Модификация конечного автомата через применение алгоритмов сжатия

Бернадотт А.[1]

Аннотация

Решение вопроса о принадлежности слова регулярному языку находит приложение в областях, где осуществляется поиск определенных паттернов в данных различной природы. Актуальной является проблема роста числа состояний распознающего детерминированного конечного автомата (ДКА) от числа регулярных выражений распознаваемого языка — проблема экспоненциального взрыва. В данной статье рассматривается модификация конечного автомата через применение алгоритмов сжатия, работающих без изменения распознаваемого языка и без добавления дополнительных структурных элементов автомата.

Ключевые слова: ДКА, НДКА, регулярный язык, экспоненциальный взрыв, алгоритм сжатия.

1. Введение

Решение вопроса о принадлежности слова регулярному языку находит приложение в разных областях, где является актуальным поиск определенных паттернов в данных различной природы. Так, алгоритмы проверки принадлежности слова регулярному языку применяются в системах определения сетевого протокола, системах обнаружения вторжений с поиском по сигнатурам, сетевых процессорах, в анализе биохимических данных (в частности для поиска лиганда рецептора в протеоме) [1, 2, 3, 4, 5, 6, 7].

[1] *Бернадотт Александра* — к.м.н., аспирантка каф. математической теории интеллектуальных систем мех.-мат. ф-та МГУ, e-mail: alexandra.bernadotte@gmail.com.

Bernadotte A. — PhD in Med, graduate student, Lomonosov Moscow State University, Faculty of Mechanics and Mathematics, Chair of Mathematical Theory of Intellectual Systems.

Решение вопроса о принадлежности слова регулярному языку в классическом варианте осуществляется детерминированным конечным автоматом (ДКА), обеспечивающим быстроту работы, но большую пространственную сложность для некоторых регулярных языков. Классической альтернативной ДКА является недетерминированный конечный автомат (НДКА), представляющий регулярный язык. Согласно теореме о эквивалентности детерминированных и недетерминированных конечных автоматов, всякий язык принимается некоторым НДКА тогда и только тогда, когда этот язык принимается некоторым ДКА [8]. Кроме того, существует алгоритм построения из НДКА эквивалентного ему ДКА [8, 9]. При этом в худшем случае НДКА требует удержания в памяти всех состояний.

Таким образом, актуальной остается проблема экспоненциального взрыва — проблема роста числа состояний распознающего детерминированного конечного автомата (ДКА) от числа регулярных выражений распознаваемого языка. В общем случае при входном алфавите мощности более или равной двум мощность множества состояний экспоненциально возрастает [10, 11, 12]. К признакам, вызывающим экспоненциальный взрыв, относятся: использование в регулярных выражениях сочетаний ".+" и ".*" и "считающих" выражений, типа ".{n}" и "[$\neg a_i$]{n}", представленных в виде PCRE (Perl Compatible Regular Expressions) [13].

Глобально можно выделить выделить три алгоритмических подхода к решению проблемы экспоненциального взрыва:
1) модификация распознаваемого регулярного языка;
2) модификация структуры конечного автомата через добавление специальных структурных элементов;
3) модификация конечного автомата через применение алгоритмов сжатия.
Первый подход — подход через модификацию распознаваемого регулярного языка основан на изменении распознаваемого языка с целью упрощения распознающего автомата. Александровым была предложена оптимизация языков вида $\bigcup_{i=1}^{n} .*R_i.*R_i^{'}.*$ за счет замены пары слагаемых на слагаемое $(R_{i_1}|R_{i_2}).*(R_{i_1}^{'}|R_{i_2}^{'})$. Такая замена очевидным образом расширяла язык, однако практически в системах обнаружения вторжений модификация снижала требование к памяти на порядок. Оценка числа состояний конечного автомата для такого вида языка и ошибка погрешности при распознавании принадлежности слова искомому регулярному языку при данном расширении языка приведены Александровым в следующих работах [14, 15, 16].

К второму подходу относятся следующие решения: двойной конечный автомат, представляющий собой комбинацию конечных автоматов, один из которых несет вспомогательную функцию хранения метасостояния другого; расширенный ДКА (в первоисточнике — extended finite automata), где дополнительно хранится битовый массив метасостояний автомата и используются счетчики для языков типа $.\{n\}$; ДКА с счетчиками для решения проблемы экспоненциального взрыва для языка $\bigcup_{i=1}^{n} .*\alpha_i.*\beta_i.*$, где α_i, β_i — непустые слова в алфавите A [5, 7, 17].
В данной статье нам бы хотелось глубже рассмотреть вопросы, описанные в работах Александрова [18, 19], и сфокусироваться на третьем подходе — модификации конечного автомата через применение алгоритмов сжатия. Подход заключается в модификации структуры конечного автомата без изменения распознаваемого языка и без добавления дополнительных структурных элементов.

