Go

Rossmery Loayza Soto, Daniel Salinas Téllez

[a] *Ingeniería Informática*
[b] *Universidad Católica San Pablo*

1. Introducción

Go es un lenguaje de propósito general. Está fuertemente tipado asi como tambien posee recogida de basura, y cuenta con el apoyo explícito de la programación concurrente. Los programas se construyen a partir de paquetes, cuyas propiedades permiten la gestión eficiente de las dependencias. Las implementaciones existentes utilizan un modelo de compilación / enlace tradicional para generar los binarios ejecutables.

Se especifica la sintaxis utilizando Extended Backus-Naur Form (EBNF):

$Production = production_name" = "[Expression]"."$.
$Expression = Alternative"|"Alternative$.
$Alternative = TermTerm$.
$Term = production\_name|token["..."token]|Group|Option|Repetition$.
$Group = "("Expression")"$.
$Option = "["Expression"]"$.
$Repetition = ""Expression""$

2. Elementos léxicos

2.1. Comentarios

Comentarios sirven como documentación del programa. Hay dos formas:

- Comentarios de línea comienzan con la secuencia de caracteres // y se detienen al final de la línea.
- Comentarios generales comienzan con la secuencia de caracteres $/, *$ y terminan con la primera secuencia de caracteres posterior $*, /$.

2.2. Tokens

Los tokens forman el vocabulario del lenguaje Go. Hay cuatro clases:

- Identificadores
- Palabras clave
- Operadores y delimitadores
- Literales

2.3. Punto y Coma

La gramática formal utiliza punto y coma ";" como terminadores en un número de producciones. Los programas en Go pueden omitir la mayoría de estos puntos y comas utilizando estas dos reglas:

Cuando la entrada es separada en tokens, un punto y coma se inserta automáticamente en el flujo de tokens inmediatamente después símbolo definitivo de una línea si ese token es: un identificador un número entero, punto flotante, imaginario, runa, o una cadena literal una de las vacaciones de palabras clave, continúan, fallthrough, o devolución uno de los operadores y delimitadores $++, --,),]$, o } Para permitir que las declaraciones complejas puedan ocupar una sola línea, un punto y coma puede omitirse ante un cierre ")" o "}".

2.4. Identificadores

Los identificadores nombran entidades de programas tales como variables y tipos. Un identificador es una secuencia de una o más letras y dígitos. El primer carácter de un identificador debe ser una letra.

$identifier = letterletter|unicode\_digit.$

2.5. Palabras Clave

Las siguientes palabras clave son reservados y no pueden ser usados como identificadores:

break default func interface select case defer go map struct chan else goto package switch const fallthrough if range type continue for import return var

2.6. Operadores y Delimitadores

Las siguientes secuencias de caracteres representan los operadores, los delimitadores y otros símbolos especiales: **??**

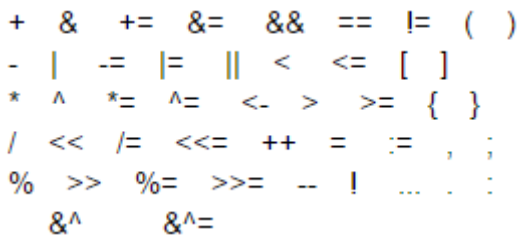

Figure 1: Operadores y Delimitadores

2.7. Literal entero

Un entero literal es una secuencia de dígitos que representan una constante entera. Un prefijo opcional establece una base no decimal: 0 para octal, 0x o 0X para hexadecimal. En literales hexadecimales, letras af y AF representan valores 10 a 15.

$int\_lit = decimal\_lit|octal\_lit|hex\_lit.$
$decimal\_lit = (\text{"}1\text{"}\ldots\text{"}9\text{"})decimal\_digit.$
$octal\_lit = \text{"}0\text{"}octal\_digit.$
$hex\_lit = \text{"}0\text{"}(\text{"}x\text{"}|\text{"}X\text{"})hex\_digithex\_digit.$

2.8. Literal de punto flotante

Un literal de coma flotante es una representación decimal de una constante de coma flotante. Cuenta con una parte entera, un punto decimal, una parte fraccionaria, y una parte exponente. El entero y parte fraccionaria comprenden dígitos decimales; la parte exponente es un e o E seguida de un exponente decimal opcionalmente firmado. La parte entera o la parte fraccionaria pueden ser eludidas.

