Эффективные методы реализации проверки содержания сетевых пакетов регулярными выражениями

Д. Е. Александров

В статье рассмотрены основные способы реализации поиска по регулярным выражениям, используемым в сетевых системах обнаружения вторжений для исследования содержимого сетевых пакетов. Представлены как простейшие недетерминированные и детерминированные конечные автоматы, так и различные их модификации, показывающие более высокую производительность и (или) использующие меньшие объемы оперативной памяти.

Ключевые слова: конечные автоматы, регулярные выражения, сетевые системы обнаружения вторжений, исследование содержимого сетевых пакетов.

1. Введение

Основой обеспечения безопасности вычислительных сетей и их отдельных компонентов являются сетевые системы обнаружения вторжений. Наиболее популярными среди них являются такие программные продукты как Snort [4], Bro [5], L7-filter [6], а также аппаратные продукты фирмы Cisco [7]. Все они используют так называемый «поиск по сигнатурам», то есть поиск по заранее известному набору строк или регулярных выражений, при обнаружении которых можно сказать, что проверяемые данные вредоносны или, наоборот, безопасны для системы. Однако, при большом количество сигнатур, которые надо сопоставить с информацией, содержащейся в сетевых пакетах, возникает проблема оптимального поиска по набору регулярных выражений.

Самые простые реализации сравнения с регулярными выражениями — построение *недетерминированного конечного автомата* (НКА) или *детерминированного конечного автомата* (ДКА) [1, 2]. НКА имеет относительно небольшое количество состояний — порядка $O(l)$, где l — сумма длин реализуемых регулярных выражений, а значит и объем требуемой памяти будет небольшим. Но количество переходов между состояниями, задействованными для каждого входного символа, в худшем случае может достигнуть общего числа состояний, что приводит к снижению производительности за счет роста числа обращений к памяти. ДКА же, напротив, имеет на порядок больше состояний (в худшем случае 2^n, где n — число состояний соответствующего НКА), но при этом время, затрачиваемое на каждый символ, составляет всего $O(1)$.

В основе другой реализации, предложенной A. Ахо (Alfred V. Aho) и M. Корасик (Margaret J. Corasick) [10], также лежит ДКА, но изменен способ хранения переходов: для каждого состояния хранятся переходы только для некоторых символов входного алфавита и «переход в случае ошибки». Алгоритм работы такого автомата следующий: если для нового входного символа существует переход из текущего состояния, то осуществляем его как в обычном ДКА, если же переход отсутствует, то используем «переход в случае ошибки» и еще раз проверяем наличие перехода для того же входного символа (теперь уже для нового состояния). Такая техника впоследствии была расширена на регулярные выражения и получила *название детерминированный конечный автомат с задержкой* (D^2FA) [11].

Третий рассматриваемый способ реализации — обычный ДКА, но с некоторым набором вспомогательных данных, зависящим от конкретной модификации. При этом общей чертой всех модификаций является наличие быстровычислимого алгоритма изменения этого набора данных при каждом новом входном символе. Так, например, *дуальный конечный автомат* ($DualFA$) [13] состоит из ДКА и битового массива. При поступлении на вход нового символа производится обычный переход у ДКА и несколько битовых операций с массивом. После этого осуществляется еще 1 переход в ДКА в зависимости от значений массива и состояний ДКА. Другие варианты такого типа реализации — *расширенный конечный автомат* (XFA) [15] и *конечный автомат с счетчиками* (HFA) [14], сочетающие в себе ДКА и набор счетчиков, которые тривиальным образом изменяются в зависимости

от входного символа и текущего состояния и предотвращают «экспоненциальный взрыв» числа состояний конечного автомата в случае, когда PCRE-совместимые регулярные выражения [3] содержат символы повтора («`*`», «`+`» и «`{ , }`»).

Вышеперечисленные подходы к построению систем проверки на соответствие регулярным выражениям будут рассмотрены более подробно в следующих разделах.

2. Простейшие конечные автоматы

2.1. Регулярные языки и выражения

В дополнение к традиционному определению регулярного выражения [1, 2] как задания набора операций объединения, конкатенации и «звезды Клини» над некоторым набором символов, далее будут использоваться следующие операции PCRE-совместимых регулярных выражений:

- «`.`» — символ, обозначающий объединение всех символов языка
- «`[`$a_1a_2\ldots a_n$`]`» — обозначение объединения символов $a_1, a_2 \ldots a_n$
- «`[^`$a_1a_2\ldots a_n$`]`» — обозначение объединения всех символов языка кроме символов $a_1, a_2 \ldots a_n$
- «`r+`» — выражение, обозначающее повтор регулярного подвыражения «`r`» не менее одного раза
- «`r{m,n}`» — выражение, обозначающее повтор регулярного подвыражения «`r`» не менее m раз и не более n раз

Заметим, что вышеперечисленные операции не расширяют определение регулярного выражения, так как выводятся из базового набора операций.

Под регулярным языком над заданным алфавитом Σ далее будет подразумеваться подмножество множества Σ^*, описываемое некоторым регулярным выражением.

2.2. Недетерминированный конечный автомат

Определение. Пусть Σ — конечный алфавит, Q — конечное множество, задана функция $\delta\colon Q \times (\Sigma \cup \{\Lambda\}) \to 2^Q$, $q_0 \in Q$ и $A \subseteq Q$, тогда

набор $V = \langle Q, \Sigma, q_0, \delta, A\rangle$ назовем недетерминированным конечным автоматом (НКА) с множеством состояний Q, функцией перехода δ, инициальным (начальным) состоянием q_0 и множеством конечных состояний A.

Определение. Пусть задан НКА $V = \langle Q, \Sigma, q_0, \delta, A\rangle$, $S \subseteq Q$. Тогда назовем Λ-замыканием множества S такое множество $\Lambda(S)$, что $q \in \Lambda(S)$ тогда и только тогда, когда либо $q \in S$, либо $\exists q_0 \in S, q_1, \ldots, q_n \in Q\colon \forall i \in \overline{1,n}, q_i \in \delta(q_{i-1}, \Lambda), q \in \delta(q_n, \Lambda)$.

