А. В. Ковалев, *студент кафедры системного моделирования и инженерной графики, ФГБОУ ВО «Московский авиационный институт (национальный исследовательский университет)», г. Москва, science@blockset.ru*

П. П. Кейно, *преподаватель кафедры системного моделирования и инженерной графики, ФГБОУ ВО «Московский авиационный институт (национальный исследовательский университет)», г. Москва, science@blockset.ru*

Вычисление мощности множества регулярного выражения как критерия оптимальности в задачах динамической маршрутизации *web*-адресов

Статья рассматривает задачу вычисления мощности множества регулярного выражения с целью использования данной характеристики в качестве критерия оптимальности при принятии решения в выборе локации (динамической страницы). Предлагается методика роутинга адресов URL с динамическими страницами внутри инструментария разработки *web*-узлов. Каждая локация идентифицируется регулярным выражением. В случае, когда под запрошенный адрес подпадает сразу несколько локаций, выбор происходит на основе искомого критерия мощности множества. Наименьшая мощность множества укажет на наиболее вероятный шаблон.

Ключевые слова: регулярное выражение, регулярный язык, регулярное множество, мощность множества, конечный автомат, детерминизация, роутинг, URL, формальный язык.

Введение

Задача поиска мощности множества регулярного выражения носит как научный, так и прикладной характер и связана с такими научными областями, как формальные языки и конечные автоматы. Основоположником регулярных языков по праву считается американский математик Стивен Коул Клини [1], разработавший теоретическую базу регулярных языков и внесший большой вклад в теорию конечных автоматов. Первое практическое внедрение в информационные технологии регулярных выражений произвел пионер компьютерной науки, американский ученый Кен Томпсон [2]. Вопрос поиска мощности множества (или, в терминах авторов, «размера») регулярного выражения широко рассматривался Германом Грубером и Маркусом Хольцером [3].

На данный момент нет точного алгоритма, позволяющего вычислить все возможные комбинации регулярного выражения. Перед нами стоит задача разработать алгоритм, способный посчитать приблизительную мощность множества с возможными ограничениями, удовлетворяющими условию практической задачи.

В практической составляющей задачи имеется сущность, уникальным идентификатором которой служит шаблон — регулярное выражение. На вход подается специальная строка, сопоставляемая с каждым шаблоном для идентификации объекта. Сущность представляет собой динамическую *web*-страницу, а входная строка — пользова-

тельский адрес URL. Задача рассматривает случай, когда входная строка подпадает под несколько шаблонов и сводится к разрешению данного противоречия. При возникновении противоречий между шаблонами системе необходимо проставить приоритеты каждого из них, чтобы выбрать наиболее подходящий. Таким приоритетом будет служить некий коэффициент, характеризующий мощность множества каждого из шаблонов. Под мощностью множества регулярного выражения в настоящей статье понимается результат подсчета всех возможных комбинаций в заданном шаблоне.

Для решения задачи требуется классифицировать теоретическую базу, необходимую для исследования. Для начала рассмотрим основные представления одного из вариантов класса формальных языков — *регулярные языки*.

Они могут быть выражены регулярными множествами (выражениями), а также конечными автоматами.

Регулярные множества

Для каждого множества можно найти *регулярное выражение*, обозначающее это множество. *Регулярное выражение* — формальный язык, служащий для поиска и осуществления манипуляций с подстроками в тексте.

Примером *регулярного выражения* (множества) могут служить следующие выражения:

$$ab,\ ab^*,\ a|b,\ (ab|c)^*.$$

Последнему выражению соответствуют следующие множества:

$$(ab|c)^* = ab \mid abc \mid abab \mid c \mid cc \mid cab \ \ldots$$

и т. д.

Согласно классификации Хомского [4] регулярные языки относятся к третьему типу грамматик, а значит, по теореме Клини [5] могут быть выражены с помощью конечных автоматов.

Конечные автоматы

Для распознавания регулярных множеств служат *конечные автоматы* — наиболее известный подход к реализации сопоставления с регулярными выражениями. По регулярному выражению строится конечный автомат, в котором одно или несколько состояний объявлены *начальными*, одно или несколько состояний — *терминальными* (конечными) и некоторые состояния соединены дугами (рис. 1).

Такой автомат называется *недетерминированным*, так как в каждый момент могут быть активны сразу несколько состояний. Он дает возможность понять, сколько можно перебрать множеств, удовлетворяющих заданному регулярному выражению.

