Регулярные выражения

Романенко Владимир Васильевич,
к.т.н., доцент каф. АСУ ТУСУР

Способы определения языка

Способ 2. Непосредственная запись РВ. В этом случае можно действовать следующим образом:

Вариант 1 (последовательное построение РВ).

- Определить начальную конструкцию, которая может следовать в начале правильной цепочки в языке *L*. Составить РВ, соответствующее этой конструкции.
- Составить РВ для следующей конструкции, и т.д. до тех пор, пока не получим РВ, полностью описывающее язык *L*.

Способы определения языка

При этом можно использовать следующие типовые приёмы:

Приём 1. Конкатенация. Если конструкции a и b должны следовать друг за другом, то этому будет соответствовать часть РВ

…(ab)…

Приём 2. Объединение. Если следующей конструкцией входной цепочки является либо a, либо b, то этому будет соответствовать часть РВ

…(a + b)…

Способы определения языка

При этом можно использовать следующие типовые приёмы:

Приём 3. Положительная итерация. Если конструкция a в данной части входной цепочки может повторяться от 1 и более раз, то этому будет соответствовать часть РВ

$$\ldots(aa^{*})\ldots \text{ или } \ldots(a^{+})\ldots$$

Приём 4. Обычная итерация. Если в предыдущем случае цепочка a может вообще отсутствовать, то

$$\ldots(a^{*})\ldots$$

Способы определения языка

При этом можно использовать следующие типовые приёмы:

Приём 5. Конкатенация с итерацией. Если в данной части входной цепочки после конструкции a может повторяться от 0 и более раз другая конструкция b, то

$$\ldots(ab^{*})\ldots$$

Если конструкция b должна повторяться от 1 и более раз, то

$$\ldots(abb^{*})\ldots \text{ или } \ldots(ab^{+})\ldots$$

Способы определения языка

При этом можно использовать следующие типовые приёмы:

Приём 6. Итерация объединения. Если в данной части входной цепочки может в произвольном порядке располагаться произвольное количество (от 0 и более) конструкций a и b, то

$$\ldots(a+b)^{*}\ldots$$

Если должна быть как минимум одна такая конструкция, то

$$\ldots(a+b)(a+b)^{*}\ldots \text{ или } \ldots(a+b)^{+}\ldots$$

Способы определения языка

При этом можно использовать следующие типовые приёмы:

Приём 7. Итерация конкатенации. Если в данной части входной цепочки может располагаться произвольное количество (от 0 и более) конструкций a и b, следующих друг за другом, то

$$\ldots(ab)^{*}\ldots$$

Если должно быть как минимум одно вхождение таких конструкций, то

$$\ldots(ab)(ab)^{*}\ldots \text{ или } \ldots(ab)^{+}\ldots$$

Примеры определения языка

Пример 1. В языке *L*, описывающем число с фиксированной точкой, в начале входной цепочки может располагаться:

- точка;
- или цифра;
- или знак (плюс или минус).

Поэтому получим следующее РВ:

.x + (0-9)y + (+ + –)z

Т.е. (точка и x) или (цифра и y) или (знак (плюс или минус) и z).

Примеры определения языка

$$.x + (0\text{-}9)y + (+ + -)z$$

1. Начнём с точки (.). После точки могут следовать только цифры, от 1 и более, поэтому

$$x = (0\text{-}9)^{+} \Rightarrow .x = .(0\text{-}9)^{+}$$

2. Далее, цифра (0-9). Это значит, у числа есть целая часть. После первой цифры могут следовать от 0 и более других цифр (цифра и от 0 и более других цифр), т.е. получим

$$y = (0\text{-}9)^{*}y_2 \Rightarrow (0\text{-}9)y = (0\text{-}9)(0\text{-}9)^{*}y_2 = (0\text{-}9)^{+}y_2$$

Примеры определения языка

$$.(0\text{-}9)^{+} + (0\text{-}9)^{+}y_2 + (+ + -)z$$

После целой части может следовать:

- точка.
- или ничего (конец входной цепочки).

$$(0\text{-}9)^{+}y_2 = (0\text{-}9)^{+}.y_3 + (0\text{-}9)^{+}e$$

После точки могут следовать:

- цифры, от 1 и более (дробная часть числа);
- или ничего.

$$(0\text{-}9)^{+}.y_3 = (0\text{-}9)^{+}.(0\text{-}9)^{+} + (0\text{-}9)^{+}.e$$

Примеры определения языка

$.(0\text{-}9)^+ + (0\text{-}9)^+.(0\text{-}9)^+ + (0\text{-}9)^+. + (0\text{-}9)^+ + (+ + -)z$

3. И последнее – знак, плюс или минус (+ + –). После него может следовать точка или цифра и т.д., т.е. уже полученные ранее выражения:

$$z = .x + (0\text{-}9)y =$$

$$= .(0\text{-}9)^+ + (0\text{-}9)^+.(0\text{-}9)^+ + (0\text{-}9)^+. + (0\text{-}9)^+$$

Окончательно имеем:

$$.(0\text{-}9)^+ + (0\text{-}9)^+.(0\text{-}9)^+ + (0\text{-}9)^+. + (0\text{-}9)^+ +$$

$$(+ + -)(.(0\text{-}9)^+ + (0\text{-}9)^+.(0\text{-}9)^+ + (0\text{-}9)^+. + (0\text{-}9)^+)$$

Способы определения языка

Вариант 2 (построение РВ методом декомпозиции).

- Определяем конструкции верхнего уровня, т.е. самые главные блоки, из которых состоят цепочки языка.
- Каждый полученный блок декомпозируем на более мелкие блоки до тех пор, пока все блоки не будут записаны в виде элементарных операций РВ.

