INSTITUTO DE INFORMÁTICA
Universidade Federal de Goiás

Compiladores
Go Golang e Rust

Alunos:
Arthur Gomes de Siqueira
Joao Batista de Oliveira Netto

INSTITUTO DE INFORMÁTICA
UFG

UFG
UNIVERSIDADE FEDERAL DE GOIÁS

Go Golang

Breve História

- Lançado em setembro de 2007 por Robert Griesemer (criador do V8), Rob Pike e Ken Thompson.
- Google insatisfeita
- Propósito da linguagem:
  - Criar uma linguagem de programação de fácil uso.
  - Eliminar a lentidão
  - Melhorar os processos de desenvolvimento no Google, tornando-os mais produtivos e escaláveis.
- Atualmente:
  - Resolveu perfeitamente este problema
  - Conhecida pela comunidade de desenvolvedores como uma linguagem simples e progressiva.
  - Melhorias no seu tempo de execução, novas bibliotecas
  - Inclusão de frameworks, bibliotecas e outras ferramentas.

Processo de compilação e interpretação

Go Compiler

- É um compilador para a linguagem Go
- O compilador lê do código-fonte e gera um arquivo binário ou executável que usamos para executar nosso programa.
- O Go Compiler fornece três benefícios:
  - Checagem de erro
  - Optimizador
  - Facilidade de implementação

Há duas maneiras de executarmos o código:

- Esses comandos compilam esse arquivo para um binário com o mesmo nome
- Depois poderá ser executado em máquina *NIX
- Comandos:
  - go build main.go
  - ./main

Já o comando abaixo, compila e executa o binário facilitando assim para o desenvolvedor ver o output de seu código:

- go run main.go

Scanner

- A primeira etapa de cada compilador é quebrar o texto do código-fonte bruto em tokens.

- Os tokens podem ser palavras-chave, strings, nomes de variáveis, nomes de funções, etc.

- Cada “palavra” de programa válida é representada por um token. Exemplo: Em Go, temos “package”, “main”, “func”, etc.

- Cada token é representado por sua posição, tipo e texto bruto no Go.

Exemplo

Entrada

```
package main

import "fmt"

func main() {
	fmt.Println("Hello, world!")
}
```

Saída

```
1   1:1   package "package"
2   1:9   IDENT   "main"
3   1:13  ;       "\n"
4   2:1   import  "import"
5   2:8   STRING  "\"fmt\""
6   2:13  ;       "\n"
7   3:1   func    "func"
8   3:6   IDENT   "main"
9   3:10  (       ""
10  3:11  )       ""
11  3:13  {       ""
12  4:3   IDENT   "fmt"
13  4:6   .       ""
14  4:7   IDENT   "Println"
15  4:14  (       ""
16  4:15  STRING  "\"Hello, world!\""
17  4:30  )       ""
18  4:31  ;       "\n"
19  5:1   }       ""
20  5:2   ;       "\n"
21  5:3   EOF     ""
```

```
package main

import "fmt"

func main() {
	fmt.Println("Hello, world!")
}
```

```
1   1:1   package "package"
2   1:9   IDENT   "main"
3   1:13  ;       "\n"
4   2:1   import  "import"
5   2:8   STRING  "\"fmt\""
6   2:13  ;       "\n"
7   3:1   func    "func"
8   3:6   IDENT   "main"
9   3:10  (       ""
10  3:11  )       ""
11  3:13  {       ""
12  4:3   IDENT   "fmt"
13  4:6   .       ""
14  4:7   IDENT   "Println"
15  4:14  (       ""
16  4:15  STRING  "\"Hello, world!\""
17  4:30  )       ""
18  4:31  ;       "\n"
19  5:1   }       ""
20  5:2   ;       "\n"
21  5:3   EOF     ""
```

Parser

- Depois que o código-fonte foi analisado pelo Scanner, ele será passado para o Parser.

- O Parser é uma fase do compilador que converte os tokens em uma árvore de sintaxe abstrata, em inglês Abstract Syntax Tree (AST).

- O AST é uma representação estruturada do código-fonte. No AST poderemos ver a estrutura do programa, como funções e declarações de constantes.