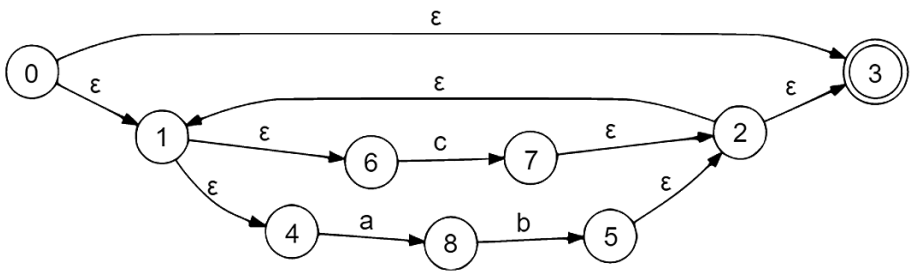

Рис. 1. Пример недетерминированного конечного автомата
Fig. 1. Non-deterministic finite automaton example

Для упрощения и ускорения сопоставлений из автомата убирают **ε**-дуги, обозначающие немедленный переход из одного состояния в другое. Процесс удаления таких дуг называется *детерминизацией*, а сам автомат — *детерминированным* (рис. 2).

Недостаток детерминированного конечного автомата — увеличение размеров, состояний и переходов, влияющих на конечный перебор всех его возможных вариантов.

Конечные автоматы очень хороши при обнаружении шаблонов и могут подсчитать количество символов, но лишь до заранее известного предела. Они не обладают памятью и не способны сравнивать одну часть с другой. Например, конечный автомат в состоянии определить, что во входном слове по крайней мере 5 символов «*C*», но не может сравнивать количества символов «*A*» и «*B*», поскольку не имеет счетчика. Ключевой фактор этой особенности — при работе с автоматом мы заранее знаем нужное нам число символов «*C*» [6]. Иначе говоря, конечные автоматы описывают поведение регулярного языка в соответствии с заданным шаблоном, но не более того.

Задача

Вариантов применения регулярных выражений, по сути, бесчисленное множество, в нашем же случае они используются для роутинга (маршрутизации) динамических *web*-страниц пользователя. Прежде чем говорить о самой задаче, скажем, что на данный момент по роутингу URL существует большое количество различных решений [7–9]. В представленной работе рассматривается вариант максимальной автоматизации задачи на основе критерия оптимальности.

Критерием оптимальности нашего регулярного выражения служит его мощность множества как результат подсчета всех возможных комбинаций в заданном шаблоне. Требуется построить математический аппарат для нахождения этой мощности.

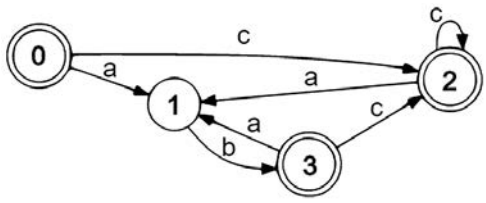

Рис. 2. Детерминированный конечный автомат для выражения (*ab*|*c*)*

Fig. 2. Deterministic finite automaton for (*ab*|*c*)* expression

Рассмотрим следующие аспекты его построения:

- упрощение регулярного выражения;
- алгоритм вычисления мощности множества;
- алгоритм вычисления мощности в пересекаемых множествах.

Упрощение регулярного выражения

Будем использовать следующие правила упрощения исходного регулярного выражения:

- *условие избыточности*. Если мы имеем некоторое множество с алфавитом $L = \{x, y, xy \ldots\}$, определенное квантификатором «+» (от одного и более), то мы можем сделать эквивалентную замену:

$$x+ = xx*$$

$$(xy) + = xy\,(xy)*;$$

- *условие конечности множества*. Если в регулярном выражении имеется квантификатор «*» (от **ε** до ∞), то количество комбинаций, подходящих заданному регулярному выражению, неограниченно большое. Для устранения неопределенности такого рода используем замену, ограничивающую максимальную длину строки:

$$xx* = x\ \{1,50\}\ (\text{от 1 до 50}).$$

Также имеют смысл следующие эквивалентные замены:

$$(x^*)^* = x^*$$

$$x|x = x$$

$$(xy) \mid (yx) = (yx) \mid (xy);$$

• алгоритм вычисления мощности множества. Возьмем простой пример регулярного выражения

$$[a\text{-}z]\ \{2\} \mid [0{-}9]?\ [a\text{-}z]\ \{2, 3\} \mid \mid [a\text{-}z]\ ab\ [a\text{-}z]?$$

Разобьем данное выражение на три группы, разделяемые символом «|» и вычислим мощность каждой.