2. Обзор модификаций конечного автомата через применение алгоритмов сжатия

2.1. Методы сжатия с использованием недерминированных переходов на случай ошибки

Большинство методов сжатия без использования дополнительных структурных элементов основаны на добавлении недетерминированных переходов в распознающий заданный язык ДКА.

2.1.1. Алгоритм Ахо-Корасик — НДКА с переходами в случае ошибки [20]

В 1975 году Альфред Ахо и Маргарет Корасик (Alfred Aho и Margaret Corasick) разработали алгоритм, осуществляющий поиск слов над алфавитом A из заданного словаря в строке. Алгоритм строит модифицированный инициальный ДКА на основе префиксного дерева [21]. Сжатие осуществляется через использование “переходов в случае ошибки” — переходов, которые отсутствуют в предварительно построенном префиксном дереве. То есть для каждого состояния нужно хранить информацию только о двух исходящих переходах: о детерминированном переходе по соответствующему символу из алфавита A (переход соответсвует переходу префиксного дерева) и о переходе в случае “иначе”. Такая мо-

дификация ДКА позволяет сократить количество переходов, оставляя количество состояний исходного ДКА интактным.

Формально, полученный НДКА с переходами в случае ошибки, описывается следующим образом: $V = (A, Q, F, \varphi_{Q\times A}, \varphi_{er}, q_0)$, где A — входной алфавит, Q — алфавит состояний, F — алфавит принимающих состояний, $F \in Q$, $\varphi_{Q\times A}$ — функция перехода, определенная на $Q \times A$, φ_{er} — функция перехода в случае ошибки, определенная на Q. Алгоритм построения функции в случае ошибки представлен ниже — см. Algorithm 1. В алгоритме построения функции в случае ошибки используется термин “глубина состояния”, что есть минимальное число переходов от данного состояния до начальное состояния конечного автомата. Аналогично используется термин “глубина вершины”. Вычислительная

Algorithm 1 Алгоритм построения функции в случае ошибки, φ_{er}

1: Вход: префиксное дерево G для соответствующего множества слов S над алфавитом A. Q — множество вершин префиксного дерева, которые соответствуют состояниям искомого НДКА с переходами в случае ошибки, для которого не определена φ_{er}. Выход: φ_{er} НДКА с переходами в случае ошибки.
2: **for** q_i из Q **do**
3: Определить глубину вершины(глубину состояния искомого НДКА с переходами в случае ошибки), d_i
4: **if** $d_i \leq 1$ **then**
5: $\varphi_{er}(q_i) = 0$
6: **end if**k = 1
7: **for** q_i из Q, при условии $d_i \geq k$ **do**
8: **for** q_j из Q, при условии $d_j \geq k$ **do**
9: **for** a_k из A, для которых не определена $\varphi(q_i, a_k)$ **do**
10: Найти q_j, которое соответствует самому длинном суффиксу подслова, состоящего из букв, являющимися непустыми метками подряд идущих переходов, ведущих в состояние q_i, и для которого определена $\varphi(q_j, a_k)$
11: $\varphi(q_i, a_k) == \varphi(q_j, a_k)$
12: **end for**
13: **end for**
14: увеличиваем k на единицу
15: **end for**
16: **end for**

сложность алгоритма Ахо-Корасик напрямую зависит от организации

данных. Пространственно, НДКА Ахо-Корасик с переходами в случае ошибки имеет $\Sigma_{i=1}^{n}|s_i|$ состояний, где s_i — слово из множества распознаваемых слов S, а число переходов сокращается с $|A||Q|$ для исходного ДКА ($V = (A, Q, F, \varphi, q_0)$) до 2 $\Sigma_{i=1}^{n}|s_i|$ в худшем случае для искомого НДКА Ахо-Корасик с переходами в случае ошибки.

2.1.2. Сжатие ДКА на основе общего префикса регулярных выражений

Логичным переходом от алгоритма Ахо-Корасик, работающего на строках, является поиск и использование префикса на регулярных выражениях, позволяющий сжать ДКА на основе нахождения эквивалентных вершин, соответствующих регулярным выражениям, Perl-совместимая запись которых имеет один префикс. Иллюстрация данного перехода от строкового алгоритма Ахо-Корасик на регулярные выражения приведена в работе [2] (см. рис 1). Однако более подробно данный переход на регулярные выражения был осуществлен Кумаром и соавторами и представлен в виде НДКА с задержками.