Для любого состояния $q\,(q \in Q)$ и строки $w\,(w \in \Sigma^*)$ определим функцию $\delta^*(q, w)$ как множество состояний, достигаемых из состояния q при обработке строки w с помощью автомата V. То есть, $\forall q \in Q\colon \delta^*(q, \Lambda) = \Lambda(\{q\})$ и $\forall q \in Q, \alpha \in \Sigma^*, a \in \Sigma\colon \delta^*(q, \alpha a) = \Lambda\left(\bigcup_{p\in\delta^*(q,\alpha)} \delta(p, a)\right)$. Пусть задан НКА $V = \langle Q, \Sigma, q_0, \delta, A\rangle$. Тогда, согласно Теореме Клини [1, 2], $L = \{\alpha \mid \alpha \in \Sigma^*, \delta^*(q_0, \alpha) \cap A \neq \varnothing\}$, называемое множеством, представимым в автомате V, является регулярным языком, а также верно и то, что каждый регулярный язык может быть представлен НКА.

Несмотря на то, что общее количество состояний НКА имеет порядок $O(l)$, где l — длина соответствующего регулярного выражения (или сумма длин, если НКА реализует набор регулярных выражений), в каждый момент времени нам необходимо хранить не одно текущее состояние, а целое множество состояний. Причем в худшем случае мощность множества текущих состояний будет иметь порядок $O(|Q|)$, где Q — множество состояний НКА, а вычислительная сложность обработки одного символа достигнет $O\left(|Q|^2\right)$.

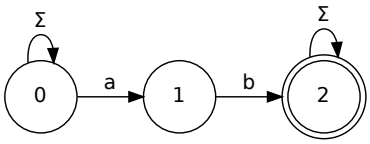
Рассмотрим простейший пример — автомат, реализующий регулярное выражение «`.*[bc][cd]`» (рис. 1(b)). Пусть на вход автомата дана строка «*abcd*», тогда изменения состояния автомата будут:

$$\{3\} \xrightarrow{a} \{3\} \xrightarrow{b} \{3,4\} \xrightarrow{c} \{3,4,5\} \xrightarrow{d} \{3,5\}.$$

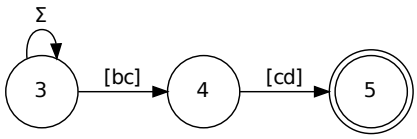
Таким образом в некоторый момент количество состояний в текущий момент времени в точности равно общему числу состояний.

2.3. Детерминированный конечный автомат

Детерминированный конечный автомат — это набор $V = \langle Q, \Sigma, q_0, \delta, A\rangle$. Здесь, как и в случае НКА, Q — конечное множе-



(a) «`.*ab.*`»

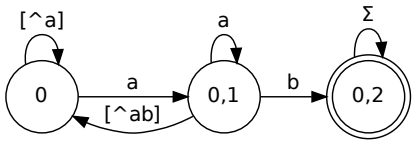


(b) «`.*[bc][cd]`»

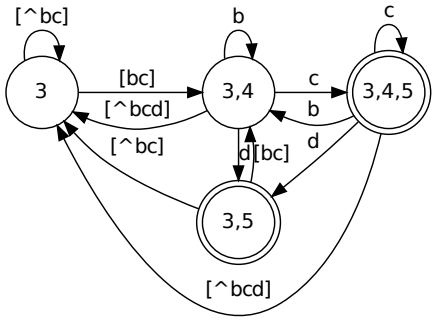
Рис. 1. Недетерминированные автоматы для регулярных выражений «`.*ab.*`» и «`.*[bc][cd]`».

ство состояний, $\delta\colon Q \times \Sigma \to Q$ — функция перехода, $q_0 \in Q$ — начальное состояние и $A \subseteq Q$ — множество конечных состояний. При этом функция достижимых состояний $\delta^*(q, w)$ определяется как $\delta^*(q, a) = \delta(q, a)$, $\delta^*(q, \alpha a) = \delta(\delta^*(q, \alpha), a)$, где $q \in Q, \alpha \in \Sigma^*, a \in \Sigma$. Таким образом, ДКА по сути является НКА без Λ-переходов, в котором в каждый момент времени хранится только одно состояние, и, следовательно, время, требуемое на обработку одного входного символа, имеет порядок $O(1)$. Однако реализация ДКА может потребовать на порядок больше памяти для хранения переходов между состояниями, так как в худшем случае количество состояний имеет порядок $O(2^n)$, где n — число состояний эквивалентного ему НКА [1].

В качестве примера рассмотрим ДКА, реализующий поиск по выражению «`.*[bc][cd]`» (рис. 2(b)). На примере изменений состояний в случае входной последовательности «*abcd*» можно заметить, что состояния 4 и 5 «независимы» в соответствующем НКА, то есть наличие или отсутствие одного из них в данный момент не означает наличие или отсутствие второго. Поэтому при построении эквива-



(a) «`.*ab.*`»



(b) «`.*[bc][cd]`»

Рис. 2. Детерминированные автоматы для регулярных выражений «`.*ab.*`» и «`.*[bc][cd]`».

лентного ДКА получаются состояния, соответствующие различным комбинациям состояний 4 и 5 НКА.

Для построения ДКА, реализующего проверку по m регулярным выражениям, строятся m НКА (обозначим их как $V_i = \langle Q_i, \Sigma, q_0^i, \delta_i, A_i \rangle$), каждый из которых реализует одно регулярное выражение. Далее они объединяются в один НКА $V = \langle Q_1 \sqcup \ldots \sqcup Q_m \sqcup \{q_0\}, \Sigma, q_0, \delta, A_1 \sqcup \ldots \sqcup A_m \rangle$, где $\delta(q_0, a) = \bigcup_{i=1}^{n} \delta_i(q_0^i, a)$ и $\delta(q, a) = \delta_i(q, a)$ при $q \in Q_i$. Полученный НКА трансформируется в ДКА [2]. В качестве примера можно рассмотреть построение ДКА, реализующего поиск по двум выражениям: «`.*ab.*`» и «`.*[bc][cd]`» (рис. 3).

