Регулярный язык

Регулярный язык (регулярное множество) в теории формальных языков - множество слов, которое распознает некоторый конечный автомат.

Определение

Пусть Σ — конечный алфавит.

Регулярными языками в алфавите Σ называются множества слов, определяемые по индукции следующим образом:

- Пустое множество $\varnothing$ является регулярным языком.
- Множество, состоящее из одной лишь пустой цепочки $\{\varepsilon\}$, является регулярным языком.
- Множество, состоящее из одного однобуквенного слова $\{a\}$, где $a \in \Sigma$, является регулярным языком.
- Если α и β — регулярные языки, то их объединение $\alpha \cup \beta$, конкатенация $\alpha\beta$ и итерация α^* тоже являются регулярными языком.
- Других регулярных языков нет.

Регулярные выражения

Регулярные выражения – обозначение для регулярных множеств.

Например, при помощи регулярных выражений можно задать шаблоны, позволяющие:

- найти все последовательности символов <кот> в любом контексте, как то: <кот>, <котлета>, <терракотовый>;
- найти отдельно стоящее слово <кот> и заменить его на <кошка>;
- найти слово <кот>, которому предшествует слово <персидский> или <чеширский>;
- убрать из текста все предложения, в которых упоминается слово кот или кошка.

Регулярные выражения позволяют задавать и гораздо более сложные шаблоны поиска или замены. Результатом работы с регулярным выражением может быть:

- проверка наличия искомого образца в заданном тексте;
- определение подстроки текста, которая сопоставляется образцу;
- определение групп символов, соответствующих отдельным частям образца.

Регулярные выражения, состоящие из букв алфавита и пустой цепочки, создаются с помощью операций:

- (*сцепления или конкатенации регулярных множеств* R *и* S) - RS обозначает множество {αβ | α ∈ *R*, β ∈ *S*}, например: {"boy", "girl"}{"friend", "cott"} = {"boyfriend", "girlfriend", "boycott", "girlcott"};
- (*объединения регулярных множеств* R *и* S) - *R*|*S* обозначает объединение *R* и *S*, например: {"ab", "c"}|{"ab", "d", "ef"} = {"ab", "c", "d", "ef"}[4];
- (*итерации регулярного множества* R) - *R** обозначает минимальное надмножество множества *R*, которое содержит ε и замкнуто относительно конкатенации (это есть множество всех строк, полученных конкатенацией нуля или более строк из *R*, например: {"Run", "Forrest"}* = {ε, "Run", "Forrest", "RunRun", "RunForrest", "ForrestRun", "ForrestForrest", "RunRunRun", "RunRunForrest", "RunForrestRun", …})[

Синтаксис

Обычные символы (литералы) и специальные символы (метасимволы)

Большинство символов в регулярном выражении представляют сами себя за исключением специальных символов `[` `]` `\` `/` `^` `$` `.` `|` `?` `*` `+` `(` `)` `{` `}` (в разных типах регулярных выражений этот набор различается), которые могут быть экранированы символом `\` (обратная косая черта) для представления самих себя в качестве символов текста. Можно экранировать целую последовательность символов, заключив её между `\Q` и `\E`.

Пример	Соответствие
`a\.?`	`a.` или `a`
`a\\\\b`	`a\\b`
`a\[F\]`	`a[F]`
`\Q+-*/\E`	`+-*/`

Аналогично могут быть представлены другие специальные символы (набор символов, требующих «экранирования», может отличаться в зависимости от конкретной программной реализации). Часть символов, которые в той или иной реализации не требуют «экранирования» (например, угловые скобки `<` `>`), могут быть экранированы из соображений удобочитаемости.

Любой символ

Метасимвол `.` (точка) означает один любой символ, но в некоторых реализациях исключая символ новой строки.

Вместо символа `.` можно использовать `[\s\S]` (все пробельные и непробельные символы, включая символ новой строки).

Символьные классы

Набор символов в квадратных скобках `[ ]` именуется символьным классом. Позволяет указать интерпретатору регулярных выражений, что на данном месте в строке может стоять один из перечисленных символов. В частности, `[абв]` задаёт возможность появления в тексте одного из трёх указанных символов, а `[1234567890]` задаёт соответствие одной из цифр. Возможно указание диапазонов символов: например, `[А-Яа-я]` соответствует всем буквам русского алфавита, за исключением букв «Ё» и «ё».