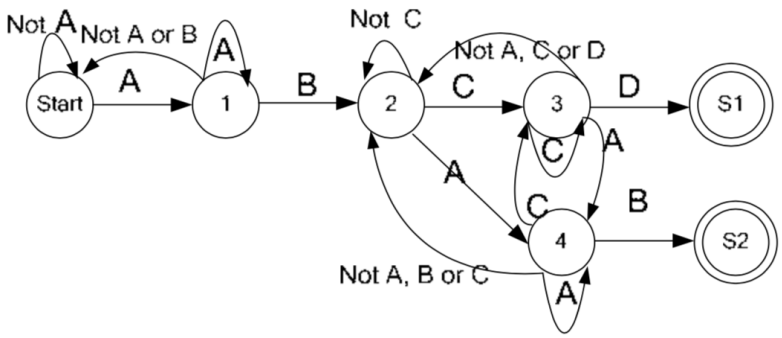

Рис. 1. ДКА, принимающий регулярные выражения $.*ABCD$ и $.*ABAB$ с общим префиксом. Источник: [2].

2.1.3. НДКА с задержками [1, 22, 23]

НДКА с задержками (в первоисточнике — Delayed Input DFA, или сокращенно — D2FA) представляет собой инициальный автомат, переходы которого помечены символами входного алфавита или пустым словом: $D2FA = \{A \cup \{\lambda\}, F, Q, \varphi, q_0\}$, A — входной алфавит, λ — пустое слово,

F — алфавит выходных состояний, φ — функция перехода, q_0 — начальное состояние (рис. 2) [1].

Ребра, помеченные пустым словом, носят название — “переход по умолчанию”. Каждое состояние данного ДКА с задержками может иметь не более одного перехода по умолчанию. Само построение НДКА с задержками из ДКА опирается на лемму:

Лемма 1. *Если $\varphi(a, q_i) = \varphi(a, q_j)$ для всех a_k из алфавита A, то множество переходов из состояния q_i эквивалентно множеству, состоящему из перехода по умолчанию в состояние q_j.*

При построении НДКА с задержками учитывается такая характеристика, как “самый длинный δ-путь” — самый длинный путь, состоящий исключительно из переходов по умолчанию. При выборе автомата предпочтение отдается автомату с минимальной данной характеристикой из эквивалентных НДКА с задержками. При этом эквивалентность НДКА_1 с задержками и НДКА_2 с задержками это выполнимость для любой строки x следующих условий $\varphi_{1i}(x) = \varphi_{2j}(x)$, где φ_1, φ_2 —функции переходов НДКА_1 и НДКА_2 соответственно, и эквивалентность множеств принимающих состояний НДКА_1 и НДКА_2.

Алгоритм построения оптимального НДКА с задержками состоит из нескольких стадий:

1) Построение ненаправленного графа.

Вершины графа соответствуют состояниям НДКА, а ненаправленные ребра, соединяющие каждые два (пусть q_m и q_k) состояния, имеют вес равный числу переходов по непустому символу a_i из алфавита , для которых $\varphi(a_i, q_m) = \varphi(a_i, q_k)$. Ребра с наибольшим весом являются потенциальными кандидатами на переход по умолчанию при построении НДКА с задержками, вес ребра $w_{q_m q_k} - 1$ сообщает, на сколько можно сократить количество переходов при выборе в качестве перехода по умолчанию перехода между состояниями q_m и q_k.

2) Построение остовных деревьев алгоритмом Крускала с ограничением на длину δ-пути (см. далее).

При выборе графа накладывается ограничение на появления циклов, состоящих из переходов по умолчанию, и обеспечивающих бесконечность самого длинного δ-пути. Очевидно, что построенный из ненаправленного графа НДКА с задержками в общем случае не единственен, а число возможных НДКА с задержками в крайнем случае достигает числа всех направленных графов без циклов при заданном числе вершин.

Из полученных ненаправленных графов строят множество остовных деревьев с использованием алгоритма Крускала (Kruskal algorithm), требующего $O(n2\log n)$, итерационно при каждой вершине графа добавляя в остовное дерево ребро с максимальным весом, если дочерняя вершина уже не принадлежит остовному дереву [24]. Длина δ-пути является одним из параметров, ограничивающим эффективность сжатия и скорости работы НДКА с задержками, поэтому при построении дерева накладывается ограничение на его диаметр. Корнем дерева выбирается вершина графа, являющаяся серединой диаметра дерева, а направления переходов по умолчанию выбираются в сторону корня дерева.

Построение остовного дерева, удовлетворяющего ограничению на максимальный диаметр, является NP-трудной задачей.