$float\_lit = decimals\text{"}.\text{"}[decimals][exponent]|$
$decimalsexponent|\text{"}.\text{"}decimals[exponent].$
$decimals = decimal\_digitdecimal\_digit.$
$exponent = (\text{"}e\text{"}|\text{"}E\text{"})[\text{"}+\text{"}|\text{"}-\text{"}]decimals.$

2.9. Literal imaginario

Un literal imaginaria es una representación decimal de la parte imaginaria de una constante compleja. Consiste en un literal entero o decimal de coma flotante seguido de la letra minúscula i.

$imaginary\_lit = (decimals|float\_lit)\text{"}i\text{"}.$

2.10. Literal de cadena

Un literal de cadena representa una cadena obtenida de la concatenación de una secuencia de caracteres. Hay dos formas:

- **literales de cadena en bruto:** son secuencias de caracteres entre comillas simples, como en "foo".

- **literales de cadena interpretados:** son secuencias de caracteres entre comillas dobles, como en "bar". Dentro de las comillas, cualquier personaje puede aparecer a excepción de nueva línea y comillas dobles.

$string\_lit = raw\_string\_lit | interpreted\_string\_lit.$
$raw_string\_lit = "'" unicode\_char | newline "'".$
$interpreted\_string\_lit = '"' unicode\_value | byte\_value '"'.$

3. Constantes

Hay constantes booleanas, constantes enteras, constantes de punto flotante, constantes complejas, y las constantes de cadena. Runa, número entero, punto flotante, y las constantes complejas se denominan colectivamente constantes numéricas.

Un valor constante está representado por una runa, número entero, de punto flotante, imaginario, o una cadena literal, un identificador que denota una constante, una expresión constante, una conversión con un resultado que es una constante, o el valor resultado de algún incorporado funciones como unsafe. Sizeof aplican a cualquier valor, cap o len aplican a algunas expresiones, real e imag aplican a una constante compleja y complex aplicada a constantes numéricas. Los valores booleanos están representados por las constantes true y false.

Las constantes numéricas representan los valores de precisión arbitraria y no hacen desbordamiento.
Las constantes pueden ser con o sin tipo. Las constantes literales, verdadero, falso, iota, y ciertas expresiones constantes son sin tipo, solo tienen operandos constantes sin tipo.

A una constante se le puede dar un tipo de forma explícita mediante una declaración o conversión constante, o implícitamente cuando se utiliza en una declaración de variable o una asignación o como operando en una expresión. Es un error si el valor constante no puede ser representado como un valor del tipo respectivo.

Restricción de implementación: A pesar de las constantes numéricas tienen precisión arbitraria en el lenguaje, un compilador puede implementarla usando una representación interna con una precisión limitada. Dicho esto, cada implementación debe:

- Representar constantes enteras con al menos 256 bits.

- Representan constantes de punto flotante, incluyendo las partes de una constante compleja, con una mantisa de al menos 256 bits y un exponente firmada de al menos 32 bits.

- Dar un error si no puede representar una constante entera con precisión.
- Dar un error si no puede representar un punto flotante o constante compleja debido al desbordamiento.
- Estos requisitos se aplican tanto a las constantes literales y al resultado de la evaluación de expresiones constantes.

4. Variables

Una variable es una ubicación de almacenamiento para mantener un valor. El conjunto de valores permisibles está determinada por el tipo de la variable.

Una declaración de variable o, para los parámetros de la función y los resultados, la firma de una declaración de función o función literal reserva almacenamiento para una variable nombrada. Llamar a la función incorporada de nuevo o tomar la dirección de un material compuesto de almacenamiento asigna literal para una variable en tiempo de ejecución. Dicha variable anónima se conoce a través de una (posiblemente implícita) dirección del puntero.

Variables estructuradas de matriz, slice, y los tipos struct tienen elementos y campos que pueden ser tratados de forma individual. Cada uno de estos elementos actúa como una variable.
El tipo estático (o simplemente el tipo) de una variable es el tipo dado en su declaración, el tipo previsto en la nueva llamada o compuesto literal, o el tipo de un elemento de una variable estructurada. Variables de tipo interfaz también tienen un tipo dinámico distinto, que es el tipo concreto del valor asignado a la variable en tiempo de ejecución (a menos que el valor es el identificador nil declarar con anterioridad, que no tiene ningún tipo). El tipo dinámico puede variar durante la ejecución, pero los valores almacenados en las variables de interfaz son siempre imputables al tipo estático de la variable.

