doi:10.3969/j.issn.1000-436x.2014.10.012

新颖的正则 NFA 引擎构造方法

敬茂华[1,2,3]，杨义先[2,4]，汪韬[1]，辛阳[3,4]

（1.东北大学秦皇岛分校 计算机与通信工程学院，河北 秦皇岛 066004；2.东北大学 信息科学与工程学院，辽宁 沈阳 110819；
3. 云安全技术北京市工程实验室，北京 100082；4. 北京邮电大学 信息安全中心，北京 100876）

摘 要：提出了一种新颖的正则 NFA 引擎构造方法——PFA 构造法。PFA 构造法包括 3 个主要算法：预处理算法、解析树编码算法和基于编码树的 NFA 构造算法。采用 PFA 构造法能够构造出只含有一个开始状态和一个终止状态的规模更小的 NFA，称其为 NFA_p。NFA_p 的规模与正则表达式组的长度线性相关，较 Thompson 自动机、后跟自动机、位置自动机以及部分派生自动机的规模都要小，是 Thompson NFA 的 1/3，比已经接近最优的后跟自动机构造法所获得的 NFA 还要小。

关键词：深度分组检测；模式匹配；正则表达式；有穷自动机；构造算法

中图分类号：TP393 **文献标识码**：A **文章编号**：1000-436X(2014)10-0098-09

Novel NFA engine construction method of regular expressions

JING Mao-hua[1,2,3], YANG Yi-xian[2,4],WANG Tao[1],XIN Yang[3,4]

(1. School of Computer and Communication Engineering, Northeastern University at Qinhuangdao, Qinhuangdao 066004, China;
2. College of Information Science and Engineering, Northeastern University, Shenyang 110819, China;
3. Beijing Engineering Lab for Cloud Security, Beijing 100082, China;
4. Information Security Center, Beijing University of Posts and Telecommunications, Beijing 100876, China)

Abstract: A novel method for constructing smaller non-deterministic finite automata (NFA) engine from given regular expressions named PFA was proposed. There are three main algorithms in PFA, the pretreatment algorithm, the coding parser tree algorithm and the NFA construction algorithm based on the coded binary tree. The smaller NFA named NFA_p with only one start state and one final state can be obtained by using PFA construction method. NFA_p have linear size in terms of the size of given regular expressions. It is the smallest NFA comparing with current methods like Thompson NFA, follow automata, position automata and partial derivatives automata. The size of NFA_p is one third of Thompson's and it is smaller than the size of follow automata whose size has nearly closed to optimal.

Key words: deep packet inspection; pattern matching; regular expression; finite automata; construction algorithm

1 引言

随着网络技术特别是云计算的迅猛发展，安全攻击的技术和手段越来越多样化并逐渐由网络层和传输层转向应用层。针对应用层实施的攻击实现简单、效果明显，且看起来与正常的业务流并无二异，这使现有的基于分组头检测的防火墙过滤技术以及入侵检测技术都无法有效地对其进行检测和防御，基于深度分组检测（DPI, deep packet inspection）的入侵检测技术应运而生。典型的入侵检测通过建立攻击特征模式集，采用模式匹配技术来实时地发现攻击。DPI 系统不但检测数据分组头，而且深入到数据分组的有效载荷中对内容进行识别、分析、分类和统计，因而能够有效地实现对针对应用层所实施的安全攻击的检测。应用层上各种复杂协议以及多样化的攻击方式使提取准确的攻

收稿日期：2013-07-16；**修回日期**：2014-01-23

基金项目：国家自然科学基金资助项目（61100021，61121061，61202447）；河北省自然科学基金资助项目 (F2012501014)；河北省教育厅自然科学指导基金资助项目(Z2010215)

Foundation Items: The National Natural Science Foundation of China (61100021, 61121061, 61202447); The Natural Science Foundation of Hebei Province (F2012501014); The Hebei Provincial Education Department Natural Science Guide Project (Z2010215)

击特征变得越来越困难，这种情况导致基于精确模式匹配的各种模式匹配方法无法有效地实现检测。正则表达式(RE, regular expression)[1] 灵活、高效且表达力强大，在编译器构造、协议分析、网络流量实时处理以及模式匹配等众多领域得到了广泛的应用。基于正则表达式的多模式匹配算法逐渐代替了基于字符串的精确模式匹配算法，已经成为模式匹配的首选，Snort 入侵检测系统[2]、Bro 入侵检测系统[3]、Linux 的应用协议分类器 L7-filter[4]以及.NET 等都提供了对正则表达式的支持。

基于正则表达式的模式匹配使用自动机（称为正则引擎）来执行匹配，可以是非确定性有穷自动机(NFA, non-deterministic finite automata)，也可以是确定性有穷自动机(DFA, deterministic finite automata)，或者是 NFA 和 DFA 的混合体。根据所采用的引擎的不同，其匹配技术大致可分为 3 种：1) 基于 NFA[5~7]；2) 基于 DFA[8,9]；3) NFA/DFA 混合模式（hybrid finite automata）。NFA 占用的存储空间较小，但在最坏情况下处理每个字符需要 $O(n)$次访问状态表（n 为 NFA 的状态数目），因此匹配速度较 DFA 而言非常慢；且一个 NFA 一般只对应一条正则表达式，如果要支持多正则表达式匹配，则需要多个 NFA 并发工作，其性能将随着正则表达式数量的增加而迅速下降[10]。DFA 匹配时每读入一个字符只需维护一个激活状态即可，因此处理速度快，对于每个输入字符仅进行一次状态转换，性能高效且稳定，但存在状态空间严重膨胀问题[11]。以 NFA 的存储空间和 DFA 的匹配性能来实现正则表达式的匹配是正则表达式模式匹配研究领域一直以来所追求的目标。