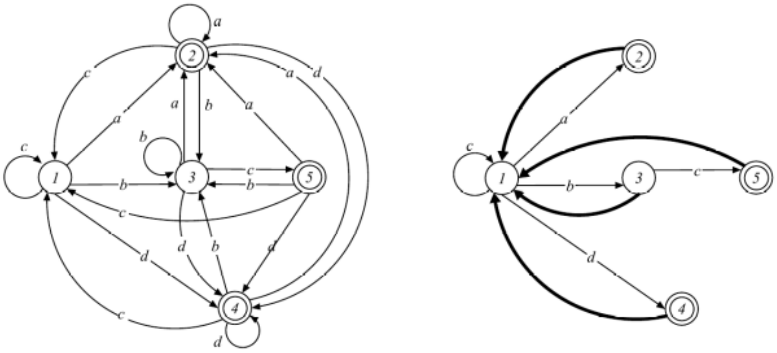

Рис. 2. Примеры НДКА (слева) и НДКА с задержками (справа), принимающих язык $L = \{a^+ \cup b^+c \cup c^*d^+\}$. Источник: [1].

Как и НДКА Ахо-Корасик с переходами в случае ошибки НДКА с задержками позволяет сократить таблицу переходов, при этом не затрагивая число состояний. Применение такого подхода авторами статьи к сокращению числа переходов, требуемого для реализации ДКА, через конструирование автомата с задержками на практике (на базах Cisco System (2008), Snort (2008), Bro NIDS (2008)) показало, что сокращение числа переходов может достигать 99 % от начального числа переходов [1, 22, 23].

2.1.4. δ-КА — НДКА с задержками и ссылкой на родительское состояние [25]

Продолжением идеи НДКА с задержками (D2FA) является сокращение таблицы переходов, и тем самым уменьшение объема необходимой памяти. Идея Фикара (Ficara) и соавторов модификации D2FA, названная δ-КА, заключается в том, что для каждого состояния автомата в таблице переходов хранятся не все значения переходов, а лишь те, которые отличаются от родительского состояния (при его наличии). Так, авторы δ-КА исходили из следующих предположений, основанных как на особенностях построения остовного дерева для D2FA, так и на эмпирических данных баз сигнатур систем обнаружения вторжений:

- большинство переходов по умолчанию НДКА с задержками имеют направление в состояния, близкие к начальному;
- большинство переходов по одному и тому же входящему символу идут в направлении одного и того же состояния автомата [25].

Авторы наглядно модифицируют пример автомата, изначально приведенный авторами НДКА с задержками (рис. 3). δ-КА обладает тем же недостатком, что и его предшественник ДКА с задержками (D2FA) — задержкой, которая обоснована тем, что при отсутствии в таблице состояний переходов по определенным символам для данного состояния, требуется перейти к участку таблицы, описывающего родительское состояние. Последнее необходимо повторять столько раз, пока в таблице переходов не встретится переход по соответствующему символу в родительском состоянии. Так, в D2FA, состояние, где требуется переход по умолчанию приводит к задержке равной количеству требуемых переходов. Для такого же состояния в δ-КА задержка будет соответствовать количеству тактов, необходимому для перехода в таблице переходов в родительское состояние, обеспечивающее данный переход по символу. Так, на рисунке 4.b для автомата D2FA переход по символу "a" из состояния 5 требует задержки в один такт, обеспечивающей переход из состояния 5 в состояние 1, и далее в состояние 2. Для автомата δ-КА на рисунке 4.c. для перехода по символу "a" требуется задержка в 2 такта — в таблице состояний из состояния 5 требуется переход к записи с родительским состоянием 3, далее в состояние 1, где и хранится переход по символу "a".

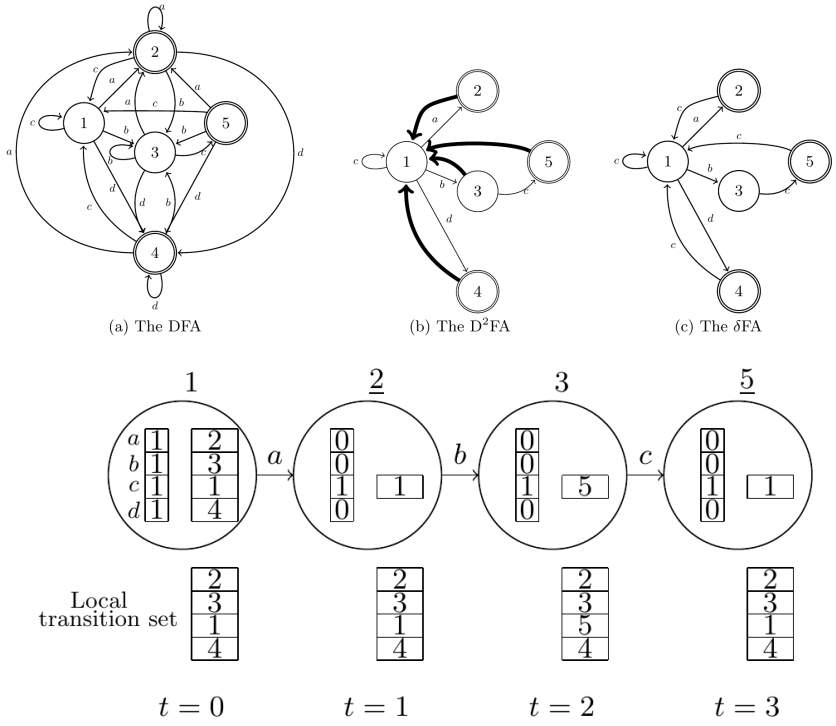

Рис. 3. Примеры ДКА (верх, a), ДКА с задержками (верх, b) и δ-КА (верх, c), принимающих язык $L = \{a^+ \cup b^+c \cup c^*d^+\}$, и таблица переходов для δ-КА (низ рисунка). Источник: [25].