Valor de una variable se recupera haciendo referencia a la variable en una expresión; que es el valor más reciente asignado a la variable. Si una variable no se ha asignado un valor, su valor es el valor cero para su tipo.

5. Tipos de Datos

Un tipo determina el conjunto de valores y operaciones específicas a valores de ese tipo. Los tipos pueden ser nombrados o sin nombre. Tipos de datos con nombre se especifican mediante el nombre del tipo; los tipos de datos sin nombre se especifican usando un tipo literal, que compone un nuevo tipo de los tipos existentes.

$Type = TypeName|TypeLit|"("Type")".$
$TypeName = identifier|QualifiedIdent.$
$TypeLit = ArrayType|StructType|PointerType|FunctionType|InterfaceType|$
$SliceType|MapType|ChannelType.$

Las instancias con nombre de los booleanos, numéricos y tipos de cadena se declarar con anterioridad. -Tipos pueden ser construidos tipos-array compuestas, struct, puntero, la función, la interfaz, rebanada, el mapa y el uso de los canales de los literales de tipo.

5.1. Booleano

Un tipo booleano representa el conjunto de valores denotados por las constantes declaradas con anterioridad verdaderos y falsos. El tipo es bool.

5.2. Numérico

Un tipo numérico representa conjuntos de valores enteros o de punto flotante. Los tipos numéricos declarar con anterioridad independientes de la arquitectura son:

- uint8 el conjunto de todos los enteros sin signo de 8 bits (0 a 255)
- uint16 el conjunto de todos los enteros sin signo de 16 bits (0 a 65535)
- Uint32 el conjunto de todos los enteros sin signo de 32 bits (0 a 4294967295)
- uint64 el conjunto de todos los enteros sin signo de 64 bits (0-18446744073709551615)
- int8 el conjunto de todos los enteros firmó 8 bits (-128 a 127)
- Int16 el conjunto de todas firmó enteros de 16 bits (-32768 a 32767)
- int32 el conjunto de todas firmó enteros de 32 bits (-2147483648 a 2147483647)
- int64 el conjunto de todas firmó enteros de 64 bits (-9223372036854775808 a 9223372036854775807)
- float32 el conjunto de todos los 754 IEEE-números de punto flotante de 32 bits
- float64 el conjunto de todos los 754 IEEE-números de punto flotante de 64 bits
- complex64 el conjunto de todos los números complejos con Float32 partes real e imaginaria
- complex128 el conjunto de todos los números complejos con float64 partes real e imaginaria

5.3. String

Un tipo string representa el conjunto de valores de cadena. Un valor de cadena es una (posiblemente vacía) secuencia de bytes. Las cadenas son inmutables: una vez creado, no es posible cambiar el contenido de una cadena.

5.4. Array

Una matriz es una secuencia numerada de elementos de un solo tipo, llamado el tipo de elemento.
$ArrayType = "[" ArrayLength "]" ElementType.$
$ArrayLength = Expression.$
$ElementType = Type.$

5.5. Slice

Una slice es un descriptor para un segmento contiguo de una matriz subyacente y proporciona acceso a una secuencia numerada de elementos de la matriz. Un tipo de sector denota el conjunto de todos los cortes de las matrices de su tipo de elemento. El valor de una rebanada sin inicializar es nula.

$SliceType = "[""]" ElementType.$

Al igual que las matrices, las rebanadas son indexables y tienen una longitud. La longitud de una rebanada 'S' puede ser descubierto por la función 'len' incorporado; a diferencia de las matrices se puede cambiar durante la ejecución. Los elementos pueden ser abordadas por índices enteros 0 a len (s) -1. El índice de rebanada de un elemento dado puede ser menor que el índice del mismo elemento de la matriz subyacente.

5.6. Struct

Una estructura es una secuencia de elementos con nombre, llamados campos, cada uno de los cuales tiene un nombre y un tipo. Los nombres de campo se pueden especificar de forma explícita (IdentifierList) o implícitamente (AnonymousField). Dentro de una estructura, nombres de los campos no están en blanco deben ser únicos.
Ejemplo:

```
struct {
x, y int
u float32
_ float32 // padding
A *[]int
F func()
}
```

5.7. Puntero

Un tipo de puntero denota el conjunto de todos los punteros a las variables de un tipo determinado, llamado el tipo base del puntero. El valor de un puntero no inicializado es nula.