Exemplo

Entrada

```
package main

import "fmt"

func main() {
	fmt.Println("Hello, world!")
}
```

Saída

```
0  *ast.File {
1  .  Package: 1:1
2  .  Name: *ast.Ident {
3  .  .  NamePos: 1:9
4  .  .  Name: "main"
5  .  }
6  .  Decls: []ast.Decl (len = 2) {
7  .  .  0: *ast.GenDecl {
8  .  .  .  TokPos: 3:1
9  .  .  .  Tok: import
10 .  .  .  Lparen: -
11 .  .  .  Specs: []ast.Spec (len = 1) {
12 .  .  .  .  0: *ast.ImportSpec {
13 .  .  .  .  .  Path: *ast.BasicLit {
14 .  .  .  .  .  .  ValuePos: 3:8
15 .  .  .  .  .  .  Kind: STRING
16 .  .  .  .  .  .  Value: "\"fmt\""
17 .  .  .  .  .  }
18 .  .  .  .  .  EndPos: -
19 .  .  .  .  }
20 .  .  .  }
21 .  .  .  Rparen: -
22 .  .  }
```

```
23 .  .  1: *ast.FuncDecl {
24 .  .  .  Name: *ast.Ident {
25 .  .  .  .  NamePos: 5:6
26 .  .  .  .  Name: "main"
27 .  .  .  .  Obj: *ast.Object {
28 .  .  .  .  .  Kind: func
29 .  .  .  .  .  Name: "main"
30 .  .  .  .  .  Decl: *(obj @ 23)
31 .  .  .  .  }
32 .  .  .  }
33 .  .  .  Type: *ast.FuncType {
34 .  .  .  .  Func: 5:1
35 .  .  .  .  Params: *ast.FieldList {
36 .  .  .  .  .  Opening: 5:10
37 .  .  .  .  .  Closing: 5:11
38 .  .  .  .  }
39 .  .  .  }
40 .  .  .  Body: *ast.BlockStmt {
41 .  .  .  .  Lbrace: 5:13
42 .  .  .  .  List: []ast.Stmt (len = 1) {
43 .  .  .  .  .  0: *ast.ExprStmt {
44 .  .  .  .  .  .  X: *ast.CallExpr {
45 .  .  .  .  .  .  .  Fun: *ast.SelectorExpr {
46 .  .  .  .  .  .  .  .  X: *ast.Ident {
47 .  .  .  .  .  .  .  .  .  NamePos: 6:2
```

```
49      48  . . . . . . . . . .  Name: "fmt"
50      49  . . . . . . . . . }
51      50  . . . . . . . . . Sel: *ast.Ident {
52      51  . . . . . . . . . .  NamePos: 6:6
53      52  . . . . . . . . . .  Name: "Println"
54      53  . . . . . . . . . }
55      54  . . . . . . . . }
56      55  . . . . . . . . Lparen: 6:13
57      56  . . . . . . . . Args: []ast.Expr (len = 1) {
58      57  . . . . . . . . .  0: *ast.BasicLit {
59      58  . . . . . . . . . .  ValuePos: 6:14
60      59  . . . . . . . . . .  Kind: STRING
61      60  . . . . . . . . . .  Value: "\"Hello, world!\""
62      61  . . . . . . . . . }
63      62  . . . . . . . . }
64      63  . . . . . . . . Ellipsis: -
65      64  . . . . . . . . Rparen: 6:29
66      65  . . . . . . . }
67      66  . . . . . . }
68      67  . . . . . }
69      68  . . . . . Rbrace: 7:1
70      69  . . . . }
71      70  . . . }
72      71  . . }
```

```
0  *ast.File {
1  .  Package: 1:1
2  .  Name: *ast.Ident {
3  .  .  NamePos: 1:9
4  .  .  Name: "main"
5  .  }
6  .  Decls: []ast.Decl (len = 2) {
7  .  .  0: *ast.GenDecl {
8  .  .  .  TokPos: 3:1
9  .  .  .  Tok: import
10 .  .  .  Lparen: -
11 .  .  .  Specs: []ast.Spec (len = 1) {
12 .  .  .  .  0: *ast.ImportSpec {
13 .  .  .  .  .  Path: *ast.BasicLit {
14 .  .  .  .  .  .  ValuePos: 3:8
15 .  .  .  .  .  .  Kind: STRING
16 .  .  .  .  .  .  Value: "\"fmt\""
17 .  .  .  .  .  }
18 .  .  .  .  .  EndPos: -
19 .  .  .  .  }
20 .  .  .  }
21 .  .  .  Rparen: -
22 .  .  }
```

```
1  1:1   package "package"
2  1:9   IDENT   "main"
3  1:13  ;       "\n"
4  2:1   import  "import"
5  2:8   STRING  "\"fmt\""
6  2:13  ;       "\n"
7  3:1   func    "func"
8  3:6   IDENT   "main"
