

1 NAME

Parse::Eyapp::treematchingtut - Tree Matching and Tree substitution: an introduction

2 TREE MATCHING AND TREE SUBSTITUTION

Most of the examples in this section can be found in the directory `examples/MatchingTrees` that comes with the distribution of *Parse::Eyapp*.

Matching Trees

Both the transformation objects in `Parse::Eyapp::YATW` and the nodes in `Parse::Eyapp::Node` have a method named `m` for matching. For a `Parse::Eyapp::YATW` object, the method -when called in a list context- returns a list of `Parse::Eyapp::Node::Match` nodes.

```
@R = $t->m($yatw1, $yatw2, $yatw3, ...)
```

A `Parse::Eyapp::Node::Match` object describes the nodes of the actual tree that have matched. The nodes in the returned list are organized in a hierarchy. They appear in the list sorted according to a depth-first visit of the actual tree `$t`. In a scalar context `m` returns the first element of the list.

Let us denote by `$t` the actual tree being searched and `$r` one of the `Parse::Eyapp::Node::Match` nodes in the resulting forest `@R`. Then we have the following methods:

- The method `$r->node` return the node `$t` of the actual tree that matched
- The method `$r->father` returns the father of `$r` in the matching forest. The father of `$r` is defined by this property: `$r->father->node` is the nearest ancestor of `$r->node` that matched with the `treeregexp` pattern. That is, there is no ancestor that matched between `$r->node` and `$r->father->node`. Otherwise `$r->father` is `undef`
- The method `$r->coord` returns the coordinates of `$r->node` relative to `$t`. For example, the coordinate `".1.3.2"` denotes the node `$t->child(1)->child(3)->child(2)`, where `$t` is the root of the search.
- The method `$r->depth` returns the depth of `$r->node` in `$t`.
- When `m` was called as a `Parse::Eyapp::Node` method, i. e. with potentially more than one `YATW` `treeregexp`, the method `$r->names` returns the array of names of the transformations that matched with `$r->node`.

Use of `m` as a `Parse::Eyapp::Node` Method

The example in `examples/MatchingTrees/m2.pl` shows the use of `m` as a `Parse::Eyapp::Node` method.

```
examples/MatchingTrees$ cat -n m2.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use Rule6;
 4  use Parse::Eyapp::Treeregexp;
 5
 6  Parse::Eyapp::Treeregexp->new( STRING => q{
 7    fold: /TIMES|PLUS|DIV|MINUS/(NUM, NUM)
 8    zxw: TIMES(NUM($x), .) and { $x->{attr} == 0 }
 9    wxz: TIMES(., NUM($x)) and { $x->{attr} == 0 }
10  }->generate();
11
12  # Syntax analysis
13  my $parser = new Rule6();
14  my $input = "0*0*0";
15  my $t = $parser->Run(\$input);
16  print "Tree:",$t->str,"\n";
17
18  # Search
19  my $m = $t->m(our ($fold, $zxw, $wxz));
20  print "Match Node:\n",$m->str,"\n";
```

When executed with input 0*0*0 the program generates this output:

```
examples/MatchingTrees$ m2.pl
Tree:TIMES(TIMES(NUM(TERMINAL),NUM(TERMINAL)),NUM(TERMINAL))
Match Node:
Match[[TIMES:0:wxz]](Match[[TIMES:1:fold,zxw,wxz]])
```

The representation of Match nodes by `str` deserves a comment. Match nodes have their own `info` method. It returns a string containing the concatenation of the class of `$r->node` (i.e. the actual node that matched), the depth (`$r->depth`) and the names of the transformations that matched (as provided by the method `$r->names`)

Use of `m` as a `Parse::Eyapp::YATW` Method

A second example can be found inside the file `examples/typechecking/Simple-Types-XXX.tar.gz`. It illustrates a use of `m` as a `Parse::Eyapp::YATW` method. It solves a problem of scope analysis in a C compiler: matching each `RETURN` statement with the function that surrounds it. The parsing was already done, the AST was built and left in `$t`. The `treeregexp` used (see `lib/Simple/Trans.trg`) is:

```
retscope: /FUNCTION|RETURN/
```

and the code that solves the problem (see subroutine `compile` in file `lib/Simple/Types.ey`) is:

```
# Associate each "return exp" with its "function"
my @returns = $retscope->m($t);
for (@returns) {
  my $node = $_->node;
  if (ref($node) eq 'RETURN') {
    my $function = $_->father->node;
    $node->{function} = $function;
    $node->{t} = $function->{t};
  }
}
```

The first line gets a list of `Parse::Eyapp::Node::Match` nodes describing the actual nodes that matched `/FUNCTION|RETURN/`. If the node described by `$_` is a `'RETURN'` node, the expression `$_->father->node` must necessarily point to the function node that encloses it.