5.8. Función

Un tipo de función denota el conjunto de todas las funciones con los mismos tipos de parámetros y de resultados. El valor de una variable no inicializada del tipo de función es nula.

$FunctionType = Firma"Func".$
$Firma = Parámetros[Resultados].$
$Resultado = Parámetros|Escribe.$
$Parámetros = "("[ParameterList[","]]")".$
$ParameterList = ParameterDecl","ParameterDecl.$
$ParameterDecl = [IdentifierList]["..."]Tipo.$

Dentro de una lista de parámetros o resultados, los nombres (IdentifierList) deben estar presentes ya sea todo o todos estén ausentes. Si está presente, cada nombre representa un elemento (parámetro o resultado). En su defecto, cada tipo representa un elemento de ese tipo.

5.9. Interface

Un tipo de interfaz especifica un conjunto método llamado de su interfaz. Una variable de tipo interfaz puede almacenar un valor de cualquier tipo con un conjunto método que es cualquier superconjunto de la interfaz. Tal tipo se dice es el que implementa la interfaz. El valor de una variable no inicializada del tipo de interfaz es nula.

$InterfaceType = "interfaz""""MethodSpec";""".$
$MethodSpec = MethodNameFirma|InterfaceTypeName.$
$MethodName = identificador.$
$InterfaceTypeName = TypeName.$

Al igual que con todos los conjuntos de método, en un tipo de interfaz, cada método debe tener un nombre no esté en blanco único.

interfaz
Leer (b Buffer) bool
Escriba (b Buffer) bool
Cerca()

5.10. Map

Un map es un grupo desordenado de los elementos de un tipo, llamado el tipo de elemento, indexada por un conjunto de claves únicas de otro tipo, llamado el tipo de clave. El valor de un mapa sin inicializar es nula.

$MapType = "mapa""["KeyType"]"ElementType.$
$KeyType = Tipo.$

Los operadores de comparación == y = debe ser totalmente definido para operandos del tipo llave!; por tanto, el tipo de clave no debe ser una función, map, o slice. Si el tipo de clave es un tipo de interfaz, estos operadores de comparación deben ser definidos por los valores claves dinámicos; de lo contrario se producirá un error en tiempo de ejecución.

5.11. Canal

Un canal proporciona un mecanismo para la ejecución de funciones al mismo tiempo para comunicarse enviando y recibiendo valores de un tipo de elemento especificado. El valor de un canal sin inicializar es nula.
Este es el método que el lenguaje utiliza para realizar concurrencia dentro de sus programas.
$ChannelType = ("chan"|"chan""< -"|"< -""chan")ElementType.$

El operador '< −' especifica la dirección del canal, enviar o recibir. Si no se da la dirección, el canal es bidireccional. Un canal puede estar limitado sólo para enviar o sólo para recibir por la conversión o la sesión.

6. Ámbitos y tiempos de vida

La forma en cómo el lenguaje go utiliza el ámbito de es forma léxica. Go tiene:

- Un ámbito para identificadores pre declarados (universe block)
- Un ámbito para un identificador denotando una constante, tipo, variable o función declarada en un nivel superior (fuera de alguna función) (package block)
- Un ámbito para la importación de paquetes o archivo que contenga la importación de estos
- Un ámbito para un identificador denotando un método, parámetro de función o variable resultado
- Un ámbito para un identificador de variable o constante declarada dentro el inicio de una función
- Un ámbito para un identificador de tipo declarado dentro del inicio de una función.

7. Expresiones Aritméticas

Las expresiones aritméticas aplican a valores numéricos y dan un resultado del mismo tipo que el primero operando.
Los operadores principales $(+, -, *, /)$ se aplican a enteros, flotantes, tipos complejos. "+" se aplica a strings. y los operadores lógicos de un bit aplica a variables enteras únicamente.

+	suma	enteros, flotantes, valores complejos, strings
-	diferencia	enteros, flotantes, valores complejos, strings
*	producto	enteros, flotantes, valores complejos, strings
/	cociente	enteros, flotantes, valores complejos, strings
%	residuo	enteros
&	AND	enteros
|	OR	enteros
^	XOR	enteros
&^	(AND NOT)	enteros
<<	desplazamiento izquierda	entero << unsigned entero
>>	desplazamiento derecha	entero >> unsigned entero

Figure 2: Operadores y Delimitadores

8. Sobrecarga de Operadores

El lenguaje Go no utiliza la sobrecarga de operadores.
Según el creador del lenguaje, la experiencia con otros lenguajes dice que habiendo una variedad de métodos con el mismo nombre pero diferentes formas, fue ocasionalmente útil pero que esto podría ser también confuso y frágil en práctica.

