Лекция 2. Язык программирования Рефал-5

Коновалов А.В.

4 марта 2023 г.

О Рефале (1)

РЕФАЛ (**РЕ**курсивный **Ф**ункциональный **АЛ**горитмический язык, REFAL — **RE**cursive **F**unctional **A**lgorithmic **L**anguage) — семейство функциональных языков программирования, характеризующееся следующими общими чертами:

- данные представлены цепочками (*объектными выражениями*), составленными из символов и скобок, причём скобки образуют правильную структуру;
- конкатенация объектных выражений является фундаментальной операцией;
- анализ значений выполняется путём сопоставления объектного выражения с образцом;
- построение значений выполняется путём интерполяции переменных в выражении с переменными и вызовами функций.

О Рефале (2)

В семейство входит несколько языков, синтаксически несовместимых между собой. На сайте http://refal.ru описаны реализации Рефал-2, Рефал-5, Рефал-6, Рефал Плюс. Можно встретить публикации, в которых описываются разрабатываемые на данный момент реализации Рефала (автор этих строк встречал как минимум 3), тоже несовместимые между собой.

В 2006 году Скоробогатовым С.Ю. был предложен диалект Refal-7 с функциями высшего порядка.

С 2016 года на кафедре ИУ9 разрабатывается диалект Рефал-5λ с функциями высшего порядка, являющийся расширением Рефала-5 (любая программа на Рефале-5 является допустимой программой на Рефале-5λ).

О Рефале (3)

В нашем курсе будет использоваться диалект Рефал-5 (хотя использовать Рефал-5λ не возбраняется ☺). Актуальная документация по этому диалекту доступна только на английском языке по ссылке

http://www.botik.ru/pub/local/scp/refal5/

(На сайте http://refal.ru можно найти перевод на русский язык, однако он описывает устаревший синтаксис, не читайте его.)

Синтаксис Рефала-5

Данные Рефала-5 (1)

Рефал-5 — динамически типизированный язык. Данные представлены **объектными выражениями** — цепочками, составленными из символов и круглых скобок, при этом круглые (их называют **структурные**) скобки должны быть сбалансированы.

Символы бывают трёх видов:

- **символы-литеры** представляют собой запись ASCII-символов, записываются в одинарных кавычках: `'a'`, `'%'`, `'+'`, `'\n'`, `'\x1B'`;
- **символы-числа** или **макроцифры** записываются как десятичные беззнаковые целые числа от `0` до `4294967295` ($2^{32} - 1$);
- **составные символы (*compound symbols*)** или **символы-слова** записываются как последовательности знаков в двойных кавычках: `"Word"`, `"\n\n\n"`, `"+"`, `"<<"`.

Данные Рефала-5 (2)

Несколько символов-литер можно записывать слитно: запись `'hello'` эквивалентна записи `'h' 'e' 'l' 'l' 'o'`.

Если символ-слово является корректной записью идентификатора (имени функции) в Рефале-5 — начинается на букву, состоит из букв, цифр, знаков - и _ — то его можно записывать без двойных кавычек. Запись `Scan-ForName` эквивалента `"Scan-ForName"`.

Заметим, что запись `'abcd'` — это объектное выражение, состоящее из четырёх литер (эквивалентно `'a' 'b' 'c' 'd'`), а запись `"abcd"` — это единственный символ-слово (его можно записать без кавычек: `abcd`).

Выражение, записанное в круглых скобках, называется **скобочным термом**.

Объектный терм (далее, просто **терм**) — это либо символ, либо скобочный терм.

Данные Рефала-5 (3)

Примеры объектных выражений:

```
'abc' 2 3 (4 x 5 y (((6)))) 7 8

(2 "*" (Count "+" 7))

(name 'Alexander') (surname 'Konovalov')

(function gcd (x y)
  (var (rem 1))
  (while ((L y) "<>" 0)
    (rem "=" ((L x) "%" (L y)))
    (x "=" (L y))
    (y "=" (L rem))
  )
  (return (L y))
)
```

Данные Рефала-5 (4)

Данные Рефала-5 напоминают s-выражения языка LISP. Можно даже построить следующую табличку, сопоставляющую понятия Рефала-5 и понятия Scheme:

Рефал-5	Scheme
объектное выражение	список
скобочный терм	вложенный список
литера `'a'`	литера `#\a`
макроцифра	целое число
составной символ	строка `"Hello!"`
составной символ	атом `'hello`

Отличие между данными Лиспа и Рефала — в операциях, применимых к ним. Списки Лиспа однонаправленные (их можно разбирать и наращивать только слева), объектные выражения Рефала можно разбирать с обеих сторон и конкатенировать.

Синтаксис программ на Рефале-5 (1)

Программа на Рефале-5 состоит из определений функций.

Определение функции записывается следующим образом:

```
ИмяФункции {
  предложение;
  предложение;
   ...
}
```

Имя функции должно быть корректным идентификатором Рефала (начинаться с латинской буквы, состоять из латинских букв, цифр и знаков _ и -).