Если требуется указать символы, которые не входят в указанный набор, то используют символ `^` внутри квадратных скобок, например `[^0-9]` означает любой символ, кроме цифр.

Добавление в набор специальных символов путём экранирования — самый бесхитростный способ. Однако в современных регулярных выражениях унаследован также и традиционный подход — см. Традиционные регулярные выражения.

Некоторые символьные классы можно заменить специальными метасимволами:

Символ	Эквивалент	Соответствие
`\d`	`[0-9]`	Цифровой символ
`\D`	`[^0-9]`	Нецифровой символ
`\s`	`[ \f\n\r\t\v]`	Пробельный символ
`\S`	`[^ \f\n\r\t\v]`	Непробельный символ
`\w`	`[[:word:]]`	Буквенный или цифровой символ или знак подчёркивания
`\W`	`[^[:word:]]`	Любой символ, кроме буквенного или цифрового символа или знака подчёркивания

Позиция внутри строки

Следующие символы позволяют позиционировать регулярное выражение относительно элементов текста: начала и конца строки, границ слова.

Представление	Позиция	Пример	Соответствие
^	Начало текста (или строки при модификаторе ?m)	^a	aaa aaa
$	Конец текста (или строки при модификаторе ?m)	a$	aaa aaa
\b	Граница слова	a\b	aaa aaa
		\ba	aaa aaa
\B	Не граница слова	\Ba\B	aaa aaa
\G	Предыдущий успешный поиск	\Ga	aaa aaa (поиск остановился на 4-й позиции — там, где не нашлось a)

Специальные символы

\n — перевод строки

\r — возврат каретки

Обозначение группы

Круглые скобки используются для определения области действия и приоритета операций. Шаблон внутри группы обрабатывается как единое целое и может быть квантифицирован. Например, выражение (тр[ау]м-?)* найдёт последовательность вида трам-трам-трумтрам-трум-трамтрум.

Перечисление

Вертикальная черта разделяет допустимые варианты. Например, gray|grey соответствует gray или grey. Следует помнить, что перебор вариантов выполняется слева направо, как они указаны.

Если требуется указать перечень вариантов внутри более сложного регулярного выражения, то его нужно заключить в группу. Например, `gray|grey` или `gr(a|e)y` описывают строку `gray` или `grey`. В случае с односимвольными альтернативами предпочтителен вариант `gr[ae]y`, так как сравнение с символьным классом выполняется проще, чем обработка группы с проверкой на все её возможные модификаторы и генерацией обратной связи.

Квантификация (поиск последовательностей)

Квантификатор после символа, символьного класса или группы определяет, сколько раз предшествующее выражение может встречаться. Следует учитывать, что квантификатор может относиться более чем к одному символу в регулярном выражении, только если это символьный класс или группа.

Представление	Число повторений	Эквивалент	Пример	Соответствие
?	Ноль или одно	{0,1}	colou?r	color, colour
*	Ноль или более	{0,}	colou*r	color, colour, colouur

Представление	Число повторений	Пример	
{*n*}	Ровно *n* раз	colou{3}r	co
{*m*,*n*}	От *m* до *n* включительно	colou{2,4}r	co
{*m*,}	Не менее *m*	colou{2,}r	co и
{,*n*}	Не более *n*	colou{,3}r	co co

				и т. д.
`+`	Одно или более	`{1,}`	`colo u+r`	`colour`, `colouur` и т. д. (но не `color`)

Часто используется последовательность `.*` для обозначения любого количества любых символов между двумя частями регулярного выражения.

Символьные классы в сочетании с квантификаторами позволяют устанавливать соответствия с реальными текстами. Например, столбцами цифр, телефонами, почтовыми адресами, элементами HTML-разметки и др.

Если символы `{` `}` не образуют квантификатора, их специальное значение игнорируется.

