Рефал и Агда как воплощения идеи "метаалгоритмического языка"

С.А.Романенко

ИПМ им. М.В. Келдыша РАН, Москва

"Научный сервис в сети Интернет"

Пансионат «Моряк», г. Новороссийск

19-22 сентября 2017

Появление языка Рефал

Предложен В.Ф.Турчиным в 1966 году:

- Турчин В.Ф. **Метаязык для формального описания алгоритмических языков.** // В сборнике: Цифровая вычислительная техника и программирование. М.: Советское радио, 1966, с. 116–119.
- Турчин В.Ф. **Метаалгоритмический язык.** // Кибернетика № 4, 1968, с. 116–124.

Рефал = **РЕ**курсивно-**Ф**ункциональный **АЛ**горитмический язык

Особенности Рефала

- Обработка **символьной информации** (цепочек и деревьев).
- **Декларативность:** описываем не последовательность действий, а то, что нам хочется *получить* (и *из чего*).
- **Управления в программе** организовано не с помощью переходов и циклов, а с помощью *вызовов функций* и *рекурсии*.
- **Сопоставление с образцами** используется для организации *разборов случаев* и *ветвлений* в программе (вместо условных переходов и других подобных управляющих конструкций).

"Экзотичность" Рефала (при появлении)

Не из-за "сектантства", а вследствие его *предназначения*!

Предполагалось использовать Рефал не как "обычный" язык программирования, а для "мета-деятельности".

Например:

- Описание семантики языков программирования.
- Быстрое изготовление реализаций языков программирования.

Удалось ли использовать как "мета-язык"?

Да! Одно из применений – компилятор с Алгола-60 в язык ассемблера:

- Турчин В.Ф. **Транслятор с АЛГОЛа, написанный на языке РЕФАЛ.** // В сб.: Труды 1-ой всесоюзной конференции по программированию. В. Процессоры с известных языков. – Киев: 1968, с. 134-151.

А если – "нецелевым образом"?

> Не в качестве (диковинного) "метаалгоритмического",
> а в качестве (обыкновенного) "алгоритмического" языка?

Для задач, где требоваласть работа не с числами, а с какой-то "символьной" информацией (что было необычно в 1966 г.).

Например, для "машинной аналитики" (работы с формулами)?

Попробовали – и получилось:

- Турчин В.Ф., Сердобольский В.И. **Язык Рефал и его использование для преобразования алгебраических выражений.** // Кибернетика № 3, 1969, с. 58–62.

Решение других задач "имеющих большое народнохозяйственное значение"

> Дело дошло даже до реализации Рефала "в металле"!

- Задыхайло И.Б., Котов Е.И., Мямлин А.Н., Поздняков Л.А., Смирнов В.К. **Вычислительная система с внутренним языком повышенного уровня.** // М.: ИПМ АН СССР, 1975, препринт № 41. – 42 с.
- Арсентьева Н.Г., Янова Э.К. **Опыт программирования одной лингвистической задачи на языке РЕФАЛ.** – М.: ИПМ АН СССР, 1976, препринт № 113. – 37 с.
- Наумов Н.А., Рубин А.Г., Смирнов В.К. **Об одном способе реализации входных языков для символьного процессора.** – М.:ИПМ им.М.В.Келдыша АН СССР, 1981, препринт № 146. – 27 с.

Возвращение к первоначальным целям

> Рефал – язык. Значит – можно применить Рефал к Рефалу!

- Турчин В.Ф. **Эквивалентные преобразования рекурсивных функций, описанных на языке РЕФАЛ.** В сб.: Труды симпозиума "Теория языков и методы построения систем программирования". – Киев-Алушта: 1972, с. 31-42.
- Турчин В.Ф. **Эквивалентные преобразования программ на РЕФАЛе.** В сб.: Автоматизированная система управления строительством. Труды ЦНИПИАСС, N 6. – Москва: ЦНИПИАСС, 1974, с. 36-68.
- *Аноним А.А.* **Базисный Рефал и его реализация на вычислительных машинах.** // М.: ЦНИПИАСС, 1977.

Специфика эпохи: "Имя твоё неизвестно..."

Книга по Рефалу была издана ***анонимно***! 😀 👏

Правильная ссылка должна выглядить так:

- Анд.В.Климов, Арк.В.Климов, А.Г.Красовский, С.А.Романенко, Е.В.Травкина, В.Ф.Турчин, В.Ф.Хорошевский, И.Б.Щенков. **Базисный РЕФАЛ и его реализация на вычислительных машинах (методические рекомендации).** Фонд алгоритмов и программ для ЭВМ (в отрасли "Строительство"), специальный раздел, vol. 5, N 40. Москва, 1977. – 258 с.