正则引擎实现模式匹配的经典方法通常是先将正则表达式解析成表达式树，然后将解析树转化为 NFA，进一步可再转化为 DFA，这个过程称为自动机构造。一般从 2 个方面来衡量构造算法的优劣，一是所获得的自动机的规模（size, 状态数和状态转换弧数之和）；另一个则是算法本身的时间空间复杂度。构造正则表达式的 NFA 的方法有多种，实践中最常用的 NFA 构造方法是 Thompson 构造法[12]和 Glushkov 构造法[13]。Thompson 自动机的缺点是存在 ε-转移，即空转移，这使在进行模式匹配的时候不读入字符（或读入空字符串 ε）时都会发生状态转移。空转移不仅导致了空匹配，还使系统所维护的活动状态中存在大量的无效的状态和弧转换，即不能产生有效匹配的状态及状态转移，这种现象称为无效匹配，它极大地影响了正则引擎的执行效率。Glushkov 构造法虽然解决了 Thompson 构造法所带来的无效匹配问题，但它所获得的 NFA 的 size 却不再是线性相关的，而是 $O(n^2)$，这在有数以千计条正则表达式（比如 Snort）的实际系统中，将消耗 TB 级内存且导致系统效率急剧下降。因此，研究的目标是找到一种有效的 NFA 正则引擎构造方法，不但保持其 size 与正则表达式长度线性，并且尽可能得小，且能有效地避免无效匹配现象，即尽可能地减少 NFA 中的空弧转移的数量。

基于上述目标，针对 Thompson 构造法含有大量 ε–转移的情况，提出了 PFA 构造法。PFA 构造法其本质是对 Thompson 方法的一种改进。该方法与 Thompson 构造法一样具有线性时间复杂度，所获得的 NFA 的规模也是线性的，构造所获得的 NFA 包含极少的空弧转移，能够有效地减少无效匹配。

2　相关工作

最早最经典的构造方法是 Thompson 提出来的。Thompson 构造法所获得的 ε-NFA 规模与正则表达式的长度线性相关[14]。Soisalon-Soininen [15]在 Thompson 方法的基础之上提出了构造规模更小的 ε-NFA 的方法。Gloshkov 等人[16]提出了位置自动机（Position automata）的概念，并在此基础上提出了著名的 Gloshkov 构造法[17]。Antimirov 于 1996 年通过将 Brzozowski 提出的词派生（word derivative）概念一般化为与 NFA 相关的部分派生（partial derivative）的概念，并在此基础上提出了一种 NFA 构造方法——部分派生自动机（partial derivatives automata）[18]构造法。Ilie 和 Yu 共同提出的后跟自动机(follow automata)[19]构造算法能够获得规模比 Thompson 自动机更新的 NFA，其算法复杂度为 $O(n^2)$。

近年来，解决 DFA 状态空间膨胀问题的研究主要有减少状态数目[20~22]和减少状态转换[23~26]2 类。文献[27]利用规则分类和规则改写的方法来进行简化处理，遗憾的是其改写前后不能完全等价，极大地限制了其使用范围。文献[28]通过提取子状态和父状态相同的元素来减少状态转换的数目，虽然在空间上能得到压缩，但匹配的时间效率受到了很大的影响。文献[29]提出的 XFA 方法能够比较显著地解决状态空间膨胀问题，但同样时间效率降低且无法处理多字符

重复的情况。文献[30]提出一种簇分割算法，实验表明其能够达到较好的压缩效果，但却没实现状态数的压缩，也没有针对由正则表达式转换为等价 DFA 的优化处理。文献[31]提出了一种分组算法 GRELS 来解决分组问题，其分组结果明显优于其他分组算法，但其分组的所基于的“冲突”前提在一定程度上限制了算法的实用性。此外，基于 FPGA 结构来实现正则表达式匹配也是其主要研究方向之一。

3 PFA 构造法

3.1 相关概念及定义

正则表达式能够描述所有通过对某个字母表上的符号应用各种运算而得到的语言。正则表达式描述的语言是正规语言，简称正规集或正则集。正则表达式以及其所表示的正规集的递归定义如下。

定义 1 正则表达式

设字母表为 Σ，辅助字母表 $\Sigma' = \{\varepsilon, \varnothing, +, \cdot, *, (,)\}$。

1) ε 和 $\varnothing$ 都是 Σ 上的正则表达式，所描述的正规集分别为 $\{\varepsilon\}$ 和 $\varnothing$；

2) 对于任意单个输入符号 $a \in \Sigma$，则 a 是 Σ 上的一个正则表达式，所描述的正规集为 $\{a\}$；

3) 设 r_1 和 r_2 均为 Σ 上的正则表达式，它们所描述的正规集分别为 $L(r_1)$ 和 $L(r_2)$，则 (r_1)，$r_1 + r_2$，$r_1 \cdot r_2$（或 $r_1 r_2$）以及 r_1^* 也都是正则表达式，所描述的正规集分别为 $L(r_1)$，$L(r_1) \cup L(r_2)$，$L(r_1)L(r_2)$ 以及 $L(r_1)^*$；