2.1.5. Алгоритм пространственной и временной амортизации ДКА [4]

Авторы Микела Бечи и Патрик Кроули придумали алгоритм (в англоязычной литературе названый Amortized time/bandwidth overhead DFAs, а сокращенно — A-ДКА), который использует избыточность переходов распознающего регулярный язык конечного автомата. Избыточность была замечена авторами эмпирически и заключалась в том, что большая часть состояний ДКА, построенного на наборах данных сигнатурных баз систем обнаружения вторжений, имела сходные наборы исходящих переходов [4]. Ранее это было отмечено и в работе Кумара и соавторов [1]. В своей работе авторы A-ДКА усовершенствовали решение ДКА с задержками (D2FA), предложенное Кумаром и соавторами в 2006 году.

При разработке алгоритма Микела Бечи и Патрик Кроули используют глубину состояния. Теоретически алгоритм построения A-ДКА основан на леммах:

Лемма 2. *Если ни один из переходов по умолчанию в D2FA не ведет из состояния с глубиной d_i в состояние с глубиной d_j, где $d_j \leq d_i$, то любая строка длиной N потребует обработки не более $2N$ обходов состояний.*

Лемма 3. *Если все переходы по умолчанию в D2FA приводят из состояния глубины d_i к состоянию глубины d_j, где $d_j \leq d_{i-k}$, $k \in \mathbb{N}$, то для любой строки длины N потребуется не более $N(k+1)/k$ переходов для обработки.*

Таким образом, проблема построения A-ДКА в соответствии с леммами может быть сформулирована как проблема поиска максимального остовного дерева на ориентированном графе (в оригинале — space reduction graph)(См Algorithm 2). Максимальное остовное дерево приводит к самому высокому уровню сжатия НДКА с задержками — A-ДКА. Согласно леммам авторов корнем этого дерева всегда выбирается начальное состояние соответствующего графу ДКА. Так как при построении направление ребер обращается к корню, то в таком графе отсутствуют ориентированные циклы, и поиск максимального остовного дерева может быть осуществлен за $O(n^2)$, где n — число состояний начального ДКА [4].

Алгоритм построения A-ДКА предполагает обход ДКА в ширину, что позволяет вычислять и переходы по умолчанию, и глубину состояний за один проход. Фактически, алгоритм сводится к тому, чтобы каждое состояние переходило по переходу по умолчанию в состояние, имеющее наибольшее общее количество исходящих переходов и обладающее меньшей глубиной.

Алгоритм A-ДКА Микелы Бечи и Патрика Кроули в худшем случае использует $2N$ переходов и $N(k+1)/k$ переходов для обработки строки длиной N, где k — наперед заданное значение, $k \in \mathbb{N}$. Показано, что в общем случае A-ДКА дает улучшение сжатия по сравнению с НДКА с задержками (D2FA). В то время как НДКА с задержками (D2FA) балансирует объем и пропускную способность памяти, алгоритм A-ДКА направлен на достижение постоянной границы на уровне $2N$ в худшем случае. Фактически, $2N$ переходов сопоставимо с реализацией НДКА с задержками (D2FA) с минимальным диаметром, равным двум, что практически мало применимо. Что касается асимптотики временной сложно-

Algorithm 2 Алгоритм построения A-ДКА.

Вход: 1) Граф $G = \{e, v, d\}$, v — вершины которого соответствуют ДКА, распознающему заданный язык, с присвоенным каждой i-ой вершине значения глубины i-ого состояния — $d(v_i)$, $e = \varnothing$ 2) Значение k, $k \in \mathbb{N}$. Состояния хранятся в виде сортированного по возрастанию глубины состояний списка — S_d. Выход: Ориентированный граф $G = \{e, v\}$, $m(v_i, v_j)$ — число общих переходов v_i и v_j(начальное значение соответсвует 0).

for v_i из S_d **do**
  for v_j из S_d при условии $d(v_j) < d(v_i)$ **do**
    if $d(v_i) - d(v_j) \geq k$ **then**
      Посчитать m_{ij}
      if $m(v_i, v_j) = max(m(v_i, v_j)|v_j \in \cup_{j=1}^{n} v_j)$ **then**
        Построить переход по умолчанию e_{ij}
      end if
    end if
  end for
end for

сти, алгоритмы построения D2FA и A-ДКА имеют временные границы $O(n^2 log_2 n)$ и $O(n^2)$ соответственно [4].