(Впрочем, авторам были выданы *официальные справки*, какие страницы кем были написаны.)

Является ли Рефал "живым" языком?

В момент появления Рефала он резко отличался от Фортрана, Алгола-60 и Кобола, ибо предлагал:

- обработку символьной информации (цепочки и деревья),
- декларативность,
- функциональность и рекурсия (вместо циклов),
- сопоставление с образцом (для разбора случаев).

> Сегодня эти идеи и принципы никуда не делись!
> Они "растащены" по другим языкам и выглядят как нечто "естественное"!

Пример – язык ***Агда***.

(Агда – это Рефал в овечьей/волчьей шкуре??? 🐑 или 🐺 ?)

Язык Агда

- Norell U. *Towards a practical programming language based on dependent type theory*. Ph.D. thesis, Chalmers University of Technology and Göteborg University, 2007.
- The Agda wiki. URL: http://wiki.portal.chalmers.se/agda/.

> Синтаксис у Рефала и Агды разный...
> А основа – та же самая! (= Растут из одного корня.)

Однако, при использовании Рефала и Агды в качестве "метаалгоритмических" языков, Агда предоставляет кое-какие **дополнительные возможности**.

> Сравним Рефал и Агду на конкретном примере!

Рефал: обрабатываемые данные

- Рефал-программы обрабатывают **выражения**, состоящие из **символов** и круглых **скобок**.
- Символ – это либо **число**, либо **идентификатор** (начинающийся с заглавной буквы), либо **последовательность знаков**, вроде `+*"%?
- Символы и выражения вида `( ... )` называются **термами**.

Примеры:

- `1966`, `Add`, `+` – символы, `25`,
- `Add`, `(2 + 3)` – термы,
- `Push 5` – выражение из двух термов: `Push` и `5`.

Рефал: интерпретатор арифметических выражений

Арифметическое выражение:

- либо число (символ),
- либо терм (t1 + t2), где t1 и t2 – арифметические выражения.

Интерпретатор таких выражений на Рефале:

```
<Eval sN> = sN
<Eval (t1 + t2)> = <Add <Eval t1> <Eval t2>>
```

- Программа на Рефале – набор **предложений**. Левая часть предложения эквивалентна правой (= сводится к правой).
- Предложения соответствуют разным **случаям**.
- Случаи распознаются с помощью **образцов**.
- `Add` – примитивная (встроенная) функция.

Рефал: вычисление значения выражения

В Рефале вложенные вызовы функций вычисляются изнутри наружу (= "вызов по значению").

```
<Eval (1 + 2)> →
<Add <Eval 1> <Eval 2>> →
<Add 1 <Eval 2>> →
<Add 1 2> →
3
```

Рефал: компилятор арифметических выражений

Скомпилированная "программа" состоит из "команд" `Push`, `Add` и `Seq`.

- `(Push n)` помещает число `n` на вершину стека.
- `Add` складывает два числа с вершины стека и помещает результат на вершину стека.
- `(Seq c1 c2)` последовательность из двух команд. Сначала нужно выполнить `c1`, а потом – `c2`.

Компилятор:

```
<Compile sN> = (Push sN)
<Compile (t1 + t2)> =
  (Seq (Seq <Compile t1> <Compile t2>) Add)
```

Рефал: интерпретатор "программ" (= их семантика)

`<Exec c s>` – результат исполнения "программы" `c` по отношению к стеку `s`. Считаем, что пустой стек имеет вид `()`, непустой – вид `(n s)`, где `n` – вершина стека, а `s` – остаток стека.

```
<Exec (Seq t1 t2) tS> = <Exec t2 <Exec t1 tS>>
<Exec (Push sN) tS = (sN tS)
<Exec Add (s2 (s1 tS)) = (<Add s1 s2> tS)
```

Вывод: Рефал может использоваться для определения интерпретаторов и компиляторов других языков.

Проблема: непонятно, как формализовать на Рефале теорему корректности компилятора по отношению к интерпретатору...

```
<Exec <Compile t> s> ≡ (<Eval t> s)
```

Посмотрим, как обстоит дело в случае Агды.

Агда: интерпретатор арифметических выражений

> Агда – язык со **статической типизацией**.

В случае Рефала, структура арифметических выражений объяснялась неформально, отдельно от программы.

В случае Агды, описание структуры данных – часть программы (и компилятор проверяет, что данные и программа не противоречат друг другу).

```
data Tm : Set where
  val : (n : ℕ) → Tm
  _⊕_  : (t1 t2 : Tm) → Tm

eval : Tm → ℕ
eval (val n) = n
eval (t1 ⊕ t2) = eval t1 + eval t2
```

Агда: "программы" и их интерпретатор

> Агда позволяет использовать "зависимые типы".