4) 仅由有限次上述规则定义的表达式才是 Σ 上的正则表达式，仅由这些正则表达式所描述的集合才是 Σ 上的正规集。

在上述定义中包含 3 种运算：或（+）运算、连接（·）运算和闭包（*）运算，其中，闭包运算的优先级最高，连接运算次之，或运算最低，且均遵循左结合律。

定义 2 后缀正则表达式

所谓后缀正则表达式实质上是通过将正则表达式的操作符置于其操作对象之后多得到的一种等价表示形式，其递归定义如下。

设字母表为 Σ，辅助字母表 $\Sigma' = \{\varepsilon, \varnothing, +, \cdot, *\}$。

1) ε 和 $\varnothing$ 都是 Σ 上的正则表达式，所描述的正规集分别为 $\{\varepsilon\}$ 和 $\varnothing$；

2) 对于任意单个输入符号 $a \in \Sigma$，则 a 是 Σ 上的一个正则表达式，所描述的正规集为 $\{a\}$；

3) 设 r_1 和 r_2 均为 Σ 上的后缀正则表达式，它们所描述的正规集分别为 $L(r_1)$ 和 $L(r_2)$，则 $r_1 r_2 +$，$r_1 r_2 \cdot$ 和 $r_1{}^*$ 也都是后缀正则表达式，其中 $r_1 r_2 \cdot$ 中的连接运算符不能省略，它们所描述的正规集分别为 $L(r_1) \cup L(r_2)$，$L(r_1)L(r_2)$ 以及 $L(r_1)^*$；

4) 仅由有限次上述规则，按照原正则表达式中每个运算符实际的运算顺序所得到的表达式才是 Σ 上的后缀正则表达式，仅由这些后缀正则表达式所描述的集合才是 Σ 上的正规集。

定理 1 对于任意给定的正则表达式 r 所对应的后缀正则表达式 r'，根据后缀正则表达式的定义可得出如下 3 个结论。

1) 后缀正则表达式中的运算对象从左至右的出现顺序与原正则表达式完全一致；

2) 后缀正则表达式中的运算符从左至右的出现顺序与原正则表达式中实际的运算顺序完全一致；

3) 后缀正则表达式中的连接运算符是显式的且不再有小括号出现。

证明 显然，由后缀正则表达式的定义可知结论 1）和结论 2）的正确性。对于结论 3），前者是由定义规定的，对于后者，由定义 1 显然可知，后缀正则表达式即为原正则表达式的逆波兰形式，在表达式的逆波兰形式中是没有小括号的，因为运算符的出现顺序已经明确地表达了实际的运算顺序。

定理 2 后缀正则表达式与正则表达式完全等价，描述的是同一正规集。

证明 要证明 2 种表达方式的等价性，可以从 2 个角度来证明。一个角度是运算的等价性，另一个角度是所描述的集合的等价性。

首先考虑运算的等价性。由于后缀正则表达式实质上就是将中缀形式的正则表达式按照逆波兰式的构造方法转换得到的，这仅仅是同一运算序列的 2 种不同表现形式而已，显然其运算是等价的，因此 2 种表示形式所描述的语言集也是等价的。

再考虑 2 种不同形式所描述的集合的等价性。只要能够证明后缀正则表达式的运算保持了正规集运算的正则性和封闭性，即可证明其等价性。由文献[32]可知，首先，正规语言是一个抽象语言族（简称 AFL，即在并、连接、闭包、ε 无关同态映射、正则集的交、逆同态这 6 种基本操作下仍然封闭的语言族）。如果一个语言族在与正规集的交、逆同态和 ε 无关同态下仍然封闭，则该语言族称为 trio。如果一个语言族在所有同态、逆同态及正则集

的交下任然闭合，那么称它为 full trio。正规语言既是 full AFL，又是 full trio，且可得出如下定理[32]。

定理 3　正规语言集在并、交、补、连接、耦合闭包和逆运算操作下都是封闭的。

定理 4　正规语言在同态运算下是封闭的。

定理 5　设 $h:\sum_1^* \to \sum_2^*$ 是一个同态映射。如果 $L' \subseteq \sum_2^*$ 是正则的，那么 $h^{-1}(L')=L \subseteq \sum_1^*$ 也是正则的。

由以上定理可知，后缀正则表达式的运算同样具有封闭性和正规性，所生成的语言亦为正规集。得证。

定义 3　正则解析树

正则解析树 *RETree* 是后缀正则表达式的语法树（parser tree），具有下述性质。

1) 树中任意一个非叶子节点都表示了某一个运算；

2) 树中的每个叶子节点都对应了一个输入符号 $a \in \Sigma$ 或 ε；

3) 任意一个节点所构成的子树都对应了正则表达式中的一个子串（sub-string），即一个子表达式。

定义 4　正则后缀树

正则后缀树 *RPTree* 是一种编码正则解析树（coded *RETree*），是具有如下性质的二叉树。

1) 树中的每个叶子节点的标号均为某个输入符号 $a \in \Sigma$ 或 ε；

2) 树中的非叶子节点的标号要么是或运算符（+），要么是闭包运算符（*），要么是一个正整数 $n(n=1,2,3,\cdots)$。

通过对正则解析树进行编码得到的正则后缀树，可以获知与正则表达式等价的 NFA 的绩效状态集合状态转换集，从而获得更小的 NFA。称对正则解析树进行编码的过程为构造正则后缀树，其算法将在 3.2 节中介绍。

有限自动机（FA, finite automata）也称为有穷自动机，能够准确地识别正规集。正则表达式和正则文法是从描述的角度来表示正规集，自动机是一种识别装置，它从识别角度来表示正规集。有限自动机分为确定的有限自动机 DFA 和不确定的有限自动机 NFA。下面给出 NFA 的定义。

