Language Translation, Domain Specific Languages, and ANTLR

Loring Craymer

JPL

and

Terence Parr

jGuru.com and USF
Intro to ANTLR

- PCCTS (now ANTLR) first published in SIGPLAN Notices (1990)
- Very powerful, easy to use:
  
  “I know how to use lex and yacc. What’s ANTLR” -- newbie to PCCTS news group
  “I know how to use an axe. What’s a chainsaw?” – Gary Funck response
- Development focus has been on building useful tools
- Leading tool for Java translators
- Supports rapid development of “little” languages
- Domain-specific languages (DSLs)
  - enabled by ANTLR and other modern “compiler construction” tools
  - Are useful for avoiding repetitive coding.
  - Provide custom solutions to otherwise difficult problems.
Terence’s Motto

"Why program by hand in five days what you can spend five years of your life automating."
Languages, Languages, Everywhere

• Very few people write compilers
• People build parsers for lots of things
  – xml
  – property, config files
  – data formats
  – scripting and domain specific languages
  – latex, shells
The Nature of Computer Languages

- A language is simply a set of valid sentences; or "what you can say"
- Structure imparts meaning;
  - Terence says the emperor has no clothes \textit{versus}
  - Terence, says the emperor, has no clothes
- Two problems: generation and recognition
- Blue lyrics generation state machine; e.g., My dog is really lazy \textit{and} You was gone

- Cannot generate invalid: My are
The Big Picture

- A translator is typically composed of the following elements:

- When you read, brain automatically converts letters into words then applies grammatical structure to word stream

- Hawaiian state fish _Humuhumunukunukuapua'a_ seen as one word
ANTR features

- Can parse grammars for
  - char streams    lexer
  - token streams   parser
  - 2D trees        tree walker
- EBNF syntax (consistent across 3 grammar types)
- Automatic tree construction
- Error recovery uses exception model
- LL(k) vs LALR(1); advantage in attributes, actions
- Context-sensitive parsing (semantic predicate)
- Arbitrary lookahead (syntactic predicate)
- Generates recursive-descent parsers implemented as class with methods for rules
Sample ANTLR Parser grammar

class SimpleParser extends Parser;

program : (variable)* (method)+ ;

variable : "int" ID (EQUALS expr)? SEMI ;

method : "method" ID LPAREN RPAREN LCURLY (variable)* (statement)+ RCURLY ;

statement : ID EQUALS expr SEMI
    | "return" expr SEMI ;

expr : ID | INT ;
Trees and Transformations

- ANTLR uses Abstract Syntax Trees (ASTs) rather than parse trees—simplifies rewriting
- Child-Sibling format: #(( A B C D ) describes

\[ \begin{align*}
\text{A} \\
\downarrow \\
B \to C \to D
\end{align*} \]

- Construction driven by simple annotation:
  - TOKEN^ says make TOKEN the root of the current rule subtree.
  - TOK! says “ignore” TOK
- Translator phase is series of tree transformations, followed by tree-to-output format conversion
Automating tree grammars

- Annotation solves most construction problems, but not all

⇒ Added tree construction syntax:

\[
\#{ \ #( A B C ) } \text{ generates}
\]

```
A
 ↓
B → C
```

or conceptually:

```
A

B → C
```
Tree Grammar Automation (2)

- Default construction is to copy input structure to output => no problem
- Problem: A ( B C^ D )+ E generates trees like
Critical algorithm

• Note tree is recursively structured.

• Four subtree types:
  - $A B C \cdot D E \Rightarrow #( C A B D E )$
  - Lowest subtree: #( C A B D )
  - Middle subtrees: #( C childTree D B )
  - Top subtree: #( C childTree D E )

Where childTree is either a mid-subtree or the lowest subtree.
Recursive construction

\[
A (B C^D) + E \implies \\

\text{tree0} : \#(C \text{ tree1 D E}) \\
\quad | \#(C A B C E) // single iteration case \\
\quad ; \\
\text{tree1} : \#(C \text{ tree1 D B}) \\
\quad | \#(C A B D) // lowest subtree \\
\quad ;
\]
Domain specific languages

- Use domain-specific notation
  - Custom—ad hoc “notation” for ease of expression
  - Predefined (e.g.: document format)

- May or may not embed domain-specific processing in the translator

- May be used to
  - Avoid repetitive coding
  - Simplify problem expression
  - Simplify problem solution

- Familiar examples (complex!)
  - spreadsheet
  - word processor
  - Matlab/Mathematica
DSLs translated with ANTLR

- IVL
- Cassini Command Language Specification
- TML
- Processing of binary data
IVL

- 2D diagrams are easy to build with editor, 3D are not. Placement, alignment are hard
- IVL: develop VRML models, scenes with simple English-like descriptions
  
  a = Cone
  b = Sphere
  c = Cylinder
  d = Sphere
  draw a
  draw b to the left of a
  draw c to the right of a
  draw d above a
IVL Robot Example From "The Inventor Mentor"

```
// Define a leg
thigh = Cube
thigh.scaleFactor = [.6 1.1 .6]
calf = Cube
calf.scaleFactor = [.5 1.1 .5]
foot = Cube
foot.scaleFactor = [.4 .4 1]
draw foot
draw calf above foot and -1 behind foot
draw thigh above calf
save as ` `leg.iv``
// Construct robot
head = Sphere
head.scaleFactor = [1.5 1.5 1.5]
body = Cylinder
body.scaleFactor = [2.5 3 2.5]
draw head
draw body below head
class Leg = ` `leg.iv``
llleg = Leg
rleg = Leg
draw llleg below body and -2.25 to the left of
   body and -3.1 in front of body
draw rleg 0.5 to the right of llleg
```

*Note definition and reuse of "Leg"*
IVL Robot Rendered
Cassini Command Language

• Problem: verify that implementation matches specification.