Если функция должна быть доступна из других модулей, перед её именем ставится ключевое слово `$ENTRY`.

Предложений может быть 1 и более (в Рефале-5λ — 0 и более). После последнего предложения точку с запятой можно не ставить.

Синтаксис программ на Рефале-5 (2)

Предложение в простейшем случае имеет вид

```
образец = результат;
```

где `образец` — выражение Рефала, которое может содержать переменные (т.н. **образцовое выражение**), а `результат` — выражение Рефала, которое может содержать и переменные, и вызовы функций (т.н. **результатное выражение**).

Синтаксис программ на Рефале-5 (3)

Переменные в Рефале-5 записываются как `‹вид›.‹индекс›`, где `‹вид›` — одна из трёх букв `s`, `t`, `e`, `‹индекс›` — идентификатор или целое число.

Вид переменной определяет множество значений, на которые переменная может заменяться:

- **s-переменные** могут заменяться на произвольный символ,
- **t-переменные** могут заменяться на произвольный терм (символ или выражение в скобках),
- **e-переменные** могут заменяться на произвольное выражение.

Синтаксис программ на Рефале-5 (4)

Функции принимают ровно один аргумент, он и сопоставляется с образцом.

Вызов функции записывается в угловых скобках:

```
<ИмяФункции аргумент>
```

где `аргумент` — результатное выражение.

Угловые скобки также называются **скобками вызова**, **скобками активации** или **скобками конкретизации**.

Образцы в Рефале-5 (1)

Образцовое выражение (или **образец**) состоит из символов, структурных (круглых) скобок и переменных.

Образец описывает множество объектных выражений, которые можно получить, подставив вместо переменных какие-либо значения соответствующего вида.

Образцы могут содержать несколько вхождений одной и той же переменной (одного и того же вида и с одним и тем же индексом). Такие переменные называются **повторными**. Все вхождения повторной переменной должны иметь одинаковые значения.

Дальше пошли импровизированные слайды…

Образцы в Рефале-5 (2)

Примеры образцов:

- `s.1 s.2 s.3` — выражение любых трёх символов: `1 2 3`, `'abc'`, `1 "+" 2`, `1 '+' 2`, `One Two Three`.
- `s.A s.A s.A` — выражение из трёх одинаковых символов: `'aaa'`, `1 1 1`, `hello hello hello`.
- `t.A t.B t.A` — выражение из трёх термов, первый и последний должны быть одинаковыми: `'aba'`, `( ) (a) ( )`, `(1 2) 3 (1 2)`.
- `e.X '+' e.Y` — выражение, которое содержит знак `'+'` на верхнем уровне, например: `'+'`, `'2+3=5'`, `( ) '-' ( ) '+' ( )`, `'+++'`.
- `(e.X) e.Y (e.Z)` — выражение как минимум из двух термов, первый и последний — скобочные: `(a b) c d (e f)`.

Образцы в Рефале-5 (3)

Сопоставление с образцом $E : P$ — поиск таких значений переменных, подстановка которых в образец P даёт объектное выражение E.

Пример.

```
'hello' : e.X s.R s.R e.Y
```

При сопоставлении переменная `e.X` получит значение `'he'`, `s.R` — `'l'`, `e.Y` — `'o'`.

Образцы в Рефале-5 (4)

Сопоставление с образцом в Рефале *неоднозначно*. Это означает, что для некоторых сопоставлений $E : P$ можно найти несколько различных подстановок переменных, таких что P превращается в E. Например:

```
'abracadabra' : e.X s.R1 s.R2 e.Y s.R1 s.R2 e.Z
```

Возможные подстановки:

```
e.X -> пусто, s.R1 -> 'a', s.R2 -> 'b', e.Y -> 'racad', e.Z -> 'ra'
e.X -> 'a', s.R1 -> 'b', s.R2 -> 'r', e.Y -> 'acada', e.X -> 'a'
e.X -> 'ab', s.R1 -> 'r', s.R2 -> 'a', e.Y -> 'cadab', e.Z -> пусто
```

Образцы в Рефале-5 (5)

Неоднозначность разрешается по следующему правилу: выбирается подстановка с кратчайшим (в количестве термов) значением самой левой e-переменной. Если это не разрешает неоднозначности, то выбирается следующая e-переменная и так далее.

Т.е. в предыдущем примере будет выбрана подстановка, где `e.X` пустая.

Ещё пример:

```
1 2 3 2 4 3 2 : e.X s.R e.Y s.R e.Z
```

Переменные получат следующие значения: `e.X` — `1`, `s.R` — `2`, `e.Y` — `3`, `e.Z` — `4 3 2`.

Образцы в Рефале-5 (6)

e-переменные, которые участвуют в разрешении неоднозначности, называются **открытыми e-переменными**. Например, в образце `e.X s.R e.Y s.R e.Z` открытыми переменными будут `e.X` и `e.Y`.