定义 5　非确定性有限自动机 NFA

一个不确定的有限自动机 NFA 是一个五元组，$NFA=(K,\sum,f,S,F)$。其中，K 是一个有穷集，每个元素称为一个状态，即 K 为 NFA 的所有状态组成的集合，称为状态集；$\sum$ 是一个输入符号的有穷字母表，称为输入符号集；f 是一个从 $K\times\sum^* \to K$ 的子集的映像，也就是 NFA 中所有的状态转换弧集；$S \subset K$ 是一个非空初态集；$F \subset K$ 是 NFA 的所有终止状态组成的集合，称为终态集。

通常用带标记弧的有向图表示一个 NFA，图中的节点对应了 NFA 的状态集 K 中的状态，有向边对应了状态转换，边上的标记为输入符号。

将使用 PFA 构造方法所获得的非确定性有限自动机命名为 NFA_p，规定采用正整数来表示每一个状态。下面给出 NFA_p 的定义。

定义 6　非确定性有限自动机 NFA_p

一个 NFA_p 是采用 PFA 算法获得的只有一个初始状态和一个终止状态的 NFA，即 NFA_p 是一个五元组，$NFA_p=(K,\sum,f,S,F)$，其中，K 是一个有穷的非负整数集，每个整数代表了一个状态；$\sum$ 是一个输入符号的有穷字母表，称为输入符号集；f 是一个从 $K\times\sum^* \to K$ 的子集的映像，也就是 NFA_p 中所有的状态转换弧集；$S \in K$ 是唯一的一个初态；$F \in K$ 是 NFA_p 的惟一的一个终态。

3.2　PFA 相关算法

PFA 构造法共包含 4 个步骤：构造后缀正则表达式及预处理（简称预处理）；构造正则解析树；构造正则后缀树；构造更小的 NFA。

3.2.1　预处理

预处理主要完成如下 3 个工作。

1) 通过等价变换进行规范化处理，保证式中的连接运算符的个数与最终获得的 NFA 的极小化状态空间中的状态的个数相同。

2) 将给定的一个或多个正则表达式的中缀形式转换为等价的后缀形式。

3) 根据需要，将多个后缀正则表达式用或运算组合成一个单一的表达式，解决当前实际系统中每一条正则表达式都对应了一个单独的 NFA，从而导致大量 NFA 并行执行时入侵检测系统运行效率指数级下降的问题。

算法 1　预处理算法

输入：正则表达式组 $r_1,r_2,\cdots,r_n$

输出：后缀正则表达式 r'

Begin

step1　//拆分处理：考虑正则表达式组 $r_1,r_2,\cdots,r_n$ 中的第一个正则表达式 r_1：

if $r_1=s+t$　　//其中，s 和 t 分别为子正

则表达式，且 s 中不存在或运算符（小括号中出现或运算符的情况除外）

then $\begin{cases} r_1 = s \\ s = r_{n+1} \end{cases}$，则当前的正则表达式组为 $r_i(i=1,2,\cdots,n,n+1)$

else 不对 r_1 做拆分处理，转入 step2；

step2 //等价变换：考虑才分处理后的正则表达式组 $r_1,r_2,\cdots,r_n$ 或 $r_1,r_2,\cdots,r_n,r_{n+1}$

if r_1 中没有闭包运算

then $r_1 = r_1 \cdot \varepsilon$；

else if $r_1 = s \cdot t^* \cdot x \cdot a \cdot b$； //其中，$s,t,x$ 为子表达式，a,b 为输入符号（即闭包运算后至少有 2 个连续的连接运算）

then $r_1 = r_1 \cdot \varepsilon$；

else 不对 r_1 做等价变换；

step3 //后缀式构造：将正则表达式组 $r_i(i=1,2,\cdots,n)$ 用或运算组合成一条正则表达式

$r = r_1 + r_2 + \cdots + r_n$，

step4 将 r 转换为其后缀形式 r'；

End

例 1 考虑最简单的情况下的算法 1 处理：$r = abx^* cd + efy^* gh$，根据算法 1 得到 $r' = ab \cdot x^* \cdot c \cdot d \cdot \varepsilon \cdot ef \cdot y^* \cdot g \cdot h \cdot +$

3.2.2 构造正则解析树

构造正则解析树的过程就是构造其语法解析树的过程。一个正则表达式的正则解析树描述了这条正则表达式的计算过程，实际上，将正则表达式或正则表达式组转换成其后缀正则表达式的过程，也就是构造正则表达式的过程。

例 2 正则表达式 $r = (a+b)(a^*+ba^*+b^*)^*$ 的后缀正则表达式为 $r' = ab + a^* ba^* \cdot + b^* + ^* \cdot$，其正则解析树 *RETree* 如图 1 所示。

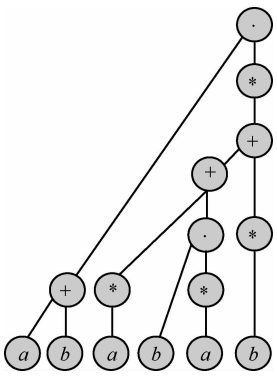


图 1 正则表达式 $r = (a+b)(a^*+ba^*+b^*)^*$ 的正则解析树

例 3 正则表达式组 $r_1 = abx^* cd, r_2 = cdy^* ef$，$r_3 = baab$ 的后缀正则表达式为 $r' = ab \cdot x^* \cdot c \cdot d \cdot \varepsilon \cdot cd \cdot y^* \cdot e \cdot f \cdot + ba \cdot a \cdot b \cdot +$，其正则解析树 *RETree* 如图 2 所示。

