CMSC 430 : Midterm 1 Problems Solutions

---

1 a) Syntax-directed translation can perform actions not specified
in the grammar, both for analyzing the input program (type-checking)
and generating output (symbol tables, AST).

b) Context-sensitive parsing can decide whether to apply a production
for a nonterminal based on input surrounding the nonterminal.
Context-free parsing applys a production based solely on the
nonterminal and the result of the production.

c) context-free grammar (least powerful)
context-sensitive grammar
attribute grammar
syntax-directed translation (most powerful)

d) Synthesized attributes may be calculated only from attributes of children.
Inherited attributes may also be calculated from attributes of parents
or siblings.

e) E := E PLUS E { E.val = E1.val + E2.val; } // val is synthesized

decl := typeSpec id { id.type = typeSpec.type; } // type is inherited
typeSpec := INT { typeSpec.type = INT; }

---

2 a) pointer(char) X array(0..3, int) -> int

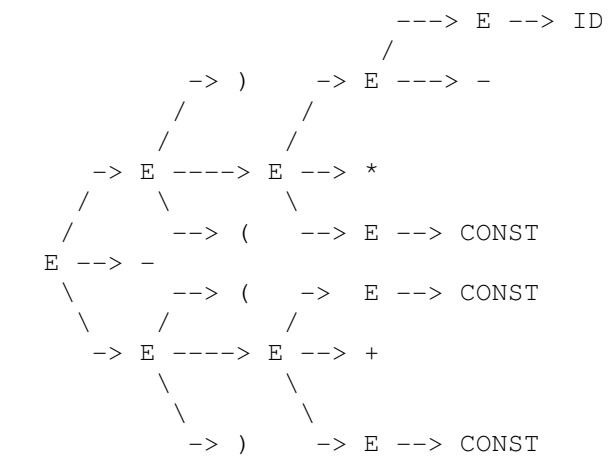
b) array(0..99, (int X int))

c) int X (int X int) -> pointer((int X int))

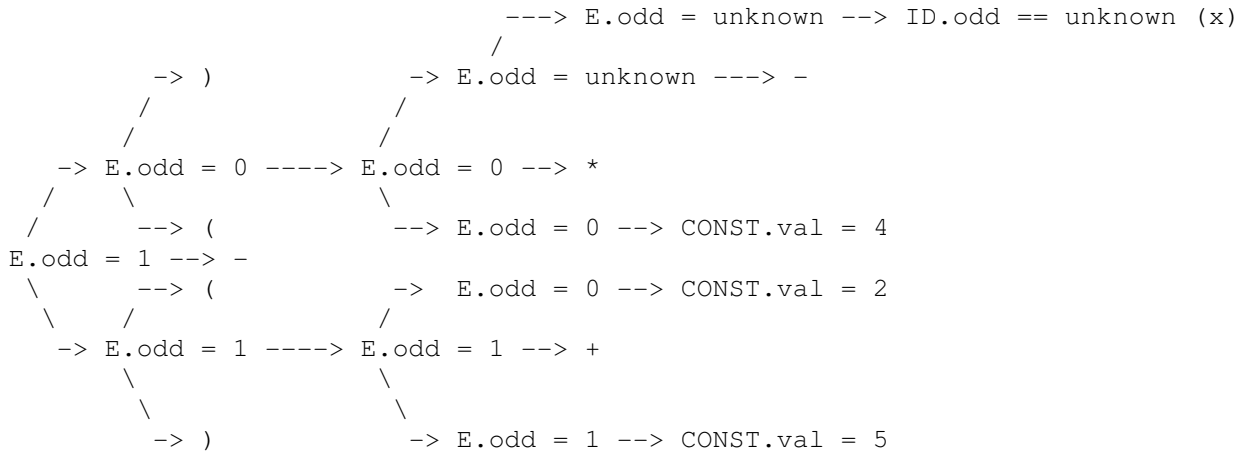
```
3 a)  E  :=  CONST    { E.odd = (CONST.val % 2); } // 0 if even, 1 if odd
       |   ID       { E.odd = unknown; }
       |   E + E    { if ((E1.odd == unknown) || (E2.odd == unknown))
                        E.odd = unknown;
                      else
                        E.odd = (E1.odd + E2.odd) % 2;
                    }
       |   E - E    { if ((E1.odd == unknown) || (E2.odd == unknown))
                        E.odd = unknown;
                      else
                        E.odd = (E1.odd + E2.odd) % 2;
                    }
       |   E * E    { if ((E1.odd == 0) || (E2.odd == 0))
                        E.odd = 0;   // even * X = even for all X
                      else if ((E1.odd == unknown) || (E2.odd == unknown))
                        E.odd = unknown;
                      else //  assert(E1.odd && E2.odd)  { both must be odd }
                        E.odd = 1;
                    }
       |   ( E )    { E.odd = E1.odd; }
       |   - E      { E.odd = E1.odd; }
```