9  3:10  (       ""
10 3:11  )       ""
11 3:13  {       ""
12 4:3   IDENT   "fmt"
13 4:6   .       ""
14 4:7   IDENT   "Println"
15 4:14  (       ""
16 4:15  STRING  "\"Hello, world!\""
17 4:30  )       ""
18 4:31  ;       "\n"
19 5:1   }       ""
20 5:2   ;       "\n"
21 5:3   EOF     ""
```

```
24  23  .  .  1: *ast.FuncDecl {
25  24  .  .  .  Name: *ast.Ident {
26  25  .  .  .  .  NamePos: 5:6
27  26  .  .  .  .  Name: "main"
28  27  .  .  .  .  Obj: *ast.Object {
29  28  .  .  .  .  .  Kind: func
30  29  .  .  .  .  .  Name: "main"
31  30  .  .  .  .  .  Decl: *(obj @ 23)
32  31  .  .  .  .  }
33  32  .  .  .  }
34  33  .  .  .  Type: *ast.FuncType {
35  34  .  .  .  .  Func: 5:1
36  35  .  .  .  .  Params: *ast.FieldList {
37  36  .  .  .  .  .  Opening: 5:10
38  37  .  .  .  .  .  Closing: 5:11
39  38  .  .  .  .  }
40  39  .  .  .  }
41  40  .  .  .  Body: *ast.BlockStmt {
42  41  .  .  .  .  Lbrace: 5:13
43  42  .  .  .  .  List: []ast.Stmt (len = 1) {
44  43  .  .  .  .  .  0: *ast.ExprStmt {
45  44  .  .  .  .  .  .  X: *ast.CallExpr {
46  45  .  .  .  .  .  .  .  Fun: *ast.SelectorExpr {
47  46  .  .  .  .  .  .  .  .  X: *ast.Ident {
48  47  .  .  .  .  .  .  .  .  .  NamePos: 6:2
```

```
1   1:1   package "package"
2   1:9   IDENT   "main"
3   1:13  ;       "\n"
4   2:1   import  "import"
5   2:8   STRING  "\"fmt\""
6   2:13  ;       "\n"
7   3:1   func    "func"
8   3:6   IDENT   "main"
9   3:10  (       ""
10  3:11  )       ""
11  3:13  {       ""
12  4:3   IDENT   "fmt"
13  4:6   .       ""
14  4:7   IDENT   "Println"
15  4:14  (       ""
16  4:15  STRING  "\"Hello, world!\""
17  4:30  )       ""
18  4:31  ;       "\n"
19  5:1   }       ""
20  5:2   ;       "\n"
21  5:3   EOF     ""
```

```
49      48  . . . . . . . . . .  Name: "fmt"
50      49  . . . . . . . . .  }
51      50  . . . . . . . . .  Sel: *ast.Ident {
52      51  . . . . . . . . . .  NamePos: 6:6
53      52  . . . . . . . . . .  Name: "Println"
54      53  . . . . . . . . .  }
55      54  . . . . . . . .  }
56      55  . . . . . . . .  Lparen: 6:13
57      56  . . . . . . . .  Args: []ast.Expr (len = 1) {
58      57  . . . . . . . . .  0: *ast.BasicLit {
59      58  . . . . . . . . . .  ValuePos: 6:14
60      59  . . . . . . . . . .  Kind: STRING
61      60  . . . . . . . . . .  Value: "\"Hello, world!\""
62      61  . . . . . . . . .  }
63      62  . . . . . . . .  }
64      63  . . . . . . . .  Ellipsis: -
65      64  . . . . . . . .  Rparen: 6:29
66      65  . . . . . . .  }
67      66  . . . . . .  }
68      67  . . . . .  }
69      68  . . . . .  Rbrace: 7:1
70      69  . . . .  }
71      70  . .  }
72      71  .  }
```

```
1   1:1   package "package"
2   1:9   IDENT   "main"
3   1:13  ;       "\n"
4   2:1   import  "import"
5   2:8   STRING  "\"fmt\""
6   2:13  ;       "\n"
7   3:1   func    "func"
8   3:6   IDENT   "main"
9   3:10  (       ""
10  3:11  )       ""
11  3:13  {       ""
12  4:3   IDENT   "fmt"
13  4:6   .       ""
14  4:7   IDENT   "Println"
15  4:14  (       ""
16  4:15  STRING  "\"Hello, world!\""
17  4:30  )       ""
18  4:31  ;       "\n"
19  5:1   }       ""
20  5:2   ;       "\n"
21  5:3   EOF     ""
```