Примеры определения языка

Пример 2. Цепочки языка L, описывающего число с фиксированной точкой, на самом верхнем уровне состоят из необязательного знака и мантиссы:

[знак]мантисса

Т.е. знак может отсутствовать, либо это может быть плюс или минус:

$(+ \; + \; - \; + \; e)$мантисса

Примеры определения языка

Как мы уже выясняли ранее, мантисса может принимать четыре разные формы: «*N*.*M*», «*N*.», «.*M*», «*N*», где *N* – целая, а *M* – дробная часть числа, т.е. получим

$$\text{мантисса} = (\text{N.M} + \text{N.} + \text{.M} + \text{N})$$

Также мы отмечали, что числа *N* и *M* имеют одинаковый формат – это последовательность из одной и более цифр в диапазоне от 0 до 9, т.е.

$$\text{N} = \text{M} = (0\text{-}9)^{+}.$$

Примеры определения языка

Окончательно получим

$$(+ + - + e)(N.M + N. + .M + N) =$$

$$= (+ + - + e)((0\text{-}9)^{+}.(0\text{-}9)^{+} + (0\text{-}9)^{+}. + .(0\text{-}9)^{+} + (0\text{-}9)^{+})$$

Оба полученных выражения можно упростить, чтобы они совпали с полученным способом №1 выражением

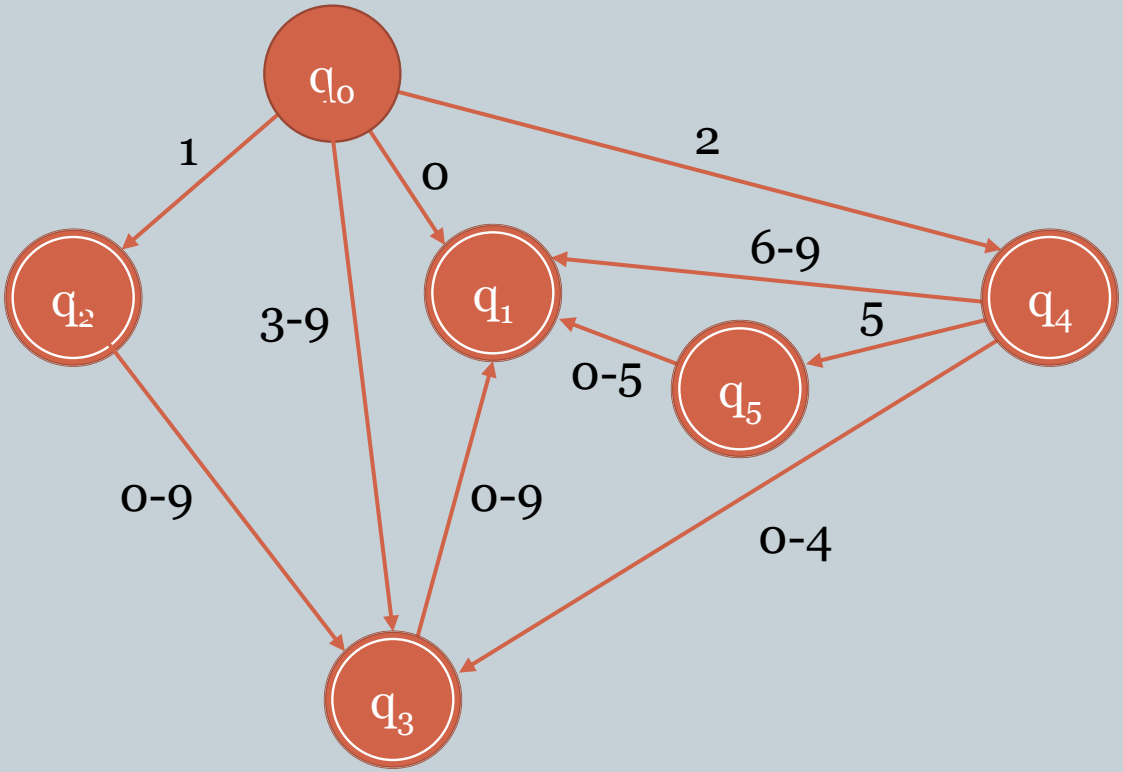
$$(+ + - + e)(.(0\text{-}9)^{+} + (0\text{-}9)^{+}(e + .(0\text{-}9)^{*}))$$

При этом полагается, что

$$0\text{-}9 = 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9$$

Примеры определения языка

Пример 3. Язык *L* описывает десятичные числа в диапазоне от 0 до 255, без ведущих нулей.



Примеры определения языка

Получим РВ:

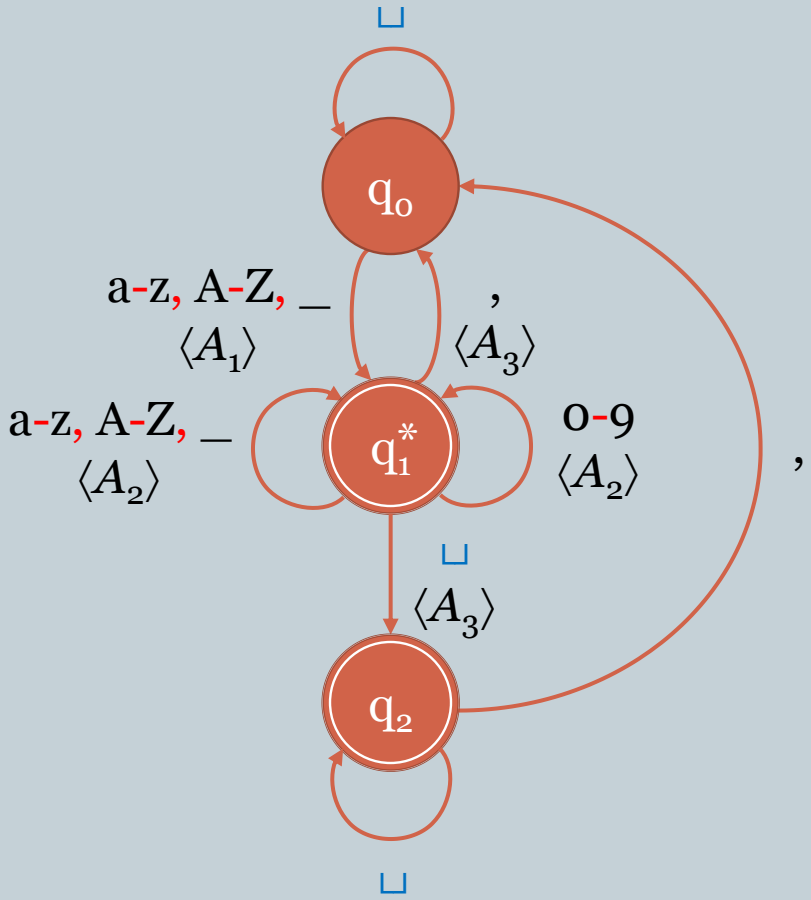
$$0 + 1(e + (0\text{-}9) + (0\text{-}9)(0\text{-}9)) +$$
$$+ 2(e + (0\text{-}4)(e + (0\text{-}9)) + 5(e + (0\text{-}5)) + (6\text{-}9)) +$$
$$+ (3\text{-}9)(e + (0\text{-}9))$$

Примеры определения языка

Пример 4. Пусть язык *L* описывает список идентификаторов, разделенных запятыми. Идентификатор должен начинаться с буквы латинского алфавита или подчёркивания, далее могут следовать другие необязательные буквы, подчёркивания или цифры. Также в этой записи допустимы пробелы и другие символы-разделители в начале и конце входной цепочки, а также между идентификаторами и запятыми. Идентификаторы должны быть уникальными.

Примеры определения языка

Граф функции переходов:



Примеры определения языка

id, id, ..., id =

= (id,)*id или id(,id)*

␣* id ␣*, ␣* id ␣* , ␣* ...

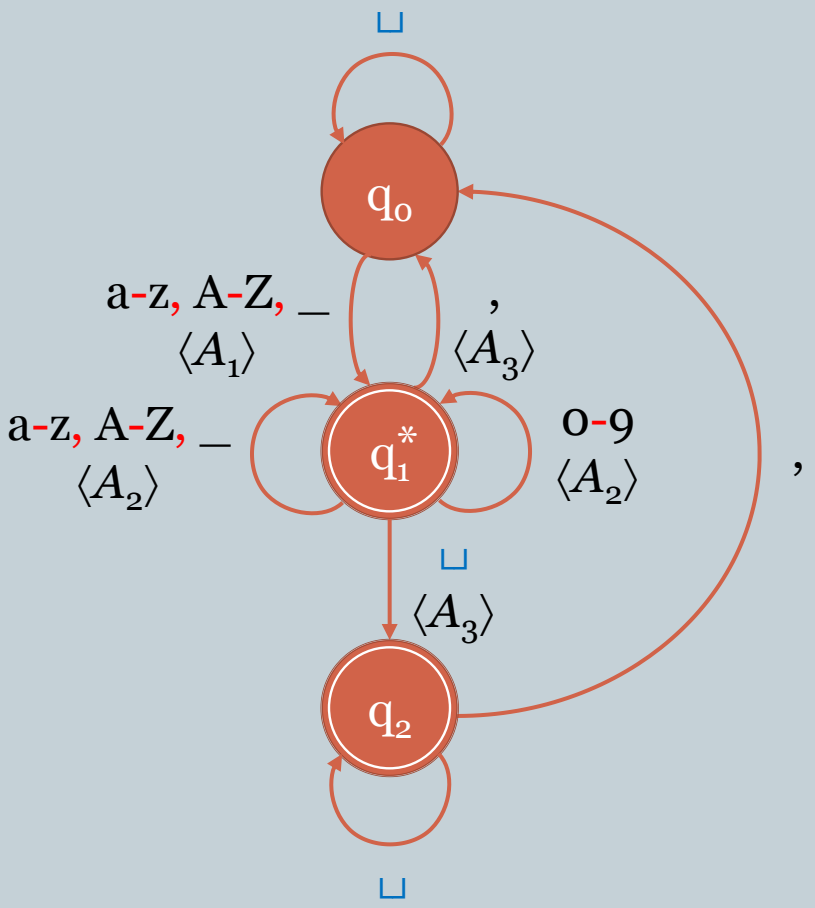
␣* , ␣* id ␣* =

(␣* id ␣* , ␣*)* ␣* id ␣* =

(␣* id ␣* ,)* ␣* id ␣*

или

␣* id ␣* (, ␣* id ␣*)*



Примеры определения языка

Получим РВ:

$$␣^{*}\ \mathrm{id}\ ␣^{*}\ (,␣^{*}\ \mathrm{id}\ ␣^{*})^{*} =$$

$$= ␣^{*}\ (\_ + a\text{-}z + A\text{-}Z)(\_ + a\text{-}z + A\text{-}Z + 0\text{-}9)^{*}\ ␣^{*}$$
$$(,␣^{*}\ (\_ + a\text{-}z + A\text{-}Z)(\_ + a\text{-}z + A\text{-}Z + 0\text{-}9)^{*}\ ␣^{*})^{*}$$