В образце `(e.X) e.X A e.Y` открытых переменных нет, т.к. образец описывает множество выражений, начинающихся со скобочного терма, сопоставление скобочного терма с `(e.X)` выполняется однозначно — однозначно определяется значение переменной `e.X`. Второе её вхождение повторное и оно тоже определяется однозначно.

В образце `e.A '*' e.A` первое вхождение `e.A` открытое, второе — повторное.

В образце `(e.X s.R e.Y) (e.U s.R e.V)` открытые только `e.X` и `e.U`.

Образцы в Рефале-5 (7)

Открытые переменные переменные в образцах неявно компилируются в циклы при сопоставлении с образцом. Таким образом, образец с одной открытой переменной имеет линейную сложность сопоставления, с двумя — квадратичную, с тремя — кубическую и т.д.

Кроме того, сопоставления с повторными e- и t-переменными также требуют сравнения на равенство соответствующих фрагментов сопоставимого выражения, что тоже требует времени.

Таким образом, сопоставление с образцом в Рефале может выполняться не за $O(1)$, а зависеть по времени от размера входных данных.

Программы в Рефале-5 (1)

Выполнение программы на Рефале-5 начинается с вызова функции `Go` с пустым аргументном (аналогично функции `main()` языка Си). Т.к. эта функция должна вызываться извне модуля, перед её именем должно быть записано ключевое слово `$ENTRY`.

Встроенная функция, распечатывающая объектное выражение на экране, называется `Prout` (от print out).

Таким образом, кратчайшая программа на Рефале имеет вид

```
$ENTRY Go {
  /* пусто */ = <Prout 'Hello, World!'>
}
```

Программы в Рефале-5 (2)

```
$ENTRY Go {
  /* пусто */ = <Prout 'Hello, World!'>
}
```

Здесь на месте пустого образца написан *комментарий.* Комментарии в Рефале-5 записываются как в языке Си: `/* ... */`. Также допустимы однострочные комментарии — строка текста целиком, начинающаяся со `*`.

Программы в Рефале-5 (3)

Программы записываются в текстовых файлах с расширением `.ref`. Для компиляции программы нужно ввести в командной строке:

```
$ refc hello.ref
```

В результате получится файл байткода (т.н. RASL) с расширением `.rsl`. Его можно запустить интерпретатором байткода

```
$ refgo hello.rsl
```

Программы в Рефале-5 (4)

Для ввода строки текста из `stdin` используется встроенная функция `Card`, вызываемая без аргументов (название восходит к тем временам, когда компьютеры читали перфокарты, функция `Card` считывала очередную перфокарту).

Напишем программу, которая приветствует пользователя:

```
$ENTRY Go {
  = <Prout 'Как тебя зовут?'>
    <Prout 'Привет, ' <Card>>
}
```

Если в выражении имеется несколько вызовов функций, то они выполняются слева направо.

Программы в Рефале-5 (5)

Напишем функцию, которая запрашивает у пользователя строку и заменяет в ней все вхождения слова `cat` на слово `dog`:

```
$ENTRY Go {
  = <Prout <Replace <Card>>>
}

Replace {
 e.Before 'cat' e.After = e.Before 'dog' <Replace e.After>;
  e.NoCat = e.NoCat;
}
```

Программы в Рефале-5 (6)

Напишем функцию, которая запрашивает у пользователя строку и печатает уникальные символы из этой строки.

```
$ENTRY Go {
  = <Prout <Unique <Card>>>
}

Unique {
  e.Begin s.Rep e.Mid s.Rep e.End
    = <Unique e.Begin s.Rep e.Mid e.End>;

  e.Unique = e.Unique;
}
```

Программы в Рефале-5 (7)

Функцию Unique можно немного оптимизировать:

```
$ENTRY Go {
  = <Prout <Unique <Card>>>
}

Unique {
  e.Begin s.Rep e.Mid s.Rep e.End
    = e.Begin <Unique s.Rep e.Mid e.End>;

  e.Unique = e.Unique;
}
```

Программы в Рефале-5 (8)

Напишем программу, которая принимает у пользователя две строки и находит пересечение множеств символов, составляющих эти строки.

```
$ENTRY Go {
  = <Prout <Intersect (<Unique <Card>>) (<Unique <Card>>)>>
}

Intersect {
  (e.Set1-B s.Rep e.Set1-E) (e.Set2-B s.Rep e.Set2-E)
      = s.Rep <Intersect (e.Set1-B e.Set1-E) (e.Set2-
B e.Set2-E)>;

  (e.Set1) (e.Set2) = /* пусто */;
}

Unique {
   ... см. ранее ...
}
```

Программы в Рефале-5 (8)

Её тоже можно оптимизировать:

```
$ENTRY Go {
  = <Prout <Intersect (<Unique <Card>>) (<Unique <Card>>)>>
}

Intersect {
  (e.Set1-B s.Rep e.Set1-E) (e.Set2-B s.Rep e.Set2-E)
    = s.Rep <Intersect (e.Set1-E) (e.Set2-B e.Set2-E)>;

  (e.Set1) (e.Set2) = /* пусто */;
}

Unique {
   ... см. ранее ...
}
```