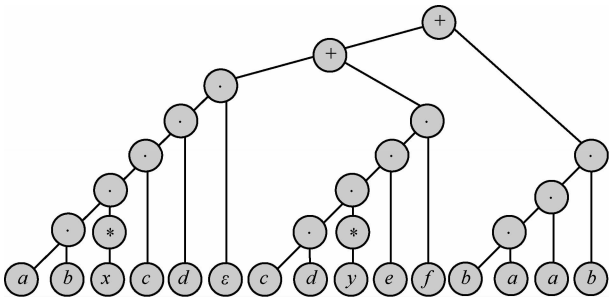


图 2 正则表达式组 $r_1 = abx^* cd, r_2 = cdy^* ef, r_3 = baab$ 的正则解析树

显然，正则解析树与其对应的后缀正则表达式是等价的，它们描述了同一正规集。

3.2.3 编码正则解析树得正则后缀树

本文提出了一种基于正则解析树进行编码来获得其对应的 NFA 的状态空间和弧转移逻辑的算法如下。

算法 2 正则后缀树编码算法

输入：正则解析树 *RETree*

输出：正则后缀树 *RPTree*

Begin

step1 //中序遍历 *RETree*，生成其所有非叶子节点的中序遍历节点序列，对该序列中的所有连接运算符节点 N_i 按其在序列中的顺序采用如下方法用正整数 $n(n=1,2,3,\cdots)$ 进行编码（假设序列中连接运算符的个数为 k：

{$n=1$；

for (i=1，i++，$i \leqslant k$) do

{Code(N_i , n) //用整数 n 对节点 N_i 编码；

if N_i 前面分别是一个闭包运算符和一个连接运算符

then $n = n$；

else $n = n+1$；

}

step2 //设置终止状态：

{ if 树根为已编号的连接运算符节点

then 设置树根所对应的编号状态为终止状态；

else if 树根为弧运算符节点

then 设置根节点左子树中与根节点

层次差最小的编号节点为终止状态；

else 不设置终止状态；

}

End

例 4　根据算法 2 得到的例 2 和例 3 的正则后缀树 *RPTree* 分别如图 3 和图 4 所示。

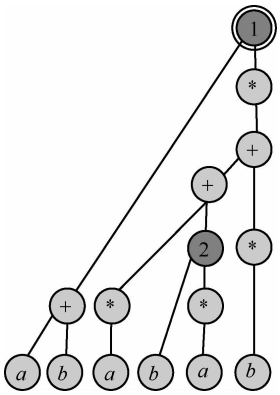


图 3　$r=(a+b)(a*+ba*+b*)*$ 的正则后缀树

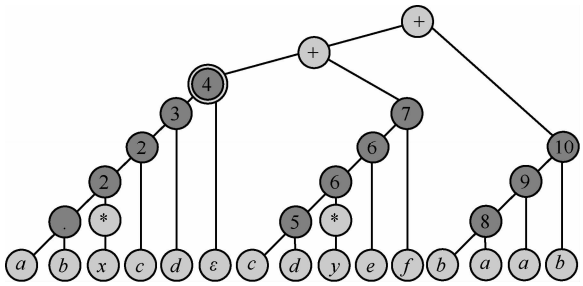


图 4　正则表达式组 $r_1=abx*cd, r_2=cdy*ef, r_3=baab$ 的正则后缀树

3.2.4　依据正则后缀树构造 NFA

在正则后缀树的基础上，提出了依据编码正则后缀树构造 $NFA_p=(K,\Sigma,f,S,F)$ 的算法。假设当前正则后缀树中共有 n 个叶子节点，显然，正则后缀树中所有的叶子节点标记为输入符号 $a\in\Sigma$ 或 ε，将所有叶子节点按从左至右的顺序序列化为一个符号串 $s=a_1a_2\cdots a_n$。

算法 3　NFA_p 构造算法

输入：正则后缀树 *RPTree*

输出：NFA_p

Begin

step1　　//初始化

int $S:=0$;　　//置状态 0 为初始状态

$K\in S$;　　//将状态 0 加入状态集合 K

$f:=\varnothing$;　　//初始化状态转换集合为空集

int F;　　//定义终止状态

int $P:=0$;　　//定义当前逻辑前驱状态 P，并将 0 设置为当前逻辑前驱

int T;　　//定义但却逻辑后继状态

char a;　　//定义输入符号变量，用于存放当前状态转换弧上的输入符号

$s:=a_1a_2\cdots a_n$;　//将树中所有叶子节点按从左至右的顺序序列化

$a:=\text{Left}(N)$;　　//获取状态 N 的左叶子节点标记 a

step2　　//获取 NFA_p 的状态集 K

K:= GetStates (*RPTree*) //将树中所有的编号赋值给状态集 K

step3　　//设置终止状态

if (N is F-state)　　//若树中的编号 N 被设置为终态，详见算法 2

then $F:=N$;

else $F:=0$;　　//否则，将开始状态 0 设置为终止状态

step4　　//遍历 *RPTree* 获取所有的状态转换弧

for (i=1, i<=n, i++)