The SEVERITY option of `Parse::Eyapp::Treeregexp::new`

The `SEVERITY` option of `Parse::Eyapp::Treeregexp::new` controls the way matching succeeds regarding the number of children. To illustrate its use let us consider the following example. The grammar used `Rule6.y` is similar to the example in the section `SYNOPSIS` in `Parse::Eyapp::Node`.

```
examples/MatchingTrees$ cat -n numchildren.pl
 1 #!/usr/bin/perl -w
 2 use strict;
 3 use Rule6;
 4 use Parse::Eyapp::Treeregexp;
 5
 6 sub TERMINAL::info { $_[0]{attr} }
 7
 8 my $severity = shift || 0;
 9 my $input = shift || '0*2';
10
11 my $parser = new Rule6();
12 my $t = $parser->Run(\$input);
13
14 my $transform = Parse::Eyapp::Treeregexp->new(
15   STRING => q{
16     zero_times_whatever: TIMES(NUM($x)) and { $x->{attr} == 0 } => { $_[0] = $NUM }
17   },
18   SEVERITY => $severity,
```

```

19  FIRSTLINE => 14,
20  )->generate;
21
22  $t->s(our @all);
23
24  print $t->str,"\n";

```

The program gets the severity level from the command line (line 9). The specification of the term `TIMES(NUM($x))` inside the transformation `zero_times_whatever` does not clearly state that `TIMES` must have two children. There are several interpretations of the `treregexp` depending on the level fixed for `SEVERITY`:

- 0: `TIMES` must have at least one child. Don't care if it has more.
- 1: `TIMES` must have exactly one child.
- 2: `TIMES` must have exactly one child. When visit a `TIMES` node with a different number of children issue a warning.
- 3: `TIMES` must have exactly one child. When visit a `TIMES` node with a different number of children issue an error.

Observe the change in behavior according to the level of `SEVERITY`:

```

pl@nereida:~/LEyapp/examples/MatchingTrees$ numchildren.pl 0 '0*2'
NUM(TERMINAL[0])
pl@nereida:~/LEyapp/examples/MatchingTrees$ numchildren.pl 1 '0*2'
TIMES(NUM(TERMINAL[0]),NUM(TERMINAL[2]))
pl@nereida:~/LEyapp/examples/MatchingTrees$ numchildren.pl 2 '0*2'
Warning! found node TIMES with 2 children.
Expected 1 children (see line 15 of ./numchildren.pl)"
TIMES(NUM(TERMINAL[0]),NUM(TERMINAL[2]))
pl@nereida:~/LEyapp/examples/MatchingTrees$ numchildren.pl 3 '0*2'
Error! found node TIMES with 2 children.
Expected 1 children (see line 15 of ./numchildren.pl)"
at (eval 3) line 29

```

Tree Substitution: The `s` methods

Both `Parse::Eyapp::Node` and `Parse::Eyapp::YATW` objects (i.e. nodes and tree transformations) are provided with a `s` method.

In the case of a `Parse::Eyapp::YATW` object the method `s` applies the tree transformation using a single bottom-up traversing: the transformation is recursively applied to the children and then to the current node.

For `Parse::Eyapp::Node` nodes the set of transformations is applied to each node until no transformation matches any more. The example in the section `SYNOPSIS` in `Parse::Eyapp::Node` illustrates the use:

```

1  # Let us transform the tree. Define the tree-regular expressions ..
2  my $p = Parse::Eyapp::Treeregexp->new( STRING => q{
3    { # Example of support code
4      my %Op = (PLUS=>'+', MINUS => '-', TIMES=>('*', DIV => '/');
5    }
6    constantfold: /TIMES|PLUS|DIV|MINUS/:bin(NUM($x), NUM($y))
7      => {
8        my $op = $Op{ref($_[0])};
9        $x->{attr} = eval "$x->{attr} $op $y->{attr}";
10       $_[0] = $NUM[0];
11     }
12    minus: UMINUS(NUM($x)) => { $x->{attr} = -$x->{attr}; $_[0] = $NUM }
13    zero_times_whatever: TIMES(NUM($x), .) and { $x->{attr} == 0 } => { $_[0] = $NUM }
14    whatever_times_zero: TIMES(., NUM($x)) and { $x->{attr} == 0 } => { $_[0] = $NUM }
15  },
16  OUTPUTFILE=> 'main.pm'
17 );
18 $p->generate(); # Create the transformations

```

19

```
20 $t->s($uminus); # Transform UMINUS nodes
21 $t->s(@all);     # constant folding and mult. by zero
```

The call at line 20 can be substituted by `$uminus->s($t)` without changes.

3 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
```

- The tutorial *Parsing Strings and Trees with Parse::Eyapp* (An Introduction to Compiler Construction in seven pages) in <http://nereida.deioc.ull.es/~pl/eyasimple/>
- *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>
- 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-ocamyacc/ocamyacc-tutorial/ocamyacc-tutorial.html>

4 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

5 CONTRIBUTORS

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

6 AUTHOR

Casiano Rodriguez-Leon (casiano@ull.es)

7 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.

8 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

ACKNOWLEDGMENTS, 5

AUTHOR, 5

CONTRIBUTORS, 5

LICENCE AND COPYRIGHT, 5

Matching Trees, 1

NAME, 1

REFERENCES, 5

SEE ALSO, 4

The SEVERITY option of Parse::Eyapp::Treeregexp::new,
2

TREE MATCHING AND TREE SUBSTITUTION,
1

Tree Substitution: The s methods, 3

Use of m as a Parse::Eyapp::Node Method, 1

Use of m as a Parse::Eyapp::YATW Method, 2