Жадная и ленивая квантификация

Пример использования жадных и ленивых выражений

Выражение `(<.*>)` соответствует строке, содержащей несколько тегов HTML-разметки, целиком.

```
<p><b>Википедия</b> — свободная энциклопедия, в которой <i>каждый</i> может изменить или дополнить любую статью.</p>
```

Чтобы выделить отдельные теги, можно применить ленивую версию этого выражения: `(<.*?>)` Ей соответствует не вся показанная выше строка, а отдельные теги (выделены цветом):

```
<p><b>Википедия</b> — свободная энциклопедия, в которой <i>каждый</i> может изменить или дополнить любую статью.</p>
```

В некоторых реализациях квантификаторам в регулярных выражениях соответствует максимально длинная строка из возможных (квантификаторы являются *жадными*, англ. *greedy*). Это может оказаться значительной проблемой. Например, часто ожидают, что выражение `(<.*>)` найдёт в тексте теги HTML. Однако если в тексте есть более одного HTML-тега, то этому выражению соответствует целиком строка, содержащая множество тегов.

```
<p><b>Википедия</b> — свободная энциклопедия, в которой <i>каждый</i> может изменить или дополнить любую статью.</p>
```

Эту проблему можно решить двумя способами.

1. Учитывать символы, *не* соответствующие желаемому образцу (`<[^>]*>` для вышеописанного случая).

2. Определить квантификатор как *нежадный* (*ленивый*, англ. *lazy*) — большинство реализаций позволяют это сделать, добавив после него знак вопроса.

Использование ленивых квантификаторов может повлечь за собой обратную проблему, когда выражению соответствует слишком короткая, в частности, пустая строка.

Жадный	Ленивый
`*`	`*?`
`+`	`+?`
`{n,}`	`{n,}?`

Также общей проблемой как жадных, так и ленивых выражений являются точки возврата для перебора вариантов выражения. Точки ставятся после каждой итерации квантификатора. Если интерпретатор не нашёл соответствия после квантификатора, то он начинает возвращаться по всем установленным точкам, пересчитывая оттуда выражение по-другому.

Ревнивая квантификация (сверхжадная)

При поиске выражения `(a+a+)+a` в строке `aaaaa` интерпретатор пойдёт приблизительно по следующему пути:

1. `aaaaa`
2. `aaaa`
3. `aaaaa`
4. `aaa`
5. `aaaaa`
6. `aaaa`
7. `aaaaa` — и только тут, проверив все точки возврата, остановится.

При использовании ревнивого квантификатора будет выполнен только первый шаг алгоритма.

В отличие от обычной (жадной) квантификации, ревнивая (possessive) квантификация не только старается найти максимально длинный вариант, но ещё и не позволяет алгоритму возвращаться к предыдущим шагам поиска для того, чтобы найти возможные соответствия для оставшейся части регулярного выражения.

Использование ревнивых квантификаторов увеличивает скорость поиска, особенно в тех случаях, когда строка не соответствует регулярному выражению. Кроме того, ревнивые квантификаторы могут быть использованы для исключения нежелательных совпадений.

Жадный	Ревнивый
`*`	`*+`
`?`	`?+`
`+`	`++`
`{n,}`	`{n,}+`

Пример	Соответствие
`ab(xa)*+a`	`abxaabxaa`; но не `abxaabxaa`, так как буква **a** уже занята

Это аналогично атомарной группировке.

Группировка. Обратная связь

Одно из применений группировки — повторное использование ранее найденных групп символов (*подстрок*, *блоков*, *отмеченных подвыражений*, *захватов*). При обработке выражения подстро́ки, найденные по шаблону внутри группы, сохраняются в отдельной области памяти и получают номер, начиная с единицы. Каждой подстроке соответствует пара скобок в регулярном выражении. Квантификация группы не влияет на сохранённый результат, то есть, сохраняется лишь первое вхождение. Обычно поддерживается до 9 нумерованных подстрок с номерами от 1 до 9, но некоторые интерпретаторы позволяют работать с бо́льшим количеством. Впоследствии в пределах данного регулярного выражения можно использовать обозначения от `\1` до `\9` для проверки на совпадение с ранее найденной подстрокой.

Например, регулярное выражение `(та|ту)-\1` найдёт строку `та-та` или `ту-ту`, но пропустит строку `та-ту`.

Также ранее найденные подстро́ки можно использовать при замене по регулярному выражению. В таком случае в замещающий текст вставляются те же обозначения, что и в пределах самого выражения.

