a typing mistake.

Write a CFG that matches palindromes (words that are the same backwards as forwards). To make it a bit easier, assume that the only legal characters are 'a', 'b', 'c', and 'd'.

Suppose you have a JavaCC input file that contains something like this:

```
List<AST> Statements() {
  List<AST> sl,;
  AST s;
}
{
  sl=Statements() s = Statement()
                { sl.add(s); }
|
  { sl = new ArrayList<AST>(); }
}
```

When you attempt to process this with JavaCC you get an error like:

Error: left recursion detected: "Statements … -> Statements"

1. What does this mean?
2. What is the general strategy to fix it?
3. Give a new JavaCC input that accepts the same language and builds the same AST.

Describe the new type checking rules that you would need if we extended the Functions language to allow a new type "long" which is like an "int" but uses more bits to store the value. The general rule is that it is legal to promote a value from an "int" to a "long", but it is an error to "demote" a "long" value to an "int".

Convert the following IR trees to "canonical" form (no ESEQ nodes, CALL nodes only in EXP or MOVE(TEMP, CALL(…)).

1.
```
MOVE(
  TEMP t312 <-
  ESEQ(
    SEQ(
      MOVE(
        TEMP t313 <-
        CONST 0),
      CMOVE(LT,
        CONST 1,
        CONST 2,
        TEMP t313 <- CONST 1)),
    TEMP t313))
```

2.
```
MOVE(
  TEMP(t311 <-
    CALL(
    NAME(__equals),
    CALL(
      NAME(__succ),
      TEMP t316,
      TEMP t312),
    CALL(
      NAME(__pred),
      TEMP t316,
      TEMP t314))))))
```

What are the three properties that are required of all basic blocks?

Modify the following code to identify each of its basic blocks:

```
List {
  MOVE(TEMP t366 <- TEMP %rdi:%rdi)
  MOVE(TEMP t367 <- TEMP %rsi:%rsi)
  MOVE(TEMP t371 <- TEMP %rbx:%rbx)
  MOVE(TEMP t372 <- TEMP %r12:%r12)
  MOVE(TEMP t373 <- TEMP %r13:%r13)
  MOVE(TEMP t374 <- TEMP %r14:%r14)
  MOVE(TEMP t375 <- TEMP %r15:%r15)
  MOVE(TEMP t368 <- BINOP(MUL, TEMP t366,  CONST 2))
  CJUMP(NE,TEMP t367, CONST 0, _L_113, _L_114)
  LABEL _L_113
  MOVE(TEMP t370 <- TEMP t368)
  JUMP(NAME(_L_115))
  LABEL _L_114
  MOVE(TEMP t370 <- BINOP(MUL, TEMP t368,  CONST 2))
```

```
  LABEL _L_115
  MOVE(TEMP t369 <- TEMP t370)
  MOVE(TEMP %rax:%rax <- TEMP t369)
  MOVE(TEMP %rbx:%rbx <- TEMP t371)
  MOVE(TEMP %r12:%r12 <- TEMP t372)
  MOVE(TEMP %r13:%r13 <- TEMP t373)
  MOVE(TEMP %r14:%r14 <- TEMP t374)
  MOVE(TEMP %r15:%r15 <- TEMP t375)
}
```

What is the rule for CJUMP nodes that trace scheduling must satisfy?

Working with the previous example, perform trace scheduling.

The "Norms" processor architecture has the following instructions:

```
1. imove    $const, dst        # move const to dst reg
2. move     src, dst           # register to register copy
3. mmove    (src), (dst)       # memory to memory copy
4. add      src1, src2, dst    # add 2 src regs and assign to dst reg
5. sub      src1, src2, dst    # sub src2 from src1 and assign to dst reg
6. mul      src1, src2, dst    # mul 2 src regs and assign to dst reg
7. load     (src), dst         # load from memory to dst reg
8. store    src, (dst)         # store from src reg to memory
```

Draw tree patterns corresponding to each of these machine instructions.

Given the following IR tree, give two different tilings of it.

```
MOVE(
    MEM(
        BINOP(PLUS,
          TEMP %rbp,
          CONST 16)) <-
    MEM(
        BINOP(PLUS,
          TEMP %rbp,
          BINOP(PLUS,
            TEMP t000,
            BINOP(MUL,
                TEMP t001,
                CONST 8)))))
```

Write down the assembly code corresponding to each tiling.