

1 NAME

Parse::Eyapp::YATW - Tree transformation objects

2 SYNOPSIS

```
#!/usr/bin/perl -w
use strict;
use Rule6;
use Parse::Eyapp::YATW;

my %BinaryOperation = (PLUS=>'+', MINUS => '-', TIMES=>'*', DIV => '/');

sub set_terminfo {
    no warnings;
    *TERMINAL::info = sub { $_[0]{attr} };
}

sub is_foldable {
    my ($op, $left, $right);
    return 0 unless defined($op = $BinaryOperation{ref($_[0])});
    return 0 unless ($left = $_[0]->child(0), $left->isa('NUM'));
    return 0 unless ($right = $_[0]->child(1), $right->isa('NUM'));

    my $leftnum = $left->child(0)->{attr};
    my $rightnum = $right->child(0)->{attr};
    $left->child(0)->{attr} = eval "$leftnum $op $rightnum";
    $_[0] = $left;
}

my $parser = new Rule6();
my $input = "2*3";
my $t = $parser->Run(\$input);
&set_terminfo;
print "\n***** Before *****\n";
print $t->str;
my $p = Parse::Eyapp::YATW->new(PATTERN => \&is_foldable);
$p->s($t);
print "\n***** After *****\n";
print $t->str."\n";
```

3 INTRODUCTION

Parse::Eyapp::YATW objects implement tree transformations. They have two attributes PATTERN and NAME. PATTERN is a reference to the code implementing the transformation. NAME is the name of the transformation.

Though usually you build a transformation by means of Treeregexp programs you can directly invoke the method new to build a tree transformation. A transformation object can be built from a function that conforms to the YATW tree transformation call protocol

For a subroutine pattern_sub to work as a YATW tree transformation - as subroutine is_foldable in the SYNOPSIS section - has to conform to the following call description:

```
pattern_sub(
    $_[0], # Node being visited
    $_[1], # Father of this node
    $index, # Index of this node in @Father->children
    $self, # The YATW pattern object
);
```

The pattern_sub must return TRUE if matched and FALSE otherwise.

The function is_foldable in the SYNOPSIS section (file examples/YATW/foldrule6.pl) holds the properties to be a YATW tree transformation

```

1  sub is_foldable {
2    my ($op, $left, $right);
3
4    return 0 unless defined($op = $BinaryOperation{ref($_[0])});
5    return 0 unless ($left = $_[0]->child(0), $left->isa('NUM'));
6    return 0 unless ($right = $_[0]->child(1), $right->isa('NUM'));
7
8    my $leftnum = $left->child(0)->{attr};
9    my $rightnum = $right->child(0)->{attr};
10   $left->child(0)->{attr} = eval "$leftnum $op $rightnum";
11   $_[0] = $left;
12 }

```

First, checks that the current node is one of PLUS, MINUS, TIMES or DIV (line 4). Then checks that both children are NUMBERS (lines 5 and 6). In such case proceeds to modify its left child with the result of operating both children (line 10). The matching tree is finally substituted by its left child (line 11).

This is the output of the program in the *SYNOPSIS* section:

```

pl@nereida:~/LEyapp/examples$ eyapp Rule6.ypp; foldrule6.pl
***** Before *****
TIMES(NUM(TERMINAL[2]),NUM(TERMINAL[3]))
***** After *****
NUM(TERMINAL[6])

```

Follows the grammar description file in Rule6.ypp:

```

pl@nereida:~/LEyapp/examples$ cat -n Rule6.ypp
1  %{
2  use Data::Dumper;
3  %}
4  %right  '=',
5  %left   '- ' '+',
6  %left   '* ' '/',
7  %left   NEG
8  %tree
9
10 %%
11 line: exp { $_[1] }
12 ;
13
14 exp:      %name NUM
15          NUM
16          | %name VAR
17          VAR
18          | %name ASSIGN
19          VAR '=' exp
20          | %name PLUS
21          exp '+' exp
22          | %name MINUS
23          exp '-' exp
24          | %name TIMES
25          exp '*' exp
26          | %name DIV
27          exp '/' exp
28          | %name UMINUS
29          '-' exp %prec NEG
30          | '(' exp ')' { $_[2] } /* Let us simplify a bit the tree */
31 ;
32
33 %%
34
35 use Tail2;

```

The module `Tail2` in file `examples/Tail2.pm` implements the lexical analyzer plus the `error` and `run` methods.

