Implementation Of Virtual Lookahead

- Shift The ANTLR Grammar Analysis Code To The Parser
  - Existing Grammar Analysis Is Powerful
  - Ter’s Thesis Lays Groundwork For More Powerful Parsers
- Simplifies ANTLR (except for backward compatibility!)
  - ANTLR 2.X Implementation?
- Build a New ANLTR Object Model
  - Parser Objects
  - Rule Objects (Ruling Class!!!)
  - Alternative Objects?

Advantages Of Virtual Lookahead

- Virtual Lookahead Can Make Parsers More Extensible
    - Rules May Be Added or Altered by Descendant Classes
- Performance Impact Can Be Minimized
    - Lookahead Computation Can Be Performed At Initialization
    - Code Will have Minimal Runtime Differences
- Dynamic Parsers Can Be Built
    - An Extensible Parser With Rules Added At Runtime
    - Bundle A Grammar With A Source File
    - C++ May Not Support This Very Well (Java!)
- Parser Development Can Be Interactive
    - An Interpretive Grammar Development Environment
    - ANTLR Grammar Analysis Can Influence Parser Design

What Should a Rule Object Look Like?

- Lookahead Sets/Tokens Stored per Instance
  - Computed At Rule Initialization
  - Can Be Recomputed by a Class Method
- Instance Methods for Each Alternative
  - Generated by ANTLR When Rule is Built
  - Could Make Alternatives an Object as Well, But Is It Worth It?
- Class Methods for Lookahead Computation
  - All Rules Must Manage Lookahead Data In The Same Manner
  - Lookahead May be Handled by Rule or Passed Upward to Parser
- If This Gets Implemented, We Have to Call it a “Ruling Class”

How About a New Object Model?

- A Parser Object Becomes a List Of Rule Objects
  - Rule Objects Manage Their Own Lookahead Information
    - Parser Calls a Lookahead Method
    - Rule Object Performs the Lookahead Task

  - OR -

  - Parser Manages the Lookahead Information
    - Parser Calls a Return_Lookahead Method
    - Rule Returns a Lookahead Set to the Parser
- Both Types Of Lookahead Management Can be Used
  - For k Tokens of Lookahead, Return The Lookahead Set
  - For Syntactic Predicates, Let the Rules Manage Lookahead

How Should Virtual Lookahead Be Computed?

- A Stored List of Arrays of Tokens?
  - Store Lookahead Tokens in Arrays
  - Define a tokcmp() function like strcmp()
- A Stored List of Arrays of Sets?
  - Sets Can Store a Range of Valid Tokens
  - Larger Grammars May Require More Storage
- What About Semantic Predicates and Hoisting?
  - An “Escape” Token (Set) Which Identifies the Tests to Perform
  - An Array of Pointers to Methods Returning Boolean Values
- What About Syntactic Predicates?
  - Check Start Set to Determine if Rule is Valid
  - Then Just Call `rule(action_flag)`

When Should Virtual Lookahead Be

- At Parser Initialization Time?
    - Sufficient For Parser Subclassing
    - Lowest Performance Penalty (Slower Start-up)
- Whenever A Rule Is Added or Deleted?
    - Allows A Parser to Change Dynamically While Parsing
    - A Truly Generic Parser Could be Built
        - Partial or Complete Parser Specification Included With Input
    - Potential Significant Delays When a Rule is Added
    - Lookahead Must Be Recomputed With Each Added Rule
- On The Fly? (Interpretive)
    - Useful For Interactive Grammar Development
    - Too Slow For a Production Compiler?

What Is Virtual Lookahead?

- The Static Computation of Lookahead Limits Extensibility
- Why Not Shift The Computation to Runtime?
    - Lookahead Could be Recomputed When a Grammar is Subclassed
    - Static Rule Methods Could be Made Virtual
        - Thus, Rule Methods Could be Overridden
        - Overridden Rule Methods Could Invoke New Actions
    - Subclassed Grammars Could Also Add New Rule Methods
    - A Grammar Could Be Built With an Interpretive Environment
        - Interactive Grammar Design, Development, Testing
        - Then Code Generation From Completed Grammar

Definition: Virtual Lookahead

The Runtime Computation of Parser Lookahead.

ANTLR and C++ Mode

- With C++ Mode, an ANTLR Parser Can Be Subclassed
    - A Single Grammar Can Be Used in Several Applications
- ANTLR C++ Mode Object Model is a Parser Object
    - Each Rule Yields a Static Method
        - Rule Methods are Static Due to Lookahead
    - Each Action Becomes a Virtual Method
        - Existing Actions Can be Overridden When Subclassing
        - New Actions Cannot be Added (Not Invoked, Anyway)

ANTLR and Lookahead

- ANTLR Generates Recursive Descent Parsers
- Ambiguities in Grammars are Resolved With Lookahead
    - LL(k) Lookahead
        - Demand Lookahead
        - Linear Approximate Lookahead
        - Full Lookahead
    - Infinite Lookahead (Syntactic Predicates)
- Lookahead Sets are Computed at Code Generation Time
    - Lookahead is Hard-Coded into the Parser
    - Any Change in a Grammar Requires a Complete Rebuild

The Case For Virtual Lookahead

PCCTS Workshop ‘95

Steve Robenalt

USCS/CableData

7/23/95