一种获得有限自动机状态间关系的高效算法[1]

乔登科[1,3,4]，柳厅文[2,3,4]，孙 永[1,4]，郭 莉[1,4]

[1]（中国科学院 信息工程研究所, 北京 100093）
[2]（中国科学院 计算技术研究所, 北京 100190）
[3]（中国科学院 研究生院, 北京 100049）
[4]（信息内容安全技术国家工程实验室, 北京 100190）
（qiaodengke@gmail.com）

An Efficient Algorithm to Obtain Relations between States of Finite Automata

Qiao Dengke[1,3,4+], Liu Tingwen[2,3,4], Sun Yong[1,4], Guo Li[1,4]

[1] (Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093)
[2] (Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)
[3] (Graduate University of Chinese Academy of Sciences, Beijing 100049)
[4] (National Engineering Laboratory for Information Security Technologies, Beijing 100190)

Abstract Regular expression matching plays an important role in many network security applications. DFA is preferred to perform regular expression matching in backbone networks because of its high and robust matching efficiency. However, it may experience the problem of state explosion. Some recent work try to address the problem based on the relation between states. However, there is still lack of a time-efficient and memory-efficient method to obtain exact relations between states. We give a new algorithm to achieve the goal by analyzing all active state sets of finite automata, and introduce a method to obtain the active state sets. Experimental results show that our work can obtain exact state relations with only 1/256 memory consumption and 15% time cost comparing with prior methods.

Key words Regular Expression; State Explosion; Relationship between States; Active State Set

摘 要 正则表达式匹配在网络安全应用中发挥着重要的作用。确定性有限自动机(Deterministic Finite Automaton，DFA)具有高速稳健的性能，因而更适合于在骨干网络环境下执行正则表达式匹配。然而，DFA 存在状态膨胀的问题。很多研究工作基于状态关系来解决 DFA 的状态膨胀问题。然而目前对如何获得状态间的关系仍然缺少一种时空高效的解决办法。我们提出了一个通过有限自动机(Finite Automaton, FA)的活跃状态集来准确计算状态关系的算法，并给出了一个高效的获取所有活跃状态集的方法。实验结果证明，该方法不仅能准确地得到状态关系，而且其空间占用和时间消耗仅是已有方法的 1/256 和 15%左右。

关键词 正则表达式；状态膨胀；状态关系；活跃状态集

基金支持：国家高技术研究发展计划 863（2011AA010703，2011AA010705）；国家自然科学基金（61070026，61003295）
通信作者：乔登科，电子邮箱：qiaodengke@gmail.com，联系电话：13522539871

中图法分类号 TP391

现在很多的网络应用和安全应用都依赖于深度包检测(Deep Packet Inspection, DPI)技术进行基于感知流量内容的处理。例如，网络入侵检测系统将网络数据包的内容负载与一些预先设定的模式集(入侵特征、攻击特征等)进行匹配，以检测该数据包是否携带恶意数据。由于具有简洁、表达式能力强和扩展性好的优点，正则表达式取代了字符串成为描述规则的首选工具，并且广泛地应用在一些开源工具(如L7-Filter[1]、Snort[2]等)和很多公司(如思科、华为等)的商用产品中。但是网络带宽和模式集规模的快速增长使得 DPI 应用中的正则表达式匹配面临严重挑战。作为其核心操作，正则表达式需要在有限的内存资源下进行高速地匹配。

非确定型有限自动机(Non-deterministic Finite Automaton, NFA)和确定型有限自动机 (Deterministic Finite Automaton, DFA) 是两种经典的匹配自动机，并且与正则表达式具有等价的表达能力。相应地，匹配方法可以分为两种：基于 NFA 的匹配方法和基于 DFA 的匹配方法。基于 NFA 的匹配方法的优点是占用内存少，但是它很难满足 DPI 应用所要求的高速匹配。这是因为 NFA 每处理一个字符需要进行多次状态转移，使得其状态转移时间复杂度最坏情况下为 $O(n^2)$ (n 是 NFA 的状态数目)。与 NFA 相反，DFA 在匹配过程的任意时刻都只有一个活跃状态，从而处理每个字符只需访问一次状态表；并且多条正则表达式可以生成一个合并 DFA，从而仅扫描一遍数据包即可完成对所有正则表达式的匹配。因而在高速网络环境下大部分的 DPI 应用选择了基于 DFA 的匹配方法。然而，NFA 确定化生成 DFA 的过程容易发生 DFA 状态膨胀，导致生成的 DFA 需要消耗巨大的存储空间甚至超过现有 DPI 设备的内存上界。因此，如何减少 DFA 的巨大存储空间需求是实现高速正则表达式匹配的关键问题。