4 `Parse::Eyapp::YATW` Methods

`Parse::Eyapp::YATW` objects represent tree transformations. They carry the information of what nodes match and how to modify them.

`Parse::Eyapp::YATW->new`

Builds a `treeregexp` transformation object. Though usually you build a transformation by means of `Treeregexp` programs you can directly invoke the method to build a tree transformation. A transformation object can be built from a function that conforms to the `YATW` tree transformation call protocol (see the section `The YATW Tree Transformation Call Protocol`). Follows an example (file `examples/12ts_simplify_with_s.pl`):

```
neraida:~/src/perl/YappWithDefaultAction/examples> \
sed -ne '68,$p' 12ts_simplify_with_s.pl | cat -n
1  sub is_code {
2    my $self = shift; # tree
3
4    # After the shift $_[0] is the father, $_[1] the index
5    if ((ref($self) eq 'CODE')) {
6      splice(@{$_[0]->{children}}, $_[1], 1);
7      return 1;
8    }
9    return 0;
10 }
11
12 Parse::Eyapp->new_grammar(
13   input=>$translationscheme,
14   classname=>'Calc',
15   firstline =>7,
16 );
17 my $parser = Calc->new();           # Create the parser
18
19 $parser->YYData->{INPUT} = "2*-3\n"; print "2*-3\n"; # Set the input
20 my $t = $parser->Run;              # Parse it
21 print $t->str."\n";
22 my $p = Parse::Eyapp::YATW->new(PATTERN => \&is_code);
23 $p->s($t);
24 { no warnings; # make attr info available only for this display
25   local *TERMINAL::info = sub { $_[0]{attr} };
26   print $t->str."\n";
27 }
```

After the `Parse::Eyapp::YATW` object `$p` is built at line 22 the call to method `$p->s($t)` applies the transformation `is_code` using a bottom-up traversing of the tree `$t`. The achieved effect is the elimination of `CODE` references in the translation scheme tree. When executed the former code produces:

```
neraida:~/src/perl/YappWithDefaultAction/examples> 12ts_simplify_with_s.pl
2*-3
EXP(TIMES(NUM(TERMINAL, CODE), TERMINAL, UMINUS(TERMINAL, NUM(TERMINAL, CODE), CODE), CODE), CODE), CODE)
EXP(TIMES(NUM(TERMINAL[2]), TERMINAL[*], UMINUS(TERMINAL[-], NUM(TERMINAL[3])))
```

The file `foldrule6.pl` in the `examples/` distribution directory gives you another example:

```
neraida:~/src/perl/YappWithDefaultAction/examples> cat -n foldrule6.pl
1  #!/usr/bin/perl -w
2  use strict;
3  use Rule6;
4  use Parse::Eyapp::YATW;
```

```

5
6 my %BinaryOperation = (PLUS=>'+', MINUS => '-', TIMES=>'*', DIV => '/');
7
8 sub set_terminfo {
9     no warnings;
10    *TERMINAL::info = sub { $_[0]{attr} };
11 }
12 sub is_foldable {
13     my ($op, $left, $right);
14     return 0 unless defined($op = $BinaryOperation{ref($_[0])});
15     return 0 unless ($left = $_[0]->child(0), $left->isa('NUM'));
16     return 0 unless ($right = $_[0]->child(1), $right->isa('NUM'));
17
18     my $leftnum = $left->child(0)->{attr};
19     my $rightnum = $right->child(0)->{attr};
20     $left->child(0)->{attr} = eval "$leftnum $op $rightnum";
21     $_[0] = $left;
22 }
23
24 my $parser = new Rule6();
25 $parser->YYData->{INPUT} = "2*3";
26 my $t = $parser->Run;
27 &set_terminfo;
28 print "\n***** Before *****\n";
29 print $t->str;
30 my $p = Parse::Eyapp::YATW->new(PATTERN => \&is_foldable);
31 $p->s($t);
32 print "\n***** After *****\n";
33 print $t->str."\n";

```

when executed produces:

```

nereida:~/src/perl/YappWithDefaultAction/examples> foldrule6.pl

```

```

***** Before *****
TIMES(NUM(TERMINAL[2]),NUM(TERMINAL[3]))
***** After *****
NUM(TERMINAL[6])

```

The YATW Tree Transformation Call Protocol

For a subroutine `pattern_sub` to work as a YATW tree transformation - as subroutines `is_foldable` and `is_code` above - has to conform to the following call description:

```

pattern_sub(
    $_[0], # Node being visited
    $_[1], # Father of this node
    $index, # Index of this node in @Father->children
    $self, # The YATW pattern object
);

```

The `pattern_sub` must return TRUE if matched and FALSE otherwise.