Группировка без обратной связи

Если группа используется только для группировки и её результат в дальнейшем не потребуется, то можно использовать группировку вида `(?:`*`шаблон`*`)`. Под результат такой группировки не выделяется отдельная область памяти и, соответственно, ей не

назначается номер. Это положительно влияет на скорость выполнения выражения, но понижает удобочитаемость.

Атомарная группировка

Атомарная группировка вида (?>*шаблон*) также, как и группировка без обратной связи, не создаёт обратных связей. В отличие от неё, такая группировка запрещает возвращаться назад по строке, если часть шаблона уже найдена.

Пример	Соответствие	Создаваемые группы
a(bc|b|x)cc	abccaxcc abccaxcc	abccaxcc abccaxcc
a(?:bc|b|x)cc	abccaxcc, abccaxcc	нет
a(?>bc|b|x)cc	abccaxcc но не abccaxcc: вариант x найден, остальные проигнорированы	нет
a(?>x*)xa	не найдётся axxxa: все x заняты, и нет возврата внутрь группы	

Атомарная группировка выполняется ещё быстрее, чем группировка без обратной связи, и сохраняет процессорное время при выполнении остального выражения, так как запрещает проверку любых других вариантов внутри группы, когда один вариант уже найден. Это очень полезно при оптимизации групп со множеством различных вариантов.

Это аналогично ревнивой квантификации.

Модификаторы

Модификаторы действуют с момента вхождения и до конца регулярного выражения или противоположного модификатора. Некоторые интерпретаторы могут применить модификатор ко всему выражению, а не с момента его вхождения.

Синтаксис	Описание	
(?i)	Включает	нечувствительность выражения к регистру символов (англ. *case insensitivity*)
(?-i)	Выключает	

(?s)	Включает	режим соответствия точки символам переноса строки и возврата каретки
(?-s)	Выключает	
(?m)	Символы ^ и $ вызывают соответствие только	после и до символов новой строки
(?-m)		с началом и концом текста
(?x)	Включает	режим без учёта пробелов между частями регулярного выражения и позволяет использовать # для комментариев
(?-x)	Выключает	

Группы-модификаторы можно объединять в одну группу: (?i-sm). Такая группа включает режим i и выключает режим s, m. Если использование модификаторов требуется только в пределах группы, то нужный шаблон указывается внутри группы после модификаторов но перед двоеточием. Например, (?-i)(?i:tv)set найдёт TVset, но не TVSET.

Комментарии

Для добавления комментариев в регулярное выражение можно использовать группы-комментарии вида (?#*комментарий*). Такая группа интерпретатором полностью игнорируется и не проверяется на вхождение в текст. Например, выражение А(?#тут комментарий)Б соответствует строке АБ.

Просмотр вперёд и назад

В большинстве реализаций регулярных выражений есть способ производить поиск фрагмента текста, «просматривая» (но не включая в найденное) окружающий текст, который расположен до или после искомого фрагмента текста. Просмотр с отрицанием используется реже и «следит» за тем, чтобы указанные соответствия, напротив, не встречались до или после искомого текстового фрагмента.

Представление	Вид просмотра	Пример	Соответствие
(?=*шаблон*)	Позитивный просмотр вперёд	Людовик(?=XVI)	ЛюдовикXV, ЛюдовикXVI, ЛюдовикXVIII, ЛюдовикLXVII, ЛюдовикXXL

(?!*шаблон*)	Негативный просмотр вперёд (с отрицанием)	Людовик(?!XVI)	ЛюдовикXV, ЛюдовикXVI, ЛюдовикXVIII, ЛюдовикLXVII, ЛюдовикXXL
(?<=*шаблон*)	Позитивный просмотр назад	(?<=Сергей)Иванов	Сергей Иванов, Игорь Иванов
(?<!*шаблон*)	Негативный просмотр назад (с отрицанием)	(?<!Сергей)Иванов	Сергей Иванов, Игорь Иванов

Поиск по условию

Во многих реализациях регулярных выражений существует возможность выбирать, по какому пути пойдёт проверка в том или ином месте регулярного выражения на основании уже найденных значений.

