Runi: Lexical and Syntax Analyzer

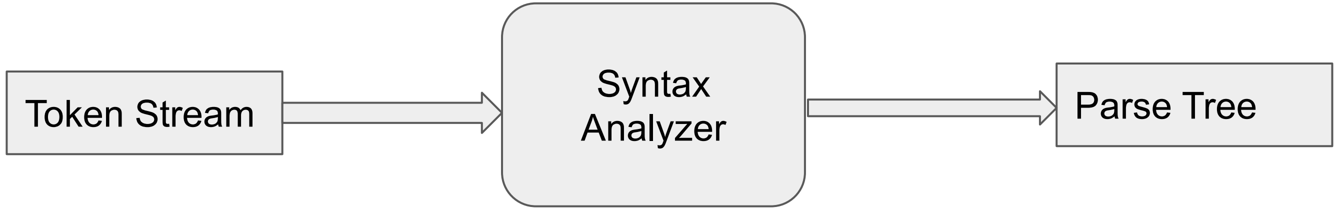
Nipun Wahi & Rupanshu Soi

In a Nutshell

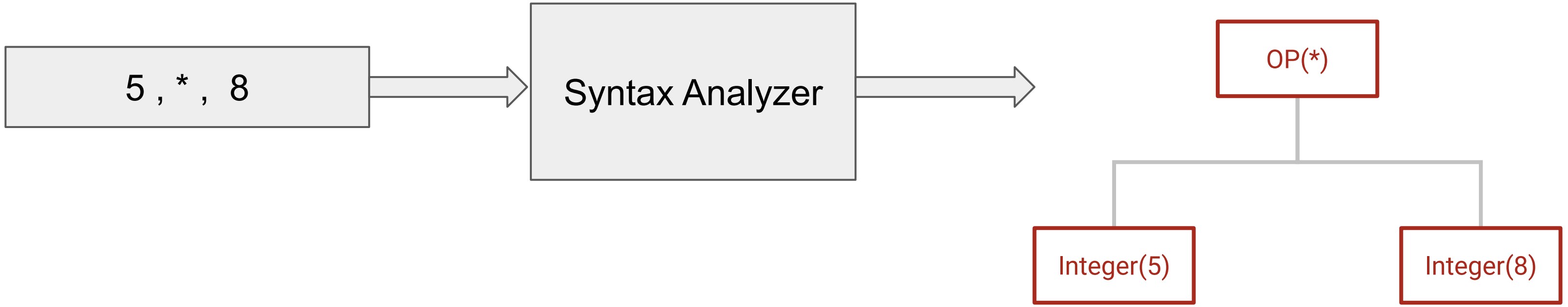
- Implemented a syntax analyzer, i.e. parser
- Predictive recursive descent parsing
- Without using a parser generator like Yacc or Bison
- Targeting a small subset of C
- Using Go
- Visualized parse trees using Graphviz

Syntax Analysis

Second stage in compilation



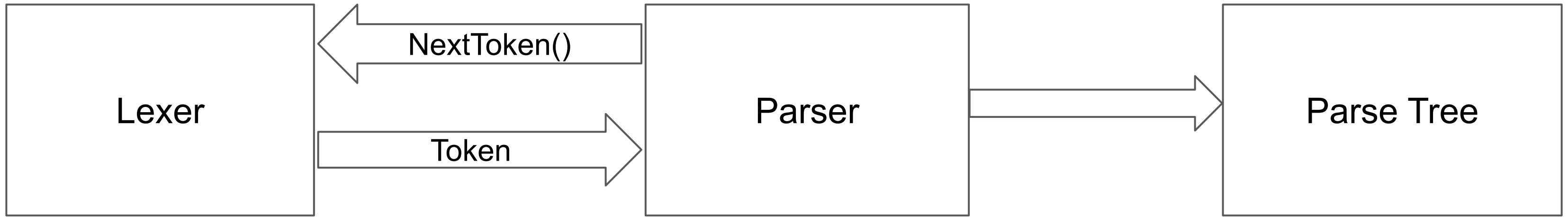
Example



Why Not Use Bison?