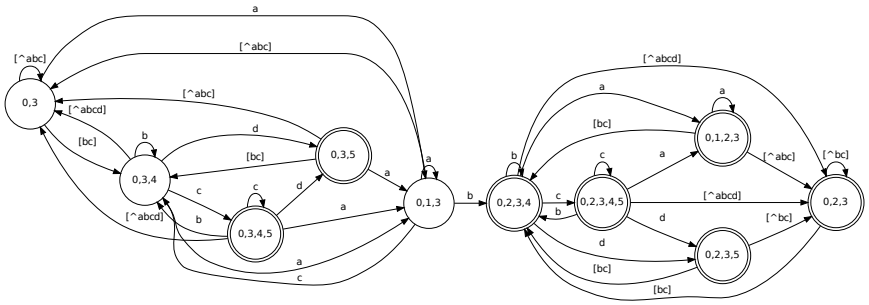


Рис. 3. Детерминированный автомат для регулярных выражений «`.*ab.*`» и «`.*[bc][cd]`».

При реализации m регулярных выражений длины n в худшем случае результирующий ДКА будет иметь порядка $O(2^{mn})$ состояний при вычислительной сложности $O(1)$.

2.4. МДКА

Альтернативой обычному ДКА, реализующему поиск по m регулярным выражениям, является МДКА — объединение нескольких параллельно выполняющихся ДКА. Например, для каждого из m регулярных выражений длины n строится свой ДКА, а обработка входных символов производится независимо на каждом автомате. Вычислительная сложность такой системы будет соответственно порядка $O(m)$, но число состояний в худшем случае будет всего $O(m2^n)$.

В случае многоядерных систем для обеспечения высокой производительности при минимизации требуемого объема памяти используется выборочное объединение регулярных выражений [8]. То есть m заданных выражений разделяются на k групп, каждая из которых компилируется в один автомат. В силу того, что современные процессоры имеют небольшую (порядка нескольких мегабайт) внутреннюю память (кэш), скорость работы которой в несколько раз превышает скорость доступа ко внешней памяти [9], основным принципом объединения выражений по группам берется ограничение на число состояний ДКА группы — их должно быть столько, чтобы таблица переходов могла полностью вместиться в кэш.

При выделении групп используется «жадный алгоритм». Для этого вводится понятие «взаимодействия» двух регулярных выражений.

Определение. Регулярные выражения R_1 и R_2 «взаимодействуют», если ДКА, реализующий поиск по $(R_1 + R_2)$, имеет больше состояний чем в сумме имеют ДКА, реализующие R_1 и R_2.

Алгоритм разбиения на группы:

```
Дано: регулярные выражения R_1, ..., R_m
      ограничение на число состояний S_max
Надо: разбиение выражений на группы
Начало:
    Построить граф G(V, E), |V| = m
    Для каждой пары вершин v_i, v_j графа G
        Если R_i и R_j взаимодействуют, провести ребро (v_i, v_j)
    Пока граф G не пуст
        Выбрать вершину V_i с меньшим числом инцидентных ребер
        Новая группа NG := R_i
        Пока не проверены все вершины
            Выбрать V_j с меньшим числом ребер, общих с NG
            Скомпилировать NG∪R_j в ДКА
            Если число состояний больше S_max
                Выйти из цикла
            Иначе
                NG := NG∪R_i
        Добавить NG в список групп
        Удалить вершины G, соответствующие выражениям из NG
Конец.
```

Экспериментальные результаты [8] над регулярными выражениями, используемыми в системах L7-filter и Bro, показывают существенное (порядка 10–40 раз) превосходство по производительности МДКА по сравнению с одной из реализаций ДКА.

3. ДКА с модифицированными переходами

3.1. Алгоритм Ахо – Корасик

В обычном ДКА $V = \langle Q, \Sigma, q_0, \delta, A \rangle$ для хранения таблицы переходов используется $|\Sigma| \cdot |Q| \cdot \lfloor \log_2 |Q| \rfloor$ бит памяти, так как для

каждого состояния $q \in Q$ и каждого символа $a \in \Sigma$ должно быть указано состояние, в которое переходит автомат из состояния q при поступившем на вход символе a. Поэтому для сокращения объема требуемой памяти важно не столько сократить общее количество состояний, сколько количество переходов.

Одним из самых известных алгоритмов, действующих по принципу сокращения количества переходов, а не состояний ДКА, является алгоритм Ахо – Корасик [10], решающий задачу поиска конечного множества слов в строке. В основе данной реализации лежит построение модификации ДКА $V = \langle Q, \Sigma, q_0, \delta, f, A\rangle$, где функция перехода δ определена только для некоторого подмножества $G \subseteq Q \times \Sigma$, а функция «перехода на случай ошибки» f отображает множество состояний Q на себя.

Алгоритм вычисления нового состояния модифицированного ДКА:

```
Дано: модифицированный ДКА V
      текущее состояние q∈Q
      входной символ a∈Σ
Надо: новое состояние q'
Начало:
    Пока (q, a) ∉G
        q := f(q)
    q' := δ(q, a)
Конец.
```

Построение такой модификации состоит из двух этапов:

1) Построение минимального ориентированного дерева, ребрам которого приписаны буквы алфавита Σ и у которого для любого слова из заданного для поиска набора существует такой путь из корня дерева, что последовательность букв ребер пути образует данное слово. Полученное дерево определяет функцию переходов: каждому узлу ставится в соответствие состояние, а корню приписывается инициальное состояние q_0; функция δ определена на паре (s_1, a) и равна s_2 (где $s_1, s_2 \in Q, a \in \Sigma$) тогда и только тогда, когда от узла, соответствующего состоянию s_1, к узлу, соответствующему состоянию s_2 идет ребро с символом a. Кроме того, доопределим инициальным состоянием q_0 функцию δ на оставшемся подмножестве множества $q_0 \times \Sigma$, если из

корня не выходит ребро с меткой a. В качестве примера можно рассмотреть дерево, соответствующее набору $\{he, she, his, hers\}$ (см. рис 4).

