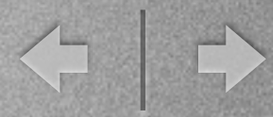



Parser Generation in Perl

an Overview and Available Tools

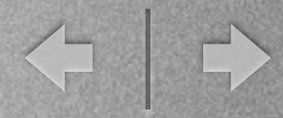
Hugo Areias[1]

Alberto Simões[2], Pedro Rangel Henriques[1], Daniela da Cruz[1]

University of Minho[1], Polytechnic Institute of Porto[2]

September 10th, 2010

Parser Generation



A parser generator is a program that based on a grammar creates a parser for the language defined by that grammar.

Typically parser generators produce parsers in languages such as Pascal, C or Java

…and why not in Perl?

Perl is widely known for its text processing mechanisms mainly based on regular expressions.

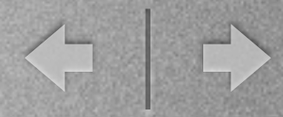
Our motivation is to review and compare existing tools that produce parsers in Perl:

- Parse::RecDescent
- Parse::Yapp
- Parse::Eyapp
- Parse::Earley
- HOP::Parser
- GNU Bison
- Perl-Byacc
- Regexp::Grammars
- …

Comparative Study

- Tested modules
  - Parse::RecDescent (*recursive-descent parsing*)
  - Parse::Yapp (*bottom-up parsing*)
  - Parse::Eyapp (*bottom-up parsing*)
  - Regexp::Grammars (*recursive-descent parsing*)
- Grammars used
  - Swedish chef (lexical language)
  - Lavanda (complex Domain Specific Language)
  - S-expressions (highly recursive)
- Parameters evaluated
  - Time evolution for the 12 (4x3) solutions (using time command)
  - Memory consumption for Lavanda language (using massif tool, from valgrind)
  - Grammars supported and Usability Analysis

Lavanda Grammar



The Lavanda grammar will be used for the tests in this presentation.

Lavanda	→	Cabec Sacos
Cabec	→	date IdPR
Sacos	→	Saco '.'
		l Sacos Saco '.'
Saco	→	num IdCli Lotes
Lotes	→	Lote
		l Lotes Lote
Lote	→	Tipo Qt
Tipo	→	Classe Tinto Fio
IdPR	→	id
IdCli	→	id
Qt	→	num
Classe	→	corpo l casa
Tinto	→	br l cor
Fio	→	alg l la l fib

Time Consumption

Parsing time (seconds) evolution for Lavanda test files

Input Lines	Parse::Yapp	Parse::Eyapp	Parse::RecDescent	Regexp::Grammars
10	0.031	0.090	0.123	0.069
100	0.115	0.184	0.258	0.163
1000	1.240	1.380	4.041	1.399
10000	34.896	37.640	331.814	out of memory
100000	> 2488.348	> 4973.639		
1000000				

Memory Consumption

Memory (megabytes) used by the generated parsers when parsing Lavanda test files

Input Lines	Parse::Yapp	Parse::Eyapp	Parse::RecDescent	Regexp::Grammars
10	0.933	3.866	3.583	3.490
100	1.934	4.867	4.607	22.545
1000	12.141	15.214	15.175	181.809
10000	108.697	131.242	115.383	out of memory
100000				
1000000				

Grammars Supported & Usability Analysis

Grammars Supported

Module	Supported Grammars	Grammar Legibility	Attribute Grammars	Abstract Syntax Tree	Semantic Actions	Lexical Analyser
Parse::Yapp	LALR	+	No	No	+	No
Parse::Eyapp	LALR	+	No	Yes	++	No
Parse::RecDescent	LL(1)	++	No	No	++	Yes
Regexp::Grammars	LL(1)	++	No	No	++	Yes

Usability

Module	Debugging	Generated Parser Legibility	Integration with external code	Development Time
Parse::Yapp	+/—	+/—	++	+/—
Parse::Eyapp	+/—	+/—	+	+/—
Parse::RecDescent	+	NA	+	—
Regexp::Grammars	++	NA	+/—	— —

Conclusions

- No support for attribute grammars

- Recursive-descent parsers lack the efficiency to parse large input streams

- Perl is lacking an efficient lexical analyser (storing the input in memory increases the resource consumptions and decreases the performance of the parser)

- Parsing time highly increases with larger input streams

- Other languages provide much better solutions (ex.: C)

Future Work

Possible solutions to overcome the identified drawbacks:

- Create a new Perl module (probably would have the same limitations)

- Combine the Perl modules with tools written in another languages

- Create a backend for an existing tool (ex.: AnTLR, LISA)