1. Группа [*a-z*] {2}

Множество [*a-z*] имеет 26 символов, квантификатор « {*2*}» говорит о том, что множество повторяется два раза. Иными словами, у нас две «*единичные позиции*» — два места: на каждом месте может быть один из 26 символов. А значит, количество возможных вариантов сочетаний равно 26·26, или 26^2, где 26 — мощность алфавита *a-z*, а степень 2 — количество повторений.

2. Группа [0–9]? [*a-z*] {2, 3}

Квантификатор «?» сообщает нам о том, что первое множество [0–9] может встретиться с набором множества [*a-z*] {2, 3}, а также может и **не** встретиться. Также множество [0–9] может встретиться как с множеством [*a-z*] {2}, так и с [*a-z*] {3}. Итого мы получаем следующий набор вариантов:

$$[0..9] * [a..z] * [a..z] * [a..z] = 10 \cdot 26^3,$$

где степень 3 квантификатор; {3}

$$[0..9] * [a..z] * [a..z] = 10 \cdot 26^2,$$

где 10 — мощность множества [0–9];

$$[a..z] * [a..z] * [a..z] = 26^3;$$

$$[a..z] * [a..z] = 26^2.$$

Вычислив мощности множеств каждого слагаемого, найдем сумму всех результатов каждого из вариантов: $10 \cdot 26^2 + 10 \cdot 26^3 + 26^3 + 26^2$.

3. Группа [*a-z*] *ab* [*a-z*]?

Мощность такого выражения равна $26^2 + 26$. Дело в том, что слово «ab» служит разделителем между двумя множествами, и его можно не учитывать, так как в перестановках слово «ab» не играет никакой роли.

Теперь о том, как распоряжаться нашими результатами от каждой группы.

• Результаты мощностей групп, разделенных символом «|», необходимо сложить.

• В случае, когда группы обособлены скобками «()», результаты мощностей таких групп необходимо перемножить.

• В случае, когда идет чередование символа «|» и отдельных групп «()», результаты складываются и перемножаются соответственно, приоритетнее операция умножения.

Исходя из представленной методики, получим следующее выражение:

$$(26^2) + (10 \cdot 26^3 + 10 \cdot 26^2 + 26^3 + 26^2) + + (26^2 + 26).$$

Но на этом решение не заканчивается. Дело в том, что мы не учли возможные пересечения множеств, т. е. повторения. В данном примере два раза повторяется множество [*a-z*] {2} — в первой и во второй группах. Мощность этого множества необходимо вычесть из результата. Таким образом, объединив все замечания, получим окончательный ответ: мощность нашего регулярного выражения равна 201 474.

Остается главный вопрос — как подобрать алгоритм, учитывающий все пересечения множеств (повторения)?

Алгоритм вычисления мощности в пересекаемых множествах

Найдем количество возможных комбинаций в следующем регулярном выражении — *A*? *B*? *A*? *B*?.

Если бы слагаемые *A* и *B* не повторялись, то решением такого выражения было бы (2·2·2·2) – 1 = 15, где 2 — количество состояний (символ может быть или не быть), вы-

читаем единицу, так как в контексте практической части задачи не может быть пустого варианта.

В нашем же примере переменные повторяются, т. е. всего 15 вариантов и минус повторения.

A*BAB* или *AB****A****B*

*A****B****AB* или *ABA****B***

AB*AB* или *AB****AB*** или ***AB****A****B*** и так далее.

Окончательный ответ $(2 \cdot 2 \cdot 2 \cdot 2) - (2 \cdot 2) - 1 = 11$.

Для более сложных выражений найти общее математическое решение по нахождению пересечений таких комбинаций не удается. Проблемой служит неизвестное поведение роста некоторой функции повторений таких комбинаций с добавлением новых слагаемых или групп чередующихся слагаемых. Например, *A*? *A*? *B*? *A*? *B*? *A*? *B*? *A*? *A*?, где группа *ABA* может встретиться больше 6 раз, каждая из которых разбивается на отдельные повторяющиеся группы.

Также хотим отметить, что такой подход трудно применим к регулярным выражениям, имеющим группы, внутри которых могут содержаться как простые разделители между группами, так и целые множества. Примеры таких выражений представлены в табл. 1 и 2. Данное ограничение стоит учесть при реализации алгоритма [10].

Заключение

Итак, подобрать общую математическую модель, учитывающую все возможные пересечения множеств в регулярном выражении, затруднительно. В таких случаях необходимо вести отдельный список всех комбинаций, чтобы понять, повторялось ли определенное множество, и, если подобный случай имел место, исключить его.