2.2. Методы сжатия с использованием группировки регулярных выражений

Данные методы объединяет стратегия не подстроиться под начальную структуру автомата, распознающего язык, а выделить в языке независимые (или условно независимые) подмножества. Данное выделение подмножеств позволяет построить ДКА для каждого подмножества и объединить полученные ДКА в единый НДКА, сокращая при этом требования к памяти.

2.2.1. Алгоритм сжатия ДКА, строящий мультиавтомат на основе группировки взаимодействующих регулярных выражений [2]

Построение мультиавтомата является принципиально другим подходом в отношении сжатия распознающего язык конечного автомата, по сравнению с алгоритмами, которые были рассмотрены выше.

Алгоритм построения мультиавтомата базируется на том, что регулярные выражения имеют общие части, и работает через выборочное группирование регулярных выражений на основе их "взаимодействия". Регулярные выражения взаимодействуют, если ДКА, принимающий язык, являющийся объединением этих выражений, имеет больше состояний, чем сумма состояний ДКА, принимающих языки, описанные данными регулярными выражениями в отдельности [2].

Данный алгоритм ищет регулярные выражения, которые взаимодействуют, и объединяет в группы не взаимодействующие регулярные выражения. Количество групп задается до начала построения и зависит от заданного объема памяти, используемой автоматом. Поиск же наилучшей по памяти группировки регулярных выражений — NP-трудная задача. Таким образом, на основе m выделенных групп регулярных выражений строится m соответствующих ДКА. Соответствующие ДКА объединяются в один НДКА параллельно. Таким образом, мультиавтомат (multistride DFA) — это объединение нескольких ДКА, каждый из которых построен для распознавания слов, описываемых одной группой регулярных выражений, тогда как язык принимаемый мультиавтоматом, описывается суммой групп этих регулярных выражений. Группировка регулярных выражений позволяет использовать многопоточность для параллельной проверки принадлежности слова регулярному языку, используя в каждом потоке проверку по каждой выделенной группе взаимодействующих регулярных выражений. На практике, в системах обнаружения вторжений и системах анализа сетевого трафика авторами данного алгоритма достигнуто снижение необходимой памяти для реализации мультиавтомата на порядок при сравнении с классическим ДКА. Так, верхняя граница числа состояний такого мультиавтомата — $O(m2^l)$, где m — число регулярных выражений R_i, из языка $\cup_{i=1}^{m} R_i$, где $l = max(|R_i|)$ [2].

2.2.2. Алгоритм сжатия ДКА, строящий мультиавтомат на основе группировки взаимодействующих регулярных выражений с оценкой взаимодействия [3]

Рохрер (Rohrer) и соавторы дополнили предыдущий жадный алгоритм оценкой степени взаимодействия регулярных выражений. Так, для каждых регулярных выражений R_i и R_j рассчитывали значение $I_{ij} = S_{ij} - S_i - S_j$, где S_i — мощность алфавита состояний ДКА, соответствующе-

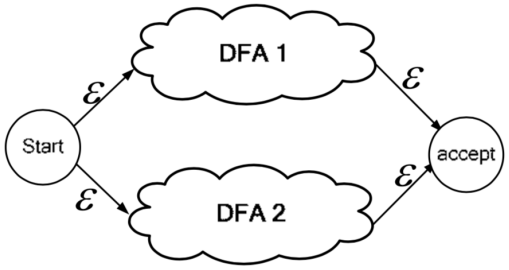

Рис. 4. Строение ДКА-мультиавтомата, состоящего из двух ДКА ($DFA1$ — соответствует регулярному выражению R_1, $DFA2$ — соответствует регулярному выражению R_2) и принимающего язык $L = R_1 \cup R_2$. Источник: [2].

го R_i, S_j — мощность алфавита состояний ДКА, соответствующего R_j, S_{ij} — мощность алфавита состояний ДКА, соответствующего $R_i \cup R_j$.

Авторами показано, что для k групп n регулярных выражений мощность состояний мультиавтомата S_{nR_i}, распознающего $\cup_i^n R_i$ будет оцениваться как:

$$O(\sum_{i=1}^{n} S_i + \sum_{l=1}^{k} \sum_{i=2, i \in R_l}^{n} \sum_{j=1, j \in R_l}^{i-1} I_{ij})$$

Задача поиска оптимального разбиения сводится к задаче минимизации S_{nR_i}, с учетом фиксированной $\sum_{i=1}^{n} S_i$. Таким образом, задача остается NP-трудной. Учитывая, что число возможных разбиений на группы — $O(k^n)$, время поиска разбиения достаточно большое для практического применения [3].

