CS224N/Ling 237 Homework #2
Programming Project: A CKY Parser

Due: Wed, April 14 2004

This assignment may be done individually or in groups of two. See the statement on homework collaboration policy on the website `http://cs224n.stanford.edu`

Read this assignment soon, and especailly if you are less familiar with programming, start working on it, discover stumbling blocks, and ask questions early. If anything is unclear or you need some kind of assistance, feel free to send a question to **cs224n-spr0304-staff@lists.stanford.edu**. However, before you do that, you're encouraged to look among the FAQs on the class website: `http://cs224n.stanford.edu`

The aim of this project is to gain experience writing a simple parser. You may write the program in any language you wish, but we will be providing a convenient code framework in Java which means you will only need to implement the core parsing algorithm if you work in Java. However, you are free to change the code any way you like.

If you have other hashtable, etc., modules from other classes that you are familiar with, feel free to use those instead. However, we will not be able to provide support in that case. While you are welcome to reuse code for standard data structures in this way, you naturally shouldn't be scouring the web to find a complete parser to reuse! That's not the aim of the project. In any case, *if you are reusing code of others from places other than Stanford system libraries, you should be sure to acknowledge it in your write-up.*

1 What to do

1.1 Part 1

Write a simple CKY parser, which reads a grammar and a lexicon, and will then parse sentences. The grammar and lexicon are specified in separate files in the following format:

S $\longrightarrow$ NP VP
NP $\longrightarrow$ DT NN
NP $\longrightarrow$ NNP

DT $\longrightarrow$ the
NNP $\longrightarrow$ Chris
NNP $\longrightarrow$ Adil

The **S** symbol is always assumed to be the distinguished start symbol of the grammar. You should assume all other symbol names to be arbitrary.

Similarly, the input sentences will be provided one per line in a file:

```
Chris saw Adil
Adil saw Chris
```

Your program should accept 3 arguments corresponding to the respective filenames to use. For example, with our code, the parser should be run as:

```
java Parser grammarfile lexiconfile sentencefile
```

Your parser take in the three file names as arguments, and write out the parses for each sentence in the sentences file in some reasonable (e.g, bracketed [p.98 of the book] or indented list) format. At the end of each sentence, it should print out a line like:

```
This sentence had 5 parses
```

Important: It *must* print out exactly that. We're going to check the correctness of your parser by searching the output files for regular expressions of the form "`This sentence had [0-9]+ parses`". Make sure your output doesn't cause us a headache. Note that your parser should work for any grammar and lexicon: we should be able to try it out with a little grammar of, say, Inuit, and expect it to work.

If you use the code framework provided in our Java files, you will only need to implement the core parsing function in `CKYParser.java`. The grammar, lexicon and the sentences have been read in, and are passed to your parsing function. Have a look at the comments in that file, and feel free to ask if some part of the code is unclear.

If you are using Java but not our code, please put your main routine in a class labeled Parser, as in our framework. For other languages, please provide an executable or shell-script named Parser that will run with arguments as shown above.

1.2 Part 2: Grammar development

Write a grammar and lexicon (only as much as you need!) for your parser. Ensure that your grammar and lexicon can parse the correct sentences, and returns an interesting selection of the ambiguities present in them:

cats with claws scratch people
cats that bite people scratch people in the face
Chris saw the man in the corner of the room through a telescope
the post office will hold out discounts as incentives to the franchisees

Try to make sure that your grammar also doesn't parse ungrammatical sentences:

*Chris saw

Keep a copy of your grammar, lexicon and sentence files in your code directory under the names `mygrammar`, `mylexicon`, and `mysentences`. Submit these along with the rest of the code (as described at the end). Try to choose your sentences to demonstrate different aspects of the grammar, and also include ungrammatical sentences that are rejected by your grammar. You can also write a brief note on your observations with these in your write-up.

1.3 Extra credit

Extra credit of up to 10% will be given for significant improvements over the basic requirements. For example, extend the CKY parser so that it handles empty categories or unary rules.

1.4 Files

Some materials that you may use or need for this assignment are in the following directory in AFS on the Sweet Hall machines: `/afs/ir/class/cs224n/pp1`. We'll put a sample grammar, lexicon and test sentences there. To test your code, make sure that:

```
java Parser gfile lfile sfile
```

results in the 2 correct parser for the sentence in the sfile.

The Java code is in the `/afs/ir/class/cs224n/pp1` directory. You will need to fill in the findAllParses(...) method in `CKYParser.java`. Feel free to change the code in any way you like. The grammar and lexicon are represented in the same way for simplicity (as `Grammar` objects), and some hints on using them are included in the comments in `CKYParser.java`.

2 Submitting the assignment

2.1 The program: electronic submission

Submission of the program code will be via a script (avaliable nearer the submission date). To submit your program, put all the needed files in one directory on a Sweet Hall machine (or, possibly, another machine with AFS on it and with you being correctly authenticated) and type:

```
/afs/ir/class/cs224n/bin/submit-pp1
```

If you need to resubmit it type

```
/afs/ir/class/cs224n/bin/submit-pp1 -replace
```

You should make sure that you include the source code for your programs, a Makefile that will build them, and data files for your grammar and lexicon. We will run programs on the Sweet Hall systems, so they should run without problems there. If there is something we should know to correctly run your parser, include that in a README file along with your code.

You may wish to complete the assignment using whatever programming language(s) you wish, but we expect clear readable code, and clear textual description of the algorithms employed in your write-up.

2.2 The report: turn in a hard copy

The report need not be long but should accurately describe the architecture of your program, the testing you did, and extensions you implemented, and any relevant discussion of the structure of your grammar, or problems you had implementing the grammar. **Your write-up must be submitted as a hard copy.**

3 Evaluation criteria

What we're looking for in the grading is:

- A parser that works correctly when we run it over the test grammars [This is the most important thing!]
- A linguistically sensible grammar for the range of sentences indicated.
- A clear discussion of the algorithm.
- Clean, intelligible code.
- A discussion of the testing you did.
- A discussion of alternatives or extensions you implemented