Поскольку поиск эффективного алгоритма сопряжен с поиском практического решения, допускается упрощение общего решения. Предлагается добавление функции, устанавливающей факт неопределенности регулярного выражения, что будет предупреждать пользователя о неточной или неэффективной установке шаблона, давая возможность исправить ошибку. Поскольку случаи, когда под один URL подпадает несколько шаблонов, встречаются не часто, а их пересечение происходит еще реже, данное решение имеет законченный практический вид. Таким образом, допустимы некоторые ограничения на регулярное выражение, которые вполне подходят под условия поставленной задачи.

Таблица 1. Пример групповых регулярных выражений

Table 1. Regexps grouping example

Выражение	Группы	Разделители	Множества
(тр [ау] м-?) {1,2}	(тр [ау] м-?)	– ?	[ау]
(b ([o] {2} |a) m) – ?	(b ([o] {2} | [a]) m)	– ?	[o] {2}

Таблица 2. Возможные варианты групповых регулярных выражений

Table 2. Possible combinations of grouping regexps

Выражение	Варианты
(тр [ау] м-?) {1,10}	Трам, трам-, трам-трам, трам-трам-… Трум, трум-, трум-трум-… Трам-трум-трам-…
(b ([o] {2} |a) m-?) {1,10}	Boom, bam, bam-boom, bam-boom-bam-…

Что касается решения в общем виде, то для дальнейшего продолжения исследований предлагается детальный анализ детерминированного конечного автомата. Для решения данной задачи остается метод перебора всех возможных вариантов комбинаций в детерминированном конечном автомате (ДКА). Грубым вариантом решения может быть создание механизма ДКА для перебора всех возможных комбинаций по заданному регулярному выражению, а также обеспечение высокой скорости подобного перебора, что весьма актуально при экспоненциально растущих состояниях автомата [11]. Подобное решение, безусловно, позволит вычислить точный результат, но в рамках практической задачи неэффективно. Конечно, на помощь может прийти использование механизма отложенных вычислений, но тогда данное решение может работать только совместно с основным, представленным в настоящей статье.

Также совместно с другими решениями в реализации проекта BlockSet допустимо применение предварительной детерминизации. Поскольку каждый блок имеет строго заданное регулярное выражение [12], становится возможным на этапе разработки инструментария самостоятельно установить мощности множеств регулярных выражений для каждого блока. Исключение составляет блок типа «*string*» (строка), имеющий атрибут *regexp*, с помощью которого задается пользовательское регулярное выражение (мощность которого также придется высчитать с помощью одного из представленных методов). Мощность множества регулярного выражения у каждого блока в таком случае зависит от типа данных блока и его атрибутов. Например, для блока «*number*» (число) такими атрибутами являются «*min*» и «*max*», устанавливающие допустимый числовой отрезок. Так, при *min* = «1» и *max* = «10» мощность множества будет равна 10. У блока «*float*» (число с плавающей запятой) добавляется также атрибут «*precision*» (точность), устанавливающий число знаков после запятой, что может многократно увеличивать число комбинаций. Но несмотря на то что вычисляемое значение вариативно, структура регулярного выражения достаточно детерминирована, чтобы предсказать точную мощность множества в каждом конкретном случае (кроме блока типа «*string*»). Именно это решение легло в основу методологии BlockSet в алгоритме выбора локации (динамической страницы).

Список литературы

1. *Kleene S. C.* Representation of events in nerve nets and finite automata. 1951.
2. *Thompson K.* Programming techniques: Regular expression search algorithm. Communications of the ACM. 1968. Vol. 11. No. 6. P. 419–422.
3. *Gruber H.*, *Holzer M.* Finite automata, digraph connectivity and regular expression size. In L. Aceto, I. Damgaard, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walkuwiewicz, editors, *Proceedings of the 35th International Colloqium on Automata*, *Languages and Propgramming*, no. 5126 in LNCS, 2008. P. 39–50.
4. *Chomsky N.* On certain formal properties of grammars. Information and control. 1959. Vol. 2. No. 2. P. 137–167.
5. *Sakarovitch J.* Kleene's theorem revisited. International Meeting of Young Computer Scientists. 1986. P. 39–50.
6. *McNaughton R.*, *Yamada H.* Regular expressions and state graphs for automata. IEEE Transactions on Electronic Computers. 1960. Vol. 1. No. EC-9. P. 39–47.
7. *Gage CA et al.* URL-based sticky routing tokens using a server-side cookie jar US Patent 8. 239, 445. Aug 2012.
8. *Genova Z.*, *Christensen K. J.* Using signatures to improve URL routing. Performance, Computing, and Communications Conference, 2002. 21st IEEE International. 2002. P. 45–52.
9. *Prodanoff Z. G.*, *Christensen K. J.* Managing routing tables for URL routers in content distribution networks. International Journal of Network Management. 2004. Vol. 14. No. 3. P. 177–192.
10. Jeffrey J. McConnell. Analysis of Algorithms An Active Learning Approach: JONES AND BARTLEFT PUBLISHERS, Sudbury of Massachusetts, 2008.
11. *Кирпичев Е.* Инкрементальные регулярные выражения // Практика функционального программирования. URL: http://fprog.ru/2010/issue6/eugene-kirpichov-incremental-regular-expressions/ (дата обращения: 08.03.2016).
12. *Кейно П. П.*, *Силуянов А. В.* Автоматизированная разработка динамических *web*-узлов средствами декларативного языка программирования // Прикладная информатика. 2014. № 6 (54). С. 70–78.