The protocol may change in the near future. Avoid using other information than the fact that the first argument is the node being visited.

Parse::Eyapp::YATW->buildpatterns

Works as `Parse::Eyapp->new` but receives an array of subs conforming to the YATW Tree Transformation Call Protocol.

```

our @all = Parse::Eyapp::YATW->buildpatt(\&delete_code, \&delete_tokens);

```

\$yatw->delete

The root of the tree that is currently matched by the YATW transformation `$yatw` will be deleted from the tree as soon as is safe. That usually means when the processing of their siblings is finished. The following example (taken from file `examples/13ts_simplify_with_delete.pl` in the *Parse::Eyapp* distribution) illustrates how to eliminate CODE and syntactic terminals from the syntax tree:

```
pl@nereida:~/src/perl/YappWithDefaultAction/examples$ \
  sed -ne '62,$p' 13ts_simplify_with_delete.pl | cat -n
 1 sub not_useful {
 2   my $self = shift; # node
 3   my $pat = $_[2]; # get the YATW object
 4
 5   (ref($self) eq 'CODE') or ((ref($self) eq 'TERMINAL') and ($self->{token} eq $self->{attr}))
 6     or do { return 0 };
 7   $pat->delete();
 8   return 1;
 9 }
10
11 Parse::Eyapp->new_grammar(
12   input=>$translationscheme,
13   classname=>'Calc',
14   firstline =>7,
15 );
16 my $parser = Calc->new();          # Create the parser
17
18 $parser->YYData->{INPUT} = "2*3\n"; print $parser->YYData->{INPUT};
19 my $t = $parser->Run;              # Parse it
20 print $t->str."\n";               # Show the tree
21 my $p = Parse::Eyapp::YATW->new(PATTERN => \&not_useful);
22 $p->s($t);                        # Delete nodes
23 print $t->str."\n";               # Show the tree
```

when executed we get the following output:

```
pl@nereida:~/src/perl/YappWithDefaultAction/examples$ 13ts_simplify_with_delete.pl
2*3
EXP(TIMES(NUM(TERMINAL[2],CODE),TERMINAL[*],NUM(TERMINAL[3],CODE),CODE),CODE))
EXP(TIMES(NUM(TERMINAL[2]),NUM(TERMINAL[3])))
```

\$yatw->unshift

The call `$yatw->unshift($b)` safely unshifts (inserts at the beginning) the node `$b` in the list of its siblings of the node that matched (i.e in the list of siblings of `$_[0]`). The following example shows a YATW transformation `insert_child` that illustrates the use of `unshift` (file `examples/26delete_with_trreereg.pl`):

```
pl@nereida:~/src/perl/YappWithDefaultAction/examples$ \
  sed -ne '70,$p' 26delete_with_trreereg.pl | cat -n
 1 my $transform = Parse::Eyapp::Treeregexp->new( STRING => q{
 2
 3   delete_code : CODE => { $delete_code->delete() }
 4
 5   {
 6     sub not_semantic {
 7       my $self = shift;
 8       return 1 if ((ref($self) eq 'TERMINAL') and ($self->{token} eq $self->{attr}));
 9       return 0;
10    }
11  }
12
13  delete_tokens : TERMINAL and { not_semantic($TERMINAL) } => {
14    $delete_tokens->delete();
```

```

15     }
16
17     insert_child : TIMES(NUM(TERMINAL), NUM(TERMINAL)) => {
18         my $b = Parse::Eyapp::Node->new( 'UMINUS(TERMINAL)',
19             sub { $_[1]->{attr} = '4.5' }); # The new node will be a sibling of TIMES
20
21         $insert_child->unshift($b);
22     }
23 },
24 )->generate();
25
26 Parse::Eyapp->new_grammar(
27     input=>$translationscheme,
28     classname=>'Calc',
29     firstline =>7,
30 );
31 my $parser = Calc->new();           # Create the parser
32
33 $parser->YYData->{INPUT} = "2*3\n"; print $parser->YYData->{INPUT}; # Set the input
34 my $t = $parser->Run;               # Parse it
35 print $t->str."\n";                # Show the tree
36 # Get the AST
37 our ($delete_tokens, $delete_code);
38 $t->s($delete_tokens, $delete_code);
39 print $t->str."\n";                # Show the tree
40 our $insert_child;
41 $insert_child->s($t);
42 print $t->str."\n";                # Show the tree