Пусть кусок кода `c` имеет тип `Code i j`. Это означает, что исполнение кода `c` применительно к стеку `s`, имеющему глубину `i`, порождает стек глубины `j`.

```
data Code : (i j : ℕ) → Set where
  seq  : ∀ {i j k}
           (c1 : Code i j) (c2 : Code j k) → Code i k
  push : ∀ {i} (n : ℕ) → Code i (1 + i)
  add  : ∀ {i} → Code (2 + i) (1 + i)
```

> Агда проверяет, что `compile` выдает код, который увеличивает стек на 1! (Фактически – доказывает теорему.)

```
compile : ∀ {i} (t : Tm) → Code i (1 + i)
compile (val n) = push n
compile (t1 ⊕ t2) = seq (seq (compile t1) (compile t2)) add
```

Агда: интерпретатор "программ"

`exec` получает программу типа `Code i j` и стек глубины `i`, и выдает стек глубины `j`.

```
Stack : ℕ → Set
Stack i = Vec ℕ i

exec : ∀ {i j} (c : Code i j) (s : Stack i) → Stack j
exec (seq c1 c2) s = exec c2 (exec c1 s)
exec (push n) s = n :: s
exec add (n2 :: n1 :: s) = (n1 + n2) :: s
```

`Stack i` представляется вектором длины `i`. Таким образом, пустой стек – это `[]`, а непустой – выглядит как `n :: s`, где `n` – число на вершине стека, а `s` – остаток стека.

> Агда проверяет, что `exec` правильно изменяет глубину стека.

Агда: формулировка корректности `compile`

Корректность `compile` и `exec` по отношению к `eval`:

> если задано выражение `t` и стек `s`, то вычисление `exec (compile t) s` помещает `eval t` на вершину стека.

Или формально на Агде:

```
correct : ∀ {i} (t : Tm) (s : Stack i) →
  exec (compile t) s ≡ eval t ∷ s
```

> Это – пример **соответствия Карри-Ховарда**, которое заключается в том, что всякий тип `τ` – это некоторое "утверждение", а любое значение `v`, имеющее этот тип, является "доказательством" утверждения `τ`.

Теперь нужно определить "доказательство" – функцию `correct`!

Агда: доказательство корректности `compile`

```
correct (val n) s =
  exec (compile (val n)) s ≡⟨⟩ n :: s ≡⟨⟩ eval (val n) :: s ■
correct (t1 ⊕ t2) s =
  exec (compile (t1 ⊕ t2)) s
    ≡⟨⟩
  exec (seq (seq c1 c2) add) s
    ≡⟨⟩
  exec add (exec c2 (exec c1 s))
    ≡⟨ cong (exec add ∘ exec c2) (correct t1 s) ⟩
  exec add (exec c2 (n1 :: s))
    ≡⟨ cong (exec add) (correct t2 (n1 :: s)) ⟩
  exec add (n2 :: n1 :: s)
    ≡⟨⟩
  n1 + n2 :: s
    ≡⟨⟩
  eval (t1 ⊕ t2) :: s ■
  where
  n1 = eval t1; n2 = eval t2
  c1 = compile t1; c2 = compile t2
```

Агда: структура доказательства

Разбор случаев

```
correct (val n) s = ...
correct (t1 ⊕ t2) s = ...
```

Цепочки равенств

```
t1 ≡⟨ p ⟩ t2 ≡⟨⟩ t3
```

- `t1 ≡⟨ p ⟩ t2` – `t1` *равно* `t2` согласно доказательству `p`.
- `t1 ≡⟨⟩ t2` – `t1` *совпадает* с `t2` (после приведения к нормальной форме).

Индукция = рекурсия

```
    = ... correct t1 s ... correct t2 (n1 :: s) ...
```

Чем хороша Агда с точки зрения *программиста*?

- Топор хорош для колки дров, а молоток – для забивания гвоздей.
- Достоинства/недостатки инструмента следует оценивать не абстрактно, а применительно к **решаемой задаче**.

Если **программист** использует Агду для записи утверждений и доказательств о **свойствах программ**, то её достоинства таковы.

- Не нужно учить два разных языка: и программы, и доказательства записываются **на одном и том же** языке.
- Доказательства = программы. Можно воспользоваться **программистскими** навыки.

Примеры: https://github.com/sergei-romanenko/agda-samples .

Заключение

- Идеи, на которых был основан Рефал (декларативность, функциональность, рекурсия и сопоставление с образцом) в настоящее время можно обнаружить (в той или другой степени) и в других языках.
- Язык Агда основан на тех же самых принципах, что и Рефал, но предоставляет ещё и возможность записывать утверждения о программах и доказательства этих утверждений. При этом проверка правильности доказательств выполняется автоматически.