TIE-20306 Principles of Programming Languages, autumn 2019

Last change 19.11.2019 16:31.


```
This page change history:
2019-10-31: first public version
2019-11-19: added link to public examples for phase 3, linked to a better spot in PLY documentat
```

Project work, phase 3

Syntax Tree

To process a program source code further we need a data structure for it, the syntax tree. The (first iteration of the) tree is typically collected during the syntax checking. This syntax tree is then the base for all the next operations (semantic checks, optimizations, generating code, etc.)

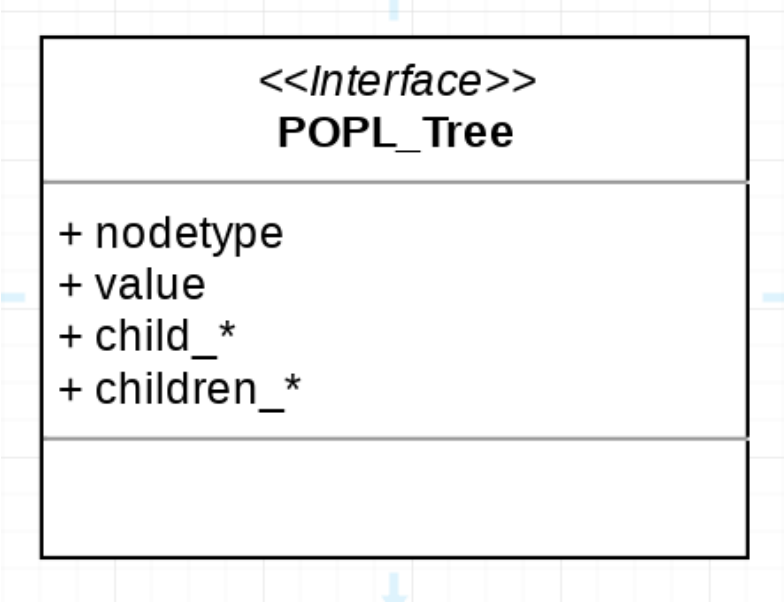
Tree in Python

Use python objects to implement your tree structure. Your submitted program MUST have two distinct phases:

1. Collecting the tree datastructure in PLY
2. Printing out the tree

(I.e., you are not allowed to directly print the tree while creating it in the PLY actions, you must first create it completely and then print it. This is because we need the tree in phase 4.)

Because a tree printing operation is not the core problem in this course, we provide you a fully functional module for it. It expects that each of your tree nodes have attributes: `nodetype` and a optional `value` (which has to be something Python knows how to print). If the node has child nodes then they are named with prefix `child_` (a single node) or `children_` (a list of nodes). The tree_print module automatically recognizes these and prints the child trees as well.



A tree node interface for the tree_print.py module

tree_print-module can print out a tree in this structure in three different output formats:

- Unicode (using Unicode box-drawing characters)
- ASCII (using |, -, and +, use if Unicode doesn't work for you)
- Dot (a graph visualization format used by Graphwiz, see Extras at the bottom of this document for instructions on how to convert it to graphical form)

A fully functional example on how to use tree_print-module can be found here. File `tree_generation.py` has an example tree generation using PLY-rules and objects created from ASTnode-class. This ASTNode-tree is then printed out using the tree_print-module.

NEW!!! The public examples repository now also has the same example programs as in phase 2, and syntax trees generated from them. **PLEASE NOTE** that these are for *example* purposes only, your own syntax trees *do not* have to be the same. There are multiple equally good ways to create a syntax tree, and we don't want to force you to any single form. (You are also allowed to change your syntax tree any way you find useful in phase 4, when you actually start using the tree.)

Tree in PLY

Implement the syntax tree using PLY rules. If there is a syntax error, stop the whole compiler with error message (same as in the previous phase). If there are no errors, then print out your tree using the tree_print-module.

NOTE 0: The PLY documentation has two parser examples. In one (6.1) PLY is used to directly execute (interpret) the code as it is parsed. This is *not* what we want, since we want to create a syntax tree. The second example (6.10) is closer to what we want, we just use the ASTnode class as tree nodes instead of a Python tuple.

NOTE 1: Your tree node objects can contain any attributes you need (and you probably need to add them in phase 4), the tree_print-module does not care about any extras - it only uses the ones defined in it's interface.

NOTE 2: Omitting "useless"/trivial intermediate nodes is allowed. So if you end up in a situation where you would have a node, which serves no other purpose than to have a child node, then you can make those into a single node (i.e. leave the intermediate node out). In the public example, this is done for "cmd_block", for example. Please note however that in some cases you might have to revert that decision during phase 4, if those nodes end up being important, after all.

NOTE 3: Similarly you are allowed to combine a recursive structure into a list of child nodes. I.e., a "list_node" that contains a value and a list node, which contains a value and a list_node, etc, could be implemented in the tree as a list_node that contains a list of value nodes. In the public example, this is done for "program".

Minimum requirements

In order to pass phase 3, you must implement syntax tree generation of the grammar given in phase 2. Again, 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 tree first, then add functions last if you have time.

Running the syntax tree generator

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.)

The default output format should be unicode (this is the default in the tree_print-module). If you want to support the other available formats (ascii and dot), you can add an extra command line argument(s) to select them (as is done in the example).

Errors

Your code should be built on top the previous phase (syntax), so the same errors messages from there should be used here also.

Required text document

In addition to the code, the 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 an (abstract) syntax tree and how is it related to other parts in compilation?
2. How is the syntax tree generated using the PLY tool? I.e., what things are needed in the code and how are they related to syntactic rules of the language and the tree?
3. Explain in English what kind of tree is formed in your code from the following syntactic elements:
    a. Variable definitions
    b. Pipe expressions
    c. Function call (if you implemented it)
4. Answer the following based on the syntax definition and your implementation:
    a. In which cases is it possible in your implementation to end up with a tree with empty child attributes (somewhere in the tree there is a place for a child node (or nodes), but there is none)? I.e., in which situations you end up with tree nodes with child_... attribute being None, or children_... attribute being an empty list?
    b. Are there places in your implementation where you were able to "simplify" the tree by omitting trivial/non-useful nodes or by collecting a recursive repeating structure into a list of child nodes?
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

We actually did not invent any potential extra functionality for this phase. If you have an idea, feel free to implement it (AND remember to document it)

If you want to see a graphical view of your tree, using the GraphViz DOT language, you can either

- copy-paste the dot-description to an online tool, or
- convert the dot-output to an image:
  `python3 tree_generation.py -f test.rom -t dot | dot -Tpng -o test-tree.png`

Questions with answers

Nothing at the moment.

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