NFA 的状态之间的活跃关系不同，目前已有很多研究工作基于状态间的不同关系来实现时空高效的正则表达式匹配[3][4][5][6]。然后它们的计算状态关系方法有些只能针对特定关系计算[3][4][5]，有些虽能计算出所有的关系但其效率低下使得无法在大规模正则表达式上应用[6]。本文提出了一个新的计算方法 ROBAS (Relation Obtaining By Analyzing active state Sets)，该算法通过分析 FA 的活跃状态集来计算 FA 状态间的关系，并且其时间复杂度与活跃状态集的数目是线性相关的。并且本文还引入了一种获得所有活跃状态集的高效算法 ASS-SUBSET(Active State Set-SubSet)，该算法通过优化将 NFA 确定化为 DFA 的子集构造法[7]来得到。实验结果证明我们的方法不仅能准确的得到所有的状态关系，而且其空间占用和时间消耗仅为已有方法的 1/256 和 15%左右。

1 相关工作

DFA 的存储开销主要来自于其状态转移表，表的行宽和带宽分别对应于 DFA 的状态数目和字符集的大小（ASCII 编码该值为 256，每个字符对应一条转移边）。因此，目前减少 DFA 存储空间消耗的研究热点主要分为压缩 DFA 状态转移边和减少 DFA 状态数目两个方面。

由于 DFA 的状态转移边中存在大量的冗余边，DFA 压缩方法可以通过消除冗余边来降低 DFA 状态转移表的开销。例如，Kumar 等人[8]提出一种称为延迟输入 DFA(Delay Input DFA, D^2FA)的自动机来减少内存开销，它通过引入默认转移边来消除相邻 DFA 状态间的相同转移边；Ficara 等人[9]通过消除子状态和父状态的相同转移边来减少需要保存的转移边数目。但是两种方法压缩生成的自动机处理

每个输入字符都需要多次访问内存。而且两种方法是对已生成的 DFA 进行压缩，因此需要事先生成正则表达式集合的 DFA，而这一点在实际情况中是很难满足的。

减小 DFA 的状态数目的方法主要是基于 DFA 中存在很多等价状态的现象，通过消除 DFA 中的等价情况来降低 DFA 状态转移表的存储开销。Hopcroft[9]提出了一个时间复杂度为 $n\log n$ 的算法对 DFA 的等价状态进行压缩。但是这个算法同样要求先生成正则表达式所对应的 DFA，而这一点在实际情况中是很难满足的。而且该算法的压缩效果很有限。

由于正则表达式所对应的最小 DFA 状态数目是固定的，很难进行再压缩，因此越来越多的研究开始关注于 NFA 和 DFA 的融合，目的是大幅降低自动机的存储开销的同时尽量保持 DFA 的匹配速度。例如，Becchi 等人[11]提出的 Hybrid-FA，依据经验启发式地找到导致 DFA 状态膨胀的位置，然后在这些位置不进行确定化来避免状态膨胀。由于不能准确找到导致膨胀的位置，Hybrid-FA 适用的范围很有限，其构造过程花费时间过长并且在某些正则表达式上效果很差。

基于 NFA 的状态相互关系实现时空高效的正则表达式匹配成为目前的研究热点。Ilie 等人[13]首次提出了通过合并 NFA 的等价状态来达到消除 NFA 中等价关系的想法。在这篇文章里，作者提出了用左等价关系和右等价关系来找到 NFA 的等价状态并合并，并给出了等价关系的计算方法。但是作者给出的方法只能计算一部分的等价关系，无法求出其他 NFA 状态关系。Yang 等人[6]引申了状态间的关系，得到互斥，包含，等价和独立四种关系，并提出 SFA (Semi-deterministic Finite Automaton)来避免状态膨胀。但作者给出的计算 NFA 状态间关系的算法不仅很难保证正确性，而且过程过于复杂，软件实现效率很低;同时 SFA 的结构太过于复杂，实际情况下很难应用。

