GLASS

Generalized Language Abstraction and Specification System

Team: Tommy Galletta, Alexander Lockard

Faculty Advisor: Dr. Stansifer

Goals and Motivation

Goals:

- To examine and understand the inner workings of parser generators, both from theoretical and practical standpoints.
- To build our own parser generator tool that attempts to resolve some issues with pre-existing tools in an elegant manner.
- To implement a GUI system and a macro system around the parser generator that gives the tool additional flexibility and ease of use.

Motivation:

- We hope to learn about the inner workings of parser generators, and the theory that goes with it.
- We hope to develop a tool that addresses the complexities of other parser generators by having a simpler way to specify syntax and semantics.
- We hope to build a small extension for the parser generator (the macro system) that can manipulate parsed code in a variety of ways.

Approach

Grammar specification and parser generation

- Users can specify a desired grammar using GLASS's grammar specification language, based on the grammar specification languages of other parser generators.
- GLASS can receive a grammar specification along with a source code file, and parsing it into an XML-like parse tree.
- If the user's source file contain errors conflicting with their grammar specification, GLASS will provide a series of error messages to assist in debugging.

Approach (cont.)

GUI based grammar specification tool

- Users can build a grammar graphically through an interface.
- Users will interact with a visualization similar to a railroad/syntax diagram.
- Users will export the constructed diagram to a corresponding grammar file for use with GLASS.

Syntax markup language generation

- GLASS will output an XML-like markup representation of the input file corresponding to user-defined tokens.
- Users can specify which tokens/grammars to markup (eg. ignore whitespace tokens)
- Users can write the XML to a specified file to use GLASS with other applications.

Approach (cont.)

Querying and macro-based refactoring tool

- Users can execute queries on the XML-like tree to search the input file.
- Users can call predefined macros on the XML-like tree to perform transformations on specified tokens.
- Users can specify macros based on basic logical expressions through the query system.

Extensive documentation

- All features within GLASS are well documented on a publicly available, actively maintained documentation website.
- Tutorial-like sections within the documentation allow users to quickly learn GLASS.

Novel Features

Querying and macro-based refactoring tool

While tools already exist for markup language-based source code recommendation such as srcML and pyRegurgitator, GLASS is the only tool able to do this for user-specified grammars. GLASS can be used as a tool for source code representation to query any language defined by the user.

Technical Challenges

Defining the structure of the syntax specification file

A syntax specification language is something that all parser generators already have, however with the goal of our tool being simplicity, we will have to closely examine the pros and cons of the different specification languages used within these tools to determine which choices we should implement in our language and which we should avoid.

Making the parser generator

After defining the structure of our syntax specification files, we will have to construct the parser generator to allow our tool to parse specified languages. This will require a dive into how parsers work, as well as the theory that goes along with it.

Technical Challenges (cont.)

Designing the macro-based refactoring tool

While interpreting the macro files will probably not be a terribly difficult task, we will have to decide which macro functions we should include in our application, and then determine how to implement the functionalities for each macro function.

Milestone One

- Compare and select the best programming language for writing this tool in. Possible languages include Java, Python, C++, and Rust.
- Investigate the grammar specification languages in other parser generator tools.
- Compare and select tools, libraries, or packages for handling XML.
- Compare and select a tool for building the GUI.
- A “Hello world” example for the chosen programming language, including several benchmark tests, for example speed measurements for graph traversal algorithms.
- A “Hello World” example for choosing the GUI framework. An example app implemented from a mockup to demonstrate the full features of the framework.

Milestone One (cont.)

- A “Hello World” example for the framework of text ingestion, including benchmarks for ingestion and an example of handling errors within the file.
- Based on the investigation described above, develop a design pattern for the structure of the defined grammar file.
- Research existing algorithms for parsing a file given grammar (e.g. LALR, LR, and LL algorithms). Select algorithms from research and provide examples of why it is “better” than the others.
- Develop a design pattern for how the macro-based refactoring tool will work. Provide a “Hello world” example of the refactoring in action. (not code just a written example)
- Create requirements documents for both the GUI tool and parser generator.
- Create a design document for both the GUI tool and parser generator.
- Create a test document for the parser.

Milestone Two

- Reach checkpoint for the parser. At this point, the overall design should be created in a modular format. All components should be easily extensible.
- Implement, test, and demo outputting to XML-like format

Milestone Three

- Implement, test, and demo final parser generator
- Implement, test, and demo GUI tool for outputting grammar files based on user input.

Task Matrix

Task	Tommy	Xander
Compare and select technical tools	Grammar specification languages, programming languages	XML, GUI
“Hello world” demos	Grammar specification languages, programming languages	XML, GUI
Resolve technical challenges	Defining syntax specification structure, syntax specification to parse tree algorithm	Designing the macro-based refactoring tool
Compare and select collaboration tools	**N/A (tools already decided)**	
Requirements Document	80%	20%
Design Document	20%	80%
Test Plan	80%	20%