CS 310, Assignment 5

Answers

1. Consider the following context-free grammar with start symbol S, nonterminals $\{S, A, B, C\}$, and terminals $\{0, 1\}$:

$$S \to A0 \qquad A \to B \qquad A \to C1 \qquad B \to 1 \qquad B \to \varepsilon \qquad C \to 0$$

(a) Compute *all* the sets First and Follow necessary to implement a recursive decent parser for this grammar. However, do not list any unnecessary such a set.

Answer: We do not need to compute any sets for S (there is a single rule for that symbol).

A has two rules and it is also the case that $A \Rightarrow^* \varepsilon$ so we need:

$$\text{First}(B) = \{1, EOS\} \qquad \text{First}(C1) = \{0\} \qquad \text{Follow}(A) = \{0\}$$

B also has two rules, one of them an ε-rule, so we need:

$$\text{First}(1) = \{1\} \qquad \text{Follow}(B) = \{0\}$$

We do not need to compute any sets for C either (for again, there is a single rule for that nonterminal).

(b) Investigate all the combinations of sets First and Follow that are involved in the implementation of a recursive descent parser for this grammar. Explain how these combination make the grammar suitable or unsuitable (as the case might be) for recursive descent parsing.

Answer: The grammar is not suitable for recursive descent parsing since $\text{First}(C1) \cap \text{Follow}(A) = \{0\} \neq \emptyset$. In other words, when the next token is 0 we have no idea which rule to use to rewrite A.

As far as B is concerned we are fine. When we see a 0 we know we should rewrite B using the first rule, and a 1 directs us to the second rule. Unfortunately the parser is for the whole grammar not just for parts of it, so overall we cannot use this grammar to implement a recursive descent parser.

---

2. What should the pre-condition P be in each of the following correctness statements for the statement to be an instance of Hoare's assignment axiom scheme?

(a) P `{ x = 1; } x <= 2`

(b) P `{ y = x - y; } y*y > 5`

(c) P `{ i = i - k; } ForAll (i=0; i<10) i+k > 0`

(d) P `{ i = i - k; } Exists (k=0; k<i) k+m > i`

ANSWER:

(a) `1 <= 2` or `true`

(b) `(x-y)*(x-y) > 5` (note the parentheses)

(c) `ForAll (i=0; i<10) i+k > 0` (no change, since all the occurrences of `i` refer to the bound variable; this can be verified by renaming the bound variable `i`)

(d) `Exists (p=0; p<i) p+m > i-k` (the bound variable `k` was renamed because a free variable with the same name is introduced by the substitution; without such a renaming there will be two `k` variables, one bound and another free)

---

3. Add all the intermediate assertions and so produce the proof tableau for the following statements. If a statement is not valid then include in your respective tableau a pre-condition that is just strong enough to make the statement valid.

(a)
```
ASSERT(true)
m = 1;
n = 1;
n = a-b;
ASSERT(m*n > 0)
```

ANSWER: We have the following tableau:

```
ASSERT(a > b)       // 4 - math
ASSERT(1*(a-b) > 0) // 3 - assignment
m = 1;
ASSERT(m*(a-b) > 0) // 2 - assignment
n = 1;
ASSERT(m*(a-b) > 0) // 1 - assignment
n = a-b;
ASSERT(m*n > 0)
```

The original precondition `true` is not stronger than `a>b` and so the statement is not valid. The weakest pre-condition that makes the statement valid is given in the tableau.

(b)
```
ASSERT(x == y*(y+1))
y = y + 1;
x = x + 2*y;
ASSERT(x == y*(y+1))
```

ANSWER: The statement is valid:

```
ASSERT(x == y*(y+1))          // 6 - math, qed
ASSERT(x == (y+1)*(y+1-1))    // 5 - assignment
y = y + 1;
ASSERT(x == y*(y-1))          // 4 - math
ASSERT(x == y*(y+1-2))        // 3 - math
ASSERT(x == y*(y+1) - 2*y)    // 2 - math
ASSERT(x + 2*y == y*(y+1))    // 1 - assignment
x = x + 2*y;
ASSERT(x == y*(y+1))
```

(c)
```
ASSERT(false)
y = 1;
ASSERT(x+y<=0)
```

ANSWER: The statement is (vacuously) valid because there is no possible input data that makes the precondition true. There is no need to construct the full tableau.

(d)
```
ASSERT(true)
if (x >= y) x = x + 1; else y = x - 1;
z = y - 1;
ASSERT(z < y < x)
```

ANSWER: If we assume that x and y are integers then the statement is valid:

```
ASSERT(true)                    // 10 - if, qed
if (x >= y)
    ASSERT(true && y <= x)      // 9 - strengthening
    ASSERT(y <= x)              // 8 - math
    ASSERT(y < x + 1)           // 7 - assignment
    x = x + 1;
    ASSERT(y < x)               // 3 - if
else
    ASSERT(true && !(x >= y))   // 6 - strengthening
    ASSERT(true)                // 5 - math
    ASSERT(x - 1 < x)           // 4 - assignment
    y = x - 1;
    ASSERT(y < x)               // 3 - if
ASSERT(y < x)                   // 2 - math
ASSERT(y - 1 < y < x)           // 1 - assignment
z = y - 1;
ASSERT(z < y < x)
```

If on the other hand `x` is a floating point number then Inference #8 cannot be made and therefore the tableau cannot be completed.

The most immediate answer is therefore that the statement is valid under the following declarative interface:

```
int x,y;
```