Go-- Programming Language Proposal

Chen Chen, Arya Lingyu Zhao, Yang Li, Yuyan Ke
{cc4351, lz2650, yl4111, yk2822}@columbia.edu

January 31, 2021

1 Introduction

Go-- is an imperative, statically typed language with C-like syntax and support for concurrency. We take inspiration from goroutine and aim to emulate goroutine with gofunc, and we use channels for communications in between routines. We choose to adopt goroutine and channel for concurrency because they have enabled the users to write concurrent programs with fewer lines of code, neater design, and have provided better readability.

Additionally, we would like to hide away the use of pointers and the details of memory management from the users. Time permitting, we also plan to explore the possibility of implementing first-class function.

2 Language Details

2.1 Data Types & Operations

This language has primitive data types including int, bool, float, char and string. Unlike C, this language doesn't have pointers. The declaration of array type is of the format dataType arrayName[arraySize]. For example, int [].

Data Type	Description	Operations	Examples

`int`	At least 2 bytes, usually 4 bytes	==, < , >, !=, +, -, *, /, %, =<, >=, +=, -=, ++, --	`6 ++; # 7`
			`6 / 3; # 2`
			`7 % 3; # 1`
			`6 < 7; # false`

`float`	4 bytes	==, < , >, !=, +, -, *, /, %, =<, >=, +=, -=, ++, --	`5.22 + 2.0; # 7.22` `4.3 < 5.6; # true`

`bool`	1 bit	==, !, !=, &&, ||	`x = true;`
			`!x; # false`
			`1 == 3; # false`

`char`	1 byte	=, ==, !=, +, ++, –, <, >, =<, >=	`char test = 'h';` `'A' < 'B'; # true`

`string`	Size varies. An immutable array of chars	=, ==, !=, <, >, =<, >= (lexicographical), + (concatenate)	`x = "hi";`
			`y = "boye";`
			`x + y; # returns "hiboye"`

`func`	Standard function type: runs in the same thread	N/A	`gofunc int foo(int x, int y){`
			`  int ret = x + y ;`
			`  return ret;`
			`}`

`gofunc`	Concurrent function type: runs in a new thread	N/A	`func int foo(int x, int y){`
			`  int ret = x + y ;`
			`  return ret;`
			`}`

`channel`	Shared memory for all threads	<-	`channel example= new_channel(int, 3);`
			
			`example <- 5;`
			`x <-; # x = 5`

`dataType []`	array	indexing []	`int mark[] = {19, 10, 8, 17, 9};`
			
			`mark[0]; #19`

2.2 Keywords

```
char, int, float, string, func, gofunc, void, channel, if, else,
continue, for, while, break, return, const, go, struct
```

2.3 Control Flow

```
if...else…
for
while
break...continue...
```

2.4 Functions

Go-- supports C like functions after the keyword “func” or “gofunc”, such that “func” represents a generic function and “gofuns” defines a function to be executed on a concurrent thread. func main() is an exception given that is the entrant point into the execution of the program and it runs on the main thread. Like C, all function declarations must include the return type and accept the standard data typed defined in Section 2.1 as parameters into the functions.

```
// Standard function: runs of the same thread as the calling thread
func int sum(int x, int y)
{
    return x + y;
}

// Concurrent function: this function will be executed on another thread
gofunc int sum_thread(int x, int y)
{
    return x + y;
}
```

2.5 Comments

Go-- supports the C syntax for comments, including `//` for a single-line comment and `/* … */` or `#if 0 … #endif` for multi-line comment.

```
// This is a single-line comment.

/*
    This is an example of
    a multi-line comment
*/

#if 0
All these codes are also commented.
int x = 10;
char c = 56;
#endif

int x = 5;
```

2.6 Memory

Go-- language supports “pass by value”. Logistics of memory management and pointer manipulation (e.g. malloc(), free(), etc.) are hidden from the users.

3 Language Features

3.1 Static typing

All variables in the language should be assigned with a type with the syntax

```
int x = 2;
```

This enforces the variable x to be type integer, and any declaration or instantiation without an explicit type assigned should not be allowed, for example :

```
y = 2;
```

Is not allowed in the language.