$T:=N$;　　//将 N 设置为当前逻辑后继状态

if (从 N 到其叶子节点的路径上有*节点)

then $P\xrightarrow{a}P$;　　//若有闭包运算节点，则建立环，即前驱状态与后继状态相同的弧转换

else

create $P\xrightarrow{a}T$;　　//创建状态 P 到状态 T 的弧，弧标记为 a

$a:=\text{Right}(N)$;　　//获取 N 的右叶子节点标记并更新当前弧标记

$P:=N$;　　//将 N 设置为当前逻辑前驱状态

End

例 5　采用算法 3 构造 $r=(a+b)(a*+ba*+b*)*$ 的 NFA（基于图 3），如图 5 所示。

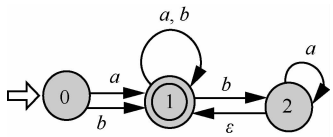


图 5　由图 3 获得的 $r=(a+b)(a*+ba*+b*)*$ 的 NFA

例 6　采用算法 3 构造正则表达式组 $r_1=abx*$

cd, $r_2 = efy*gh$ 的 NFA（基于图 4），如图 6 所示。

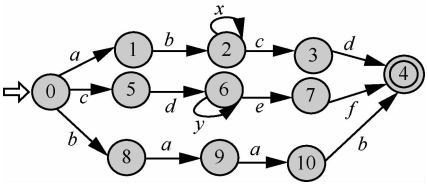


图 6 由图 4 获得的 $r_1 = abx*cd, r_2 = cdy*ef, r_3 = baab$ 的 NFA

4 复杂度分析

4.1 算法本身的复杂度分析

1) 空间复杂度

PFA 算法的主要空间开销在于存储和维护基于正则表达式的后缀形式建立的二叉树。由 PFA 算法可知，树中的节点最小时为 $n+k$，其中，n 为正则表达式组中所包含的输入符号的个数，k 为正则表达式组中运算符的个数，当需要为正则表达式组 n 增加一个空弧连接运算时，节点数最大，为 $n+k+2$。因此，算法的空间复杂度与正则表达式组的长度线性相关，为 $O(n)$。

2) 时间复杂度

首先，获取二叉树的叶子节点序列得到其状态转换弧上的输入符号符号串基于二叉树的遍历操作实现，其时间复杂度为 $O(n)$。

获取二叉树中所有编号节点的编号序列从而得到自动机的状态空间基于二叉树的遍历操作实现，其时间复杂度为 $O(n)$。获取当前状态节点的左叶子节点标记符号以及更新当前逻辑前驱状态的操作的时间复杂度为 $O(1)$。

以二叉树中每个编号节点的为子树，获取其左叶子节点、右叶子节点，以及处理路径上是否有闭包运算（*）节点的操作，需要对 $2^{\mathrm{lb}(n+1)-1}$ 个叶子节点进行创建弧的操作，其时间复杂度为 $O(n)$，由此可知，整个算法的时间复杂度为 $O(n^2)$。

4.2 PFA 方法构造的 NFA 的 size 分析

由算法可知，PFA 方法所构造的自动机其状态数最多为 $k+2$，其中，k 为正则表达式中连接运算的个数，状态转换弧的条数最多为 $n+i$，其中，n 为正则表达式中输入符号的个数，$0 \leqslant i \ll j$，符号 $\ll$ 读为“远小于”，其中，j 为正则表达式中多包含的闭包运算的个数。通常情况下，$i = 0,1$。因此，PFA 方法所获得的 NFA 的规模最好的情况下为 $size = n+k+i+2 \approx 2n+2$，即为 $O(2n)$；最坏的情况下为 $size = n+k+i+2 \approx 3n+2$，即为 $O(3n)$；通常情况下，$i = 0,1$，所以平均情况下为 $O(2n)$。

例 7 用现有的各种方法构造正则表达式 $r = (a+b)(a*+ba*+b*)*$ 如图 7~图 11 所示。

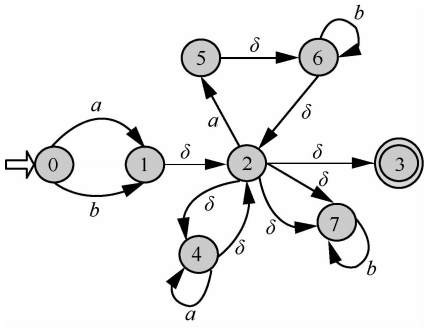


图 7 Thompson 自动机

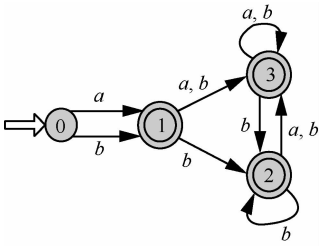


图 8 部分派生自动机

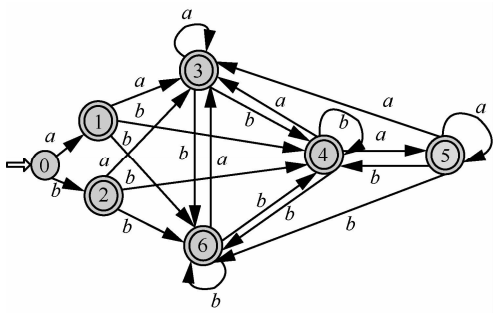


图 9 位置自动机

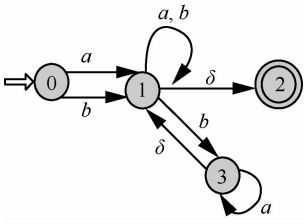


图 10 后跟自动机

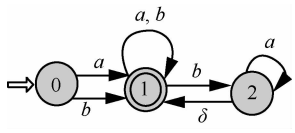


图 11 后缀自动机

表 1　　现有的 5 种基本的构造法对比情况

构造法	典型实例的 *size*	一般情况下的 *size*	算法时间复杂度
Thompson 自动机	8+14=22	$O(6\|r\|)$	$\Theta(r)$
后跟自动机	4+8=12	$\frac{3}{2}\|r\|+\frac{5}{2}$	$O(n(\log n)^2)$
位置自动机	7+22=29	$O(n^2)$	$O(n^3)$
部分派生自动机	4+11=15	$\|\|r\|\|+1$	$O(\|r^2\|\|r\|^3)$
PFA	3+7=10	理论上 $O(2\|r\|)$，实际上小于后跟自动机	$O(n^2)$