本文在总结前人工作的基础上，提出了两个新的算法 ROBAS 算法和 ASS-SUBSET 算法，可以高效准确地求出 NFA 状态的所有相互关系。

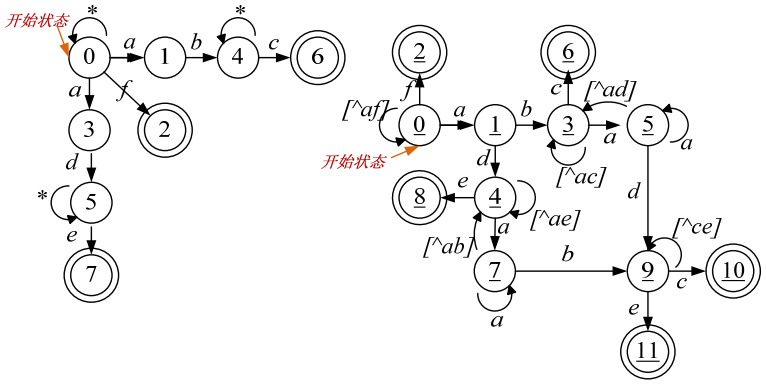
2 基本概念

有限自动机通常用五元组 $M=(Q,\Sigma,\delta,q_0,F)$ 来表示，其中 Q 是状态集合，Σ 是字符集，δ 是状态转移函数，q_0 是自动机的开始状态($q_0\in Q$)，F 是自动机的接受状态的集合($F\subseteq Q$)。NFA 和 DFA 是两种经典的有限自动机，两者的区别在于 NFA 的状态转移函数是非确定性的---$\delta:Q\times\Sigma\to \mathrm{P}(Q)$，其中 $\mathrm{P}(Q)$ 是 Q 的幂集，而 DFA 的状态转移函数是确定性的---$\delta:Q\times\Sigma\to Q$。有限自动机通过顺序处理输入串的字符来进行匹配，整个匹配过程从开始状态开始，处理一个字符时自动机沿着当前状态的所有标记为该字符的所有转移边进行迁移。当某个接受状态被访问时，认为匹配成功。由于 NFA 的状态对一个字符可能具有多条转移边，因此在 NFA 的匹配过程中可能有多个状态同时活跃。本文中称呼这些同时活跃的状态集合为活跃状态集，简称活跃集。

运用子集构造法可以将一个 NFA $M=(Q,\Sigma,\delta,q_0,F)$ 确定化为 DFA $M'=(Q',\Sigma,\delta',q_0',F')$，其中 $Q'\subseteq \mathrm{P}(Q)$，$q_0'=\{q_0\}\in Q'$。子集构造法的主要思想是从开始状态 q_0 完全遍历 NFA 得到所有可能的活跃状态集，并对每个活跃状态集在 DFA 中用一个状态表示。与 NFA 的状态相

比，DFA 的每个状态对每个字符有唯一的转移边，因而在 DFA 匹配的任意时刻仅有一个状态活跃。

以正则表达式“*ab.*c*”、“*ab.*e*”和“*f*”为例，其相应的 NFA 和经过子集构造法确定化的 DFA 如图 1 所指，在该图中我们还给出了 NFA 的活跃状态集和 DFA 状态间的映射关系。



识别 *ab.*c*、*ad.*e* 和 *f* 的NFA　　识别 *ab.*c*、*ad.*e* 和 *f* 的DFA

DFA状态	NFA 活跃状态集
0	{0}
1	{0, 1, 3}
2	{0, 2}
3	{0, 4}
4	{0, 5}
5	{0, 1, 3, 4}
6	{0, 4, 6}
7	{0, 1, 3, 5}
8	{0, 5, 7}
9	{0, 4, 5}
10	{0, 4, 5, 6}
11	{0, 4, 5, 7}

Fig.1 Mapping between DFA states and active state sets of NFA

图 1 NFA 活跃状态集和 DFA 状态间的映射

给定有限自动机 $M=(Q,\Sigma,\delta,q_0,F)$，对任意两状态 $x,y\in Q$，我们认为状态 y 能被字符串 $w\in\Sigma^+$ 从状态 x 激活如果存在着一条从 x 到 y 并且标记为 w 的路径，简记为 $x\xrightarrow{w}y$。那么，任意一个状态 x 可以定义一个字符串集合 $L(x)=\{w\in\Sigma^+|\ q_0\xrightarrow{w}x\}$。