При этом полагается, что

$$0\text{-}9 = 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9,$$

$$a\text{-}z = a + b + c + \ldots + z,$$

$$A\text{-}Z = A + B + C + \ldots + Z$$

Программирование РВ

Все современные языки программирования позволяют работать с регулярными выражениями. При этом часть языков (PHP, JavaScript и т.д.) имеют встроенные средства для работы с РВ, а в других есть готовые библиотеки или классы для работы с ними, например, класс Regex для платформы .NET:

```
using System.Text.RegularExpressions;
Regex r = new Regex("шаблон");
```

Программирование PB

Символ	Интерпретация
Escape-последовательности	
\b	При использовании его в квадратных скобках соответствует символу «←» (\u0008)
\t, \r, \n, \a, \f, \v	Табуляция (\u0009), возврат каретки (\u000D), новая строка (\u000A) и т.д.
\cX	Управляющий символ (например, \cC – это Ctrl+C, \u0003)
\e	Escape (\u001B)
\ooo	Символ ASCII в восьмеричной системе
\xhh	Символ ASCII в шестнадцатеричной системе
\uhhhh	Символ Unicode
\	Следующий символ не является специальным символом PB. Этим символом нужно экранировать все специальные символы
Пример (в примере приведен шаблон и строка поиска, в строке найденные совпадения подчеркнуты): @"\r\n\w+" – "\r\nЗдесь имеются\nдве строки".	

Программирование РВ

Подмножества символов	
.	Любой символ, кроме конца строки (\n)
[xxx]	Любой символ из множества
[^xxx]	Любой символ, кроме символов из множества
[x-x]	Любой символ из диапазона
[xxx-[xxx]]	Вычитание одного множества или диапазона из другого
\p{name}	Любой символ, заданный категорией Unicode с именем name
\P{name}	Любой символ, кроме заданных категорией Unicode с именем name
\w	Множество символов, используемых при задании идентификаторов
\W	Множество символов, не используемых при задании идентификаторов
\s	Пробелы
\S	Все, кроме пробелов
\d	Цифры
\D	Не цифры
Примеры: @".+" – "\r\nЗдесь имеются\nдве строки"; @"[fx]+" – "0xabcfx"; @"[^fx]+" – "0xabcfx"; @"[a-f]+" – "0xabcfx"; @"[^a-f]+" – "0xabcfx"; @"[a-z-[c]]+" – "0xabcfx"; @"\p{Lu}" – "City Lights"; @"\P{Lu}" – "city"; @"\p{IsCyrillic}" – "xaOS"; @"\P{IsCyrillic}" – "xaOS". // Lu – прописные латинские буквы , IsCyrillic – русские буквы	

Программирование РВ

Привязка	
^, \A	В начале строки
$, \Z	В конце строки или до символа «\n» в конце строки
\z	В конце строки
\G	В том месте, где заканчивается предыдущее соответствие
\b	Граница слова
\B	Любая позиция не на границе слова

Примеры:

@"\G\(\d\)" – "(1)(3)(5)[7](9) "; // три соответствия (1), (2) и (3)

@"\bn\S*ion\b" – "nation donation";

@"\Bend\w*\b" – "end sends endure lender".

Программирование РВ

Операции (кванторы)	
*, *?	Итерация
+, +?	Положительная итерация
?, ??	Ноль или одно соответствие
{n}, {n}?	Точно n соответствий
{n,}, {n,}?	По меньшей мере, n соответствий
{n,m},{n,m}?	От n до m соответствий

Примеры (первые кванторы – жадные, ищут как можно большее число элементов, вторые – ленивые, ищут как можно меньшее число элементов):

@"\d{3,}" – "888-555-5555";

@"^\d{3}" – "913-913-913";

@"-\d{3}$" – "913-913-913";

@"5+?5" – "888-555-5555"; // три совпадения – 55, 55 и 55

@"5+5" – "888-555-5555".

Программирование РВ

Группирование	
()	Группа, автоматически получающая номер
(?:)	Не сохранять группу
(?<имя>) или (?'имя')	При обнаружении соответствия создается именованная группа
(?<имя–имя>) или (?'имя–имя')	Удаление ранее определенной группы и сохранение в новой группе подстроки между ранее определенной группой и новой группой
(?imnsx:) (?–imnsx:)	Включает или выключает в группе любую из пяти возможных опций: i – нечувствительность к регистру; s – одна строка (тогда «.» – это любой символ); m – многострочный режим («^», «$» – начало и конец каждой строки); n – не захватывать неименованные группы; x – исключить не преобразованные в escape-последовательность пробелы из шаблона и включить комментарии после знака номера (#)
(?=)	Положительное утверждение просмотра вперед нулевой длины

Программирование РВ

Группирование (продолжение)	
(?!)	Отрицательное утверждение просмотра вперед нулевой длины
(?<=)	Положительное утверждение просмотра назад нулевой длины
(?<!)	Отрицательное утверждение просмотра назад нулевой длины
(?>)	Невозвращаемая (жадная) часть выражения

Примеры:

@"(an)+" – "bananas annals";

@"an+" – "bananas annals"; // сравните, три совпадения – an, an и ann

@"(?i:an)+" – "baNAnas annals";

@"[a-z]+(?=\d)" – "abc xyz12 555w";

@"(?<=\d)[a-z]+" – "abc xyz12 555w".