对典型实例 $r=(a+b)(a*+ba*+b*)*$ 的各种自动机构造方法结果对比

表 1 是对经典实例 $r=(a+b)(a*+ba*+b*)*$ 的各种自动机的比较。

4.3 PFA 与实际系统中 2 种常用方法在一般情况下的对比

从前文的算法分析可知，本文提出的 PFA 方法所构造得到的 NFA 的 *size* 最坏情况下为 $n+m+k$，最好的情况下为 $n+m+1$，其中，m 为正则表达式中输入符号个数，n 为正则表达式中连接运算符个数，k 为正则表达式中闭包运算符个数（显然，$m\leqslant n,k\leqslant n$）。长度为 n 的正则表达式其 Thompson 自动机中状态数最多为 $2n$ 个，状态转移数最多为 $4n$ 个，即最坏情况下 $size=6n$。平均情况下，PFA 构造法所获得的 NFA 的 $size\approx 2n$，相较 Thompson 自动机的 $size\leqslant 6n$ 而言，其规模减小了约 2/3。Glushkov 构造法（即位置自动机）所获得的 NFA 的状态数为 $m+1$（其中 m 为正则表达式中输入符号的数目）但状态转移的数量在最坏的情况下为 $O(n^2)$，其算法的时间复杂度为 $O(n^3)$，文献[33]的改进将其降低为 $O(n^2)$。可见，PFA 构造法相较现有的各种 NFA 构造方法而言，能够以 $O(n^2)$ 的时间复杂度和 $O(n)$ 的空间复杂度获得规模最小的 NFA，所获得的 NFA 与正则表达式长度呈线性关系。相较目前实际系统中常用的 2 种构造方法而言，PFA 的规模是 Thompson NFA 的 1/3；算法时间复杂度较 Gloshkov 构造法的规模和时间复杂度（规模为 $O(n^2)$，时间复杂度为 $O(n^3)$）均下降了一个数量级。

5 结束语

在基于正则表达式模式匹配的系统中，特别是在具有大规模模式集的系统中，正则引擎本身的规模对系统的效率起着决定性的作用。有效地减少系统中并行执行的 NFA 的数量以及 NFA 的规模，将会大幅度地提升系统的效率。本文提出了一种新颖的构造 NFA 引擎的方法 PFA 构造法，不但能够获得规模更小、所包含的空弧转换较 Thompson 算法要少得多的 NFA，而且其算法复杂度较 Gloshkov 构造法整整下降了一个数量级，并且，PFA 构造法还能以非常简单的方式为包含有多条正则表达式的表达式组构造单一的 NFA，大幅度地减少系统中并行执行的 NFA 的数量，从而有效地解决实际系统中因为大量 NFA 并行执行导致的性能指数级下降的问题。

参考文献：

[1] JIANG T, RAVIKUMAR B. Minimal NFA problems are hard[J]. SIAM Journal on Computing, 1993, 22(6): 1117-1141.

[2] ROESCH M. Snort: lightweight intrusion detection for networks[A]. Proceeding of LISA99, 13th Sytems Administration Conference[C]. Seattle, USA, 1999.229-238.

[3] PAXSON V. Bro: a system for detecting network intruders in real-time[J]. Computer networks, 1999, 31(23): 2435-2463.

[4] LEVANDOSKI J, SOMMER E, STRAIT M. Application layer packet classifier for Linux[EB/OL]. http://IT-filter.sourceforge.net. 2008.

[5] WANG H, PU S, KNEZEK G, *et al*. A modular NFA architecture for regular expression matching[A]. Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate arrays[C]. ACM, 2010.209-218.

[6] YAMAGAKI N, SIDHU R, KAMIYA S. High-speed regular expression matching engine using multi-character NFA[A]. Field Programmable Logic and Applications, FPL 2008[C]. 2008. 131-136.

[7] NAKAHARA H, SASAO T, MATSUURA M. A regular expression matching using non-deterministic finite automaton[A]. Formal Methods and Models for Codesign (MEMOCODE), 2010 8th IEEE/ACM International Conference on IEEE[C]. 2010.73-76.

[8] TAN F. Algorithm for constructing the simplified DFA of regular expressions[J]. Journal of Beijing University of Aeronautics and Astronautics, 1998, 24(4): 495-498.

[9] NAVARRO G, RAFFINOT M. Compact DFA representation for fast regular expression search[M]. Algorithm Engineering. Springer Berlin Heidelberg, 2001.

[10] 张伟, 薛一波, 嵩天等. 一种支持多正则表达式匹配的硬件结构[J].