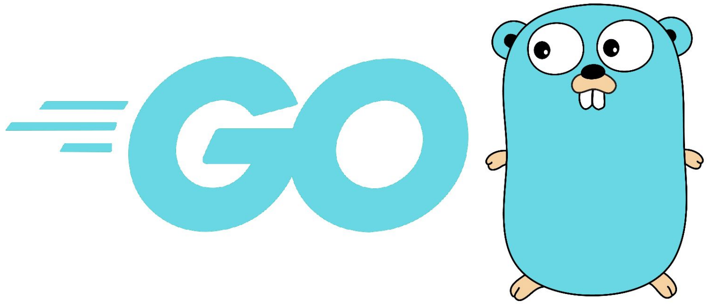
- Understand parsing by implementing it
- Understand pros and cons of different parsing techniques
- First steps of building own compiler
- Understanding parser generators like Bison

High-Level Structure



Implementation in Go

- C-like syntax
- Static typing for early detection of errors
- Memory safety, garbage collection make it safer and easier to use than C
- Improved building and testing than C
- Wanted to learn a new language

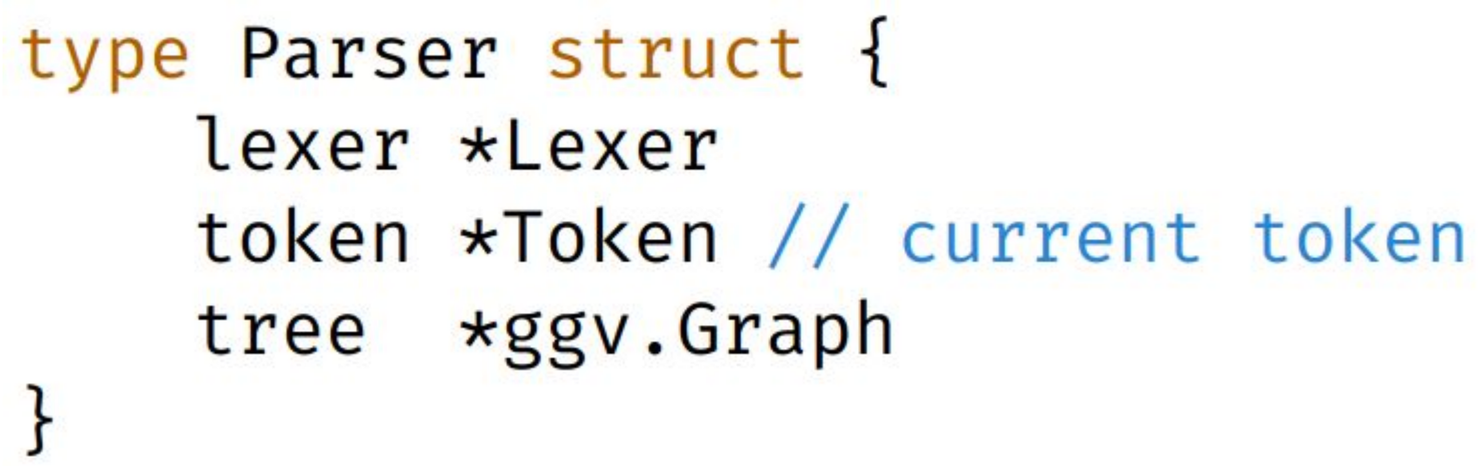


Graphviz for Visualization

- Popular open-source graph drawing library
- Generates graphs from a specification in the DOT language
- Go binding written by Walter Schulze
  https://github.com/awalterschulze/gographviz