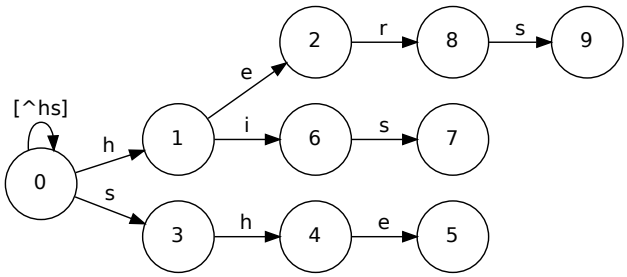


Рис. 4. Дерево, построенное по набору $(he, she, his, hers)$.

2) Построение функции «перехода на случай ошибки» f.
Вначале определим глубину состояния q как длину минимального слова $w \in \Sigma^*$ такого, что $\delta(q_0, w) = q$.
Функцию f определим рекурсивно:
   а) для состояний глубины 1 и инициального состояния положим f равной 0.
   б) пусть f определена для всех состояний глубины d (обозначим это множество как Q_d) и меньше. Для каждой пары $(q, a) \in G$, где G — область определения δ, а $q \in Q_d$, найдем такое минимальное n, что $(f^n(q), a) \in G$ и определим $f(\delta(q, a)) := \delta(f^n(q), a)$.

Пусть задан набор слов для поиска $w_1, \ldots, w_n$ и строка α. Тогда модифицированный ДКА имеет не более $2\sum_{i=1}^{n} |w_i|$ переходов и $\sum_{i=1}^{n} |w_i|$ состояний. При этом на поиск в строке α требуется не более $2|\alpha|$ переходов между состояниями [10].

3.2. Д2КА

Схожим принципом работы обладает так называемый ДКА с задержкой (Д2КА) [11], где также для каждого состояния заданы переходы только для некоторых символов алфавита и переходы для

остальных символов («на случай ошибки»). Главным отличием от предыдущей модификации является то, что данный автомат может быть построен на основе произвольного ДКА, в том числе ДКА, реализующего поиск по наборам регулярных выражений.

Пусть ДКА $V = \langle Q, \Sigma, q_0, \delta, A \rangle$ реализует поиск по заданному набору регулярных выражений. Обозначим через G неориентированный полный граф, в котором каждому состоянию автомата соответствует вершина, а каждому ребру между произвольными вершинами u и v приписаны веса $w = |\{a \in \Sigma \mid \delta(q_u, a) = \delta(q_v, a)\}| - 1$, где q_u и q_v — соответствующие состояния. Вес произвольного ребра (u, v) данного граф показывает на сколько максимально может сократиться количество переходов, если заменить часть переходов состояния q_u на «переход на случай ошибки» в состояние q_v. Пример ДКА и графа для набора выражений («`.*a+`», «`.*b+c`», «`.*c*d+`») изображен на рисунке 5.

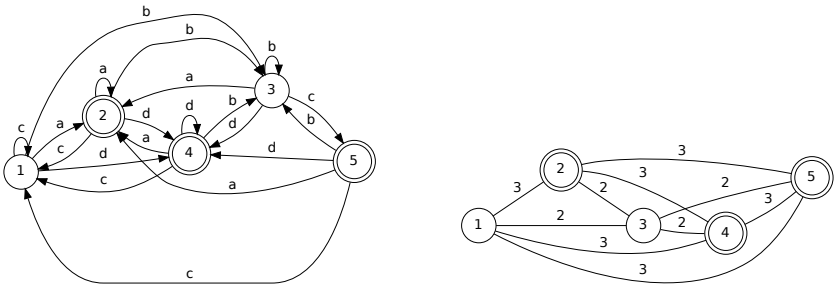


Рис. 5. ДКА и граф для выражений «`.*a+`», «`.*b+c`» и «`.*c*d+`».

Для минимизации числа переходов автомата необходимо выбрать такие «переходы на случай ошибки», чтобы сумма соответствующих весов графа G была максимальна, причем наличие циклов из данных переходов запрещено. Однако при минимизации числа состояний может получиться так, что появятся длинные цепочки из переходов «ошибки», то есть при обработке 1 символа количество осуществляемых переходов может заметно превысить 2. Например, в случае Д2КА для выражений («`.*a+`», «`.*b+c`», «`.*c*d+`»), изображенного

на рисунке 6, обработка строки «ссс» займет 12 переходов, по 4 перехода на каждый символ.

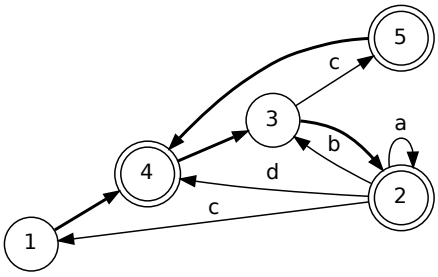


Рис. 6. Один из Д2КА для выражений («`.*a+`», «`.*b+c`», «`.*c*d+`»). «Переходы на случай ошибки» обозначены жирными линиями.

Поэтому одним из вариантов алгоритма выбора переходов «ошибки» является «жадный» алгоритм с задаваемым ограничением на максимальную длину цепочки из переходов. Для этого строится лес деревьев, в котором каждой вершине соответствует состояние исходного автомата, с диаметром не превышающим заданное значение l, и у каждого дерева корень выбирается так, чтобы глубина была минимальна ($\lfloor l/2 \rfloor$). Ребра данных деревьев, ориентированные от большей глубины к меньшей, определяют соответствующие «переходы на случай ошибки».

Алгоритм построения леса деревьев:

```
Дано: граф весов G
      диаметр l
      язык Σ
Надо: лес с диаметром деревьев не больше l
Начало:
    Лес wood := ∅
    Множество ребер weight_set [|Σ|]
    Для каждого ребра (u, v) ∈ G
        Если вес weight((u, v)) = w, w > 0, то
            weight_set[w] := weight_set[w] ∪ {(u, v)}
```

```
Для каждого i от |Σ| до 1
    Пока weight_set[i] не пусто
        Выбрать ребро (u, v) из weight_set[i], которое
        приведет к наименьшему росту диаметра wood
        Если u и v не принадлежат одному дереву в wood и
        диаметр wood ∪ (u, v) не больше l, то
            wood := wood ∪ (u, v)
        Убрать ребро (u, v) из weight_set[i]
Конец.
```

Такое построение позволяет ограничить максимальное количество переходов при обработке одного символа. Однако, во-первых, как было показано выше, количество переходов при обработке строки длины n может достигать $n \cdot (l + 1)$, где l — максимальная длина цепочки переходов «ошибки»; во-вторых, количество переходов может не достигнуть минимума среди всех эквивалентных ему Д2КА.