Представление	Пояснение	Пример	Соответствие
(?:(?=*если*)*то*|*иначе*)	Если операция просмотра успешна, то далее выполняется часть *то*, иначе выполняется часть *иначе*. В выражении может использоваться любая из четырёх операций просмотра. Следует учитывать, что операция просмотра нулевой ширины, поэтому части *то* в случае позитивного или *иначе* в случае негативного просмотра должны включать в себя описание шаблона из операции просмотра.	(?:(?<=а)м|п)	мам,пап
(?(*n*)*то*|*иначе*)	Если *n*-я группа вернула значение, то поиск по условию выполняется по шаблону *то*, иначе по шаблону *иначе*.	(а)?(?(1)м|п)	мам,пап

Флаги

В некоторых языках (например, в JavaScript) реализованы т. н. «флаги», которые расширяют функции регэкспа. Флаги указываются после регулярного выражения (порядок флагов значения не имеет). Типичные флаги:

- **g** — глобальный поиск (обрабатываются все совпадения с шаблоном поиска);
- **i** — регистр букв не имеет значения;
- **m** — многострочный поиск;
- **s** — текст трактуется как одна строка, в этом случае метасимволу . (точка) соответствует любой одиночный символ, включая символ новой строки;
- **u** — unicode трактовка. Выражение может содержать специальные паттерны, характерные для уникода, `/\p{Lu}/ - заглавные буквы` например.

Флаг указывается после паттерна, например, вот так: `/[0-9]$/m`.

Разновидности регулярных выражений

Базовые регулярные выражения POSIX

(англ. *basic regular expressions* (BRE)). Традиционные регулярные выражения UNIX. Синтаксис базовых регулярных выражений на данный момент определён POSIX'ом как устаревший, но он до сих пор широко распространён из соображений обратной совместимости. Многие UNIX-утилиты используют такие регулярные выражения по умолчанию.

В данную версию включены метасимволы:

- `.`;
- `[ ]`;
- `[^ ]`;
- `^` (действует только в начале выражения);
- `$` (действует только в конце выражения);
- `*`;
- `\{ \}` — первоначальный вариант для `{ }`;
- `\( \)` — первоначальный вариант для `( )`;
- `\n`, где *n* — номер от 1 до 9.

Особенности:

- Звёздочка должна следовать после выражения, соответствующего единичному символу. Пример: `[xyz]*`.
- Выражение `\(блок\)*` следует считать неправильным. В некоторых случаях оно соответствует нулю или более повторений строки `блок`. В других оно соответствует строке `блок*`.
- Внутри символьного класса специальные значения символов, в основном, игнорируются. Особые случаи:

- Чтобы добавить символ `^` в набор, его следует поместить туда не первым.
- Чтобы добавить символ `-` в набор, его следует поместить туда первым или последним. Например:
  - шаблон DNS-имени, куда могут входить буквы, цифры, минус и точка-разделитель: `[-0-9a-zA-Z.]`;
  - любой символ, кроме минуса и цифры: `[^-0-9]`.
- Чтобы добавить символ `[` или `]` в набор, его следует поместить туда первым. Например:
  - `[][ab]` соответствует `]`, `[`, `a` или `b`.

Расширенные регулярные выражения POSIX

- Отменено использование обратной косой черты для метасимволов `{}` и `()`.
- Обратная косая черта перед метасимволом отменяет его специальное значение (см. Представление специальных символов).
- Отвергнута теоретически **нерегулярная** конструкция `\n`.
- Добавлены метасимволы `+`, `?`, `|`.

См. также: Символьные классы POSIX

Регулярные выражения, совместимые с Perl

Perl-совместимые регулярные выражения (англ. *Perl-compatible regular expressions* (PCRE)) имеют более богатый синтаксис, чем даже POSIX ERE. По этой причине очень многие приложения используют именно Perl-совместимый синтаксис регулярных выражений.

Регулярные выражения, совместимые с Unicode

Unicode — это набор символов, целью которого является определение всех символов и символов со всех человеческих языков, живых и мертвых. Регулярные выражения, рассчитанные на множество языков, таким образом не привязываются к конкретным наборам символов, а описывают их согласно принятых правил. Так, например выражение для нахождения заглавных букв в любом алфавите будет выглядеть так: `/\p{Lu}/`.