清华大学学报, 2009, 49(10): 132-135.
ZHANG W, XUE Y B, XIAO T, *et al*. Multiple regular expression matching hardware architecture[J]. Journal of Tsinghua University (Science and Technology), 2009, 49(10):132-135.

[11] 张树壮, 罗浩, 方滨兴等. 面向网络安全的正则表达式匹配技术[J]. 软件学报, 2011, 22(8): 1838-1853.
ZHANG S Z, LUO H, FANG B X, *et al*. Regular expressions matching for network security[J]. Journal of Software,2011,22(8): 1838-1854.

[12] THOMPSON K. Programming techniques: regular expression search algorithm[J]. Communications of the ACM, 1968, 11(6): 419-422.

[13] GLUSHKOV V M. The abstract theory of automata[J]. Russian Mathematical Surveys, 1961, 16(5): 1-53.

[14] GARCÍA P, LÓPEZ D, RUIZ J, *et al*. From regular expressions to smaller NFAs[J]. Theoretical Computer Science, 2011, 412(41): 5802- 5807.

[15] SIPPU S, SOISALON S E. Parsing Theory: I: Languages and Parsing[M]. New York: Springer, 1988.

[16] GLUSHKOV V M. The abstract theory of automata[J]. Russian Mathematical Surveys, 1961, 16(5): 1-53.

[17] MCNAUGHTON R, YAMADA H. Regular expressions and state graphs for automata[J]. Electronic Computers, 1960, (1): 39-47.

[18] ANTIMIROV V. Partial derivatives of regular expressions and finite automaton constructions[J]. Theoretical Computer Science, 1996, 155(2): 291-319.

[19] ILIE L, YU S. Follow automata[J]. Information and Computation, 2003, 186(1): 140-162.

[20] KUMAR S, CHANDRASEKARAN B, TURNER J, *et al*. Curing regular expressions matching algorithms from insomnia, amnesia, and acalculia[A]. Proceedings of the 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems[C]. ACM, 2007.155-164.

[21] BECCHI M, CROWLEY P. A hybrid finite automaton for practical deep packet inspection[A]. Proceedings of the 2007 ACM CoNEXT Conference[C]. ACM, 2007: 1.

[22] SMITH R, ESTAN C, JHA S. XFA: faster signature matching with extended automata[A]. Security and Privacy, SP 2008[C]. 2008. 187-201.

[23] KUMAR S, DHARMAPURIKAR S, YU F, *et al*. Algorithms to accelerate multiple regular expressions matching for deep packet inspection[A]. ACM SIGCOMM Computer Communication Review[C]. ACM, 2006, 36(4): 339-350.

[24] BECCHI M, CADAMBI S. Memory-efficient regular expression search using state merging[A]. INFOCOM 2007, 26th IEEE International Conference on Computer Communications[C]. 2007. 1064-1072.

[25] ZHANG J, ZHANG D, HUANG K. A regular expression matching algorithm using transition merging[A]. 15th IEEE Pacific Rim International Symposium on IEEE[C]. 2009.242-246.

[26] FICARA D, GIORDANO S, PROCISSI G, *et al*. An improved DFA for fast regular expression matching[J]. ACM SIGCOMM Computer Communication Review, 2008, 38(5): 29-40.

[27] YU F, CHEN Z, DIAO Y, *et al*. Fast and memory-efficient regular expression matching for deep packet inspection[A]. Architecture for Networking and Communications systems, ANCS 2006[C]. 2006. 93-102.

[28] FICARA D, GIORDANO S, PROCISSI G, *et al*. An improved DFA for fast regular expression matching[J]. ACM SIGCOMM Computer Communication Review, 2008, 38(5): 29-40.

[29] SMITH R, ESTAN C, JHA S. XFA: faster signature matching with extended automata[A]. Security and privacy, SP 2008[C]. 2000. 187-201.

[30] 杨毅夫, 刘燕兵, 刘萍等. 正则表达式的 DFA 压缩算法 [J]. 通信学报, 2009, 30(10): 36-41.
YANG Y F, LIU Y B, LIU P, *et al*. Effective algorithm of compressing regular expression's DFA[J]. Journal on Communications, 2009, 30(10): 36-41.

[31] 柳厅文, 孙永, 卜东波等. 正则表达式分组的 1/(1-1/*k*)-近似算法[J]. 软件学报, 2012, 23(9): 2261-2272.
LIU T W, SUN Y, BU D B, *et al*. 1/(1-1/*k*)-optimal algorithm for regular expression grouping[J]. Journal of Software,2012, 23(9): 2261-2272.

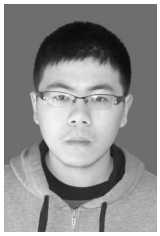
[32] KAMALA K. Introduction To Formal Languages, Automata Theory And Computation[M]. Pearson Education India, 2009.

[33] BRÜGGEMANN K A. Regular expressions into finite automata[J]. Theoretical Computer Science, 1993, 120(2): 197-213.

作者简介：

敬茂华（1977-），女，四川蓬溪人，博士，东北大学秦皇岛分校讲师，主要研究方向为网络与信息安全、物联网、云计算安全、自动机理论。

杨义先（1961-），男，四川盐亭人，博士，北京邮电大学教授、博士生导师，主要研究方向为现代密码学、网络与信息安全、灾备设备等。



汪韬（1992-），男，安徽芜湖人，东北大学秦皇岛分校本科生，主要研究方向为云计算安全、自动机理论、大数据。

辛阳（1977-），男，山东烟台人，北京邮电大学副教授，主要研究方向为移动通信安全、下一代网络安全。