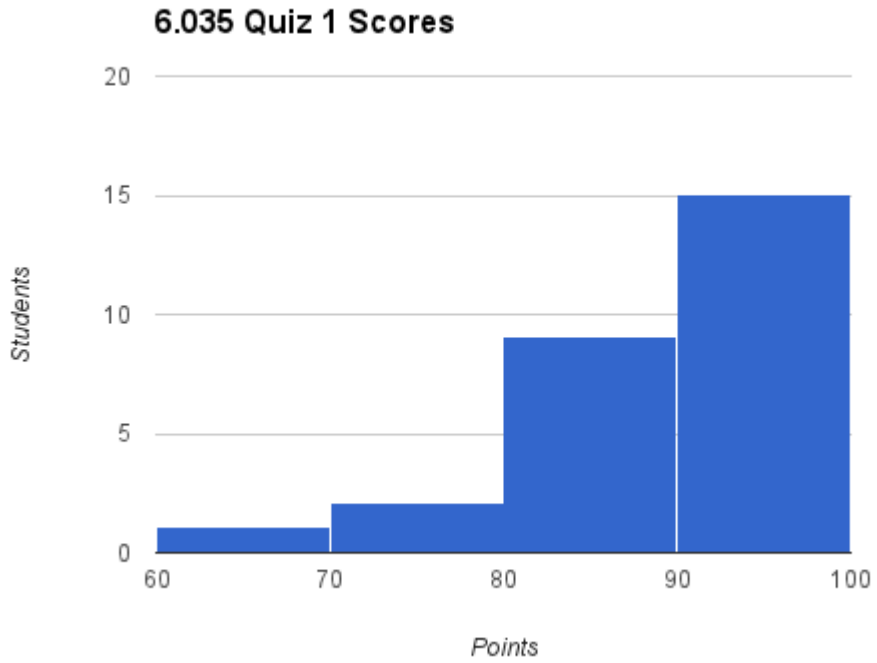
Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.035 Fall 2014

Test I Solutions



I Regular Expressions and Context-Free Grammars

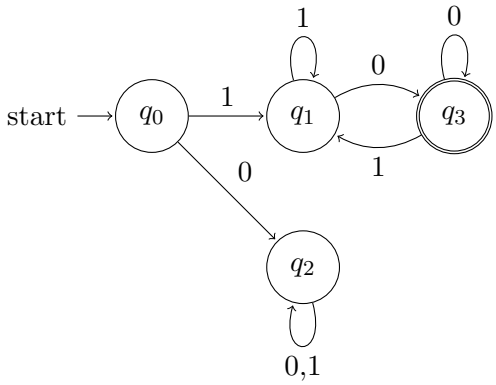
The languages in Questions 1, 2, and 3 are over the alphabet $\{0, 1\}$.

1. [6 points]: Write a regular expression that recognizes the language of sequences of zeros and ones such that the number of ones is a positive even number.

Solution: $0^*10^*10^*(10^*10^*)^*$

2. [7 points]: Draw a state diagram of a DFA that recognizes the language of non-empty strings that start with 1 and end with 0.

Solution:



We also accepted answers without the explicit error state q_2 (when the DFA is stuck on '0' in q_0).

3. [7 points]: Write a context-free grammar that recognizes the language 0^n1^n $(n > 0)$.

Solution:

$$S \to 0S1$$
$$S \to 01$$

We accepted answers for both $n > 0$ and $n \geq 0$ (with $S \to \varepsilon$).

II Hacking Context-Free Grammars

Consider the following grammar:

$$\begin{array}{lrcl} \textbf{1:} & S & \rightarrow & S; S \\ \textbf{2:} & S & \rightarrow & \texttt{while}(V)\ S \\ \textbf{3:} & S & \rightarrow & V = C \end{array}$$

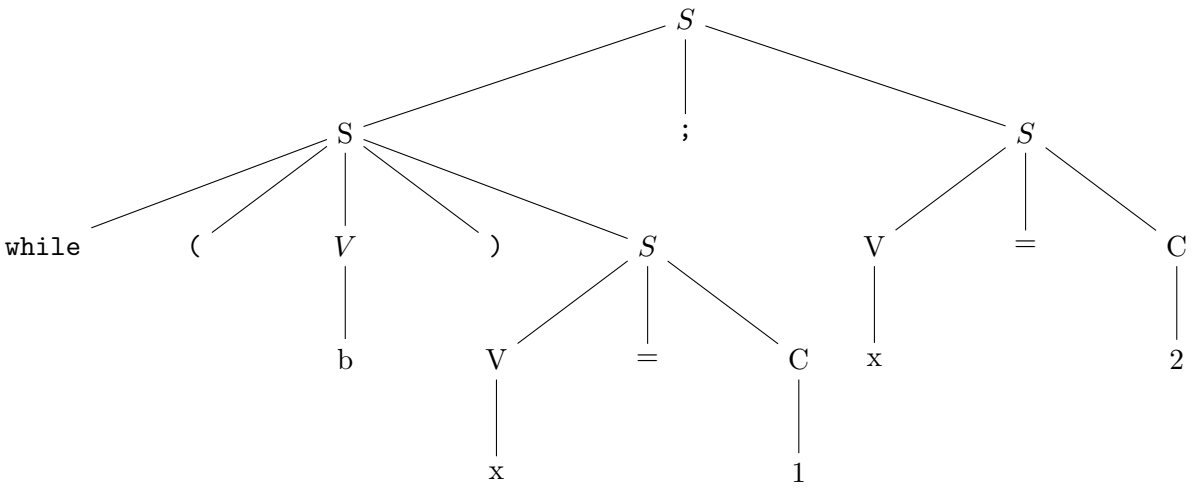
Here, V denotes string identifiers that represent variables and C denotes integer constants.