3.2 Functions

In the language, we have built-in function types, with keywords “gofunc”, “func” denoting different types of functions. And because the language is static-typed, the type of function arguments should be explicitly given.

3.2.1 Concurrent Functions

Concurrent Functions are denoted by keywords “gofunc”, which indicates the function can be runned in a concurrent manner using the keyword “go”, the return type should be explicitly given in the function signature, for example:

```
gofunc int foo(int x, int y){
      int ret = x + y ;
      return ret;
}
```

Takes two argument x and y and returns an integer as an result, And :

```
gofunc void foo2(int x, int y){
      return;
}
```

Takes two arguments and returns void. And sample code like:

```
go foo(1,2);
go foo2(1,2);
```

Runs in a concurrent manner. That is, line 2 is executed concurrently with line 1 without waiting line1 finishes its execution and could finish before line 1 finishes. For example if we have function:

```
gofunc void print1(){
      print(“first\n”);
}

gofunc void print2(){
      print(“second\n”)
}
```

And if we have code:

```
go print1();
go print2();
```

The output could either be:

```
first
second
```

or

```
second
first
```

Depending on the underlying scheduling architecture of the operating system.

3.2.2 Normal Functions

Normal Functions are denoted by keywords “func”,the return type should be explicitly given in the function signature, for example:

```
func int foo(int x, int y){
      int ret = x + y ;
      return ret;
}
```

Takes two argument x and y and returns an integer as an result, And :

```
func void foo2(int x, int y){
      return;
}
```

Takes two arguments and returns void. And sample code like:

```
foo(1,2);
foo2(1,2);
```

Runs in a sequential order. That is, line 2 will start execution after function call in line 1 returns.

3.2.3 First class functions

All functions can be treated as variables of built-in types of “gofunc” and “func”and be assigned to variables.In the assignment, the return type of the function should be explicitly given.. For example, we can have

```
gofunc int foo = gofunc int (int x, int y) {return x+y;};
```

And then we can have

```
int i = go foo(1,2);
```

And the value of i will be 3 after the code execution.

3.3 String

String is built-in in our language as a type so that users do not have to deal with the underlying pointers. Just like other primitive types, users can declare a string using syntax:

```
string s = “Hello World”
```

3.4 Channel

Channel is a built-in type that operates like FIFO pipes with fixed memory and can be shared among different concurrent running go functions.

3.4.1 Declaration

Users can declare a channel using the library function `new_channel(type,int size)`. The parameter `type` and `size` specifies the data type and size of the channel being created, For example, a channel that can contain 5 integers can be declared using the following syntax:

```
channel example= new_channel(int, 5);
```

3.4.2 Access elements

Function calls can both enqueue and dequeue the data in the channel use operator <-, For example, if we have a channel declared as in 3.4.1. The code:

```
example <- 5;
int x <- example;
```

will enqueue 5 into the channel after line 1 finishes execution, and code in line 2 will dequeue 5 and assign value 5 to variable x. Channels follow first-in-first-out rules and concurrent access is taken care of by the underlying implementation of the language.

3.5 Standard Library

In the Standard Library we will include functions like `print`,`new_channel`, and string manipulation functions like `strlen` as time allows.

4 Examples

```
// Sample program

gofunc void f(int num)
{
    for (int i = 0; i < 3; i++)
    {
        printf("%d", num);
        message <- “goodbye”; // Insert string into channel
    }
    message <- “final goodbye”;
}

func string ftwo(int num)
{
    message <- “hello”;
    return “hello”;
}
```

```
func int main()
{
    int a = 3;
    // Creating a channel for a maximum of 5 strings
    channel message = new_channel(string, 5);

    // f() is running on a concurrent thread
    go f(a);
    a = 4;

    string str = ftwo(a);      // Execute in main thread

    // Execute an anonymous function
    func void ()
    {
        printf(“anon.function”);
    }();

    // Store an existing or a new function as a variable
    func funcvar = ftwo;
    funcvar(1);
    func func2var = func void()
    {
        while (1){}
    }

    // Read strings from the channel
    for (int i = 0; i < 5; i++)
    {
        printf("%s", <- message);
    }

    return 0;
}
```