GoGo

A Go compiler written in Go

Michael Lippautz
michael.lippautz@sbg.ac.at

Andreas Unterweger
andreas.unterweger@sbg.ac.at

June 4, 2010

Contents

1 Introduction

Lorem ipsum dolor sit amet...

2 Input Language

Go is a programming language developed by Google, based on a C like syntax and fully specified in [Goo10]. The input language follows the one defined by Go. This results in programs being able to be compiled by the official Go compilers and GoGo.

2.1 Differences to Go

1. GoGo only provides only a **very** basic featureset. Expect every advanced and interesting feature to be missing.
2. GoGo forces the usage of semicolons at the end of statements. This restriction was made to make parsing easier.
3. Go is fully Unicode compatible, while GoGo uses ASCII characters only.
4. Simplified expressions, following Wirth's [Wir96] defintions.

2.1.1 EBNF

Lorem ipsum dolor sit amet...

Atoms

The following listing described the basic atoms that are possible in GoGo programs.

Listing 2.1: Atoms

```
single_char = CHR(32)|...|CHR(127).
char = "'" single_char "'".
string = """ {single_char} """.

digit = "0"|...|"9".
integer = digit {digit}.

letter = "a"|...|"z"|"A"|...|"Z"|"_".
identifier = letter { letter | digit }.
selector = { "." identifier
    | "[" (integer | identifier selector) "]" }.
```

Expressions

Although not as expressive as the ones from Go, these rules define expressions that have comparisons, relations and arithmetical terms.

Listing 2.2: Expressions

```
cmp_op = ">" | "<" | ">=" | "<=" | "==" | "!=".
unary_arith_op = "+" | "-".
binary_arith_op = "*" | "/" .

factor = identifier selector | integer | char | string
    | "(" expression ")" | "!" factor.
term = factor { (binary_arith_op | "&&") factor}.
simple_expression = [ unary_arith_op ] term
    { (unary_arith_op | "||") term }.
expression = "&" identifier selector
    | simple_expression [ cmp_op simple_expression ].
```

Types and Variable Declarations

Listing 2.3: Types

```
type = ([ "[" integer "]" ] identifier | "uint64" | "byte")
    | "string".
var_decl = "var" identifier type [ "=" expression ] ";".
var_decl_list = { var_decl }
```

Structs

Listing 2.4: Structs

```
struct_var_decl = identifier type ";".
struct_var_decl_list = { struct_var_decl }.
struct_decl = "type" identifier "struct" "{"
    struct_var_decl_list "}" ";".
struct_decl_list = { struct_decl }.
```

Statements

Listing 2.5: Statements

```
package_stmt = "package" identifier ";".
import_stmt = "import" string.
import_stmt_list = { import_stmt }.

stmt_sequence = { stmt }
stmt = assignment ";" | function_call_stmt ";" | if_stmt
```

```
    | for_stmt | ";".

assignment = identifier selector "=" expression
if_stmt = "if" expression "{" stmt_sequence "}" [ else_stmt ].
else_stmt = "else" "{" stmt_sequence "}".
for_stmt = "for" [assignment] ";" [expression] ";" [assignment]
    "{" stmt_sequence "}".
```

Functions

Listing 2.6: Functions

```
expression_list = expression { "," expression }.
function_call = "(" [expression_list] ")".
function_call_stmt = identifier selector function_call.

identifier_type = identifier [ "*" ] type.
identifier_type_list = [ identifier_type
    { "," identifier_type } ].
func_decl_head = "func" identifier "(" identifier_type_list ")"  [type].
func_decl = "{" var_decl_list stmt_sequence
    ["return" expression ";"] "}".
func_decl_raw = ";".
func_decl_list = { func_decl_head (func_decl | func_decl_raw)
```

The GoGo Program

Finally, the main program structure is defined by `go_program`. The sequence of the various program parts has been forced to the following to make parsing easier.

Listing 2.7: GoGo Program

```
go_program = package_stmt import_stmt_list struct_decl_list
    var_decl_list func_decl_list.
```

3 Output Language

The output language is Plan-9 assembler [Pik00]. It is a modified version of 64 bit assembly for Intel x86 processors with AT&T syntax that has been created by Bell Labs to be used in their compiler and assembler collection.

4 Scanner

Lorem ipsum dolor sit amet...

5 Parser

Lorem ipsum dolor sit amet...

6 Symbol table

Lorem ipsum dolor sit amet...

6.1 Supported data types

Lorem ipsum dolor sit amet...

6.2 Local variables and offset calculations

Lorem ipsum dolor sit amet...

7 The code generator

Lorem ipsum dolor sit amet...

7.1 Assembly output

Lorem ipsum dolor sit amet...

7.2 Register allocation

Lorem ipsum dolor sit amet...

7.3 The generation of arithmetical expressions

Lorem ipsum dolor sit amet...

7.4 The generation of assignments

Lorem ipsum dolor sit amet...

7.4.1 The generation of conditional expressions

Lorem ipsum dolor sit amet...

7.5 The generation of loops

Lorem ipsum dolor sit amet...

7.6 The generation of functions

Lorem ipsum dolor sit amet...

7.7 Global variable initialization

Lorem ipsum dolor sit amet...

7.8 String constants

Lorem ipsum dolor sit amet...

8 Library and run time

Lorem ipsum dolor sit amet...

8.1 I/O syscalls

Lorem ipsum dolor sit amet...

8.2 The memory manager

Lorem ipsum dolor sit amet...

9 Building

Lorem ipsum dolor sit amet...

10 Testing

In order to test the compiler, a test suite has been constructed that may be used to verify results against an already existing result set.

The test suite offers the following functions:

- **newvalids/ackvalids/fullclean** – These commands are used to create a new result set as reference for further tests. While `fullclean` deletes the old set, `newvalids` is used to create a new one. After verifying that the compiled output is correct (by manually checking it), the command `ackvalids` can be used to acknowledge the set (resulting in a checksum file).
- **test/clean** – `test` is used to perform a compilation and compare the results against the last valid result set. In order to do so, checksums of the tests are compared. If they are not equal, a `diff` is printed to the user.

Bibliography

[Goo10] Google Inc. The Go Programming Language Specification. `http://golang.org/doc/go_spec.html` (4.6.2010), 2010.

[Pik00] Pike, R. A Manual for the Plan 9 assembler. `http://doc.cat-v.org/plan_9/4th_edition/papers/asm` (4.6.2010), 2000.

[Wir96] Wirth, N. *Compiler Construction*. Addison-Wesley, 1996.