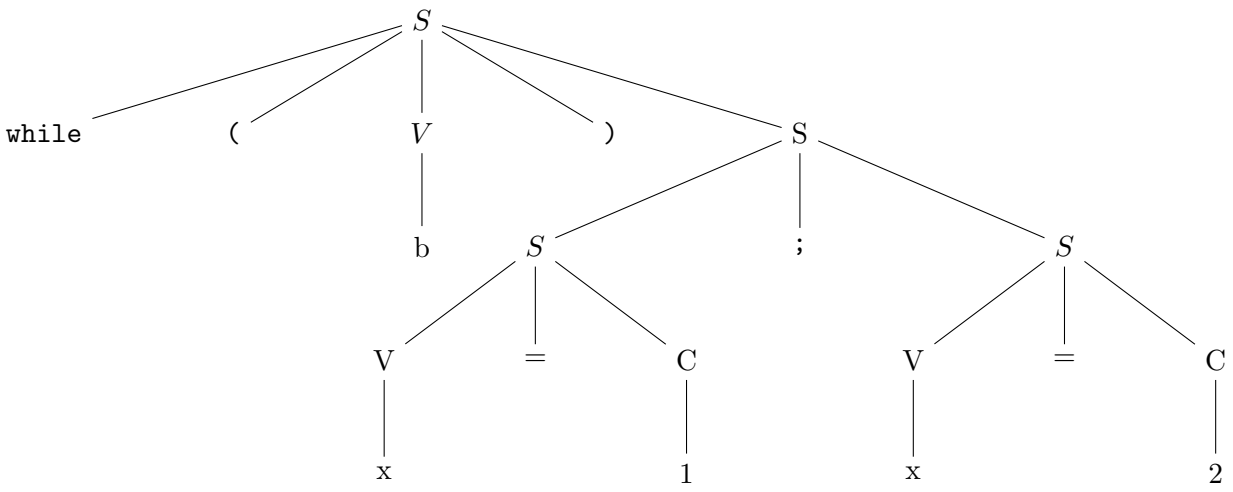
4. [10 points]: Show that the grammar is ambiguous by providing:

A. A sentence in the language with two parse trees.

B. The two parse trees.

Solution: Examples include: `while (b) x = 1; x = 2` or `x = 1; x = 2; x =3`




S
while
(
V
)
S
b
S
;
S
V
=
C
V
=
C
x
1
x
2

5. [10 points]: Consider the following sentence (^{1} and ^{2} denote two locations):

`y = 6; while (x)`^{1} `x = 5`^{2}

Assume you are working with a shift-reduce parser for this language.

A. Draw the parser's stack after shifting the token ')' (location 1) and after shifting the token '5' (location 2).

B. What will be the next two actions after shifting the token '5'? If the action is shift, then specify the token. If the action is reduce, then specify the production rule.

Solution:

Stack at location 1: S, '`;`', '`while`', '`(`', V, '`)` (top)

Stack at location 2: same as previous, and also V, `'='`, `'5'` (top).

The next two steps are $REDUCE(C)$ and $REDUCE(V)$.

We also accepted the situation in which the constant reduction is done immediately (and C is put on top of the stack instead of `5`). The next two steps are $REDUCE(V)$ and $REDUCE(S \rightarrow V = C)$.

6. [10 points]: For the same language, provide a grammar that is not ambiguous.

Solution:

$$
\begin{aligned}
S &\rightarrow S; W \\
S &\rightarrow W \\
W &\rightarrow \texttt{while}(V)\ W \\
W &\rightarrow V = C
\end{aligned}
$$

III Symbol Tables and Semantic Analysis

Consider the following Decaf program:

```
callout printf;

int A[100];

    // Reads 'len' elements from input file to the array 'A' and sorts the array
void read_and_sort(int len) { /* ... */ }

    // Binary search for the element 'value'
int bins(int value, int len) {
    int left;
    int right;
    left = 0; right = len - 1;
    while (left < right) {
        int mid;
        mid = (left + right)/2;
        if (value ==  A[mid]) { return mid; }
        else { if (value > A[mid]){ left = mid + 1; }
        else { right = mid - 1; } }

        //--- LOC 1 ---//
    }

    return -1;
}

void main () {
     int l;
     int res;

     l = 100;
     read_and_sort(l);
     res = bins(42, l);

     if (res >= 0) { printf ("Yup!\n"); }
     else { printf ("Nope!\n"); }
}
```

7. [15 points]: Complete the entries of the symbol table for the program at the position **LOC 1**.