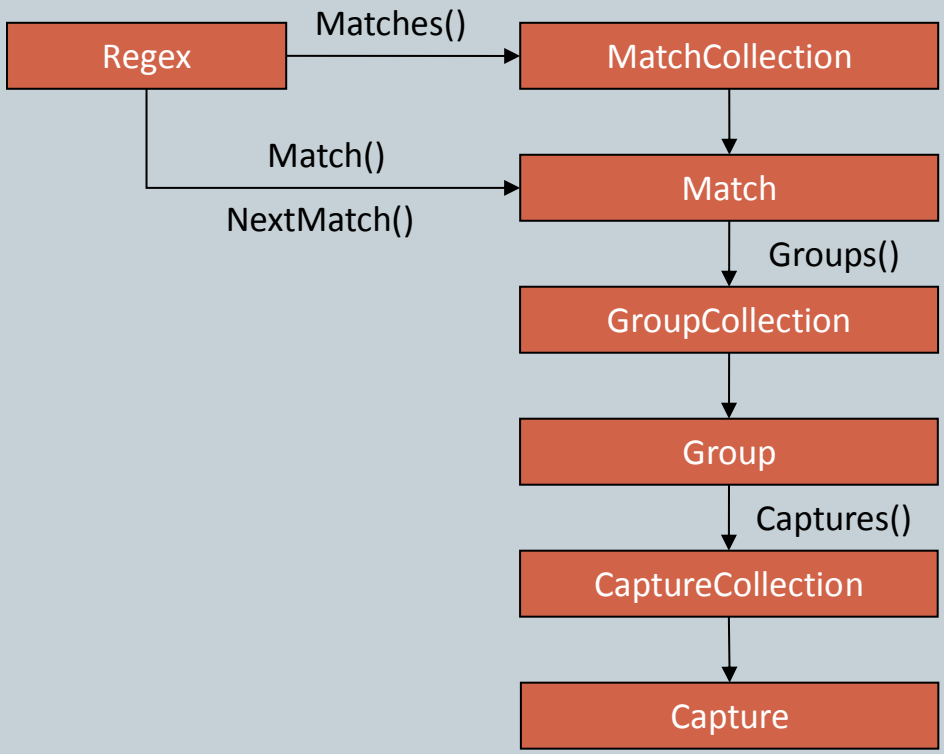
Программирование РВ

Ссылки	
\число	Ссылка на группу
\k<имя>	Ссылка на именованную группу
Примеры: @"(\w)\1" – "deep"; @"(?<char>\w)\k<char>" – "deep".	
Конструкции изменения	
|	Альтернатива (соответствует операции объединения)
(?(выражение)да|нет)	Сопоставляется с частью «да», если выражение соответствует; в противном случае сопоставляется с необязательной частью «нет»
(?(имя)да|нет), (?(число)да|нет)	Сопоставляется с частью «да», если названное имя захвата имеет соответствие; в противном случае сопоставляется с необязательной частью «нет»
Пример: @"th(e|is|at)" – "this is the day";	

Программирование PB

Подстановки	
$число	Замещается часть строки, соответствующая группе с указанным номером
${имя}	Замещается часть строки, соответствующая группе с указанным именем
$$	Подставляется $
$&	Замещение копией полного соответствия
$`	Замещение текста входной строки до соответствия
$'	Замещение текста входной строки после соответствия
$+	Замещение последней захваченной группы
$_	Замещение всей строки
Комментарии	
(?#)	Встроенный комментарий
#	Комментарий до конца строки

Результаты работы Regex:



Программирование РВ

Пример:

```
Regex r = new Regex(@"((\d)+)+");
Match m = r.Match("123 456");
int matchCount = 0;
while (m.Success)
{
    Console.WriteLine("Соответствие {0}", ++matchCount);
    for (int i = 1; i < m.Groups.Count; i++)
    {
        Group g = m.Groups[i];
        Console.WriteLine("   Группа {0} = '{1}'", i, g.Value);
        for (int j = 0; j < g.Captures.Count; j++)
        {
            Capture c = g.Captures[j];
            Console.WriteLine("      Захват {0} = '{1}', позиция = {2},
длина = {3}", j, c, c.Index, c.Length);
        }
    }
    m = m.NextMatch();
}
```

```
Соответствие 1
  Группа 1 = '123'
    Захват 0 = '123', позиция = 0, длина = 3
  Группа 2 = '3'
    Захват 0 = '1', позиция = 0, длина = 1
    Захват 1 = '2', позиция = 1, длина = 1
    Захват 2 = '3', позиция = 2, длина = 1
Соответствие 2
  Группа 1 = '456'
    Захват 0 = '456', позиция = 4, длина = 3
  Группа 2 = '6'
    Захват 0 = '4', позиция = 4, длина = 1
    Захват 1 = '5', позиция = 5, длина = 1
    Захват 2 = '6', позиция = 6, длина = 1
```

Программирование РВ

Пример 1. Язык *L*, описывающий двоичные числа без незначащих нулей:

$$0 + 1(0 + 1)^* \Rightarrow \text{"}\hat{}(0|1(0|1)^*)\$\text{"}$$

Программирование РВ

Пример 2. Язык L, описывающий числа с фиксированной точкой:

$$(+ + - + e)(.(0\text{-}9)^{+} + (0\text{-}9)^{+}(e + .(0\text{-}9)^{*}))$$

Во-первых, 0-9 ⇒ 0|1|2|3|4|5|6|7|8|9 = [0-9] = \d

Во-вторых, $+ + - + e \Rightarrow$ (\+|-)?

$e + .(0\text{-}9)^{*} \Rightarrow$ (\.\d*)?