References

1. Kleene S. C. *Representation of events in nerve nets and finite automata*, 1951.
2. Thompson K. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 1968, vol. 11, no. 6, pp. 419–422.
3. Gruber H., Holzer M. Finite automata, digraph connectivity, and regular expression size. In L. Aceto, I. Damgaard, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walkuwiewicz, editors. *Proceedings of the 35th International Colloqium on Automata, Languages and Propgramming*, 2008, no. 5126 in LNCS, pp. 39–50.
4. Chomsky N. On certain formal properties of grammars. *Information and control*, 1959, vol. 2, no. 2, pp. 137–167.
5. Sakarovitch J. Kleene's theorem revisited. *International Meeting of Young Computer Scientists*, 1986, pp. 39–50.
6. McNaughton R., Yamada H. Regular expressions and state graphs for automata. *IEEE Transactions on Electronic Computers*, 1960, vol. 1, no. EC-9, pp. 39–47.
7. Gage CA et al. URL-based sticky routing tokens using a server-side cookie jar US Patent 8, 239, 445, Aug 2012.
8. Genova Z., Christensen K. J. Using signatures to improve URL routing. *Performance, Computing and Communications Conference*, 2002, 21st IEEE International, 2002, pp. 45–52.
9. Prodanoff Z. G., Christensen K. J. Managing routing tables for URL routers in content distribution networks. *International Journal of Network Management*, 2004, vol. 14, no. 3, pp. 177–192.
10. Jeffrey J. McConnell. *Analysis of Algorithms An Active Learning Approach*. JONES AND BARTLEFT PUBLISHERS, Sudbury of Massachusetts, 2008.
11. Kirpichov E. *Incremental Regular Expressions*. URL: http://jkff.info/articles/ire/ (accessed 08.03.2016) (in Russian).
12. Keyno P. P., Siluyanov A. V. Automated Development of Dynamic Websites by Using Declarative Programming Language. *Prikladnaya Informatika* — Journal of Applied Informatics, 2014, vol. 9, no. 6 (54), pp. 70–78 (in Russian).

A. Kovalev, *Department of System Modeling and Engineering Graphics, Moscow Aviation Institute (National Research University), Moscow, Russia, science@blockset.ru*

P. Keyno, *Department of System Modeling and Engineering Graphics, Moscow Aviation Institute (National Research University), Moscow, Russia, science@blockset.ru*

Regular expression cardinality (size) calculation as optimality criterion in URL routing

The article is devoted to the subject of finite automata theory and regular expressions. It has practical and theoretical issues. The theoretical issue considers a calculation of cardinality (size) of regular expression. The main problem is in the redundant parts of regular expressions. We are considering their simplification. This process splits into two parts: redundancy removing, finite condition adding. Finite condition introduced to remove uncertainty and allows calculate exact number of all combinations. In case of complex regular expressions the intersection of patterns appeared. The most difficult problem is to calculate cardinality with that intersection. Authors propose to avoid using of complex regular expressions with intersections. The parameter of cardinality will be used in practical part as optimality criterion. The practical issue operates with objects identified by pattern and input string on the other side. Input string represents user-defined URL and objects represented by dynamic Web-page with variable parts. Each variable of the input data involved in behavior of the web-server in data selection. The main thesis proposed by authors in assignment of priority to the ambiguous patterns. The pattern with the lowest cardinality will be the most suitable. The solution should help to route URLs in classical developing methods.

Keywords: regular expression, regular languages, regular set, cardinality, size of regexp, finite state machine, determinization, routing, URL, formal languages.

About authors: A. Kovalev, *Student*; P. Keyno, *Lecturer*

For citation: Keyno P., Kovalev A. Regular expression cardinality (size) calculation as optimality criterion in URL routing. *Prikladnaya Informatika* — Journal of Applied Informatics, 2016, vol. 11, no. 5 (65), pp. 90–96 (in Russian).