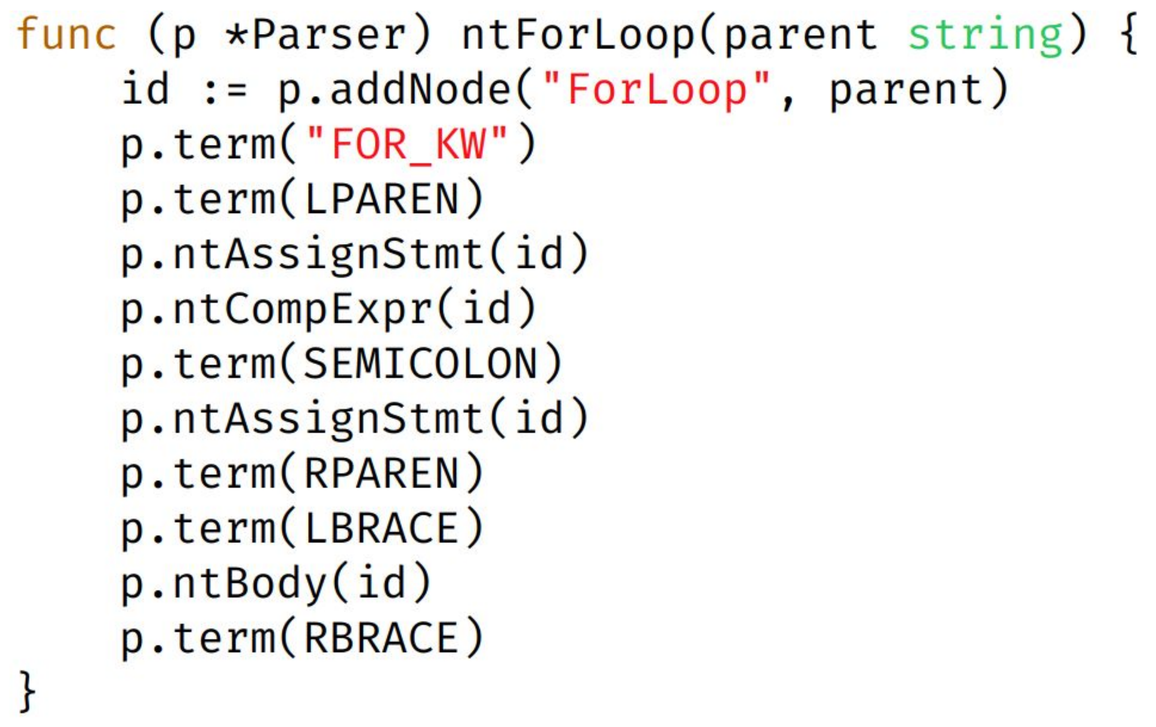
Parser Object (Struct)

- Stateful
- Stores the parse tree data structure
- Go is not an OO language



```go
type Parser struct {
    lexer *Lexer
    token *Token // current token
    tree  *ggv.Graph
}
```

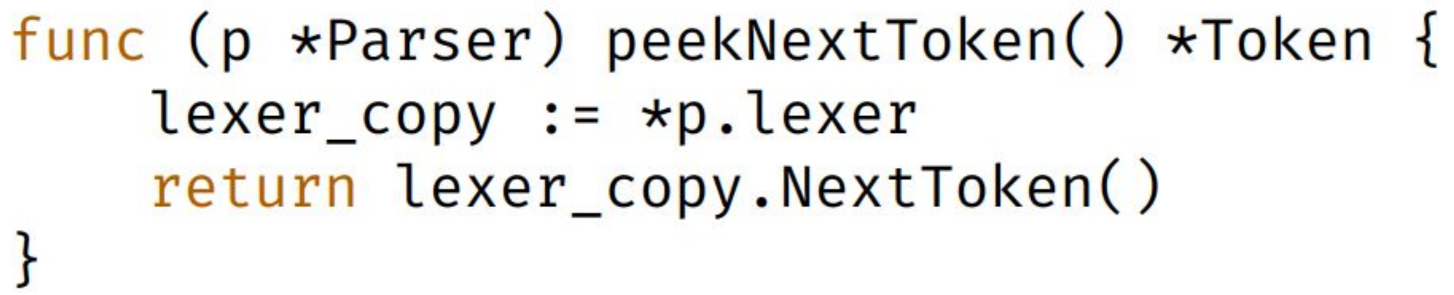
Parsing Example: For Loop



```
func (p *Parser) ntForLoop(parent string) {
	id := p.addNode("ForLoop", parent)
	p.term("FOR_KW")
	p.term(LPAREN)
	p.ntAssignStmt(id)
	p.ntCompExpr(id)
	p.term(SEMICOLON)
	p.ntAssignStmt(id)
	p.term(RPAREN)
	p.term(LBRACE)
	p.ntBody(id)
	p.term(RBRACE)
}
```