b) (5 + 2) - (4 * - x)

Parse Tree



Annotated Parse Tree (with values of attributes)

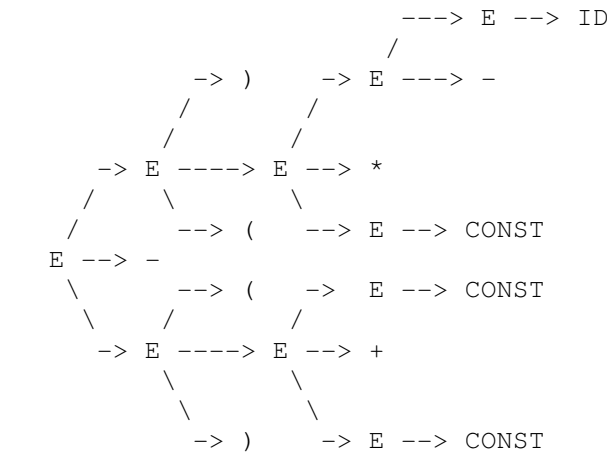


---

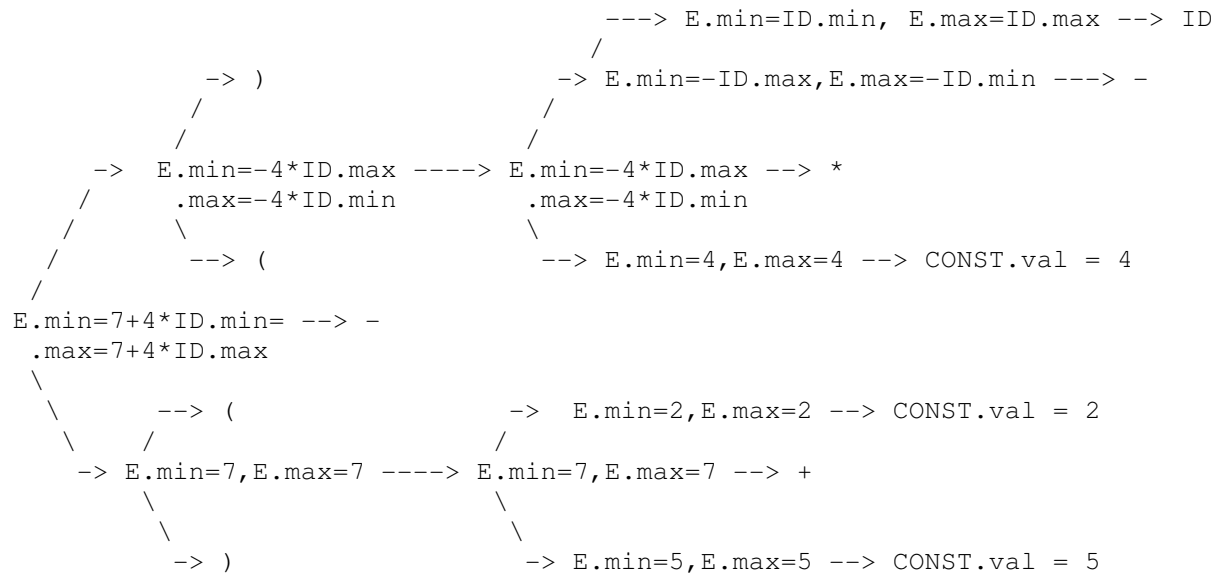
4 a) E := CONST { E.min = CONST.val; E.max = CONST.val; }
| ID { E.min = ID.min; E.max = ID.max; }
| E + E { E.min = E1.min+E2.min; E.max = E1.max+E2.max; }
| E - E { E.min = E1.min-E2.max; E.max = E1.max-E2.min; }
| E * E { t1 = E1.min*E2.min;
t2 = E1.min*E2.max;
t3 = E1.max*E2.min;
t4 = E1.max*E2.max;
E.min = min(t1,t2,t3,t4);
E.max = max(t1,t2,t3,t4);
}
| (E) { E.min = E1.min; E.max = E1.max; }
| - E { E.min = -E1.max; E.max = -E1.min; }

