PegCpp: An efficient implementation of parser generator in C++.

Andrzej Grosser Grzegorz Michalski
Institute of Computer and Information Sciences, Faculty of Mechanical Engineering and Computer Science,
Czestochowa University of Technology, Czestochowa, Poland

Introduction

Parsing expression grammars (pegs) is an alternative notation to BNF. The basic difference between these notations results from the way, in which the alternative is implemented. In the case of the parsing expression grammar alternatives are ordered. Prior alternatives have a higher priority in choosing from later ones. In this way, the description of language using pegs is always unambiguous. By using pegs it is possible describing all the context-free languages and even some context-sensitive languages.

```
pegs    <- spacing pattern+ EOF
pattern <- rule '<-' spacing expression
expression <- alternative ('/' spacing alternative)*
alternative <- (prefix? suffix)*
prefix <- ('&' / '!') spacing
suffix <- primary (('*' / '+' / '?') spacing)?
primary <- '(' spacing expression ')' spacing / literal /
           '.' spacing / charclass / rule !'<-'
literal <- ['] (!['] .)* ['] spacing
charclass <- '[' (!']' (. '-' . / .))* ']' spacing
rule <- [a-z,A-z]([a-z,A-z,0-9])* spacing
spacing <- (' ' / '\t' / EOL)*
EOL <- '\r' / '\n' / '\r\n'
EOF <- !.
```