Таким образом,

⇒ "^(\\+|-)?(\\.\\d+|\\d+(\\.\\d*)?)$" или

@"^(\+|-)?(\.\d+|\d+(\.\d*)?)$"

Программирование РВ

Пример 3. Язык L описывает десятичные числа в диапазоне от 0 до 255, без ведущих нулей:

$$0 + 1(e + (0\text{-}9) + (0\text{-}9)(0\text{-}9)) +$$
$$+ 2(e + (0\text{-}4)(e + (0\text{-}9)) + 5(e + (0\text{-}5)) + (6\text{-}9)) +$$
$$+ (3\text{-}9)(e + (0\text{-}9)) \Rightarrow$$

@"^(0|1\d{0,2}|2([0-4]\d?|5[0-5]?|[6-9])?|
[3-9]\d?)$"

Или

\d{0,2} = \d?\d?

Программирование РВ

Пример 4. Язык L описывает список уникальных идентификаторов:

$$␣^* (\_ + a\text{-}z + A\text{-}Z)(\_ + a\text{-}z + A\text{-}Z + 0\text{-}9)^* ␣^*$$
$$(, ␣^* (\_ + a\text{-}z + A\text{-}Z)(\_ + a\text{-}z + A\text{-}Z + 0\text{-}9)^* ␣^*)^*$$

$$\Rightarrow @"\text{^}\backslash s*[\_a\text{-}zA\text{-}Z][\_a\text{-}zA\text{-}Z0\text{-}9]*\backslash s*$$
$$(,\backslash s*[\_a\text{-}zA\text{-}Z][\_a\text{-}zA\text{-}Z0\text{-}9]*\backslash s*)*\$"$$

или

$$@"\text{^}((?<!\text{^}),|(?<=\text{^}))\backslash s*$$
$$[\_a\text{-}zA\text{-}Z][\_a\text{-}zA\text{-}Z\backslash d]*\backslash s*)*\$"$$

Программирование РВ

Пример 4. Язык *L* описывает список уникальных идентификаторов:

$$␣^* (\_ + a\text{-}z + A\text{-}Z)(\_ + a\text{-}z + A\text{-}Z + 0\text{-}9)^* ␣^*$$
$$(, ␣^* (\_ + a\text{-}z + A\text{-}Z)(\_ + a\text{-}z + A\text{-}Z + 0\text{-}9)^* ␣^*)^*$$

$$\Rightarrow$$ @"^\s*[_a-zA-Z][_a-zA-Z0-9]*\s*
(,\s*[_a-zA-Z][_a-zA-Z0-9]*\s*)*$"

или (с русскими буквами)

@"^\s*[_\p{L}]\w*\s*(,\s*[_\p{L}]\w*\s*)*$"

Внедрение действий в синтаксис

Внедрение действий осуществляется с помощью анализа групп вхождений:

- нумерованных;
- именованных.

С нумерованными группами работать сложнее.

Внедрение действий в синтаксис

Пример 1. Язык *L* описывает список уникальных идентификаторов:

```
@"^(((?<!^),|(?<=^))\s*
[_a-zA-Z][_a-zA-Z\d]*\s*)*$"
```

Входная строка «L1b2 , cdX34»:

```
Соответствие 1
   Группа 1 = ', cdX34 '
      Захват 0 = ' L1b2 ', позиция = 0, длина = 6
      Захват 1 = ', cdX34 ', позиция = 6, длина = 8
   Группа 2 = ','
      Захват 0 = '', позиция = 0, длина = 0
      Захват 1 = ',', позиция = 6, длина = 1
```

Внедрение действий в синтаксис

Модифицируем выражение:

```
@"^(((?<!^),|(?<=^))\s*
([_a-zA-Z][_a-zA-Z\d]*)\s*)*$"
```

```
Соответствие 1
   Группа 1 = ', cdX34 '
      Захват 0 = ' L1b2 ', позиция = 0, длина = 6
      Захват 1 = ', cdX34 ', позиция = 6, длина = 8
   Группа 2 = ','
      Захват 0 = '', позиция = 0, длина = 0
      Захват 1 = ',', позиция = 6, длина = 1
   Группа 3 = 'cdX34'
      Захват 0 = 'L1b2', позиция = 1, длина = 4
      Захват 1 = 'cdX34', позиция = 8, длина = 5
```

Внедрение действий в синтаксис

Введём именованную группу:

```
@"^(((?<!^),|(?<=^))\s*
(?<id>[_a-zA-Z][_a-zA-Z\d]*)\s*)*$"
```

```
Match m = r.Match(" L1b2 , cdX34 ");
foreach (Capture c in m.Groups["id"].Captures)
{
    Console.WriteLine(c.Value);
}
```

```
L1b2
cdX34
```

Внедрение действий в синтаксис

Пример 2. Язык *L* описывает десятичные числа в диапазоне от 0 до 255, без ведущих нулей:

@"^(0|1\d{0,2}|2([0-4]\d?|5[0-5]?|[6-9])?|[3-9]\d?)$"

Введём именованную группу:

@"^(?<num>\d+)$"

```
Match m = r.Match("123");
int matchCount = 0;
int value;
if (!int.TryParse(m.Groups["num"].Value, out value) ||
      value > 255) Console.WriteLine("Ошибка!");
```

Сбалансированные определения

Для проверки рекурсивно вложенных описаний (таких, для анализа которых используется ДМПА) применяются *сбалансированные определения*. В этом случае именованные группы играют роль стека. Так, конструкция «(?'x')» добавляет в коллекцию с именем «x» один элемент, а конструкция «(?'-x')» убирает из коллекции «x» один элемент. Поэтому в конце остается лишь проверить, что в коллекции не осталось элементов – «(?(x)(?!))».

