The ANTLR Parser Generator

Credits

Terence Parr	U of MN & Parr Research Corporation
Russell Quong	Purdue University
Will Cohen	U of AL Huntsville
Hank Dietz	Purdue University

My motto:
"Why program by hand in five days what you can spend five years of your life automating?"

What's Important?

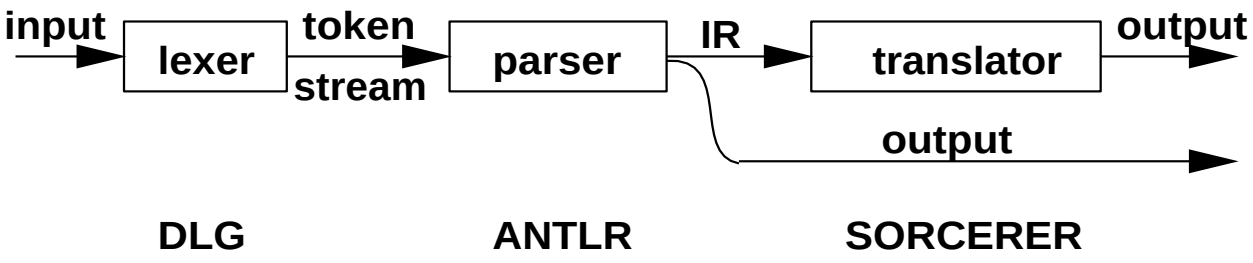
Language tool wish list:

- powerful enough to solve problem
- yield fast code
- good error recovery and reporting
- seemless integration with existing code
- flexible/understandable description language
- yield "debuggable" code
- must actually reduce development time

1. The BIG Picture

Goal: Accept phrases in an input language and generate phrases in an output langage.

Gratuitous figure:



2. Complete ANTLR Example

```
<<
typedef ANTLRCommonToken ANTLRToken;
#include "DLGLexer.h"
main() {
    ParserBlackBox<DLGLexer,
                   BartSimpsonParser,
                   ANTLRToken> bart(stdin);
    bart.parser()->a();
}
>>

#token     "[\ \t\n]+"  <<skip();>>
#token MAN "man"

class BartSimpsonParser {
a   :   "no" "way" MAN
    |   "don't" "have" "a" "cow" "man"
    ;
}
```

3. Notation

What	Example
rule	`a, varName`
token/token class	`ID, REGISTERS`
regular expression	`"do" "[a-z]+"`
optional	{`else stat`}
zero-or-more	`ID ( "," ID )*`
one-or-more	`(stat)+`
action	`<< i++; >>`
semantic predicate	`<<isType(LATEXT(1))>>?`
syntactic predicate	`(declaration)?`

```
rule : alternative_1
     | alternative_2
       ...
     | alternative_n
     ;
```

Rule Arguments (Inheritance)

Communication across rules.

Example: passing scope information to a declaration rule

```
<<enum Scope { GLOBAL, LOCAL };>>

globals :   decl[GLOBAL] ;

block   :   "\{" decl[LOCAL] "\}" ;

decl[Scope s] : ... ;
```

Actions, Local Variables

Location gives time of execution.

Actions cannot introduce ambiguities.

Example:

```
a   :   <<int n=0;>>  /* init-action */
        <<action_1>>  A B
    |   <<action_2>>  C <<action_3>> D
    |   E
    ;
```

Attributes

How parser communicates with scanner.

Example using element labels:

```
a   :   "woof" t:"[a-z]+" u:"[0-9]+"
        <<printf("%s %s\n", $t->getText(), $u->getText());>>
    ;
```

We support the old $\$i$ (integer i) notation,
but prefer new form.

Abstract Syntax Tree Construction

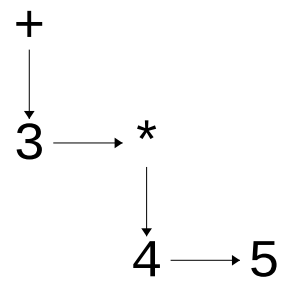
Idea: Annotate grammar to indicate what is a root, what is a leaf, and what is to be excluded from the tree.

Example:

```
e    :   mop ("\+"^ mop)* ;

mop  :   atom ("\*"^ atom)* ;

atom:    "[0-9]+" ;
```

input	resulting tree
3+4*5	

Parser Exception Handling

- Alternative to the automatic mechanism.
- Works with C or C++ interface.
- Good error handling requires programmer's experience.

Given input "`if 3+* then ...`", we would like

`bad if-conditional at "*"`

rather than

`bad expression` or

`parser error before 'then', bailing out` or

`core dumped`

ANTLR can properly report this error and recover gracefully.

```
stat: "if" e:expr "then" stat { "else" stat }
    | "while" expr "do" stat
      ⋮
    ;
    exception[e]
        catch MismatchedToken :
        catch NoViableAlt :
           <<
           fprintf(stderr,
             "bad if-conditional at \"%s\"\n",LATEXT(1));
           consumeUntilToken(THEN);
           >>
```

4. Parsing Strength

Using more than one lookahead symbol.

To distinguish between input token streams "`ID :`" and "`ID =`", rule `stat` requires two lookahead symbols—$LL(2)$.

```
stat:   ID ":" stat          /* statement label */
    |   expr ";"             /* assignment stat */
    ;
expr:   ID "=" expr
    |   INT
    ;
```

Semantic Predicates

Fortran array reference versus function call:

`A(I,3)` versus `MAX(A,B)`

Semantic predicate solution: Call symbol table to choose between alternative productions.

```
expr : <<isvar(LATEXT(1))>>?  ID "\(" exprlist "\)"
       <<array_ref_action>>
     | <<isfunc(LATEXT(1))>>? ID "\(" exprlist "\)"
       <<fn_call_action>>

     ;
```

No change to lexer. Simple, direct solution.

Syntactic Predicates

Ellis and Stroustrup on C++:

"There is an ambiguity in the grammar involving *expression-statements* and *declarations*... The general cases cannot be resolved without backtracking... In particular, the lookahead needed to disambiguate this case is not limited."

```
T(*a)->m=7; //  expression-statement; type cast to T
T(*a)(int); //  pointer to function declaration
```

Ellis and Stroustrup's Solution:

"In a parser with backtracking the disambiguating rule can be stated very simply:

1. If it looks like a *declaration*, it is; otherwise
2. if it looks like an *expression*, it is; otherwise
3. it is a syntax error."

ANTLR solution using syntactic predicates:

```
stat:   (declaration)? declaration
    |   expression
    ;
```

5. ANTLR C++ Parsers

Class hierarchy reflects conceptual separation between recognition subtasks:

Rules become member functions of parser class.

One parser can be comprised of multiple parser objects.

C++ allows same parser to be used for multiple purposes.

```
class OOLangParser {
<<
public:
    virtual void defClass(char *class_name);
    ⋮
>>
class_def : "class" id:ID <<defClass(id->getText());>>
            "\{" members "\}" ";"
          ;
          ⋮
}
```

Can derive classes from `OOLangParser` for building a compiler, browser, etc...

7. Conclusions

1. PCCTS is powerful, flexible, understandable, generates fast/debuggable code, and can easily be integrated into virtually any application.

2. ANTLR is perhaps second most-used parser generator (first = yacc/bison).

3. Well-supported and constantly upgraded.

4. Public domain—in use for several years at many academic and industrial sites.

5. `ftp` site: `ftp.parr-research.com` in `pub/pccts`.

6. Active newsgroup: `comp.compilers.tools.pccts`.