Другой способ выбора «переходов на случай ошибки» [12] не гарантирует минимальность общего числа переходов, но гарантирует, что число переходов при разборе строки длины n не превысит $2n$. В основе этого способа лежит тот факт, что если все переходы «ошибки» идут от состояний автомата с большей глубиной к состояниям с меньшей глубиной, то количество осуществляемых переходов не может превысить двух длин обрабатываемой строки.

Построение такой модификации осуществляется следующим образом образом: строится граф G, для каждого состояния ДКА V находится его глубина и для каждого состояния выбирается переход «ошибки» к состоянию с меньшим весом.

Результаты экспериментов над реально использующимися регулярными выражениями, описанных в статьях [11, 12], показывают, что сжатие числа переходов исходного ДКА может достигать порядка 99% в обоих способах. Причем если в первом способе ограничить длину цепочки двумя (что сократит максимальное количество переходов до максимального количества из второго способа), то сжатие может сократиться до 90–97%.

3.3. Дуальный автомат

Как уже было отмечено выше, одной из главных причин «экспоненциального взрыва» числа состояний при переходе от НКА к ДКА

является наличие большого числа независимых состояний НКА. Одним из методов решения данной проблемы является выделение состояний, независимых с большим числом других состояний, в отдельный НКА с быстрым алгоритмом вычисления нового набора состояний, а оставшиеся состояния исходного НКА преобразуются в ДКА с дополнительными переходами [13]. Для взаимодействия параллельно работающих ДКА и НКА используется дополнительный механизм, который будет описан позднее.

В качестве быстро (за время $O(1)$) вычислимого НКА берется автомат, называемый линейным конечным автоматом (ЛКА) [13], определение которого отличается от стандартного определения ЛКА [1, гл. 3.11].

Определение. Назовем НКА $V = \langle Q, \Sigma, q_0, \delta, A\rangle$ линейным конечным автоматом, если $\forall q \in Q, a \in \Sigma : |\delta(q,a) \setminus \{q\}| \leqslant 1$, $|\{q' \mid q' \in Q, q' \neq q, \exists a \in \Sigma : \delta(q',a) = q\}| \leqslant 1$. То есть из любого состояния можно попасть в не более чем 1 состояние, не считая данного, и не существует двух различных состояний, которые могут перейти в третье.

Для хранения текущего набора состояний в таком ЛКА используется битовый массив, в котором каждому состоянию автомата соответствует один бит. Причем если из состояния q_i, соответствующего биту i, существует переход в состояние q_j, то состоянию q_j соответствует бит $(i-1)$. Для вычисления нового состояния в памяти хранится $2|\Sigma|$ битовых массивов длины k бит, где Σ — алфавит автомата, а k — общее число состояний НКА автомата ($|Q|$). Из них $|\Sigma|$ массивов (обозначаемых $self[]$) хранят информацию о переходах состояний в себя: пусть $self[a]$ — массив, соответствующий входному символу $a \in \Sigma$, тогда i-ый бит равен 1 тогда и только тогда, когда для соответствующего состояния q_i верно, что $\{q_i\} \in \delta(q_i, a)$. Оставшиеся $|\Sigma|$ массивов (обозначаемых $next[]$) используется для вычисления переходов в другие состояния: пусть $next[a]$ — массив, соответствующий входному символу a, тогда i-ый бит массива равен 1 тогда и только тогда, когда $\{q_{i-1}\} \in \delta(q_i, a)$. Тогда, если l — массив текущих состояний, то новый массив состояний при обработке символа a будет равен $(l \& self[a]) \,|\, ((l \& next[a]) \gg 1)$, где $\&$, $|$ и $\gg$ — битовые операции «и», «или» и операция сдвига соответственно.

ДКА имеет 2 типа дополнительных переходов, которые используются для взаимодействия с ЛКА: *расширенный переход* — обычный

переход, к которому приписан k-битный массив, задающий активные состояния ЛКА, и *условный переход* — переход, осуществляющийся при определенном наборе активных состояний (для каждого символа $a \in \Sigma$ задается массивом длины k $extern[a]$).

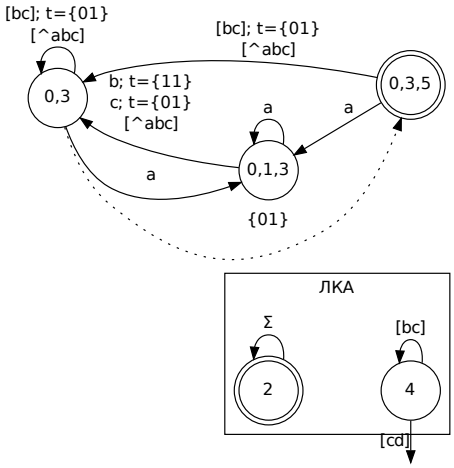
Алгоритм работы дуального автомата:

```
Дано: ДКА с дополнительными переходами V = ⟨Q, Σ, q_0, δ, A⟩
      ЛКА L
      текущее состояние(q, l)
      входной символ a
Надо: вычислить новое состояние (q', l')
Начало:
    q' := δ(q, a)
    Если переход q --a--> q' - расширенный с массивом t, то
        l' := t
    иначе
        l' := 0
    Если (l&extern[a]) ≠ 0, то
        q' =условный переход(q', l&extern[a])
    l' := l'|(l&self[a])|((l&next[a]) ≫ 1)
Конец.
```

В качестве примера можно рассмотреть дуальный автомат, реализующий поиск выражений «`.*ab.*`» и «`.*[bc]+[cd]`», изображенный на рисунке 7.

