УДК Lorem ipsum dolor sit amet

Операционная семантика выражений в языке Go на языке ABML

Харьков А.А. (Студент ММФ НГУ)

Ануреев И.С. (Институт систем информатики СО РАН)

Lorem ipsum dolor sit amet

1. Введение

Lorem ipsum dolor sit amet

2. Модель объектов языка Go

Прежде чем разбираться с моделированием действий различных программ или их составляющих, необходиомо предварительно создать модель для элементов, составляющих язык Go, которые явно присутсвуют в коде программы: переменные, функции, операторы, декларации и т.д. Далее, используя язык ABML, будет описано создание моделей соответсвующих объектов языка Go.

Классификация и порядок изложения опираются на официальную спецификацию языка Go. Большая часть названий объектов, моделирующихся в этом разделе, будет совпадать с их названиями в официальной спецификации для удобства восприятия и облегчения поиска дополнительной информации по каждому из них. Однако некоторые сущности были признаны излишними с точки зрения моделирования их функицональности, а некоторые имена недостаточно отражающими суть описываемого объекта. Это вызвало некоторую разницу в именовании нескольких объектов в построенной модели.

2.1. Лексические элементы

Ниже представлено создание моделей для таких объектов, как комментарий `"comment"` - класс объектов, включающий в себя одиночные комментарии `"line comment"` и многострочные комментарии `"general comment"`. А так же идентификатор `"identifier"`, моделирующий имя объекта программы, например переменной, функции и т.д.

```
(typedef "comment" (uniont "line comment" "general comment"))
(mot "line comment" :at "text" string)  ; // single-line
(mot "general comment" :at "text" string)  ; /* multi-line */
(typedef "identifier" (uniont string))
```

2.2. Константы

Данный подраздел описывает модели объектов, представляющих постоянные значения, допускающие взаимодейтсвие с программой: логические константы и числа, в том числе комплексные.

```
(typedef "constant" (uniont "bool constant" "numeric constant"
   string))
(typedef "bool constant" (enumt "true" "false"))
(typedef "numeric constant" (uniont int real "complex constant"))
(mot "complex constant" :at "re" real :at "im" real)
```

2.3. Типы данных

В данном подразделе моделируются типы данных, большая часть которых должна быть интуитивно понятна из названий. Типы можно разделить на базовые - представляющие некоторые числа, символы или строки, и составные типы данных - состоящие зачастую совокупность базовых или моделирующих более сложные концепции хранения данных.

Сперва рассмотрим базовые типы данных.

```
(typedef "type" (uniont "base type" "composite type"))
(typedef "base type" (uniont "boolean type" "numeric type"
   "string type" "rune type"))
(typedef "boolean type" (enumt "bool"))
(typedef "numeric type" (uniont "real-valued type" "complex
   type" "byte type"))
(typedef "real-valued type" (enumt "int type" "float type"))
(typedef "int type" (uniont "signed int type" "unsigned int
   type"))
```

```
(typedef "signed int type" (enumt "int" "int8" "int16" "int32"
  "int64"))
(typedef "unsigned int type" (enumt "uint" "uint8" "uint16" "
  uint32" "uint64" "uintptr"))
(typedef "float type" (enumt "float32" "float64"))
(typedef "complex type" (enumt "complex32" "complex64"))
(typedef "byte type" (enumt "byte"))
(typedef "rune type" (enumt "rune"))      ; 'a' | '\t' | '\377'
(typedef "string type" (enumt "string"))
```

Теперь несколько слов про каждый из типов выше. "boolean type" - моделирует логический тип данных, "real-valued type" - вещественные числа, "complex type" - комплексные числа, "byte type" - двоичное представление чисел, "rune type" состоит из символов: буква, цифра, специальный символ и т.д., "string type" включает в себя любые строки.

Далее перейдём к рассмотрению составных типов данных.

```
(typedef "composite type" (uniont "array type" "slice type"
  "struct type" "pointer type" "function type" "interface type"
  "map type" "channel type"))
```

Начнём с простейших из них.

```
(mot "array type" :at "len" nat :at "element type" "type")
(mot "slice type" :at "element type" "type")
(mot "struct type" :at "fields" (listt "field decl"))
(typedef "field decl" (uniont "named field" "embedded field"))
(mot "named field" :at "name" "identifier" :at "type" "type")
(mot "embedded field" :at "type" "type")
(mot "pointer type" :at "type" "type")
```

Тип список "array type" - структура данных, которую можно описать как упорядоченный набор элементов одного типа.

Тип срез "slice type" - ссылка на последовательную часть исходного массива.

Тип структура "struct type" позволяет создавать новые объекты, состоит из множе-

ства полей. Поля структуры могут быть именованными "named field" или встроенными "embedded field".

Именованные поля представляют собой пару имя-значение, устроенные аналогично переменным. Встроенные поля обычно являются другими структурами, таким образом позволяя встраивать все поля вложенной структуры в текущую.

Тип указатель "pointer type" - ссылка на исходный объект.

Далее речь пойдёт о функциях.

```
(mot "function type" :at "signature" "signature")
(mot "signature" :at "parameters" (listt "parameter decl") :at
   "result" "function result")
(mot "parameter decl" :at "names" (listt "identifier") :at
   "type" (uniont "type" "variadic type"))
(mot "variadic type" :at "type" "type")  ; func f(numbers
   ...int) -> f(1, 2, 3, 4)
(typedef "function result" (uniont (listt "parameter decl")
   "type"))
```

Тип функция "function type" характеризует своей сигнатурой "signature", которая в свою очередь является совокупностью параметров "parameter decl" и типом результата функции "function result".

В следующем блоке определяется интерфейс и связанные с ним объекты.

```
(mot "interface type" :at "elements" (listt "interface elem"))
(typedef "interface elem" (uniont "method elem" "type elem"))
(mot "method elem" :at "name" "identifier" :at "signature"
   "signature")
(typedef "type elem" (uniont (listt "type term")))
(typedef "type term" (uniont "type" "underlying type"))
; type A interface {
;   f(n int) (q bool)     // method elem: "name" = ‘f‘,
; }                       // "signature" = ‘(n int) (q bool)‘
```

Тип интерфейс "interface type" состоит из элементов-методов и элементов-типов.

Тип элементы-методы "method elem" представляют собой пару из имени функции и её

сигнатуры.

Тип элементы-типы `"type elem"` обычно являются описанными ранее интерфейсами, что позволяет включить в элементы текущего интерфейса все элементы описанного ранее.

Осталось определить ещё несколько типов данных.

```
1 (mot "underlying type" :at "type" "type")  ; ~Type
2 (mot "map type" :at "key type" "type" :at "element type" "type")
3 (mot "channel type" :at "direction" "direction" :at "type"
    "type")
4 (typedef "direction" (enumt "bidirectional" "send" "receive"))
```

Объемлющий тип `"underlying type"` определяется следующим образом: если это логический, числовой или сточный тип, то объемлющим для него является он сам; иначе, за некоторым исключением, объемлющим для него является тип, на который он ссылается в своём объявлении.

Тип словарь `"map type"` является реализацией хэш-таблицы.

Тип канал `"channel type"` используется преимущественно в многопоточных процессах. Каналы можно представить как переменную с дополнительной особенность - при считывании значения "пустой"ячейки происходит блокирование процесса до тех пор, пока другой процесс не положит в данную ячейку какое-либо значение. Причём каждый канал имеет своё направление `"direction"` - некоторые каналы предназначены только для записи или только для получения данных из него, или всё вместе.