Function Name	IsCallout?	Return Type	Parameter Types
printf	yes	int	-
read_and_sort	no	void	int
bins	no	int	int, int

Figure 1: ***Function Symbol Table:*** "IsCallout?" is yes if the function is a callout, no otherwise. Return type ∈ {int, bool, void}, parameter type ∈ {int, bool}.

Global Level	Variable Name	IsParameter?	Type	Length
	A	no	int[]	100

bins Level 1	Variable Name	IsParameter?	Type	Length
	val	yes	int	0
	len	yes	int	0
	left	no	int	0
	right	no	int	0

bins Level 2	Variable Name	IsParameter?	Type	Length
	mid	no	int	0

Figure 2: ***Variable Symbol Table:*** "IsParameter?" is yes if the variable is a parameter, no otherwise. Variable's type ∈ {int, bool, int[], bool[]}. Length is 0 for scalars, or a positive number for arrays. Table has multiple scope levels: for global variables (global level), function parameters and locals declared at the function block level (level 1), and locals declared at the levels of the function's internal blocks (levels greater than 1).

8. [5 points]: Assume that in the function `bins` we renamed the parameter "`value`" to "`bins`". Then the new definition of the function `bins` looks like:

```
int bins(int bins, int len) {
  // ... //
      if (bins == A[mid]) // ...
  // ... //
}
```

Is this new program still a legal Decaf program? Explain your answer.

Solution: Yes, it is legal. The function's parameter (which is in a local scope) shadows the function name.

9. [5 points]: Assume that we declared a local variable "`int A;`" at the beginning of the body of the function `bins`. Is this new program still a legal Decaf program? Explain your answer.

Solution: No, it is not legal. Note that the local scalar variable shadows the global array variable. However, the array access `A[mid]` is not defined for the scalar variables.

10. [5 points]: Would the program be legal if instead we declared the local variable "`int A;`" at the beginning of the body of the function `main`? Explain your answer.

Solution: Yes, it would be legal. The global array A is not used in the `main` function and it does not affect the body of the `bins` function.

IV Semantic Analysis

11. [20 points]: In this exercise, you will implement a function that checks whether an expression (e.g., a condition of an `if` or a `while` statement) is of type `bool`.

IMPORTANT NOTE: We provided two skeletons for implementation, one using pattern matching, another using visitor pattern. Please choose **only one** skeleton in which to implement the analysis. **Write here which skeleton you chose:** _____

Pattern matching version: Assume you are given the function `get_type(id: string)`, which returns the type of the declared identifier `id`. The type of the identifier and the return value of `get_type` can be "`int`", "`bool`", or "`err`" (when the identifier is not declared).

The type `Expr` is the parent class for all arithmetic, relational, and logical operators. We use several representative subtypes in the matching rule below. We omit their definitions, but they should be clear from the context.

Assume you are given the function `is_expr_int(expr)` that checks if an expression's type is integer.

Fill in the lines in the pattern matching statement so that the function `is_expr_boolean` checks whether an expression has the type `bool`:

```
def is_expr_boolean (expr : Expr) : Boolean = expr match {

        case IntConst (intValue) => return false;


        case BoolConst (boolValue) => return true;


        case Variable (strVarName) => return "bool" == get_type(strVarName);


        case Add (expr1, expr2) => return false;


        case LessThan (expr1, expr2) =>
                return is_expr_int(expr1) && is_expr_int(expr2);

        case Or (expr1, expr2) =>
                return is_expr_boolean(expr1) && is_expr_boolean(expr2);

        // ...
    }
```

Visitor pattern version: Assume you are given the function `Util.get_type(String id)`, which returns the type of the declared identifier `id`. The type of the identifier and the return value of `get_type` can be "`int`", "`bool`", or "`err`" (when the identifier is not declared).

The class `Expr` is the parent class for all arithmetic, relational, and logical operators. It has an overloaded method `accept(Visitor visitor)` that calls the appropriate visitor. Each subclass implements the method `accept`. We use several representative subclasses of the class `Expr` below. We omit their full definitions, but the relevant fields are passed as arguments to the visitor functions.

Assume you are also given the visitor class `IsIntExprVisitor` (with a default constructor) that checks if an expression's type is integer.

Fill in the lines in the `IsBoolExprVisitor` so that it checks whether an expression's type is `bool`:

```
class IsBoolExprVisitor { // ...

   bool visitIntConst (int intValue) { return false; }

   bool visitBoolConst (bool boolValue) { return true; }

   bool visitVariable (String varName) { return "bool" == get_type(varName); }

   bool visitAdd (Expr expr1, Expr expr2) { return false; }

   bool visitLessThan (Expr expr1, Expr expr2)
       {
            IsIntExprVisitor iev = new IsIntExprVisitor();
            return expr1.accept(iev) && expr2.accept(iev);
       }

   bool visitOr (Expr expr1, Expr expr2)
       {
            return expr1.accept(this) && expr2.accept(this);
       }

   // ...
}
```