CS 5000: Theory of Computation

Assignment 7: Parsing Texts with Deterministic & Probabilistic CFGs

Vladimir Kulyukin
Department of Computer Science
Utah State University

Learning Objectives

1. Context-Free Grammars
2. Probabilistic Context-Free Grammars
3. Parsing Texts with Deterministic and Probabilistic Context-Free Grammars

Introduction

There are two problems on this assignment. You may choose either the 1st one or the 2nd. My original intention was to have only the 2nd problem for this assignment. But then I thought that, since this is a theory of computation class and not an NL class, I should give you an option to have a formal language parsing problem. You can work on both problems to compensate for points you may have lost on the previous assignments. If your scores are perfect though, just choose one and go with it.

Problem 1 (5 points)

Consider the following mathematical operators: +, -, *, /, and %. In the programming language Python, we can define two or three argument functions with one-line bodies that use at most 2 of the above operators as follows:

```
def f(x, y): return x + y
def g(x, y): return x – y
def h(x, y, z): return x + y %  z
def w(x, y, z): return x * y – z
```

Write a CFG grammar for these types of definitions and use the Early Parser to process them. Remember that my implementation of the Early Parser

(https://github.com/VKEDCO/EarlyParser) shown in class requires the input symbols to be separated by space. In other words, when running a parser on the definition of the first function above: you will have to type: “def f (x , y) : return x + y”. You may also restrict the names of the functions to sequences of alphanumeric characters that begin with an alpha character. Another twist is to introduce parentheses in the function bodies to prioritize the operators:

```
def h1(x, y, z): return (x + y) % z
def h2(x, y, z): return x + y % z
```

Depending on how you write your CFG, processing the first definition, should produce only one parse tree whereas processing the second definition may produce two parse trees, one of which is wrong.

Save your grammar in python_cfg.txt and submit in the same file below it all definitions you attempted to parse and the corresponding parse trees.

Problem 2 (5 points)

Consider a drone flying over a grid of landmarks. For example, Figure 1 shows a DJI Phantom 2 hovering over the lower part of the USU Amphitheater with an Android phone on top of it. The phone has a camera that takes pictures of the ground and recognizes distinctive landmarks. For example, we may assume that the drone may recognize letters positioned on the ground in some order. Figures 2, 3, and 4 show some examples.

Suppose that our job is to design an NL interface to the drone. Write 10 or so commands that a human operator, e.g., a search and rescue worker or a forest fire fighter, may want to ask of the drone. Here are some examples:

Which landmark are you over? Fly to A/B/C/D. What is your altitude? Could you fly higher? Fly higher/lower. Can you see A?

Now write a CFG grammar for the Early Parser and parse your sample inputs with it. Install either the Stanford Parser or Open NL and parse your sentences with the parser. Which parser gives you better trees? Which parser would you prefer and why? Submit your CFG in

drone_cfg.txt and the parse trees from the Early Parser and either the Stanford parser or the Open NL parser. State briefly in your submission file why preferred one parser over the other.

The installation links are given in Lecture 16. Figure 4 gives you the list of JAR files I had to install to get the Stanford and Open NL parsers going. The versions will be different, of course. This link https://github.com/VKEDCO/AI/blob/master/NL/PaserDemo.java contains the source for the Stanford Parser demo I did in class.

Figure 1. DJI Phantom 2 Hovering at the USU Amphitheater

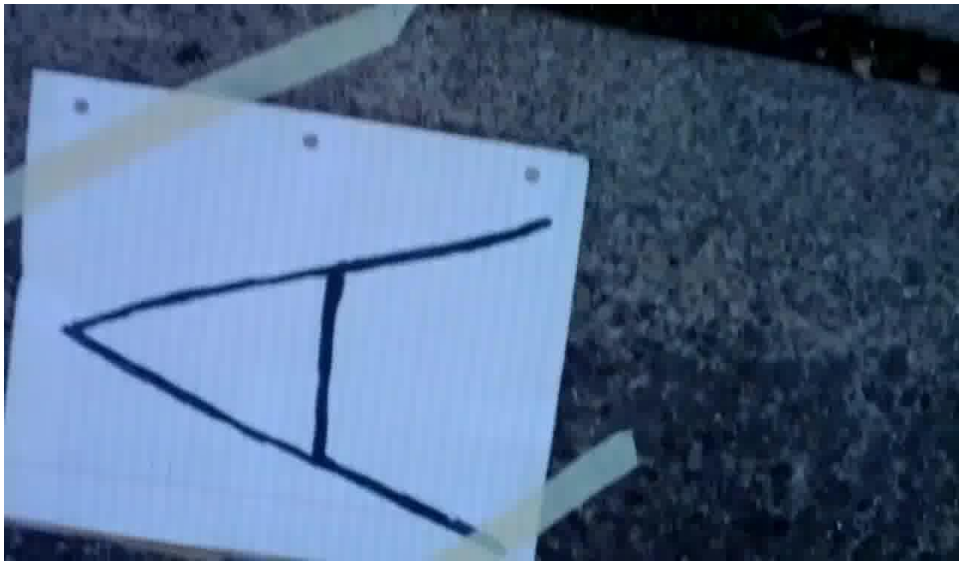

Figure 2. Image of Letter A Taken from an Android Smartphone’s Camera on the Drone

Figure 3. Image of Letter L Taken from an Android Phone's Camera on the Drone

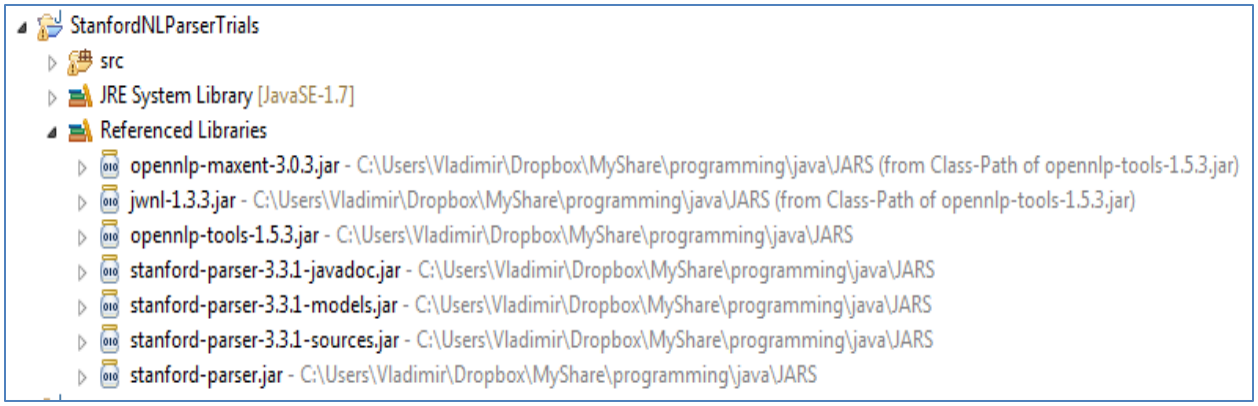

Figure 4. JAR Files in My Eclipse Stanford Parser Project