9. Programa de Prueba

Para comprobar cada punto realizado en este documento, se realizó la implementación de un algoritmo robabilístico.
Este algoritmo se realizo en base a sintáxis y mejores practicas en la implementación de este lenguaje.

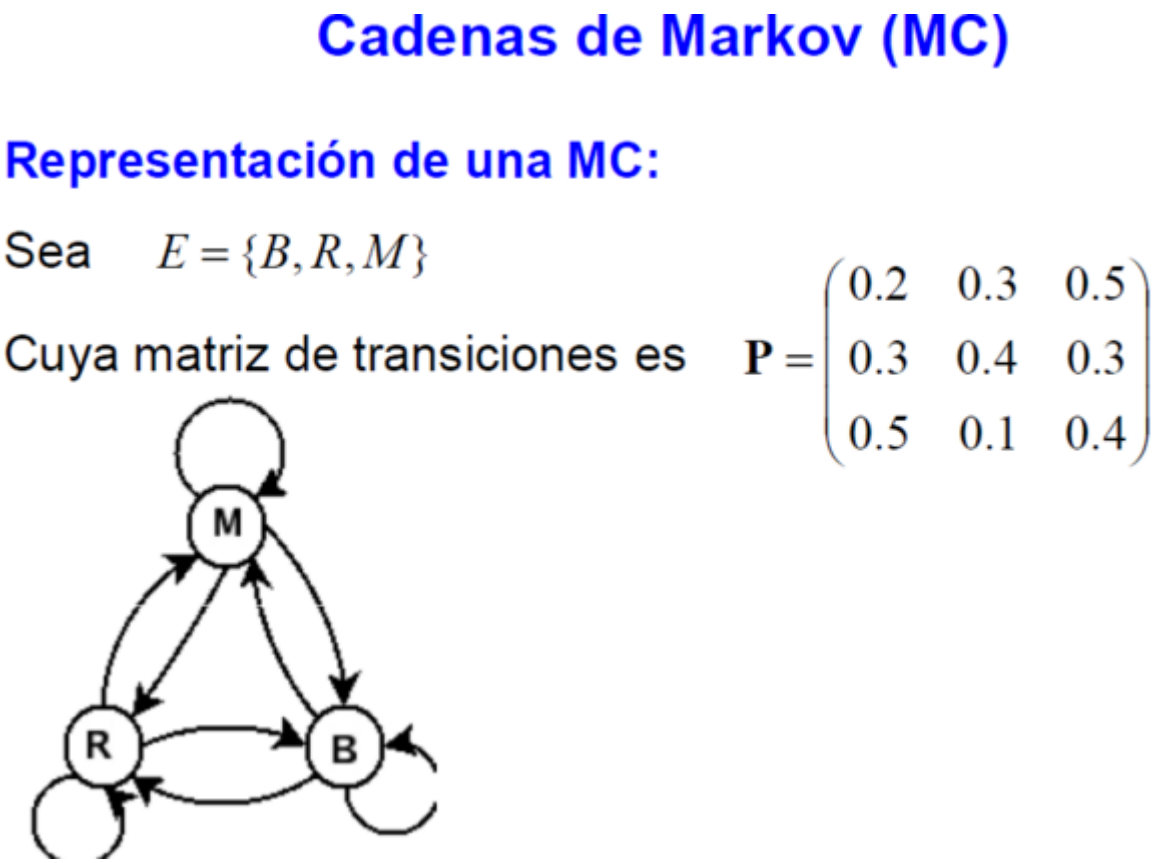

Figure 3: Operadores y Delimitadores

en la implementación del algoritmo se utilizaron los puntos tocados dentro del documento.
dentro del algoritmo que está anexado al documento hay 2 tipos de prueba del algoritmo probabilístico:

- probar el algoritmo con un número pequeño de estados o transiciones de estado
- probar el algoritmo con un número mayor de estados o transiciones aleatorias. En esta parte fue donde se implementó la función utilizando la concurrencia del lenguaje.

References

- David Chisnal, The GO Programming Language 2012
- The Go Programming Language Specification, 2015, golang.org/ref/spec
- Frequently Asked Questions (FAQ), golang.org/doc/faq