CFG Experimenter

This is an *individual* assignment, not a team assignment.

Your task is to implement the key algorithms used in LL(1) and LR(1) parser generators. You'll do this within a skeleton of the *CFGExperimenter* project that Brian Kelley and I developed.

The project is intended to be completed using Eclipse with the CUP/JFlex plug-in. You may use other development environments, though I wouldn't suggest doing so. Instructions on installing the CUP/JFlex plug-in are available here:

http://www.rose-hulman.edu/class/csse/csse404/Handouts/CUPLEX.html

I've created individual repositories for each of you at the following URL:

http://svn.csse.rose-hulman.edu/repos/csse404-201030-*username*

where *username* is your Rose-Hulman user name. You should add that repository location to Eclipse (using the *SVN Repository Exploring* view). You should then be able to check out the *CFGExperimenter* project into your Eclipse workspace.

Once you've checked out the project, use *Window* → *Views* to add the *Tasks* view (as opposed to the Mylyn Task *List* view). Within the *Tasks* view you should see the ToDo items listed below. You can double click on an item in the *Tasks* view to jump to the code.

Phase 1

1. Construct Nullable Nonterminals
2. Construct First Sets
3. Construct the Follow Sets
4. Build the LL(1) Parser Table
5. Decide if the grammar is LL(1) Parseable
6. Implement the LL(1) Parsing Loop — parsing animation should work at this stage

Phase 2

7. Implement Closure Algorithm
8. Implement Goto Algorithm
9. Construct the Canonical Collection
10. Populate Action and Goto Tables

11. Implement the LR(1) Parsing Algorithm — parsing animation should work at this stage

For the most part, **the tasks must be completed in the order listed** as subsequent tasks are dependent on prior ones. We've included all the fields and classes that we used in our solution, but stripped out the key algorithms. You'll need to study the program to determine what data types we expect, but **you won't need to create any new fields or classes**. Variable, method, and field names in our code are chosen to correspond closely to the algorithms given chapter 3 of *Engineering a Compiler*, by Cooper and Torczon. **Open your textbook before you start coding!**

Testing

Unit tests are included for all of the algorithms. Right-click on the project in the *Project Explorer* view and choose *Run As → JUnit Test* to run all the unit tests for the project.

Several sample grammars for your use are included in *tests → edu.roseHulman.cfg*. Among these is *cfgGrammarExample* which gives the grammar of grammars and includes comments about them. The file *solution.jar* in the project is a working version of the program so you can compare your output to ours. Just double-click the jar file to run it. You shouldn't attempt to reverse engineer the jar file. Doing so would violate the spirit of the assignment and is likely to be more work that just writing the algorithms yourself.

Questions

This project includes *a lot* of provided code. Please ask questions about it.

Submission

Submit your solution by committing it to your Subversion repository by midnight on each milestone deadline.