ANTLR exercises

Christopher League*

16 March 2016

Below is a fragment of the grammar for the language Standard ML.. We've done a lot with standard arithmetic expressions, so this is a bit of a departure — we're representing *types.*

```
grammar LittleML;

top : type EOF;

type    : type ID
        | '(' type (',' type)+ ')' ID
        | type '*' type
        | <assoc=right> type '->' type
        | '{' record? '}'
        | '(' type ')'
        | TVAR
        | ID
        ;

record  : ID ':' type (',' record)?;

ID : [A-Za-z] [A-Za-z0-9]*;
TVAR : '\'' [A-Za-z0-9]+;
WS : (' '|'\t'|'\n')+ -> skip;
```

1. Which of the following expressions is valid according to this grammar? If it's valid, draw the parse tree.

   a. `int32`
   b. `unsigned_int`
   c. `'a`
   d. `'a list`
   e. `int list tree`
   f. `int * float + char`
   g. `int * char tree`
   h. `{}`
   i. `{x:int, y:float}`

j. `()`
k. `(int, char) tree`
l. `('a, 'b) (int * float)`
m. `'a -> 'b`
n. `'a list * int -> int * int`

2. Which of these methods are part of the generated `LittleMLBaseVisitor` class?

   a. `visitTVAR(TVARContext cxt)`
   b. `visitType(TypeContext cxt)`
   c. `visitRecord(RecordContext cxt)`
   d. `visitWS(WSContext cxt)`
   e. `visitOp(OpContext cxt)`

3. Now suppose we label the different alternatives of the `type` rule using the hash notation:

```
type    : type ID                        #TypeApp
        | '(' type (',' type)+ ')' ID    #TypeApp
        | type '*' type                  #TypePair
        | <assoc=right> type '->' type   #TypeArrow
        | '{' record? '}'                #TypeRecord
        | '(' type ')'                   #TypeParen
        | TVAR                           #TypeVar
        | ID                             #TypeName
        ;
```

   Now which of these methods are part of the generated `LittleMLBaseVisitor` class?

   a. `visitTypeRecord(TypeRecordContext cxt)`
   b. `visitType(TypeContext cxt)`
   c. `visitRecord(RecordContext cxt)`
   d. `visitTypeArrow(TypeArrowContext cxt)`

4. When the following type expressions are parsed, what is the order in which the (hash-tagged) visitor methods are called in a standard pre-order traversal? Hint: draw the tree first.

   a. `int list tree`
   b. `'a list * int -> int * int`
   c. `{x:int, y:float}`
   d. `{x:int * float, y: 'a list} * d`