2.2.3. Алгоритм сжатия ДКА на основе группировки взаимодействующих регулярных выражений с оценкой вклада в экспоненту [26]

Данный жадный алгоритм является продолжением предыдущего и также базируется на взаимодействии регулярных выражений. Авторы сводят проблема группировки регулярных выражений к NP-трудной задаче максимального разбиения графа на k частей и предлагают эвристический алгоритм, называемый одношаговым жадным алгоритмом сжатия

ДКА, для решения этой NP-трудной задачи. Идея алгоритма и его название возникли из наблюдения того, что не все регулярные выражения вносят одинаковый вклад в экспоненту. Как мы уже упоминали, особое внимание в проблеме экспоненциального взрыва при построении конечного автомата, соответствующего множеству регулярных выражений, уделяется использованию в регулярных выражениях сочетаний ".+" и ".∗" и "считающих" выражений, типа ".{n}" и "[a_i]{n}". Авторы предложили использовать средний вклад выражения в проблему экспоненциального взрыва и сформулировали следующую EP_i метрику для регулярных выражений:

$$EP_i = \frac{1}{n}\sum_{j=1}^{n}\frac{I_{ij}}{S_j}, i \neq j,$$

где используются обозначения предыдущего раздела. Чем больше значение EP_i, тем больше вклад данного регулярного выражения в проблему экспоненциального взрыва [26].

Авторы представили инициальный алгоритм распределения регулярных выражений по группам на основе полученных для каждого регулярного выражения значения EP. Алгоритм заключается в нисходящей сортировке регулярных выражений в отношении значений EP и последующем добавлении регулярного выражения с минимальным значением EP в группу с суммарным для группы значением EP (см. Algorithm 3). Авторами показано, что данный инициальный алгоритм быстрее для дальнейшей сортировки по группам, чем случайный выбор группы для регулярного выражения. Далее для распределения по группам авторы представили одношаговый жадный алгоритм, который перемещает регулярное выражение в ту группу, которая дает наибольший суммарный выигрыш для задачи минимизации числа состояний автомата, распознающего язык, представленный данными регулярными выражениями из $\cup_{i=1}^{n} R_i$ [26].

3. Заключение

В направлении решения задачи экспоненциального взрыва без добавления специальных структурных элементов особо успешными были некоторые идеи и алгоритмы, ставшие прародителями целых направлений в сигнатурном анализе. Во-первых, к таким находкам относится алгоритм

Algorithm 3 Жадный алгоритм сжатия ДКА на основе группировки взаимодействующих регулярных выражений

1: Начальное состояние формируется инициальным алгоритмом с оценкой вклада регулярного выражения в экспоненту. Считается начальное значение C_0, равное мощности алфавита состояний начального ДКА, полученного после работы инициального алгоритма.
2: **for** R_i из $\cup_{i=1}^{n} R_i$ **do**
3: **for** группа G_j из $\cup_{j=1}^{k} G_j$ **do**
4: Рассчитать C_{ij} для $R_i \in G_j$
5: Сохранить C_{ij}
6: **end for**
7: **end for**
8: **for** R_i из $\cup_{i=1}^{n} R_i$ **do**
9: **if** $C_i j = \max \cup_{i=1}^{n} \cup_{j=1}^{k} C_{ij}$ **then**
10: **if** R_i не в G_j **then**
11: Переместить R_i в G_j
12: Перейти на шаг 2
13: **else**
14: Остановка
15: **end if**
16: **end if**
17: **end for**

Ахо-Корасик с переходами в случае ошибки для поиска подстрок, появление которого породило целую плеяду алгоритмов. Во-вторых, идея построения мультиавтомата на основе группировки взаимодействующих регулярных выражений, которая инициировала направление параллельных вычислений для реализации поиска по сигнатурам. Однако несмотря на приличное число идей оптимизации, алгоритмы группировки взаимодействующих регулярных выражений остаются NP-полной задачей и не гарантируют решение проблемы экспоненциального взрыва в общем случае. Алгоритмы, использующие переходы в случае ошибки, также не решают проблему экспоненциального взрыва в общем случае, тем не менее позволяя значительно сократить потребности в памяти для реализации поиска по сигнатурам на практике.

Список литературы

[1] Kumar S., Chandrasekaran B., Yu F., Crowley P. Turner J., "Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection", ACM SIGCOMM Computer Communication Review, **36**:4 (2006), 155–164.

[2] Yu F., Chen Zh., Diao Y., Lakshman T.V., Katz R.H., "Fast and memory-efficient regular expression matching for deep packet inspection", Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems. — ACM., 2006, 93-102.

