Compilers

ANTLR Introduction: Lambda-Calculus

3rd Bachelor Computer Science 2016-2017

Brent van Bladel
brent.vanbladel@uantwerpen.be

March 2, 2017

The goal of this assignment is to get familiar with the ANTLR lexer, parser and parse tree in Python.

1 Installation and usage of ANTLR and Python bindings

ANTLR (`http://www.antlr.org`) can be used out-of-the-box as a Java jar package. The latest version can be downloaded from `http://www.antlr.org/download/antlr-4.6-complete.jar`. It requires Java to run.
ANTLR converts a grammar to Python classes using the following command:

```
java -jar antlr-4.5.2-complete.jar -Dlanguage=Python2 MyGrammar.g4 -visitor
```

`MyGrammar.g4` is the text file containing your grammar. Multiple examples can be found here: `https://pragprog.com/titles/tpantlr2/source_code`. Using the `-Dlanguage` flag, the target language can be chosen (*i.e.,* Python2 or Python3). Using the `-visitor` flag, a default parse tree visitor is generated.
In order to manipulate the generated parse in Python, bindings must be installed. This can be done either automatically by executing the pip command:

```
pip install antlr4-python2-runtime
```

Or in case of Python 3:

```
pip install antlr4-python3-runtime
```

Alternatively (if you do not have installation rights), the source code of the bindings can be downloaded from:

`https://pypi.python.org/pypi/antlr4-python2-runtime`

`https://pypi.python.org/pypi/antlr4-python3-runtime`

Place the subfolder `antlr4` in your Python path.
A quick introduction on the Python bindings can be found here: `https://github.com/antlr/antlr4/blob/master/doc/python-target.md`.

See also Appendix A for more details on how to use ANTLR.

2 Lambda-Calculus in ANTLR

- Write a grammar that describes a simple language for the lambda-calculus using the first notation, such as the following example:

  ```
  ((lambda x.lambda y.(x) y) lambda z.z ) p
  ```

  Generate Python files from this grammar using ANTLR as described above. A brief description of the lambda-calculus and both notations can be found in Appendix B.

- In Python, extend the generated listener to print the flattened version (because of the simplicity of this example, we do not convert the more verbose ANTLR parse tree to a more convenient and concise abstract syntax tree). For the example:

  ```
  lambda x, lambda y, x, y, lambda z, z, p
  ```

- In Python, extend the generated visitor to print the expression in the second notation. Visitors can be used to customize the parse tree traversal (default is visitChildren), although you will not need this in this simple example.

  ```
  ((lambda x.lambda y.(x y) lambda z.z ) p)
  ```

Use the python `dir()` function with ANTLR objects as parameter to find out what fields and methods are available, and use the Python API: `http://msdl.cs.mcgill.ca/people/bart/compilers/antlr4-python2-doxygen.zip` (open index.html). For more information on the API, you can use the Java API: `http://www.antlr.org/api/Java/index.html`.

Appendix A: ANTLR Overview

The following steps are required to succesfully complete the assignment:

1. Create a grammar file (.g4 extension). This file contains the grammar of the language you want to parse. Multiple examples can be found here: `https://pragprog.com/titles/tpantlr2/source_code`.

2. Generate the language parser in python using the command:
   ```
   java -jar antlr-4.5.2-complete.jar -Dlanguage=Python2 MyGrammar.g4
   -visitor
   ```
   This will generate python classes that you can use to parse files written according to your specific grammar. Do not edit these files, as they will be overwritten everytime you change your grammar.

3. Now you need to start writing python code:

   (a) You will need a main.py that will handle the input and call the parser. You can find an example here: `https://github.com/antlr/antlr4/blob/master/doc/python-target.md`

   (b) You will need to create a subclass to the generated listener and visitor classes. The reason we use inheritence is to avoid losing your code when generating the python classes again.

4. Create a simple text file with an example of your language (following the rules of your grammar). Run the main.py script and use this example file as input (via command line argument).

Appendix B: Lambda-calculus

The lambda-calculus is a formal and Turing complete language. It was first introduced by Alonzo Church.

The syntax of the lambda-calculus comprises just three sorts of terms. A variable x by itself is a term, the abstraction of a variable x from a term t_1, written $\lambda x.t_1$, is a term; and the application of a term t_1 to another term t_2, written $t_1 t_2$, is a term. These ways of forming terms are summarized in the following grammar.

```
t ::= x | λx.t | t t
```

To avoid confusion, brackets are used to group the terms of an application. There are two ways of notation used: the first one puts brackets around the left term of the application, e.g. $(t)\,t$; and the second puts brackets around both terms, e.g. $(t\,t)$.