Yang 等人依据字符串集合间的关系定义了三种有限自动机状态间的关系，准确地说状态之间有五种，本文将之扩展并具体描述为：1）互斥关系---x 与 y 互斥，当且仅当 $L(x)\cap L(y)=NULL$，即不存在字符串使得 x 与 y 同时活跃，如图 1 中的状态 6 和状态 7；2）等价关系---x 与 y 相互等价，当且仅当 $L(x)=L(y)$，即任意字符串使 x 活跃的同时必然使 y 活跃，使 y 活跃的同时必然使 x 活跃，如图 1 中的状态 1 和状态 3；3）包含关系---x 包含 y，当且仅当 $L(x)\cap L(y)\neq NULL, L(x)\subset L(y), L(y)\not\subset L(x)$，即任意字符串使 x 活跃的同时必然使 y 活跃，但使 y 活跃的同时未必使 x 活跃，如图 1 中的状态 4 和状态 6；4）包含于关系---x 包含于与 y，当且仅当 $L(x)\cap L(y)\neq NULL, L(y)\subset L(x), L(x)\not\subset L(y)$，即任意字符串使 y 活跃的同时必然使 x 活跃，但使 x 活跃的同时未必使 y 活跃，如图 1 中的状态 1 和状态 0；5）独立关系---x 与 y 相互独立，当且仅当 $L(x)\cap L(y)\neq NULL, L(x)\not\subset L(y), L(y)\not\subset L(x)$，即

任意字符串使 y 活跃的同时未必使 x 活跃，使 x 活跃的同时未必使 y 活跃，但存在字符串使两者同时活跃，如图 1 中的状态 4 和状态 5。

由上述定义可知，DFA 的任意两状态之间都是互斥的，因为在任何时刻都不可能有两个 DFA 状态同时活跃。而对 NFA 来说，上述的五种状态关系在其状态间都可能存在。Yang 等人[6]还证明了在 NFA 到 DFA 的确定化过程中，如果 NFA 的任意两状态间不满足独立关系，那么确定化得到的 DFA 状态规模不会超过 NFA 的状态规模。

3 ROBAS 算法

Y.H.E. Yang 等人依据状态关系的定义，根据状态定义的字符串集合之间的关系来计算 NFA 状态间的关系。由于相应字符串集合中的字符串规模可能很大，甚至可能为无限集，因此该方法的效率不高。通过上述分析可知，NFA 的两个状态 x 和 y 间的关系（表示为 $x \otimes y$ ）可以通过如下三个条件进行判定：1） $x \wedge y$ ，即 NFA 的某个活跃状态集 S_i 同时包含 x 和 y ；2） $x \wedge \bar{y}$ ，即 NFA 的某个活跃状态集 S_i 包含 x 但不包含 y ；3） $\bar{x} \wedge y$ ，即 NFA 的某个活跃状态集 S_i 包含 y 但不包含 x 。

获得 $x \wedge y$ 、 $\bar{x} \wedge y$ 和 $x \wedge \bar{y}$ 三个条件的布尔值后，我们可以根据表 1 来得到 $x \otimes y$ 的准确结果。

Table 1 Relationship between NFA state A and NFA state B

表 1　NFA 状态 A 和 B 的相互关系

$x \wedge y$	$\bar{x} \wedge y$	$x \wedge \bar{y}$	$x \otimes y$
0	0	0	互斥
0	0	1	互斥
0	1	0	互斥
0	1	1	互斥
1	0	0	等价
1	0	1	包含
1	1	0	包含于
1	1	1	独立

那么计算 $x \otimes y$ 的结果就转化为了如何计算 $x \wedge y$ 、 $\bar{x} \wedge y$ 和 $x \wedge \bar{y}$ 三个条件的布尔值。由于 NFA 的活跃状态集包含了不同状态间的可能活跃情况，因此我们可以通过分析 NFA 的活跃状态集来得到 NFA 状态间的关系，即本文所提出的 ROBAS 算法。

给定 NFA，假设我们已经得到其所有可能的活跃状态集（获取方法在下一章讨论），构成集合 S ，对 S 中的每一个活跃状态集 S_i ，如果 NFA 的两个状态 $x,y \in Q$ 满足 $x \in S_i$ 且 $y \in S_i$ ，那么置 $x \wedge y = 1$ ；若满足 $x \notin S_i$ 且 $y \in S_i$ ，那么置 $\bar{x} \wedge y = 1$ ；若满足 $x \in S_i$ 且 $y \notin S_i$ ，那么置 $x \wedge \bar{y} = 1$ 。当对集合 S 的所有元素处理一遍后，我们就能得到该 NFA 的任意两状态之间的关系。