Lookahead Implementation

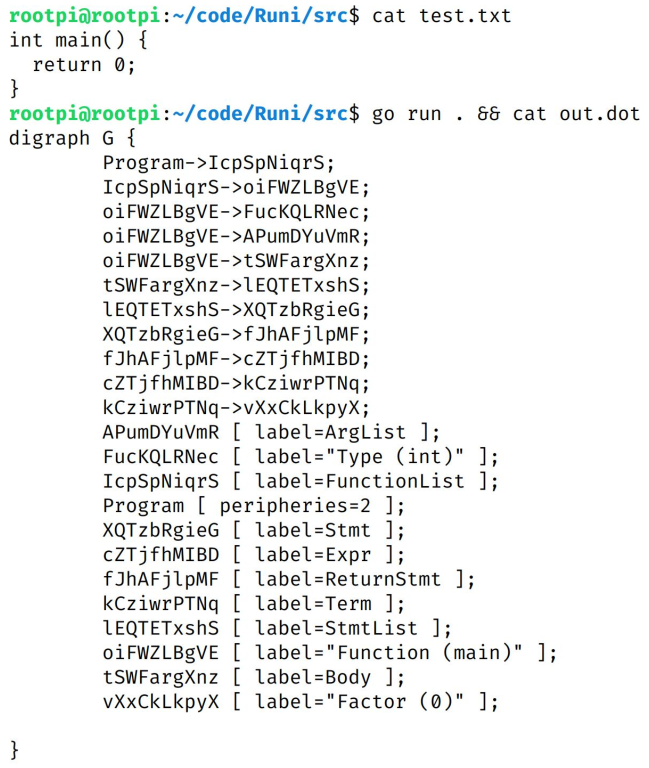
- By creating a copy of the parser’s state
- Good encapsulation and abstraction
- Easily extendible for greater lookahead values



```
func (p *Parser) peekNextToken() *Token {
    lexer_copy := *p.lexer
    return lexer_copy.NextToken()
}
```

Parser Output: DOT Specification

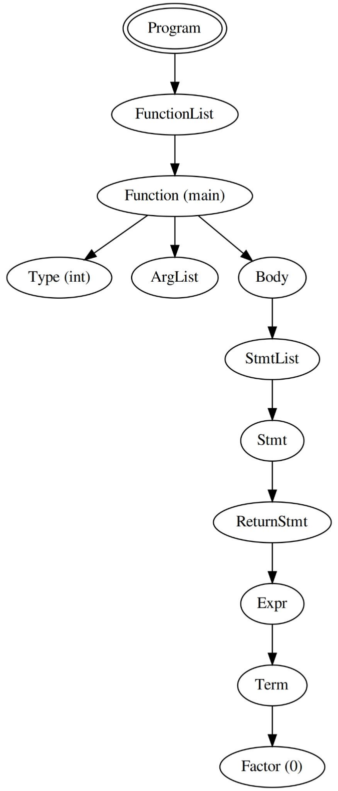
- DOT is a declarative language
- Randomly generated IDs
- Can fail! (but with astronomically low probability)