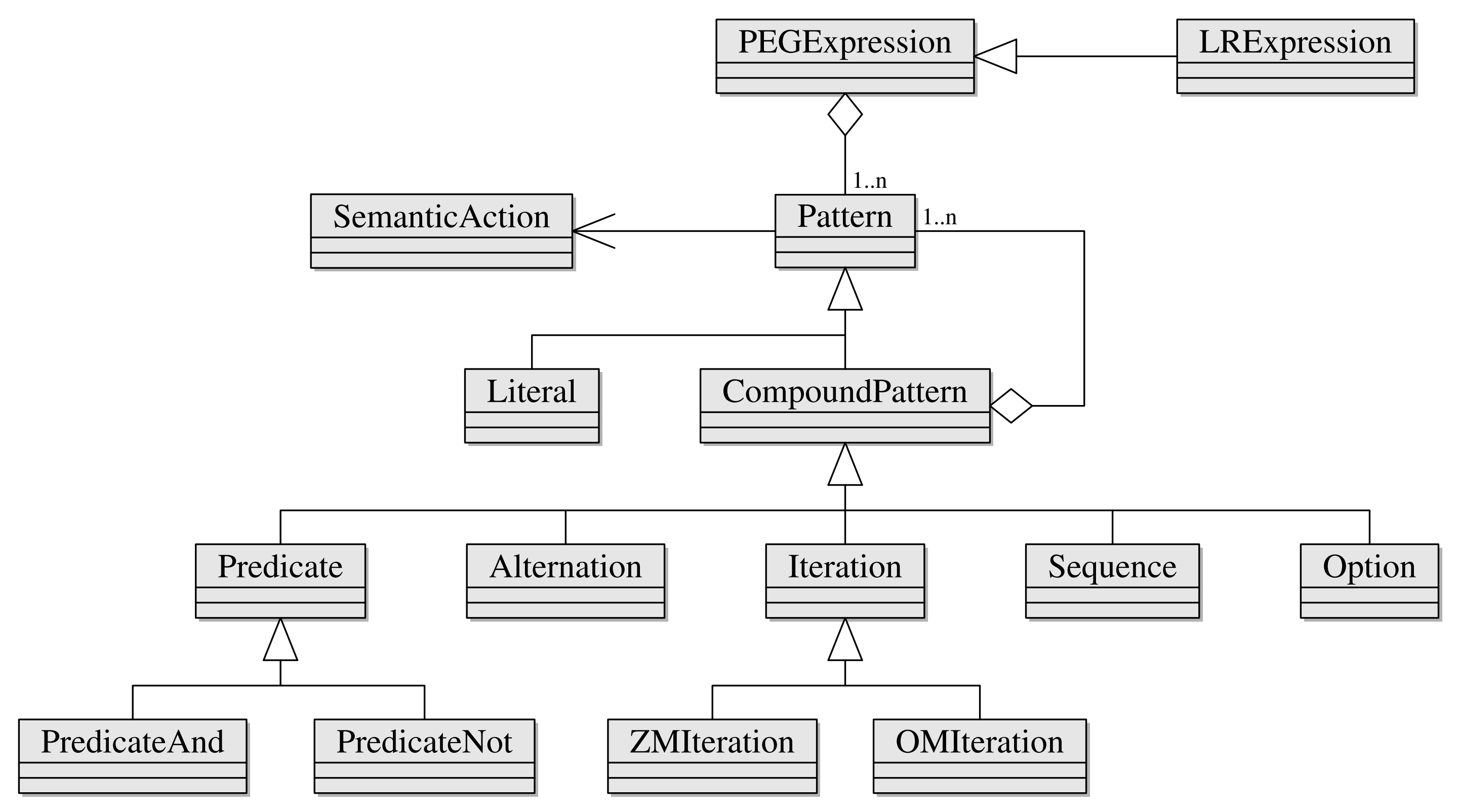
Implementation

The parsing expression grammar can easily be translated into a parser by using the recursive descent method. However, there is a problem with backtracking. When one of the alternatives fails, it is necessary to consider the next one, which may require a recalculation. It is possible to apply memoization of previous calculations, thus reducing the computational complexity to linear. It is also possible to make pegs using a parser machine.

Translators using parsing expressions have been implemented in many programming languages, including C++. C++ implementations use templates and overloading operators. In this way, the description of the language are implemented directly in the code of the translator program. However, there is a problem with the understanding of the description, since it needs an adaptation the notation of parsing expressions to C++ syntax. In addition, the compiler messages associated with advanced metaprogramming can be obscure.

This implementation is based on converting the specification language script to the C++ source code. This way, the language description is clear and can be build high performance parser. By separating the description of the parser from its executable form is possible to support the language designer with the tools built into the specification language translator. The generated parser code is syntactically correct, so the language designer does not need to look C++ compiler error messages. The generated code can include optimizations related to the advanced capabilities of C++.

Parsing expression grammar representation



The diagram UML presents classes for handling parsing expressions. The class `PEGExpression` is the most important. It aggregates objects from the pattern responsible for recognizing parsing expressions - both as composite patterns (`CompoundPatter`) and simple patttern (`Literal`). Rest classes realizes the

recognition of the following types of parsing predicates:

- `Predicate`:
  - and `PredicateAnd` (&e),
  - not `PredicateNot`) (!e)
- ordered selection - `Alternation` (e_1/e_2),
- literal - `Literal` (e?),
- sequence - `Sequence` ($e_1 e_2$)
- optional expression - `Optional`,
- iteration - `Iteration`:
  - `ZMIteration` ($e*$),
  - `OMIteration` ($e+$).

This class allows to store the results of previous computations, while only the results that are reused in the processing of alternative pattern paths are stored to reduce memory occupancy.
The `Pattern` class is associated with the `SemanticAction` class. It implements actions, also known as semantic actions, in response to the recognition in the stream of characters of the relevant pattern. Typically, semantic actions are insertions in the target programming language code. Semantic actions can provide an alternative way to build a syntax tree when the standard implementation provided by the library does not meet the requirements of the user.

Grammar

```
expr <- (term ('+'/'-') expr / term) !.
term <- primary ('*'/'/') term / primary
primary <- '(' expr ')' / int
int <- [0-9]
```

The parsing table for expression `(2+5)*4`

expr	28	7						
term	7	2		5			4	
primary	7	2		5			4	
int		2		5			4	
Input	'('	'2'	+	'5'	)	'*'	'4'	#

Ministerstwo Nauki i Szkolnictwa Wyższego

Organizacja IX Konferencji Modelowanie Matematyczne w Fizyce i Technice (MMFT 2017)
- zadanie finansowane w ramach umowy 829/P-DUN/2017
ze środków Ministra Nauki i Szkolnictwa Wyższego przeznaczonych
na działalność upowszechniającą naukę.

POLITECHNIKA CZĘSTOCHOWSKA