Curiosidades do processo de tradução

Code Generation

- Após os "imports" terem sido resolvidas e os tipos serem checados, temos certeza de que o programa é um código Go válido.
- Inicia o processo de conversão do AST em (pseudo) código de máquina.
- A primeira etapa neste processo é converter o AST em uma representação de nível inferior do programa, o Atribuição Única Estática, em ingles Static Single Assignment(SSA)
- A segunda etapa converte o SSA não específico da máquina em SSA específico da máquina e
- genssa é o código de máquina final gerado.

Exemplo

Entrada

```
package main

import "fmt"

func main() {
	fmt.Println(2)
}
```

Saída

```
b1:
        BlockInvalid (6)
b2:
        v2 (?) = SP <uintptr>
        v3 (?) = SB <uintptr>
        v10 (?) = LEAQ <*uint8> {type.int} v3
        v11 (?) = LEAQ <*int> {"".statictmp_0} v3
        v15 (?) = MOVQconst <int> [1]
        v20 (?) = MOVQconst <uintptr> [0]
        v25 (?) = MOVQconst <*uint8> [0]
        v1 (?) = InitMem <mem>
        v6 (6) = VarDef <mem> {.autotmp_0} v1
        v7 (6) = LEAQ <*[1]interface {}> {.autotmp_0} v2
        v9 (6) = LEAQ <*[1]interface {}> {.autotmp_0} v2
        v16 (+6) = LEAQ <*interface {}> {.autotmp_0} v2
        v18 (6) = LEAQ <**uint8> {.autotmp_0} [8] v2
        v21 (6) = LEAQ <**uint8> {.autotmp_0} [8] v2
        v30 (6) = LEAQ <*int> [16] v2
        v19 (6) = LEAQ <*int> [8] v2
        v23 (6) = MOVOconst <int128> [0]
        v8 (6) = MOVOstore <mem> {.autotmp_0} v2 v23 v6
        v22 (6) = MOVQstore <mem> {.autotmp_0} v2 v10 v8
        v17 (6) = MOVQstore <mem> {.autotmp_0} [8] v2 v11 v22
        v14 (6) = MOVQstore <mem> v2 v9 v17
        v28 (6) = MOVQstoreconst <mem> [val=1,off=8] v2 v14
        v26 (6) = MOVQstoreconst <mem> [val=1,off=16] v2 v28
        v27 (6) = CALLstatic <mem> {fmt.Println} [48] v26
        v29 (5) = VarKill <mem> {.autotmp_0} v27
Ret v29 (+7)
```

Rust

História

- Rust é relativamente recente, a primeira versão estável foi lançada em 2015.
- Considerada a linguagem "mais amada por 5 anos seguidos".
- Originalmente patrocinada pela Mozilla, mas nesse ano surgiu a Rust Foundation e a Google anunciou suporte para Rust no Android.
- Usada amplamente em várias aplicações como Discord, Figma, alguns componentes do Facebook.

Características

- Memory Safety - Não existem ponteiros nulos, soltos ou condições de corrida em Rust.
- Memory Management - Não tem coleta de lixo.
- Ownership - Todos os valores têm um único dono.

Compilador

Analisador Léxico

- O código fonte é dividido em um fluxo de código fonte atômico chamado Tokens em um Analisador Léxico de baixo nível.
- Os Tokens tem apenas informação do tipo e do tamanho do Token, mas não dos dados contidos.
- Após a fase de identificação do Token, é passado por um Analisador Léxico de alto nível para fazer um conjunto de validações e garantir que existem apenas uma cópia imutável de cada string distinta.

Parser

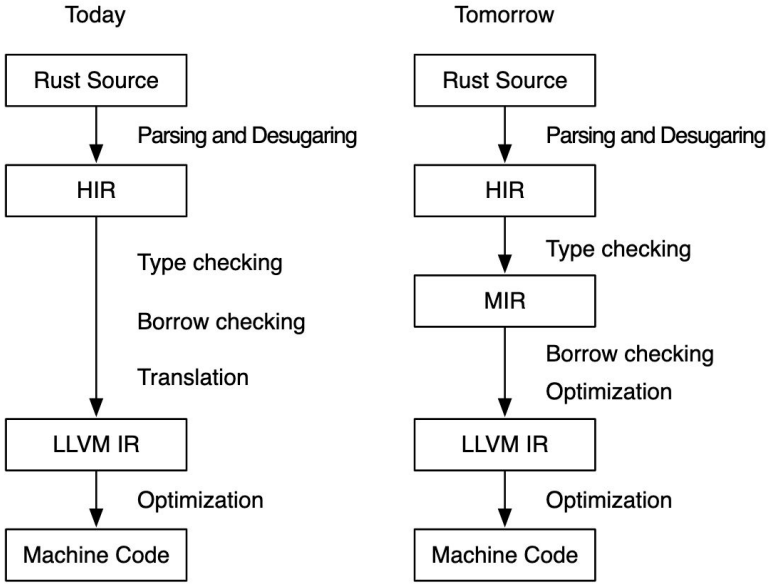
- O Parser traduz os tokens lidos na última fase e monta uma AST.
- Essa fase usa uma abordagem recursiva top-down para análise.
- Durante essa fase, também tem análise semântica. Expansão de macros, validação da AST e resolução de nomes são realizadas nessa fase.
- Essa fase usa uma API específica, *DiagnosticBuilder* para continuar a análise em caso de erros. Emitindo o que houve de errado.

