

Trees and ANTLR 2.8

Loring Craymer
Jet Propulsion Laboratory

Background

- PCCTS (ANTLR 1)
 - enabled rapid development of parsers and lexers
 - LL(k)
 - Parser predicates—LL(infinity) and semantic context
 - Introduced transformable ASTs (SORCERER)
- ANTLR 2
 - Lexer grammars – can even lex FORTRAN

The Goal is Rapid Development

- ANTLR parser development is usually fast
- Then development slows for AST and tree grammar development
- And gets further bogged down at the code generation stage.

=> There is room for significant improvement!

Observations on Tree development

- Tree grammars can be automatically generated
 - The troublesome case is that of roots inside loops
 - Other cases are straightforward
- Tree restructuring
 - Minimal set of operations on AST nodes
 - Make root
 - Add child
 - Add sibling
 - Need not depend on target-language

Tree Construction Syntax Design Considerations

- Conformance
 - Needs to “fit” with existing ANTLR syntax
 - i. e.: Should look like part of ANTLR, not an add-on
 - Extends but does not modify current syntax
- Completeness
- Comfort (ease of use)

Tree Construction Syntax

- Tree actions – $\wedge\{ \dots \}$
 - Encloses explicit tree construction
- Tree Constructor -- $\wedge(\dots)$
 - Mirrors $\#(\dots)$ tree walker syntax
- Construction predicates -- $\{ \dots \}$?
 - Construction may be driven by semantics
- Extras
 - Restart tree construction – RETURN
 - Duplicate node -- COPY
 - Node constructor -- $\wedge[\dots]$ in line or in tree action
 - Translocation support

Tree Action Examples

- Interchange elements in a rule
 - $a_0 : A \ b:B! \ C \ ^{\{ b \}} ;$
 - Output is $A - C - B$
- Duplicate a node
 - $a_1 : A \ b:B \ C \ ^{\{ \text{COPY } b \}} \ D ;$
 - Output is $A - B - C - B - D$

Tree construction examples

- Convert phrase to tree
 - $t_0 : A b:B! c:C! d:D! \wedge\{ \wedge(b c d) \} E ;$
 - Output: $A \#(B C D) E$
- Complex tree construction
 - $T_1 : A b:B! c:C! d:D! e:E! f:F! \wedge\{ \wedge(b \wedge(c d) e) f \} G ;$
 - Output: $A \#(B \#(C D) E) F G$

Construction Predicate Example

C0 : a:A! b:B! c:C! d:D!

```
  ^{ { a.equals("foo") }? ^( a b c d )
  |  { b.equals("bar") }? ^( b a c d )
  |  ^( d c b a )
  }
  ;
```

- Construction predicates can be used for
 - C++ -- function call or constructor?
 - antlr.g – collect null alternatives for tree construction
 - Multi-pass language recognition
 - Some features can be recognized in tree passes instead of in the parser

Tree Grammar Construction

- Simple cases are easy

- $a1: A B^C$; $\Rightarrow a1 : \#(B A C)$;

- $a2 : A | B | C!$; $\Rightarrow a2 : (A | B)?$;

- Roots in loops involve recursion

$R1 : (A B^C)+$; \Rightarrow

$r1a : \#(B r1b C)$;

$r1b : \#(B r1b C A) | A$;

Slightly messy example

$ta : A (B C^{\wedge} D | E F^{\wedge} G) + H ;$

\Rightarrow

$ta : \#(C ta0a D H) | \#(F ta0b G H) ;$

$ta0a : \#(C ta0a D B) | \#(C ta0b D E) | A B ;$

$ta0b : \#(F ta0a G B) | \#(F ta0b D E) | A E ;$

Grammar Testing

- Based on Parr's ParseTreeDebugParser
- Adds TreeParser version (need to add Lexer)
- Added package (namespace) antlr.test
 - ParseTreeDebugParser becomes antlr.test.Parser
 - Similar for antlr.test.TreeParser
- Added -s command line option to set superclass package for Lexer/Parser/TreeParser
- Added TestFilter to support testcase by testcase processing (instead of entire file)

Testing ANTLR Additions

- Components of test:
 - test.g – Parser grammar includes AST construction actions.
 - TestParserTree.g
 - Initial version generated from test.g
 - Tree actions added to further test transformations
 - TestParserTreeTree.g – generated from TestParserTree.g
 - Input file (exercises all grammar rules)

Features of testing

- (Almost) language independent
 - Some actions in lexer
 - Used to test Java, C++ output
- Multiple phases
 - Output of each transformation pass verified by next pass
- Adding an `antlr.test.Lexer` would enable regression testing for ANTLR core.

Status

- Tree construction supported for Java, C++
- Automatic grammar generation
- Minor fixes needed for test framework
- Okayed for release through Open Channel
 - Will be under Open Source License
 - Makes 2.8 eligible for NASA “Software of the Year”

Okay, What's next

- Restructuring of generated tree grammars
 - Currently, grammars need some manual edits to add or avoid syntactic predicates
 - Need to automate merging of generated alternatives
- Parse Tree code generation
 - Can generate Parse Trees and grammars now
 - Source-to-source output from parse tree
 - Message input from one language to parse tree for another
 - Output directly from generated parse tree

Some Other Ideas

- Grammar refactoring
 - Help to structure grammar for action placement
 - Simplify generated grammars if necessary
- Editing grammar from tree view
- RAD via ANTLR Workbench

Very rapid turnaround should be possible for generating DSL processors!

ANTLR 3 could effect a new programming paradigm – every problem can be cast in a form that has language processing elements!