b) (5 + 2) - (4 * - x)

Parse Tree



Annotated Parse Tree (with values of attributes)



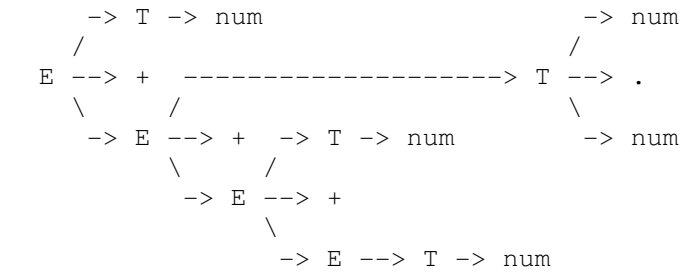
---

5 a) E := E + T if ((E'.type == int) && (T.type == int)) E.type = int;
else E.type = float;
| T E.type = T.type

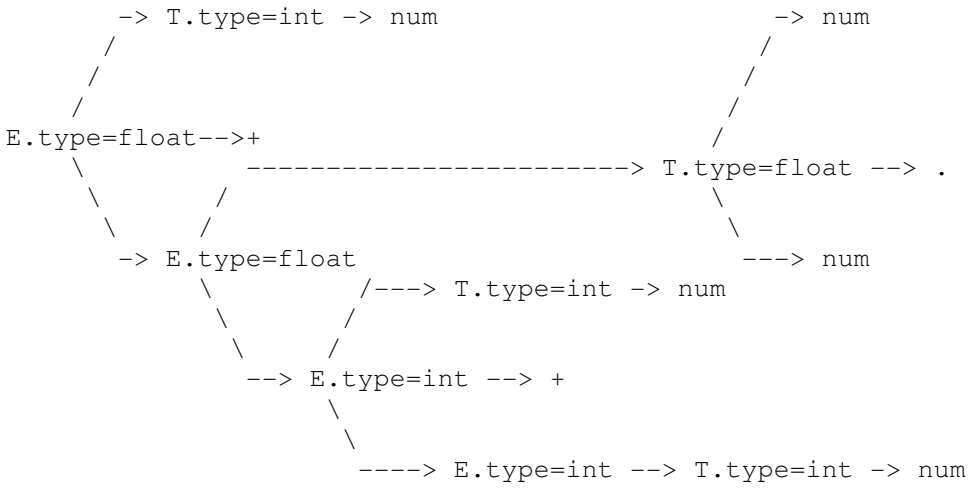
T := num.num T.type = float
| num T.type = int

b) 5 + 4 + 3.2 + 1

Parse Tree



Annotated Parse Tree



---

6 a) Different scopes may be nested within each other, and each occurrence of a variable refers to the closest lexical declaration, working outwards from where the variable is used.

b) Symbol tables can handle nested lexical scoping in many ways. One simple method is to maintain a separate symbol table for each scope, nesting the symbol tables.

c) When using nested symbol tables, a variable can find its matching declaration by looking in symbol table for the current scope, then working out to the symbol tables of enclosing scopes until symbol is found.

d) Example of nested symbol tables