

Parsing an XML subset

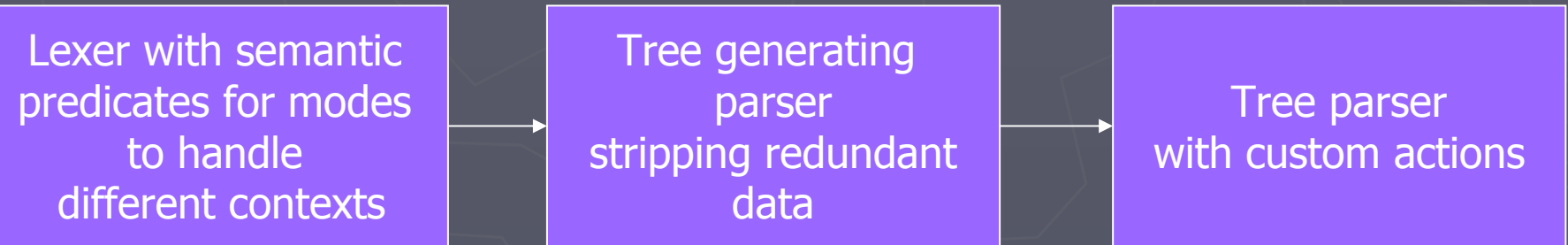
An introduction to ANTLR 3 lexing,
parsing and tree parsing

oliver@zeigermann.de

Just how hard is XML parsing?

- ▶ Restriction to
 - well formed documents (No DTDs)
 - subset without CDATA, Comments, PI, etc.
- ▶ Problem: XML is a sick language
 - `<`, `=`, `>`, etc. have special meanings as tags, but none as text
 - Switching between tag and text context is not indicated by context independent character
 - Especially bad with `>` that can end tag mode, but can be part of attribute value as well

Parsing Overview



Alternative approaches

► Lexer-only

- All processing inside lexer
- Possible because lexer is not restricted to regular expressions
- Done in ANTLR 2.x examples

► Island Grammars / Multiple Lexers

- One grammar / lexer for tags, one for text

Problems:

- No whitespace filtering
- ANTLR (tree) parser analysis capabilities not used

Problems:

- Very simple text grammar is overkill
- Multiple lexers require custom code in ANTLR 3

Parser

```
parser grammar XMLParser;  
options {  
    tokenVocab=XMLLexer;  
}
```

```
element  
: ( startTag (element | PCDATA)* endTag  
  | emptyElement  
  )  
;
```

```
startTag : TAG_START_OPEN GENERIC_ID attribute* TAG_CLOSE ;
```

```
emptyElement : TAG_START_OPEN GENERIC_ID attribute* TAG_EMPTY_CLOSE ;
```

```
attribute : GENERIC_ID EQ VALUE ;
```

```
endTag : TAG_END_OPEN GENERIC_ID TAG_CLOSE;
```

An element can have
sub elements or
is empty

common left prefix
-> LL(*)

Lexer #1

```
lexer grammar XMLLexer;
```

```
{  
  boolean tagMode = false;  
}
```

mode for text/tag
context
broken in EA5,
fixed in EA6

```
TAG_START_OPEN : { !tagMode }? '<' { tagMode = true; } ;  
TAG_END_OPEN  : { !tagMode }? "</" { tagMode = true; } ;  
TAG_CLOSE    : { tagMode }? '>' { tagMode = false; } ;  
TAG_EMPTY_CLOSE : { tagMode }? "/>" { tagMode = false; }
```

```
EQ : { tagMode }? '=' ;
```

```
VALUE : { tagMode }? ( ""! (~""!)* ""! | \"! (~\"!)* \"! ) ;
```

```
GENERIC_ID : { tagMode }? ( LETTER | '_' | ':' ) ( NAMECHAR )* ;
```

syntactic predicate
checking

Lexer #2

```
PCDATA : { !tagMode }? (~'<')+ ;
```

```
fragment NAMECHAR : LETTER | DIGIT | '.' | '-' | '_' | ':' ;
```

```
fragment DIGIT : '0'..'9' ;
```

```
fragment LETTER : 'a'..'z' | 'A'..'Z' ;
```

```
WS : { tagMode }?
```

```
(  
  | '\t'  
  | ('\n'  
    | "\r\n"  
    | '\r'  
  )  
)+
```

```
{ channel=99; }  
;
```

mixture of semantic predicates and DFA predication broken in EA5!

Natural looking WS rule
-> LL(*)

Tree Parser

```
tree grammar XMLTreeParser;  
options {  
  tokenVocab=XMLLexer;  
}
```

```
element  
: ^ ( ELEMENT
```

```
  name=GENERIC_ID
```

```
  { System.out.print("Tag: "+((Tree)name).toString()); }
```

```
  ( ^ ( ATTRIBUTE GENERIC_ID VALUE ) ) *
```

```
  ( element | PCDATA ) *
```

```
);
```

New syntax

That's how we get the
Token text

Augmented Parser #1

```
parser grammar XMLParser;  
options {  
  tokenVocab=XMLLexer;  
  output=AST;  
}
```

```
tokens {  
  ELEMENT;  
  ATTRIBUTE;  
}
```

```
element  
: ( startTag ^^  
  (element | PCDATA)*  
  endTag!  
  | emptyElement  
  )  
;
```

Root operator
works for sub trees
as well

Throw away crap
we do not need

Augmented Parser #2

```
startTag : TAG_START_OPEN GENERIC_ID attribute* TAG_CLOSE  
  -> ^(ELEMENT GENERIC_ID attribute*)  
;
```

```
emptyElement : TAG_START_OPEN GENERIC_ID attribute* TAG_EMPTY_CLOSE  
  -> ^(ELEMENT GENERIC_ID attribute*)  
;
```

```
attribute : GENERIC_ID EQ VALUE -> ^(ATTRIBUTE GENERIC_ID VALUE);
```

```
endTag! : TAG_END_OPEN GENERIC_ID TAG_CLOSE;
```

Tree creation by
example using same
syntax as matching
-> literally equal to
fragment in tree
grammar

ANTLR 3 needs a stream it can navigate on (rewind)

Glue Code

Every rule has its own return type, to include optional custom return values

```
CharStream input = new ANTLRFileStream(args[0]);  
XMLLexer lex = new XMLLexer(input);
```

```
CommonTokenStream tokens = new CommonTokenStream(lex);  
XMLParser parser = new XMLParser(tokens);  
XMLParser.element_return root = parser.element();  
System.out.println("tree=" + ((Tree)root.tree).toStringTree());
```

```
CommonTreeNodeStream nodes =  
    new CommonTreeNodeStream((Tree)root.tree);  
XMLTreeParser walker = new XMLTreeParser(nodes);  
walker.element();
```

A stream for a tree seems funny, but navigation Tokens give structure:

- DOWN: to indicate following tokens are children
- UP: to indicate children list is ended

Summary

- ▶ LL(*) allows for much more natural grammars
- ▶ Unified tree construction matching syntax
- ▶ Tree construction “by example”
- ▶ Complete sub trees can be root
- ▶ ANTLR 3 is not yet ready for “prime time”
- ▶ Full source code available at <http://www.zeigermann.de/xmlSubset.zip>