Отчет по лабораторной работе «Использование автоматических генераторов анализаторов Bison и ANTLR»

студента группы 3538
Андрея Козлова

НИУ ИТМО
2 мая 2013

Вариант 4. Функциональный язык

Придумайте примитивный функциональный язык программирования и реализуйте его трансляцию в императивный язык.

Пример:

```
fac :: Integer -> Integer
fac 0 = 1
fac n | n > 0 = n * fac(n - 1)
```

Вывод:

```
function fac(n: integer): integer;
begin
 if n = 0 then
        fac := 1;
 else if n > 0 then
        fac := n * fac(n - 1);
end;
```

Постановка задачи

В качестве транслируемого языка было решено взять подмножество одного из наиболее популярных функциональных языков – Haskell. В качестве императивного языка был выбран язык Java.

Такой выбор обосновывается следующим соображением: после трансляции кода можно проверить его работоспособность. Скомпилированные исходный код теста и полученный код можно запустить и сравнить выдаваемый результат.

В качестве автоматического генератора анализаторов был выбран ANTLR, и как следствие в качестве языка для выполнения лабораторной работы был также выбран язык Java.

Описание подмножества Haskell

Реализованный язык поддерживает работу только с численными функциями, то есть и аргументы, и возвращаемое значение являются числами. Далее арифметические выражения обозначаются <arithmetic_expession>. Используются стандартные типы данных: Int для целых и Double для вещественных чисел. Поддерживаются элементарные арифметические операции: сложение, вычитание, умножение, деление. Операции целочисленного деления (`div`) и взятия остатка (`mod`) записываются в инфиксной форме. В арифметических выражениях могут присутствовать числа в десятичной системе счисления, переменные, результаты вычисления функций. Вызов функции вместе с ее аргументами нужно писать в скобках.

- пример <arithmetic_expression>: n * (function n), a `mod` 3

Для работы с условиями введена булева логика. Далее логические выражения обозначаются <boolean_expression>. Используются значения стандартного типа данных Bool. Поддерживаются элементарные логические операции: конъюнкция, дизъюнкция, отрицание. На логических выражениях введено отношение эквивалентности, на арифметических – отношения порядка.

- пример <boolean_expression>: n > 0 && (k > 0)

Определение функции делится на две части: интерфейс и реализация. В первом задается название, типы аргументов, тип возвращаемого значения. Реализация состоит из следующих частей: описание аргументов, возвращаемое значение, условие выполнения (опционально). В качестве аргумента может использоваться число или переменная. (Последнее целесообразно, если результат функции или условие выполнения зависит от значения этой переменной. Иначе можно ставить символ подчеркивания.)

- интерфейс: <function_name> '::' (<argument_type> '->')* <return_type>
- реализация: <function_name> (<number> | <variable_name> | '_')* ('|' <boolean_expression>)? '=' <arithmetic_expression>

Названием функции может быть любая последовательность заглавных, строчных букв, знаков подчеркиваний и цифр, не начинающаяся с цифры. В названиях переменных может также встречаться апостроф, но не на первом месте.

Помимо численных функций предусмотрена возможность трансляции функции main, необходимой для вывода данных на экран.

- интерфейс: 'main :: IO()'
- реализация: 'main = print' <arithmetic_expression>

В конце любой строчки может находиться однострочный комментарий, начинающийся символами '--'.

В качестве пробела может использоваться любое число пробелов или знаков табуляции.

Трансляция

В результате трансляции образуется java-класс с публичными статичными методами, код которого соответствует Java Coding Conventions. Комментарии игнорируются.

Названия аргументов функции могут меняться в каждой новой строке реализации. При трансляции все имена заменяются на единообразные: arg0, arg1 и так далее. При необходимости условия рассматриваются при помощи операторов if … else … . При попытке программиста передать неверный аргумент генерируется исключение IllegalArgumentException.

Запуск

Класс Tester, находящийся в директории src/, транслирует все файлы с расширением '.hs' из директории tests/ и сохраняет результат в эту же папку. Название полученного файла – Translated_<filename>.java, где <filename>.hs – название исходного файла.

Скрипт buildGrammar.sh для грамматики в файле src/Language.g генерирует файлы в папку src/.

Скрипт run.sh компилирует исходные файлы с расширениями '.hs' и '.java' из директории tests/, запускает полученные исполняемые файлы и классы и выводит результат на экран.

```
[1 of 1] Compiling Main             ( 00.hs, 00.o )
Linking 00 ...

haskell out:

6

java out:

6

[1 of 1] Compiling Main             ( 01.hs, 01.o )

Linking 01 ...

haskell out:

120

java out:

120
```

Тестирование

00.hs

```
fac :: Int -> Int

constant   ::   Int
constant= 0

strange ::   Int -> Int -> Int
strange _ a_ = a_ `mod` 3


division :: Double -> Double -> Double
division a b = a / b + 0.5 -- kjdkfjkj

fac 0 = 1
fac n | n > 0 = n * (fac (n - 1))



multiplication ::Int -> Int->Int
multiplication n' _3| n'>0= 0
multiplication n' k  | n' < 1 && (k > 0) = 1

main ::     IO()

main = print (fac (5+2 * (3 - 4)))
```

Translated_00.java

```
public class Translated_00 {

 public static int strange(int arg0, int arg1) {
        return arg1 % 3;
 }

 public static double division(double arg0, double arg1) {
        return arg0 / arg1 + 0.5;
 }

 public static int constant() {
        return 0;
 }

 public static int multiplication(int arg0, int arg1) {
        if (arg0 > 0) {
                return 0;
        } else if (arg0 < 1 && (arg1 > 0)) {
                return 1;
        } else {
                throw new IllegalArgumentException();
        }
 }

 public static int fac(int arg0) {
        if (0 == arg0) {
                return 1;
        } else if (arg0 > 0) {
                return arg0 * fac((arg0 - 1));
        } else {
                throw new IllegalArgumentException();
        }
 }

 public static void main(String[] args) {
        System.out.println(fac((5 + 2 * (3 - 4))));
 }

}
```

Ссылки

- Исходный код на github.com.