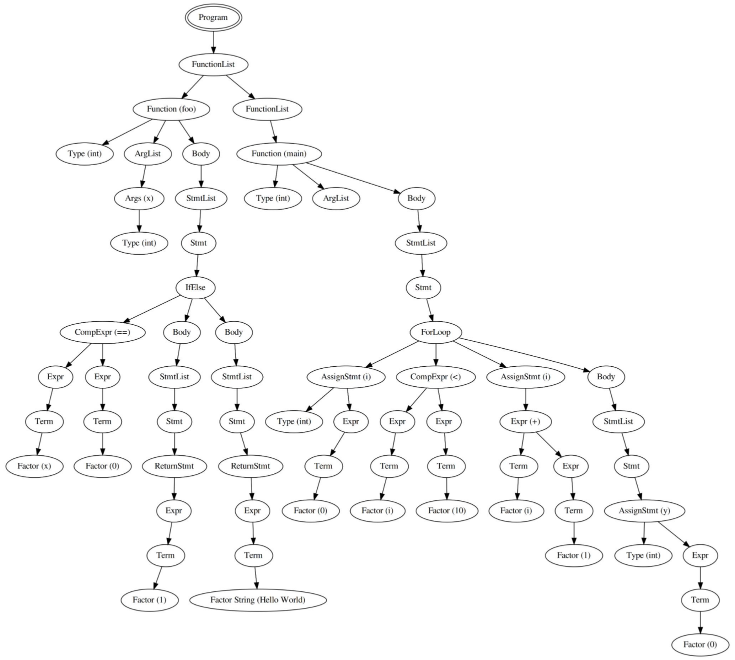
```
rootpi@rootpi:~/code/Runi/src$ cat test.txt
int main() {
  return 0;
}
rootpi@rootpi:~/code/Runi/src$ go run . && cat out.dot
digraph G {
        Program->IcpSpNiqrS;
        IcpSpNiqrS->oiFWZLBgVE;
        oiFWZLBgVE->FucKQLRNec;
        oiFWZLBgVE->APumDYuVmR;
        oiFWZLBgVE->tSWFargXnz;
        tSWFargXnz->lEQTETxshS;
        lEQTETxshS->XQTzbRgieG;
        XQTzbRgieG->fJhAFjlpMF;
        fJhAFjlpMF->cZTjfhMIBD;
        cZTjfhMIBD->kCziwrPTNq;
        kCziwrPTNq->vXxCkLkpyX;
        APumDYuVmR [ label=ArgList ];
        FucKQLRNec [ label="Type (int)" ];
        IcpSpNiqrS [ label=FunctionList ];
        Program [ peripheries=2 ];
        XQTzbRgieG [ label=Stmt ];
        cZTjfhMIBD [ label=Expr ];
        fJhAFjlpMF [ label=ReturnStmt ];
        kCziwrPTNq [ label=Term ];
        lEQTETxshS [ label=StmtList ];
        oiFWZLBgVE [ label="Function (main)" ];
        tSWFargXnz [ label=Body ];
        vXxCkLkpyX [ label="Factor (0)" ];

}
```

Example Output: Small Parse Tree

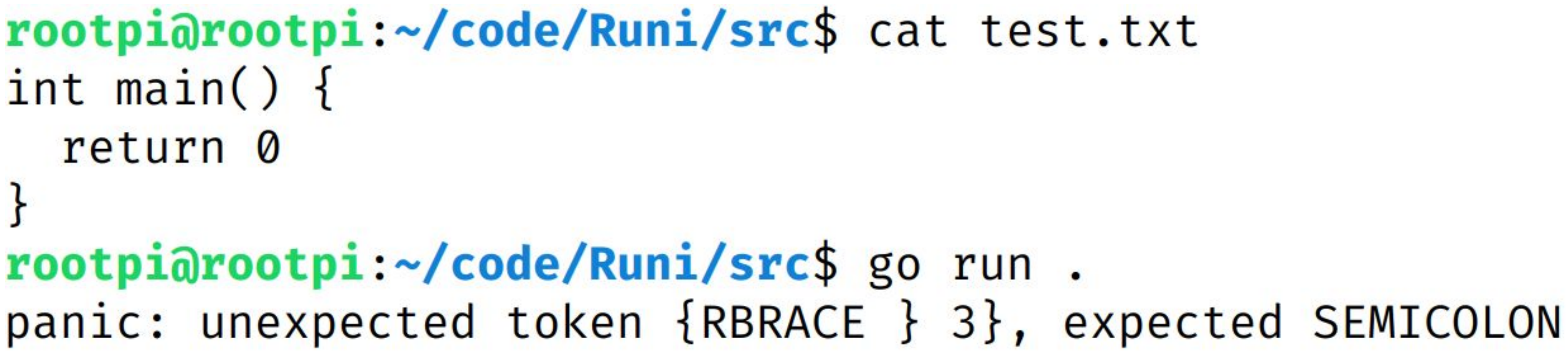
- All program information is present in the tree



