

ANTLR 3.0 Feature Discussion

Terence Parr
University of San Francisco
parrt@cs.usfca.edu
parrt@ANTLR.org

Outline

- Grammar reuse
- Attributes
- Trees
- StringTemplate integration
- Context-sensitive lexing
- Error handling
- UNICODE

Grammar reuse

- Need to add actions to stock grammars to make them do something useful; how do you incorporate grammar updates?
- How to get grammar derivatives, extensions?
- Aspects, subclassing for actions not optimal
- Composition?
 - “Re-use rule expr from grammar g”
 - Solutions I’ve seen are messy, messy, messy
- People often use “cp” and then edit
 - let’s formalize via RCS-like push-forward mechanism
 - goes with programmer’s natural inclination
- Problem not proper domain of ANTLR I feel

Attributes

- Dynamic scoping; attribute accessible to any rule invoked from rule defining it
- Single pass evaluation (L-attributed) only
- Attribute stack like local variables; invocation of surrounding rule creates new value
- StringTemplates can refer to / set them, so can trees

Attribute Example

```
declaration
attribute id;
    : type declarator {print(id);}
;
declarator
    : STAR declarator
    | id=ID
;

```

definition

reference

set

Trees

- Loring Craymer's notation?
- Keep simple operators ^ and !
- Move tree construction out of actions
- Auto tree grammar construction
- "reference nodes"; sym links for trees
- Given a parser/tree builder object, could allow "expr:3+4" instead of #(PLUS #["3"] #[4]) in tree transformer grammar

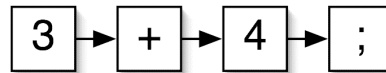
Preserving Token Order When Building Trees

- Tree is 2D version of 1D input; how can you get output in original order?
- Can use min/max tree in antlr.org article
- Better, have array of pointer “planes”
 - Plane 0 is a flat tree preserving original order
 - Plane n is nth tree transformation, usually n=1

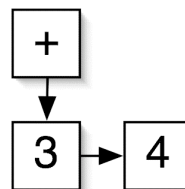
Example Plane 0/1 AST

- Input: “3+4;” -> Tree: (+ 3 4) -> “???”
 - “;” is gone even; yikes!

- Plane 0 pointers:



- Plane 1 pointers:



StringTemplate integration

- C to Java decls: "int a[][]" -> "int[][] a"

```
decl : type dtor ;  
type : "int" | ID ;  
dtor : ID ("[]")* ;
```

```
decl : type dtor => "<t><array> <varid>" ;  
type : t:="int" | t:=ID ;  
dtor : varid:=ID (array:="[ ]")* ;
```

- Template implicitly defines attributes rules may reference/set

Goal-oriented lexing

- Having a separate lexer normally means having a non-goal-oriented lexer
 - "a : ID ;" does not mean go get "ID"
 - it means "get anything and check to see it's an ID"
- Single spec for parser/lexer easier (even if you generate separate lexer); still distinguish lexer vs parser rules with upper/lower case
- Goal-oriented implies parser specifically invokes ID rule in lexer; removes newbie hurdle

Context-sensitive lexing

- Each parser decision point generates special rule in lexer with possible choices: e.g., (ID | INT)
- Difficulties
 - “for”, find “fore” must say “missing for, found ID”
 - whitespace
- The C++ template vs “>>” token problem simply disappears; i.e., when lexing

```
List<List<int>> a;
```

nested template has “>>” in it. Lexer, without context, cannot know which to pick. Only the parser knows that it expects “>” followed by “>” not “>>” token

Another lexer issue

- What goes in literals table vs rule (DFA)?
- DFA too big for grammars with many literals like SQL
- Combined approach:
 - Literals in grammar -> literals table of lexer
 - Rules like LE : “<=“ ; result in actual rule
- Want “<=“ in parser, but do not want it in lexer literals table; perhaps:
 - ‘for’ (keywords) vs “<=“ (implicit lexer rule)?

Error Handling

- Still want exception based semantics, rule paraphrase
- Add back single-token insert like PCCTS
- Add rule resync (stop) sets; like manual FOLLOW sets
- What to do about targets w/o exceptions like C? longjmp? error return codes?
- FIRST/FOLLOW info available for reporting
- Hook tree parsing errors back to input stream

Debug Error handling

- Error messages not always useful:
line 20:12: expecting ID found FLOAT
- Where in grammar was that? How about
line 20:12: expecting ID found LPAREN
at: methodDef : (access)* returnType @ ID (<args>)? "{" ...
input: ... public static void @ (int x) { float ...
- Grammar position particularly useful for tree grammars (less close to input stream)
- Can track position with variable or add argument to match(token, gposition)
- Must make grammar available to parser
 - remember where file.g is? Ok, for debugging I guess
- Derivations and parse trees useful when input accepted, but improperly recognized

UNICODE

- Java (implementation language for ANTLR) is 16-bit UNICODE “clean”
- UNICODE now has supplemental codes above 16 bits (21 bits) for math symbols etc...
- Lookahead analysis algorithm is 32-bit, but all support and runtime code using 16-bit chars
- Shall we convert everything to 32-bit strings?
 - pain to use unnatural word size (recall 8 bit char vs 16 bit unicode)
 - can still use UTF-16 to encode above 16 bits
 - other languages? C, C++, C#, Python?

UTF-16

- “UNICODE to follow, 16 bits”
- Sun: in memory, “...supplementary characters are represented as a pair of char values, the first from the high-surrogates range, (`\uD800-\uDBFF`), the second from the low-surrogates range (`\uDC00-\uDFFF`).”
- How do we match a 21-bit char against a UTF-16 stream?
 - input must be UTF-16 also? ANTLR could pretend everything is 16 bits
- Grammar could say, however, `'\u1039f'` or 'UGARITIC WORD DIVIDER' converts to `"\ud800\udf9f"` (seen as two chars); could not use actual 21-bit char in grammar
- What about column numbers?
 - UTF-16 makes lexer harder (count suppl. codes once)
- UTF-16 screws up: `isIdentifierStart`, `isLowerCase`. ...

Encoding Areas of Concern

- Encoding regions:
 - input grammar spec
(can't easily do 32-bits; UTF-16?)
 - analysis
(handles 32-bits no problem)
 - generated code
(target dependent, Java could use int)
 - input stream
(see UTF-16 encoded or 32 bit chars)
- <Java 1.5, no 32-bit char support, so isUpper won't work anyway unless char code <=16bits

UNICODE Continued

- Character.UnicodeBlock stuff such as vocabulary=BENGALI
 - Is that useful or are the char classes more useful?
 - Do we want DIGIT, punctuation, identifier_start, identifier_extend, ... classes?

XML

- XML as they relate to ASTs
- Serialize ASTs to XML?
- How is XML relevant?
- Oliver Zeigermann's XPA (<http://xpa.sourceforge.net>)?
 - Lets you parse
`<name>anybody but incumbent</name>`
with
rule : "`<name>`" PCDATA "`</name>`" ;
 - Uses ANTLR instead of DTD for verification
 - Better than SAX / DOM-based (hand-built) parsers
 - From token def text file, builds lexer that sets tag token types appropriate for the grammar