**Top Down Operator Precedence Parsing**

**Introduction**

“Parsing is the process of structuring a linear representation in accordance with a given grammar” (Grune and Jacobs, 2008). Such broad abstraction of definition implies great application area therefore it is of critical value to implement it in optimal way into code.

In 1973 Vaughan Pratt suggested his method for parsing. He called it top down operator precedence, today it is also known as pratt parser.

His motivation was that in process of language designing and implementing some preoccupy with syntax while others deny its use. His solution was compromise between those two, parser which when writing feels more like grammar then program, in other words he embedded syntax directly into program (Pratt, 1973).

**Related work**

Automaton theory found great application in parsing. Diverse grammars are put in use of defining languages and automatons are used for construction of parsers (mostly LR and its derivatives and recursive descent).

Through years need for parser generators has grown. Yacc[[1]](#footnote-1) and Bison[[2]](#footnote-2) are examples of parser generators but there are many more[[3]](#footnote-3). Such solution is adequate when in need for parser where optimization is not issue or if user is not in position // ili u stanju ili ne zeli to write his own parser. Drawback of generators is that they can not be fined tuned as hand written parsers consequently hand written parsers are sometimes more suitable.

Pratts method is one way to solve this problem. He stated that his method is great for error handling and that it is trivial to implement. This

There are many alternatives to pratt parser.

Parsers can be divided into two main groups; top-down and bottom-up (Grune and Jacobs, 2008).

llvm

Packrat, lr, lalr parseri, alternativa za yacc i bisson kad se zele hendlat errori

**Algorithm**

Algorithm is based on operator precedence and recursive descent (Crockford, 2007). Tokens with same properties will be part of same class and we will use functions as main callable units.

Pratt used following syntax for variables and functions

* lbp – left binding power
* rpb – right binding power
* nud() – null denotation

Function which handles prefixes. This function is called when for given token we can predict which tokens need to follow it and bind them (ie in most languages after if we can expect Boolean expression)

* led() – left denotation

Function which handles infixes and suffixes. This function is called when for given token we can bind tokens that are left to it and optionally right.

(Pratt, 1973)

Each token class can contain function nud and led. If logicaly nud or led are not necessary they can be used for error handling.

Parser driver is next function:****

We can assume that next token is loaded in variable *token* by assigning *next token*.

Function *expression* is called with argument equal to zero at the start of parsing tokens. Variable *t* acts as buffer for previous token where variable *token* is current token. When function is called it tries to match tokens for given prefix token *t* and assign given value to variable *left*. Then it tries to bind ????? again to the variable left for as long as the current *rpb* is lower than current tokens *lpb*.

for full implementation of this algorithm I suggest this[[4]](#footnote-4) article. In this explanation I heavily relied on that article.

**Problem and solution**

Koji problemi se mogu svest na ovaj

Recursive descent parsers have drawback when parsing left recursive grammars. For any production that has the following structure

S → Sa | a

algorith for recursive parsing would look like this



Problem with this algorith is that it will loop infinitely in recursion. This is generaly solved by rewriting algorithm like this:



Pratt used similar, more general aproach using recursion and loop

Power of pratt parser is in this simplicity. Every token is passed through this function. When will this function return value is controlled by rbp value that is passed as argument

As I stated; parsing is big domain. In the world of compilers

Besides using parsers for parsing data it can be put to use in favour of data compression, generating machine instructions in compilers and as a support for logic language.

Data compression is achieved by swapping big and common chunks with smaller. This swapping (or maping) is threated same as in grammar transitions therefore parser can be utilized for this task. Generating instructions is also done by obeying grammar rules that (Grune and Jacobs, 2008).

**Experiments**

There have been made many articles on the topic of *top down operator precendance* algorithm. One of implementations using toy language is extensively explained in article[[5]](#footnote-5).

Pratt stated that his algorithm has been implemented in SCRATCH-PAD and in MACSYMA (Pratt, 1973).

Crockford exploited this algorithm for his application JSlint[[6]](#footnote-6). He also wrote a paper on how it works[[7]](#footnote-7) and displayed code publicly[[8]](#footnote-8).

Another interesting use is for application Desmos[[9]](#footnote-9). They made an article[[10]](#footnote-10) on how they did and mentioned that speed was positive side while negatives were memory overflow because of recursion calls and no guarantee that parser will work intendedly for every case (Lantsman, 2018).

**Conclusion**

Jel cemu ovo

Za sta je dobar

Top down operator precedence parsing is best utilized when is in service of dynamic and functional programming languages (Crockford, 2007).

Error handling (pratt)

**References**

PRATT, Vaughan R. 1973. "Top down operator precedences". *Proceedings of the 1st annual ACM SIGACTSIGPLAN symposium on Principles of programming languages*, 41-51

GRUNE, Dick and Ceriel J.H. JACOBS. 2008. *Parsing Techniques*. New York: Springer Science+Business Media, LLC.

AHO, Alfred V. et al. 1986. *Compilers: Principles, Techniques, & Tools*. Boston: Addison-Wesley

CROCKFORD, Douglas. 2007. *Top Down Operator Precedence*. Available at: https://crockford.com/javascript/tdop/tdop.html (Accessed: 13 January 2021).

LANTSMAN, Denis. 2018. *How Desmos uses Pratt Parsers*. Available at: <https://engineering.desmos.com/articles/pratt-parser/> (Accessed: 13 January 2021).

KLADOV, Aleksey. 2020. Simple but Poweful Pratt Parsing. Available at: <https://matklad.github.io/2020/04/13/simple-but-powerful-pratt-parsing.html> (Accessed: 13 January 2021)

BENDERSKY, Eli. 2010. Top-Down operator precedence parsing. Available at: <https://eli.thegreenplace.net/2010/01/02/top-down-operator-precedence-parsing> (Accessed: 14 January 2021)

1. http://dinosaur.compilertools.net/yacc/index.html [↑](#footnote-ref-1)
2. https://www.gnu.org/software/bison/ [↑](#footnote-ref-2)
3. see <http://catalog.compilertools.net/lexparse.html> and <https://java-source.net/open-source/parser-generators> [↑](#footnote-ref-3)
4. https://eli.thegreenplace.net/2010/01/02/top-down-operator-precedence-parsing [↑](#footnote-ref-4)
5. https://eli.thegreenplace.net/2010/01/02/top-down-operator-precedence-parsing# [↑](#footnote-ref-5)
6. http://jslint.com/ [↑](#footnote-ref-6)
7. https://crockford.com/javascript/tdop/tdop.html [↑](#footnote-ref-7)
8. https://github.com/douglascrockford/JSLint [↑](#footnote-ref-8)
9. https://www.desmos.com/ [↑](#footnote-ref-9)
10. https://engineering.desmos.com/articles/pratt-parser/ [↑](#footnote-ref-10)