A declarative approach to syntax parsing using Prolog

Tom Borgia, Connor Davis, Angela Huang, Nate Milkosky

About Prolog (**Pro**gramming in **Log**ic)

- Developed in 1972 by Alain Colmerauer and Phillipe Roussel (natural language processing) with assistance from Robert Kowalski (automated theorem proving)
- Categorized as a **declarative programming language**
- Contains 3 Main Building Blocks
  - **Facts** - Basic assertions about some world
  - **Rules** - Inferences about the facts in the world
  - **Queries** - Questions about the world
- Comparable to SQL as it allows users to:
  - Express Data
  - Query Data

Project Overview

- The goal of our project was to create a Prolog program that would parse a Java input file and determine if the program is syntactically correct.

- This mimics a syntax parser in a compiler and offers insight into compiler development.

- Prolog is useful for processing languages (natural and programming) because of its structure.

Project Motivation

- Expand on a topic we had prior knowledge in
  - We learned about grammars in class and were familiar with the thoery behind how parsers are used within the context of syntax analysis
- Design a project that utilized Prolog's strengths:
  - Automatic Backtracking (can be utilized for a recursive descent parsing)
  - Logical Variables
  - Trees as the central data type
- Choose a project with a broader application
  - Although less popular than the procedural approach, the declarative paradigm is one of the two basic paradigms used for compiler development.

Methods

1. Understand the concepts behind logic programming in order to understand how the Prolog interpreter works
2. Analyze Java syntax and construct a grammar for it in Backus Naur Form (BNF)
3. Get familiar with Definite Clause Grammar (DCG) syntax in Prolog
4. Implement the parser: Convert the BNF description of Java to Prolog code using the DCG syntax
5. Construct meaningful test queries

Implementation

- Implemented using DCG, from a BNF description of Java syntax.
- Separated into multiple parts: Declarations, Types, Blocks and Commands, Expressions, and Tokens.
- The file reader performs the lexical analysis, and splits up the tokens using whitespace.

Demonstration

Challenges

- There were many issues with infinite recursion and backtracking, because of the way the syntax was specified.

- Combining all of the pieces together was more complicated than expected: we had to modify code to make parts compatible

- Ensuring that no edge cases cause issues with the program isn't easy

- Some problems with the code are hard to debug in Prolog, as it can be confusing how the code executes.

Future Direction

- Produce helpful error messages during the syntax analysis stage.
- Move onto semantic analysis. For example, the Prolog program should ensure a Java function is actually defined somewhere in the Java program.

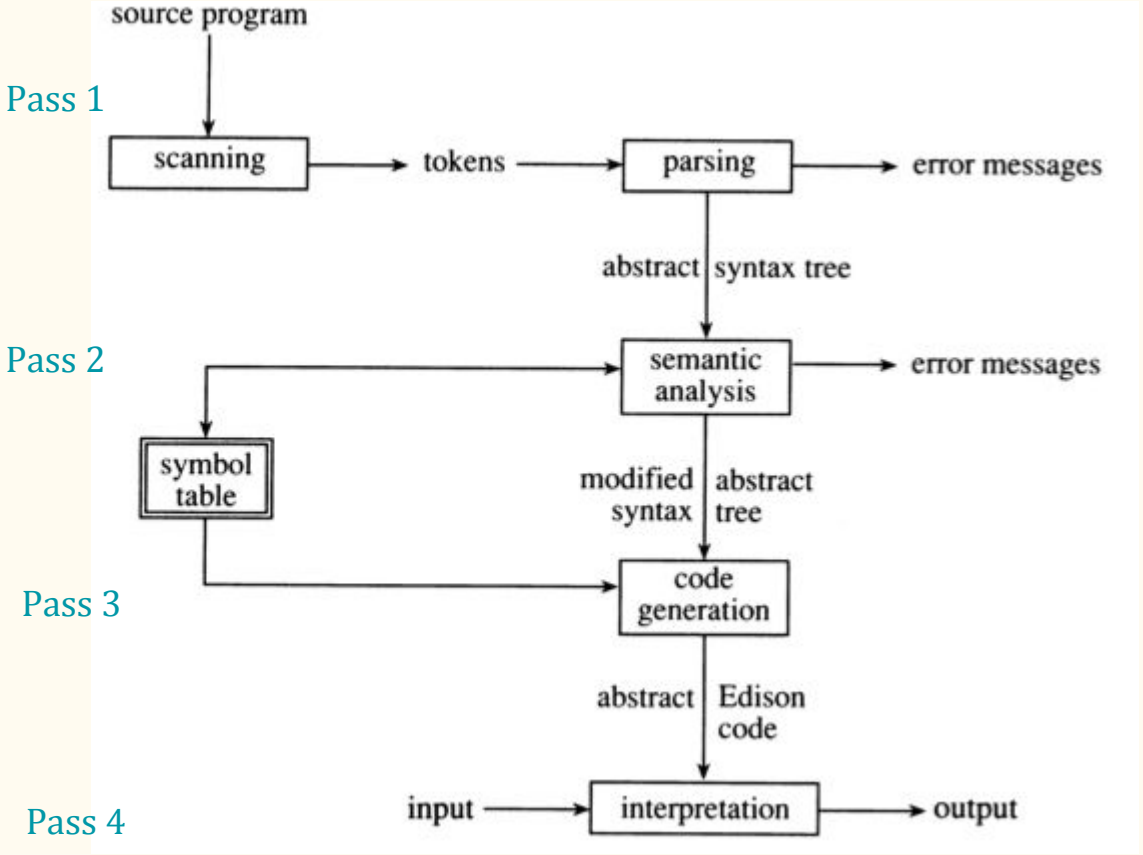

Figure 1: Compiler control flow

Future Direction (cont.)

- Expand our grammar description to include all Java syntax. We are missing some corner-case expressions, as well as hexadecimal and octal numbers.

- Work on the optimization of the parser. The backtracking stack could get huge. Backtracking evaluates all possibilities, and may take take a long time to find whether the syntax is correct.

References

D.H.D. Warren. Logic programming and compiler writing. Software—Practice and Experience, 10(2):97–125, February 1980.

J. Paakki. A practical implementation of DCGs. In Dieter Hammer, Third International Workshop on Compi ler Construction, volume 477 of LNCS, p. 224–225.

J. Paakki. Prolog in Practical Compiler Writing. The Computer Journal 34(1):64–72, 1991.

Figure 1:

http://comjnl.oxfordjournals.org/content/34/1/64.full.pdf

Questions?