Transformações de Árvores

- Durante a compilação, a AST é transformada em uma HIR(High-Level Intermediate Representation), que é uma forma de representação *compiler-friendly*.
- A HIR é usada para fazer inferência de tipos, checagem de tipos e resolução de traits.
- Após isso, ela é reduzida para uma MIR(Medium-Level Intermediate Representation).
- Essa fase assegura as propriedades de Rust, que todas as variáveis são inicializadas, um objeto não é movido duas vezes, etc. Muitas outras otimizações são feitas nessa fase.

Geração de código

- Rust usa LLVM para gerar o código.
- O primeiro passo é transformar a MIR em uma representação intermediária da LLVM.
- Essa representação intermediária é passada para a LLVM que faz mais otimizações em cima do código e emite um código de máquina. Esse código é basicamente Assembly mas tem alguns tipos de baixo nível adicionado e anotações.
- Os diferentes binários são combinados para produzir um binário final.


Today
Rust Source
Parsing and Desugaring
HIR
Type checking
Borrow checking
Translation
LLVM IR
Optimization
Machine Code
Tomorrow
Rust Source
Parsing and Desugaring
HIR
Type checking
MIR
Borrow checking
Optimization
LLVM IR
Optimization
Machine Code

Exemplos

Source Code

```
1  fn main() {
2      let mut x = 10;
3      x = x * 2;
4  }
```

HIR

```
#[prelude_import]
use ::std::prelude::rust_2015::*;
#[macro_use]
extern crate std;
fn main() { let mut x = 10; x = x * 2; }
```

MIR

```
fn main() -> () {
    let mut _0: ();                      // return place in scope 0 at src/main.rs:1:11: 1:11
    let mut _1: i32;                     // in scope 0 at src/main.rs:2:9: 2:14
    scope 1 {
        debug x => _1;                   // in scope 1 at src/main.rs:2:9: 2:14
    }

    bb0: {
        StorageLive(_1);                 // scope 0 at src/main.rs:2:9: 2:14
        _1 = const 10_i32;               // scope 0 at src/main.rs:2:17: 2:19
        _1 = const 20_i32;               // scope 1 at src/main.rs:3:5: 3:14
        StorageDead(_1);                 // scope 0 at src/main.rs:4:1: 4:2
        return;                          // scope 0 at src/main.rs:4:2: 4:2
    }
}
```

Warning

```
   Compiling playground v0.0.1 (/playground)
warning: value assigned to `x` is never read
 --> src/main.rs:3:5
  |
3 |     x = x * 2;
  |     ^
  |
  = note: `#[warn(unused_assignments)]` on by default
  = help: maybe it is overwritten before being read?
```

Error


Compiling playground v0.0.1 (/playground)
error: expected expression, found `}`
--> src/main.rs:4:1
4 | }
| ^ expected expression

```
   Compiling playground v0.0.1 (/playground)
error: expected one of `!`, `.`, `::`, `;`, `?`, `{`, `}`, or an operator, found `y`
 --> src/main.rs:4:9
  |
4 |     int y = 20;
  |         ^ expected one of 8 possible tokens

error: could not compile `playground` due to previous error
```

Referências

- Golang: Conheça a linguagem do futuro criada pelo Google
- Tutorial GoLang: uma introdução prática ao Go
- The translation process
- UM ESTUDO SOBRE A EFICIˆENCIA DOS COMPILADORES DA LINGUAGEM GO COM O AUXILIO DE ALGORITMOS GEN´ETICOS
- The Go Programming Language
- Overview of the Compiler - Guide to Rustc Development
- Introducing MIR
- Rust Playground