算法 1. ROBAS 算法---从自动机的活跃状态集求得状态关系

算法输入：NFA 的所有活跃状态集的集合 S 和 NFA 的状态规模 n

算法输出：NFA 状态间的关系矩阵 *relation*[*n*][*n*]

1. 申请整型二维矩阵 *contact*[*n*][*n*]，置初始值为-1；申请布尔型二维矩阵 *intersection*[*n*][*n*]，置初始值为 0
2. while(S 不空){
3. 　　从 S 中取出的一个元素 S_i ；
4. 　　对于 S_i 中的任意两状态 x 和 y ，置

intersection[*x*][*y*]=1 ； 如 果 *contact*[*x*][*y*]=-1，置 *contact*[*x*][*y*]=1；

5. 对于 S_i 中的任意状态 *x* 和任意不在 S_i 中的状态 *y*，置 *contact*[*x*][*y*]=0；

6. 从 S 中删除元素 S_i；

7. } // end of while

8. 计算 *relation*[*n*][*n*]的值，并返回。

最后 *contact*[*x*][*y*]=1 其实就等价于 $x \wedge \bar{y} = 1$，*intersection*[*x*][*y*]=1 其实就等价于 $x \wedge y = 1$，因此我们得到了最后一步中 *relation*[*x*][*y*] 的计算规则：若 *intersection*[*x*][*y*] = 0 则 *relation*[*x*][*y*] = 011；若 *intersection*[*x*][*y*] = 1 且 *contact*[*x*][*y*] = 1 且 *contact*[*y*][*x*] = 1，则 *relation*[*x*][*y*] = 100; 若 *intersection*[*x*][*y*] = 1 且 *contact*[*x*][*y*] = 1 且 *contact*[*y*][*x*] = 0，则 *relation*[*x*][*y*] = 101; 若 *intersection*[*x*][*y*] = 1 且 *contact*[*x*][*y*] = 0 且 *contact*[*y*][*x*] = 1，*relation*[*x*][*y*] = 110; 若 *intersection*[*x*][*y*] = 1 且 *contact*[*x*][*y*] = 0 且 *contact*[*y*][*x*] = 0，则 *relation*[*x*][*y*] = 111。

假设某 NFA 的所有活跃状态集的集合 *S* ={{0,1},{0,3},{1,2}}，且该 NFA 共有 4 个状态，ROBAS 算法计算状态关系的运算过程如图 2 所示。

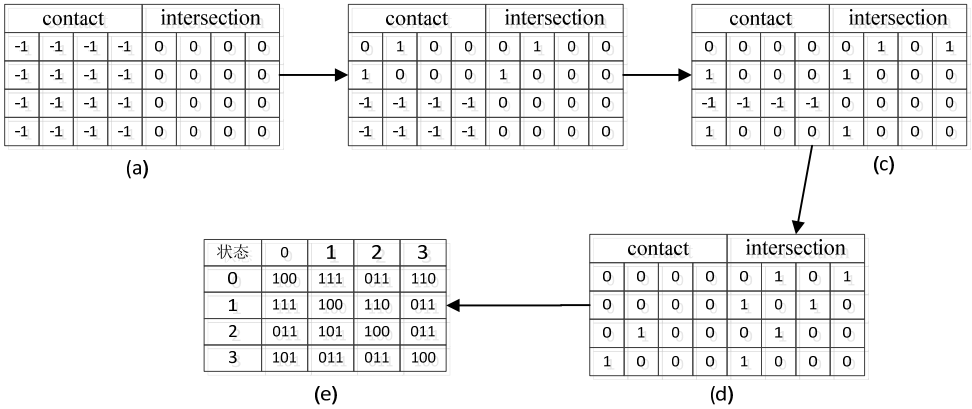


Fig.2 The computing process for ROBAS algorithm

图 2 ROBAS 算法的具体运算过程

初始化，*contact* 矩阵中的数值全部为 -1，*intersection* 矩阵中的数值全部为 0，如图 2 (a)所示。依次处理 *S* 中的{0,1}、{0,3}和{1,2}三个活跃状态集，*contact* 矩阵和 *intersection* 矩阵的依次变化为图 2 (b)、图 2 (c)和图 2 (d)所示。最后我们可以计算出 *relation* 矩阵的值，如图 2 (e)所示。根据 *relation* 矩阵，我们得出该 NFA 的状态 0 和状态 1 相互独立、状态 0 和状态 2 互斥、状态 0 包含状态 3 等关系。

