



Parsing with Parse::Eyapp

Casiano Rodriguez-Leon
casiano@ull.es

Text analysis is Perl's strength. Along with regular expressions, modules such as `Parse::RecDescent` and `Parse::Yapp` make text analysis and transformation very easy. The core Perl 5 language doesn't have generic tools for the subsequent stages of text processing, though.

`Parse::Eyapp` (Extended yapp) is a collection of modules that extends Francois Desarmenien's `Parse::Yapp`; any yapp program runs without changes with eyapp. `Parse::Eyapp` provides new features including named attributes, Extended Backus-Naur Format (EBNF)-like expressions, modifiable default actions, abstract syntax tree building, and translation schemes. I'll introduce the basics of translator construction with `Parse::Eyapp` through an example that compiles infix expressions into Parrot Intermediate Representation (PIR). The input to my program will be a semicolon-separated list of infix expressions, such as these:

```
b = 5;
a = b+2;
a = 2*(a+b)*(2-4/2); # is zero
print a;
d = (a = a+1)*4-b;
c = a*b+d;
print c;
print d
```

The output will be the equivalent PIR program. Those expressions translate into:

```
.sub 'main' :main
  .local num a, b, c, d
  b = 5
  a = b + 2
  a = 0 # expression at line 3
  print "a = " # above was
  print a # reduced to zero
  print "\n" # at compile time
  a = a + 1
  $N5 = a * 4
  d = $N5 - b
```

```
$N7 = a * b
c = $N7 + d
print "c = "
print c
print "\n"
print "d = "
print d
print "\n"
.end
```

The Phases of a translator

There are several stages of translation:

- lexical and syntax analysis
- tree transformations and decorations
- address assignments
- code generation
- peephole optimization

The simplicity of my example language (no types, no control structures) permits me to skip context handling, also called semantic analysis, which includes tasks such as type checking, live analysis, and so on.

My goal is to create the module `Infix.pm` so I can go through these steps using `Parse::Eyapp`:

```
# examples/infix2pir.pl
my $parser = Infix->new();

# Set input
$parser->YYData->{INPUT}
  = slurp_file($filename, 'inf');

# Lexical and syntax analysis
my $t = $parser->YYParse(
  yylex => \&Infix::Lex,
  yyerror => \&Infix::Err);
```