Example Output: Large Parse Tree



Example Output: Error Reporting



```
rootpi@rootpi:~/code/Runi/src$ cat test.txt
int main() {
  return 0
}
rootpi@rootpi:~/code/Runi/src$ go run .
panic: unexpected token {RBRACE } 3}, expected SEMICOLON
```

What We Learnt: Advantages of RDP

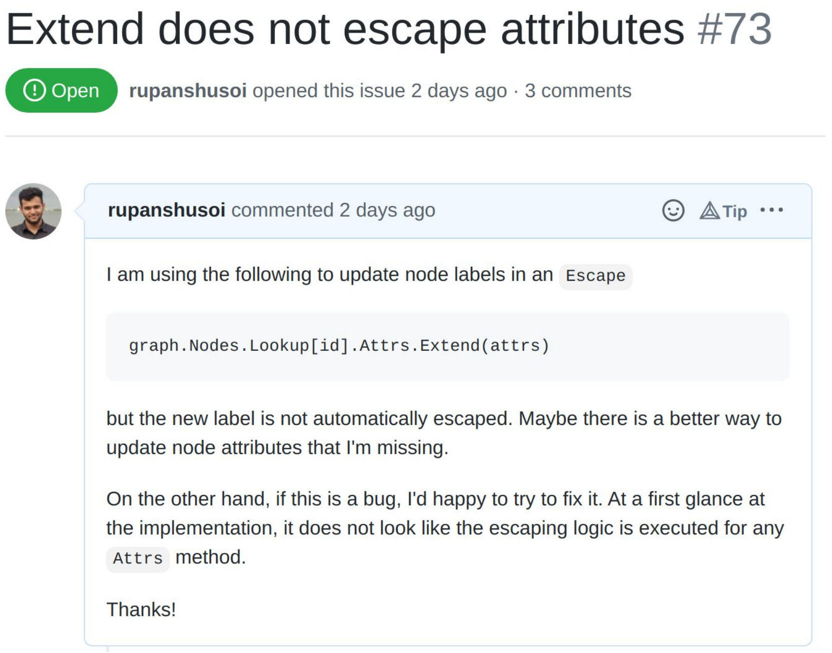
- Straightforward implementation
- Choice of implementation: backtracking or predictive
- LL(k) grammars can be handled

What We Learnt: Drawbacks of RDP

- Implementation is coupled with the grammar
- Parse table is hard-coded into the control-flow statements
- Grammar may need left factoring
- Infeasible for large grammars
- That’s why parser generators use bottom-up parsing!

Open-source Contribution: Found a Bug! 🐛

- Found a bug in the Go binding for Graphviz
- Confirmed by the author



Extend does not escape attributes #73

Open rupanshusoi opened this issue 2 days ago · 3 comments

rupanshusoi commented 2 days ago

I am using the following to update node labels in an `Escape`

```
graph.Nodes.Lookup[id].Attrs.Extend(attrs)
```

but the new label is not automatically escaped. Maybe there is a better way to update node attributes that I'm missing.

On the other hand, if this is a bug, I'd happy to try to fix it. At a first glance at the implementation, it does not look like the escaping logic is executed for any `Attrs` method.

Thanks!

Future Improvements

- Extend grammar to support more constructs
- Better error handling
- Symbol table (and later stages of compilation)

Work Plan Recap

- Lexer before mid-sem ✔️
- Parser after mid-sem (with parse tree visualization) ✔️

Thank You