[3] Rohrer J., Atasu K., van Lunteren J., Hagleitner C., "Memory-efficient distribution of regular expressions for fast deep packet inspection.", Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis. — ACM., 2009, 147-154.

[4] Becchi M., Crowley P., "A-DFA: A Time- and Space-Efficient DFA Compression Algorithm for Fast Regular Expression Evaluation", Transactions on Architecture and Code Optimization. — ACM., **10**:1 (2013), 4.1-4.4.

[5] Liu C. and Wu J., "Fast Deep Packet Inspection with a Dual Finite Automata.", IEEE Transactions on Computers., **62**:2 (2013), 310-321.

[6] Bernadotte A., Kumar R., Winblad B., Pavlov P.F., "In silico identification and biochemical characterization of the human dicarboxylate clamp TPR protein interaction network", FEBS Open Bio, **8** (2018), 1830–1843.

[7] Бернадотт А., Галатенко А.В., "Аппаратная конструкция для решения проблемы экспоненциального взрыва для одного класса регулярных языков", *Интеллектуальные системы. Теория и приложения*, **23**:4 (2019), 121–146.

[8] Хопкрофт Дж., Мотвани Р., Ульман Дж., *Введение в теорию автоматов, языков и вычислений*, Вильямс, Москва, 2017.

[9] Rabin M.O., Scott D., "Finite automata and their decision problems", *IBM J. Research and Development*, **3**:2 (1959), 115–125.

[10] Кудрявцев В. Б., Алешин С. И., Подколзин А. С., *Введение в теорию автоматов*, Наука, Москва, 1985.

[11] Кудрявцев В. Б., Гасанов Э. Э., Подколзин А. С., *Основы теории интеллектуальных систем*, Макс Пресс, Москва, 2016.

[12] Лупанов О.Б., "О сравнении двух типов конечных источников", Проблемы кибернетики, 1963, № 9, 321–326.

[13] Документация для Perl-совместимых регулярных выражений, http://perldoc.perl.org/perlre.html.

[14] Александров Д. Е., "Об оценках автоматной сложности распознавания класса регулярных языков", *Интеллектуальные системы. Теория и приложения*, **18**:4 (2014), 121–146.

[15] Александров Д. Е., "Об уменьшении автоматной сложности за счет расширения регулярных языков", *Программная инженерия*, 2014, № 11, 26–34.

[16] Александров Д. Е., "Об оценках мощности некоторых классов регулярных языков", *Дискретная математика*, **27**:2 (2015), 3–21.

[17] Smith R., Estan C., Jha S., "XFA: Faster Signature Matching with Extended Automata", IEEE Symposium on Security and Privacy, 2008, 187-201.

[18] Александров Д. Е., "Эффективные методы реализации проверки содержания сетевых пакетов регулярными выражениями", *Интеллектуальные системы. Теория и приложения*, **18**:1 (2014), 37–60.

[19] Александров Д. Е., *Сложность распознавания принадлежности слова регулярному языку в системах обнаружения вторжений*, Диссертация на соискание ученой степени кандидата физико-математических наук, Москва, 2015.

[20] Aho A.V., Corasick M.J., "Efficient string matching: an aid to bibliographic search", Communication of the ACM, **18**:6 (1975), 333-340.

[21] De La Briandais R., "File Searching Using Variable Length Keys", Proceedings of Western Joint Computer Conference. — ACM., 1959, 295-298.

[22] Kumar S., Chandrasekaran B., Turner J., Varghese G., "Curing regular expressions matching algorithms from insomnia, amnesia, and acalculia", ANCS '07 Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems , 2008, 155–164.

[23] Kumar S., *A Thesis on Accelerationof Network Processing Algorithms*, Doctor of Science Thesis, Washington University, Saint Louis, Missouri, 2008.

[24] Kruskal J. B., "On the shortest spanning subtree of a graph and the traveling salesman problem", Proceedings of the American Mathematical Society, **7**:1 (1956), 48-50.

[25] Ficara, D., Giordano, S., Procissi, G., "An improved DFA for fast regular expression matching", ACM SIGCOMM Computer Communication Review, **38**:5 (2008), 29–40.

[26] Xu C., Su J., Chen S., "Exploring efficient grouping algorithms in regular expression matching", PLoS ONE, **13**:10 (2018), e0206068.

Finite automaton modification by using compression algorithms

Bernadotte A.

Аннотация

The question of whether a word belongs to a given regular language finds its application in areas where it is relevant to search for certain patterns in data of different nature. Growth of the deterministic finite automaton (DFA) states number which is exponential in number of regular expressions of the given language is still a real problem. In this article, we take a look at modifying a finite automaton by using compression algorithms. These approaches modify the finite automaton without changing the language and without adding additional structural elements.

Keywords: DFA, NDF, regular language, exponential blowup, compression algorithm.