Сбалансированные определения

Пример. Пусть язык *L* описывает вложенные операторы языка Pascal «**begin end**;». Учитывая, что необходимо проверять их парность, используем ДМПА с посимвольным разбором. Операторы отделяются друг от друга разделительными символами (пробелами, табуляциями, знаками возврата каретки и перехода на новую строку) в произвольном количестве, но не менее одного (в таблице обозначены символом подчеркивания). Также пробелы могут окружать знак «;».

Сбалансированные определения

Получим следующее РВ:

```
@"^\s*((?'begin'begin\s+)+
(?'-begin'end\s*;\s*)+)*(?(begin)(?!))$"
```

Определение позиции ошибки

Структура РВ должна быть такой, чтобы не просто проверить соответствие шаблона входной цепочке, но найти максимальное количество соответствий, пусть даже частичных. Тогда позицией ошибки будет:

1) первая позиция входной цепочки (1), если первое соответствие не начинается с позиции Index = 0;

2) позиция, следующая за последним соответствием (Index + Length + 1), если она не совпадает с последней позицией входной цепочки;

3) позиция первого разрыва между соответствиями, если символ, следующий за предыдущим соответствием, не является первым символом следующего соответствия;

4) позиция специального захвата некорректных цепочек.

Определение позиции ошибки

Пример.

```
Regex r = new Regex(@"\w+(\.\w+)*");
string str = "abc.xyz.pqr";
MatchCollection m = r.Matches(str);
if (m.Count == 1 && m[0].Value == str) Console.WriteLine("OK");
else if (m.Count == 0) Console.WriteLine("Ошибка в позиции 1 '{0}'",
str[0]);
else
{
        int index = 0;
        for (int i = 0; i < m.Count; i++)
        {
                if (m[i].Index > index) break;
                index = m[i].Index + m[i].Length;
        }
        Console.WriteLine("Ошибка в позиции {0} '{1}'", index + 1,
str[index]);
}
```

Определение позиции ошибки

Результаты работы:

- «abc.xyz.pqr» – правильно;
- «+abc.xyz.pqr» – ошибка в позиции 1 («+»);
- «abc.xyz.pqr!» – ошибка в позиции 12 («!»);
- «abc.xyz!.pqr» – ошибка в позиции 8 («!»).
- «abc.xyz.+pqr» – ошибка в позиции 8 («.»).

Определение позиции ошибки

Исправляем:

@"\w+(\.\w+)*(\.(?!$))?"

Результаты работы:

- «abc.xyz.+pqr» – ошибка в позиции 9 («+»);
- «abc.xyz.pqr.» – ошибка в позиции 12 («.»).

Лабораторная работа №2

Порядок выполнения лабораторной работы:

1. Описать требуемый язык заданным способом (в виде РВ).
2. Написать программу, реализующую требуемый механизм синтаксического анализа.
3. Внедрить в синтаксис анализатора действия для проверки семантики языка или его интерпретации.
4. Протестировать программу.
5. Написать отчёт, включающий все требуемые пункты (в т.ч. формальное описание построенного анализатора) и удовлетворяющий требованиям ОС ТУСУР 01-2013.

Лабораторная работа №2

Требования к программе:

- В программе должно быть описано единственное РВ, полностью описывающее заданный язык.
- Для анализа РВ рекомендуется применять уже готовые классы и библиотеки (Regex и т.п.).
- Внедрение в синтаксис РВ действий и поиск позиции ошибки обеспечивается добавлением нумерованных и именованных групп:

  (…) (?<имя>…) (?'имя'…)

- Требования к входным и выходным данным – лабораторной работе №1.

Лабораторная работа №1

Способы поиска ошибок при помощи РВ:

- Построить РВ так, чтобы выделить все правильные совпадения во входной цепочке и искать разрывы между ними:

int a, 12b, c3[10], x[];

- Построить РВ так, чтобы выделить правильную часть цепочки, начиная от её начала:

int a, 12b, c3[10], x[];

Для этого используются утверждения просмотра вперёд и назад нулевой длины:

(?!...) (?<=...) (?<!...) (?>...)

Домашнее задание

Язык *L* описывает обращение к элементу массива. При этом размерность массива может быть любой, а в качестве индексов можно использовать целые константы $i \geq 0$, а также идентификаторы, в т.ч. элементы других массивов. Идентификатор начинается с латинской буквы, после которой могут следовать другие буквы и цифры. Индексы заключаются в квадратные скобки и отделяются друг от друга запятыми. Например:

- a[1];
- a2[5,b[2],z];
- mas[x[4],y[4]] и т.д.

Домашнее задание

Вариант 1:

```
const string pattern = @"^
    (
        (
            (           (?<=^|,|\[)       (?<op>[a-z]+) (?=,|\[|\])   ) |
            (           (?<=,|\[)         (?<op>\d+)    (?=,|\])       ) |
            (           (?<=[a-z]|\d|\]) (?<op>,)       (?=[a-z]|\d) ) |
            (?<level>   (?<=[a-z])        (?<op>\[)     (?=[a-z]|\d) ) |
            (?<-level>  (?<=[a-z]|\d|\]) (?<op>\])     (?=,|\]|$)   )
        )+
        (?(level)(?!))
    )$";
Regex r = new Regex(pattern, RegexOptions.IgnorePatternWhitespace);
```