4 NFA 活跃状态集的获取

给定 NFA，如何获取其所有可能的活跃状态集仍是一个未解决的问题。由前文可知，在 NFA 到 DFA 的确定化过程中，子集构造法对 NFA 的每个活跃状态集用一个 DFA 状态表示，很显然一个简单的获取方法在将该 NFA 确定化，在生成 DFA 的同时也得到了全部的活跃状态集。然而该获取方法是相当低效的，具体表现在：1）该方法需要保存生成的 DFA，DFA 的每个状态需要保留 256 条转移边，导致其消耗的存储空间可能非常巨大；2）对每个 DFA 状态，其 256 条出边都需要进行计算，导致该方法得到所有活跃状态集的时间开销比较大。该获取方法的时间复杂度和空间

复杂度均为$O(n\Sigma)$，其中n为 DFA 的状态数目。在本文中，我们将该方法称为 NFA2DFA 方法，即 NFA 确定化生成 DFA 的方法。

本文提出了一种新的活跃状态集的获取方法---ASS-SUBSET 算法。ASS-SUBSET 算法引入了如下两种技术来优化子集构造法获取活跃状态集的过程。

首先，我们的目标获得 NFA 的所有活跃状态集，不是获得 DFA。因而在子集构造法的运行过程中，对每个 DFA 状态计算其转移边时，若得到一个新的 NFA 活跃状态集，我们只需保存活跃状态集而不需保存转移边。可以证明，不保存转移边不影响算法获取所有的活跃状态集。经过此步优化，ASS-SUBSET 获取方法的空间复杂度为$O(n)$。实际上 ASS-SUBSET 所需的内存空间仅为子集构造法的$1/256$。

其次，每个 DFA 状态的任意一条边都可能转移到新的活跃状态集，因而为了获得所有的活跃状态集，每个 DFA 状态的其经过所有 256 个字符可能到达的活跃状态集都要进行求解。然而，这 256 个活跃状态集并不是完全不相同的。文献中的实验结果显示对大部分 DFA 来说，其每个状态的 256 个转移边所到达的活跃状态集大部分都是相同，不同的活跃状态集数目为数十个[13]。因而，如果一个 DFA 状态在某些字符集上到达同一个活跃状态集，那么实际上我们只需计算其中一个即可。例如图 1 中的 DFA 状态 0，其所有的转移所能到达的活跃状态集有{0, 1, 3}和{0, 2}两种可能性。因此我们只需要对每个 DFA 状态计算其经过转移边所能到达的所有不同的活跃状态集，而不需要求出所有 256 个活跃状态集。

对每个 NFA 状态，我们将指向同一个目的状态的转移字符放在一起，由此形成了此目的状态的字符转移区间。需要注意的，该字符转移区间可能是不连续的，此时我们需将该区间分为若干个连续的字符区间。以图 1 中的 NFA 状态 0 为例，其转移到状态 0 的字符区间为[0, 255]，转移到状态 1 和状态 3 的字符区间为[97, 97]，转移到状态 2 的字符区间为[102, 102]。我们取所有这些字符区间的首尾字符和首尾字符的邻居字符作为该状态的完备转移字符。图 1 中的状态 0 的完备转移字符集是{0, 1, 96, 97, 98, 101, 102, 103, 255}。同理状态 1 的完备转移字符集为{97, 98, 99}，状态 3 的完备转移字符集为{99, 100, 101}。

一个状态集的完备转移字符就是该状态集里面各个状态的完备转移字符的并集。比如图 1(a)，状态集{0,1}的状态转移字符集就是{0, 1, 96, 97, 98, 99, 101, 102, 103, 255}，这样状态集{0,1}只需要扫描 10 条转移边就能获得{0,1}经过状态转移所产生的所有新的活跃状态集。相比之前需要扫描 256 个字符，ASS-SUBSET 算法能极大地提升运算效率。

我们通过数学归纳法来证明 ASS-SUBSET 算法的正确性。假设 NFA 状态集N'的字符转移区间集合为$S_{N'}=\{S_{N'1},S_{N'2},S_{N'3}...\}$，其中$S_{N'1}$、$S_{N'2}$、$S_{N'3}$等就是状态集$N$的字符转移区间。