```

When is executed the program produces the following output:

```

pl@nereida:~/src/perl/YappWithDefaultAction/examples$ 26delete_with_trreereg.pl
2*3
EXP(TIMES(NUM(TERMINAL[2],CODE), TERMINAL[*],NUM(TERMINAL[3],CODE),CODE))
EXP(TIMES(NUM(TERMINAL[2]),NUM(TERMINAL[3])))
EXP(UMINUS(TERMINAL[4.5]),TIMES(NUM(TERMINAL[2]),NUM(TERMINAL[3])))

```

Don't try to take advantage that the transformation sub receives in \$_[1] a reference to the father (see the section The YATW Tree Transformation Call Protocol) and do something like:

```
unshift $_[1]->{children}, $b
```

it is unsafe.

\$yatw->insert_before

A call to `$yatw->insert_before($node)` safely inserts `$node` in the list of siblings of `$_[0]` just before `$_[0]` (i.e. the node that matched with `$yatw`). The following example (see file `examples/YATW/moveinvariantoutofloopcomplex`) illustrates its use:

```

my $p = Parse::Eyapp::Treeregexp->new( STRING => q{
    moveinvariant: WHILE(VAR($b), BLOCK(@a, ASSIGN($x, $e), @c))
        and { is_invariant($ASSIGN, $WHILE) } => {
            my $assign = $ASSIGN;
            $BLOCK->delete($ASSIGN);
            $moveinvariant->insert_before($assign);
        }
    },
);

```

Here the `ASSIGN($x, $e)` subtree - if is loop invariant - will be moved to the list of siblings of `$WHILE` just before the `$WHILE`. Thus a program like

```
"a =1000; c = 1; while (a) { c = c*a; b = 5; a = a-1 }\n"
```

is transformed in s.t. like:

```
"a =1000; c = 1; b = 5; while (a) { c = c*a; a = a-1 }\n"
```

5 TREE MATCHING AND TREE SUBSTITUTION

See the documentation in *Parse::Eyapp::treematchingtut*

6 SEE ALSO

- The project home is at <http://code.google.com/p/parse-eyapp/>. Use a subversion client to anonymously check out the latest project source code:

```
svn checkout http://parse-eyapp.googlecode.com/svn/trunk/ parse-eyapp-read-only
```