Некоторые выражения regexp — unicode:		
представление		**функциональность**
краткая форма	**полная форма**	
Буквы		

\p{L}	\p{Letter}	любые буквы любого языка
\p{Ll}	\p{Lowercase_Letter}	буквы нижнего регистра (строчные) из тех, что имеют прописной вариант написания
\p{Lu}	\p{Uppercase_Letter}	буквы верхнего регистра (прописные) для тех, что имеют строчный вариант написания
\p{Lt}	\p{Titlecase_Letter}	прописная буква, которая появляется с начала слова из строчных букв
\p{L&}	\p{Cased_Letter}	буква, которая имеет как прописной, так и строчный варианты написания
\p{Lm}	\p{Modifier_Letter}	специальные символы, которые используются как буквы
\p{Lo}	\p{Other_Letter}	символ или идеограмма, которая не имеет прописных и строчных вариантов написания
Специальные символы		
\p{M}	\p{Mark}	символы, вставленные для комбинирования с другими символами (например акценты, умляуты, оборачивающие скобки)
\p{Mn}	\p{Non_Spacing_Mark}	символ, вставленный для комбинирования с другими символами, не занимая дополнительной ширины
\p{Mc}	\p{Spacing_Combining_Mark}	символы, вставленные для комбинирования с другими символами, занимая дополнительную ширину (как во многих восточных языках)
\p{Me}	\p{Enclosing_Mark}	символы, которые оборачивают символ. Например круг, квадрат и т.п

Пробелы и разделители		
\p{Z}	\p{Separator}	любые виды пробелов или невидимых разделителей
\p{Zs}	\p{Space_Separator}	пробельные символы, которые невидимы, но имеют ширину
\p{Zl}	\p{Line_Separator}	символ разделения в виде линии U+2028
\p{Zp}	\p{Paragraph_Separator}	знак параграфа U+2029
Математические символы		
\p{S}	\p{Symbol}	математические символы, символы валюты, символы псевдографики (рамки) и т. п.
\p{Sm}	\p{Math_Symbol}	любые математические символы
\p{Sc}	\p{Currency_Symbol}	любые символы валют
\p{Sk}	\p{Modifier_Symbol}	комбинированный символ (пометка) как комбинация самого символа и символа отметки
\p{So}	\p{Other_Symbol}	различные символы, не математические, не символы валют или их комбинации
Цифровые символы		
\p{N}	\p{Number}	любые виды цифровых символов в любых языках
\p{Nd}	\p{Decimal_Digit_Number}	цифры от нуля до девятки в любых языках

\p{Nl}	\p{Letter_Number}	число, которое может выглядеть как буквы, например как римские числа
\p{No}	\p{Other_Number}	число, представленное как верхний или нижний индекс, или число, которое не состоит из цифр (исключая числа из идеографических письменностей)
Знаки пунктуации		
\p{P}	\p{Punctuation}	любой вид пунктуационных знаков
\p{Pd}	\p{Dash_Punctuation}	любой вид дефисов или тире
\p{Ps}	\p{Open_Punctuation}	любой вид открывающих скобок
\p{Pe}	\p{Close_Punctuation}	любой вид закрывающих скобок
\p{Pi}	\p{Initial_Punctuation}	любой вид открывающих кавычек
\p{Pf}	\p{Final_Punctuation}	любой вид закрывающих кавычек
\p{Pc}	\p{Connector_Punctuation}	пунктуационные символы, такие как знаки подчёркивания или соединения слов
\p{Po}	\p{Other_Punctuation}	любые виды пунктуационных символов, что не являются точками, скобками, кавычками или соединителями
Управляющие символы		
\p{C}	\p{Other}	невидимые управляющие символы и неиспользуемые позиции

\p{Cc}	\p{Control}	ASCII или Latin-1 управляющие символы: 0x00-0x1F и 0x7F-0x9F
\p{Cf}	\p{Format}	невидимые индикаторы форматирования
\p{Co}	\p{Private_Use}	любые позиции, зарезервированные для личного использования
\p{Cs}	\p{Surrogate}	половина суррогатных пар в кодировке UTF-16
\p{Cn}	\p{Unassigned}	любые позиции, у которых не назначены символы