Project work – Phase 3

Principles of programming languages

Pinja Mikkonen 99219

1. *What is an (abstract) syntax tree and how is it related to other parts in compilation?*

   An abstract syntax tree is a representation of the general hierarchical structure of the program being compiled. It acts as an intermediary step between syntax analysis and the rest of the compilation process, and has multiple different functions that aid in the execution of code. For example a node of a syntax tree can be used to store useful information about the program, such as line numbers. It can also be used to store information needed to compile the code, such as an operator '+' and it's children – the values that need to be added together.

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?*

   To utilize the PLY tool to create an abstract syntax tree one obviously needs lexical and syntax analyzers done in PLY, as well as a `Node`-object that forms the actual tree. A `Node`-object has a name, value and possibly children. As PLY does syntax analysis it also performs code that creates the tree using `Node`-objects, starting with the root node and adding child nodes as it goes, until the end product is an abstract syntax tree.

3. *Explain in English what kind of tree is formed in your code from the following syntactic elements:*

   (a) *Variable definitions*

   The variable being created gets passed on to the correct definition function depending on it's type. The variable definition function then creates a `_definition` -node, and the name of the variable and it's possible initialization values are added as it's child nodes.

   Multiple concurrent variable definitions get added into `children_vars`-list recursively.

   (b) *For loop*

   For loop is created quite simply. Upon creation a `for_stmt`-node is created, and it gets two `children_`-lists as it's children, one for the range of repetition and another for the statements inside to for loop. After this the range and statement lists get passed on, and all the list components get added recursively.

   (c) *Function call (if you implemented it)*

   Function call gets its own `function_call`-node, with it's name as a child value. If the function call has any arguments they get added recursively into `children_args`-list

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?*

   This shouldn't be possible. In situations where it's possible for a node to have zero or more `child_` or `children_`-attributes I've utilized `if`–statements to add `child_` or `children_` attributes only when needed. This makes the resulting abstract syntax tree look neater in my opinion.

   (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?*

   In many instaces I've simplified the tree structure by simply passing nodes on. For example an argument expression doesn't need it's own node, since it only ever consists of either a scalar expression or a range expression. Hence, it gets passed on without getting a node.

   I've also implemented list structures for all possible children_-attributes, thus simplifying the tree structure.

5. *Please mention in the document if you didn't implement functions/subroutines (i.e. you are ok with passing with the minimum grade).*

   I've implemented functions and subroutines. I've also changed some recursive structures to make the creation of syntax tree easier and fixed an issue with chained comparators a reviewer notified me of.

6. *What did you think of this assignment? What was difficult? What was easy? Did you learn anything useful?*

   This part of the assignment was difficult at the beginning, but got more satisfying to work on durings it's later phases. Implementing the recursive structure needed in lists gave me some headache, but was solved with the help of assistant teachers and some exploration. I also wasn't sure how the tree should be formed, hence my decision to base my solution heavily on the examples we were given. I do feel like I learned more about PLY, and it was quite nice to work on later on though.

   All in all, I enjoyed myself during this assignment.