Как и в предыдущих модификациях для построения дуального автомата вначале по регулярным выражениям строится НКА $V = \langle Q, \Sigma, q_0, \delta, A\rangle$. Далее выбираются состояния, которые будут входить в ЛКА: пусть уже выбрано множество состояний $Q_l \subset Q$ и по ним построен соответствующий ЛКА, тогда среди всех состояний $Q \setminus Q_l \cup \{q_0\}$, которые могут быть добавлены в ЛКА, выбираем состояние q с максимальной характеристикой $(|\{a \mid a \in \Sigma, \delta(q,a) = q\}| + |\{a \mid a \in \Sigma, \exists q' \in Q_l : \delta(q',a) = q\}|)$. В качестве начального состояния q_0 берется $0^{|Q_l|}$. Массивы $extern$, $next$ и $self$ строятся следующим образом:

- $extern[a][i] = 1 \Leftrightarrow \delta(q_i, a) \nsubseteq Q_l$
- $next[a][i] = 1 \Leftrightarrow q_{i-1} \in \delta(q_i, a)$
- $self[a][i] = 1 \Leftrightarrow q_i \in \delta(q_i, a)$



`extern[cd]`	01
`extern[^cd]`	00
`next[.]`	00
`self[bc]`	11
`self[^bc]`	10

Рис. 7. Дуальный автомат для регулярных выражений «`.*ab.*`» и «`.*[bc]+[cd]`». В массиве ЛКА состояние «2» представлено левым битом, а состояние «4» — правым. Условный переход обозначен пунктиром.

Построение ДКА с дополнительными переходами осуществляется на основе ДКА V', соответствующего исходному НКА V:

Дано: НКА $V = \langle Q, \Sigma, q_0, \delta, A\rangle$
ДКА $V_d = \langle Q', \Sigma, q'_0, \delta', A'\rangle$
ЛКА $V_l = \langle Q_l, \Sigma, q^l_0 = 0, \delta\rangle$
Элементы множества Q' рассматриваются
как подмножества Q
Надо: построить ДКА с дополнительными переходами
Начало:
Множество состояний искомого ДКА $Q_d := \varnothing$
Для каждого $q \in Q'$
$q_d := q \setminus Q_l$
$Q_d := Q_d \cup \{q_d\}$
$q_l := q \cap Q_l$
Для каждого $a \in \Sigma$
$q' := \bigcup_{p \in q_d} \delta(p,\ a) \setminus Q_l$

$Q_d := Q_d \cup \{q'\}$
Если $\bigcup_{p \in q_d} \delta(p,\ a) \cap Q_l \neq \varnothing$, то
    Добавить расширенный переход $q \xrightarrow{a} q'$ с
    массивом, соответствующим множеству
    состояний $\bigcup_{p \in q_d} \delta(p,\ a) \cap Q_l$
иначе
    Добавить обычный переход $q \xrightarrow{a} q'$
$q'' := \bigcup_{p \in q_l} \delta(p, a) \setminus Q_l$
Если $q'' \not\subseteq q'$, то
    $Q_d := Q_d \cup \{q' \cup q''\}$
    Добавить условный переход $q' \xrightarrow{l\&extern[a]} \{q' \cup q''\}$
Конец.

Из построения данного автомата видно, что в худшем случае объем необходимой памяти будет иметь порядок $O\left(2^{|Q|-|Q_l|} + 2 \cdot |\Sigma| \cdot |Q_l|\right)$. При этом обработка одного символа занимает порядка $O\left(1 + 3\log_k |Q_l|\right)$, где k — максимальная длина битового массива, с которым можно производить операции &, | и $\gg$ за единицу времени.

3.4. Расширенный детерминированный конечный автомат

Последний рассматриваемый способ реализации сокращает число независимых состояний НКА, соответствующих подвыражениям вида «`[...]*`», а также значительно сокращает количество состояний, необходимых для реализации подвыражений вида «`[...]{n,k}`».

Пусть имеется n регулярных выражений длины l вида «`.*`R_i`.*`R'_i», где R_i и R'_i — различные строки. Тогда количество состояний соответствующего ДКА имеет порядок $O(nl2^n)$ [15, 16]. Рассмотрим простейший пример: реализация регулярных выражений «`.*ab.*cd`» и «`.*ef.*gh`» (рис. 8). Как видно из схемы ДКА, состояния, соответствующие подвыражениям «`.*`» в середине первого и второго выражений, независимы с состояниями автоматов, реализующих второе и первое выражения соответственно, что и приводит к росту состояний с оценкой $O(nl2^n)$ при переходе от НКА к ДКА.

Для устранения экспоненциальной составляющей 2^n в расширенном детерминированном конечном автомате [15], а равно и в конечном автомате с счетчиками [14] (так как данные два автомата схо-

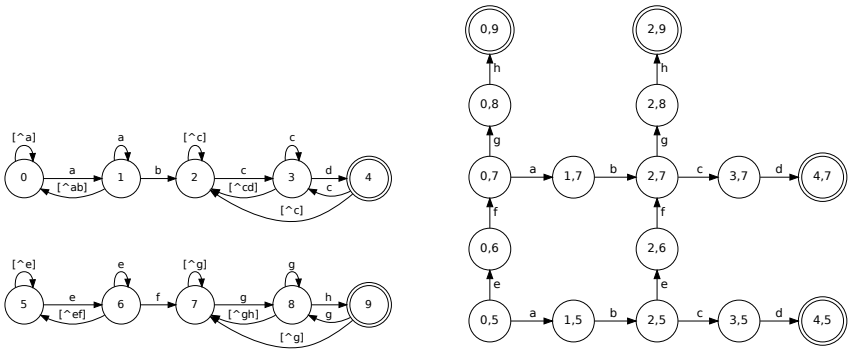


Рис. 8. Детерминированные автоматы для регулярных выражений «`.*ab.*cd`» и «`.*ef.*gh`».

жи по принципу работы, далее рассматривается только расширенный автомат), для каждого подвыражения вида «`.*`» исходного выражения «R`.*`R'» в памяти хранится один бит, равный единице, если проверенная (обработанная) часть строки соответствует регулярному выражению «R`.*`». Для манипуляций с данными битами к некоторым переходам конечного автомата приписываются инструкции типа «установить бит в 1» и «установить бит в 0», выполняемые во время перехода. Такое дополнение приводит к тому, что, например, число состояний двух РДКА для выражений «`.*ab.*cd`» и «`.*ef.*gh`» совпадает с числом состояний соответствующих ДКА (рис. 9), но число состояний расширенного автомата для обоих выражений в данном случае не превышает суммы числа состояний отдельных ДКА. Таким образом автоматы с данным расширением имеют порядка $O(nl)$ состояний при вычислительной сложности $O(1)$ в случае, когда соответствующие подстроки регулярных выражений не являются суффиксами друг друга.