- *Parse::Eyapp*, *Parse::Eyapp::eyapplanguageref*, *Parse::Eyapp::debuggingtut*, *Parse::Eyapp::defaultactionsintro*, *Parse::Eyapp::translationschemestut*, *Parse::Eyapp::Driver*, *Parse::Eyapp::Node*, *Parse::Eyapp::YATW*, *Parse::Eyapp::Treeregexp*, *Parse::Eyapp::Scope*, *Parse::Eyapp::Base*, *Parse::Eyapp::datagenerationtut*
- The pdf file in <http://nereida.deioc.ull.es/~pl/perlexamples/languageintro.pdf>
- The pdf file in <http://nereida.deioc.ull.es/~pl/perlexamples/debuggingtut.pdf>
- The pdf file in <http://nereida.deioc.ull.es/~pl/perlexamples/eyapplanguageref.pdf>
- The pdf file in <http://nereida.deioc.ull.es/~pl/perlexamples/Treeregexp.pdf>
- The pdf file in <http://nereida.deioc.ull.es/~pl/perlexamples/Node.pdf>
- The pdf file in <http://nereida.deioc.ull.es/~pl/perlexamples/YATW.pdf>
- The pdf file in <http://nereida.deioc.ull.es/~pl/perlexamples/Eyapp.pdf>
- The pdf file in <http://nereida.deioc.ull.es/~pl/perlexamples/Base.pdf>
- The pdf file in <http://nereida.deioc.ull.es/~pl/perlexamples/translationschemestut.pdf>
- The pdf file in <http://nereida.deioc.ull.es/~pl/perlexamples/treematchingtut.pdf>
- The tutorial *Parsing Strings and Trees with Parse::Eyapp* (An Introduction to Compiler Construction in seven pages) in <http://nereida.deioc.ull.es/~pl/eyapsimple/>
- perldoc *eyapp*,
- perldoc *treereg*,
- perldoc *vgg*,
- The Syntax Highlight file for vim at http://www.vim.org/scripts/script.php?script_id=2453 and <http://nereida.deioc.ull.es>
- *Analisis Lexico y Sintactico*, (Notes for a course in compiler construction) by Casiano Rodriguez-Leon. Available at <http://nereida.deioc.ull.es/~pl/perlexamples/> Is the more complete and reliable source for Parse::Eyapp. However is in Spanish.
- *Parse::Yapp*,
- Man pages of yacc(1) and bison(1), <http://www.delorie.com/gnu/docs/bison/bison.html>
- *Language::AttributeGrammar*
- *Parse::RecDescent*.
- *HOP::Parser*
- *HOP::Lexer*
- ocaml yacc tutorial at <http://plus.kaist.ac.kr/~shoh/ocaml/ocamllex-ocaml yacc/ocaml yacc-tutorial/ocaml yacc-tutorial.html>

7 REFERENCES

- The classic Dragon's book *Compilers: Principles, Techniques, and Tools* by Alfred V. Aho, Ravi Sethi and Jeffrey D. Ullman (Addison-Wesley 1986)
- *CS2121: The Implementation and Power of Programming Languages* (See <http://www.cs.man.ac.uk/~pjj>, <http://www.cs.man.ac.uk/~pjj/complang/g2lr.html> and <http://www.cs.man.ac.uk/~pjj/cs2121/ho/ho.html>) by Pete Jinks

8 CONTRIBUTORS

- Hal Finkel <http://www.halssoftware.com/>
- G. Williams <http://kasei.us/>
- Thomas L. Shinnick <http://search.cpan.org/~tshinnic/>

9 AUTHOR

Casiano Rodriguez-Leon (casiano@ull.es)

10 ACKNOWLEDGMENTS

This work has been supported by CEE (FEDER) and the Spanish Ministry of *Educacion y Ciencia* through *Plan Nacional I+D+I* number TIN2005-08818-C04-04 (ULL::OPLINK project <http://www.oplink.ull.es/>). Support from Gobierno de Canarias was through GC02210601 (*Grupos Consolidados*). The University of La Laguna has also supported my work in many ways and for many years.

A large percentage of code is verbatim taken from *Parse::Yapp* 1.05. The author of *Parse::Yapp* is Francois Desarmenien.

I wish to thank Francois Desarmenien for his *Parse::Yapp* module, to my students at La Laguna and to the Perl Community. Thanks to the people who have contributed to improve the module (see CONTRIBUTORS in *Parse::Eyapp*). Thanks to Larry Wall for giving us Perl. Special thanks to Juana.

11 LICENCE AND COPYRIGHT

Copyright (c) 2006-2008 Casiano Rodriguez-Leon (casiano@ull.es). All rights reserved.

Parse::Yapp copyright is of Francois Desarmenien, all rights reserved. 1998-2001

These modules are free software; you can redistribute it and/or modify it under the same terms as Perl itself. See *perlartistic*.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Index

\$yatw->delete, 5

\$yatw->insert before, 6

\$yatw->unshift, 5

ACKNOWLEDGMENTS, 8

AUTHOR, 8

CONTRIBUTORS, 8

INTRODUCTION, 1

LICENCE AND COPYRIGHT, 8

NAME, 1

Parse::Eyapp::YATW->buildpatterns, 4

Parse::Eyapp::YATW->new, 3

Parse::Eyapp:YATW Methods, 3

REFERENCES, 8

SEE ALSO, 7

SYNOPSIS, 1

The YATW Tree Transformation Call Protocol, 4

TREE MATCHING AND TREE SUBSTITUTION,

7