1. 在最简单的情况下，即字符转移区间集合$S_{N'}$里面只有一个区间，令该区间为$[m,n]$。假设状态集N'通过该区间可以到达状态 B，那么选择的字符集$\{m-1,m,m+1,n-1,n,n+1\}$很明显可以使得状态集N'到达新的状态集{B}；
2. 假如字符转移区间集合$S_{N'}$包含多个区间并且各个区间互不相交。我们选择的字符集可以使得每个区间内的元素在字符集中至少出现一次，所以可以使得状态集N'经过字符集转移到达所有的新的状态集。

3. 假如字符转移区间集合 $S_{N'}$ 包含多个区间并且各个区间有相交关系。我们令 $S_{N'}$ 的元素个数为 k。1）当 k=2 时，我们假设 $S_{N'}$ 的元素分别为 $[m,n]$ 和 $[p,q]$。假定状态集 N' 经过 $[m,n]$ 可以到达状态 C，经过 $[p,q]$ 可以达到状态 D。如果 $[m,n]$ 和 $[p,q]$ 相交，假设 $m<p<n<q$，这个时候状态集 N' 通过状态转移可以到达三个状态集{C}、{D}和{C，D}，这三个状态集对应的状态集 N' 的字符转移区间集合 $S=\{[m,p],[p,n][n,q]\}$。我们可以很容易地发现 S 和 $S_{N'}$ 的转移字符集是一样的，均为 $\{m-1,m,m+1,n-1,n,n+1\}$ 和 $\{p-1,p,p+1,q-1,q,q+1\}$，所以在这种情况下 S 和 $S_{N'}$ 是等价的；如果 $[m,n]$ 和 $[p,q]$ 是相互包含和被包含关系，情况和相交关系类似。综上所述，在 $S_{N'}$ 包含两个区间的情况下，不管区间关系如何，这两个区间都可以被分割成各个不相同的区间，每一个区间只对应一个新的状态集，任意两个区间最多只有一个相交点，分割成的区间的状态转移字符和我们选择的状态转移字符一样。2）当 $k>2$ 时，我们假设前 $(k-1)$ 个区间可以被分割成不相同的区间，每个区间只对应一个新的状态集；那么在这个时候向区间集合里放入第 k 个字符区间，这个区间和现有的各个区间又可以进行新的分割，所形成的区间集合里每个区间依然只对应一个新的状态集。3）由 1）和 2）可知，不管 $S_{N'}$ 里面包含几个区间，$S_{N'}$ 都可以被分割成各个不相同的区间，每个区间至少有一个元素出现在 $S_{N'}$ 的状态转移字符集中。由上述第 2 条可知，当 $S_{N'}$ 中的区间有相交关系时，算法所选的字符集同样可以使得状态集 N' 经过转移到达所有的新状态集合。

综上所述，ASS-SUBSET 算法可以求得 NFA M 的所有活跃状态集 N。

5 实验

本文实验的实验环境为 Intel Core2 Duo 2.1GHz，512MB 内存。为了使实验结果更具有可比性，我们从 Snort 和 Bro 等 2 个开源系统中提取 4 个有代表性的真实正则表达式集合，即 snort24.re、snort31.re、snort34.re 和 bro217.re 作为实验模式集。

我们分别用 NFA2DFA 算法和 ASS-SUBSET 算法对 snort24.re、snort31.re、snort34.re 和 bro217.re 四个模式集分别计算他们各自的活跃状态集和 NFA 状态关系。这两种算法都可以准确算出 NFA 状态关系。两种方法所消耗的时间如表 2 所示：

Table 2 Statistics of time spent by NFA2DFA and ASS-SUBSET

表 2 NFA2DFA 和 ASS-SUBSET 所花费的时间统计信息

Pattern Set	NFA2DFA(s)	ASS-SUBSET(s)
Snort24.re	430.60	55.02
Snort31.re	514.15	51.15
Snort34.re	579.37	62.19
Bro217.re	601.94	131.57

我们可以看出，相比 NFA2DFA 算法，ASS-SUBSET 算法的效率提升非常可观。其中对于模式集 snort24.re、snort31.re、snort34.re，ASS-SUBSET 算法相比于

NFA2DFA 算法分别节省了 87.3%、90%、89.6%的时间。即使对于模式集相对来说较为简单的 bro217.re，ASS-SUBSET 算法依然节省了 78.1%的时间。