Пусть имеется регулярное выражение «`.{n}`». Количество состояний соответствующих НКА и ДКА (рис. 10) имеет порядок $O(n)$. Для сокращения числа состояний в РДКА добавляются счетчики, изменяющие свои значения при осуществлении определенных переходов. Так для выражения «`.{n}`» используется РДКА, изображенный на рисунке 10. Таким образом, в случае ДКА минимальное число состо-

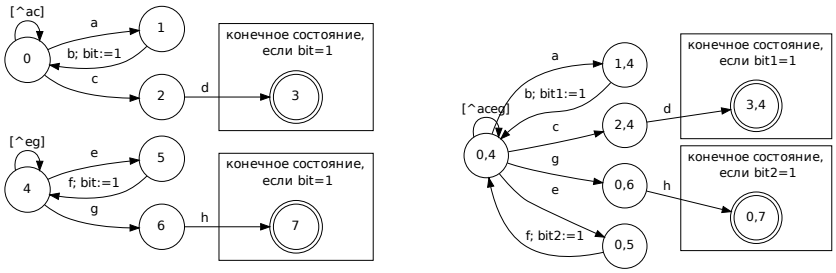


Рис. 9. Расширенные детерминированные автоматы для регулярных выражений «`.*ab.*cd`» и «`.*ef.*gh`».

яний достигает $n + 2$, а РДКА состоит всего из одного состояния и счетчика размера $\lceil \log_2 (n + 2) \rceil$ бит.

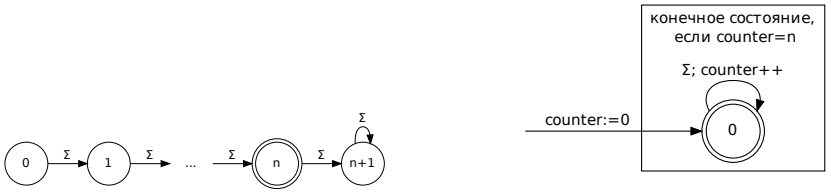


Рис. 10. ДКА и РКА для регулярного выражения «`.{n}`».

Однако стоит заметить, что такой подход применим не ко всем регулярным выражениям. Так для выражения «`.*a.{n}b`» конечный автомат в каждый момент времени должен «помнить» последние n символов, то есть хранить информацию, встречался ли символ a среди них и если да, то на каких местах. А для этого необходимо использовать не меньше n бит.

Построение РДКА для одного регулярного выражения состоит из четырех этапов:

1) Выделение в регулярном выражении подвыражений вида «`[...]*`» и «`[...]{m,n}`», где количество элементов в классах символов «`[...]`» близко к числу символов алфавита. Для обо-

значения таких подвыражений (за исключением подвыражения, являющегося началом исходного выражения, и подвыражения вида «`[...]*`» в конце выражения) далее используется символ «#». Так, например, регулярное выражение «`.*ab[^a]{n}c.*`» после обработки записывается как «`.*ab#[^a]{n}c#.*`».

2) Построение недетерминированного расширенного конечного автомата. Вначале строится расширенный НКА с Λ-переходами $V = \langle Q, D, \Sigma, (q_0, d_0), \delta, U, A\rangle$ (где множество D — множество значений дополнительных данных, таких как, например, счетчики; $d_0 \in D$ — начальное значение дополнительных данных; $U\colon Q \times (\Sigma \cup \{\Lambda\}) \times Q \to 2^{D\times D}$ — функция, определяющая изменение дополнительных данных для каждого перехода; $A \subseteq Q \times D$ — множество заключительных состояний), который представляет из себя обычный НКА с Λ-переходами, но с инструкциями изменения счетчиков или бит у некоторых из переходов. Построение автоматов для двух базовых типов регулярных выражений «R_1`#`R_2» и «`R{m,n}`» изображено в виде схемы переходов на рисунке 11 (состояния 0 и 1 — начальное и конечное состояния НКА выражения «R_1», 3 и 4 — начальное и конечное состояния НКА выражения «R_2», 1 и 2 — начальное и конечное состояния НКА выражения «`R`»).

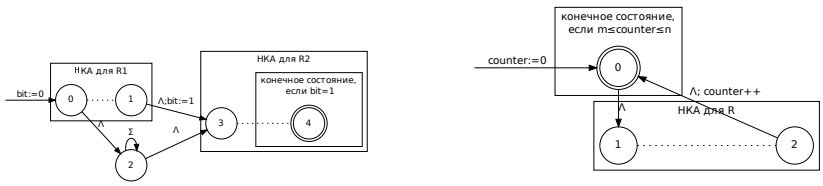


Рис. 11. РНКА для регулярных выражений «R_1`#`R_2» и «`R{m,n}`».

Далее в автомате устраняются Λ-переходы (при этом возможно, что вместо одного начального состояния, у нового автомата будет целое множество начальных состояний):

`Дано: РНКА с Λ-переходами` $V = \langle Q, D, \Sigma, (q_0, d_0), \delta, U, A\rangle$
`Надо: РНКА без Λ-переходов` $V' = \langle Q', D, \Sigma, Q'_0, \delta', U', A\rangle$
$(Q'_0 \subseteq Q' \times D)$

```
Начало:
    Для каждых q ∈ Q и a ∈ Σ таких, что δ(q,a) ≠ ∅
        Для каждой пары (q', f') ∈
        НайтиΛДостижимыеСостояния(δ(q,a))
            δ'(q,a) := δ'(q,a) ∪ q'
            U'(q,a,q') := U'(q,a,q') ∪ f'
    Q'_0 := ∅
    Для каждой пары (q', f') ∈
    НайтиΛДостижимыеСостояния(q_0)
        Q'_0 := Q'_0 ∪ (q', f'(d_0))
Конец.
Функция НайтиΛДостижимыеСостояния(q)
    result := (q, E), где E: D → D,  ∀d ∈ D :  E(d) = d
    Для каждой пары (q', f') ∈ result
        Для каждого q'' ∈ δ(q', Λ)
            result := result  ∪  {q''} × U(q', Λ, q'')(f')
Вернуть result.
```

3) Построение расширенного ДКА. Алгоритм преобразования РНКА в РДКА повторяет алгоритм преобразования НКА в ДКА с тем отличием, что в расширенных автоматах учитываются преобразования дополнительных данных $d \in D$, причем дополнительные данные в РДКА представляют собой подмножества множества $Q \times D$, где Q и D — множества состояний и дополнительных данных РНКА соответственно.