Домашнее задание

Вариант 1:

```
abc[12,zz[10],tt,zx[zy[5],q]]
Соответствие 1
    Группа 1 = 'abc[12,zz[10],tt,zx[zy[5],q]]'
        Захват 0 = 'abc[12,zz[10],tt,zx[zy[5],q]]', позиция =
0, длина = 29
    Группа 2 = ']'
        Захват 0 = 'abc', позиция = 0, длина = 3
        Захват 1 = '[', позиция = 3, длина = 1
        Захват 2 = '12', позиция = 4, длина = 2
        Захват 3 = ',', позиция = 6, длина = 1
        Захват 4 = 'zz', позиция = 7, длина = 2
        Захват 5 = '[', позиция = 9, длина = 1
        Захват 6 = '10', позиция = 10, длина = 2
        Захват 7 = ']', позиция = 12, длина = 1
        Захват 8 = ',', позиция = 13, длина = 1
        Захват 9 = 'tt', позиция = 14, длина = 2
        Захват 10 = ',', позиция = 16, длина = 1
        Захват 11 = 'zx', позиция = 17, длина = 2
        Захват 12 = '[', позиция = 19, длина = 1
        Захват 13 = 'zy', позиция = 20, длина = 2
        Захват 14 = '[', позиция = 22, длина = 1
        Захват 15 = '5', позиция = 23, длина = 1
        Захват 16 = ']', позиция = 24, длина = 1
        Захват 17 = ',', позиция = 25, длина = 1
        Захват 18 = 'q', позиция = 26, длина = 1
        Захват 19 = ']', позиция = 27, длина = 1
        Захват 20 = ']', позиция = 28, длина = 1
    Группа 3 = 'q'
        Захват 0 = 'abc', позиция = 0, длина = 3
        Захват 1 = 'zz', позиция = 7, длина = 2
        Захват 2 = 'tt', позиция = 14, длина = 2
        Захват 3 = 'zx', позиция = 17, длина = 2
        Захват 4 = 'zy', позиция = 20, длина = 2
        Захват 5 = 'q', позиция = 26, длина = 1
Группа 4 = '5'
        Захват 0 = '12', позиция = 4, длина = 2
        Захват 1 = '10', позиция = 10, длина = 2
        Захват 2 = '5', позиция = 23, длина = 1
    Группа 5 = ','
        Захват 0 = ',', позиция = 6, длина = 1
        Захват 1 = ',', позиция = 13, длина = 1
        Захват 2 = ',', позиция = 16, длина = 1
        Захват 3 = ',', позиция = 25, длина = 1
    Группа 6 = ']'
        Захват 0 = 'abc', позиция = 0, длина = 3
        Захват 1 = '[', позиция = 3, длина = 1
        Захват 2 = '12', позиция = 4, длина = 2
        Захват 3 = ',', позиция = 6, длина = 1
        Захват 4 = 'zz', позиция = 7, длина = 2
        Захват 5 = '[', позиция = 9, длина = 1
        Захват 6 = '10', позиция = 10, длина = 2
        Захват 7 = ']', позиция = 12, длина = 1
        Захват 8 = ',', позиция = 13, длина = 1
        Захват 9 = 'tt', позиция = 14, длина = 2
        Захват 10 = ',', позиция = 16, длина = 1
        Захват 11 = 'zx', позиция = 17, длина = 2
        Захват 12 = '[', позиция = 19, длина = 1
        Захват 13 = 'zy', позиция = 20, длина = 2
        Захват 14 = '[', позиция = 22, длина = 1
        Захват 15 = '5', позиция = 23, длина = 1
        Захват 16 = ']', позиция = 24, длина = 1
        Захват 17 = ',', позиция = 25, длина = 1
        Захват 18 = 'q', позиция = 26, длина = 1
        Захват 19 = ']', позиция = 27, длина = 1
        Захват 20 = ']', позиция = 28, длина = 1
    Группа 7 = ''
```

Домашнее задание

Вариант 2:

```
const string pattern = @"^
    (?:
        (?:
            (?:         (?<=^|,|\[)       (?<op>[a-z]+) (?=,|\[|\])  ) |
            (?:         (?<=,|\[)         (?<op>\d+)    (?=,|\])      ) |
            (?:         (?<=[a-z]|\d|\]) (?<op>,)       (?=[a-z]|\d) ) |
            (?<level>   (?<=[a-z])        (?<op>\[)     (?=[a-z]|\d) ) |
            (?<-level> (?<=[a-z]|\d|\]) (?<op>\])     (?=,|\]|$)   )
        )+
        (?(level)(?!))
    )$";
Regex r = new Regex(pattern, RegexOptions.IgnorePatternWhitespace);
```

Домашнее задание

Вариант 2:

```
abc[12,zz[10],tt,zx[zy[5],q]]
Соответствие 1
    Группа 1 = ']'
        Захват 0 = 'abc', позиция = 0, длина = 3
        Захват 1 = '[', позиция = 3, длина = 1
        Захват 2 = '12', позиция = 4, длина = 2
        Захват 3 = ',', позиция = 6, длина = 1
        Захват 4 = 'zz', позиция = 7, длина = 2
        Захват 5 = '[', позиция = 9, длина = 1
        Захват 6 = '10', позиция = 10, длина = 2
        Захват 7 = ']', позиция = 12, длина = 1
        Захват 8 = ',', позиция = 13, длина = 1
        Захват 9 = 'tt', позиция = 14, длина = 2
        Захват 10 = ',', позиция = 16, длина = 1
        Захват 11 = 'zx', позиция = 17, длина = 2
        Захват 12 = '[', позиция = 19, длина = 1
        Захват 13 = 'zy', позиция = 20, длина = 2
        Захват 14 = '[', позиция = 22, длина = 1
        Захват 15 = '5', позиция = 23, длина = 1
        Захват 16 = ']', позиция = 24, длина = 1
        Захват 17 = ',', позиция = 25, длина = 1
        Захват 18 = 'q', позиция = 26, длина = 1
        Захват 19 = ']', позиция = 27, длина = 1
        Захват 20 = ']', позиция = 28, длина = 1
    Группа 2 = ''
```