6 结束语

网络带宽的爆炸增长和网络应用的不断丰富给实时深度包检测带来巨大压力。传统的基于 NFA 的正则表达式匹配技术已经不能够满足网络安全应用和服务的实时检测要求。因此相关的研究热点已经转向匹配速度更快的基于 DFA 的正则表达式匹配技术。然而 DFA 存在着状态数目膨胀的问题，在某些情况下状态数目甚至呈指数增长。因此越来越多的人开始研究基于 NFA 状态关系的复合 FA 的构造方法，复合 FA 既有 NFA 存储开销小的优点又有 DFA 高速匹配的优点。然而当前的计算 NFA 状态之间关系的算法不仅效率低下，而且无法准确求出所有的关系。本文给出了一个通过 FA 活跃状态集来计算 FA 状态间关系的算法---ROBAS 算法和一个高效获取 FA 活跃状态集的算法---ASS-SUBSET 算法。实验验证，我们的算法不仅可以准确求出 FA 状态之间的所有相互关系，还比当前已有的算法节省了大概 85%左右的时间。

算法的不足：我们提出的高效获取 FA 活跃状态集的算法--- ASS-SUBSET 算法，相比已有的算法虽然大幅提升了获取速度，时间复杂度和空间复杂度都得到了很大的优化。但是在实际中，由于 FA 的活跃状态集数目往往很多甚至达到千万级别，因此生成所有的活跃状态集并不是非常可行。因此作者接下来的工作就是研究现有算法的改进策略，在不能够获得所有的活跃状态集的前提下依然可以准确求出 FA 状态的相互关系。

参 考 文 献

[1] Levandoski J, Sommer E, Strait M. Application Layer Packet Classier for Linux . http://l7-filter.sourceforge.net/.

[2] Martin Roesch. Snort - Lightweight Intrusion Detection for Networks // Proc of USENIX LISA, Monterey, California: USENIX, 1999: 229-238.

[3] Ilie L, Navarro G. Yu S. On NFA reductions. //Karhumaki J, Maurer H, Paun G, eds. LNCS 3113: Theory is Forever. Berlin: Springer, 2004: 112–124.

[4] Korenek J. Fast Regular Expression Matching Using FPGA. Information Sciences and Technologies Bulletin of the ACM Slovakia, Vol, 2010, 2(2): 103-111.

[5] Zu Yuan, Yang Ming, Xu Zhonghu, et al. GPU-based NFA implementation for memory efficient high speed regular expression matching //Proc of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming, New York: ACM, 2012: 129-140.

[6] Yang Yi-Hua, Prasanna Viktor. Space-time tradeoff in regular expression matching with semi-deterministic finite automata // Proc of IEEE INFOCOM, 2011. Piscataway, N.J: IEEE, 2011: 1853-1861.

[7] Aho, A.V. and Lam, M.S. and Sethi, R. and Ullman, J.D. Compilers: principles, techniques, and tools. 1986.

[8] Kumar S, Dharmapurikar S, Yu F, et al. Algorithms to accelerate multiple regular expressions matching for deep packet inspection. ACM SIGCOMM Computer Communication Review, 2006, 36(4): 339-350.

[9] Ficara D, Giordano S, Procissi G, et al. An improved DFA for fast regular expression matching. ACM SIGCOMM Computer Communication Review, 2008, 38(5): 29-40.

[10] Hopcroft J. An $n\log n$ algorithm for minimizing states in a finite automaton //Proc of Int Symp Theory of Machines and Computations, New York: Academic Press, 1971, 189 - 196.

[11] Becchi Michela, Crowley Patrick. A Hybrid Finite Automaton for Practical Deep Packet Inspection //Proc of the 2007 ACM CoNEXT Conference. New York: ACM, 2007.

[12] Ilie L, Yu S. Algorithms for computing small NFAs // LNCS 2420. Berlin: Springer, 2002: 328-340.

[13] Liu Tingwen, Sun Yong, Yang Yifu, et al. An Efficient Regular Expressions Compression Algorithm From A New Perspective //Proc of IEEE INFOCOM, 2011. Piscataway, N.J: IEEE, 2011: 2129-2137.

乔登科，男，1987 年生，硕士生，E-mail: qiaodengke@gmail.com，主要研究领域为信息安全、模式匹配。
柳厅文，男，1986 年生，博士生，E-mail: liutingwen@software.ict.ac.cn，主要研究领域为模式匹配、算法设计。
孙 永，男，1976 年生，博士，E-mail: sunyong@iie.ac.cn，主要研究领域为信息安全、数据流。
郭莉，女，1969 年生，正研级高工，CCF 会员，E-mail: guoli@ict.ac.cn, 主要研究领域为内容安全管理，网络安全，数据流处理。