• Solution: machine translation
  - Extract parametric descriptions from spec.
  - Generate machine-readable table for verification tools.

• Implementation:
  - Export RTF version of spec.
  - Recognize subset of RTF formatting commands
  - Ignore others
  - Identify formatting+text sequences
6.8.2 SSR Command Listings

Stem: 16CE_BKUP_PWROFF

Name: SSR-A, Bkup, Pwr, Off, Ena

Purpose: Enable Backup Power to SSR-A or SSR-B to be turned off

Routing: Channel 0; CDS-A and CDS-B

Command Input Fields: 16CE_BKUP_PWROFF, Off Cmd Status

Input Parameter Descriptions:

<table>
<thead>
<tr>
<th>Cmd Field</th>
<th>Data Type</th>
<th>Legal Values</th>
<th>Units</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off Cmd Status</td>
<td>Enumerated</td>
<td>ENABLE, DISABLE</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>

Translation:

<table>
<thead>
<tr>
<th>Word #</th>
<th>Bit #</th>
<th>Cmd Field</th>
<th>Data Type</th>
<th>Binary Data (MSB to LSB)</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>15</td>
<td>Off Cmd Status: ENABLE</td>
<td>Enumerated</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Off Cmd Status: DISABLE</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>14-13</td>
<td>Unused</td>
<td></td>
<td>60</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>12-8</td>
<td>CRC Address</td>
<td>Unsigned Int</td>
<td>0 0110</td>
<td>6</td>
</tr>
<tr>
<td>01</td>
<td>7-0</td>
<td>Discrete Bit Select</td>
<td>Unsigned Int</td>
<td>0100 00000</td>
<td>40</td>
</tr>
</tbody>
</table>

Expanded Parameter Description:

Off Cmd Status - When this CRC bit is enabled (set to "1") in either CDS string the PPS will accept an SSPS command to turn off backup power to either SSR A or SSR B.

CRC Address - The CRC address field selects which group of 8 CRC bits or CRC mask bits is to be modified by the command. 16CE_BKUP_PWROFF is in the group of 8 CRC bits with a CRC address field of 06 hexadecimal.

Discrete Bit Select - The Off Cmd Status CRC bit selects the bit from the group 8 CRC bits in which it resides. See also TBD.

Discussion:

TBD
Comments

- Grammar slightly larger than that for ANSI C
- Effort:
  - 3 weeks for initial implementation
  - 3 weeks for doc revisions and modifications to deal with non-compliance of FrameMaker RTF output with Microsoft RTF spec.
TML -- Terence's Markup Language

- Takes raw text with a few special symbols and generates nice looking documents in HTML or Lout (PS, PDF)
- Reduces typing by about 50%
- Extremely expressive, productive
- Extensible "plugin" system; e.g., trees, syntax diagrams
- I build my class notes and slides with TML
TML Example

### TML Demo

- Simple
- Powerful
- Immediately obvious syntax
- Plugin sample: `%(tree("(PLUS 3 (MULT 4 5))")`
Processing binary data

• Binary formats are usually simple
  – Type tags or numerical identifiers
  – Elementary data types—characters, integers, floating point
  – Packed or unpacked records (C structs)
  – Packed or unpacked alternatives (C unions)

• Processing is mostly “aspect-oriented”
  – “For each datum of type A, do XXX”

• Manual code generation is repetitive and error-prone
  – Example: Matching reader/writer code

• Automatically generated code is easily verified and code generator can be rapidly developed for any new function.
Example binary format spec

IDs {
    n2 tag = ID_RECORD;
    n2 parentNode;
    n4 parentId;
    n4 id;
}

strandID {
    n2 tag = STRAND_ID_RECORD;
    n2 parentNode;
    n4 parentId;
    n4 id;
    n4 func;
}

parcelID {
    n2 tag = PARCEL_ID_RECORD;
    n2 node;
    n4 parentId;
    n4 id;
    n4 func;
}

delTime {
    n2 tag = END_TIME_RECORD;
    n4 id;
}

initSync {
    n2 tag = INIT_SYNC_RECORD;
    n4 id;
    n2 slot;
    n4 count;
    n4 reset;
    n2 thread;
}
Comments

- Originally written for SF Express project to translate terrain data formats (not completed) from 32 to 64-bit
- Generated efficient read/write code
  - Packs/unpacks data items.
  - Canonical BigEndian format
- Generated logging code for SPP
- New code generator takes one to three days to implement
- ASN.1 (telecom data format spec) provides more general formatting grammar