4) Приведение РДКА к простейшему виду, то есть к РДКА, в котором дополнительные данные представляют набор счетчиков и бит, к каждому переходу между состояниями приписан набор действий над ними (например, сбросить счетчик или бит в ноль, увеличить счетчик на один, установить бит в единицу), а состояние является конечным только если счетчики и биты имеют определенные значения. С помощью перебора ищется непротиворечивое взаимнооднозначное соответствие множества дополнительных данных исходного РНКА и полученного РДКА, а также изменений, приписанных к переходам РДКА, и набора простейших инструкций изменения счетчиков и бит РНКА.
В случае если такого соответствия не существует, то невозможно построить требуемый простейший РДКА, однако на практи-

ке количество реально используемых регулярных выражений, для которых нельзя построить РДКА, крайне мало [16].

Полученные по заданному набору регулярных выражений РДКА могут быть объединены в один РДКА. Построение такого РДКА почти полностью совпадает с объединением нескольких ДКА в один с тем исключением, что операции, производимые при переходах, комбинируются, то есть если в построенном РДКА есть переход $\{q_1, \ldots, q_n\} \xrightarrow{a} \{q'_1, \ldots, q'_n\}$ (где $a \in \Sigma$; q_i и q'_i — состояния исходных автоматов РДКА $V_i = \langle Q_i, D_i, \Sigma, (q_{i0}, d_{i0}), \delta_i, U_i, A_i \rangle$ и $\delta_i(q_i, a) = q'_i$), то функция изменения дополнительных данных $D = \prod_{i=1}^{n} D_i$ на данном переходе определяется как $U = \prod_{i=1}^{n} U_i\,(q_i, a, q'_i)$, а множество заключительных состояний $A = \prod_{i=1}^{n} A_i$.

Согласно экспериментальным результатам над сигнатурами системы Snort [15, 16] из 1556 сигнатур РДКА не могут быть построены всего для 106. Автомат для оставшихся 1450 сигнатур занимает всего 47.5 МБ, а дополнительные данные требуют не более 193 бит для хранения. При этом порядка 96% переходов имеют не более одной инструкции на переход, около 20% состояний имеют только одну проверку значений данных, а 78.9% состояний вообще не имеют проверок. Производительность же РДКА в 20 и более раз превышает аналогичные МДКА, не уступая в минимизации объема требуемой оперативной памяти.

4. Заключение

В статье рассмотрены основные способы реализации поиска по регулярным выражениям, используемым в сетевых системах обнаружений вторжений: от простейших НКА и ДКА до автоматов с дополнительными структурами данных.

Несмотря на все многообразие способов реализации, можно выделить две основные методики повышения производительности и снижения требуемых объемов оперативной памяти автоматов. Первая методика состоит в уменьшении числа переходов или состояний за счет добавления небольшого числа нетривиальных переходов как, например, в детерминированном конечном автомате с задержкой. Вторая методика заключается в параллельном исполнении нескольких детерминированных конечных автоматов (МДКА) или ДКА и быст-

ровычислимого автомата, требующего небольшое число оперативной памяти (в таком виде может быть представлен РДКА). Также возможно совмещение этих методик, что и видно в дуальном конечном автомате.

Список литературы

[1] Кудрявцев В. Б., Алешин С. В., Подколзин А. С. Введение в теорию автоматов. — М.: Наука, 1985.

[2] Martin J. C. Introduction to Languages and the Theory of Computation. — Изд. 4-е. — New York: McGraw-Hill, 2011.

[3] http://www.pcre.org/

[4] http://www.snort.org/

[5] http://www.bro.org/

[6] http://l7-filter.sourceforge.net/

[7] http://www.cisco.com/

[8] Yu F. et al. Fast and memory-efficient regular expression matching for deep packet inspection // Architecture for Networking and Communications systems, 2006. ANCS 2006. ACM/IEEE Symposium. — IEEE, 2006. — P. 93–102.

[9] Molka D. et al. Memory performance and cache coherency effects on an Intel Nehalem multiprocessor system // Parallel Architectures and Compilation Techniques, 2009. PACT'09. 18th International Conference. — IEEE, 2009. — P. 261–270.

[10] Aho A. V., Corasick M. J. Efficient string matching: an aid to bibliographic search // Communications of the ACM. — 1975. — V. 18. N 6. — P. 333–340.

[11] Kumar S. et al. Algorithms to accelerate multiple regular expressions matching for deep packet inspection // ACM SIGCOMM Computer Communication Review. — ACM, 2006. — V. 36. N 4. — P. 339–350.

[12] Becchi M., Crowley P. An improved algorithm to accelerate regular expression evaluation // Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems. — ACM, 2007. — P. 145–154.

[13] Liu C., Wu J. Fast Deep Packet Inspection with a Dual Finite Automata. — 2013.

[14] Kumar S. et al. Curing regular expressions matching algorithms from insomnia, amnesia, and acalculia // Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems. — ACM, 2007. — P. 155–164.

[15] Smith R. et al. Deflating the big bang: fast and scalable deep packet inspection with extended finite automata // ACM SIGCOMM Computer Communication Review. — ACM, 2008. — V. 38. N 4. — P. 207–218.

[16] Smith R., Estan C., Jha S. XFA: Faster signature matching with extended automata // Security and Privacy, 2008. SP 2008. IEEE Symposium. — IEEE, 2008. — P. 187–201.