TIE-20306 Principles of Programming Languages, autumn 2019

Last change 01.11.2019 09:09.

Project page map

- *Course main page*
- *Project main page*
- *Project environment*
- *How to submit code*
- *Phase 1: lexical analysis*
- *Phase 2: syntax check*
- *Phase 3: syntax tree*
- *Phase 4: semantics and running*

```
This page change history:
2019-10-22: First public version
2019-10-25: Printout of a function call now includes the function name
2019-10-28: Added a note that the order of printouts doesn't have to be the same as in example print
```

Project work, phase 2

Syntax Analysis

Basic idea of syntax check is to read a stream of tokens (coming from the lexer) and decide weather the order of tokens corresponds to a syntax definition or not (syntax error). A BNF defition of a syntax can be checked with so called pushdown automaton. This is what the yacc-tool implements.

Some resources

- Extended BNF
- PLY yacc

TupLang

EBNF

Syntax is defined using a variation of EBNF metasyntax, with this notation:

```
name ::= definition # defines a non-terminal symbol 'name'

a1 a2    # 'a1' is immediately followed by 'a2'

a1 | a2  # 'a1' or 'a2'
[a]    # 'a' or nothing (optional)
{a}    # zero or more times 'a'
(a1 | a2) a3  # 'a1' or 'a2' always followed by 'a3'
```

TupLang syntax in EBNF

Terminal names are the tokens defined in previous phase. Note that the PLY tool only understands BNF, not extended BNF, so often this EBNF cannot be directly used in your code.

NOTE: We ended up making the Tupl language smaller than originally intended, so some of the tokens defined in phase 1 ended up being unused. This means that by default PLY will give you some warnings about unused tokens. You can freely ignore those warnings, or you can edit the lexer and remove those tokens (make sure you remove them from the lexer you copied over to phase 2!).

```
program ::= {function_or_variable_definition} return_value DOT

function_or_variable_definition ::=
    variable_definition | function_definition

function_definition ::= DEFINE funcIDENT LSQUARE [formals] RSQUARE
                        BEGIN
                        {variable_definitions}
```

```
                         return_value DOT
                         END DOT

formals ::= varIDENT {COMMA varIDENT}

return_value ::= EQ simple_expression | NOTEQ pipe_expression

variable_definitions ::= varIDENT LARROW simple_expression DOT
                       | constIDENT LARROW constant_expression DOT
                       | tupleIDENT LARROW tuple_expression DOT
                       | pipe_expression RARROW tupleIDENT DOT

constant_expression ::= constIDENT
                      | NUMBER_LITERAL

pipe_expression ::= tuple_expression {PIPE pipe_operation}

pipe_operation ::= funcIDENT
                 | MULT
                 | PLUS
                 | each_statement

each_statement ::= EACH COLON funcIDENT


tuple_expression ::= tuple_atom {tuple_operation tuple_atom}

tuple_operation ::= DOUBLEPLUS

tuple_atom ::= tupleIDENT
     | function_call
     | LSQUARE constant_expression DOUBLEMULT constant_expression RSQUARE
     | LSQUARE constant_expression DOUBLEDOT  constant_expression RSQUARE
     | LSQUARE arguments RSQUARE


function_call ::= funcIDENT LSQUARE [arguments] RSQUARE

arguments ::= simple_expression {COMMA simple_expression}



atom ::= function_call | NUMBER_LITERAL | STRING_LITERAL | varIDENT |
         constIDENT | LPAREN simple_expression RPAREN |
         SELECT COLON constant_expression LSQUARE tuple_expression RSQUARE

factor ::= [MINUS] atom

term ::= factor {(MULT | DIV) factor}

simple_expression ::= term {(PLUS | MINUS) term}
```

Syntax check in PLY

Implement the syntax checking using PLY rules. In predefined cases print out what non-terminal was accepted.

PLY documentation has a lot of examples. A simple but fully functional example snippet is provided by the course staff also.

In following cases print out the non-terminal name when it has been recognized (in the PLY action/function):

```
variable_definition( ** variable name **)
constant_definition( ** const name **)
tuplevariable_definition( ** tuple name **)
pipe_expression
func_definition( ** function name ** )
function_call( ** function name ** )
simple_expression
term
factor
atom( ** value of atom if its one of: NUMBER_LITERAL,
         STRING_LITERAL, varIDENT, constIDENT
      ** ) otherwise print only "atom".
```

When PLY generates the syntax check automation (to file parsetab.py) you can get a warning message:

```
Generating LALR tables
WARNING: 7 shift/reduce conflicts
```

In most cases this default action (shift) is the correct one - what we want to happen. So as a general rule shift/reduce conflicts can be ignored. If you get a reduce/reduce conflict, then something is wrong with your rules and they should be corrected.

Example code and their corresponding expected outputs are available in course repository. DO NOT DEPEND solely on those - write your own testcases so that you can be sure that your program works correctly. **NOTE! The order of printouts doesn't have to be exactly the same as in the example printouts. The order depends on your parser implementation strategies, so ending up with a slightly different order is ok.**

Minimum requirements

In order to pass phase 2, you must implement syntax checking of the grammar above, except that function definitions (`function_definition`), function calls (`function_call`), and things only needed by function definitions/calls don't have to be included in the language for passing with minimum grade. However, implementing them (well) improves your grade for the phase. As a guideline, implement the rest of the syntax first, then add functions last if you have time.

Running the syntax checker

Your program must start when executed in python by: `python3 main.py` in your submission directory. You can assume that this python interpreter has PLY installed into it.

Your program must understand the following command line arguments:

```
usage: main.py [-h] [--who | -f FILE]
  -h, --help            show this help message and exit
  --who                 print out student IDs and NAMEs of authors
  -f FILE, --file FILE  filename to process
```

(See the course snippet on how to implement this using python standard library.)

Errors

If your parser encounters an unknown syntax, it must report:
`"{LINE}:Syntax Error (token:'TOKEN')"`
AND stop the parser execution immediately. These items are available in the error token as: token.lineno and token.value

Required text document

In addition to the code, submit a text document in either plain text (.txt), markdown (.md) or pdf (.pdf) format. It should contain answers to the following questions (again, in the groups **own words**, no copying from other sources):

1. What is syntax analysis and how is it related to other parts in compilation?
2. How is the syntactic structure of the language expressed in the PLY tool? I.e., what parts are needed in the code and how are they related to syntactic rules of the language?
3. Explain in English what the syntax of the following elements mean (i.e. how would you describe the syntax in textual form):
   a. Variable definitions
   b. Function call
   c. Tuple expressions
4. Answer the following based on the syntax definition:
   a. Is it possible to define a "nested" function, i.e. to define a new function inside another function? Why?
   b. Is it syntactically possible to perform arithmetic with strings (`"Hello"+"world"`)? Why?
   c. Is it possible to initialize a variable from a constant (`N<-1. var<-N.`)? Why?

   d. Is it possible to initialize a constant from a variable (`var<-1. N<-var.`)? Why?
   e. Are the following allowed by the syntax: `xx--yy` and `--xx`? Why?
   f. How is it ensured that addition/subtraction are done after multiplication/division?
5. Please mention in the document if you didn't implement functions (i.e. you are ok with passing with the minimum grade).
6. What did you think of this assignment? What was difficult? What was easy? Did you learn anything useful?

Extras

You can also invent your own extra features (new error messages, token value checks etc.). DOCUMENT your additions. Only those features which are mentioned in the text document are considered when grading the submission.

Questions with answers

Nothing at the moment.

General practical questions about the course should be sent to popl@lists.tuni.fi.